Subsistemas típicos en circuitos integrados

PID_00206008

Marc Bara Iniesta

Universitat Oberta de Catalunya

Índice

Introducción

Ya conocemos algunos conceptos relacionados con los circuitos integrados. En concreto, hemos estudiado cuál es la estructura física de los circuitos integrados, y hemos expuesto las diferentes etapas de su proceso de fabricación. También hemos analizado con detalle la tecnología CMOS, compuesta por transistores MOSFET, que permite disponer de puertas lógicas básicas como las NAND, NOR o NOT con una serie de características y ventajas que ahora ya conocemos con precisión.

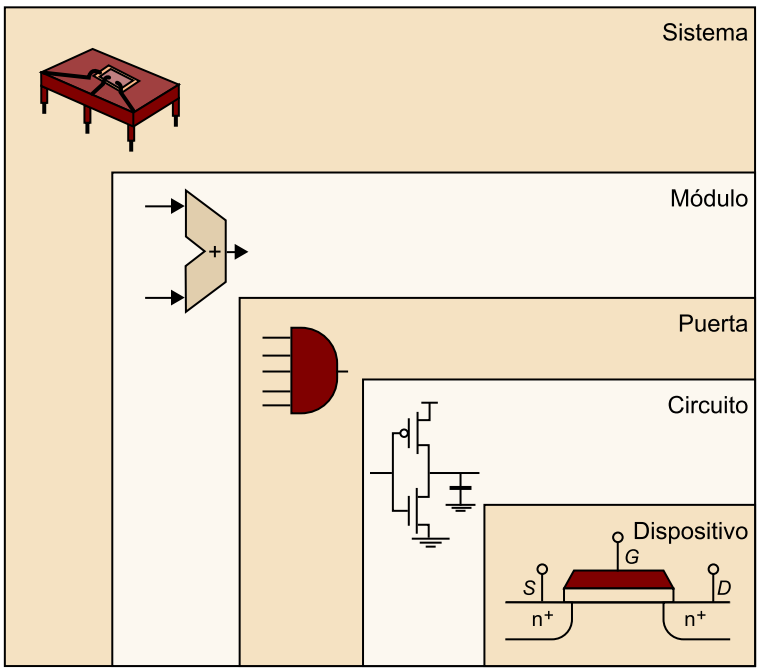
Estamos, pues, preparados para "subir" al siguiente nivel de abstracción en los circuitos integrados, y podemos hablar ya de **subsistemas**. Con este término nos referimos a agrupar conceptualmente conjuntos de dispositivos, puertas lógicas, etc., como los que conocemos hasta ahora para llevar a cabo operaciones o funciones más complejas.

Si recordamos el esquema representado en la figura 1, podemos ilustrar lo que explicamos: ya conocemos la estructura física del dispositivo, y también sabemos cómo utilizar el dispositivo más básico (el transistor) como circuito eléctrico capaz de sintetizar puertas lógicas. A partir de aquí, tenemos los elementos fundamentales para construir los denominados *módulos* o *subsistemas*.

El término *sistema*

En los módulos siguientes de la asignatura, utilizaremos el término *sistema* para referirnos a circuitos digitales complejos, como por ejemplo las FPGA y los ASIC.

Figura 1. Niveles de abstracción de diseño en circuitos



En resumen, las puertas AND, OR, NOT, NAND y NOR serán consideradas como dispositivos de varios terminales caracterizados por su tabla de verdad; serán módulos funcionales de bajo nivel, a partir de los cuales se diseñan los

módulos de nivel intermedio, los **subsistemas**, que se corresponden con un nivel de integración de escala intermedia. Se habla en este nivel de *funciones intermedias*, en el sentido de que aún no estamos exponiendo los sistemas más complejos (que se ven en otros módulos).

En este nivel intermedio, nos centraremos en primer lugar en los **subsistemas digitales**, y concretamente en dos tipos de **subsistemas combinacionales y secuenciales**. A lo largo del módulo, describiremos con detalle el funcionamiento de estos subsistemas. En asignaturas más fundamentales de electrónica digital ya habréis aprendido qué quiere decir *combinacional* y *secuencial*, así como la diferencia entre los dos términos y algunos circuitos básicos. Aquí trataremos de profundizar en estos tipos de circuitos tan utilizados en la electrónica actual.

Por un lado, veremos **subsistemas combinacionales** con tipos de circuitos como los sumadores, restadores, comparadores, generadores de paridad, etc., que tienen como objetivo llevar a cabo operaciones simples pero que son la base de todos los sistemas digitales de comunicación, instrumentación, cálculo y control. Recordemos que reciben su nombre por la "combinación" de los bits de las palabras de entrada para generar bits a la salida que se correspondan con funciones aritméticas básicas. No hacen uso de valores anteriores ni disponen de memoria de un estado previo.

Por otro lado, podremos hablar de **subsistemas secuenciales**, en los que sí se hace uso de estados previos para generar el siguiente estado a la salida del circuito. Entre los circuitos que agrupamos en este conjunto, podemos encontrar los registros y los contadores.

Finalmente, abordaremos algunos ejemplos de **subsistemas típicos del mundo analógico**, como son el interruptor analógico, el multiplexor y las referencias de voltaje.

Objetivos

Los objetivos de este módulo se pueden resumir en los puntos siguientes:

1. Estudiar la escala intermedia de integración de circuitos entre los dispositivos básicos, como son los transistores y las puertas lógicas, y los sistemas complejos programables como las FPGA y los ASIC.

2. Entender el funcionamiento de subsistemas ampliamente utilizados para operaciones combinacionales, como la suma, la resta y la comparación.

3. Saber describir subsistemas secuenciales, como los registros, registros de desplazamiento y contadores.

4. Tener la base de conocimiento necesaria para hacer uso de estos subsistemas con circuitos integrados comerciales, y entender cómo escalarlos en número de bits mediante conexiones en cascada.

5. Conocer subsistemas de base analógica, como los interruptores, los multiplexores y las referencias de voltaje.

1. Subsistemas típicos en circuitos digitales combinacionales

Tal y como hemos expuesto en la introducción, empezaremos hablando de subsistemas digitales que llevan a cabo **funciones combinacionales**, es decir, a partir de una información digital representada a un formato concreto a la entrada, llevan a cabo una función determinada, por ejemplo una función aritmética como la suma o la resta. Empezaremos por este tipo de circuitos, sumadores y restadores y, para hacerlo, debemos hablar brevemente de cómo se representa el signo en sistemas binarios. En este material se asume que tenemos los conceptos básicos de electrónica digital aprendidos, pero es necesario que destaquemos las maneras de representar el signo antes de hablar de circuitos con funciones aritméticas.

1.1. Representación de los signos positivo y negativo en sistemas binarios

Para hablar de circuitos que llevan a cabo operaciones aritméticas de n bits, debemos representar también los números enteros negativos y el rango de valores posibles, que es limitado. Veremos que una palabra de n bits solo admite 2^n configuraciones diferentes y, de estas, una parte se dedicará a los números positivos, una o dos al cero y el resto se utiliza para representar números negativos. De este modo, por ejemplo, si nos limitamos a una máquina que trabaje con palabras de 4 bits, el máximo entero positivo (1111) vale 15 y no puede haber operandos ni resultados de operaciones que superen este valor. Si queremos incluir números negativos, tendremos que reservar un bit para el signo y, por lo tanto, el máximo número decimal entero será ahora el que corresponde a los 3 bits restantes (111), es decir, al 7. El cuarto bit indicaría el signo del número decimal.

Hay tres formas básicas que permiten la representación conjunta de números positivos y negativos:

1) Signo y magnitud (S&M).

2) Complemento a 1 (Ca1).

3) Complemento a 2 (Ca2).

Dependiendo de si utilizamos una u otra, nuestro circuito sumador será lógicamente distinto.

Las tres formas de representación utilizan el bit más significativo para codificar el signo del número. Los números que empiezan por 0 son positivos y los que empiezan por 1 son negativos. El resto de los bits codifican su magnitud.

De este modo, en el **formato signo y magnitud** (**S&M**) la codificación es inmediata. Primero se pone el signo (0 = positivo, 1 = negativo) y después los bits de la magnitud. Por lo tanto, cambiar un número a negativo consiste simplemente en cambiar el bit de signo por su complementario. Observamos un hecho interesante: aquí el cero tiene una doble representación. Por ejemplo, con 4 bits tendríamos +0 = 0000 y –0 = 1000.

Para sumar dos números positivos o dos números negativos, sumamos sus bits de magnitud y asignamos al resultado el signo de los sumandos. De este modo, por ejemplo (resaltamos el bit de signo en negrita):

0100 (+4) + **0**010 (+2) = **0**110 (+6)

Cuando el signo de los operandos no coincide, restamos el de menor magnitud y el resultado hereda el signo del de mayor magnitud. Por ejemplo:

0100 (+4) + 1010 (-2) = 0010 (+2)

Por lo tanto, vemos que aunque la representación es aparentemente la más simple, esto complica relativamente la síntesis de las operaciones. Si un sumador tiene que gestionar números negativos con la representación S&M, además del sumador deberá incluir un restador y dos comparadores, uno para el signo y el otro para la magnitud, para hacer las operaciones correctamente, como hemos visto en el ejemplo.

En la **representación en complemento a 1** (**Ca1**), los números negativos se obtienen complementando uno a uno todos los bits de la representación del número positivo correspondiente. De este modo, por ejemplo, con 4 bits el número +3 sería 0011 y el –3 sería 1100. Observamos que también hay dos representaciones diferentes para el cero (+0 = 0000, –0 = 1111), pero la ventaja de esta representación es que la resta se implementa con una negación y una suma: A – B = A + (–B).

Para eliminar la complicación asociada a la doble representación del cero, se utiliza el método del complemento a la base, es decir, la **representación en complemento a 2** (**Ca2**). En este sistema, los números positivos se representan colocando un cero a la izquierda de los bits que representan la magnitud, como en las otras dos representaciones. Los números negativos se representan a partir de los positivos correspondientes complementando todos los bits y sumando un 1 al resultado. Al igual que en la representación Ca1, la resta se implementa con una negación y una suma, pero ahora el cero solo tiene una representación (+0 = 0000).

Para entender mejor todos los sistemas, conviene observar la tabla 1, en la que se ven las analogías y diferencias para palabras de 4 bits (tres de magnitud y uno de signo).

Tabla 1. Representaciones de magnitud y signo con palabras de 4 bits

Configuraciones binarias (4 bits)	Binario puro	S&M	Ca1	Ca2
0 000	0	+0	+0	0
0 001	1	+1	+1	+1
0 010	2	+2	+2	+2
0 011	3	+3	+3	+3
0 100	4	+4	+4	+4
0 101	5	+5	+5	+5
0 110	6	+6	+6	+6
0 111	7	+7	+7	+7
1 000	8	–0	–7	–8
1 001	9	–1	–6	–7
1 010	10	–2	–5	–6
1 011	11	–3	–4	–5
1 100	12	–4	–3	–4
1 101	13	–5	–2	–3
1 110	14	–6	–1	–2
1 111	15	–7	–0	–1

En la tabla 1 se representan las dieciséis configuraciones binarias posibles para palabras de 4 bits. En la parte derecha, aparecen los distintos valores decimales equivalentes dependiendo del modo de representación utilizado. De este modo, en la columna “Binario puro” aparece el valor decimal equivalente cuando la palabra de 4 bits se interpreta en la codificación binaria habitual, sin admitir números negativos. En las otras tres columnas, solo nos quedan 3 bits para la magnitud. Por lo tanto, representamos del +0 al 7 y del –0 al –7 (en S&M y Ca1), o del 0 al 7 y del –1 al –8 en Ca2.

Para cambiarle el signo a un número con la representación S&M, hay que cambiar el valor de su bit más significativo. En cambio, en la representación Ca1 debemos complementar todos los bits, y en la representación Ca2, además de esto, hay que hacer una operación de suma de una unidad. Estas representaciones Ca1 y Ca2 presentan ventajas de simplicidad a la hora de implementar operaciones aritméticas.

Podemos utilizar cualquiera de estas representaciones. Lo importante es mantener la consistencia e interpretar de manera adecuada los resultados según el formato de representación.

En cualquier representación o sistema, una cuestión que hay que tener en cuenta es la del desbordamiento[1]. El **desbordamiento** se produce cuando el resultado de una operación genera un número que no cabe en la representación para un cierto tamaño de la palabra, es decir, faltan bits para representar el resultado de la operación. Por ejemplo: (–7) + (–7) = –14, que no podemos representar con 4 bits (tabla 1). Es un tema que trataremos en los circuitos sumadores y restadores que veremos a continuación.

(1)En inglés, *overflow*.

1.2. Sumadores y restadores en binario

Hemos visto las diferentes formas de representar números positivos y negativos con palabras binarias de *n* bits. Ahora tenemos que ver cómo pueden llevarse a cabo las diferentes operaciones aritméticas con circuitos combinacionales. Las dos operaciones aritméticas más básicas son la suma y la resta. De hecho, incluso el producto o la división se pueden sintetizar a partir de sumas y restas.

1.2.1. Sumadores en binario

Empezaremos por ver los denominados **semisumadores**[2], que son los circuitos más básicos que hacen la suma de 2 bits de entrada. En la tabla 2 podemos ver su tabla de verdad, en la que las entradas son *A* y *B* y la suma está representada por *S*. Como vemos, cuando sumamos dos 1 el resultado es un cero, pero se genera un bit adicional de arrastre o **acarreo**[3] que indica que "nos llevamos" este 1 a posiciones más significativas de la palabra.

(2)En inglés, *half-adders*.

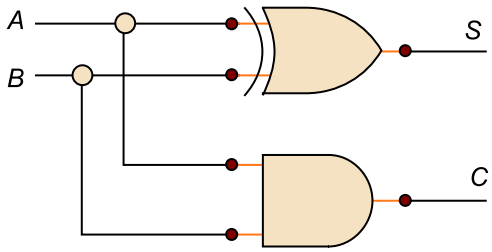
(3)En inglés, *carry*.

Tabla 2. Tabla de verdad del semisumador, con entradas *A* y *B*, suma *S* y acarreo *C*

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

Como se deduce de la tabla, la salida *S* se puede sintetizar con una OR exclusiva (XOR) y la salida *C* con una puerta AND. Así pues, tendríamos la representación del circuito de la figura 2. Observemos que las entradas de este semisumador son dos bits como sumandos, pero no es capaz de aceptar un bit adicional de acarreo que pudiera provenir de sumas de posiciones menos significativas. Es decir, no podríamos encadenar un semisumador tras otro.

Figura 2. Circuito semisumador



Ved también

Podéis ver cómo crear puertas lógicas a partir de transistores físicos –como por ejemplo los MOSFET y la tecnología CMOS– en el módulo "Circuitos CMOS" de esta asignatura.

Para tener en cuenta esta situación, podemos plantear un circuito conocido como *sumador* (o *sumador completo*[4]) en el que ya hay tres entradas, y así aceptar un acarreo a la entrada. Este circuito tendría como entradas A, B y C_{in}, y las dos salidas que ya conocemos, que son S y C (en este caso, renombrada a C_{out} para distinguirla del acarreo de entrada).

(4)En inglés, *full-adder*.

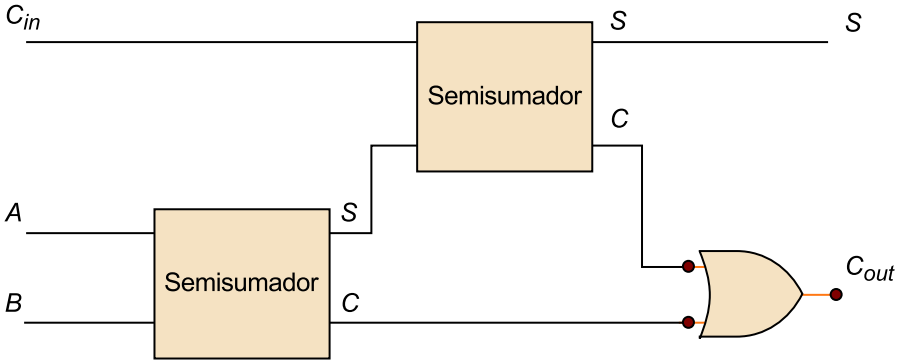
Tabla 3. Tabla de verdad del sumador, con entradas A y B, acarreo de entrada C_{in}, suma S y acarreo de salida C_{out}

A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Encontramos la tabla de verdad del sumador completo en la tabla 3.

