6

Lógica digital

Contenido

6.1 Introducción 114
6.2 Circuitos lógicos de sistemas digitales 114
6.3 Circuitos combinacionales 114
6.4 Circuitos secuenciales 131
6.5 Resumen 141
6.6 Ejercicios propuestos 142
6.7 Contenido de la página Web de apoyo 143

Objetivos

- Comprender la lógica de circuitos combinacionales en el nivel de compuerta y bloque.
- Comprender la lógica de circuitos secuenciales en el nivel de compuerta y bloque.

En la página Web de apoyo encontrará un breve comentario de la autora sobre este capítulo.

6.1 Introducción

En este capítulo se desarrollará la metodología de diseño lógico de circuitos electrónicos, digitales, combinacionales y secuenciales, así como de dispositivos lógicos programables. Se muestran los componentes más utilizados y su descripción se ajusta a la norma ANSI. Es de destacar que el diseño y la implementación de estos circuitos corresponden a materias que los alumnos cursan normalmente en un semestre o en su modalidad anual, para alcanzar la profundidad que el tema requiere. Desde el punto de vista de la arquitectura de las computadoras, el tema sólo se incluyó como una referencia para los alumnos que no tuvieron las materias Técnicas digitales o Lógica digital en la currícula de su plan de estudio.

Recuerde que las compuertas son bloques que generan salidas con señales equivalentes a 1 o 0, cuando se satisfacen los requisitos de la tabla de verdad para cada combinación de entradas. Las diversas compuertas lógicas suelen encontrarse en sistemas de computadoras digitales. Cada compuerta tiene un símbolo gráfico diferente y su operación puede describirse por medio de la expresión algebraica de la función que las representa, o por las relaciones entre las entradas y las salidas representadas en una tabla de verdad.

6.2 Circuitos lógicos de sistemas digitales

Según lo visto, una expresión booleana permite la representación algebraica del valor de una función que representa la salida de un circuito. En los circuitos lógicos de sistemas digitales, los valores *0* o *1* se representan con dos niveles de tensión. Las variables de la función booleana se estudian como las entradas en el circuito para luego estudiar su comportamiento. Estos componentes de hardware se denominan circuitos lógicos de sistemas digitales y la representación de sus compuertas lógicas es otra forma de analizar la organización de los elementos electrónicos reales o sistemas electrónicos digitales.

6.3 Circuitos combinacionales

Cada compuerta es un circuito lógico combinacional en sí. De esta manera, se puede "armar" el esquema del comportamiento total del circuito sin necesidad de conocer los elementos electrónicos que lo constituyen (así como lo son la organización de varias de ellas). Los circuitos lógicos se clasifican en combinacionales o secuenciales. Un circuito combinacional o combinatorio permite que en las salidas se obtengan valores binarios "transformados" por la operación de las compuertas vinculadas en él y cuyo valor depende únicamente de los valores establecidos en las entradas. En un circuito secuencial se considera la idea de estado del circuito y, dado un estado actual y las entradas, se puede determinar un próximo estado del circuito y las salidas. Expresado de otra manera, los valores binarios de las salidas dependen no sólo de los valores binarios de las entradas sino también de, por lo menos, una de sus salidas.

Un circuito se representa a partir de un diagrama de bloque que es "una caja negra", donde se especifican las "n" entradas y las "m" salidas, así como, eventualmente, el nombre de la función que realiza, sin definir el esquema de compuertas internas. Se aclara que en todos los circuitos se eliminaron entradas adicionales dispuestas para la alimentación y el control del integrado (Vcc, GND...) y, que en este caso en particular, "n" es igual a "m".

:

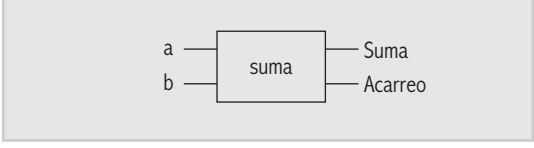


Fig. 6.1. Diagrama de bloque del circuito semisumador.

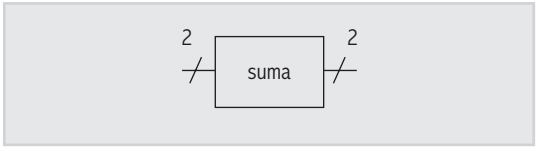


Fig. 6.2. Diagrama de bloque del circuito semisumador (otra opción).

Comenzaremos a analizar circuitos combinacionales. Para realizar el análisis se consideran todos los posibles valores que pueden darse en las entradas. La cantidad de combinaciones se calcula con la fórmula: para n entradas 2^n combinaciones posibles. Las n entradas representan los requerimientos de un problema, cada una puede asumir un valor verdadero (igual a *1*) si el requerimiento se cumple, o falso (igual a *0*) en caso contrario; por ejemplo, en el circuito semisumador, aquí identificado simplemente como "suma" en el capítulo anterior, el problema era resolver la suma de *2* bits representados ahora por *a* y *b*, que constituyen las entradas del circuito. Para cada combinación de entradas, se establece el valor de las salidas esperadas, por lo tanto, una de las formas de llegar a la expresión booleana y, así, a la organización de compuertas, es representar el problema en una tabla de verdad.

Tabla 6-1. Tabla de verdad

a	*b*	*Suma*	*Acarreo*
0	*0*	*0*	*0*
0	*1*	*1*	*0*
1	*0*	*1*	*0*
1	*1*	*0*	*1*

A partir de la tabla para cada función de salida (suma y acarreo), se puede obtener la expresión como suma de productos (forma normal disyuntiva) o como producto de sumas (forma normal conjuntiva), o bien, si se aplica un método de simplificación, se obtiene una expresión mínima de la función o salida del circuito, que para ese caso son dos (suma y acarreo).

Recuerde que para el análisis y la simplificación de sistemas digitales binarios se utiliza como herramienta el Álgebra de Boole.

Niveles de descripción de un circuito combinacional

De menor a mayor nivel de abstracción encontramos, en primer lugar, la descripción del circuito electrónico; en segundo lugar, los diagramas de lógica; luego, la representación algebraica; y, por último, el nivel algorítmico de diseño de componentes. Éste describe su comportamiento utilizando como herramienta un lenguaje, como el VHDL (*Verilog* HDL): es un lenguaje de modelado de hardware similar al C que se presentó como de dominio público y se estandarizó con los estandares 1364-1995,1364-2001, 1364-2005, http://ieeexplore.ieeee.org/xls/standardstoc.jsp?isnumber=33945.

6.3.1 Circuito generador de paridad

Un circuito generador de paridad permite la detección de error en la transmisión de información binaria entre un emisor y un receptor, cuando se asume que la probabilidad de error en la transferencia es lo suficientemente baja como para esperar que se produzca un error y no más. Ambos, emisor y receptor, se asocian a un circuito generador de paridad, de manera que el emisor envíe los bits de paridad. Si el circuito controla paridad, par quiere decir que contabiliza que la cantidad total de *1* en la transmisión es un número par. Si el circuito controla paridad impar significa que contabiliza la paridad impar de *1*. Si se analiza el presente diagrama de lógica, se observa que el circuito no sólo permite generar la función "paridad par" de una cadena de *5* bits y la función "paridad impar" para la misma cadena, sino que también permite seleccionar qué tipo de paridad se utilizará.

Descripción de un circuito digital:

1. Algorítmico o de descripción de la función.
2. Álgebra de Boole, tabla de verdad y diagrama lógico.
3. Conmutador de componentes electrónicos.

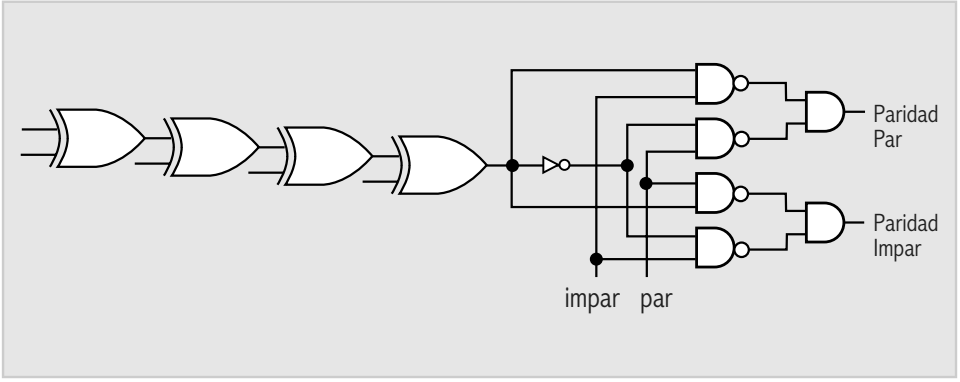


Fig. 6.3. Diagrama de lógica de un circuito generador de paridad.

Ejercicios

Siga la evolución para la combinación de entradas *00100011* en cada nivel de compuertas, primero para el caso de seleccionar paridad "impar", e indique qué ocurre con *PI*, y luego para el caso de paridad par, e indique que sucede con *PP*.

Investigue cómo sería un circuito verificador de paridad para una mensaje original de *3* bits. Represente su tabla de verdad y su diagrama de lógica.

6.3.2 Circuito comparador de magnitud

Es común que se necesite comparar dos números binarios de n bits para saber si éstos son iguales, o si uno es mayor o menor que otro. Podemos simplificar el problema analizando la comparación de *2* bits.

En este caso, es factible pensar en un circuito con dos entradas *a* y *b*, que representan los bits que se han de comparar, y tres funciones de salida que determinan: *X* si *a* es menor que *b*, Y si *a* es igual *a b*, y *Z* si *a* es mayor que *b*. El circuito se representa a partir de los elementos utilizados hasta el momento: diagrama de bloque, tabla de verdad y gráfico o diagrama de lógica.

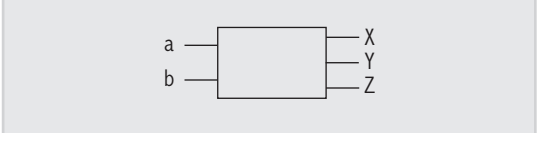


Fig. 6.4. Diagrama de bloque de un circuito comparador de magnitud.

Tabla 6-2. Tabla de verdad de un circuito comparador de magnitud

		Z	X	Y
a	*b*	*a > b*	*a < b*	*a = b*
0	*0*	*0*	*0*	*0*
0	*1*	*0*	*1*	*1*
1	*0*	*1*	*0*	*1*
1	*1*	*0*	*0*	*0*

Diagrama de lógica

Las expresiones algebraicas que representan las funciones de salida son:

$$X = \bar{a} \cdot b \qquad Y = a \cdot \bar{b} + \bar{a} \cdot b \qquad Z = a \cdot \bar{b}$$

Por lo tanto, el diagrama de lógica o gráfico de compuertas que le corresponde es:

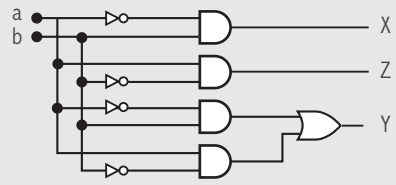


Fig. 6.5. Diagrama de lógica de un circuito comparador de magnitud.

6.3.3 Circuitos convertidores de código

Son circuitos cuya función digital es obtener en las salidas combinaciones binarias pertenecientes a alguna convención de representación de caracteres (numéricos o alfanuméricos). Las entradas pueden ser señales que provienen de algún dispositivo de entrada de información (p. ej., señales de teclado) o pueden ser combinaciones binarias de otro código para convertir.

Ejemplo:

Un circuito que convierte un dígito BCD puro en su correspondiente valor BCD *2421*.

