Estructura y Tecnología de Computadores

Alberto J. Molina Cantero
Sergio Díaz Ruiz
José Ignacio Escudero Fombuena

Departamento de Tecnología Electrónica
Universidad de Sevilla

Editorial Panella

Prólogo

Al escribir este libro no teníamos muy claro qué título le íbamos a poner. Como habrá comprobado el avispado lector al final nos decidimos por el nada original de "Estructura y Tecnología de Computadores". También se podría haber titulado "Fundamentos de Computadores" o cualquier otro título parecido. Los tres autores del libro somos profesores en la Escuela Técnica Superior de Ingeniería Informática de la Universidad de Sevilla e impartimos una asignatura en primer curso de Ingeniería Técnica en Informática de Gestión sobre Electrónica Digital, ¿cómo se llama esa asignatura? Está claro que igual que nuestro libro. También pensamos si le añadíamos al título un "1" al final, como "volumen 1", para indicar que en este libro desarrollamos sólo la parte más básica de esta materia. Desistimos de ello porque detrás del "1" debe venir el "2" y por ahora no tenemos intenciones de escribir nada más.

¿De qué trata esta asignatura? Del diseño de circuitos electrónicos digitales. No, por favor, no deje de leer, aguante un poco más. Hoy en día ya sabrá que vivimos inmersos en la llamada era digital, todo a nuestro alrededor es "digital", tanto es así que sería mucho más fácil decir qué no es digital. Piense, por ejemplo, en el teléfono, el microondas, la lavadora, la fotografía, el coche, los aviones, los satélites, la medicina, la música... ¿sigo? Creo que no hace falta. En todos ellos hay inmerso, en mayor o menor medida, un circuito digital, "algo" capaz de realizar infinidad de pequeñas tareas en un periodo de tiempo muy breve, que dan como resultado que hablemos a distancia, que se caliente el café, que centrifugue la lavadora, o que nos hagan un TAC. ¿No le parece maravilloso? ¿No le gustaría saber cómo se consigue eso? En este libro empezaremos a andar por ese mágico camino.

¿Hay que saber mucho para empezar a andar en esta materia? Nada. Sí, ha leído bien, para empezar no se necesitan conocimientos previos. Bueno, para ser sinceros, sólo hace falta saber distinguir un "1" de un "0". ¿Ha superado la prueba inicial de conocimientos? Pues bienvenido al mundo digital, le aseguro que no se arrepentirá. ¿Qué vamos a ver en este libro? Muchas cosas. Para empezar, en el capítulo 1, veremos cómo es la información digital que manejan estos circuitos, cómo es posible representar todo con sólo dos conceptos: blanco-negro, encendido-apagado, arriba-abajo, uno-cero. Después, ya en el capítulo 2, nos dedicaremos a estudiar las leyes que rigen el comportamiento de esa información digital. ¿Sabía que $1 + 1 = 1$? No se desanime.

En los capítulos siguientes, capítulos 3 y 4, veremos algunos circuitos muy básicos que son la base de todos los demás circuitos digitales, se les da el original nombre de "puertas". Mostraremos sus propiedades así como sus características más importantes. También estudiaremos un tipo especial de circuitos digitales, los llamamos "circuitos combinacionales", son aquéllos en los que entre cualquier punto del circuito y la salida de éste sólo hay un único camino posible. Aprenderemos a averiguar qué hace concretamente un circuito que nos den ya diseñado, en lo que llamamos análisis de circuitos, o a diseñar nosotros mismos un circuito de ese tipo que nos permita, por ejemplo, evitar una catástrofe en un avión por mal funcionamiento de la electrónica con un circuito "votador", en lo que denominamos síntesis de circuitos. El capítulo 5, también dedicado a los circuitos combinacionales, nos mostrará un tipo especial de estos circuitos, son aquéllos que son capaces de llevar a cabo tareas aritméticas y lógicas, lo cual nos va a permitir hacer virguerías con la información digital que manejamos.

El libro acaba con los capítulos 6 y 7 dedicados a otro tipo de circuitos digitales, son los llamados "circuitos secuenciales". En estos circuitos se permite volver atrás y por eso entre un punto cualquiera del circuito y su salida pueden existir caminos diferentes, aunque lo más sorprendente de estos circuitos es que tienen memoria, son capaces de recordar el pasado, lo cual nos va a permitir almacenar información digital. Al igual que antes estudiaremos también su componente más básico, al que llamamos "biestable" así como el análisis y la síntesis de esos circuitos.

Con todo esto, estamos hablando de un cuatrimestre, estaremos en condiciones de seguir adelante y en otras asignaturas ser capaces de tareas más complicadas como diseñar sistemas digitales complejos o incluso un computador simple, pero todo eso requiere de otro libro, ¿se acuerda del volumen 2

que por ahora no escribiremos?

Antes de acabar este prólogo quisiéramos todavía hacer un par de cosas. La primera es que un libro de estas características tiene mucho trabajo detrás, estamos hablando de años y a pesar de haber sido repasado muchas veces estamos seguros de que contendrá errores, erratas y fallos de todo tipo. Por ello le pedimos que si encuentra algo de eso en nuestro libro o simplemente quiere hacer cualquier sugerencia para mejorarlo en ediciones posteriores, mándenos un mail con sus comentarios a cualquiera de los tres autores:

almolina@us.es
sdiaz@us.es
ignacio@us.es

Se lo agradecemos sinceramente.

La segunda cosa que no queremos dejar atrás antes de terminar es la de reconocer la ayuda recibida. Queremos dar las gracias a todos nuestros compañeros que en diferentes titulaciones imparten esta misma asignatura y que a lo largo de los años nos han permitido mejorar nuestros conocimientos sobre esta materia en multitud de comentarios, charlas y cafés, nos referimos a Carmen Baena, Manolo Bellido, David Guerrero, Isabel Gómez, Jorge Juan, Alejandro Millán, Pilar Parra, Paco Pérez, Mª Carmen Romero, Paulino Ruiz, Gemma Sánchez y muy especialmente al maestro: Manolo Valencia, quien en todo momento ha tenido tiempo para resolver dudas, aclarar conceptos y hacer múltiples sugerencias. Finalmente no queremos acabar sin agradecer a nuestros alumnos de tantos años que, con sus comentarios y preguntas en clase, nos han enseñado mucho más de lo que ellos creen.

Sevilla, Septiembre 2004

Alberto Molina
Sergio Díaz
José I. Escudero

Índice

Capítulo 1

Representación numérica

1.1. Representación posicional de magnitudes

La representación de magnitudes usa los símbolos numéricos habituales (dígitos del 0 al 9), situados en distintas posiciones, si las magnitudes que representan son mayores o iguales a diez. Esto hace que un mismo dígito tenga un valor diferente dependiendo de la posición relativa que éste ocupe en la representación de una magnitud. Por ejemplo, la magnitud representada por el símbolo 3234, expresa una cantidad igual a TRES mil, más DOS cientos, mas TRES decenas mas CUATRO unidades. En este ejemplo, al dígito TRES se le ha asociado una magnitud diferente (tres mil o treinta) según la posición relativa que ocupa en el símbolo 3234. Los dígitos que se sitúan a la izquierda representan magnitudes más significativas, o de mayor peso, que los situados a la derecha o menos significativos. En el símbolo 3234, y comenzando de izquierda a derecha, el peso asociado al dígito 3 es de mil unidades, al 2, de cien unidades, al otro 3, de diez unidades, y al 4, de una unidad. La magnitud representada por el símbolo 3234 puede reescribirse según la siguiente expresión:

$$3234 = 3 \times 1000 + 2 \times 100 + 3 \times 10 + 4 \times 1 \tag{1.1}$$

Los diez dígitos usados, desde el 0 al 9, representan, por sí solos, diez magnitudes diferentes. Cualquier magnitud mayor debe usar una combina-

ción posicional de los anteriores, en la que, a cada dígito que se añade por la izquierda se le asigna un peso de valor igual a una potencia de diez. El exponente, n, de dicha potencia, depende de la posición relativa de los números y cuya referencia la establece el dígito de la derecha, que ocupa la posición 0; el siguiente situado a su izquierda, la 1, y así sucesivamente. Conocida, pues, la posición, n, de un dígito, su peso se determina mediante la potencia de 10^n. El dígito con menor peso asociado se denomina el **dígito menos significativo**. Por el contrario, el dígito con mayor peso asociado es el **dígito más significativo** .

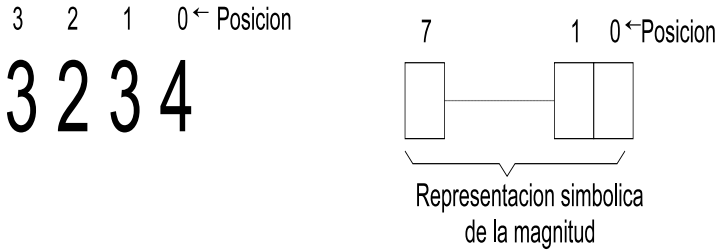


Figura 1.1. Indexación posicional para un número de 8 dígitos y ejemplo con 4 dígitos

Usando la notación exponencial para los pesos, se puede reescribir la expresión 1.1 de la forma dada por la expresión 1.2.

$$3234 = 3 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0 \tag{1.2}$$

Definición: *Al vector de dimensión k, formado por los pesos de los k dígitos de un número, se denomina* ***peso asociado de un número.***

El peso asociado al número 3234 es el vector $(1000\ ,100\ ,10,\ 1)$. Para el número 123, su peso asociado es $(100, 10, 1)$, y para el 23999, el $(10000,\ 1000,\ 100,\ 10,\ 1)$.