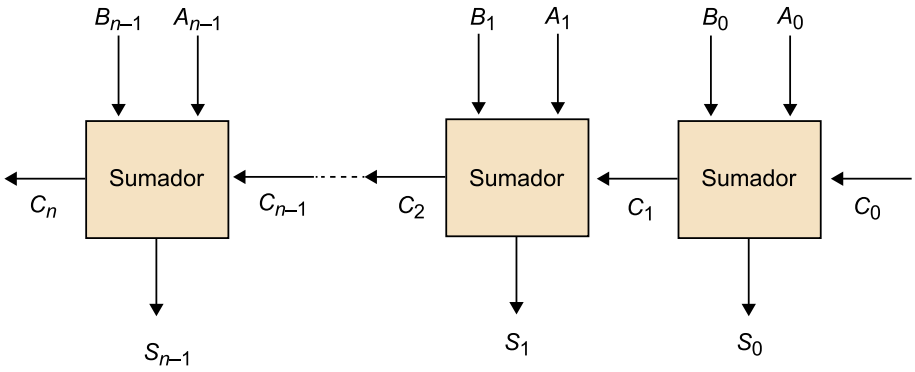
Para sintetizar este sumador, podemos hacer uso del circuito semisumador explicado anteriormente. En concreto, dispondríamos de un circuito como el de la figura 3 compuesto por dos semisumadores y una puerta OR, de modo que consiguiéramos la tabla de verdad deseada.

Figura 3. Circuito sumador basado en dos semisumadores y una puerta OR



Así pues, en este punto ya tenemos un circuito que es capaz de sumar dos palabras, A y B, de un bit cada una. En el caso general, sin embargo, queremos que la longitud de estas palabras sea tan grande como nos interese. Gracias al hecho de que el circuito básico sumador dispone de acarreo a la salida y a la entrada, lo podemos utilizar como celda fundamental para este propósito. Es bien sabido que para hacer la suma es preciso sumar las posiciones iguales de cada número A y B, y que si hay acarreo debe añadirse a la posición siguiente. De este modo, si suponemos que queremos sumar palabras de n bits, pensaremos en un circuito con n sumadores de un bit, como el de la figura 4. En la parte derecha vemos que empezamos sumando los dos bits menos significativos de las palabras A y B, que son A_0 y B_0, y prevemos que pueda haber algún acarreo anterior C_0. A partir de aquí, el acarreo se va sumando a los bits cada vez más significativos hasta llegar al n. En la figura 4, vemos que el resultado de la suma es la palabra S, que también tiene n bits, y puede haber un acarreo C_n.

Figura 4. Circuito sumador de palabras de n bits basado en sumadores de 1 bit



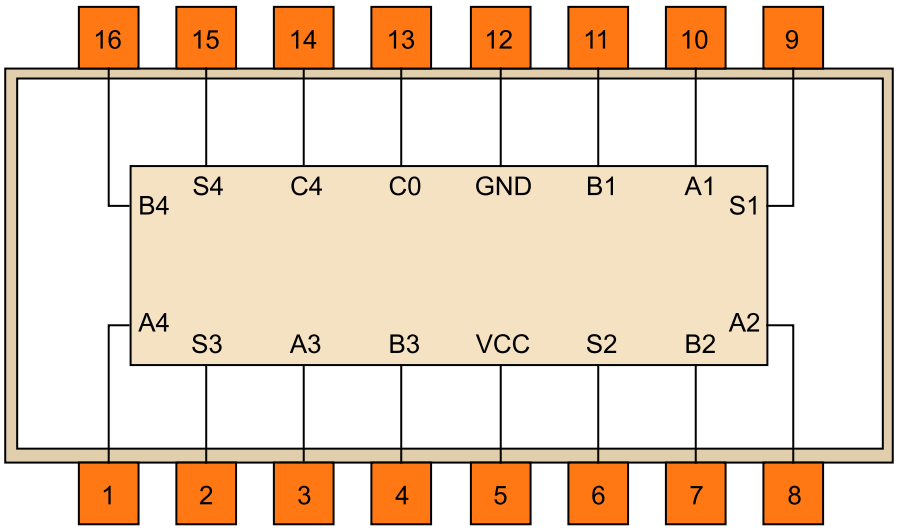
Se trata, pues, de una configuración en paralelo en la que todos los bits de los números binarios A y B entran de manera simultánea en el circuito. El acarreo de salida de cada sumador de bit se conecta a la entrada de acarreo de la etapa siguiente. Esta forma de operación exige que todos los bits de los dos números se encuentren disponibles en el mismo instante y permanezcan allí hasta que se hayan producido todos los acarreos –venciendo así el retraso de propagación de la familia lógica correspondiente– y la suma correcta se encuentre disponible y estable en los terminales de salida. Con este razonamiento, ya podemos

deducir que hay ciertas limitaciones como consecuencia de este retraso en la velocidad del sumador, puesto que debemos esperar hasta que el acarreo atraviese los *n* módulos desde el bit menos significativo hasta el más significativo.

En tecnología TTL, el circuito integrado sumador más utilizado es el 7483, que permite la suma *S* de dos números A y *B* de 4 bits cada uno. En la figura 5, vemos representado un esquema de sus pines[5]. Tiene cuatro salidas para el resultado (los cuatro bits de *S*), más una salida del acarreo final (C4) y una entrada extra para el acarreo inicial (C0), de modo que es posible conectar más de un sumador 7483 en cascada para efectuar operaciones con números mayores de 4 bits.

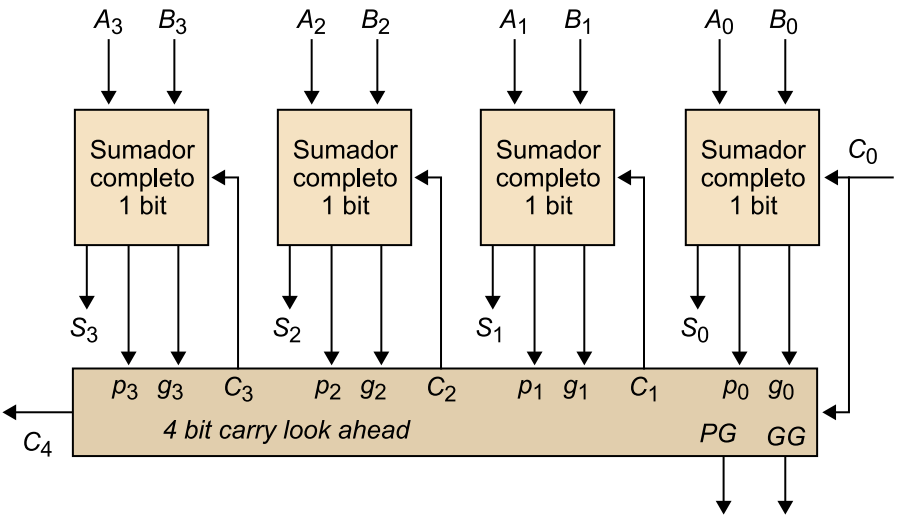
[5]En inglés, *pins*.

Figura 5. Esquema de un circuito integrado 7483 de 16 pines apto para la suma de dos números de 4 bits



Para reducir el tiempo de la operación suma (marcado, como decimos, por el camino del acarreo a través de los bloques), los fabricantes hacen una modificación en la estructura interna del circuito de modo que el acarreo no se desplaza en serie de bit en bit, sino que lo hace en paralelo, por lo que la entrada del acarreo inicial afecta directamente a todas las salidas. Como contrapartida, el circuito combinacional se complica de manera considerable a medida que aumenta el número de bits. En la figura 6 podemos ver el diagrama de bloques modificado respecto de la figura 4, en el que –recordemos– el acarreo se propaga entre sumadores de 1 bit. Aquí, en cambio, hay un bloque separado, denominado *carry look ahead*, que se encarga de hacer la propagación a todos los sumadores al mismo tiempo.

Figura 6. Sumador de 4 bits con generación de acarreo en paralelo



Fuente: Wikipedia.

Además, hay que destacar que aquí cada sumador de 1 bit se ha modificado para que dé las salidas p y g, que se corresponden con hacer una puerta AND y una puerta OR de las entradas A y B. Es decir, $p = A \cdot B$ y $g = A + B$ (en la figura ya vienen estos parámetros con subíndice según el bit). Con esta información, el bloque de generación de acarreo es capaz de dar como salidas todos los acarreos de cada bloque sumador. Y asimismo da la información de AND (PG) y OR (GG) por si queremos conectar este circuito a un circuito subsiguiente en cascada.

La información de acarreo se genera con esta lógica.

$$C_{i+1} = g_i + (p_i \cdot C_i) \qquad 3.1$$

Es decir, a partir de las salidas p y g del bit i, que se obtienen rápidamente desde sus entradas A y B, el sistema ya puede generar el siguiente acarreo C del bit $i + 1$. Así pues, la propagación es rápida dentro del circuito y, por lo tanto, la salida correcta se genera con mayor rapidez.

Como ejemplo de circuito integrado comercial con generación rápida de acarreo tenemos el 74LS83A, que es una variante del 7483, compatible con sus pines, pero que internamente utiliza un bloque de *carry look ahead* como el que hemos explicado.

1.2.2. Restadores en binario

Si razonamos de manera análoga a la de los semisumadores, podemos proponer un circuito semirrestador[6] para palabras de un bit. Si denominamos A al minuendo y B al sustraendo, en la tabla 4 vemos la tabla de verdad que queremos implementar. Cuando la resta es negativa, necesitamos que se active el acarreo C. Observemos que este número negativo no se podría representar con un número binario puro "sin signo". Por lo tanto, en un caso así la salida

(6)En inglés, *half-subtractor*.

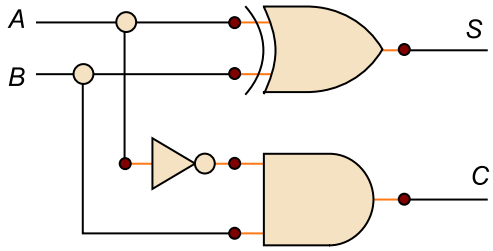
C representaría un desbordamiento o un error de representación de la codificación binaria. En sistemas que sí representen números negativos, usaríamos *C* como acarreo.

Tabla 4. Tabla de verdad del semirrestador, con entradas *A* y *B*, diferencia *D* y acarreo *C*

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

Este circuito es muy parecido al semisumador. De hecho, la salida *D* es la misma y solo tenemos que añadir la puerta inversora para generar *C*, según vemos en la figura 7.

Figura 7. Circuito semirrestador



En este caso también nos interesará disponer de un bloque con un posible acarreo a la entrada, de modo que tengamos un circuito restador completo[7], de manera análoga al sumador completo.

[7] En inglés, *full-subtractor*.

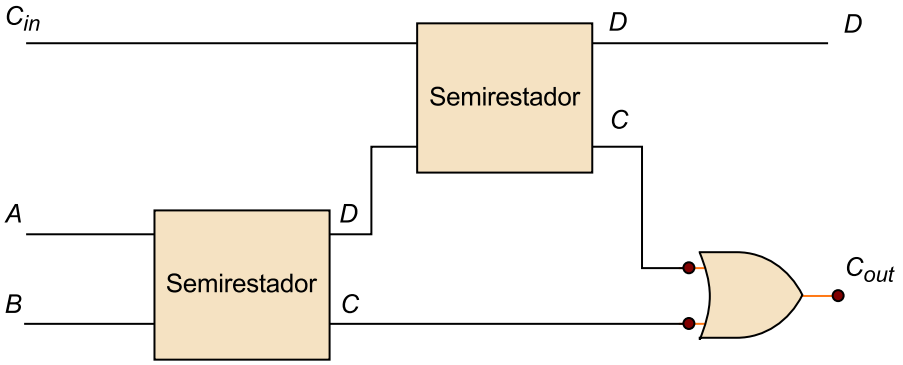
Tabla 5. Tabla de verdad del restador, con entradas *A* y *B*, acarreo de entrada C_{in}, diferencia *D* y acarreo de salida C_{out}

A	*B*	C_{in}	C_{out}	*S*
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Este restador tendría la tabla de verdad mostrada en la tabla 5, donde vemos que C_{in} también actúa como sustraendo. Es decir, por ejemplo cuando B y C_{in} son 1 y A es 0, la salida será 0 – 1 – 1= –2 y, por lo tanto, D será 0 y necesitamos el acarreo C_{out} a 1. Cuando a la salida queremos representar un –1, es preciso poner D a 1 (representa la magnitud) y arrastrar el signo con C_{out} también a 1.

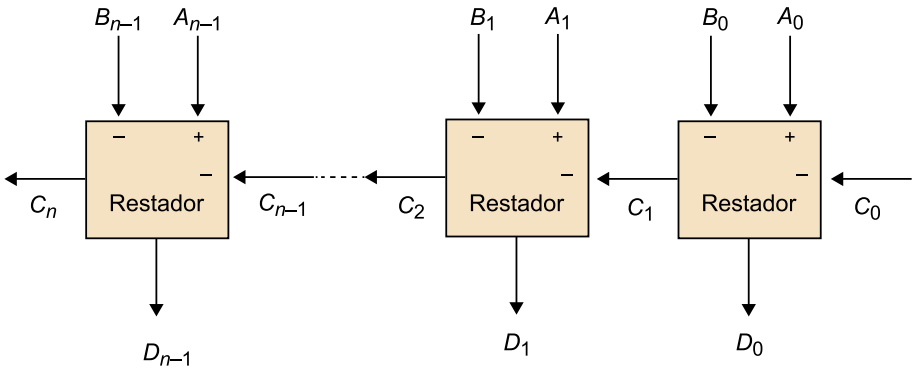
Al igual que el sumador completo, un restador completo que cumpla esta tabla de verdad se puede implementar con la configuración de la figura 8.

Figura 8. Circuito restador basado en dos semirrestadores y una puerta OR



Asimismo, podemos encadenarlos para efectuar diferencias de palabras A y B de n bits, tal y como mostramos en la figura 9. Se trata de un restador en paralelo: todos los bits de las palabras entran al mismo tiempo y se mantienen hasta que los acarreos se han propagado y han pasado por todos los módulos hasta la salida.

Figura 9. Circuito restador de palabras de n bits basado en restadores de 1 bit



Hasta este momento, hemos visto los circuitos adecuados para sumar y restar números binarios, es decir, sin representación de signo. También resulta interesante tratar el caso de sumadores y restadores en los que intervengan números negativos, representados según los sistemas que conocemos, como puede ser el complemento a 2 (Ca2).

Ved también

Podéis ver la representación de signo que se presenta en el subapartado 1.1 de este módulo didáctico.

1.3. Sumadores y restadores en complemento a 2

En este subapartado, estudiaremos las posibles modificaciones necesarias que hay que hacer en un circuito sumador o restador para que acepte no solo números positivos, sino también números negativos y, en concreto, en representación por complemento a 2. El motivo principal es que, de las distintas maneras de representar el signo, con complemento a 2 es más sencillo hacer operaciones de suma o resta. En la práctica, S&M y Ca1 no se utilizan por la doble representación del 0 (+0 y –0).

Cuando queremos hacer una suma con esta representación, es posible sumar todos los bits de la palabra y no se tiene en cuenta el acarreo de salida en el bit más significativo. Podemos ver algunos ejemplos de esto:

```
   -2      1110
 + -6      1010
------------------
   -8     11000

   +6      0110
 + -3      1101
------------------
   +3     10011

   +4      0100
 + -7      1001
------------------
   -3      1101
```

El bit destacado, que representa el último acarreo (en caso de que lo haya), no es preciso que sea considerado y el resultado es efectivamente correcto. Así pues, con complemento a 2 podremos utilizar los mismos circuitos sumadores que ya conocemos, teniendo en cuenta que no necesitamos el último acarreo.

Ved también

Podéis ver los circuitos sumadores en el subapartado 1.2 de este módulo didáctico.