Tabla 6-3. Tabla de verdad

Entradas BCD				**Salidas BCD**			
8	*4*	*2*	*1*	*2*	*4*	*2*	*1*
A0	*A1*	*A2*	*A3*	*F0*	*F1*	*F2*	*F3*
0	*0*	*0*	*0*	*0*	*0*	*0*	*0*
0	*0*	*0*	*0*	*0*	*0*	*0*	*1*
0	*0*	*1*	*0*	*0*	*0*	*1*	*0*
0	*0*	*1*	*1*	*0*	*0*	*1*	*1*
0	*1*	*0*	*0*	*0*	*1*	*0*	*0*
0	*1*	*0*	*1*	*1*	*0*	*1*	*1*
0	*1*	*1*	*0*	*1*	*1*	*0*	*0*
0	*1*	*1*	*1*	*1*	*1*	*0*	*1*
1	*0*	*0*	*0*	*1*	*1*	*1*	*0*
1	*0*	*0*	*1*	*1*	*1*	*1*	*1*
1	*0*	*1*	*0*	X	X	X	X
1	*0*	*1*	*1*	X	X	X	X
1	*1*	*0*	*0*	X	X	X	X
1	*1*	*0*	*1*	X	X	X	X
1	*1*	*1*	*0*	X	X	X	X
1	*1*	*1*	*1*	X	X	X	X

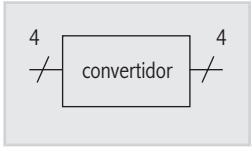


Fig. 6.6. Diagrama de bloque.

Estas entradas faltantes, desde la combinación *1010* hasta la combinación *1111*, no se van a dar nunca, porque no corresponden a combinaciones válidas de un dígito BCD y, por lo tanto, las salidas no están determinadas. Por eso las representamos con X y pueden ser consideradas individualmente como *0* o *1*, según convenga al momento de simplificación.

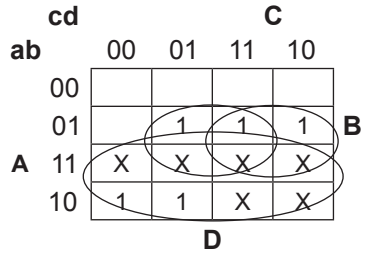


Fig. 6.7. Diagramas de simplificación (a, b, c, d) para la función F0.

En el diagrama se observan tres grupos de *1*. En este caso las "X" fueron consideradas como *1* para optimizar la simplificación:

$$F0 = a + b \cdot c + b \cdot d$$

Una vez halladas las expresiones finales del circuito *F0, F1, F2* y *F3* podemos confeccionar el diagrama lógico que lo representa. Se muestra a continuación el correspondiente a *F0*.

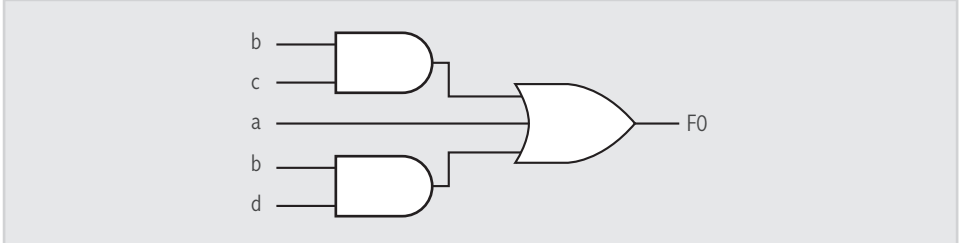


Fig. 6.8. Diagrama de lógica (parcial) de un circuito convertidor de código.

Todo circuito derivado de una expresión minimizada con un mapa tiene dos niveles de compuertas AND y OR (o sea que el nivel lógico de la entrada debe atravesar dos compuertas hasta alcanzar la salida).

Ejercicio:

Para completar el estudio del comportamiento del circuito, represente las funciones mínimas *F1, F2* y *F3*, y sus respectivos diagramas de lógica, y compruebe que para las entradas *1001* corresponden las salidas *1111*.

6.3.4 Circuitos codificadores

Veremos cómo las señales binarias provenientes de un teclado numérico pueden transformarse en salidas codificadas, por ejemplo, en BCD. Consideremos teclas numéricas sin tener en cuenta las teclas de operación (+, -, *, etc.) comunes en todo teclado. Uno de los códigos más usados en la representación de números es el BCD *8421* o BCD puro. Si consideramos solo *10* teclas identificadas como *T9, T8, ... T0* para los dígitos decimales de *9* a *0*, debemos asumir que constituyen las entradas del circuito y, por lo tanto, la tabla de verdad deberá representarse con 2^{10} combinaciones posibles; sin embargo, la tabla de verdad muestra sólo *10* de las $2^{10} = 1024$ combinaciones posibles para *10* entradas, esto se debe a que se asume por simplicidad del circuito que no se presionan dos teclas en forma simultánea. En la tabla sólo se representan las combinaciones canónicas que indican que si se presionó una tecla, no se presionó ninguna otra. Esto permite transformar cada minitérmino, que en principio es el producto de todas las variables en juego, en una sola variable, ya que si esa variable es verdadera las demás son falsas.

Tabla 6-4. Tabla de verdad

Entradas que identifican las teclas										*Salidas BCD*			
										8	*4*	*2*	*1*
T9	*T8*	*T7*	*T6*	*T5*	*T4*	*T3*	*T2*	*T1*	*T0*	F_3	F_2	F_1	F_0
0	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*
0	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*1*
0	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*1*	*0*
0	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*1*	*1*
0	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*
0	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*1*
0	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*0*
0	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*
0	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*
1	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*1*

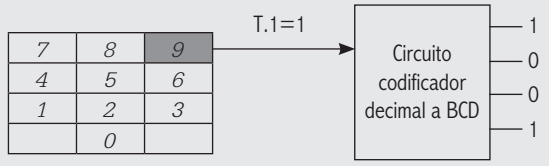


Fig. 6.9. Esquema de codificación de tecla a dígito BCD.

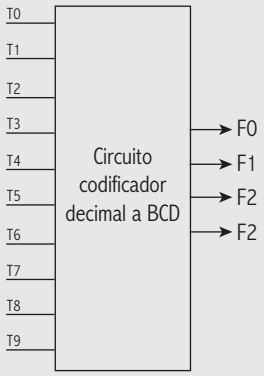


Fig. 6.10. Diagrama de bloque.

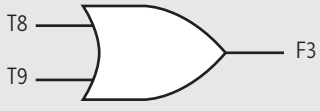


Fig. 6.11. Diagrama de lógica correspondiente a F3.

que se interpreta de la siguiente manera: *F3* "vale *1*" cuando se presiona indistintamente la tecla *T8* o la tecla *T9*.

Los terminales de entrada de un circuito digital aceptan señales binarias, dentro de las tolerancias permitidas, y los circuitos responden en los terminales de salida con señales binarias, que caen dentro de las tolerancias permitidas.

Las funciones de salida *F0*, *F1*, *F2*, *F3* no se pueden minimizar por Karnaugh, porque cada función está definida en términos de más de *6* variables; sin embargo, vemos claramente que en cada caso podemos representarlas en forma algebraica de la siguiente manera:

$$F0 = T1 + T3 + T5 + T7 + T9$$

$$F1 = T2 + T3 + T6 + T7$$

$$F2 = T4 + T5 + T6 + T7$$

$$F3 = T8 + T9$$

que se interpreta verbalmente como: *F0* se activa cuando se presiona la tecla *T1*, *T3*, *T5*, *T7* o *T9*.

Ejercicio:

Observe que cuando se presiona la tecla *0* el valor de las salidas también es *0 0 0 0*, y cuando no se presiona ninguna de ellas se observa lo mismo. Como no se ha contemplado esta situación, piense qué entrada adicional agregaría para establecer la diferencia y grafique el circuito.

Aplicaciones de los decodificadores:

1. En chips de memoria, posibilitan el acceso *random,* convirtiendo el número que identifica la dirección de la celda de memoria en la fila y la columna.
2. En bancos de memoria, permiten la decodificación que identifica el banco.
3. Permiten la selección de dispositivos de E/S de microprocesadores.
4. Facilitan la decodificación de instrucciones en la CPU para activar señales de control.
5. Decodificador BCD-7 segmentos mostrado y codificadores de teclados (asignando valores binarios a códigos alfanuméricos o numéricos); por ejemplo, asignar la combinación 1010 para desplegar la condición E de "error".

6.3.5 Circuito decodificador de código

Opera en forma inversa al codificador en el siguiente sentido: la información que ingresa al circuito son combinaciones binarias pertenecientes a alguna convención de representación de caracteres (numéricos o alfanuméricos) y las convierte a señales binarias para representar en algún dispositivo de salida de información (por ejemplo: señales para video o para impresora). El circuito más simple que podemos analizar realizando esta función es el decodificador de BCD a un *display* de *7* segmentos muy utilizado en calculadoras. Se asume que para cada segmento, un "*1*" activa un diodo emisor de luz (LED o *Light-Emitting Diode*) y un "*0*" lo vuelve inactivo. El circuito recibe un dígito BCD y genera en su salida los *0* y *1* que "dibujen" el valor decimal en el *display*. Representamos en una tabla de verdad, todos los números tal como queremos que aparezcan:

Tabla 6-5. Tabla de verdad

A	*B*	*C*	*D*	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*0*
0	*0*	*0*	*1*	*0*	*1*	*1*	*0*	*0*	*0*	*0*
0	*0*	*1*	*0*	*1*	*1*	*0*	*1*	*1*	*0*	*1*
0	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*0*	*0*	*1*
0	*1*	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*1*
0	*1*	*0*	*1*	*1*	*0*	*1*	*1*	*0*	*1*	*1*
0	*1*	*1*	*0*	*1*	*0*	*1*	*1*	*1*	*1*	*1*
0	*1*	*1*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*
1	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
1	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*0*	*1*	*1*
1	*0*	*1*	*0*	X	X	X	X	X	X	X
1	*0*	*1*	*1*	X	X	X	X	X	X	X
1	*1*	*0*	*0*	X	X	X	X	X	X	X
1	*1*	*0*	*1*	X	X	X	X	X	X	X
1	*1*	*1*	*0*	X	X	X	X	X	X	X
1	*1*	*1*	*1*	X	X	X	X	X	X	X

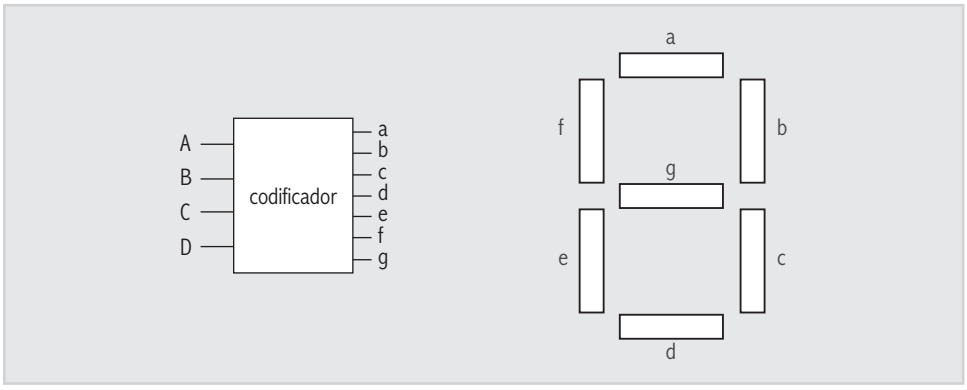


Fig. 6.12. Esquema de un display de 7 segmentos.

Encuentre un simulador de un *display* BCD de siete segmentos en la Web de apoyo.

Ejercicio:

Para completar este circuito decodificador de código, es necesario hallar las funciones mínimas *a*, *b*, *c*, *d*, *e*, *f*, *g* y confeccionar el diagrama de lógica que incluye todas las funciones. Ejercítelo usted, considerando que el análisis e implementación de cualquier otro circuito codificador o decodificador se asemeja a los ejemplos dados.

6.3.6 Circuito decodificador n · 2n

Existe otro tipo de decodificadores que permiten señalar en una de sus salidas qué combinación binaria se dio en la entrada. Los decodificadores de este tipo se denominan "$n \cdot 2^n$" (se lee "*n . 2 a la n*" o "*n a 2 a la n*" o "decodificador binario") y contemplan una salida activa para cada posible combinación de entradas. Esto quiere decir que una sola línea de salida obtendrá el valor *1*, mientras que las demás permanecen en *0*. Se puede decir que la combinación de las entradas "elige" una salida entre las demás y, por esta razón, las entradas también reciben el nombre de entradas de selección.