La base de representación numérica, B, utilizada hasta este momento, es la base diez o base decimal ($B = 10$). El valor de B muestra la cantidad de dígitos diferentes que pueden combinarse para representar magnitudes. Para el caso decimal, como es obvio, son diez: desde el dígito cero hasta el dígito nueve. Con el concepto de base B se puede reescribir el número decimal 3234

de la forma dada por la expresión 1.3.

$$3234 = 3 \times B^3 + 2 \times B^2 + 3 \times B^1 + 4 \times B^0 \;\; donde \;\; B = 10 \qquad (1.3)$$

Cada uno de los diez dígitos de la base decimal representan, por sí solos, una cantidad que varía desde cero unidades hasta nueve unidades. De forma general, en cualquier base de numeración B, existirán B dígitos diferentes que representarán, cada uno de ellos, magnitudes que vararán desde 0 unidades hasta $B-1$ unidades.

Sea a_i un dígito que representa una magnitud comprendida entre 0 y $B-1$, (expresado como $a_i \in [0,..,B-1]$), entonces a_i es un dígito en base B. El conjunto de los dígitos a_i que representan a las magnitudes comprendidas entre [0,..,B-1] constituyen una base de numeración. El símbolo formado por la unión de dígitos a_i, representa la magnitud del número, donde el subíndice i del dígito representa la posición que ocupa éste en el símbolo.

Cualquier magnitud, N, puede ser representada en una base genérica B y se expresa como, N_B, con n dígitos, a_i, de la forma dada por la relación 1.4.

$$N_B = a_{n-1} \times B^{n-1} + a_{n-2} \times B^{n-2} + \ldots + a_1 \times B^1 + a_0 \times B^0 = \sum_{i=0}^{n-1} a_i \times B^i \quad (1.4)$$

Una magnitud puede tener múltiples representaciones dependiendo de la base de numeración que se emplee. Por ejemplo, la magnitud 76 unidades, en base octal, viene representada por el número 114_8, ya que $114_8 = 1 \times 8^2 + 1 \times 8^1 + 4 \times 8^0 = 76 \; unidades$; en base B=4, por el número 1030_4, porque $1030_4 = 1 \times 4^3 + 0 \times 4^2 + 3 \times 4^1 + 0 \times 4^0 = 76 \; unidades$, etc.

De forma equivalente un numero cualquiera puede representar magnitudes diferentes si la base de numeración cambia. A continuación se muestran las magnitudes asociadas al número 1101 dependiendo de la base de numeración utilizada.

Ejemplos.

- $1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13\ unidades$
- $1101_4 = 1 \times 4^3 + 1 \times 4^2 + 0 \times 4^1 + 1 \times 4^0 = 81\ unidades$
- $1101_8 = 1 \times 8^3 + 1 \times 8^2 + 0 \times 8^1 + 1 \times 8^0 = 577\ unidades$
- $1101_{16} = 1 \times 16^3 + 1 \times 16^2 + 0 \times 16^1 + 1 \times 16^0 = 4353\ unidades$

Existen infinitas bases de numeración, tantas como posibles valores de B. Sólo unas pocas son de nuestro interés: la **base binaria**, en la que $B = 2$, la **base octal**, $B = 8$, la **base decimal**, $B = 10$ y la **base hexadecimal**, $B = 16$. Esta última, requiere del uso de símbolos o dígitos adicionales para representar las magnitudes comprendidas entre diez y quince. La tabla 1.1 ilustra los dígitos usados para estas bases de numeración.

Tabla 1.1: Dígitos usados en algunas de las bases de numeración más representativas

	Base 2	Base 8	Base 10	Base hexadecimal
0	0	0	0	0
1	1	1	1	1
2		2	2	2
3		3	3	3
4		4	4	4
5		5	5	5
6		6	6	6
7		7	7	7
8			8	8
9			9	9
10				A
11				B
12				C
13				D
14				E
15				F

Los dígitos de la base binaria, también llamados bits, son dos, el 0 y el 1. Los dieciseis dígitos de la base hexadecimal utilizan los diez dígitos conocidos para representar magnitudes menores a diez y las seis primeras letras del abecedario para las comprendidas entre diez y quince.

Ejemplos.

- $8E_{16} = 8 \times 16^1 + E \times 16^0 = 8 \times 16^1 + 14 \times 16^0 = 142_{10}$
- $1BC_{16} = 1 \times 16^2 + B \times 16^1 + B \times 16^0 = 1 \times 16^2 + 11 \times 16^1 + 12 \times 16^0 = 444_{10}$
- $\$1A0F = 1 \times 16^3 + A \times 16^2 + 0 \times 16^1 + F \times 16^0 == 1 \times 16^3 + 10 \times 16^2 + 0 \times 16^1 + 15 \times 16^0 = 6671_{10}$

Nótese que, la notación decimal, es la utilizada como referencia para la representación simbólica de magnitudes. La lectura de cualquier número, expresado en una notación diferente a la decimal, requiere de una traducción, o conversión, que permita la identificación de la magnitud indicada por dicho número.

1.1.1. Transformaciones entre sistemas de representación: cambios de base

En este apartado se presentan algunas técnicas que permiten la realización del cambio de base. Esto es, a partir de una magnitud N dada, simbolizada mediante un número en una base de numeración P, se desea obtener el número que, con una base de numeración Q diferente a P, representa la misma magnitud que el primero.

Se empezará con la transformación entre la base decimal y binaria, por la importancia de ambas.

1.1.1.1. Cambio de base decimal a base binaria

Dado un número N, en base decimal, se desea encontrar el equivalente en base binaria. Para ello se debe seguir el procedimiento que se muestra en el siguiente cuadro y que se ilustra en la figura 1.2 para el número decimal 23.

1. Sea el entero $i = 0$
2. Se divide el número N entre 2.
3. La división del punto 2 genera un resto que llamaremos a_i y un cociente C_i
4. Si el cociente C_i es distinto de cero, se hace $N = C_i$, se incrementa i y se repite desde el punto 2.
5. Si el cociente C_i es igual a cero, el proceso finaliza. El número en base 2 esta formado por el conjunto de los bits a_i donde el subíndice i indica la posición que ocupa cada bit en el número binario, esto es, el primer resto que se obtuvo (para i=0, a_0) es el bit menos significativo y, el último, el más significativo.

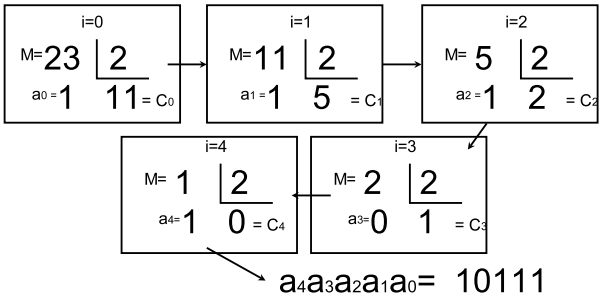


Figura 1.2. Ejemplo de cambio de base decimal a binaria

Obsérvese que al dividir cualquier número entre 2, el resto sólo puede ser 0 o 1, en definitiva, los dígitos en base dos.

1.1.1.2. Cambio de base binario a decimal

Se obtiene resolviendo o aplicando la expresión 1.5

$$N_{10} = \sum_{i=0}^{n-1} a_i \times 2^i \tag{1.5}$$

donde a_i son los dígitos o bits del número binario.

La suma extendida a todos los términos resultantes de la multiplicación de cada bit del número por su peso equivalente, enseña la magnitud del número

que, directamente, puede representarse en la base de decimal (base de referencia).

Ejemplos.

- $1001_2 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 9_{10}$
- $11101_2 = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 29_{10}$

1.1.1.3. Cambio de base decimal a base p

Sigue el mismo algoritmo que para la base 2, con la salvedad de que las divisiones se hacen entre el número p. Hay que tener especial cuidado si p es mayor que diez, puesto que algunos restos deberán ser convertidos a su correspondiente dígito en base p.

1. Sea el entero i = 0
2. Se divide el número N entre p.
3. La división del punto 2 genera un resto que llamaremos a_i y un cociente C_i
4. Si el cociente C_i es distinto de cero, se hace $N = C_i$, se incrementa i y se repite desde el punto 2.
5. Si el cociente C_i es igual a cero, el proceso finaliza. El número en base p esta formado por el conjunto de los bits a_i donde el subíndice i indica la posición que ocupa cada bit en el número binario, esto es, el primer resto que se obtuvo (para i=0, a_0) es el bit menos significativo y, el último, el más significativo.

Ejemplos.

- Representa la magnitud 23 en base 3
  - $23/3 \rightarrow a_0 = 2,\ C_0 = 7$

- $7/3 \rightarrow a_1 = 1,\ C_1 = 2$
- $2/3 \rightarrow a_2 = 2,\ C_2 = 0$

$23 = 212_3$

- Representa la magnitud 123 en base 12

  Sean α y β los dígitos que representan las magnitudes diez y once respectivamente en base doce.

  - $123/12 \rightarrow a_0 = 3,\ C_0 = 10 = \alpha$
  - $10/12 \rightarrow a_1 = 10 = \alpha,\ C_1 = 0$

  $123 = \alpha 3_{12}$

1.1.1.4. Cambio de base p a decimal

Se obtiene resolviendo o aplicando la expresión 1.6.

$$N_{10} = \sum_{i=0}^{n-1} a_i \times p^i \tag{1.6}$$

donde los coeficientes a_i representan los dígitos del número expresado en base p; n, el número de dígitos que contiene dicho número y N el número expresado en base 10.

1.1.1.5. Transformación de un número en base p a otra base q

Sea M un número expresado en base p, y N, su equivalente en base q. Se desea obtener N a partir de M. El método general se esquematiza en el siguiente cuadro.

1. Se transforma previamente M a base 10, usando las técnicas descritas en el apartado 1.1.1.4 Llamemos R al número resultante de dicha transformación.