También hay que tener en cuenta el escenario en el que se produzca desbordamiento, es decir, en el que el sistema de representación no sea capaz de representar el número resultante. En la representación en complemento a 2 para 4 bits, el desbordamiento con dos números positivos se produciría cuando la suma sea un número mayor que +7, y para el caso de dos números negativos, cuando sea un número inferior a –8. Veamos algunos ejemplos:

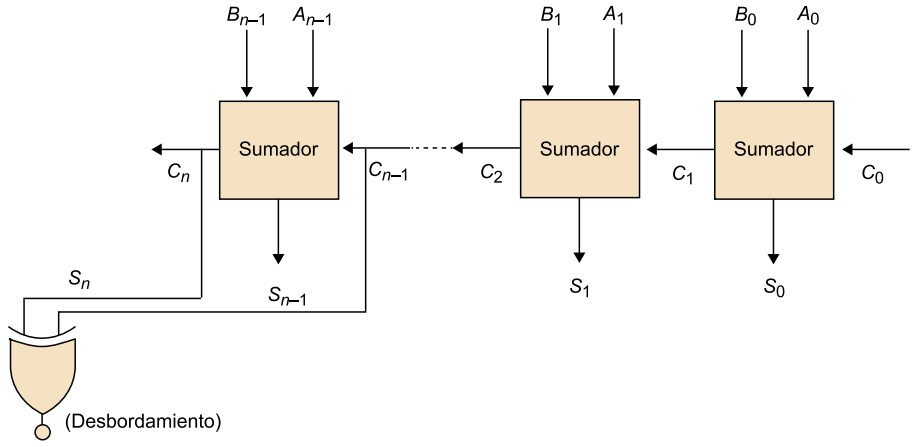
```
   -3      1101
 + -6      1010
------------------
   -9     10111 = +7

   +5      0101
 + +6      0110
------------------
  +11      1011 = -5
```

Afortunadamente, hay una regla simple para detectar estas situaciones: una suma experimenta desbordamiento si los signos de los sumandos son el mismo, y el signo de la suma resultante es el opuesto. Dado que el bit más signi-

ficativo es el que indica el signo, esto se puede detectar fácilmente con una comparación. En los dos ejemplos de desbordamiento anteriores, el bit más significativo cambia. Los circuitos integrados pueden, por lo tanto, añadir una lógica sencilla para producir una señal de aviso de desbordamiento.

La regla de desbordamiento también se puede exponer en términos de los acarreo generados durante la operación de adición: una suma experimenta desbordamiento si los acarreos de entrada y salida del bit más significativo son diferentes. Lo podemos comprobar en los dos ejemplos anteriores. Esta regla es fácil de implementar en la práctica con una puerta XOR, tal y como vemos en la figura 10.

Figura 10. Sumador en complemento a 2 con detección de desbordamiento



En cuanto a la resta con complemento a 2, en la práctica los circuitos se implementan de modo que no hacen una resta directamente, sino que en primer lugar cambian el signo del sustraendo (tomando su complemento a 2) y entonces lo suman al minuendo utilizando las reglas de suma que acabamos de ver.

Se puede negar el sustraendo y sumar el minuendo con solo una operación de adición, tal y como explicamos a continuación:

1) Se hace la negación de bits del sustraendo, bit a bit.

2) El resultado se suma al minuendo, incluyendo un acarreo inicial a 1 (C_{in}).

Veamos algunos ejemplos de esto: en la segunda columna hemos hecho la negación bit a bit del sustraendo, y entonces lo sumamos teniendo en cuenta un acarreo de entrada C_{in} a 1. Observad que continuamos considerando la regla de que el acarreo en el bit más significativo no se tiene en cuenta.

```
                                  1(Cin)
   +4          0100      0100
 - +3      -   0011    + 1100
-------------------------------
   +1                  10001

                                  1(Cin)
   +3          0011      0011
 - -4      -   1100    + 0011
-------------------------------
   +7                    0111
```

Tenemos, pues, una manera sencilla de calcular la resta: en primer lugar, negar todos los bits del sustraendo (con puertas NOT, por ejemplo) y utilizar a continuación un circuito sumador que tenga el acarreo de entrada siempre a 1. Seguiríamos sin utilizar el acarreo de salida.

Con respecto al desbordamiento de la resta, se puede detectar del mismo modo que en el caso de la suma, examinando los signos de las dos palabras a la entrada y de la salida. En este caso, se examinará en el paso de hacer la suma (es decir, una vez complementado el sustraendo).

Como resumen, podemos concluir que hemos visto que para hacer sumas y restas en complemento a 2 podemos utilizar los mismos circuitos sumadores que hemos estudiado en binario, teniendo en cuenta los factores siguientes:

- Hace falta una detección de desbordamiento sencilla y basada en la comparación de los bits más significativos de las entradas y la salida. O de manera equivalente, comparando los últimos acarreos.

- Podemos hacer una resta tan solo negando en primer lugar el sustraendo, y utilizando un sumador con el acarreo de entrada a 1.

1.4. Comparadores de magnitud

La operación lógica de comparación es, al igual que la suma y la resta, una operación muy importante en todo cálculo. Puede ser la base de ramificaciones en un programa lógico, dada una cierta condición que se cumple o no. Cualquier lógica de computación requiere hacer comparaciones, en el sentido de saber si dos palabras binarias son iguales en magnitud, o una es mayor o menor que la otra. Si las palabras tienen signo, se trata aparte, utilizando comparadores de magnitud y lógica adicional.

Un comparador de magnitud de dos palabras de n bits (A y B) es un circuito que determina cuál de estas palabras es mayor, cuál es menor y cuándo son iguales. Tiene que producir, por lo tanto, tres salidas: $A > B$, $A < B$, $A = B$.

La base para la síntesis de comparadores es el circuito coincidencia, que solo está activo cuando los dos bits son iguales ($A = B = 0$, $A = B = 1$). Por ejemplo, para palabras de un bit, podríamos plantear la fórmula siguiente, que da como resultado 1 cuando $A = B$, y 0 cuando A es diferente de B:

$$E = \overline{\overline{A} \cdot B + A \cdot \overline{B}} \qquad 3.2$$

Por otro lado, la condición $A > B$ se puede detectar a partir de la función lógica siguiente, ya que si $A = 1$ y $B = 0$ (y por lo tanto, $A > B$), da como resultado 1, y para otros casos, da como resultado 0:

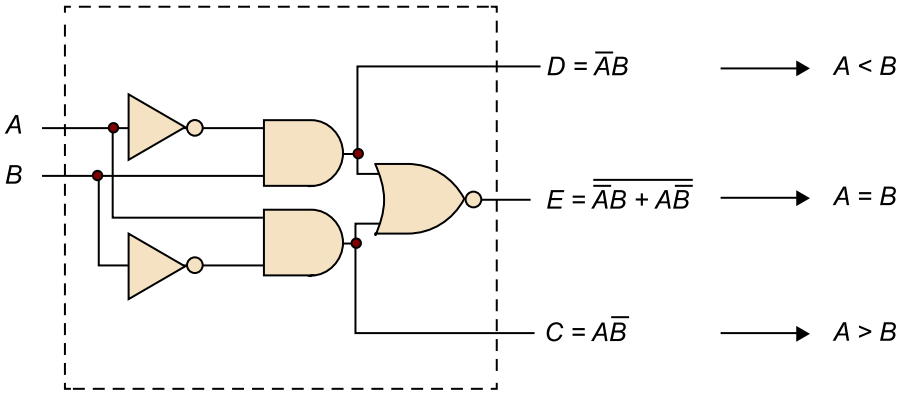
$$C = A \cdot \overline{B} \qquad 3.3$$

Del mismo modo, la condición $A < B$ se obtiene a partir de:

$$D = \overline{A} \cdot B \qquad 3.4$$

Estas tres salidas se pueden implementar con las puertas lógicas adecuadas que nos permitan hacer las negaciones, funciones AND y OR, de manera simple, como representamos en la figura 11.

Figura 11. Circuito comparador de dos palabras de un bit



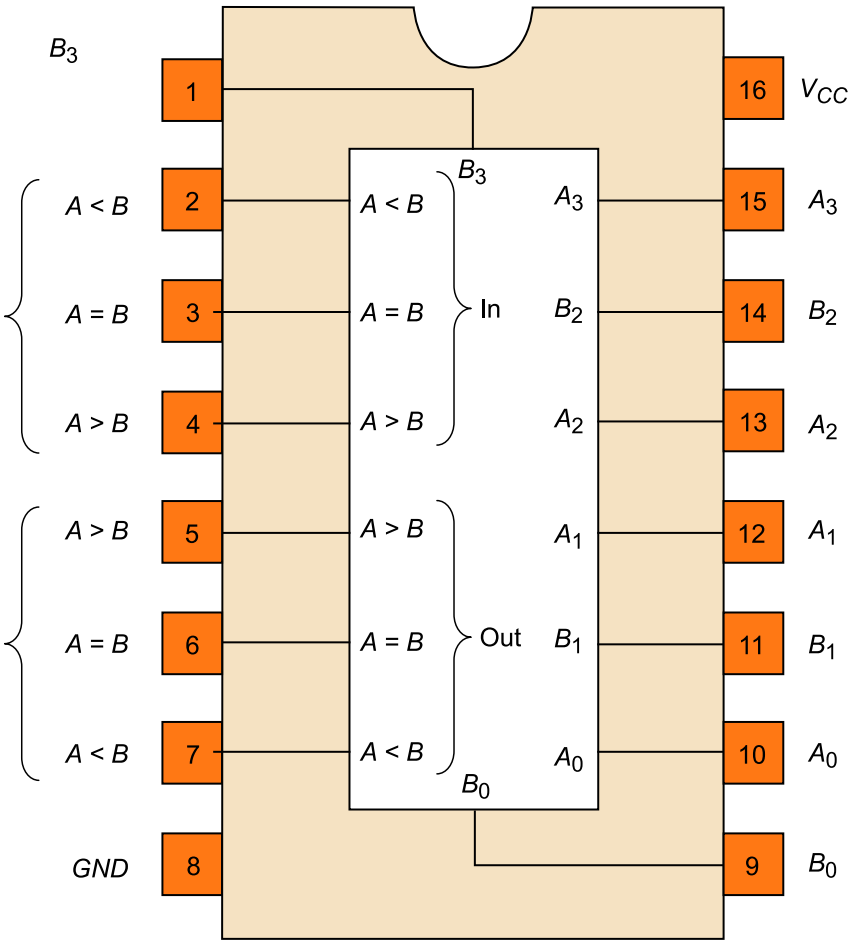
Cuando tratamos con palabras de más de un bit, el procedimiento para compararlas consiste primero en comparar el bit más significativo de cada una. Si estos son iguales, entonces se compara el siguiente bit más significativo, y así de manera sucesiva hasta encontrar una desigualdad que indica cuál de las dos palabras es mayor o menor. Si se comparan todos los bits de las dos palabras y no hay desigualdad entre estos, entonces (evidentemente) son iguales.

Un circuito integrado comercial muy utilizado es el **comparador de magnitud de 4 bits 74HC85**, que vemos representado en lo que respecta a sus pines en la figura 12. Como entradas, podemos ver las dos palabras A y B de 4 bits (en los pines 1 y 9 a 15), y las tres salidas comparándolas están en los pines 5 a 7, donde en condiciones normales solo una de las tres se encontrará en estado alto para indicar cuál es la condición detectada.

El circuito integrado 74HC85 es un comparador CMOS de alta velocidad que tiene un retraso de propagación de en torno a 27 ns y un consumo de potencia reducido, como corresponde a esta tecnología. En la figura 12 también observamos que tiene unas entradas[8] adicionales en los pines 2 a 4, que se denominan *entradas en cascada*. Sirven para aumentar la capacidad del comparador, es decir, para conectar otro comparador en cascada y operar con palabras de 8 bits. El primer comparador compara los 4 bits menos significativos (parte baja de las palabras) y sus salidas se conectan a las entradas en cascada del comparador superior, que compara los 4 bits más significativos (parte alta de las palabras). De este modo, si los datos de la parte alta son iguales, el comparador de la parte baja informa de si esta parte baja es igual, superior o inferior.

[8] En inglés, *inputs*.

Figura 12. Circuito integrado 74HC85, comparador de 4 bits



En la figura 13 observamos la tabla de verdad, que dejamos en inglés (así es como la encontraremos mayoritariamente en las especificaciones de los fabricantes, y tenemos que saber interpretarla). Efectivamente, comprobamos que las entradas en cascada (provenientes de posibles comparadores de la parte baja de la misma palabra) solo se tienen en cuenta cuando todos los bits de la parte alta son iguales. Esto se corresponde con las últimas posiciones de la tabla, donde sí importa qué valores tienen las entradas en cascada.

El símbolo *L* indica *low*, es decir, un estado 0, mientras que *H* indica 1. La cruz, *X*, indica que su estado es indiferente[9] para generar una salida[10] en aquella situación. Podemos comprobar que se trata de hacer la **comparativa de la magnitud** de ambas palabras, empezando por el bit más significativo.

[9] En inglés, *don't care*.

[10] En inglés, *output*.

Figura 13. Tabla de verdad según fabricante del circuito integrado 74HC85

FUNCTION TABLE

COMPARING INPUTS				CASCADING INPUTS			OUTPUTS		
A_3, B_3	A_2, B_2	A_1, B_1	A_0, B_0	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$	$Q_{A>B}$	$Q_{A<B}$	$Q_{A=B}$
$A_3>B_3$	X	X	X	X	X	X	H	L	L
$A_3<B_3$	X	X	X	X	X	X	L	H	L
$A_3=B_3$	$A_2>B_2$	X	X	X	X	X	H	L	L
$A_3=B_3$	$A_2<B_2$	X	X	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1>B_1$	X	X	X	X	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1<B_1$	X	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0>B_0$	X	X	X	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0<B_0$	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	H	L	L	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	L	H	L	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	L	L	H	L	L	H
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	X	X	H	L	L	H
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	H	H	L	L	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	L	L	L	H	H	L

Notes

1. H = HIGH voltage level
   L = LOW voltage level
   X = don't care

Con este circuito, que proporciona además funcionalidad en cascada, podríamos conectar tantos comparadores como queramos, y comparar palabras con un gran número de bits (múltiplos de 4). En la figura 14 observamos cómo haríamos la conexión entre dos circuitos para comparar dos palabras de 8 bits; y vemos que la salida del comparador de la parte baja de las palabras se conecta a la entrada en cascada del siguiente circuito 74HC85. Hay que destacar que no dejamos en el aire las entradas del primero, sino que forzamos a que indiquen $A = B$ para asegurar que todo el conjunto indica que las palabras son iguales, si lo son.

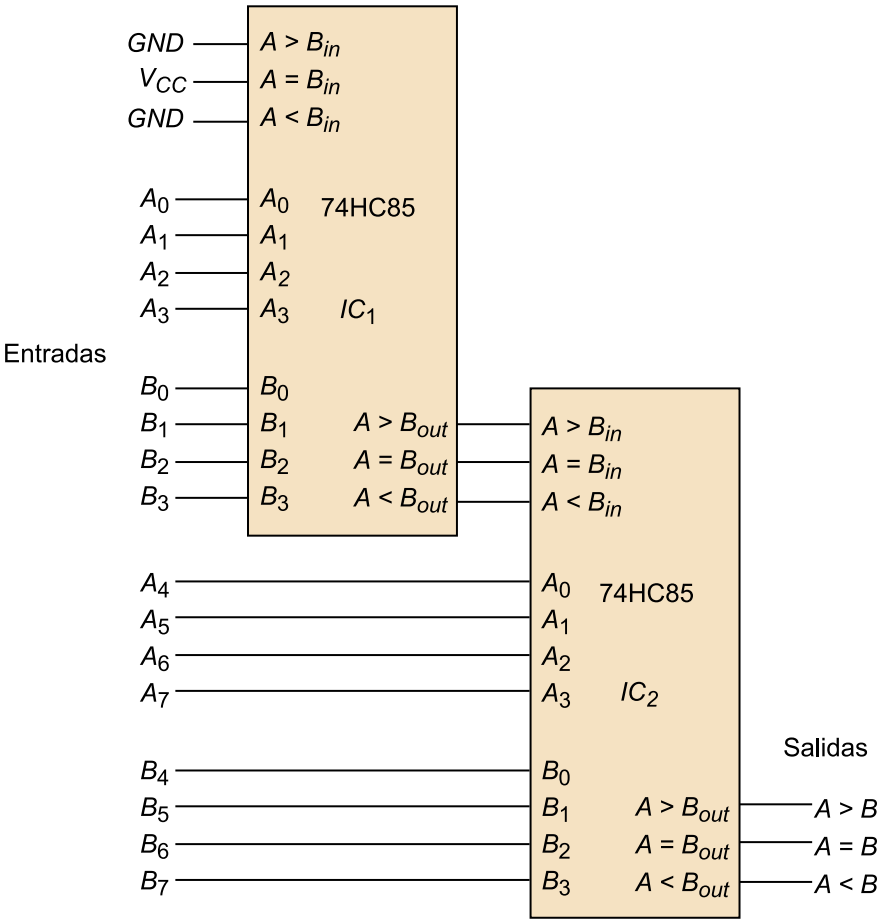
Destacamos una vez más que se trata de un **comparador de magnitud**. Si quisiéramos trabajar con números binarios negativos (que figuran en cualquiera de las representaciones vistas), habría que añadir una cierta lógica adicional, específica de nuestra aplicación, para sintetizar la operación que deseáramos a partir de comparar magnitudes con el circuito 74HC85, por ejemplo.

Ved también

Podéis ver las representaciones numéricas explicadas en el subapartado 1.1 de este módulo didáctico.