Encuentre un simulador de un decodificador de dos entradas en la Web de apoyo.

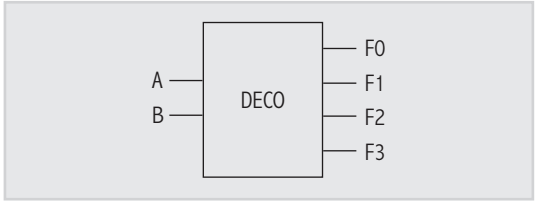


Fig. 6.13. Diagrama de bloque del decodificador $n \cdot 2^n = 2 \cdot 2^2 = 2 \cdot 4$ sin entrada de habilitación

La siguiente tabla representa las cuatro combinaciones posibles para las dos entradas. La salida *F0* señala con un *1* la combinación de entradas *00*, la salida *F1* señala con un *1* la combinación de entradas *01*, y así sucesivamente. El nivel lógico que identifica la combinación puede variar; lo importante es que ese nivel sea el opuesto de las otras tres combinaciones. Si se analiza cada función, se observa que en este caso tienen un solo "*1*" y, por lo tanto, no es posible la agrupación de *1* en un diagrama de simplificación.

Tabla 6-6. Tabla de verdad para un decodificador de 2 entradas y 4 salidas

A	*B*	*F0*	*F1*	*F2*	*F3*
0	*0*	*1*	*0*	*0*	*0*
0	*1*	*0*	*1*	*0*	*0*
1	*0*	*0*	*0*	*1*	*0*
1	*1*	*0*	*0*	*0*	*1*

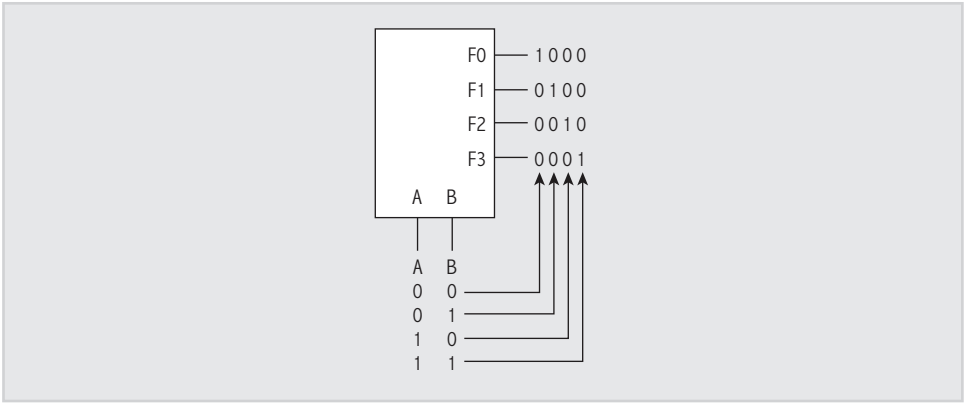


Fig. 6.14. Diagrama de bloque.

Un decodificador $nX2_n$ activa la salida correspondiente al número binario establecido en las entradas.

Luego, las funciones se representan algebraicamente con el minitérmino que corresponde al valor de entrada.

$$F0 = \bar{A} \cdot \bar{B}$$
$$F1 = \bar{A} \cdot B$$
$$F2 = A \cdot \bar{B}$$
$$F3 = A \cdot B$$

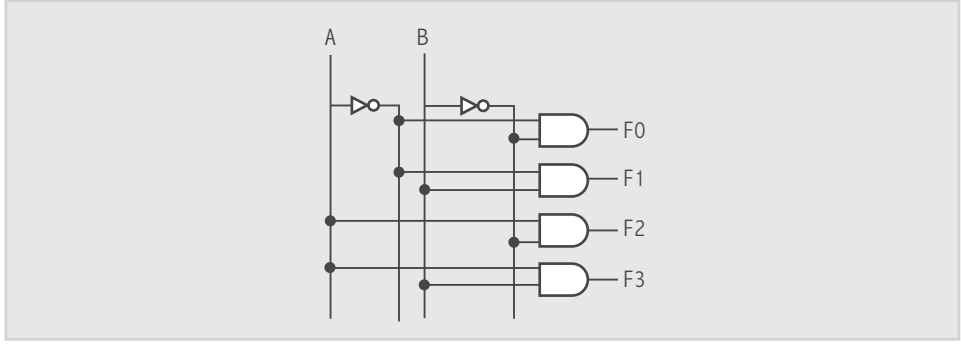


Fig. 6.15. Diagrama de lógica.

En el circuito anterior cada compuerta AND depende de dos entradas, pero puede tener una entrada adicional, asociada a cada AND, que permita controlar su operación (llamada entrada de habilitación). Si esta entrada, llamémosla *EN* (*enable*), es igual a *0*, el circuito genera *0* en todas sus salidas, o sea, "no decodifica", pues no señala la combinación dispuesta en la entrada; una habilitación de este tipo se denomina *strobe* o estrobo. Si *EN* es *1*, entonces el circuito opera según lo que hemos visto.

En términos algebraicos, si $EN = 1$, entonces los minitérminos adquieren una nueva variable y las funciones se representan así:

$$F0 = EN \cdot \bar{A} \cdot \bar{B} = 1 \cdot \bar{0} \cdot \bar{0} = 1 \cdot 1 \cdot 1 = 1$$
$$F1 = EN \cdot \bar{A} \cdot B = 1 \cdot \bar{0} \cdot 0 = 1 \cdot 1 \cdot 0 = 0$$
$$F2 = EN \cdot A \cdot \bar{B} = 1 \cdot 0 \cdot \bar{0} = 1 \cdot 0 \cdot 1 = 0$$
$$F3 = EN \cdot A \cdot B = 1 \cdot 0 \cdot 0 = 1 \cdot 0 \cdot 0 = 0$$

con entrada de habilitación.

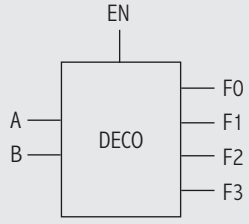


Fig. 6.16. Diagrama de bloque.

En las cuatro primeras combinaciones no decodifica.

Tabla 6-7. Tabla de verdad

EN	*a*	*b*	*F0*	*F1*	*F2*	*F3*
0	*0*	*0*	***0***	***0***	***0***	***0***
0	*0*	*1*	***0***	***0***	***0***	***0***
0	*1*	*0*	***0***	***0***	***0***	***0***
0	*1*	*1*	***0***	***0***	***0***	***0***
1	*0*	*0*	*1*	*0*	*0*	*0*
1	*0*	*1*	*0*	*1*	*0*	*0*
1	*1*	*0*	*0*	*0*	*1*	*0*
1	*1*	*1*	*0*	*0*	*0*	*1*

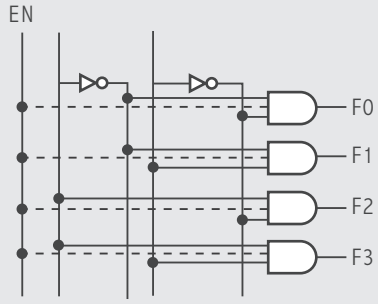


Fig. 6.17. Diagrama de lógica.

Genéricamente, un decodificador de n entradas se implementa con 2^n compuertas AND, cada una con n entradas. Un ejemplo de aplicación del circuito es el decodificador que se puede utilizar para el acceso, al seleccionar una dirección al azar en una memoria (acceso *random*).

La dirección coincide con el valor de una combinación en la entrada y permite indicar qué palabra va a ser leída o escrita. El decodificador de dirección habilita la palabra seleccionada e inhabilita las demás, de acuerdo con la dirección que reciba en sus entradas. Por ejemplo, si consideramos una memoria de cuatro palabras (ejemplo teórico) tendremos cuatro direcciones, empezando de *0* a *3*. Las direcciones binarias son entonces, *00*, *01*, *10* y *11*; la salida del decodificador habilitará con un *1* la palabra seleccionada e inhabilitará con un *0* las tres restantes.

Ejercicio:

Pruebe el circuito colocando una combinación a la entrada y siga su evolución para cada nivel de compuertas, por ejemplo, para la dirección de entrada *01*, que indica que la palabra seleccionada es la segunda.

Los decodificadores pueden usar compuertas NAND o NOR, en vez de AND, y la salida representativa de la combinación se distingue con *0*, es decir, la habilitación de la palabra se activa con *0* y las restantes con *1* (la deshabilitación de la palabra se activa con *1*), ya que la salida es el complemento de los valores de las variables de salida de la tabla de verdad.

Ejercicio:

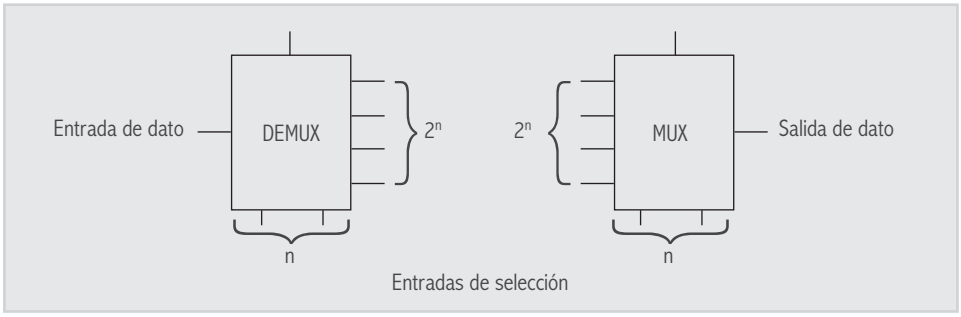
Para este caso represente la tabla de verdad y el diagrama de lógica.

6.3.7 Circuitos multiplexores y demultiplexores

Un demultiplexor es un circuito cuya función digital es "encaminar" la información que viene en una sola línea de entrada, en *1* de 2^n. Se puede pensar en un decodificador en el que la entrada de habilitación es la portadora del bit que se ha de "encaminar" y la selección de la línea de salida (habilitada por líneas de selección) determina qué salida es la encargada de transmitirlo. Por este motivo se lo conoce como decodificador/demultiplexor. Algunas de las aplicaciones de un demultiplexor son, por ejemplo, distribuir una entrada de datos en serie a una salida en paralelo o transferir bits de una línea de bus a un registro determinado; por eso también se lo denomina distribuidor.

Un multiplexor es un circuito combinacional cuya función digital es "encaminar" las señales binarias de *1* de 2^n líneas posibles de entrada en una única línea de salida. La línea de entrada de dato se "elige" a partir de los valores que puedan tomar las n líneas de selección. Ésta también se parece a un circuito decodificador $nX2^n$, donde las compuertas AND decodifican las líneas de selección de entrada, que se unifican utilizando una compuerta OR, cuya salida es el dato seleccionado. En sentido inverso, el multiplexor convierte una entrada paralela a una cadena en serie.

Por lo tanto, ambos se utilizan para transmitir datos a través de una sola línea en forma de serie, el multiplexor envía y el demultiplexor recibe. El número de entradas a un multiplexor o de salidas de un demultiplexor puede aumentarse usando más compuertas AND; para ese fin, se requieren más entradas o líneas de selección. Por ejemplo, con *2* entradas (*A* y *B*) se controlan *4* compuertas AND; con *3* se controlan *8*, etcétera.



El multiplexor conecta una de las 2^n entradas a la salida. Esta entrada se selecciona con las n entradas de selección, o sea que permite realizar una función de conmutación de n variables.

Fig. 6.18. Diagrama de bloque.

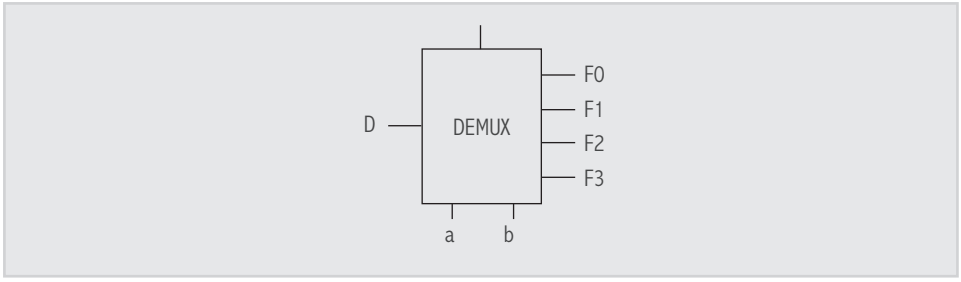


Fig. 6.19. Diagrama de bloque de un demultiplexor para una entrada de dato D, dos entradas de selección a y b y cuatro posibles salidas F0, F1, F2, F3.