2. Se transforma R, expresado en base 10, a base q, usando las técnicas descritas en el apartado 1.1.1.3

1.1.1.6. Casos especiales de cambio de base

Si la base de partida, p, del número cuyo cambio de base se desea, se relaciona con la base, q, de representación final mediante algunas de las siguientes expresiones $p = q^m$ o $p^m = q$, donde m es un número entero mayor que 1, entonces se pueden aplicar técnicas de compresión o expansión de dígitos, respectivamente.

Supóngase que se desea el cambio de base del número 11101_2, expresado en base p=2 a su correspondiente en base q=4. Se ve que la relación $p^m = q$ se cumple para m=2, lo que implica compresión. El cambio de base se realiza del siguiente modo: se forman **grupos de m dígitos** (en este caso m=2) de derecha a izquierda, es decir, desde el dígito menos significativo hasta el más significativo. El último grupo formado (el que contiene el dígito más significativo) puede contener un número de dígitos menor a m, por lo que se deberán añadir ceros hasta completar los m dígitos. Cada grupo formado, constituye un dígito de la representación final q.

Supóngase, ahora, que se desea el cambio de base del número 1230_4 (p=4= hacia su correspondiente en base q=2. En este caso se cumple $p = q^m$ para m=2 (expansión). Para la transformación se procede del siguiente modo: cada dígito del número en base p se expande a un grupo de m dígitos de la base de representación q. Cada grupo de m dígitos ocupa la misma posición relativa, con respecto a los restantes grupos, que la del dígito en representación p del que procede.

- **Cambio de base binaria a base octal**

  La base origen es p=2, y la destino q=8. Se cumple $p^m = q$ para $n = 3$. Esto implica compresión en grupos de 3 bits comenzando por el bit menos significativo.

  Ejemplos:

  - $1001111011_2 = 001\ \ 001\ \ 111\ \ 011_2 = 1173_8$
  - $11111010000000_2 = 011\ \ 111\ \ 010\ \ 000\ \ 000_2 = 37200_8$

- **Cambio de base binaria a base hexadecimal**

  La base origen es $p = 2$, y la destino $q = 16$. Se cumple $p^m = q$ para m=4. Esto implica compresión en grupos de 4 bits comenzando por el bit menos significativo.

Ejemplos:

- $1001111011_2 = 0010\ \ 0111\ \ 1011_2 = \$27B$
- $11111010000000_2 = 0011\ \ 1110\ \ 1000\ \ 0000\ \ 000_2 = \$3E80$

- **Cambio de base octal a binario**

  La base origen es $p = 8$, y la destino $q = 2$. Se cumple $p = q^m$ para m=3. Esto implica expansión en grupos de 3 bits.

  Ejemplos:

  - $7512_8 = 111\ \ 101\ \ 001\ \ 010_2 = 111101001010_2$
  - $2506_8 = 010\ \ 101\ \ 000\ \ 110_2 = \$10101000110_2$

- **Cambio de base hexadecimal a binario**

  La base origen es $p = 16$, y la destino $q = 2$. Se cumple $p = q^m$ para m=4. Esto implica expansión en grupos de 4 bits.

  Ejemplos:

  - $\$F10A0 = 1111\ \ 0001\ \ 0000\ \ 1010\ \ 0000_2 = 11110001000010100000_2$
  - $\$2506 = 0010\ \ 0101\ \ 0000\ \ 0110_2 = 0010010100000110_2$

1.1.2. Representación de magnitudes fraccionarias

Los dígitos pertenecientes a la parte fraccionaria de un número en base B, tienen unos pesos asociados iguales a B^{-i}, donde i representa la posición que ocupa el "dígito fraccionario". El dígito más significativo de la parte fraccionaria (el que está pegado a la coma) tiene asignado la posición i=1, el siguiente dígito de la parte fraccionaria, situado a su derecha, la posición i=2, y así sucesivamente. Para distinguir los dígitos de la parte fraccionaria de los de la parte entera, usaremos la expresión a_{-i}. Así, el dígito a_{-1} es el más significativo de la parte fraccionaria y tiene un peso asociado igual a B^{-1}, le sigue el dígito a_{-2} con peso asociado B^{-2}, etcétera.

Ejemplos.

- El número $23,456_{10} = 2 \times 10^1 + 3 \times 10^0 + 4 \times 10^{-1} + 5 \times 10^{-2} + 6 \times 10^{-3}$. El dígito "4" ocupa la posición i=1 de la parte fraccionaria y su peso asociado es 10^{-1} ;el dígito "5", la posición 2 de la parte fraccionaria, y su peso, 10^{-2} y así sucesivamente.
- El número $10,101_2 = 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$

Un número en base B con r bits en su parte entera y s bits en su parte fraccionaria expresa una magnitud igual a la cantidad que resulta de aplicar la expresión 1.7.

$$N_B = \sum_{i=0}^{r-1} a_i \times B^i + \sum_{i=1}^{s} a_{-i} \times B^{-i} \tag{1.7}$$

1.1.2.1. Cambio de base de decimal a binario

Dado un número N en base 10 con parte entera y fraccionaria, se desea encontrar el equivalente en base 2. Para ello se debe seguir el algoritmo que se presenta a continuación.

1. Sea el entero i = 1
2. Sea, E, la parte entera de N y, F, su parte fraccionaria.
3. De N se retira la parte entera y se convierte a binario aplicando los métodos del apartado 1.1.1.3
4. Se multiplica la parte fraccionaria F por 2.
5. El resultado del punto 4 genera un número con una parte entera, que llamaremos a_{-i} y una fraccionaria, C_{-i}
6. Si C_{-i} es distinto de cero, se hace F= C_{-i}, se incrementa i y se repite el punto 4.
7. Si el cociente C_{-i} es igual a cero, el proceso finaliza. El número en base 2 esta formado por el conjunto de los bits a_{-i}, donde el subíndice i indica la posición que ocupa cada bit en el número binario, esto es, el bit a_{-1} es el más significativo de la parte fraccionaria, le sigue el a_{-2} y, así sucesivamente.

Ejemplos.

- $4,23_{10}$

  La parte entera, E, es 100_2

  Para la parte fraccionaria, F:

  $0,23 \times 2 = 0,46 \rightarrow a_{-1} = 0$
  $0,46 \times 2 = 0,92 \rightarrow a_{-2} = 0$
  $0,92 \times 2 = 1,84 \rightarrow a_{-3} = 1$
  $0,84 \times 2 = 1,68 \rightarrow a_{-4} = 1$
  $0,68 \times 2 = 1,36 \rightarrow a_{-5} = 1$
  $\ldots 4,23_{10} = 100,00111\ldots_2$

- $18,0625_{10}$

  La parte entera, E, es 10010_2

  Para la parte fraccionaria, F:

  $0,0625 \times 2 = 0,125 \rightarrow a_{-1} = 0$
  $0,125 \times 2 = 0,25 \rightarrow a_{-2} = 0$
  $0,25 \times 2 = 0,5 \rightarrow a_{-3} = 0$
  $0,5 \times 2 = 1,0 \rightarrow a_{-4} = 1$

  $18,0625_{10} = 10010,0001_2$

- $1,310_{10}$

  La parte entera, E, es 1_2

  Para la parte fraccionaria, F:

  $0,31 \times 2 = 0,62 \rightarrow a_{-1} = 0$
  $0,62 \times 2 = 1,24 \rightarrow a_{-2} = 1$
  $0,24 \times 2 = 0,48 \rightarrow a_{-3} = 0$
  $0,48 \times 2 = 0,96 \rightarrow a_{-4} = 0$
  $0,96 \times 2 = 1,92 \rightarrow a_{-5} = 1$
  $0,92 \times 2 = 1,48 \rightarrow a_{-6} = 1$
  $0,48 \times 2 = 0,96 \rightarrow a_{-7} = 0$
  $0,96 \times 2 = 1,92 \rightarrow a_{-8} = 1$
  $0,92 \times 2 = 1,48 \rightarrow a_{-9} = 1\ldots$

  $1,31_{10} = 1,010011\overbrace{011}_{\ 2}$

Nótese que, aunque un número tenga finitos dígitos fraccionarios en una base, puede tener infinitos dígitos en otra base.

1.1.2.2. Cambio de base de binario a decimal

Se obtiene aplicando la expresión 1.8donde r representa los bits de la parte entera, s, los de la parte fraccionaria y a_i son los dígitos del número binario.

$$N_{10} = \sum_{i=0}^{r-1} a_i \times 2^i + \sum_{i=1}^{s} a_{-i} \times 2^{-i} \tag{1.8}$$

Ejemplo.

$$101,101_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 5,625_{10}$$

1.1.2.3. Cambio de base de decimal a base B

Sigue el mismo algoritmo que para base 2 pero, en este caso, las multiplicaciones se hacen por B. Especial cuidado debe tenerse si B es mayor que diez, puesto que los resultados de algunas multiplicaciones deberán ser convertidos a sus correspondientes dígitos en base B. En el siguiente cuadro se resume el algoritmo a seguir.

1. Sea el entero i = 1
2. Sea E la parte entera de del número N a convertir y, F, su parte fraccionaria.
3. De N se retira la parte entera y se convierte a base B aplicando los métodos del apartado 1.1.1.3
4. Se multiplica la parte fraccionaria F por B.
5. El resultado del punto 4 genera un número con una parte entera, que llamaremos a_{-i} y una fraccionaria, C_{-i}. El número a_{-i} debe estar un dígito válido de la base destino.
6. Si C_{-i} es distinto de cero, se hace F=C_{-i} , se incrementa i y se repite el punto 4.
7. Si el cociente C_{-i} es igual a cero, el proceso finaliza. El número en base B esta formado por el conjunto de los bits a_{-i}, donde el subíndice i indica la posición que ocupa cada bit en el número binario, esto es, el bit a_{-1} es el más significativo de la parte fraccionaria, le sigue el a_{-2} y, así sucesivamente.