El estudio de este circuito integrado nos ha proporcionado un buen ejemplo para entender no solo cómo se diseñan las puertas lógicas para implementar una función determinada, sino también cómo utilizar circuitos comerciales de manera práctica, así como las capacidades que nos ofrecen para construir subsistemas cada vez más complejos.

Figura 14. Dos comparadores 74HC85 conectados en cascada



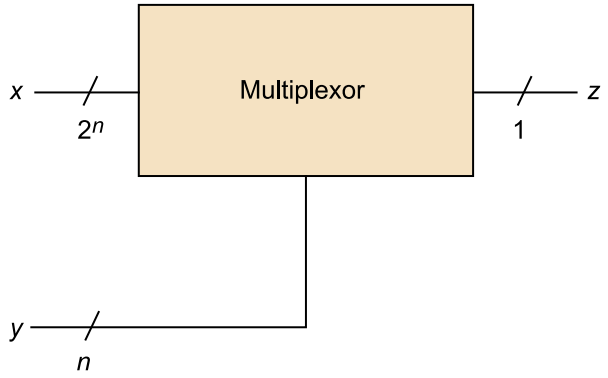
1.5. Multiplexores y demultiplexores

En este punto también nos interesa destacar la funcionalidad de los denominados *circuitos multiplexores y demultiplexores*, vistos en cursos de electrónica digital, y de los que haremos un breve recordatorio, puesto que se trata de circuitos combinacionales con todo tipo de aplicaciones.

Podemos definir un multiplexor como un sistema con dos grupos de entradas (que denominaremos x e y) y una salida z. El valor de la salida corresponde al de una de las entradas x. La entrada x que corresponderá al valor de salida se selecciona en función del valor de la dirección proporcionada por la señal y.

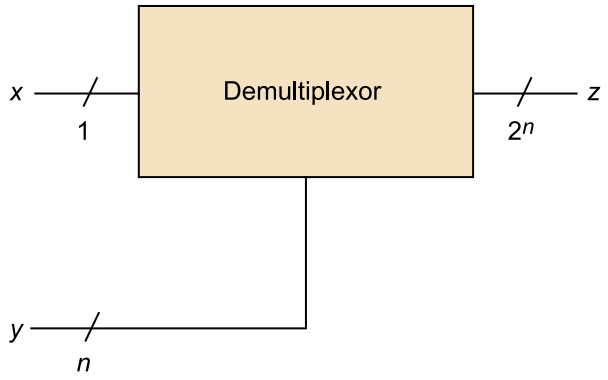
En la figura 15, podemos observar su diagrama de bloques básico. Observad que nos hacen falta n bits (señal y) para poder direccionar cuál de las 2^n entradas (señal x) es la que aparece a la salida (señal z).

Figura 15. Multiplexor de 2^n entradas de 1 bit



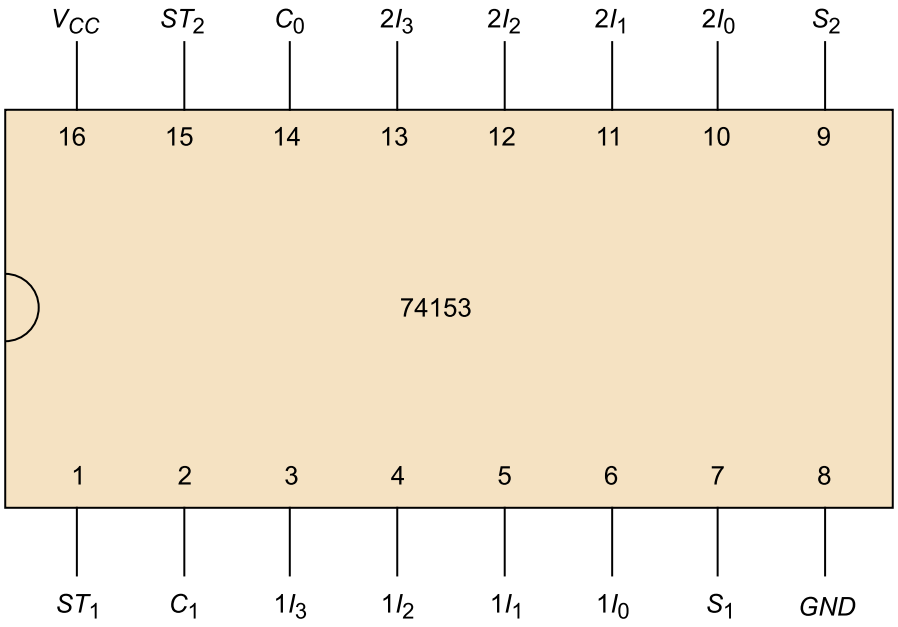
Alternativamente, un demultiplexor haría la operación inversa, es decir, se trata de un sistema con $n + 1$ entradas y 2^n salidas. En la figura 16, vemos su diagrama de bloques. El valor de una de las entradas (señal *x*) se copia en una de las líneas de salida (señal *z*) en función del valor de la dirección proporcionada por la señal *y*.

Figura 16. Demultiplexor de 2^n salidas de 1 bit



Un ejemplo de multiplexor comercial en forma de circuito integrado es el 74153. Se trata de un multiplexor dual, es decir, en un mismo circuito integrado encontramos dos multiplexores, en este caso de cuatro entradas cada uno. En la figura 17 vemos sus pines: V_{CC} y *GND* corresponden a la alimentación, mientras que las dos entradas son las palabras de 4 bits dadas por $1I_3$, $1I_2$, $1I_1$, $1I_0$ (multiplexor 1) y $2I_3$, $2I_2$, $2I_1$, $2I_0$ (multiplexor 2). La señal de direccionamiento de qué entrada encontramos a la salida viene dada por C_1 y C_0, y es común a ambos multiplexores. Las señales de salida de cada multiplexor son S_1 y S_2, respectivamente. También disponemos de ST_1 y ST_2, que nos permiten habilitar el uso de los dos multiplexores (podemos usar cualquiera de los dos o ambos a la vez).

Figura 17. Multiplexor 74153



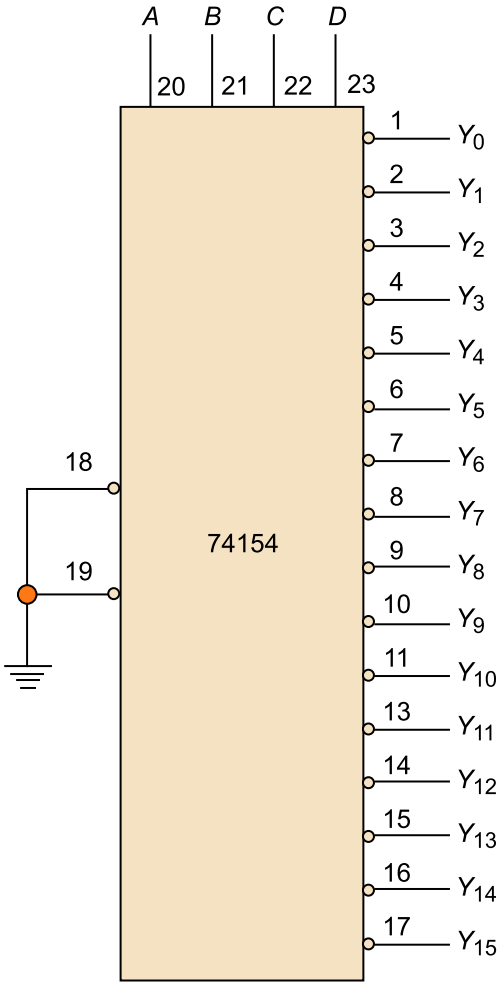
1.6. Codificadores y decodificadores

Otro conjunto de subsistemas digitales combinacionales viene dado por los denominados *codificadores* y *decodificadores*. Un **codificador** es un sistema combinacional que presenta n entradas y m salidas, y en el que el número de entradas es mayor que el número de salidas ($m \leq n$). El concepto, por lo tanto, es genérico; según la aplicación, la funcionalidad exigida al codificador será diferente. Podemos pensar, por ejemplo, en un circuito que pase de un número decimal representado con 10 bits a un número binario representado con 4 bits, u otros tipos de codificaciones.

Por otro lado, un decodificador opera en sentido inverso, es decir, también presenta n entradas y m salidas, pero en este caso el número de entradas es menor que el de salidas ($n \leq m$). Un buen ejemplo de circuito integrado comercial de este tipo es el 74154, un decodificador de 4 entradas a 16 salidas. Como entrada recibe un número binario de 4 bits, que por lo tanto puede representar valores entre 0 y 15. Y a la salida hay 16 líneas, solo una de las cuales se activará, según el valor binario de la entrada. Por defecto, todas se encuentran en estado alto (1) y cuando se activan ofrecen un estado bajo (0).

En la figura 18 observamos un diagrama del circuito, en el que vemos que hay cuatro pines de entrada (A, B, C, D) y dieciséis pines de salida (Y) marcados con una negación por el hecho de que se activan en estado bajo. El fabricante también incorpora dos pines, el 18 y el 19, que deben estar a cero para forzar que el circuito actúe como decodificador.

Figura 18. Decodificador 74154 según fabricante. La numeración indica el número del pin del circuito integrado



1.7. Generadores/detectores de paridad

Otro tipo de subsistema digital digno de mención por su gran utilidad y uso en sistemas de comunicación es el **generador de paridad** y su correspondiente **detector de paridad**.

En cualquier transmisión de información digital por un canal es importante verificar que la información recibida es igual a la emitida, y una técnica muy común consiste en añadir algún bit de redundancia a la palabra que se pretende transmitir. Este bit de redundancia puede estar relacionado con la paridad del mensaje, y se genera en el transmisor con un circuito **generador de paridad**.

En recepción, lo que podemos hacer es fijarnos en la palabra recibida, volver a generar el bit de paridad y comprobar si es el mismo que el que hemos recibido. Esta es la tarea del **detector de paridad**. Si comprueba que la palabra recibida indica la misma paridad que la comprobada, entonces el mensaje se da por bueno.

Como podemos deducir rápidamente, este sistema o principio de funcionamiento se basa en el hecho de que solo puede haber un error de bit en toda la palabra, hipótesis que para sistemas de comunicaciones con un mínimo de calidad es muy razonable. La probabilidad de error de bit es lo bastante pequeña como para asumir que el hecho de que haya 2 bits erróneos tan contiguos es negligible. En efecto, hay que destacar que si se produjeran dos errores, o un número par de errores, la comprobación de paridad no nos serviría para detectarlos. En todo caso, este sistema resulta bastante sencillo y a la vez eficaz en muchas situaciones.

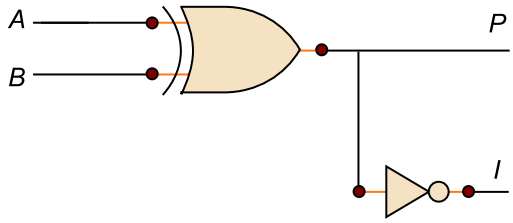
En la tabla 6 encontramos la tabla de verdad de un generador de paridad de 2 bits de entrada, *A* y *B*, a cuya salida se da una señal que indica paridad par (*P*) y una señal que indica paridad impar (*I*). Cuando hablamos de *paridad* nos referimos a la señal transmitida total, es decir, al conjunto de A, *B* y *P* bits, o bien al conjunto de A, *B* e *I* bits. En el primer caso, nos aseguramos de que el número de 1 en el mensaje enviado es par y, en el segundo, de que es impar.

Tabla 6. Tabla de verdad del generador de paridad

A B	*P*	*I*
0 0	0	1
0 1	1	0
1 0	1	0
1 1	0	1

Observemos que la señal *P* es, de hecho, una OR exclusiva (XOR) respecto de *A* y *B*, mientras que la señal *I* es su negación. Esta representación con puertas lógicas es la que encontramos en la figura 19, en la que tenemos un circuito que puede generar tanto paridad par como impar. En la práctica, elegiríamos solo una de las opciones en nuestra transmisión.

Figura 19. Generador de paridad por palabra de entrada de dos bits



Ahora imaginemos que elegimos un mensaje enviado con paridad par, es decir, *A, B* y *P*. Son tres bits que se emiten por el canal, y el receptor deberá comprobar que realmente esta paridad par se ha mantenido a lo largo del camino recorrido. La tabla de verdad del comprobador, cuando este tiene como entra-

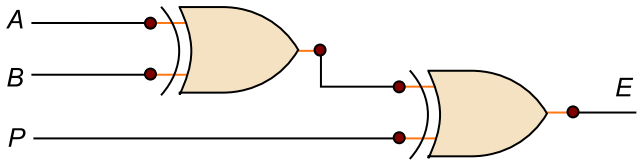
das *A, B* y *P*, viene dada por la tabla 7. Queremos un circuito que nos diga a su salida (*E*, de *error*) si hay error en el hecho de que el mensaje de entrada tenga paridad par.

Tabla 7. Tabla de verdad del comprobador de paridad par

A B P	*E*
0 0 0	0
0 0 1	1
0 1 0	1
0 1 1	0
1 0 0	1
1 0 1	0
1 1 0	0
1 1 1	1

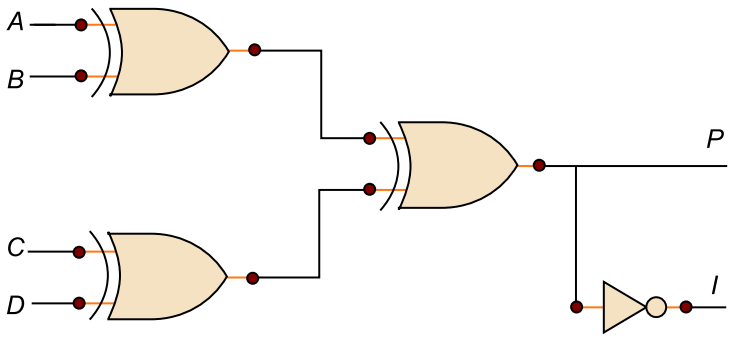
Comprobamos que esta función se puede implementar otra vez con puertas XOR, tal y como se propone en la figura 20.

Figura 20. Comprobador/detector de paridad par por palabra de entrada de dos bits más uno de paridad



La salida *E* nos indicará si ha habido un error en la transmisión, es decir, si alguno de los bits *A, B* o P no se ha recibido correctamente.

Figura 21. Generador de paridad por palabra de entrada de cuatro bits

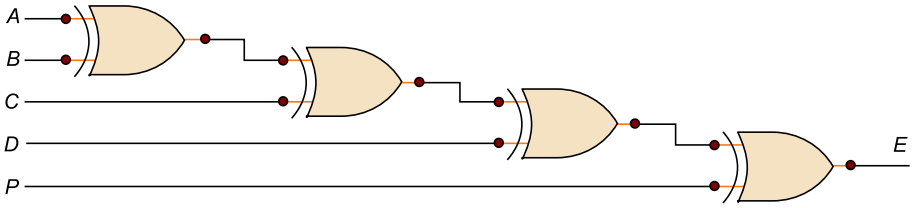


Hemos visto, pues, un generador de paridad de dos bits y su correspondiente detector de paridad. Para ampliar estos circuitos a palabras de más bits, solo hay que añadir más puertas lógicas XOR, tal y como vemos en la figura 21

para un generador de paridad de cuatro bits. En la figura observamos que se han añadido dos puertas XOR, las cuales reciben los cuatro bits de la entrada, y sus salidas pasan por una tercera puerta XOR.

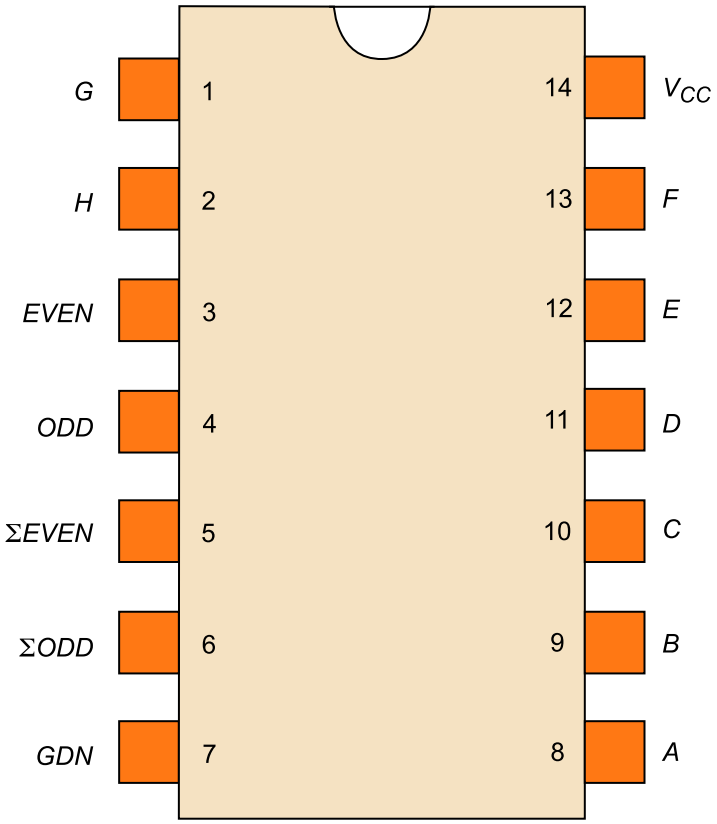
Podríamos ir concatenando estas topologías para trabajar con palabras de mayor longitud. En recepción, cuando tenemos que utilizar un detector de paridad, también podemos ampliar la configuración de la figura 20 tal y como nos muestra la figura 22. Vemos cómo vamos concatenando puertas XOR hasta llegar a obtener el bit *E* que indica error de paridad.