Tabla 6-8. Tabla de verdad

a	*b*	*F0*	*F1*	*F2*	*F3*
0	*0*	*D*	*0*	*0*	*0*
0	*1*	*0*	*D*	*0*	*0*
1	*0*	*0*	*0*	*D*	*0*
1	*1*	*0*	*0*	*0*	*D*

El demultiplexor conecta la entrada con una de las 2n salidas, ésta se selecciona con las n entradas de selección.

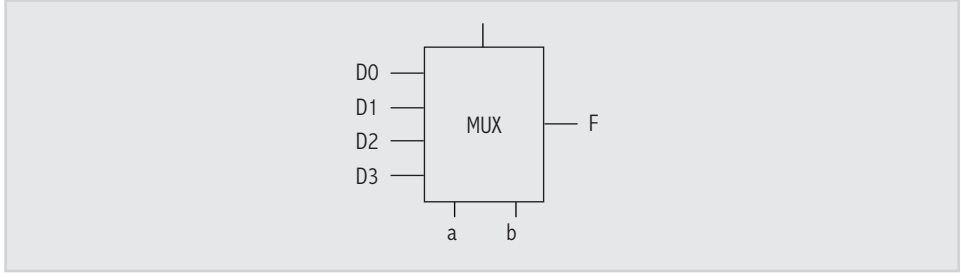


Fig. 6.20. Diagrama de bloque de un multiplexor para cuatro entradas de dato D0, D1, D2 y D3, dos entradas de selección a y b, y la única salida de dato F.

Tabla 6-9. Tabla de verdad

a	*b*	*F*
0	*0*	*D0*
0	*1*	*D1*
1	*0*	*D2*
1	*1*	*D3*

Aplicaciones de los multiplexores

1. Un MUX se puede implementar con cierto número de compuertas lógicas, porque básicamente es un decodificador con sus salidas asociadas a una única compuerta.
2. Los MUX de pocas entradas se utilizan en los dispositivos de lógica programables descriptos más abajo.
3. Asimismo, el MUX se utiliza para la conversión de bits transferidos en paralelo a serie, esto es, de a uno por vez, utilizando una sola línea.
4. La multiplexación de bits de dirección en memorias se emplea, por ejemplo, cuando un bloque de caché supera la capacidad del bus, si el bloque es de *128* bits y el bus de *64*; entonces, primero se transfiere un grupo de *64* bits y luego el otro.
5. Se puede armar un módulo desplazador que permite mover bits a derecha e izquierda en registros de desplazamiento bidireccionales, bajo microoperaciones de control, por ejemplo, las que genera la unidad de control cuando se ejecuta una instrucción de desplazamiento (ver sección "Registros con facilidad de desplazamiento").

Ejemplo de una unidad aritmético-lógica (ALU) para *4* operaciones de tipo lógico utilizando un multiplexor. Supongamos las operaciones NOT (negación o complemento), AND (conjunción o producto lógico), OR (disyunción inclusiva o suma lógica), XOR (disyunción excluyente o suma exclusiva), que identificamos con los números *00*, *01*, *10*, *11*, respectivamente. Las señales de control indican cuál de las "instrucciones" pasan por la línea de resultado, al operar los bits *Op1* y *Op2*.

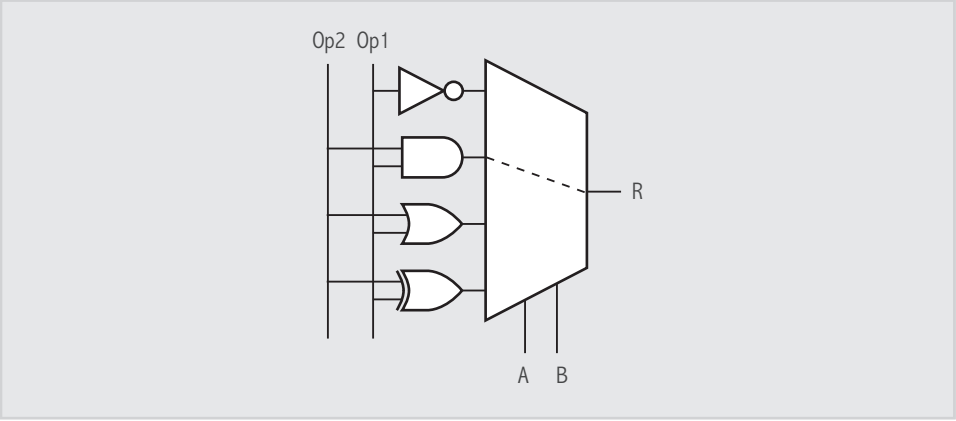


Fig. 6.21. Diagrama de lógica.

Para este ejemplo, *A = 0* y *B = 1*, entonces, el resultado *R* es el producto lógico entre ambos operadores: *Op1 AND Op2*.

6.3.7.1 Bus asociado a un multiplexor-demultiplexor

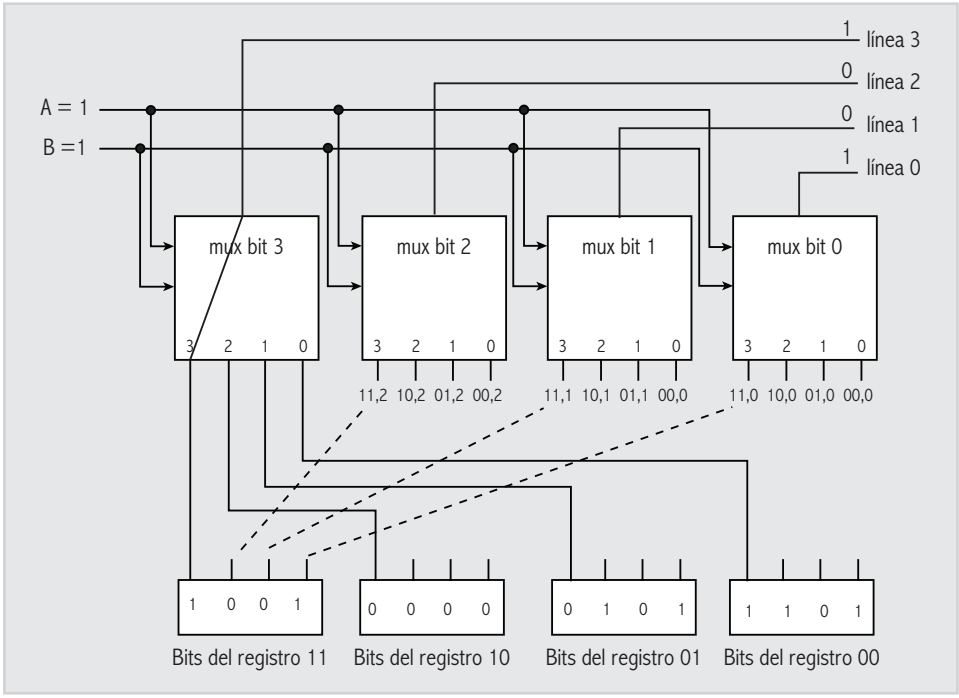
Un bus es un conjunto de conductores que permiten relacionar registros que actúan de emisores o de receptores de bits. En este caso se indica que los registros están asociados "a un bus común o bus único".

El vuelco de la información de los registros sobre el bus se realiza con circuitos multiplexores. Cada registro está asociado a un número de multiplexor y éste lo puede reconocer en sus entradas de selección. En el ejemplo de la siguiente figura los cuatro registros emisores posibles envían sus bits de mensaje sobre las n líneas del bus.

Cada multiplexor habilita el paso de una señal sobre el bus comandado por la dirección en sus entradas de selección. Así:

$$A = 1 \text{ y } B = 1$$

indican que *D3* es la línea de entrada seleccionada para salir por *F*.



Este esquema muestra cómo *4* registros de *4* bits cada uno, pueden transferir su información a través de un bus de *4* líneas. Éstas se identifican como **línea 3**, **línea 2**, **línea 1** y **línea 0.** Sobre las líneas de selección *A* y *B*, cada multiplexor recibe la identificación del registro que va a transferir, en este caso el registro identificado como *11*. Así, el Mux de bit *3* se relaciona con los "bit *3*" de cada registro, pero sólo el bit *3* del registro *11* es el "seleccionado" para ser transferido por la **línea 3**. Se debe pensar que los siguientes multiplexores operan de la misma manera, sólo que las relaciones están marcadas en el esquema en línea punteada para facilitar su comprensión. Cada señal de entrada al multiplexor se identificó con el número de registro y número de bit; o sea que *11,2* se debe interpretar como el bit *2* del registro *11*.

6.3.8 Circuitos "programables" para múltiples funciones

Si se considera que las salidas de un decodificador $n \cdot 2^n$ son entradas a una compuerta OR, se puede diseñar una estructura de compuertas AND-OR sobre la cual "programar" una función digital. Para que el circuito resulte programable las salidas de las compuertas AND se asocian a un enlace electrónico, y éste, a su vez, a la entrada de la compuerta OR. Entonces, la lógica AND-OR no es fija, sino que puede programarse.

El proceso de programación de estos circuitos es físico y, por lo tanto, la mayoría de las veces es inalterable; el ejemplo presentado pertenece al tipo de circuitos conocido como PLD (*Programmable Logic Device*), que se utilizan para implementar funciones lógicas sin mayor análisis que el de la tabla de verdad y, también, como soporte de secuencias binarias fijas. O

Algunos tipos de PLD: ROM (*Read Only Memory*), PROM (*Programable Read Only Memory*), EPROM (*Erasable PROM*), EEPROM (*Electrically Erasable* PROM), FLASH (subtipo de memoria EEPROM)

sea, "instrucciones" de programas en los que la secuencia de *0* y *1* esperada a la salida sea siempre la misma para cada combinación. Los dispositivos de lógica programable (PLD) son circuitos integrados con cientos de compuertas AND-OR que se relacionan por medio de enlaces electrónicos y conexiones fijas. Para la representación de sus diagramas de lógica en PLD se utiliza una simbología particular, por ejemplo, para representar una compuerta AND de tres entradas:

Supongamos que queremos programar la función digital que responde al circuito semisumador a partir de su tabla de verdad y que las variables de este circuito son las entradas a un decodificador *2* a *4*.

Tabla 6-10. Tabla de verdad

a	*b*	*Suma*	*Acarreo*
0	*0*	*0*	*0*
0	*1*	*1*	*0*
1	*0*	*1*	*0*
1	*1*	*0*	*1*

En total hay *4* enlaces intactos y *4* destruidos; para programar el circuito se debe destruir el enlace, según indique la tabla de verdad, de la siguiente manera:

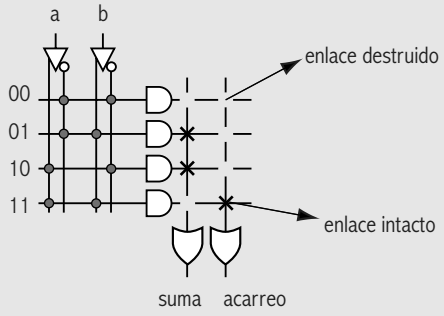


Fig. 6.22. Diagrama de lógica.

En la red OR las cruces representan la programación de *1* o enlace intacto. Los espacios vacíos representan la programación de *0* o enlace destruido. En la red AND los círculos rellenos representan uniones fijas o no programadas.

6.3.8.1 Memorias sólo de lectura

El arreglo de compuertas AND-OR, programado de esta manera, es un ejemplo simple de memoria ROM (*Read Only Memory*). Desde el punto de vista de los circuitos lógicos, forma parte de los dispositivos de lógica programable y se caracterizan por tener conexiones fijas en el arreglo de compuertas AND y conexiones programables en el arreglo de compuertas OR. Una vez establecida la combinación binaria de las salidas, éstas se mantienen inalterables.