Ejemplos.

- $4,23_{10}$ a base 3

  La parte entera, E, es 11_3

  Para la parte fraccionaria, F:

  $0,23 \times 3 = 0,69 \rightarrow a_{-1} = 0$
  $0,69 \times 3 = 2,07 \rightarrow a_{-2} = 2$
  $0,07 \times 3 = 0,21 \rightarrow a_{-3} = 0$
  $0,21 \times 3 = 0,63 \rightarrow a_{-4} = 0$
  $0,63 \times 3 = 1,89 \rightarrow a_{-5} = 1$
  $\ldots 4,23_{10} = 11,02001\ldots_3$

- $4,23_{10}$ a base 12. Use α y β como los dígitos que representan la magnitud diez y once en esta base.

  La parte entera, E, es 4_{12}

  Para la parte fraccionaria, F:

  $0,23 \times 12 = 2,76 \rightarrow a_{-1} = 2$
  $0,76 \times 12 = 9,12 \rightarrow a_{-2} = 9$
  $0,12 \times 12 = 1,44 \rightarrow a_{-3} = 1$
  $0,44 \times 12 = 5,28 \rightarrow a_{-4} = 5$

$0,28 \times 12 = 3,36 \rightarrow a_{-5} = 3$
$\ldots 4,23_{10} = 4,29153\ldots_{12}$

1.1.2.4. Cambio de base B a decimal

Se obtiene aplicando la expresión 1.7

Ejemplos.

- $123,3_5 = 1 \times 5^2 + 2 \times 5^1 + 3 \times 5^0 + 3 \times 5^{-1} = 38,6_{10}$
- $19,36_{12} = 1 \times 12^1 + 5 \times 12^0 + 3 \times 12^{-1} + 6 \times 12^{-2} = 17,2916_{10}$

1.1.2.5. Casos especiales de cambio de base

Aquí se incluyen aquellos en los que las bases destino y fuente de conversión de base están relacionadas de forma que, una, es igual a la potencia entera de la otra. Esto permite realizar rápidamente cambios de base mediante las técnicas de compresión y expansión estudiadas anteriormente. Para los números que poseen parte fraccionaria el método no es diferente.

Para una expansión, todos los dígitos del número original se convierten en los correspondientes de la base final. Se respeta la posición de la coma del número fraccionario.

Para la compresión se debe formar grupos de dígitos. Los de la parte entera se forman desde la coma hacía las posiciones hacia la izquierda del número (las más significativas), y los grupos de la parte fraccionaria, desde la coma hacia la derecha.

Ejemplos.

- $75,12_8 = 111\ \ 101,\ \ 001\ \ 010_2 = 111101,00101_2$
- $250,6_8 = 010\ \ 101\ \ 000,\ \ 110_2 = 10101000,11_2$
- $FA,0C_{16} = 1111\ \ 1010,\ \ 0000\ \ 1100_2 = 11111010,000011_2$
- $2,A98_{16} = 0010,\ \ 1010\ \ 1001\ \ 1000_2 = 10,101010011_2$
- $\$E7,0C = 32\ \ 13,\ \ 00\ \ 30_4 = 3214,003_4$
- $10,100111110_2 = 0010,\ \ 1001\ \ 1111_2 = 2,9F_{16}$
- $10,100111110_2 = 010,\ \ 100\ \ 111\ \ 110_2 = 2,476_8$

1.2. Códigos binarios

Un código es una **colección de símbolos y reglas que permite formular e identificar cierta información. Cuando dicha colección simbólica está formada por agrupaciones de bits, el código se denomina binario.** En la tabla 1.2 se muestra un código binario para la identificación de los diez números decimales y para el que se ha usado la conversión natural decimal-binario presentada en los apartados anteriores.

No obstante, se podrían codificar los números decimales de muchas otras formas distintas. En la tabla 1.3 se muestra otro ejemplo de codificación en la que se percibe que el código usado no es un código pesado. El lector debe entender, simplemente, que ciertas combinaciones de unos y ceros identifican elementos (en este caso dígitos decimales).

Existe infinitas representaciones o códigos de un mismo conjunto de elementos. De hecho, si a cada elemento del conjunto se le asigna una hilera arbitraria de unos y ceros de modo que, de forma unívoca, cada hilera identifique un único elemento del conjunto, se habrá conseguido generar un código binario. Existen múltiples codificaciones dependiendo del número de bits que

Tabla 1.2: Código binario para representar los diez dígitos decimales

Número decimal	Código binario
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Tabla 1.3: Código binario para representar los diez dígitos decimales

Número decimal	Código binario
0	0110000
1	0110001
2	0110010
3	0110011
4	0110100
5	0110101
6	0110110
7	0110111
8	0111000
9	0111001

contenga cada hilera y la asignación que se realice de unos y ceros a cada elemento. No obstante, la arbitrariedad para la generación de códigos binarios no es una realidad. En primer lugar, los elementos de mayor interés de codificación son los dígitos y las letras, los cuales disponen de cierto "orden" (magnitud en el caso de los primeros, y alfabético en el caso de los segundos) que los códigos binarios pueden reproducir de alguna forma. En segundo lugar, el número de bits que se utilice para generar un código binario va a depender, en muchas casos, del entorno para el que se ha desarrollado dicho código. Así, se pueden encontrar códigos binarios para los números decimales de 4 bits, 5 bits, 7 bits, etc, dependiendo de la máquina en la que se utilicen. En cualquier caso, es interesante conocer el mínimo número de bits necesarios para poder codificar un conjunto de elementos. En las tablas 1.2 y 1.3 se ha podido ver dos códigos binarios para los números decimales que disponían de 4 y 7 bits res-

pectivamente. ¿Se podría haber construido un código binario con 1,2 o 3 bits para representar los números decimales?. La respuesta es evidente, con 1 bit, sólo se disponen de 2 posibles combinaciones (un 0 y un 1) por lo que sólo se pueden codificar un conjunto de dos elementos; con dos bits, las combinaciones son 00, 01, 10, 11, o sea, cuatro elementos; con tres bits, las combinaciones son 000, 001, 010,011,100,101,110,111, o sea, ocho. Por tanto, el número mínimo de bits para codificar los dígitos decimales es de cuatro.

En general, un código binario de n bits es capaz de representar un conjunto de 2^n elementos. De forma inversa, para un conjunto de P elementos, el mínimo número n de bits necesarios para codificar dichos elementos debe cumplir la relación 1.9

$$2^{n-1} < P \leq 2^n \tag{1.9}$$

El número de elementos, P, debe ser menor o igual que la potencia 2^n (o número máximo de elementos que se puede codificar con n bits), pero mayor que el número máximo de elementos que se podrían codificar con n-1 bits.

Dado un número P de elementos, los n bits mínimos necesarios para la codificación se pueden obtener mediante la expresión 1.10 en la quela función RE(x) es una función que devuelve el menor valor entero comprendido entre el intervalo real $[x, x+1]$.

$$n = RE[log_2(P)] \tag{1.10}$$

En los siguientes apartados se presentarán algunos de los códigos binarios más importantes

1.2.1. Códigos binarios que representan dígitos decimales

Los códigos que se muestran a continuación permiten representar los diez dígitos decimales.

1.2.1.1. Código BCD (Binary Code for Decimal digits)

Cada elemento del código emplea cuatro bits y utiliza el peso asociado (8,4,2,1). (ver tabla 1.4).

Tabla 1.4: Código BCD

Dígito decimal	Código BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Ejemplos.

$1234_{10} = (0001\ \ 0010\ \ 0011\ \ 0100)_{BCD}$

$709_{10} = (0111\ \ 0000\ \ 1001)_{BCD}$

En los ejemplos se observa que cada dígito decimal se sustituye por su equivalente en el código BCD.

1.2.1.2. Código exceso-3 (Excess-3)

Es un código pesado que utiliza la representación posicional y cuya magnitud resultante es tres unidades mayor que la magnitud del dígito al que representa, de ahí el nombre de exceso tres. Presenta la propiedad de que el Complemento a 9 (Ca9) del dígito representado puede obtenerse, inmediatamente, cambiando los unos por ceros y los ceros por unos en los bits de dicho código. Como se demostrará en futuras secciones, el Ca9 de, por ejemplo, el dígito uno es el ocho. Según se ve en la tabla 1.5, el código exceso 3 del uno es

0100, y del ocho, 1011.

Tabla 1.5: Código Exceso tres

Dígito decimal	Código exceso tres
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

Ejemplos.

$1234_{10} = (0100\ \ 0101\ \ 0110\ \ 0111)_{excess-3}$

$709_{10} = (1010\ \ 0011\ \ 1100)_{excess-3}$

1.2.1.3. Código 2-de-5

Es un código de 5 bits. Presenta la propiedad de tener, para cada elemento del código, dos unos y tres ceros. La posición relativa de los primeros identifica un dígito decimal en concreto. La constancia del número de unos y ceros hace interesante este código para la detección de errores. Un sistema que reciba un dígito decimal codificado en este formato reconocerá si existe un error si el código recibido tiene más de dos unos o menos de dos.

Ejemplos.

$1234_{10} = (00101\ \ 00110\ \ 01001\ \ 01010)_{2-de-5}$

$709_{10} = (10010\ \ 00011\ \ 11000)_{2-de-5}$

Tabla 1.6: Código dos de cinco

Dígito decimal	Código 2-de-5
0	00011
1	00101
2	00110
3	01001
4	01010
5	01100
6	10001
7	10010
8	10100
9	11000

1.2.1.4. Códigos de siete segmentos

Es un código no pesado de siete bits utilizado para la representación de los dígitos decimales en displays o pantallas de siete segmentos (figura 1.3). Estos visualizadores disponen de siete segmentos luminosos que, individualmente, pueden estar encendidos o apagados dependiendo del valor que tomen siete señales de control (una por segmento). Dichas señales de control son: a, b, c, d, e, f, g. Si una de estas señales contiene un uno, el segmento asociado se ilumina, si contiene un cero, el segmento permanece apagado.