Figura 22. Detector/comprobador de paridad par para palabras de entrada de cuatro bits más uno de paridad



Nótese que el detector de paridad impar será un circuito parecido con un inversor a la salida *E*.

Figura 23. Circuito integrado 74180



En la práctica, podemos encontrar circuitos integrados que nos ofrecen las dos funcionalidades, tanto generación como detección de paridad, como es el caso del circuito integrado 74180. Trabaja con nueve bits de entrada, y uno de

estos puede ser el bit de paridad cuando actúa como detector. Resulta interesante conocer su topología de puertas interna para entender cómo podemos combinar ambas funciones al mismo tiempo y de manera óptima.

En primer lugar, en la figura 23 vemos el diagrama de pines según el fabricante. Está compuesto por catorce pines, ocho de los cuales se dedican a la palabra de entrada (*A a H*); hay dos para la alimentación (V_{CC} y *GND*); dos más de entrada (*EVEN* y *ODD*) para seleccionar qué tipo de paridad queremos; y dos de salida (Σ*EVEN* y Σ*ODD*). Esta nomenclatura del fabricante se utiliza para designar par[11] e impar[12]. El símbolo Σ significa 'sumatorio'. Para entender su funcionamiento, planteamos su tabla de verdad (tabla 8).

[11] Del inglés, *even*.

[12] Del inglés, *odd*.

Tabla 8. Tabla de verdad del circuito integrado 74180

Entradas			**Salidas**	
Suma de 1 a las entradas *A-H*	***EVEN***	***ODD***	**Σ*EVEN***	**Σ*ODD***
Par	1	0	1	0
Impar	1	0	0	1
Par	0	1	0	1
Impar	0	1	1	0
Indiferente	1	1	0	0
Indiferente	0	0	1	1

En primer lugar, vemos que lo podemos utilizar como generador de paridad. Si queremos utilizar paridad par, será necesario que seleccionemos *EVEN* = 1 y *ODD* = 0, y así nos situamos en las dos primeras filas de la tabla 8. Hay dos casos posibles:

- Si la palabra *A-H* tiene paridad par (primera fila), será necesario añadir al mensaje como noveno bit la salida Σ*ODD* (que es 0 en este caso), de modo que el mensaje de salida *A-H* más Σ*ODD* mantenga la paridad par.

- Si la palabra *A-H* tiene paridad impar (segunda fila), también añadiremos al mensaje como noveno bit la salida Σ*ODD* (que es 1 en este caso). De este modo, la salida formada por *A-H* más Σ*ODD* también tenga paridad par.

Por otro lado, si lo queremos utilizar para asegurar paridad impar en el mensaje de nueve bits de salida, deberemos seleccionar *EVEN* = 0 y *ODD* = 1, y nos situaremos en la tercera y la cuarta fila de la tabla 8. En estos dos casos:

- Si la palabra *A-H* tiene paridad par (tercera fila), deberemos añadir al mensaje como noveno bit la salida Σ*ODD* (que es 1 en este caso), de modo que el mensaje de salida *A-H* y Σ*ODD* tenga paridad impar.

- Si la palabra *A-H* tiene paridad impar (cuarta fila), también añadiremos al mensaje como noveno bit la salida ∑*ODD* (que es 0 en este caso). De este modo, la salida *A-H* y ∑*ODD* mantiene la paridad impar.

En ambos casos, vemos que el transmisor podrá añadir el bit ∑*ODD* a la palabra *A-H*, para generar su mensaje de nueve bits con la paridad seleccionada por las entradas *EVEN* y *ODD*.

Observamos también que el circuito no está pensado para utilizar los bits *EVEN* y *ODD* con el mismo valor, y por lo tanto, en este caso nos encontramos en el estado indiferente de la tabla.

Por otro lado, podemos utilizar este circuito como detector o comprobador de paridad. Para hacerlo, utilizaremos como entrada para el noveno bit el correspondiente a la paridad que estemos utilizando. De esta manera, si se supone que el mensaje de llegada tiene paridad impar, deberemos inyectar el noveno bit a la entrada *ODD* (y poner *EVEN* como negación de *ODD*). A partir de aquí, pueden darse diferentes casos:

- No hay error en la transmisión. Por lo tanto, los nueve bits tienen efectivamente paridad impar y el noveno bit es un 0. Estaríamos en la segunda fila de la tabla 8 y podríamos comprobar que la salida del circuito ∑*ODD* es 1, valor que tomaríamos como confirmación de que todo es correcto.

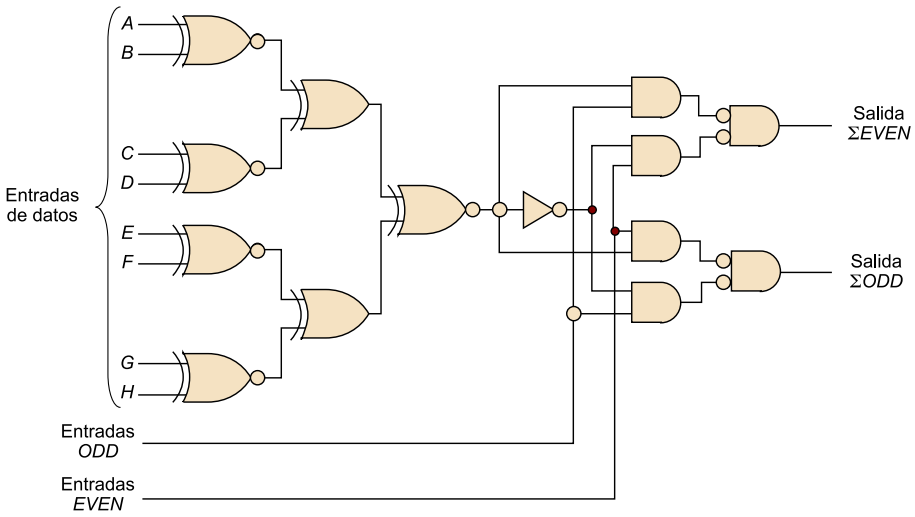
- No hay error en la transmisión. Por lo tanto, los nueve bits tienen paridad impar y el noveno bit es un 1. En este caso, nos situamos en la tercera fila de la tabla y podemos seguir utilizando la salida ∑*ODD* a 1 para ver que la transmisión es correcta.

- Hay error en la transmisión. Por lo tanto, los nueve bits tienen paridad par y el noveno bit es un 0. Estaríamos en la primera fila de la tabla y detectaríamos ∑*ODD* igual a 0 y, de este modo, error en la transmisión.

- Hay error en la transmisión y, por lo tanto, los nueve bits tienen paridad par y el noveno bit es un 1. Estaríamos en la cuarta fila de la tabla, en la que también detectaríamos ∑*ODD* como 0 y, por lo tanto, el error.

También es interesante que conozcamos el conjunto de puertas que posibilita la tabla de verdad que se representa en la tabla 8, lo que encontramos representado en la figura 24, según indica el fabricante, en la especificación del circuito 74180. Comprobamos que en la parte izquierda se utilizan puertas XOR para generar el bit de paridad. En la parte derecha, se utilizan puertas AND para combinarlo con las entradas *EVEN* y *ODD* y obtener la funcionalidad deseada.

Figura 24. Puertas lógicas del circuito integrado 74180 según fabricante



Con la explicación detallada de este circuito podemos concluir este apartado, en el que hemos descrito todo un conjunto de subsistemas digitales con funciones combinacionales. En el apartado siguiente, ya nos centraremos en los denominados *subsistemas secuenciales*.

2. Subsistemas típicos en circuitos digitales secuenciales

Hemos visto hasta ahora un conjunto de subsistemas típicos, basados en la síntesis de funciones combinacionales; es decir, a partir de unas entradas se genera una salida por cada instante de tiempo. Comentábamos en la introducción que también podemos pensar en **circuitos secuenciales** en los que se hace uso de algún elemento de memoria para calcular la salida, utilizando estados anteriores marcados por un reloj en modo síncrono, o también posiblemente algunos en modo asíncrono.

Como conocimiento previo, suponemos que habéis estudiado ya algunos de estos circuitos secuenciales, como pueden ser los biestables en otras asignaturas. En este apartado, ampliaremos estos conocimientos explicando circuitos como los registros de desplazamiento, los contadores y algunos conceptos relativos a las memorias.

2.1. Registros

Un registro es un grupo de biestables, cada uno de los cuales es capaz de almacenar un bit de información.

> Un registro de n bits consiste en un grupo de n biestables capaces de almacenar n bits de información binaria.

Además de contener biestables, un registro puede tener lógica combinacional que lleve a cabo unas ciertas tareas de procesamiento de la información. En su definición más amplia, un registro consiste en un grupo de biestables junto con puertas lógicas que afectan a la operación de estos. Los biestables guardan la información binaria, y las puertas determinan cómo se transfiere la información dentro del registro.

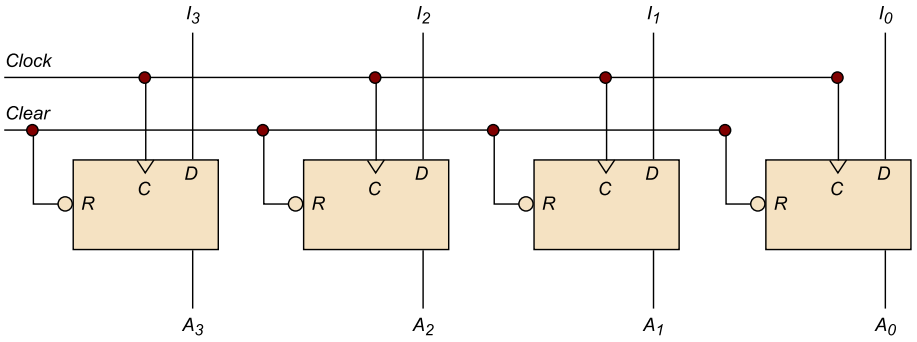
2.1.1. Registro simple de 4 bits

Hay diferentes tipos de registros disponibles de manera comercial. El más simple es aquel que consiste solo en biestables, sin ninguna puerta lógica. La figura 25 muestra un registro de este tipo, construido con cuatro biestables de tipo D, con el objetivo de almacenar datos de 4 bits. El reloj (común a todos los biestables) activa los biestables en cada flanco ascendente del pulso del reloj, y los datos binarios disponibles en las cuatro entradas (I_0, I_1, I_2, I_3) son transferidos a la salida (A_0, A_1, A_2, A_3). Recordemos que así funciona un biestable

de tipo *D* síncrono: el reloj marca cuándo se transfiere a la salida la entrada en el pin *D* (también dispone de pines *S* y *R* –*set* y *reset*– para forzar un 1 o un 0 a la salida).

Los valores de *A* son los que están disponibles en el circuito global para ser utilizados en otras etapas. También vemos una entrada *Clear*, que sirve para forzar todas las salidas a cero, que trabaja en estado bajo, lo que normalmente se hace en el momento de empezar a operar el circuito en modo síncrono de reloj. Por simplicidad, observamos que las entradas *S* de los biestables de la figura no aparecen, puesto que no se están utilizando.

Figura 25. Registro de 4 bits



2.1.2. Registro con control de carga en paralelo

Los sistemas digitales síncronos tienen un reloj maestro que proporciona un tren continuo de pulsos. Estos pulsos se aplican a todos los biestables y registros del sistema, y marcan el ritmo para todas las operaciones. A veces, sin embargo, nos interesa disponer de una señal de control separada para decidir qué operación se ejecutará en cada pulso de reloj en el registro.

La transferencia de información nueva a un registro se conoce como **carga**[13] o **actualización** del registro. Si todos los bits del registro son cargados de manera simultánea con un pulso de reloj común, decimos que la carga se hace **en paralelo**. Un flanco de reloj aplicado al registro de la figura 25 cargará las cuatro entradas en paralelo. En esta configuración, sin embargo, si queremos que los contenidos del registro permanezcan sin cambios, o bien las entradas *I* se deben mantener constantes o bien el reloj debe ser desinhibido del circuito. Comentaremos con detalle estas dos posibilidades:

(13)En inglés, *load*.

1) Mantener entradas *I* sin cambios: es posible, pero implica que el bus de datos *I* no se podría usar para otras tareas.

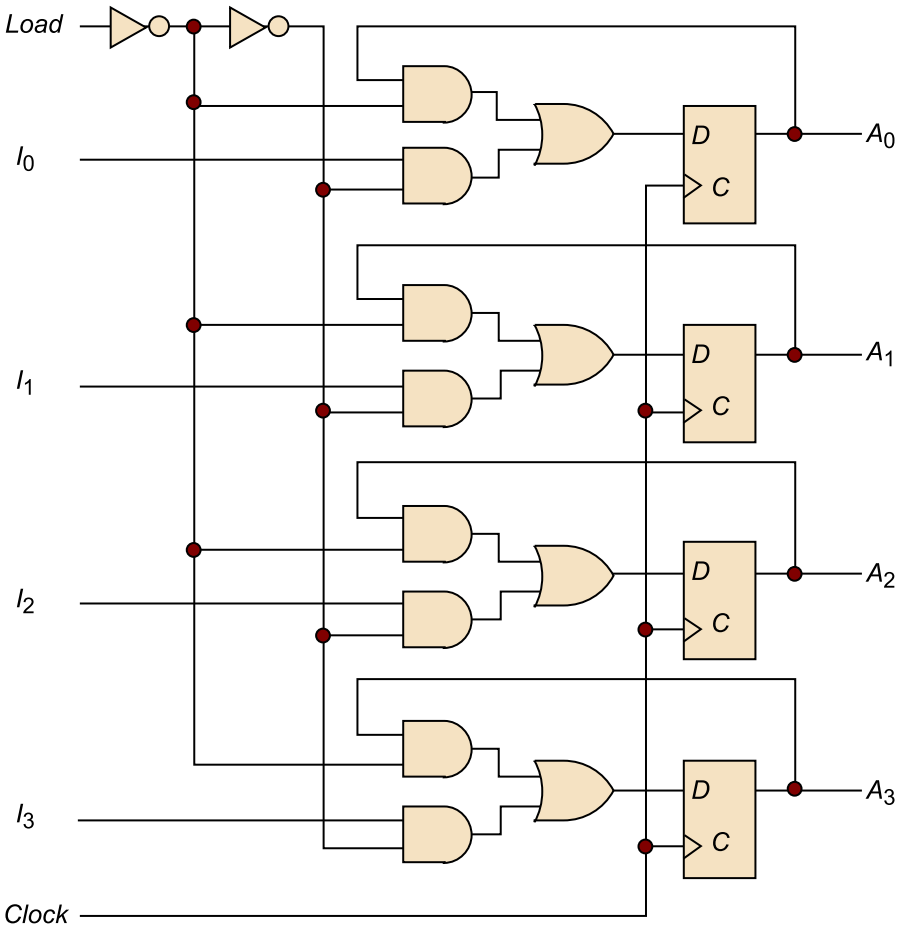
2) Introducir puertas lógicas entre la señal de reloj y los biestables: podríamos utilizar puertas lógicas para impedir que los biestables vean cambios en el reloj, pero esto no es muy recomendable puesto que estaríamos sintetizando

lógica a partir de pulsos de reloj (y no de datos). La inserción de puertas lógicas produce retrasos de propagación e introduce de esta manera retrasos variables que podrían causar que el sistema perdiera todo sincronismo.

Por estos motivos, es aconsejable controlar la operación del registro a partir de sus entradas *D* más que de sus entradas de reloj. Conseguiremos el mismo efecto si introducimos una cierta lógica a estas entradas *D*, como veremos a continuación.