Cuando una ROM se utiliza para implementar una función lógica, las entradas al decodificador AND constituyen las variables de la expresión booleana; cuando se utiliza como programa fijo,

constituyen la dirección de memoria para acceder. Cada línea de salida del decodificador se considera una palabra de memoria y cada enlace con un enlace (intacto o no) se considera una celda de "almacenamiento" de un bit. La cantidad de compuertas OR determina la cantidad de bits por palabra, y así se forma una matriz de *mXn* (*m* palabras de n bits cada una). Para identificar una de *m* palabras existe un valor *p*, tal que $2^p = m$, entonces *p* indica el número de entradas al decodificador. Este circuito puede tener entrada de habilitación y sus salidas de dato (salidas OR) pueden tener tres estados: "*1*" lógico, "*0*" lógico, o bien "alta impedancia" (este último estado se debe considerar "fuera de servicio"). Al inicio, las salidas de una ROM están todas en "*1*" (enlace intacto). La programación de los *0* es un procedimiento físico que genera la destrucción de los enlaces. Según las características de este procedimiento, se fabrican distintos tipos de dispositivos. En las ROM propiamente dichas, se confecciona la tabla de verdad con las combinaciones que se necesita en cada palabra; el proceso de fabricación incluye una máscara que representa la tabla. La grabación de esta máscara es el último paso en la fabricación del dispositivo. Según la aplicación, se pueden utilizar dispositivos tipo PROM (ROM-programable); en este caso el circuito viene "virgen" y puede ser programado individualmente. Hay ROM reprogramables llamadas EPROM (PROM borrable) o EEPROM (PROM borrable eléctricamente), que pueden utilizarse no sólo como soporte de un programa fijo, sino también como complejos codificadores. Una vez programada, la EPROM permite la restauración de los enlaces destruidos, razón por la que posibilita un nuevo ciclo de grabado, si bien los ciclos de borrado de escritura están limitados a la tecnología que se emplee.

Distribución de líneas en una memoria de sólo lectura

- La distribución de una memoria forma una matriz de *M x N bits* (*M* direcciones de *N bits* cada una).
- El bus de direcciones tiene *p* líneas, tal que $2^p = M$.
- El bus de datos tiene *N* líneas para transferir los *N bits* leídos.
- El bus de control tiene una línea que habilita el chip y normalmente se denomina CS (*Chip Select*).
- En cualquiera de los ROM, la red AND es inalterable y la red OR es programable.

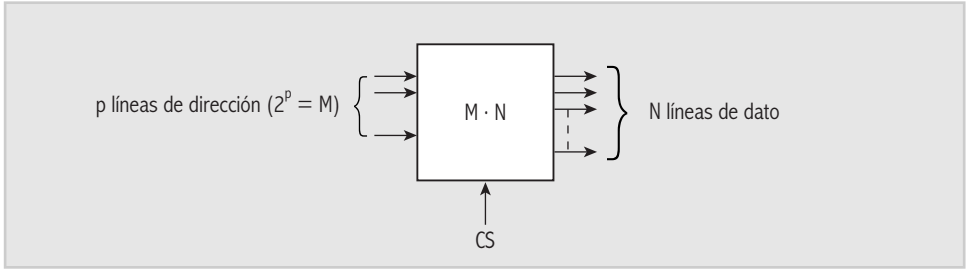


Fig. 6.23. Diagrama de bloque de cualquiera de las memorias de sólo lectura.

Diagrama de lógica de una ROM sin programar

Una memoria ROM no es más que otra forma de representar los minitérminos para cada función de *p* variables; así, en el diagrama siguiente se representan las ocho combinaciones posibles de tres variables (*a*, *b* y *c*) como enlaces fijos en una red AND que, acoplada a una red OR, podrá "programarse" para cada una de las dos funciones de salida *F0* y *F1*

En las ROM, los bits (fijos) se almacenan en posiciones de memoria. Cada posición de memoria esta asociada a una dirección.

● enlace fijo × enlace programable

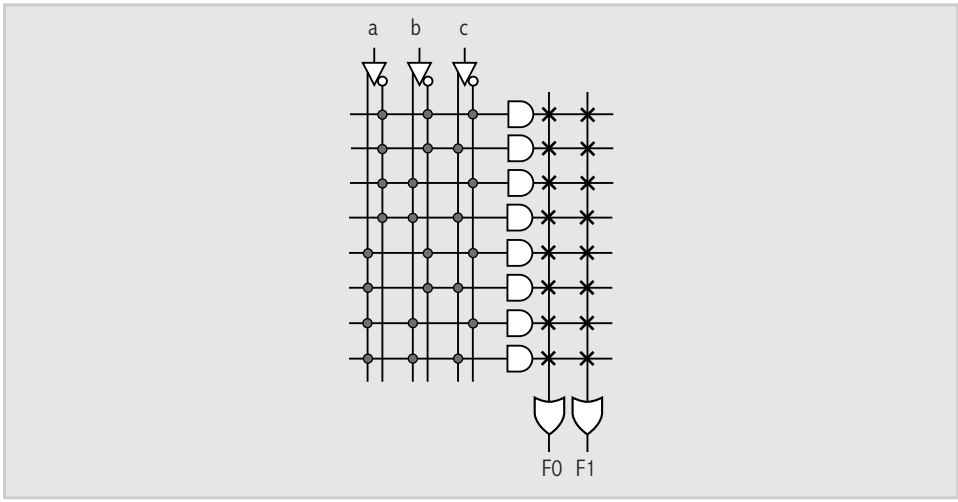


Fig. 6.24. Diagrama de lógica de una ROM sin programar.

6.3.8.2 Dispositivos tipo PAL

En una nueva categoría de dispositivos de lógica programable, se encuentran los denominados PAL (*Programable Array Logic*), más flexibles y rentables que las típicas ROM. Su función es similar a la de la ROM, pero en este caso se invierten los papeles de las redes AND-OR. La red AND es programable y la OR es fija. El diagrama para el ejemplo anterior puede ser:

En una ROM, las direcciones se indican por medio del bus de direcciones y los datos se leen desde el bus de datos.

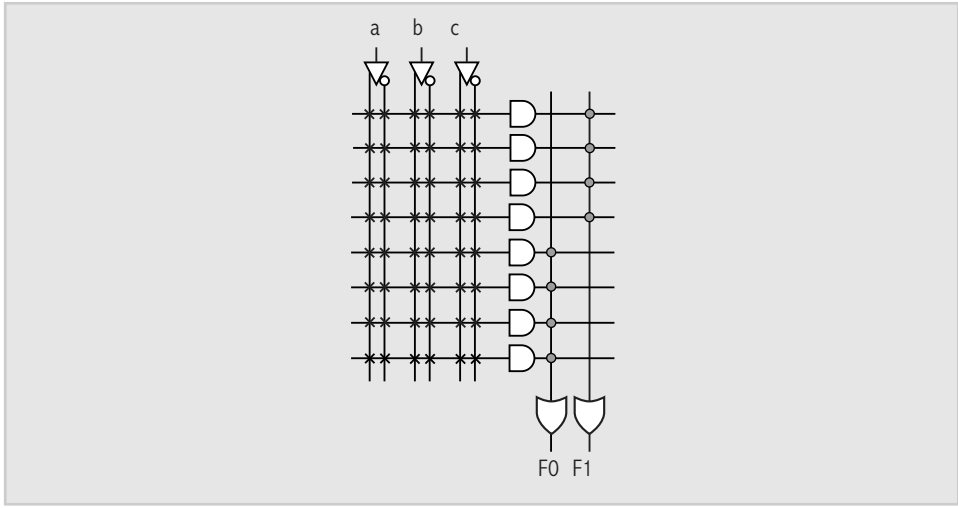


Fig. 6.25. Diagrama de lógica de un dispositivos tipo PAL.

6.3.8.3 Dispositivos tipo PLA o F-PLA (field-PLA)

Los dispositivos tipo F-PLA (*field* = campo) constituyen otra categoría de los dispositivos de lógica programable, y tienen aún mayor flexibilidad, dado que ambas redes son programables.

Diagrama de lógica para un PLA no programado, esta vez de tres entradas y tres salidas: *F0*, *F1* y *F2*.

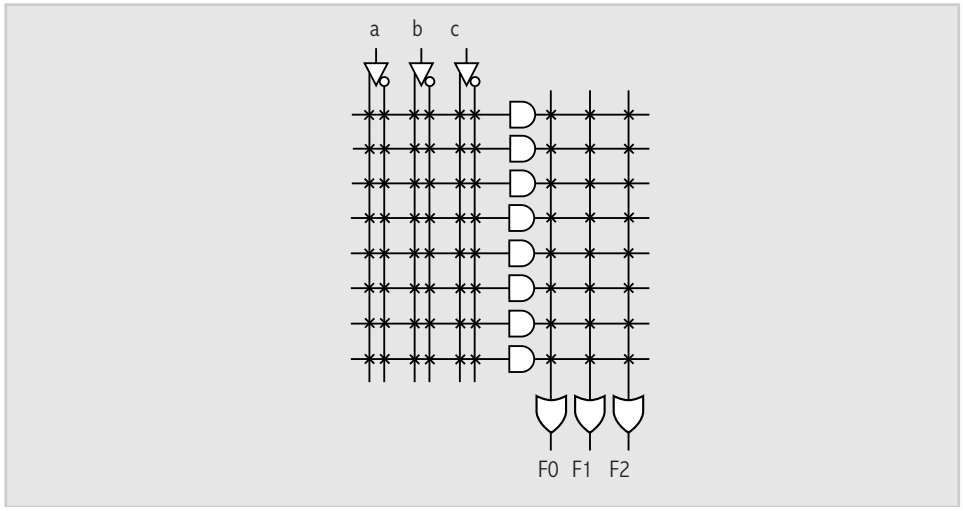


Fig. 6.26. Diagrama de lógica de un dispositivo tipo PLA o F- PLA (field- PLA).

6.4 Circuitos secuenciales

Un circuito es secuencial cuando sus salidas dependen del valor de las entradas y del estado anterior del circuito representado, que puede tener una o más salidas. Por lo tanto, la representación de estos circuitos debe tener en cuenta que las funciones de salida deben contemplarse como entradas; por ello se afirma que el circuito tiene uno o más lazos de realimentación.

Asincrónicos: los cambios sólo se producen cuando están presentes las entradas sin necesidad de una señal de reloj.

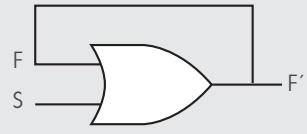
En los circuitos secuenciales asincrónicos los cambios se producen únicamente por activación de alguna de las entradas.

Sincrónicos: los cambios se producen cuando se establecen las entradas y, además, se genera una transición de señal de reloj.

6.4.1 Biestables o *flip-flops*

Un biestable es una celda binaria capaz de almacenar un bit. Tiene dos salidas, una para el valor del bit almacenado y otra que representa su complemento. Su denominación sugiere que el biestable tiene sólo dos estados posibles de funcionamiento, permaneciendo en cualquiera de ellos si los niveles lógicos de las entradas no fuerzan su cambio. El biestable es un arreglo de compuertas, caracterizado por tener lazos de realimentación que permitan mantener o memorizar el efecto de combinaciones anteriores en las entradas.

Analicemos el siguiente diagrama de lógica:

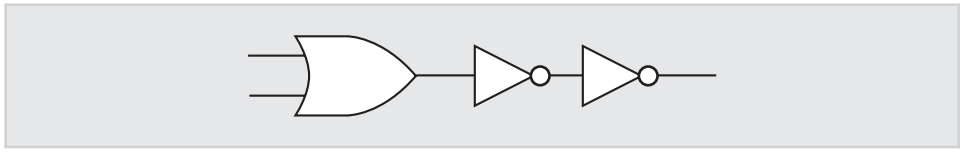


Si consideramos que F' es el valor de F después de aplicada la señal sobre S ($F' = F + S$), vemos que $S = 0$, y si $F = 0$ entonces F' permanece en 0 hasta que no se modifique el valor en S. Por lo tanto, podemos indicar que el circuito retiene o memoriza un 0.