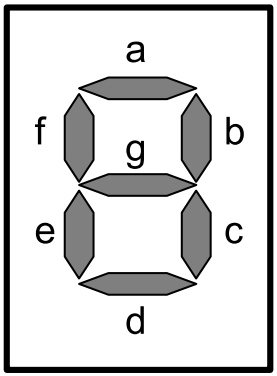
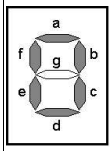
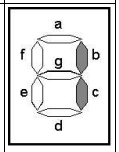
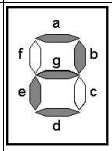
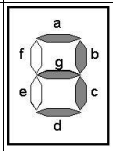
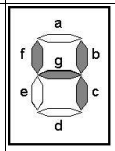
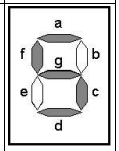
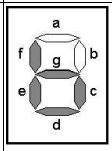
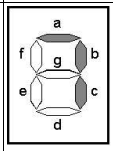
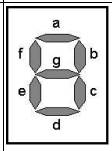
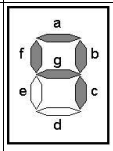


Figura 1.3. Display de 7 segmentos

La tabla 1.7 tiene el código de siete segmentos completo para todos los dígitos decimales.

Tabla 1.7: Código 7 segmentos

Número BCD	Código 7-segmentos **a b c d e f g**		***Número BCD***	Código 7-segmentos **a b c d e f g**	
0000	**1111110**		0001	**0110000**	
0010	**1101101**		0011	**1111001**	
0100	**0110011**		0101	**1011011**	
0110	**0011111**		0111	**1110000**	
1000	**1111111**		1001	**1110011**	

1.2.2. Código Gray

Es un código sin peso asociado que tiene la propiedad de que, entre dos elementos consecutivos del mismo, sólo existe un bit diferente. A diferencia de los códigos anteriores, el código Gray no tiene un número fijo de bits. En la tabla 1.8 se muestra un código Gray de 3 bits, con el que se podría codificar desde el número 0 hasta el 7. Igualmente, existe un código Gray de 4 bits, con el que se pueden codificar los números que van desde el 0 al 15, ambos incluidos, y así sucesivamente.

Tabla 1.8: Código Gray de 3 bits

Número decimal	Código Gray de 3 bits
0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

Se puede comprobar, usando la tabla 1.8, que, al pasar del código que representa el número 0, al del número 1, sólo se ha modificado un bit; lo mismo ocurre al pasar del código del número 1 al del 2 y, así, hasta el código 7 que, hacia el 0, sólo cambia en un bit. Esta propiedad es común para cualquier código Gray de n bits.

El código Gray de n+1 bits se puede construir, a partir del código de n bits, siguiendo un procedimiento que se ejemplarizará para el caso de la obtención del código Gray de 3 bits, conocido el de 2 bits.

El código Gray de 3 bits debe tener 8 elementos: los cuatro primeros son los mismos que los del código Gray de 2 bits a los que se les añade un 0 en la posición más significativa; los siguientes cuatro se obtienen "rotando", a través de un eje imaginario trazado tras el último elemento del código de 2 bits, una copia de los mismos, a los que se les antepone un 1 (ver figura 1.4).

En la tabla 1.9 se muestran los códigos Gray de 1,2,3 y 4 bits, y cómo cada uno de estos se pueden obtener a partir de los anteriores.

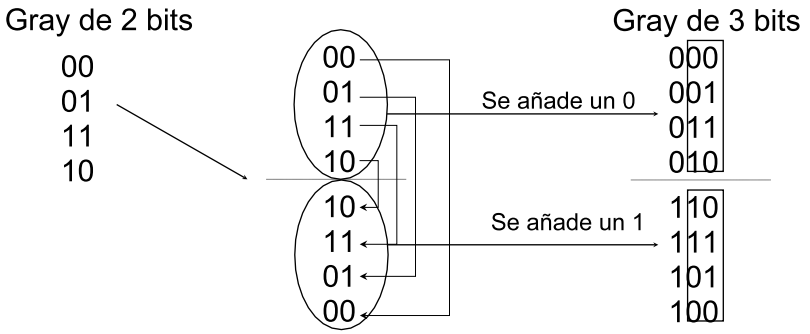


Figura 1.4. Representación de la técnica de obtención de un código Gray de n bits, conocido el de n-1 bits y particularizado para n=3

Tabla 1.9: Códigos Gray de 1, 2, 3 y 4 bits

	Gray de 1 bit	Gray de 2 bits	Gray de 3 bits	Gray de 4 bits
0	0	**00**	**000**	**0000**
1	1	**01**	**001**	**0001**
2		**11**	**011**	**0011**
3		**10**	**010**	**0010**
4			**110**	**0110**
5			**111**	**0111**
6			**101**	**0101**
7			**100**	**0100**
8				**1100**
9				**1101**
10				**1111**
11				**1110**
12				**1010**
13				**1011**
14				**1001**
15				**1000**

1.2.3. Códigos alfanuméricos

Son aquellos que representan letras, números y demás signos de puntuación. Para codificar más de 64 símbolos gráficos (26 letras minúsculas, 26 letras mayúsculas, 10 números y signos de puntuación como interrogantes, admiraciones, comas, puntos, etc) son necesarios siete bits como mínimo.

Uno de los códigos alfanuméricos más usados es el ASCII (American Standard Code for Information Interchange) que puede ser de siete u ocho bits si se le incorpora un bit de paridad.

Tabla 1.10: Códigos ASCII

		$C_6C_5C_4$							
		000	001	010	011	100	101	110	111
	0000	NUL	DEL	SP	0	@	P	'	p
	0001	SOH	DC1	!	1	A	Q	a	q
	0010	STX	DC2	"	2	B	R	b	r
	0011	ETX	DC3	#	3	C	S	c	s
	0100	EOT	DC4	$	4	D	T	d	t
	0101	ENQ	NAK	%	5	E	U	e	u
	0110	ACK	SYN	&	6	F	V	f	v
$C_3C_2C_1C$	0111	BEL	ETB	'	7	G	W	g	w
	1000	BS	CAN	(	8	H	X	h	x
	1001	HT	EM	)	9	I	Y	i	y
	1010	LF	SUB	*	;	J	Z	j	z
	1011	VT	ESC	+	:	K	[	k	{
	1100	FF	FS	´	<	L	\	l	|
	1101	CR	GS	-	=	M	]	m	}
	1110	S0	RS	.	>	N	^	n	~
	1111	S1	US	/	?	O	-	o	DEL

El código ASCII del número 4, según la tabla 1.10 viene dado por $C_{6-0} =$ 011 0100, para la letra A, $C_{6-0} =$ 100 0001, etc.

1.2.4. Códigos detectores de errores

La correcta interpretación de un determinado código (o mensaje) recibido que puede haber sido sometido a perturbaciones, requiere de técnicas de codificación que permitan la identificación de una posible alteración del mensaje

original. Los códigos detectores de errores se construyen a partir de códigos predeterminados a los que se les añade cierta información redundante.

El método más simple de codificación (y por eso poco eficiente -50 %- para errores de 1 bit) es el basado en el bit de paridad. A partir de un código determinado de n bits, se construye el código detector de error por el método del bit de paridad, incorporando, al código original, un bit adicional, denominado bit de paridad, o bit P, que se sitúa a la izquierda del bit más significativo de cada palabra del código de n bits. El código detector de error resultante tiene, entonces, n+1 bits. Existen dos tipos de bit de paridad: bit de paridad par o bit de paridad impar. El primero (Paridad Par), calcula el valor del bit P para que cada combinación del código de n+1 bits posea un número par de unos. El segundo (paridad impar), calcula el bit P para que cada combinación del código de n+1 bits posea un número impar de unos.

Un transmisor que emita un elemento de un determinado código puede añadirle el calculado bit de paridad P y transmitirlos todos conjuntamente. El receptor, a partir de la información recibida, puede determinar, por el cálculo de la paridad de los bits del mensaje, si ha existido un error o no.

Ejemplo: Supóngase el código binario de 4 bits formado por todos los números comprendidos entre 0 y 15. La tabla 1.11 muestra el código detector de error resultante empleando el método del bit de paridad par e impar.

Tabla 1.11: Ejemplo de código detector de errores por el método del bit de paridad

	Bit de paridad par	Bit de paridad Impar
0	**0**0000	**1**0000
1	**1**0001	**0**0001
2	**1**0010	**0**0010
3	**0**0011	**1**0011
4	**1**0100	**0**0100
5	**0**0101	**1**0101
6	**0**0110	**1**0110
7	**1**0111	**0**0111
8	**1**1000	**0**1000
9	**0**1001	**1**1001
10	**0**1010	**1**1010
11	**1**1011	**0**1011
12	**0**1100	**1**1100
13	**1**1101	**0**1101
14	**1**1110	**0**1110
15	**0**1111	**1**1111

1.3. Representación de números con signo

Existen tres notaciones diferentes para la representación de números con signo:

- Notación signo-magnitud (S-M)
- Notación en Complemento a 1(Ca1)
- Notación en Complemento a 2(Ca2)

1.3.1. Notación signo-magnitud

Es la más "humana" de las representaciones numéricas con signo, debido a que, al conjunto de los bits que representa la magnitud del número, se antepone (en la posición más significativa) un bit, denominado bit de signo, que toma el valor de cero para números positivos y el de uno, para los negativos. En la figura 1.5 se muestra la estructura de un número de 8 bits con representación signo-magnitud.