En la figura 26 se muestra un registro de cuatro bits con control de carga. Por simplicidad, hemos eliminado la señal de *Clear*, aunque podría estar también disponible. Las puertas lógicas introducidas implementan de hecho un multiplexor de dos canales cuya salida se introduce como entradas *D* de los biestables. Estos dos canales llevan o bien los datos del bus *I,* o la salida del registro. De este modo, controlamos qué cargamos en el registro a cada pulso de reloj: su propia salida o datos nuevos que haya en el bus *I*.

Figura 26. Registro de 4 bits con control de carga en paralelo



Observemos que, cuando la entrada *Load* tiene un valor lógico alto, pasa por la puerta inversora y se convierte en bajo, y por lo tanto impone que la puerta AND subsiguiente (la superior de cada bit) anule el paso del valor de la sali-

da. Complementariamente, la puerta AND inferior de cada bit sí deja pasar el valor del bus *I*, que acaba cargándose en los biestables (en el siguiente flanco ascendente del reloj).

Cuando la entrada *Load* tiene valor bajo, sucede justo lo contrario: son las puertas AND inferiores de cada bit las que anulan el paso al bus *I*, y en cambio abren la entrada a los biestables de la misma salida de los registros.

Vemos, pues, que disponemos de una carga en paralelo controlada por un lado por el reloj del sistema, pero con suficiente flexibilidad para imponer que el registro guarde nuevos datos disponibles en el bus *I* en un cierto momento, o para aislar el bus *I* y permitir que el registro autorrefresque sus valores a partir de sus propios valores anteriores.

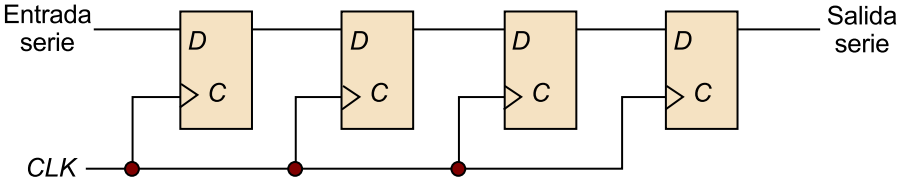
Otro bloque muy interesante, ahora que conocemos el funcionamiento de los registros, son los **registros de desplazamiento**, que analizaremos a continuación.

2.1.3. Registros de desplazamiento

Un registro capaz de desplazar la información binaria almacenada en cada celda hacia la celda vecina en un sentido elegido se denomina **registro de desplazamiento**. La configuración lógica de un registro de desplazamiento consiste en una cadena de biestables en cascada con la salida de un biestable conectada a la entrada del siguiente. Todos los biestables reciben pulsos de reloj comunes, los cuales activan el desplazamiento de los datos de una celda a la siguiente.

El registro de desplazamiento más simple es el mostrado en la figura 27, donde, por simplicidad, hemos eliminado la señal de *Clear*. La salida de un biestable se conecta al biestable de la derecha, y por lo tanto solo es posible que los datos fluyan en un sentido; se trata de un registro de desplazamiento unidireccional. Vemos que la entrada es serie (solo podemos introducir datos por el registro de la izquierda) y la salida también (solo tenemos acceso a la salida del registro de la derecha).

Figura 27. Registro de desplazamiento unidireccional



Si queremos un circuito en el que también dispongamos de entrada y salida en paralelo, y en el que además podamos elegir el sentido del desplazamiento, deberemos añadir nuevos bloques, como explicamos a continuación.

2.1.4. Registro de desplazamiento universal

Si las salidas de los biestables están accesibles, entonces la información introducida en serie se podría obtener en paralelo a partir de estas salidas. También podemos añadir al registro de desplazamiento la capacidad de poderle cargar datos en paralelo. Algunos registros de desplazamiento proporcionan los terminales de entrada y salida necesarios para hacer transferencias en paralelo, y además permiten seleccionar si el desplazamiento se hace en un sentido u otro.

Podríamos definir el registro de desplazamiento más universal como aquel que tiene las funcionalidades siguientes:

- Una señal de control (*clear* o *CLR*) para poner su estado a cero.
- Una entrada de reloj para sincronizar las operaciones.
- Una señal de control de desplazamiento a la derecha (*shift-right*) y a la izquierda (*shift-left*).
- Señales de entrada y salida serie en cada extremo.
- Una señal de control de carga en paralelo, para cargar todos los *n* biestables en el mismo momento con nuevos valores.
- *n* líneas de salida en paralelo.
- Un estado de control que permite dejar la información en el registro sin cambios, aunque haya un flanco de reloj.

Un circuito que posee estas características es el que se muestra en la figura 28. En la parte inferior, vemos los cuatro biestables que ya conocemos del registro de desplazamiento simple, y la diferencia se encuentra en el hecho de que a las entradas *D* de los biestables hacemos llegar una de cuatro posibilidades, a través de los multiplexores. Como hay cuatro posibles entradas en cada multiplexor, las operaciones son las siguientes:

- **0:** La salida del mismo biestable.
- **1:** La salida del biestable de su izquierda (o de la entrada serie de la izquierda en el del extremo). Esto es necesario cuando se hace un desplazamiento a la derecha[14].
- **2:** La salida del biestable de la derecha (o de la entrada serie de la derecha en el extremo). Esto es preciso cuando se hace un desplazamiento a la izquierda[15].

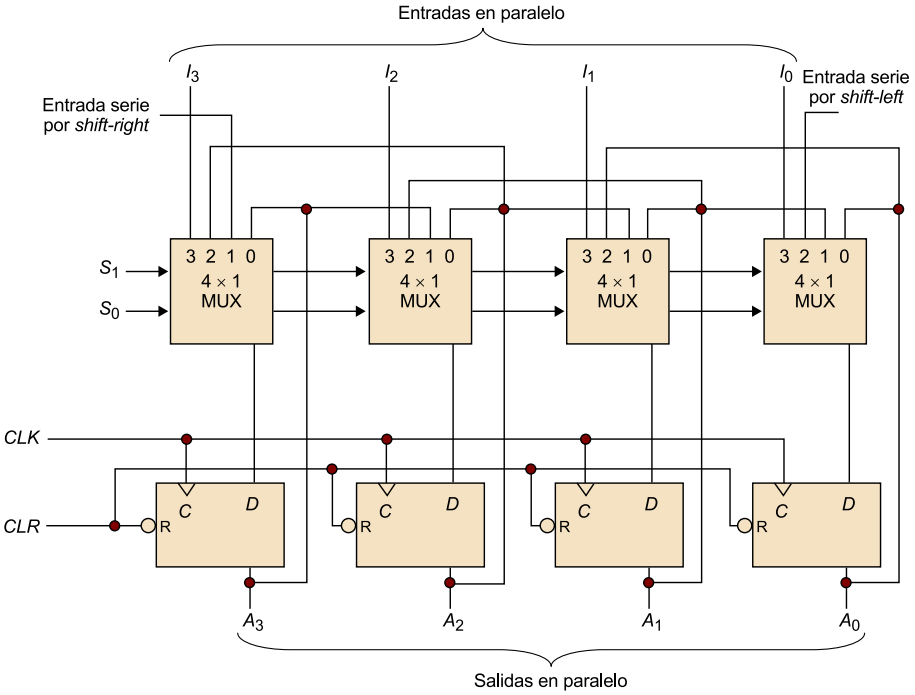
(14) En inglés, *shift-right*.

(15) En inglés, *shift-left*.

- **3:** Las entradas del bus *I* (en paralelo).

Con las señales de control de los multiplexores, S_0 y S_1, podemos seleccionar en todo momento qué valor queremos introducir en los biestables en el siguiente flanco de reloj. Puesto que tenemos cuatro entradas al multiplexor, nos basta con dos señales binarias para seleccionarlas.

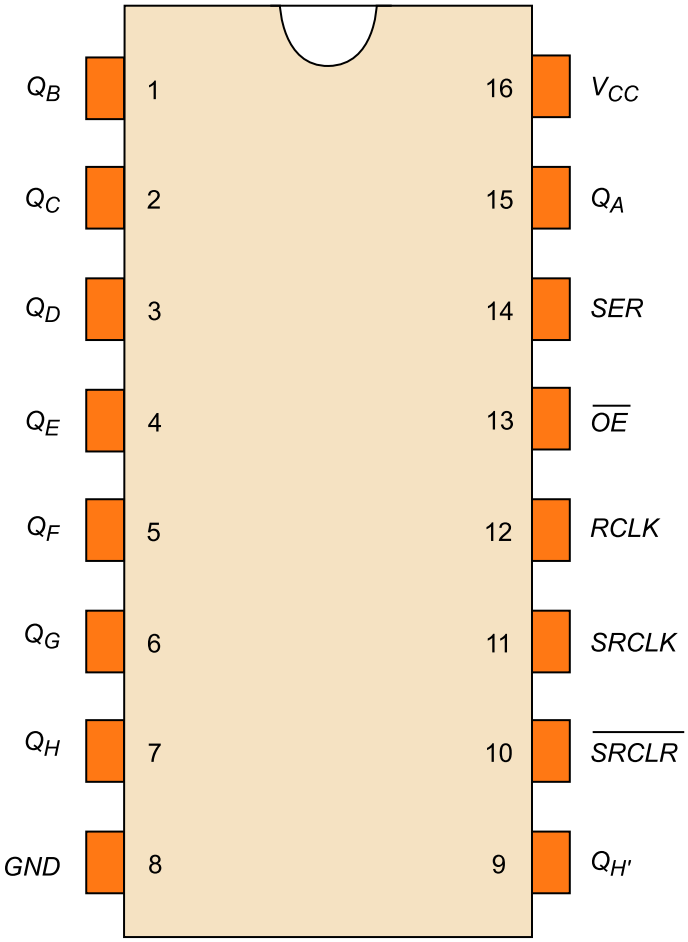
Figura 28. Registro de desplazamiento universal



Los registros de desplazamiento son utilizados normalmente como interfaz con sistemas digitales separados situados remotamente. Por ejemplo, supongamos que es necesario transmitir un cierto mensaje de *n* bits entre dos puntos. En lugar de considerar el uso de *n* líneas de manera simultánea, podríamos pensar en "serializar" estos datos en una sola línea, y transmitir estos bits unos tras otros. Los introduciríamos en un registro de desplazamiento (carga en paralelo) e iríamos desplazando el contenido de modo que saldría en serie por un extremo. En recepción se podría llevar a cabo la operación inversa, con un registro que, una vez cargado todo el mensaje, leyera en paralelo. Vemos, pues, que se trata de una solución ideal para conversiones serie a paralelo o viceversa.

Un buen ejemplo de circuito integrado comercial con funciones de registro de desplazamiento sería el 74HC595, que vemos representado en la figura 29 según la nomenclatura de pines del fabricante.

Figura 29. Registro de desplazamiento 74HC595



En este caso tenemos 8 bits de registro, denominados de Q_A a Q_H, que son las ocho salidas paralelo del circuito, y una sola entrada serie, *SER*. Para este pin se van introduciendo los bits, que después vemos en las salidas paralelo *Q*. También hay un conjunto de pines que proporcionan las funciones necesarias para la operación:

- *SRCLR*: permite poner a cero todos los registros[16], y actúa en estado bajo.

- *OE*[17]: permite que las salidas se encuentren en un estado de alta impedancia, un tercer estado, para tener la posibilidad de inhabilitarlas y que su valor no entorpezca un bus de salida al cual las conectaremos. Actúa en estado bajo.

- *RCLK* y *SRCLK*: pin de entrada para señales de reloj. Permite tener un reloj para la entrada serie y otro para los registros.

En la figura 30 podemos ver la tabla de verdad del circuito, según la hoja de especificaciones del fabricante. La cruz X indica un estado indiferente; se trata de casos en los que solo se tiene en cuenta alguna de las entradas para actuar según se indica.

[16] En inglés, *shift-register clear*.

[17] El nombre proviene de la expresión inglesa *output enable*.

Figura 30. Tabla de verdad del 74HC595, según la hoja de especificaciones del fabricante

INPUTS					FUNCTION
SER	SRCLK	$\overline{\text{SRCLR}}$	RCLK	$\overline{\text{OE}}$	
X	X	X	X	H	Outputs Q_A–Q_H are disabled.
X	X	X	X	L	Outputs Q_A–Q_H are enabled.
X	X	L	X	X	Shift register is cleared.
L	↑	H	X	X	First stage of the shift register goes low. Other stages store the data of previous stage, respectively.
H	↑	H	X	X	First stage of the shift register goes high. Other stages store the data of previous stage, respectively.
X	X	X	↑	X	Shift-register data is stored in the storage register.

Otra función interesante de este circuito integrado es un pin denominado *QH'*. Cuando hagamos un desplazamiento de los registros, este pin contiene el pin sobrante del otro extremo. Esto permite conectarlo a un circuito integrado 74HC595 subsiguiente a su entrada serie *SER* y construir un registro de desplazamiento de 16 bits (con los relojes y las otras señales en común). De este modo, podríamos ir encadenando circuitos de 8 bits del tamaño que consideremos necesario para nuestra aplicación.

2.2. Contadores

Los contadores son circuitos con mucha importancia en la electrónica, que permiten llevar un recuento del factor tiempo dentro de cualquier circuito que nos planteemos diseñar. Pueden ser utilizados para contar acontecimientos, como por ejemplo el número de pulsos de reloj en un tiempo determinado (para medir la frecuencia). Forman parte de los sistemas secuenciales, ya que hacen uso de algún registro de memoria para ir calculando el paso del tiempo de manera incremental. Podemos destacar sus características siguientes:

- Recuento hacia adelante o hacia atrás.
- Operación asíncrona o síncrona.
- Número máximo que se puede contar (módulo del contador).
- Modo libre o parada propia.

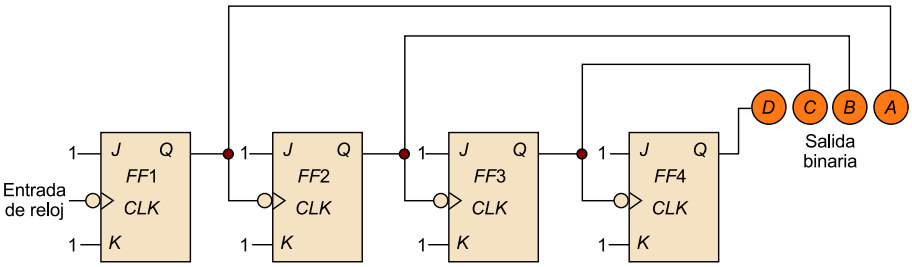
Tal y como sucede en otros circuitos secuenciales, utilizaremos biestables como base para la construcción de los contadores. En este caso, veremos que se basan típicamente en biestables *JK*, cuyo funcionamiento recordaremos brevemente.

2.2.1. Contador simple de 4 bits

Empezaremos por plantear un contador digital simple de 4 bits síncrono. Queremos que su funcionamiento sea contar hacia delante, desde su estado 0000 hasta su estado 1111, que se corresponderían con los valores decimales 0 a 15,

respectivamente. Un contador de este tipo se denomina **contador de módulo 16**. El módulo de un contador es el número de valores por los que pasa al completar todo un ciclo, antes de volver a comenzar el ciclo siguiente.

Figura 31. Contador de 4 bits basado en biestables *JK*



Una implementación de este contador con biestables *JK* es la que encontramos en la figura 31. Antes hay que recordar cuál es la tabla de verdad de un biestable *JK*, y la encontramos representada en la tabla 9. Vemos que permite o bien mantener el estado anterior (cuando *JK* es 00), o bien invertirlo (cuando *JK* es 11). El caso en el que los *JK* son 11 se denomina *conmutar*[18]. También podemos imponer un 0 o un 1, según la tabla.

[18] En inglés, *toggle*.

Tabla 9. Tabla de verdad de un biestable *JK*, con entradas *J* y *K* y salida *Q*

J	*K*	*Q*
0	0	Mantiene el estado anterior
0	1	0
1	0	1
1	1	Invierte el estado anterior (*toogle*)

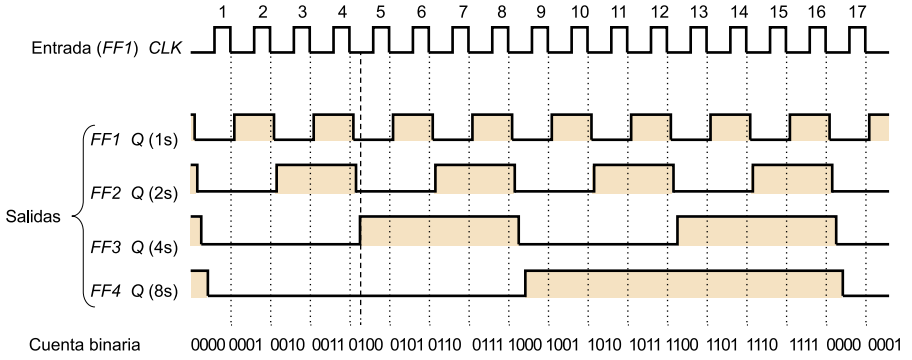
En lo que respecta al contador de la figura 31, en primer lugar hay que observar que todas las entradas *J* y *K* están siempre conectadas a un estado lógico 1. Con esto forzamos a que el biestable esté siempre en modo de conmutación o cambiando; es decir, en cada **flanco descendente** de su entrada de reloj *CLK* la salida cambiará con respecto al valor anterior (tabla 9). La bola de negación a la entrada de *CLK* de los biestables indica este hecho: el cambio es por flanco descendente.