Si consideramos ahora $S = 1$ y $F = 0$, luego del tiempo en que la señal de entrada se propaga a través de la compuerta, F' cambia a 1 y podemos afirmar que, a partir de allí, el estado de F' permanecerá en 1 de manera indefinida, en forma independiente del valor de S. En consecuen-

cia, aunque este tipo de circuito retiene o almacena un bit, no podrá volver a *0* nunca y, por lo tanto, no sirve como "para ser escrito con *0*" luego de haberse escrito con *1*.

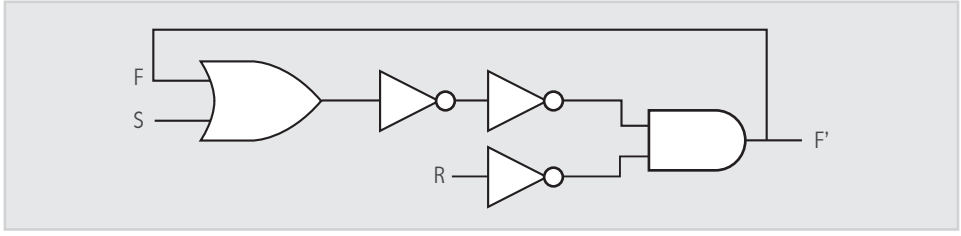
Primero observemos que si a una compuerta OR le acoplamos dos compuertas NOT en serie, la salida final no cambia el funcionamiento de la compuerta, como lo demuestra la tabla de verdad.



A	*B*	*A + B*	*no (A + B)*	*no(no(A + B)*
0	*0*	*0*	*1*	*0*
0	*1*	*1*	*0*	*1*
1	*0*	*1*	*0*	*1*
1	*1*	*1*	*0*	*1*

Para que un circuito sea un elemento de memoria tiene que poder cumplir con dos condiciones: "retener un bit", aún después de la desactivación de las entradas, y poder ser "puesto a *0* o puesto a *1*", independientemente del valor del bit almacenado con anterioridad. Una posibilidad de lograr el cambio del valor almacenado sería modificar el circuito original, agregando una puerta AND que permita el ingreso de una "línea de borrado" (en el diagrama, la línea *R*).

Analicemos el siguiente gráfico. Si ahora agregamos a esta salida una compuerta AND, que realimenta una de las entradas de la compuerta OR, si a la otra entrada de la compuerta la llamamos *S* de SET o puesta a *1* y si, a su vez, se agrega una compuerta NOT a la entrada –la otra entrada de la AND a la que denominamos *R* de RESET o puesta a *0*–, obtenemos el circuito siguiente que cumple con las condiciones enunciadas:



La entrada *F* es el estado del bit antes de que se produzca un cambio.

La entrada *S* es la "acción de puesta a uno" o set.

La entrada *R* es la "acción de puesta a cero" o reset.

La salida *F'* es el estado del bit luego de activar las entradas *S* o *R* con un *1* en cualquiera de ellas;

Al adicionar la compuerta AND se ha logrado que el circuito "resetee" la salida o, lo que es lo mismo, se ha logrado su "puesta a cero". De este modo, si se analiza su funcionamiento, ahora con cada una de las combinaciones en las entradas y la correspondiente realimentación de la salida, se podrá comprobar que su funcionamiento coincide con la siguiente tabla de verdad.

Tabla 6-11. Tabla de verdad

R	*S*	*F*	*F´*
0	*0*	*0*	*0*
0	*0*	*1*	*1*
0	*1*	*1*	*1*
0	*1*	*1*	*1*
1	*0*	*0*	*0*
1	*0*	*0*	*0*
1	*1*	*0*	X
1	*1*	*1*	X

Este circuito, tal como está graficado, se parece a un biestable llamado "*flip-flop*" o "biestable *R-S*". Para transformarlo como tal, podemos operar algebraicamente a partir de la doble negación y las leyes de De Morgan, de manera que se permita lograr ese circuito implementado con compuertas NOR:

6.4.1.1 Biestable R–S asincrónico

Tabla 6-12. Tabla de verdad

R	*S*	*Q t*	*Q t + 1*
0	*0*	*0*	*0*
0	*0*	*1*	*1*
0	*1*	*1*	*1*
0	*1*	*1*	*1*
1	*0*	*0*	*0*
1	*0*	*0*	*0*
1	*1*	*0*	X
1	*1*	*1*	X

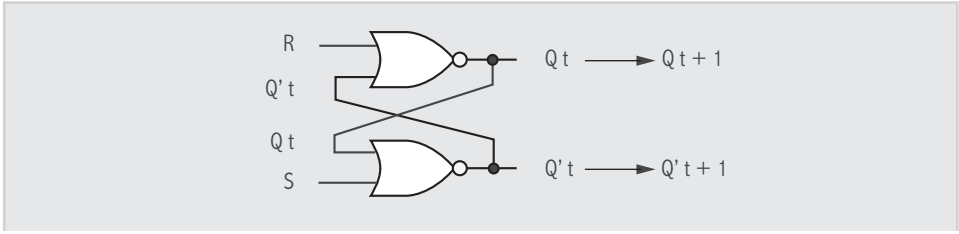


Fig. 6.27. Diagrama de lógica con compuertas NOR.

El gráfico del biestable *R-S* asincrónico reemplaza al expuesto antes, ya que resultan más visibles las interconexiones entre las dos compuertas NOR.

Siendo:

- *R* la entrada de reseteo o puesta a *0*.
- *S* la entrada de seteo o puesta a *1*.
- *Qt* el almacenamiento del valor del biestable antes de ser activado.
- *Qt + 1* el almacenamiento del estado posterior del biestable.
- *Q't* el almacenamiento del complemento del valor del biestable antes de ser activado.
- *Q't + 1* el almacenamiento del estado posterior del complemento del biestable antes de ser activado.

Debido a las distintas nomenclaturas aceptadas en diversos manuales, se debe considerar que la salida Q puede denominarse Q_t y que el estado posterior de Q puede denominarse indistintamente Q_{t+1} o Q'. De la misma manera, no Q es equivalente a $\overline{Q}$ y, por lo tanto, $\overline{Q}'$ es el estado posterior de $\overline{Q}$.

6.4.1.2 Biestable R–S sincrónico (temporizado)

En los sincrónicos o temporizados, la presencia de un pulso en una entrada especial determina el instante a partir del cual las entradas modifican su estado. Los pulsos de sincronismo son generados por un sistema de reloj y, por eso, llamaremos a la señal que lo representa Ck (*clock*).

La señal Ck oscila entre *0* y *1* a intervalos regulares, de manera que un biestable sincrónico es accionado cuando *Ck = 1*. Así, si varios biestables están relacionados entre sí por la señal de sincronismo, se modifican en el mismo instante.

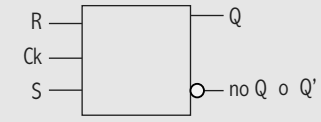


Fig. 6.28. Diagrama de bloque.

6.4.1.3 Biestable J–K sincrónico

El comportamiento del biestable *JK* es similar al del biestable *RS,* sólo que para las entradas *J* y *K* iguales a "*1*", las salidas se complementan según la siguiente tabla de verdad:

Tabla 6-13. Tabla de verdad de un biestable J-K sincrónico

J	*K*	*Q t*	*Q t+1*
0	*0*	*0*	*0*
0	*0*	*1*	*1*
0	*1*	*1*	*0*
0	*1*	*1*	*0*
1	*0*	*0*	*1*
1	*0*	*0*	*1*
1	*1*	*0*	*1*
1	*1*	*1*	*0*

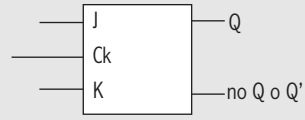


Fig. 6.29. Diagrama de bloque de un biestable J-K sincrónico.

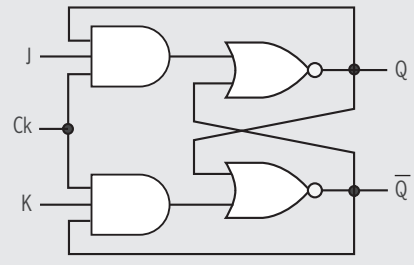


Fig. 6.30. Diagrama de lógica de un biestable J-K sincrónico.

Recuérdese que en la tabla la salida *Q* del diagrama es *Q t* y que el estado posterior de la misma señal *Q* es *Q t + 1*

6.4.1.4 Biestable T sincrónico

El biestable *T* se puede analizar como un derivado del *J-K*, que se logra considerando que las entradas *J* y *K* son iguales, de manera que cuando éstas valen *00* o *11*, se logra que se comporten según la tabla de funcionamiento siguiente:

Tabla 6-14. Tabla de verdad

J	*K*	*Q t*	*Q t+1*	
0	*0*	*0*	*0*	
0	*0*	*1*	*1*	
0	*1*	*1*	*0*	no se cumple
0	*1*	*1*	*0*	
1	*0*	*0*	*1*	
1	*0*	*0*	*1*	
1	*1*	*0*	*1*	
1	*1*	*1*	*0*	

Tabla 6-15. Tabla de verdad

T	*Q t*	*Q t+1*	
0	*0*	*0*	no se cumple
0	*1*	*1*	
1	*0*	*1*	no se cumple
1	*1*	*0*	

Fig. 6.31. Diagrama de bloque.

Fig. 6.32. Diagrama de bloque.

Al igualar las entradas *J-K*, el biestable sólo recuerda y complementa.

6.4.1.5 Biestable D sincrónico

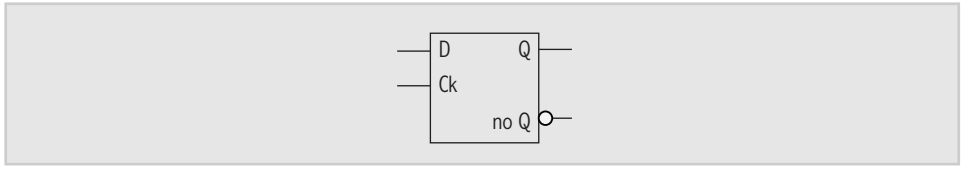


Fig. 6.33. Diagrama de bloque de un biestable D sincrónico

Aquí se utilizó un *R-S* sincrónico con entradas unidas para representar un biestable *D* sincrónico.

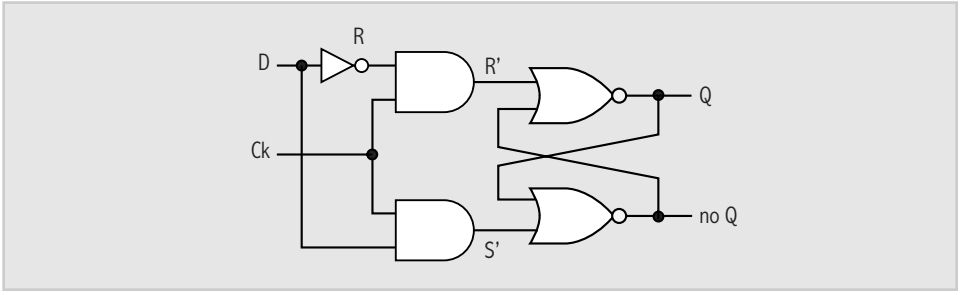


Fig. 6.34. Diagrama de lógica de un biestable D sincrónico

El biestable *D* también se puede analizar como un derivado del *J-K,* unificando las entradas *J* y *K* negada, de manera que cuando éstas son distintas (valen *01* o *10)*, se logra que se comporten según la siguiente tabla de verdad:

Tabla 6-16. Tabla de verdad

D	*Qt*	*Q t+1*
0	*0*	*0*
0	*1*	*0*
1	*0*	*1*
1	*1*	*1*

6.4.2 Registros

Un registro es una función digital, que suele utilizarse para retener información binaria temporariamente. Puede estar formado por uno o más biestables. La CPU tiene una pequeña memoria de acceso rápido, formada por varios registros que almacenan información intermedia, parte de los cuales ya mencionamos como registros auxiliares (PC, IR, AC, SR, etc.).