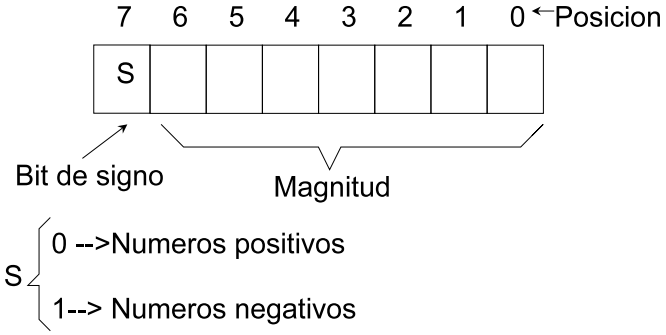


Figura 1.5. Representación S-M para un número de 8 bits

Ejemplos.

+4 usando 4 bits para la magnitud = 00100_{SM}
-4 usando 4 bits para la magnitud = 10100_{SM}

En la tabla 1.12 se presentan todas las combinaciones posibles de números de 4 bits y su interpretación siguiendo la lógica signo-magnitud.

Tabla 1.12: Representación SM con 4 bits

	Notación SM		Notación SM
+0	0000	-0	1000
+1	0001	-1	1001
+2	0010	-2	1010
+3	0011	-3	1011
+4	0100	-4	1100
+5	0101	-5	1101
+6	0110	-6	1110
+7	0111	-7	1111

En general, se puede afirmar que si se utilizan n bits para representar un número A con signo en notación S-M, el rango de valores posibles para A viene dado por la expresión 1.11.

$$-\left(2^{n-1}-1\right) \leq A \leq \left(2^{n-1}-1\right) \tag{1.11}$$

1.3.2. Notación Complemento a 1

Para la comprensión de esta sección se requiere el estudio del anexo 3, en el que se define el operador Ca1.

Los números positivos en notación Ca1 se expresan igual que en SM. Es decir, a los bits de la magnitud se les antepone un bit igual a cero. En cambio, para obtener la representación de un número negativo, primero éste se expresa como si fuera positivo, con bit de signo igual a cero y luego se le aplica el operador Ca1, el resultado obtenido representa el número negativo en esta notación.

Ejemplos.

+4 usando cuatro bits para la magnitud = 00100_{Ca1}

-4 usando cuatro bits para la magnitud = $Ca1(00100) = 11011_{Ca1}$

En la tabla 1.13 se presentan todas las combinaciones posibles de números de 4 bits y su interpretación siguiendo la lógica complemento a uno.

Tabla 1.13: Representación Ca1 con 4 bits

	Notación Ca1		Notación Ca1
+0	0000	-0	1111
+1	0001	-1	1110
+2	0010	-2	1101
+3	0011	-3	1100
+4	0100	-4	1011
+5	0101	-5	1010
+6	0110	-6	1001
+7	0111	-7	1000

Obsérvese que, también para Ca1, el bit más significativo actúa como bit de signo y de idéntica forma que en S-M.

Si se utilizan n bits para representar un número A con signo en notación Ca1, el rango de valores posibles, para A, viene dado por la relación 1.11

1.3.3. Notación Complemento a 2

Para la comprensión de esta sección se requiere el estudio del anexo 3, en el que se define el operador Ca2.

Los números positivos en notación Ca2 se expresan igual que en SM y en Ca1. A los bits de la magnitud se le antepone el bit 0. Los números negativos se obtienen siguiendo el mismo procedimiento que en el Ca1 pero, en este caso, aplicando el operador Ca2.

Ejemplos.

+4 usando cuatro bits para la magnitud = 00100_{Ca1}
-4 usando cuatro bits para la magnitud = $Ca2(00100) = 11100_{Ca1}$

En la tabla 1.14 se presentan todas las combinaciones posibles de números de 4 bits y su interpretación siguiendo la lógica complemento a dos.

Tabla 1.14: Representación Ca2 con 4 bits

	Notación Ca2		Notación Ca2
+0	0000	-0	0000
+1	0001	-1	1111
+2	0010	-2	1110
+3	0011	-3	1101
+4	0100	-4	1100
+5	0101	-5	1011
+6	0110	-6	1010
+7	0111	-7	1001
+8	–	-8	1000

Obsérvese que, también para Ca2, el bit más significativo actúa como bit de signo y de idéntica forma que en S-M y Ca1, aunque, a diferencia de las notaciones anteriores, el número 0 tiene una única codificación. También se observa que existe una codificación adicional, la 1000 asociada al número -8. Para demostrar que el código 1000 se corresponde con el -8 se puede emplear el siguiente razonamiento: si a los bits del código 1000 se les suman[1], los del número +1, es decir, 0001, el resultado es 1001 que corresponde con el número -7, por lo que 1000 representa al -8.

Si se utilizan n bits para representar un número A con signo en notación Ca2, el rango de valores posibles para A está comprendido entre:

$$-2^{n-1} \leq A \leq \left(2^{n-1} - 1\right) \tag{1.12}$$

1.3.4. Comparación entre las notaciones numéricas

Todas las representaciones numéricas (SM, Ca1 y Ca2) necesitan un bit de signo situado en la posición más significativa que toma el valor de 0 para los números positivos, y el de 1 para los negativos. Los números positivos se representan igual en las tres notaciones, sólo cambian para los negativos. La

[1] Aunque no se han tratado las operaciones aritméticas binarias, están no difieren de las reglas decimales conocidas y, por consiguiente, 1000 + 0001 = 1001

notación SM y Ca1 tienen dos codificaciones distintas para un mismo número (+0 y -0), situación, ésta, que no ocurre en Ca2 que dispone, adicionalmente, del número -2^{n-1}, siendo, n, la cantidad de bits usados para representar un número cualquiera en esta notación.

A pesar de que la notación SM parece la más lógica y cercana a nuestra realidad, no existen muchas máquinas digitales en el mundo que incluyan este tipo de representación. Las más extendida es la notación Ca2 seguida por la Ca1. Se demostrará más adelante, en otros capítulos, que las operaciones aritméticas (suma y resta) con números en notación SM requieren más dispositivos electrónicos para su implementación, que las realizadas con números expresados en Ca1 o Ca2.

1.4. Representación binaria de números reales

Los números reales son aquellos que pueden representarse en cualquier punto de la recta real e incluyen los números positivos y negativos que pueden tener parte entera y fraccionaria sin ningún tipo de restricción. En secciones anteriores ya se han mostrado la representación binaria de estos números, por lo que, en este apartado, se realizarán algunas consideraciones adicionales y se introducirá el formato en coma flotante.

1.4.1. Representación en coma fija

Los números binarios, ya sean enteros o fraccionarios, se almacenan en las máquinas digitales en unos dispositivos denominados "registros" los cuales tienen una capacidad finita de almacenamiento. Un registro de n bits almacena un total de n bits como parece evidente. Para almacenar números reales, un segmento del registro se debe destinar a la parte entera y, el resto, a la fraccionaria. En la figura 1.6 se ha representado un registro de 8 bits, constituido por 8 "celdas" que almacenan, cada una de ellas, un bit del número y en las que, las tres menos significativas, (de la 0 a la 2), se han destinado a la parte fraccionaria. Asimismo se ha representado una "ficticia coma" como delimitador de las celdas que almacenan los bits de la parte fraccionaria de los de la parte entera.

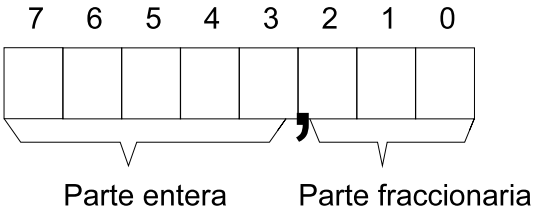


Figura 1.6. Registro de 8 bits. Las tres celdas de la derecha se reservan para la parte fraccionaria y las restantes de la izquierda, a la parte entera

El número $5,5_{10}$ que en binario es $101,1_2$, se almacenaría, en el registro anterior, como 00101100 donde, las celdas que no se usan se rellenan con ceros para no alterar la magnitud del número (ver figura 1.7).

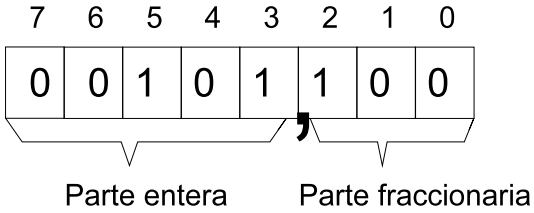


Figura 1.7. Almacenamiento del número $5{,}5_{10}$

Este formato se denomina de **coma fija**, porque la posición de la coma en el registro no cambia y, por tanto, el número de bits de la parte fraccionaria (al igual que el de la parte entera) es constante. Esta rigidez limita las magnitudes fraccionarias y enteras representables. Para el ejemplo de la figura 1.6, los tres bits de la parte fraccionaria permiten almacenar con exactitud a, tan sólo, unas pocas fracciones. En la figura 1.8 se muestra el trozo de recta real comprendida entre 0 y 1. Las líneas verticales gruesas representan los valores fraccionarios que se pueden almacenar con exactitud usando tres bits para la parte fraccionaria. Cualquier otro valor, como por ejemplo, los comprendidos entre $(0,0{,}125)$ o en el intervalo $(0{,}125, 0{,}25)$, etc., no son susceptibles de ser representados con exactitud.

En el formato de coma fija, la única posibilidad de incrementar el número de valores fraccionarios que se puedan representar con exactitud, pasa por aumentar el número de bits, que en el registro, se destinan a la parte fraccionaria. No obstante, se usen el número de bits que se usen, siempre existirán infinitos números fraccionarios que no serán representables con exactitud. En tales casos, se deben definir los criterios que permitan su almacenamiento y que son: el *truncamiento* o el *redondeo*.