Observad también que el primer biestable (izquierda) sí tiene la entrada de reloj conectada al reloj del sistema, pero el resto de los biestables tienen las salidas directamente conectadas a la entrada de reloj del siguiente. Finalmente, también hay que comentar que, en la salida binaria, *D* representa el bit más significativo y *A* el menos significativo.

Así pues, si analizamos su comportamiento, vemos que el primer biestable (a la izquierda) cambiará su estado en cada flanco descendente del reloj, lo que es el comportamiento esperado de un biestable *JK* (se podría haber configurado también para que actuara con flancos ascendentes de *CLK*). Con esto conse-

guimos que la salida *A* (menos significativa) vaya variando en una unidad, como pretendíamos. Esta variación se traslada al segundo biestable, pero puesto que solo cambiará el estado en flancos descendentes (y no de subida), su ritmo de cambio será la mitad. Y lo mismo para el tercero, a un ritmo de la mitad del segundo. De este modo, conseguimos que los bits vayan cambiando tal y como pretendíamos, entre 0000 y 1111, de manera unitaria en valores decimales. Vemos representado este ciclo de tiempo en la figura 32.

Figura 32. Cronograma de tiempo del contador de 4 bits por flanco descendente



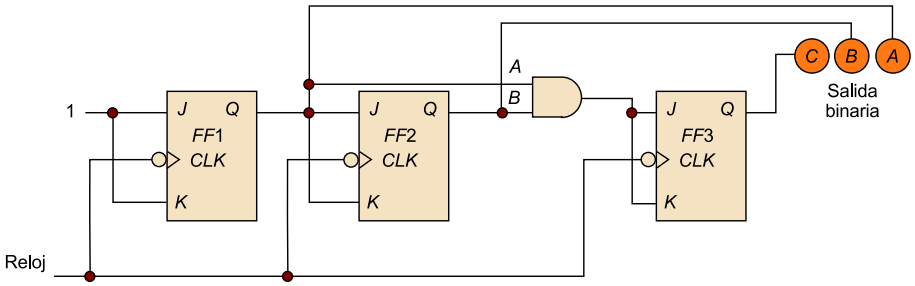
Vemos que cada señal de salida tiene, de hecho, una frecuencia que es la mitad de la anterior. Esta es otra función inherente a los contadores, como lo es el hecho de ser divisores de la frecuencia del reloj en múltiples de 2. La única desventaja que podemos encontrar en esta configuración es la propia propagación de la señal entre biestables, lo que hace que haya un cierto retraso acumulado entre el primero y el último. Se representa por los pequeños retrasos dibujados también en la figura 32.

2.2.2. Contador paralelo

El contador presentado en el subapartado anterior tiene la limitación del tiempo de retraso del cambio de estado de todos los biestables. Para solucionar este problema, se puede plantear el uso de contadores paralelos.

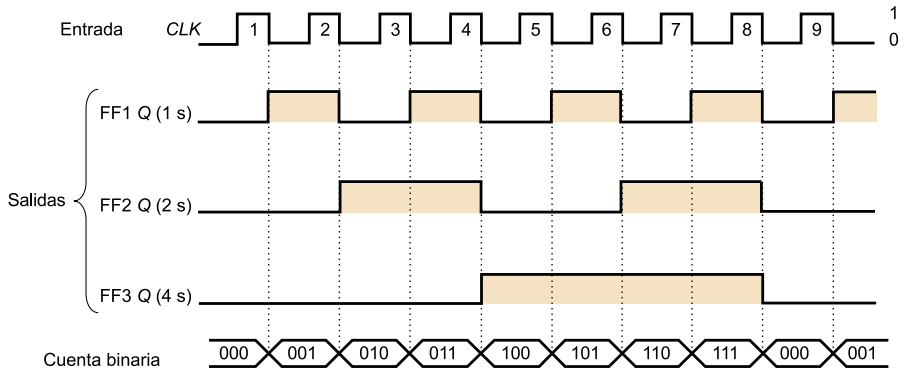
Planteamos el diagrama lógico de un contador paralelo de 3 bits en la figura 33, para entender su funcionamiento. Vemos que en este caso el reloj externo está conectado a todas las entradas *CLK* de los biestables, para asegurar que vayan conmutando al mismo tiempo. El primer biestable (a la izquierda) se alimenta con valores altos para *J* y *K*, con el objetivo de forzar el modo de conmutación, al igual que antes. Esto hace que en cada flanco descendente del reloj externo haya conmutación y, por lo tanto, se comporta como pretendemos para representar el bit menos significativo *A*.

Figura 33. Diagrama de contador paralelo de 3 bits



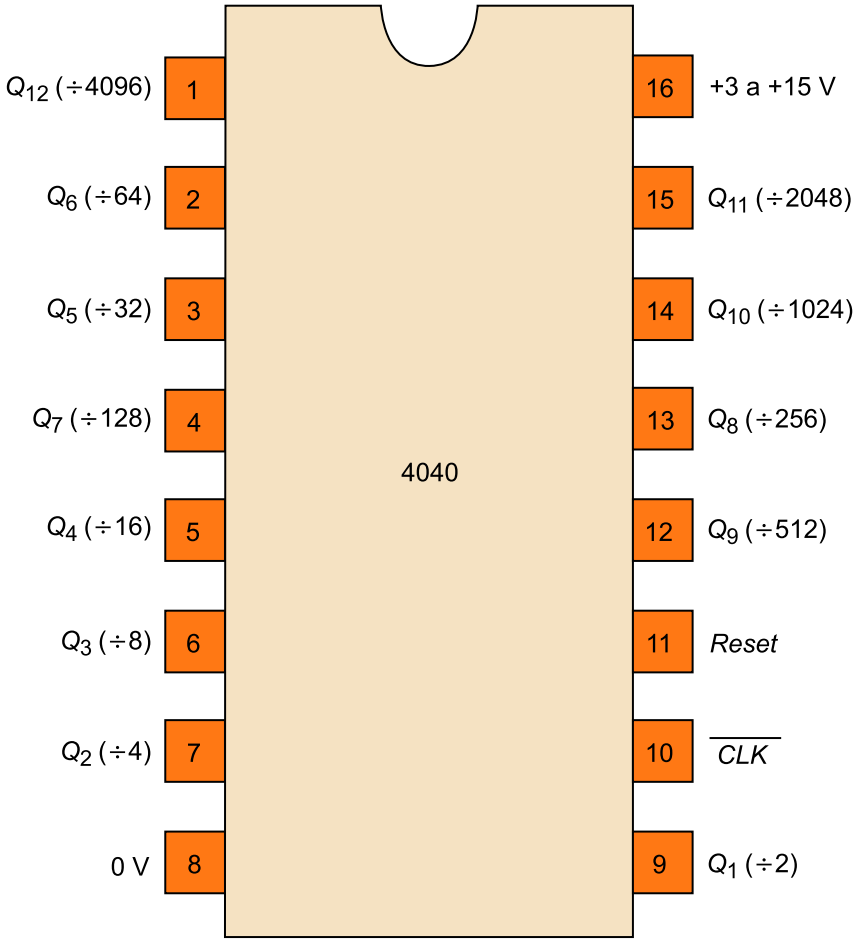
Para entender el funcionamiento del segundo biestable hay que recordar que, cuando las entradas *J* y *K* se encuentran en estado bajo (0), su salida *Q* se mantiene en el estado anterior (recordad su tabla de verdad en la tabla 9). Por lo tanto, solo conmutará si el *CLK* tiene flanco descendente y además en el bit *A* hay un 1, lo que proporciona el bit *B* a la salida. La frecuencia de cambio será, pues, la mitad de la del bit *A*. Del mismo modo, el tercer biestable solo conmutará cuando se den estas condiciones y, además, el bit *B* sea 1 (observemos la puerta AND entre *A* y *B*). Así pues, el bit *C* se moverá a la mitad de frecuencia que el *B*. Todo esto está representado en la figura 34, en la que vemos las formas de onda en el tiempo para los tres bits de salida. Observamos que ahora las formas de onda se sincronizan en el tiempo, independientemente de su posición dentro del diagrama, y de aquí el nombre de *contador paralelo*.

Figura 34. Diagrama de tiempo del contador paralelo de 3 bits



Hemos visto, pues, dos configuraciones para hacer contadores módulo 16 y módulo 8. Un ejemplo de circuito integrado comercial, en este caso de 12 bits, sería el circuito integrado 4040, que vemos representado en la figura 35.

Figura 35. Esquema del contador 4040



Podemos ver que dispone de doce salidas (Q_1 ... Q_{12}) que van proporcionando la cuenta de los pulsos de entrada en el pin 10 (reloj por flanco descendente). También dispone de los dos pines de alimentación (números 8 y 16), y de un pin para el *reset* (número 11).

A continuación, veremos cómo hacer un contador que tenga módulo decimal (módulo 10, por ejemplo), que son muy utilizados también en todo tipo de aplicaciones. En muchos circuitos debemos poder trabajar directamente con valores decimales.

2.2.3. Contador de década

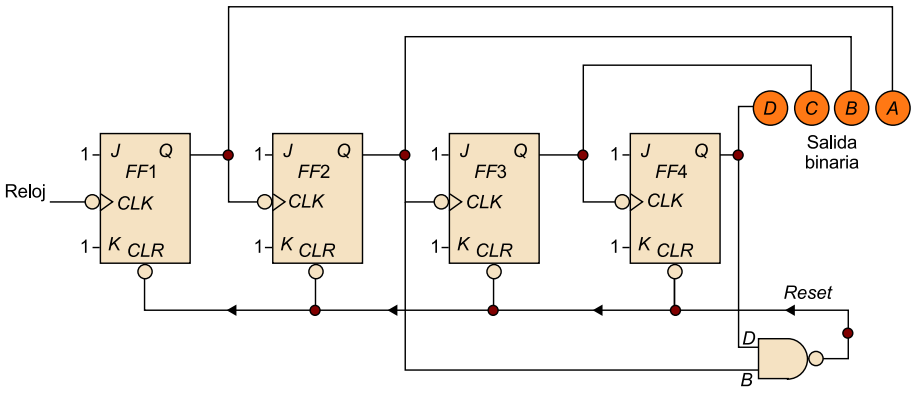
Para muchas aplicaciones, nos puede interesar que el ciclo del contador tenga módulo decimal, es decir, que cuente en módulo 10, por ejemplo. Para construir un dispositivo de este tipo, nos podemos basar en un contador simple de 4 bits módulo 16. Por lo tanto, tendrá también la limitación de tiempo de propagación que hemos visto, pero es una característica que, dependiendo de la aplicación y la velocidad que requiera el circuito, no tiene por qué convertirse en un problema.

Ved también

Podéis ver el contador simple de 4 bits en el subapartado 2.2.1 de este módulo didáctico.

Sobre aquel circuito, deberemos añadir una lógica que fuerce a que, una vez el contador llega a 10 (1010 en binario), los biestables tengan un *Reset* y el sistema empiece otra vez por 0 (0000 en binario). Esto es posible con el sistema de la figura 36.

Figura 36. Diagrama de contador módulo 10

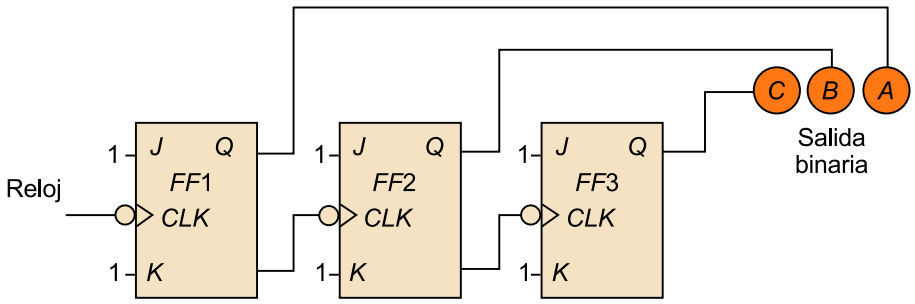


En la figura 36 vemos que se trata del mismo sistema módulo 16, al que hemos añadido una puerta NAND. Esta puerta, cuando detecta que tanto el bit *B* como el *D* son 1, establece una señal 0 a *Reset*, que hace que los cuatro biestables vuelvan al estado inicial (0000). Por lo tanto, cuenta entre 0 (0000) y 9 (1001). Hay que destacar que la entrada de *Reset* de los biestables actúa por flanco descendente (como se indica en el diagrama con negación).

2.2.4. Contador atrás

Antes hemos comentado que una característica de los contadores consiste en que hagan incrementos o decrementos de valores, es decir, que cuenten adelante o atrás. A continuación veremos qué configuración podemos utilizar para conseguirlo, a partir de los mismos biestables que conocemos.

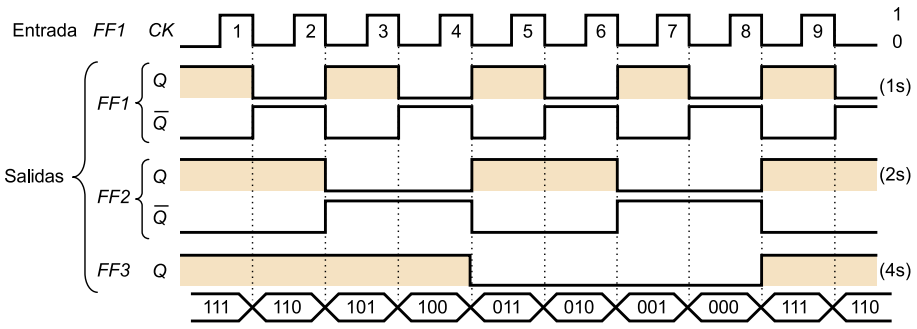
Figura 37. Diagrama de contador hacia atrás



En la figura 37, vemos un contador de 3 bits que ha sido modificado para que el sentido del contador sea hacia atrás. En concreto, observamos que ahora la salida $\overline{Q}$ es la que se encadena entre las señales de *CLK* de los biestables, mientras que las entradas *J* y *K* siempre se mantienen a 1.

Imaginemos que empezamos en el estado 111; el primer biestable cambiará de estado a cada flanco descendente del reloj, como ya sabemos. Este cambio al valor 0 se verá en la salida Q, mientras que la salida $\overline{Q}$ será 1 y, por lo tanto, esto no supondrá ningún cambio en el segundo biestable. Hasta el cambio siguiente en el primer bit, no conmutará hacia 0. Y lo mismo para el tercer biestable. Esto hace que las formas de onda sean las mostradas en la figura 38, en la que vemos cómo, empezando a 111, vamos bajando en incrementos de 1 hasta 000, y el ciclo vuelve a empezar con 111.

Figura 38. Diagrama de tiempo del contador hacia atrás de 3 bits



Podríamos disponer los mismos elementos sobre la topología de un circuito con más biestables y construir contadores del número de bits que nos hiciera falta. También habría que tener en cuenta posibles limitaciones con respecto a la propagación de los flancos ascendente y descendente entre las señales internas del circuito.

Para resumir, en este apartado hemos sido capaces de entender, a partir de los subsistemas secuenciales más básicos, cómo llegar a construir circuitos más complejos como pueden ser los contadores (con distintas características) o los registros de desplazamiento en su variante más universal.

3. Subsistemas típicos en circuitos analógicos

En apartados anteriores, hemos tratado un conjunto de subsistemas típicos basados en lógica digital, tanto de la rama de sistemas combinacionales como los secuenciales. En este apartado, nos interesa conocer también algunos ejemplos típicos de subsistemas que funcionen en un contexto de **circuito analógico**. A partir de elementos básicos (como los transistores), veremos algunos bloques que se pueden construir en la escala de integración siguiente (recordemos la figura 1) y que proporcionan funciones analógicas.

3.1. Interruptor analógico

Un **interruptor analógico** es un componente que se comporta de manera similar a un relé mecánico, pero al estar basado en dispositivos de estado sólido no tiene partes móviles, y la conmutación se consigue a nivel eléctrico. Esto es posible gracias a los transistores MOSFET.

Ved también

Podéis ver los transistores MOSFET en el módulo "Circuitos CMOS" de esta asignatura.