6.4.2.1 Registros paralelo-paralelo

Algunos registros cumplen la función de transferir la información que viene por las líneas de entrada en paralelo a salidas en paralelo. Estos registros reciben el nombre de paralelo-paralelo. En este ejemplo se utilizaron biestables D.

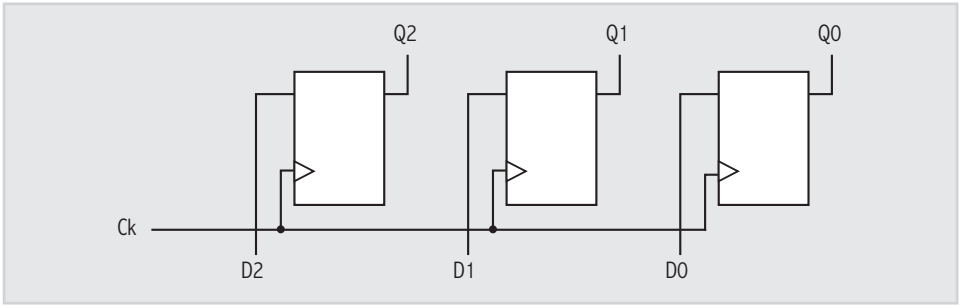


Fig. 6.35. Diagrama de bloque de un registro paralelo-paralelo.

6.4.2.2 Registros contadores

6.4.2.2.1 Contador progresivo de 8 eventos con biestables T

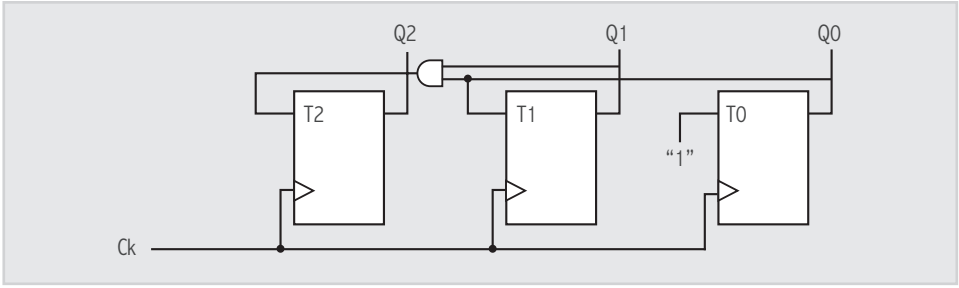


Fig. 6.36. Diagrama de bloque de un contador progresivo de 8 eventos con biestables T.

Como su nombre lo indica, un registro contador progresivo tiene la característica de contar eventos en orden ascendente, dependiendo de la cantidad de biestables que lo compongan. Si el registro está compuesto por n biestables, contará de *0* a $2^n - 1$, después de lo cual volverá a su estado inicial.

Para verificar su funcionamiento, veremos un ejemplo que se basa en un registro conformado por tres biestables "*T*" (*To*, *T1* y *T2*), donde la salida "*Q*" de cada uno de ellos ingresa en los biestables siguientes, relacionándose con compuertas AND, para posibilitar el conteo correcto. Cada uno de los biestables se activa en el mismo instante de tiempo *Ck* proveniente del reloj o temporizador. Este registro sólo puede contar hasta *8* (*0-7*) eventos, ya que tiene tres biestables.

Para seguir la tabla de funcionamiento de este registro contador progresivo de *8* eventos, confeccionados con biestables *T*, debemos partir de un estado inicial, donde tanto las salidas "Q" de los biestables como las entradas a los biestables *T* (*T1* y *T2*) se encontrarán inicializadas a *0* (estas últimas por recibir la información de los biestables que las preceden). La entrada *T0* siempre recibirá un valor *1* (*1*). Por cada pulso de reloj, todos los biestables serán activados en forma simultánea, de acuerdo con el valor que posean en su entrada, teniendo en cuenta la tabla de funcionamiento del biestable *T*. Entonces, en el primer instante de tiempo, en el primer biestable, *Q0* está en *0* y recibe un *1* por *T0*; por lo tanto, debe complementar el valor que tenía y dejar un *1* en *Q0* **antes del siguiente Ck**. En el segundo biestable, *Q1* está en *0* y recibe un *0* por *T1* **relacionado con Q0 anterior**; por lo tanto, debe permanecer con el valor que tenía, *0*. Asimismo, en el último biestable, *Q2* está en *0* y recibe un *0* de *Q1* y un *0* de *Q0*, que ingresan en una compuerta AND, cuya salida por el producto lógico es *0*; por lo tanto, debe permanecer con el valor que tenía, o sea, *0*.

Si completamos la tabla de funcionamiento por cada pulso de reloj, veremos, hasta ahora, que los valores obtenidos reflejan el cambio de *0*, que era el estado inicial, a *1*. Antes de que observemos cómo el registro se activa por segunda vez, debemos recordar que las salidas del pulso anterior ingresan en los biestables subsecuentes durante el nuevo pulso, como lo habíamos indicado antes.

Con el segundo pulso de reloj, en el primer biestable *Q0* hay un *1*, y recibe un *1* por *T0*; por lo tanto, debe complementar el valor que tenía y deja un *0* **antes del siguiente Ck**. En el segundo biestable, *Q1* está en *0* y recibe un *1* por *T1* **relacionado con el Q0 anterior**; por lo tanto, debe complementar el valor que tenía y deja un *1*. Asimismo, en el último biestable, *Q2* está en *0* y recibe un *0* por *Q1* y un *1* por *Q0*, que ingresan en la compuerta AND, cuya salida por el producto lógico es *0*, y este valor es el que ingresa por *T2*, provocando que el valor de *Q2* siga siendo *0*.

Al completar la tabla de funcionamiento, observamos que los valores de salida de los biestables (*Q2*, *Q1* y *Q0*) muestran un incremento de una unidad; en este último caso, *2* en binario. Si completamos el análisis de la tabla de funcionamiento, podremos observar que en el octavo pulso de reloj las salidas de los biestables vuelven a su valor original, o sea, *0* en binario.

Tabla 6-17. Tabla de funcionamiento del registro

Ck	*Q2*	*Q1*	*Q0*	*T2*	*T1*	*T0*
	0	*0*	*0*			
1	*0*	*0*	*1*	*0*	*0*	*1*
2	*0*	*1*	*0*	*0*	*1*	*1*

Ejercicio:

Complete la tabla hasta lograr que las salidas del registro vuelvan a *0*.

6.4.2.2.2 Contador regresivo de 8 eventos (con biestables T)

Un registro contador regresivo, en forma inversa al anterior, tiene la característica de contar eventos en orden descendente, según la cantidad de biestables que lo compongan. Si el registro está compuesto por n biestables, contará de 2^n-1 a *0*, después de lo cual volverá a su estado inicial (2^n-1). Para verificar su funcionamiento, también veremos un ejemplo que está basado en un registro conformado por tres biestables "*T*" (*T2*, *T1* y *T0*), donde, en este caso y por ser regresivo, la salida "*no Q*" de cada uno de ellos ingresa en los biestables siguientes, relacionándose con compuertas AND, para posibilitar, así, el funcionamiento correcto.

Cada uno de los biestables se activa en el mismo instante de tiempo *Ck* proveniente del reloj o temporizador. Como tiene tres biestables, este registro sólo puede contar, en forma regresiva, de siete a cero (*7-0*). En forma análoga, para seguir la tabla de funcionamiento de este registro contador regresivo de *8* eventos confeccionados con biestables *T*, debemos partir de un estado inicial, donde las salidas "*Q*" de los biestables se encontrarán inicializadas con *1* y las salidas "*no Q*", por ser el complemento de las salidas *Q*, se encontrarán inicializadas con *0*. Los valores de las salidas complementadas de *Q* serán los que ingresan en los biestables subsecuentes; por lo tanto, la tabla de funcionamiento se verá afectada de acuerdo con la evaluación que se encuentra más adelante.

La entrada *T0* siempre recibirá un valor *1* y, por cada pulso de reloj, todos los biestables se activarán en forma simultánea de acuerdo con el valor que posean en su entrada, teniendo en cuenta la tabla de funcionamiento del biestable *T*. A partir de estas premisas, el mecanismo de funcionamiento de este registro es similar al estipulado para el contador progresivo de *8* eventos, y no vale la pena desarrollarlo de nuevo. Sólo mencionaremos que es conveniente incorporar a la tabla de funcionamiento los valores complementados de las salidas *Q* de cada biestable para realizar un mejor seguimiento.

Ejercicio:

Dibuje un contador regresivo de *3* bits con biestables T, represente su tabla de funcionamiento y explique cómo funciona el registro para los dos primeros cambios. Utilice como referencia la siguiente tabla:

Ck	*Q2*	*Q1*	*Q0*	*no Q1*	*no Q0*	*T2*	*T1*	*T0*
	1	*1*	*1*	*0*	*0*			
1	*1*	*1*	*0*	*0*	*1*	*0*	*0*	*1*
2	*1*	*0*	*1*	*1*	*0*	*0*	*1*	*1*

6.4.2.3 Registros con facilidad de desplazamiento

En el set de instrucciones de una computadora hay instrucciones de desplazamiento, ya que la multiplicación de dos números se puede realizar sobre la base de sumas sucesivas y desplazamientos, al igual que las divisiones que, también, se pueden realizar sobre la base de restas sucesivas y desplazamientos.

Recuerde el mecanismo de desplazamiento de la coma al multiplicar o dividir por la base en el sistema decimal. En el sistema binario los desplazamientos se realizan corriendo los bits de un registro hacia la derecha o hacia la izquierda. Aunque los procesadores sencillos sólo disponían de operaciones de desplazamiento de una posición a derecha o izquierda, casi todos los procesadores actuales permiten el desplazamiento de varios bits en ambos sentidos.

Para que esto se produzca, las celdas binarias que forman parte del registro tienen que estar relacionadas de manera que, por cada orden que reciban, se produzca un desplazamiento de un bit, o sea que la salida de una celda alimente la entrada de la otra.

6.4.2.3.1 Desplazamientos lógicos

Cuando el bit del extremo queda libre y se rellena con un *0*, se indica que el registro produce un desplazamiento lógico.

6.4.2.3.2 Desplazamientos circulares

Cuando en un registro de desplazamiento los n bits que se vacían en un extremo se completan con los que salen por el otro, se indica que el registro admite un desplazamiento circular. Con estos desplazamientos no hay pérdida de bits, sino que éstos circulan a través del registro.

6.4.2.3.3 Desplazamientos aritméticos

Los desplazamientos aritméticos afectan a números que pueden ser o no signados. Son parecidos a los desplazamientos lógicos, pero mantienen el signo del número. Estos desplazamientos producen una multiplicación o una división por una potencia de *2*.

6.4.2.3.4 Desplazamientos concatenados

Son desplazamientos que afectan a un conjunto concatenado de dos o más elementos, que pueden ser:

a) Dos registros.

b) Un registro con el biestable de acarreo.

c) Un registro con el biestable de signo.

d) Combinaciones de algunos de los anteriores.

A continuación, se tratará el tema de la relación entre los desplazamientos y la rotación de bits en un registro, y las instrucciones que activan su operación. En cualquiera de los casos, los bits del registro pueden desplazarse o rotar su información dentro del registro o fuera de él.

Un desplazamiento mueve los bits del registro en forma lineal, mientras que una rotación lo hace de manera circular.

Las instrucciones que utilizaremos a modo de ilustración son:

SHL (*Shift Left*: desplazamiento lógico a la izquierda).

SHR (*Shift Right*: desplazamiento lógico a la derecha).

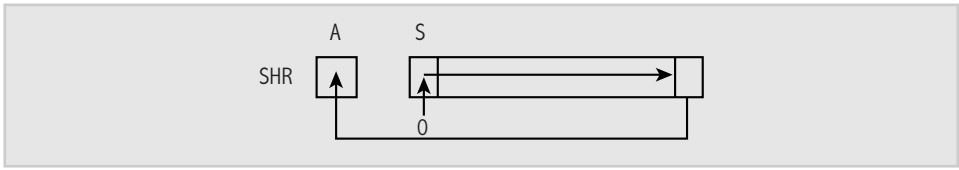
SAL (*Shift Arithmetic Left*: desplazamiento aritmético a la izquierda, para valores signados).