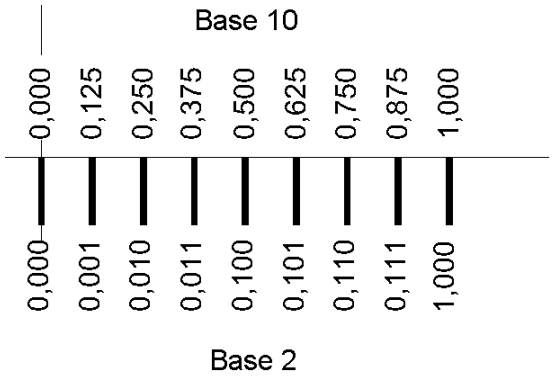


Figura 1.8.

- El truncado elimina aquellos bits de la parte fraccionaria del número que no se pueden almacenar en el registro.

- El redondeo de un número A tiende a almacenar dicho número como si fuera otro número, B, lo más cercano posible a A, que si sea representable con exactitud en el registro.

Como ilustración de estos procedimientos, supóngase que se desea realizar el almacenamiento de la cantidad +3,71875 en notación signo-magnitud, en un registro como el de la figura 1.6. La parte entera, +3, en signo-magnitud con 5 bits es igual a 00011. La parte fraccionaria, 0.71875, en binario es 0,10111, que, tal y como se aprecia, no es posible almacenar en aquel registro. Si se realiza el truncado, la cantidad que se almacenaría sería 00011,101 (los dos bits de la derecha se eliminan; por el método del redondeo, primero, se determinan los números más cercanos a 0,71875 que puedan ser representados con tres bits en la parte fraccionaria (en este caso el 0,75 y el 0,625) y, segundo, de entre ambos números representables, se escoge el que cometa menor error (el más cercano al número). En este ejemplo sería el 0,75 (que en binario sería 0,110) , por consiguiente, el número almacenado: 00011,110.

1.4.2. Representación en coma flotante

Esta representación busca el almacenamiento del número real en forma de exponencial. Cualquier número real, N, en base B puede ser expresado según

1.13

$$N_B = mB^e \tag{1.13}$$

Donde m representa la mantisa del número y, e, el exponente. Así, por ejemplo, el número 2,450 puede expresarse como 245×10^{-2}, siendo la mantisa, $m = 245$, y el exponente, $e = -2$. Téngase en cuenta que, tanto la mantisa como el exponente, se expresan en la base B. El número binario 100,11 se representa, exponencialmente, como 10011×2^{-10}. La mantisa, ahora, es $m = 10011$, y el exponente, $e = -10$ (-2 en decimal).

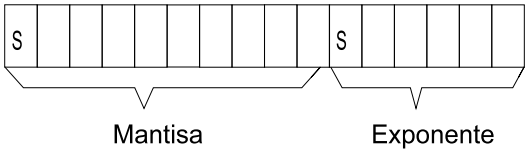


Figura 1.9. Ejemplo de almacenamiento en formato coma flotante

Cualquier número real binario que se almacene con el formato exponencial sólo precisa guardar la mantisa y el exponente, la base se da por conocida. Nótese que para representar números grandes y pequeños, se debe permitir que el exponente tenga signo y si se desea representar números, tanto positivos como negativos, la mantisa también debe tenerlo. En la figura 1.9 se muestra un posible modo de almacenamiento de un número en coma flotante usando 16 bits. Los 6 bits menos significativos se reservan para el exponente, los restantes, para la mantisa y, para ambos, el bit más significativo es el bit de signo. De entre las tres posibles representaciones numéricas con signo existentes con las que se podrían almacenar la mantisa y el exponente, se optará, por su sencillez, por la signo-magnitud.

En la representación por coma flotante, la "coma" del número que se representa no está en una posición concreta, porque ésta depende del valor del exponente almacenado. Incrementando o decrementando el contenido del exponente, la coma del número representado se desplaza hacia la izquierda o hacia la derecha, de aquí el nombre de "coma flotante".

Un inconveniente, a priori, de este tipo de notación numérica, es la multitud de representaciones distintas para un mismo número. Esto se muestra en el siguiente ejemplo: supóngase que se desea representar el número binario +100,1 en notación coma flotante con 8 bits para la mantisa y 4 para el ex-

ponente. El número 100,1 es igual a 1001×2^{-01} (se almacenaría la cantidad 00010011001) , o igual a 10010×2^{-10} (00100101010), o igual a 100100×2^{-11} (01001001011) o igual a $0,0001001 \times 2^{110}$ (01001000110). O sea, existen infinitas formas de representación de un único número real. Para no crear confusión en el modo de interpretación de los números en coma flotante almacenados se utilizan normalizaciones. Estas buscan que el dígito más significativo de la mantisa, el siguiente al bit de signo, sea distinto de cero. Existen dos tipos: **normalización por mantisa entera** y **normalización por mantisa fraccionaria** . La primera busca que el contenido de la mantisa sea un número entero o, dicho de otra forma, que la coma del número real se encuentre detrás del último dígito representable de la mantisa. La segunda, que el contenido sea una fracción o, equivalentemente, que la coma del número real se encuentre inmediatamente delante de la posición del dígito más significativo de la mantisa. En la figura 1.10 se muestra gráficamente estos conceptos.

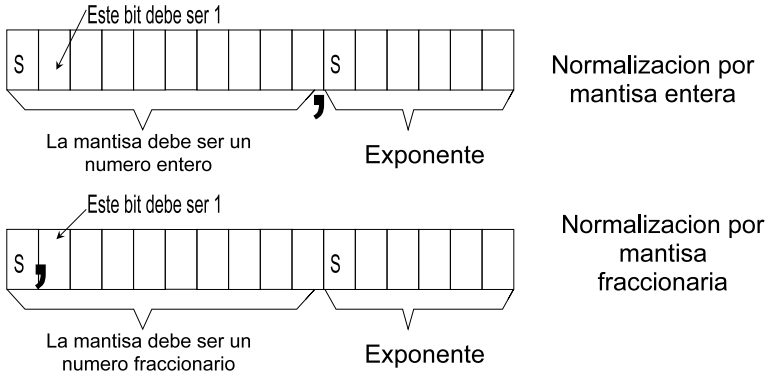


Figura 1.10. Normalización por mantisa entera y fraccionaria para números en coma flotante usando 10 bits para la la mantisa y 6 para el exponente

---

Ejemplos.

Se dispone de 8 bits para la mantisa y 5 para el exponente. Represente el número binario +1000,001 usando a) la normalización por mantisa entera y b) la normalización por mantisa fraccionaria.

a) Se debe obtener una mantisa, m, de 8 bits entera en notación S-M de forma que el bit a continuación del bit de signo, sea distinto de cero. Para el caso que nos ocupa, m=01000001 y el exponente debe ser -3_{10} que, en binario con notación SM, es e=10011.

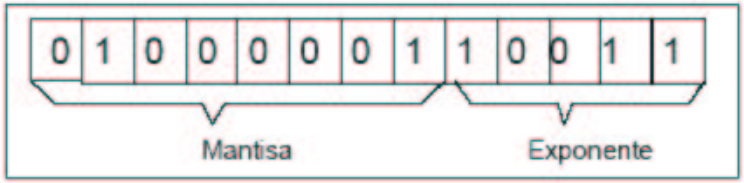


Figura 1.11. Representación del número 1000,001 normalización por mantisa entera

b) Se debe obtener una mantisa , m, también de 8 bits en notación S-M, que represente un valor fraccionario y de forma que, el bit a continuación del de signo, sea distinto de cero. Para este caso, m, sigue siendo 01000001, pero el exponente, e,= $+4_{10}$ o e= 00100, para que el número representado siga siendo el mismo.

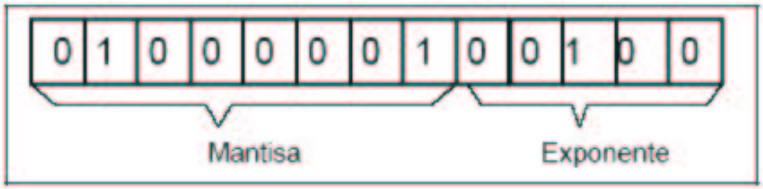


Figura 1.12. Representación del número 1000,001 normalización por mantisa fraccionaria

ANEXO 1

Demostración del método de conversión de la parte entera de un número en decimal a base B

Cualquier número decimal entero M puede expresarse en base B, mediante la expresión 1.4 y convertir el número N en otro expresado en base B, es equivalente a determinar los dígitos a_i de la expresión 1.4.

Como a_i es un dígito que representa un número comprendido entre [0,..,B-1], la cantidad expresada por el primer término de la suma 1.4, $a_0 B_0 = a_0$ es menor que B, por tanto, $(a_0/B < 1)$. El resto de los términos de la suma $a_1 B_1 + \ldots + a_{n-1} B_{n-1}$ representan una magnitud que, salvo el caso que sea 0, es mayor que el valor de la base B. La división del número N, entre la base B, se representa por la expresión 1.14 y genera un cociente entero, $C_0 = \sum_{i=1}^{n-1} a_i \times B^{i-1}$ y un resto igual a a_0.

$$\frac{N}{B} = \frac{a_0 \times B^0 + a_1 \times B^1 + \ldots + a_{n-1} \times B^{n-1}}{B} = \frac{a_0}{B} + a_1 \times B^0 + a_2 \times B^1 + \ldots + a_{n-1} \times B^{n-2} \tag{1.14}$$

Obsérvese que ahora el término $a_1 B_0 = a_1$ es menor que B, y por tanto, no divisible por él. Dividiendo nuevamente C_0 por B, se obtendrá un resto, el dígito a_1 buscado, y un nuevo cociente, C_1. Si el proceso se extiende sucesivamente, los restos obtenidos representan los dígitos del número expresado en base B.