En efecto, podemos pensar en un MOSFET que opera **en conducción** o **en corte** y que, por lo tanto, deja pasar o no corriente eléctrica. Este estado ON u OFF se puede controlar con la tensión de puerta, que haría que el interruptor proporcione o no contacto. El dispositivo puede conducir señales en cualquier dirección cuando está ON, y aísla los terminales entre sí cuando está OFF.

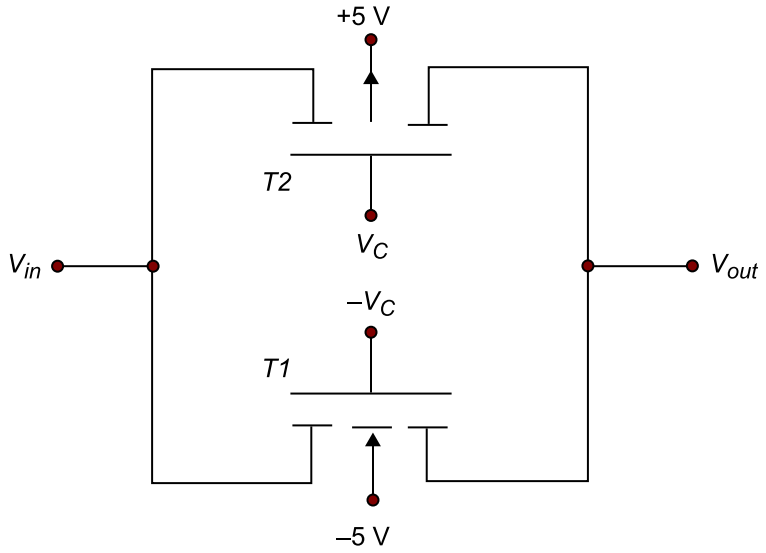
De hecho, hay muchas configuraciones de circuito que se podrían utilizar, no solo un único transistor MOSFET. Podríamos plantear un par de transistores, uno de canal *N* y uno de canal *P*, con la topología de la figura 39. Sería un interruptor CMOS, en el que la tensión V_C sería el control que regula el estado ON u OFF.

Los dos transistores se conectan de tal manera que sus terminales fuente están en lados opuestos del circuito (es decir, uno está a la entrada y otro a la salida). Por lo tanto, sus terminales de drenador también están en lados opuestos. La tensión de control V_C puede ser o bien +5 V o bien –5 V, de modo que podemos provocar que los dos transistores estén en corte (y por lo tanto, el interruptor en OFF), o bien que al menos uno de los dos esté en conducción (y de este modo, todo el bloque se considera en ON). En este último caso, el transistor que conduzca dependerá de la corriente de la señal de entrada V_{in}.

Por el hecho de utilizar dos transistores en lugar de uno, el fabricante tendrá más opciones para mejorar los parámetros del interruptor. En general, este tipo de interruptor proporciona un buen aislamiento (alta impedancia) entre el terminal de control y los de entrada y salida, y hay que destacar que no

se utiliza para conmutar señales de alta tensión: al tratarse de dispositivos de estado sólido ya sabemos que tenemos que limitar las tensiones dentro de un rango de funcionamiento seguro, que son del orden de algunos voltios.

Figura 39. Interruptor analógico CMOS



Al hablar de interruptores analógicos, hay que saber qué tipos de parámetros o especificaciones son importantes. En este caso, podemos hablar de los siguientes.

- **Resistencia ON:** se trata de la resistencia cuando el interruptor "hace contacto". Estos valores están típicamente entre 5 Ω y unos pocos centenares de ohmios.

- **Resistencia OFF:** es la resistencia cuando el interruptor "está apagado". Típicamente, se trata de un valor de más de 10^6 Ω.

- **Rango de señal:** las tensiones máximas y mínimas permitidas de la señal que pasa por el circuito.

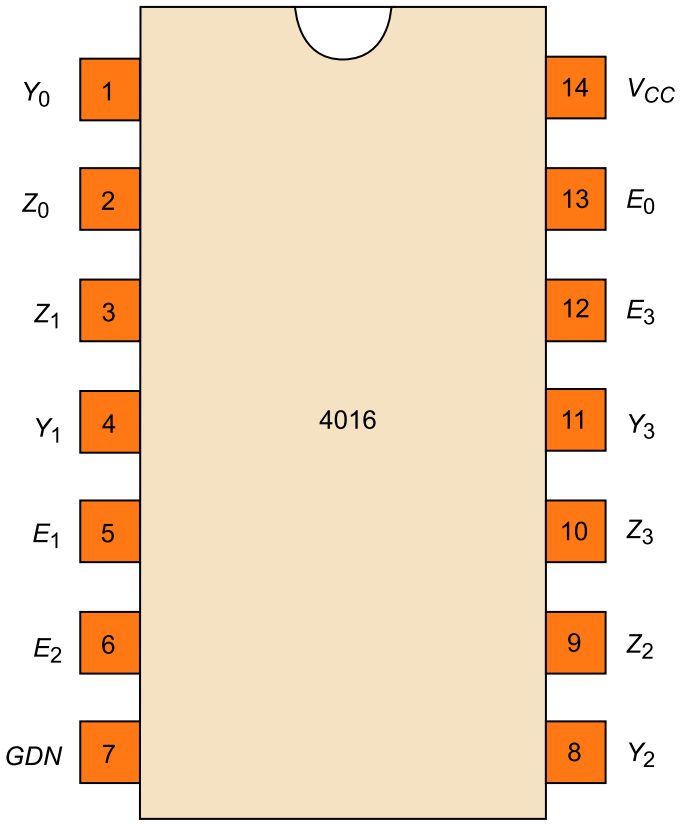
- **Inyección de carga:** este efecto hace que el interruptor inyecte una pequeña carga eléctrica en la señal cuando se pone en estado ON, y provoca un pequeño pico de señal.

Estos interruptores se pueden encontrar en la forma de circuitos integrados, que en muchos casos contienen múltiples interruptores (típicamente dos, cuatro u ocho). Entre estos, encontramos el circuito integrado 4016, que describimos a continuación.

En la figura 40, observamos el diagrama de pines del interruptor analógico 4016, que de hecho contiene cuatro circuitos independientes basados en CMOS. Además de la alimentación V_{CC} y *GND*, por un lado las entradas/salidas independientes de los cuatro interruptores vienen dadas por Y_0, Y_1, Y_2, Y_3

y Z_0, Z_1, Z_2, Z_3, y las señales de control (ON en estado alto) son E_1, E_2, E_3 y E_4. Sus parámetros típicos son una resistencia ON de 120 Ω cuando V_{CC} es de 9 V, y un margen de operación de tensiones V_{CC} entre 2 V y 10 V. Soporta unas corrientes del orden de decenas de mA.

Figura 40. Interruptor analógico 4016



3.2. Multiplexor analógico

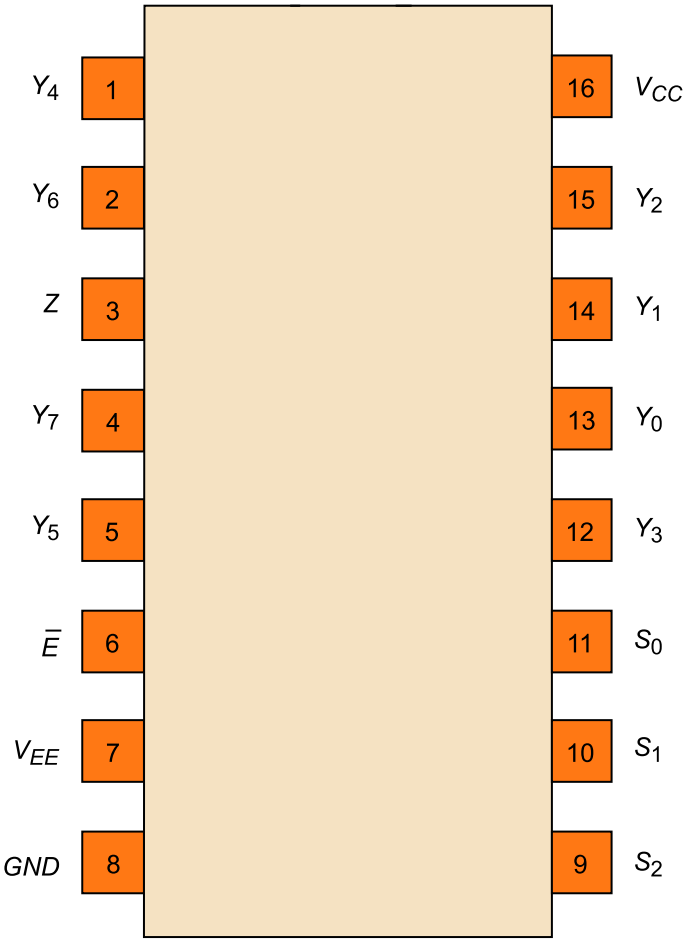
Ya hemos expuesto la función básica de los multiplexores y demultiplexores digitales. Recordemos que se trata de seleccionar una señal de salida entre un conjunto de señales de entrada, y a la inversa. De hecho, ahora que ya tenemos el concepto de interruptor aprendido, podemos afirmar que un **multiplexor** se puede considerar como un interruptor de múltiples entradas y una sola salida, y un **demultiplexor**, un interruptor de una sola entrada y múltiples salidas.

Ved también

Podéis ver la función básica de los multiplexores y demultiplexores digitales en el subapartado 1.5 de este módulo didáctico.

En el mundo analógico, y basados en la misma tecnología que los interruptores analógicos, podemos encontrar circuitos integrados que manejan señales de entrada y salida dentro de unos rangos concretos de tensión y corriente. Como ejemplo tenemos el circuito integrado 4051, representado en la figura 41. Se trata de un multiplexor y un demultiplexor al mismo tiempo puesto que, por construcción interna, las entradas pueden ser también las salidas, y a la inversa, y pasa de 1 a 8 o de 8 a 1.

Figura 41. Multiplexor/demultiplexor analógico 4051



En este caso, disponemos de hasta 8 entradas/salidas independientes, denominadas Y_0 ... Y_7, que se pueden conectar o desconectar del pin Z, según el direccionamiento proporcionado por S_0 ... S_2. Es el usuario quien elige si necesita utilizarlo en un sentido u otro; lo que hace el circuito integrado es conectar eléctricamente los pines Y con el Z.

Además, también hay dos posibles tensiones de alimentación V_{CC} y V_{EE}, así como un pin de *enable* o habilitación en estado bajo $\overline{E}$. Sus parámetros están en el orden de centenares de W para la resistencia ON, una tensión de alimentación máxima de 11 V y unas corrientes máximas de en torno a 20 mA.

3.3. Referencia de voltaje

Otro subsistema muy común en circuitos analógicos es un circuito que proporcione una tensión o un **voltaje de referencia**. Se trata de un módulo cuya funcionalidad es proporcionar una tensión fija y constante, de manera independiente de su carga de impedancia, variaciones de la alimentación y cambios de temperatura y del tiempo.

Estos sistemas se utilizan en reguladores de tensión en fuentes de alimentación, conversores analógicos-digitales y digitales-analógicos y todo un amplio abanico de sistemas de control y medida. De hecho, la tecnología y las prestaciones de estos sistemas varían enormemente según la aplicación. Un regulador para una fuente de tensión de un ordenador puede mantener su valor dentro de un pequeño porcentaje del valor nominal, mientras que tensiones de instrumentos de laboratorio tienen precisiones y estabilidades medidas en partes por millón.

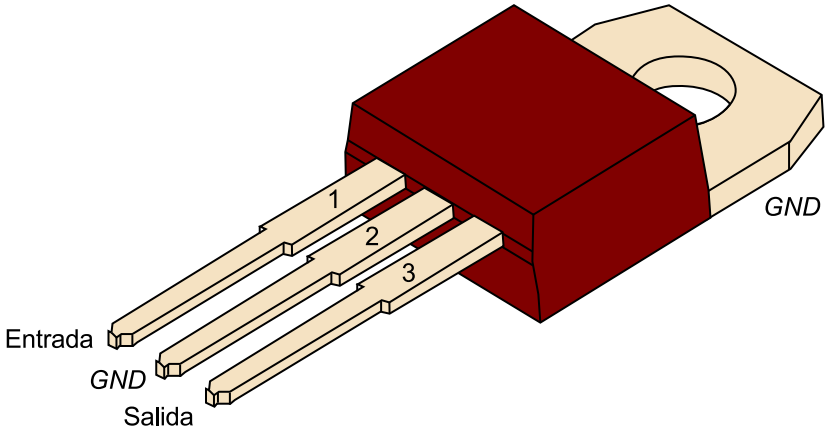
La aplicación, por lo tanto, nos marcará qué tecnología de regulador necesitamos. En nuestro caso, nos interesa hablar de reguladores basados en dispositivos de estado sólido, y en este caso destacaríamos, por ejemplo, el uso de diodos. Como sabemos, el **diodo** tiene una característica *I-V* con forma exponencial, con un "codo" que se puede utilizar como referencia de tensión. Este codo puede estar en 0,3 V para los diodos de germanio, o en 0,6-0,7 V para los diodos de silicio. Dado que esta referencia tiene una dependencia fuerte con la temperatura, estos dispositivos se utilizan precisamente como compensadores de efectos de temperatura en la tensión de circuitos analógicos. El diodo utilizado más ampliamente como referencia es el Zener, puesto que su tensión de ruptura es muy estable y precisa, suficiente para muchos dispositivos electrónicos.

En el mercado encontramos multitud de circuitos integrados de estado sólido que nos proporcionan un bloque capaz de regular tensiones, independientemente de si están basados en diodos o en otros tipos de topologías. Como ejemplo tendríamos la familia de circuitos integrados 78xx, que son de funcionamiento muy sencillo y muy populares. La parte xx indica la tensión regulada de salida. De este modo, por ejemplo, el circuito integrado 7805 proporciona 5 V, y es un sencillo circuito integrado de solo tres pines:

- Pin 1: tensión de entrada, entre 5 V y 18 V.
- Pin 2: masa (0 V).
- Pin 3: tensión regulada de salida, a 5 V. Rango posible entre 4,8 y 5,2 V.

En la figura 42, vemos un diagrama de estos circuitos integrados 78xx de tres pines.

Figura 42. Diagrama de un regulador de tensión 78xx



Fuente: Wikipedia

Resumen

En este módulo hemos cubierto todo un conjunto de circuitos que, por su nivel de complejidad, ya se denominan *subsistemas*. Están en una escala media de integración, y se sitúan entre los dispositivos básicos que son los constituyentes de la electrónica hoy día, como los transistores, y los sistemas complejos programables, como las FPGA y los ASIC, que estudiaremos en los siguientes módulos.

Hemos visto cómo a partir de puertas lógicas básicas somos capaces de construir subsistemas con unas funcionalidades más complejas que las de un solo transistor o una sola puerta lógica, y al mismo tiempo muy útiles para disponer de bloques y prestaciones avanzadas.

Hemos conocido, por un lado, sistemas con lógica combinacional que se orientan a proporcionar un valor de información a la salida del subsistema a partir de la información recibida en aquel momento. En estos sistemas, podemos destacar funciones como las del sumador, el restador o el comparador, aplicadas con lógica binaria que represente números decimales con signo positivo y negativo.

Por otro lado, hemos profundizado en subsistemas secuenciales, en los que se trata de registrar valores binarios para hacer un uso concreto de ellos más adelante. Son elementos con una cierta capacidad de memoria y que nos proporcionan funciones como el registro, el desplazamiento de bits, la conversión serie-paralelo y la posibilidad de disponer de contadores de números para todo tipo de aplicaciones.

Finalmente, hemos visto algunos ejemplos sobre subsistemas típicos analógicos, como los interruptores o las referencias de voltaje.

Glosario

acarreo *m* En una operación aritmética, indica el arrastre de una cifra a la siguiente cifra más significativa.

CI *m* Sigla de *circuito integrado*.

CMOS *m* Sigla de la expresión inglesa *complementary metal oxide semiconductor* (semiconductor óxido-metal complementario). Tecnología empleada en la fabricación de circuitos integrados.

MOSFET *m* Sigla de la expresión inglesa *metal oxide semiconductor field effect transistor* (transistor de efecto de campo en semiconductor metal-óxido). Una de las tecnologías empleadas en la fabricación de circuitos integrados.

Bibliografía

Mano, M. (2002). *Digital design* (3.ª ed.). Nueva Jersey: Prentice Hall.

Tokheim, R. L. (1994). *Theory and problems of digital principles*. McGraw Hill.

Wakerly, J. F. (1999). *Digital design. Principles and practices* (3.ª ed.). Nueva Jersey: Prentice Hall.