SAR (*Shift Arithmetic Right*: desplazamiento aritmético a la derecha, para valores signados).

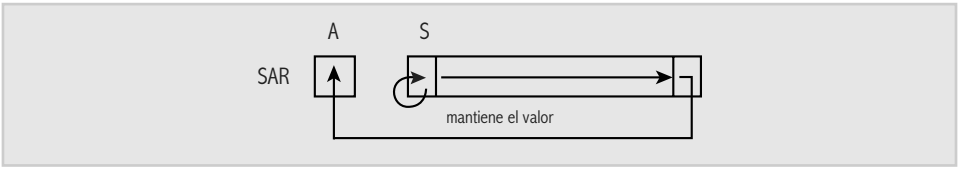
Los desplazamientos lógicos no permiten conservar el signo del operando, razón por la cual para valores signados se deberán realizar desplazamientos aritméticos.

SHL y SAL desplazan todos los bits hacia la izquierda, menos el más significativo (bit *15* en el caso de una palabra y *7* en el caso de un byte), que se inserta en el bit de acarreo del registro de banderas (*status register*). A su vez, se coloca un *0* en el bit menos significativo (bit *0* en ambos casos), ya sea una palabra o un byte.

SHR desplaza todos los bits hacia la derecha, excepto el menos significativo (bit *0*), que lo inserta en el bit de acarreo y coloca un *0* en el bit más significativo.



SAR desplaza todos los bits a la derecha, excepto el menos significativo (bit *0*), que lo inserta en el bit de acarreo y no modifica el bit más significativo (bit de signo), o sea, mantiene el valor del signo.



El desplazamiento aritmético a izquierda multiplica el contenido del registro por la base, en este caso por *2*; mientras que el desplazamiento aritmético a derecha divide el contenido del registro por la base. En el caso de la instrucción SAR, la división puede producir un resto, lo que provoca un redondeo "por defecto" del resultado.

Las instrucciones que permiten rotar un bit a la vez en un registro son:

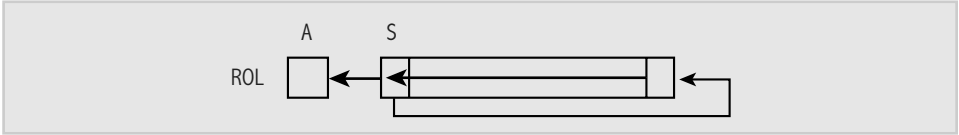
ROL (*Rotate Left*: rotación a la izquierda).

ROR (*Rotate Right*: rotación a la derecha).

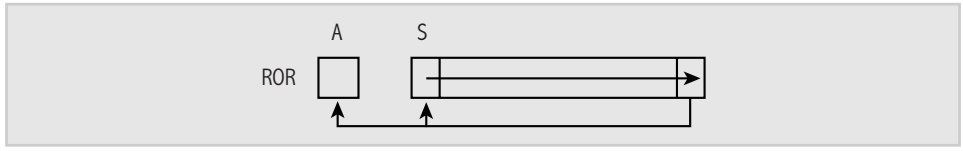
RCL (*Rotate Left Through Carry*: rotación con acarreo a la izquierda).

RCR (*Rotate Right Through Carry*: rotación con acarreo a la derecha).

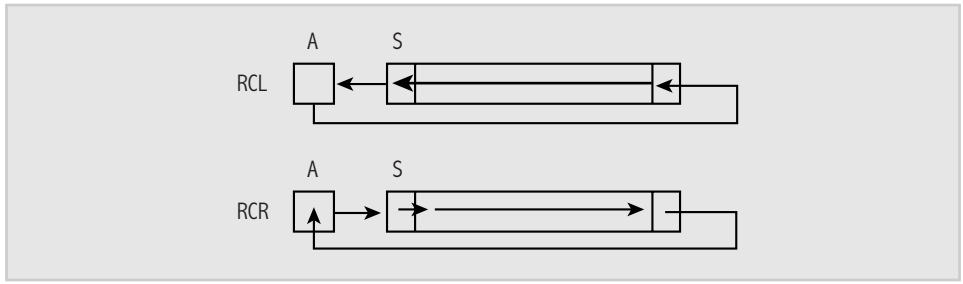
ROL rota los bits del registro, ya sea de *16* u *8* bits, a izquierda, colocando el bit más significativo en el bit de acarreo del registro de banderas y, también, en el bit menos significativo del registro.



ROR rota los bits del registro a la derecha, insertando el bit menos significativo en el de acarreo del registro de banderas y en el bit más significativo del byte o palabra.

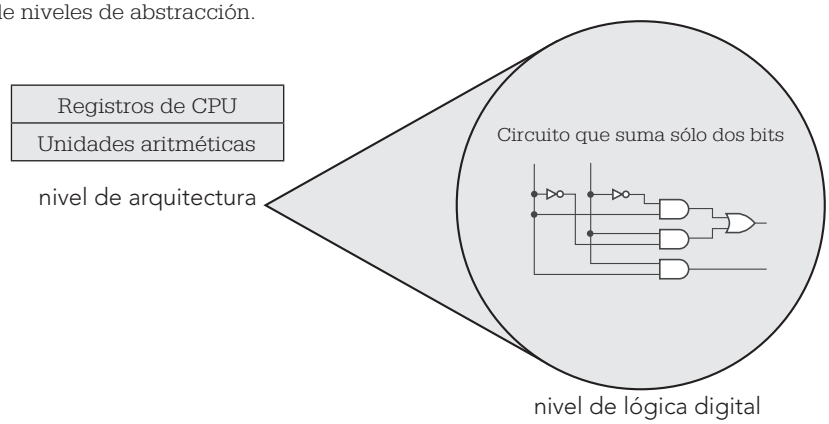


RCL y RCR utilizan el bit de acarreo del registro de banderas como una extensión del propio registro durante la rotación. Los bits de estas instrucciones rotan a izquierda o derecha, respectivamente, arrastrando con ellos al bit de acarreo.



6.5 Resumen

El estudio de la lógica digital permite la representación y el análisis de un componente definido como un bloque funcional en la arquitectura. En la figura siguiente se observa la diferencia de niveles de abstracción.



Los circuitos combinacionales son aquellos cuyas salidas dependen sólo de la combinación binaria establecida en las entradas; en el capítulo se incluyeron las funciones digitales más importantes para la comprensión del funcionamiento del hardware al nivel de abstracción de la arquitectura de una computadora. Así, se puede entender, por ejemplo, cómo se decodifica una dirección en un acceso *random*, si se considera la combinación de las entradas de un decodificador como la dirección de una posición en la matriz de memoria, y si se considera que cada una de las salidas del decodificador es la entrada de habilitación de cada una de las celdas que constituyen la posición direccionada.

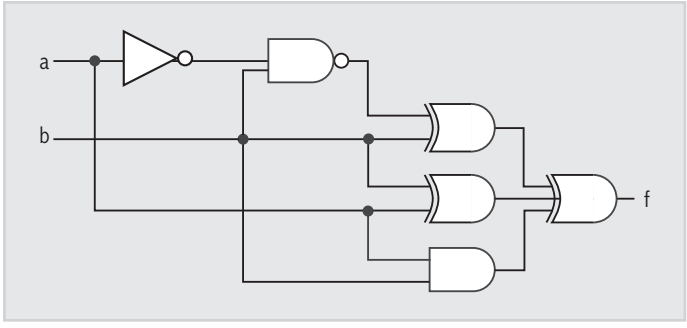
Los circuitos secuenciales son los sistemas digitales cuyas salidas no sólo dependen de sus entradas en un momento dado, sino también de su estado anterior. Se los puede pensar como un sistema combinacional con entradas independientes y la realimentación de una o más de sus funciones de salida. Las salidas adoptarán un "valor nuevo", que dependerá de las

entradas y de "su valor anterior". En un sistema secuencial sincrónico, el instante en el que se "habilitan" las entradas se produce durante un pulso de reloj.

Los biestables son circuitos secuenciales básicos y permiten la construcción de circuitos secuenciales más complejos. Se utilizan para implementar matrices de memoria estática y registros de desplazamiento, contadores, de banderas, punteros, etcétera. Los registros son elementos de memoria de poca capacidad y alta velocidad, cuyo tamaño se expresa en bits. En una microarquitectura se estudian los registros asociados a la CPU no sólo para conocer su funcionamiento, sino también para utilizarlos con instrucciones del set de instrucciones que la CPU puede ejecutar. Estos registros se conocen como banco de registros de la CPU o registros del entorno de la programación de aplicaciones. Los registros pueden relacionarse entre sí por medio de un bus. La relación de cada registro con el bus se puede realizar utilizando un multiplexor para la transferencia de-registro-a-bus y con un demultiplexor para la transferencia de-bus-a-registro. Otra forma de establecer la transferencia es utilizar compuertas *buffer* triestado de manera que, en estado de alta impedancia, el registro pueda "desconectarse del bus".

6.6 Ejercicios propuestos

1) Para el siguiente circuito:



a) Halle la tabla de verdad.
b) Exprese la forma normal conjuntiva.
c) Exprese la forma normal disyuntiva.
d) Rediseñe el diagrama de lógica utilizando la forma normal disyuntiva.
e) Rediseñe el diagrama de lógica utilizando la forma normal conjuntiva.
f) Demuestre si los tres circuitos son equivalentes para la combinación de las entradas a=0 b=1 (recorrer cada diagrama con el valor binario que resulta a la salida de cada compuerta lógica hasta llegar a la función).

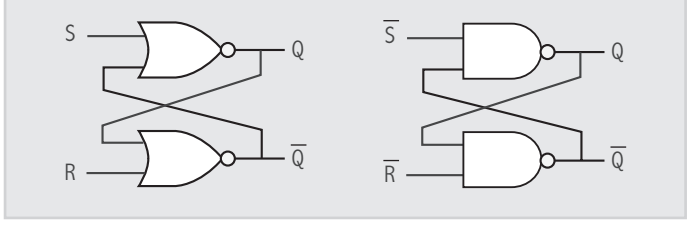
2) Diseñe un circuito que tome un número de 4 bits (a, b, c, d) y produzca una salida f que sea verdadera si la entrada presenta un número impar.

3) Diseñe un circuito que tome un número BCD de 4 bits (a, b, c, d) y produzca cuatro salidas f0, f1, f2, f3 en código Aiken.

4) Un circuito digital activa las cuatro luces de un semáforo. La combinación 00 activa la luz R (roja), la 01 activa la luz A (amarilla), la 10 activa la luz V (verde) y la 11, la luz G (flecha de giro). Represente la tabla de verdad con entradas a y b y salidas R, A, V, G y demuestre que este circuito en realidad es un decodificador.

5) Diseñe el diagrama de lógica del circuito anterior y agregue una entrada de habilitación para que, cuando ésta se encuentre en 0, apague todas las luces.

6) Analice el comportamiento de los siguientes circuitos y realice la tabla de verdad para cada uno de ellos



7) a. Implemente un *flip-flop* tipo T sobre la base de un *J-K*.

b. Realice la tabla de verdad del *flip-flop* tipo T.

8) a. Implemente un biestable tipo *J-K* sobre la base de un *R-S* y compuertas NOR.

b. Realice la tabla de verdad del *flip-flop* tipo *J-K*.

9) Diseñe el diagrama de un contador sincrónico progresivo que realice la secuencia de cuenta binaria de 0000 a 1111 y muestre su comportamiento en una tabla de verdad.

6.7 Contenido de la página Web de apoyo

El material marcado con asterisco (*) sólo está disponible para docentes.

Resumen gráfico del capítulo

Simulación

Decodificador de dos entradas

Display BCD siete segmentos

Autoevaluación

Video explicativo (01:34 minutos aprox.)

Audio explicativo (01:34 minutos aprox.)

Evaluaciones Propuestas*

Presentaciones*