Demostración del método de conversión de la parte fraccionaria de un número en decimal a base B

La parte fraccionaria, F, de un número puede expresarse, usando base B y s dígitos, de la forma dada por la expresión 1.15

$$F = \sum_{i=1}^{s} a_{-i} \times B^{-i} = a_{-1} \times B^{-1} + a_{-2} \times B^{-2} + \ldots + a_{-s} \times B^{-s} \qquad (1.15)$$

Para expresar la magnitud F en base B se necesitan obtener los coeficientes a_{-i} de la expresión 1.15. Si se procede a multiplicar la magnitud F, por la base B, el resultado generado tiene una parte entera, el dígito a_{-1} y una parte fraccionaria formada por la relación $F_0 = \sum_{i=2}^{s} a_{-i} \times B^{-i+1}$.

$$F \times B = a_{-1} + a_{-2} \times B^{-1} + \ldots + a_{-s} \times B^{-s+1} \qquad (1.16)$$

Si al resultado de $F \times B$, se le retira la parte entera, a_{-1}, se obtiene un número fraccionario, F_0. Multiplicando este nuevo número fraccionario por B, se obtendrá un resultado cuya parte entera está formada por el dígito a_{-2}, y otra fraccionaria formada por $F_1 = \sum_{i=3}^{s} a_{-i} \times B^{-i+2}$. Iterando el procedimiento, se obtendrán los dígitos del número F expresados en base B.

ANEXO 2

Demostración de los métodos rápidos de conversión mediante "expansiones y compresiones"

La demostración se llevará a cabo usando dos casos particulares de conversión rápida: de octal a binario y de binario a octal.

Método de "compresión"

Sea C_2 un número en base 2 con n dígitos, y sea D_8 el número, que representado en base 8, expresa la misma magnitud que C_2 en binario ($C_2 = D_8$). La magnitud de C_2 se puede desarrollar según la expresión 1.17

$$C_2 = \sum_{i=0}^{n-1} c_i \times 2^i = \ldots + c_8 2^8 + c_7 2^7 + c_6 2^6 + c_5 2^5 + c_4 2^4 + c_3 2^3 + c_2 2^2 + c_1 2^1 + c_0 2^0 + \ldots \tag{1.17}$$

Para convertir el número binario a la base octal, se deben formar grupos de tres bits, comenzando desde el dígito menos significativo y hacia los más significativos (expresión 1.18).

$$C_2 = \ldots + (c_8 2^8 + c_7 2^7 + c_6 2^6) + (c_5 2^5 + c_4 2^4 + c_3 2^3) + (c_2 2^2 + c_1 2^1 + c_0 2^0) \tag{1.18}$$

Más convenientemente, la expresión 1.18 se puede expresar de la forma dada por la relación 1.19, en la que se ha factorizado por el término de peso más pequeño de cada grupo de 3 dígitos.

$$C_2 = \ldots + (c_8 2^2 + c_7 2^1 + c_6 2^0) 2^6 + (c_5 2^2 + c_4 2^1 + c_3 2^0) 2^3 + (c_2 2^2 + c_1 2^1 + c_0 2^0) 2^0 \tag{1.19}$$

Como $2^0 = 1 = 8^0$, $2^3 = 8^1$, $2^6 = 8^2$, y así sucesivamente, entonces la relación 1.19 puede expresarse según la relación 1.20

$$C_2 = \ldots + (c_8 2^2 + c_7 2^1 + c_6 2^0)8^2 + (c_5 2^2 + c_4 2^1 + c_3 2^0)8^1 + (c_2 2^2 + c_1 2^1 + c_0 2^0)8^0 \tag{1.20}$$

Las cantidades encerradas entre paréntesis en la expresión 1.20 representan magnitudes comprendidas entre 0 y 7 para cualquier combinación de los bits c_i, justo los valores posibles de un dígito en base octal. Sea d_i el dígito que ocupa la posición i del número D expresado en octal. Entonces, $d_0 = a_2 2^2 + a_1 2^1 + a_0 2^0$, $d_1 = a_5 2^2 + a_4 2^1 + a_3 2^0$, $d_2 = a_8 2^2 + a_7 2^1 + a_6 2^0$ y así sucesivamente.

$$C_2 = D_8 = \ldots + (d_2)8^2 + (d_1)8^1 + (d_0)8^0 \tag{1.21}$$

Método de "expansión"

Sea D_8 un número expresado en base 8, con n dígitos. Se desea encontrar el equivalente, C_2, en base dos. El número D_8, se puede desarrollar según la expresión 1.22, donde d_i son los dígitos en base ocho de la magnitud D.

$$D_8 = \sum_{i=0}^{n-1} a_i \times B^i = \ldots + d_2 8^2 + d_1 8^1 + d_0 8^0 \tag{1.22}$$

Cada dígito octal representa un magnitud entre cero y siete que, a su vez, se pueden representar en una base de numeración distinta como, por ejemplo, la binaria. Para la base dos, se necesitarían tres bits para codificar cualquier magnitud de un dígito octal. Sean c_{3i+2}, c_{3i+1} y c_{3i} los dígitos binarios que expresan la magnitud del dígito octal d_i. Sustituyéndolos en la expresión 1.22 nos da la relación 1.23.

$$\begin{aligned} D_8 = \ldots + (c_8 2^2 + c_7 2^1 + c_6 2^0)8^2 + \\ +(c_5 2^2 + c_4 2^1 + c_3 2^0)8^1 + (c_2 2^2 + c_1 2^1 + c_0 2^0)8^0 \end{aligned} \tag{1.23}$$

Como $8^0 = 2^0$, $8^1 = 2^3$, etcétera, sustituyendo en 1.23 y desarrollando los productos, se obtiene la expresión 1.24 que representa el número en binario.

$$C_2 = \ldots + c_8 2^8 + c_7 2^7 + c_6 2^6 + c_5 2^5 + c_4 2^4 + c_3 2^3 + c_2 2^2 + c_1 2^1 + c_0 2^0 + \ldots \quad (1.24)$$

ANEXO 3

El operador unario Complemento

Se define el operador complemento a N de un número M expresado en base N con p dígitos en su parte entera y se representa como $CaN(M_N)$, como la transformación que genera el siguiente resultado

$$CaN(M_N) = N^p - M_N$$

Ejemplos.

- $Ca10(1234_{10}) = 10^4 - 1234_{10} = 10000 - 1234_{10} = 8766_{10}$
- $Ca10(220{,}23_{10}) = 10^3 - 220{,}23_{10} = 10000 - 220{,}23_{10} = 779{,}77_{10}$
- $Ca10(11000_{10}) = 10^5 - 11000_{10} = 100000 - 11000_{10} = 89000_{10}$

Se define el operador complemento a N-1 de un número M expresado en base N con p dígitos en su parte entera y q en la parte fraccionaria y se representa como $CaN-1(M_N)$, como la transformación que genera el siguiente resultado

$$CaN-1(M_N) = N^p - N^{-q} - M_N$$

Ejemplos.

- $Ca9(1234_{10}) = 10^4 - 10^{-0} - 1234_{10} = 10000 - 1 - 1234{-}_{10} = 9999 - 1234_{10} = 8765_{10}$
- $Ca9(220{,}23_{10}) = 10^3 - 10^{-2} - 220{,}23_{10} = 10000 - 0{,}01 - 220{,}23_{10} = 9999{,}99 - 220{,}23_{10} = 779{,}76_{10}$
- $Ca9(11000_{10}) = 10^5 - 1 - 11000_{10} = 99999 - 11000_{10} = 88999_{10}$

La definición del operador complemento a N o del operador complemento a N-1 lleva implícita la realización de la operación aritmética de la resta. En los

ejemplos anteriores se ha podido obtener los valores que generan los operadores Ca10 y Ca9 porque las operaciones aritméticas en base 10 son conocidas. La definición de un operador CaN o CaN-1 para una base N distinta a la base 10 requiere que las operaciones aritméticas que se apliquen cumplan con los reglas de la base N en que se desarrollan. Un caso ilustrativo puede ser la obtención del Ca9 y del Ca8 de un número expresado en base 9. En dicho ejemplo se han destacado los resultados de las operaciones aritméticas que más pueden sorprender.

- $Ca9(1234_9) = 9^4 - 1234_9 = \underline{10000_9} - 1234_9 = \underline{7655_9}$
- $Ca8(1234_9) = 9^4 - 1 - 1234_9 = \underline{8888_9} - 1234_9 = 7654_9$

Es lógico que, en base 9, la cantidad 9^4 sea igual a 10000. De hecho, si partimos de la representación 10000_9 es más fácil ver que la magnitud representada es $1 \times 9^4 + 0 \times 9^3 + 0 \times 9^2 + 0 \times 9^1 + 0 \times 9^0 = 9^4$. Por último queda tratar el caso de la operación de resta en base N.

Para la operación de resta entre dos dígitos en base N, se pueden distinguir dos casos: que el minuendo sea mayor o igual que el substraendo, o que sea menor. En el primer caso se opera de forma normal, el resultado expresa una magnitud igual a la diferencia entre las magnitudes del minuendo y el substraendo. El segundo caso es más complejo: El resultado de la resta es igual a la suma del minuendo con el resultado de la diferencia entre la base N y el substraendo. Además se genera un acarreo.

Ejemplos.

- $7_9 - 1_9 = 6_9$
- $1_9 - 7_9 = 1_9 + (9 - 7_9) = 1_9 + 2_9 = 3_9$ y se genera un acarreo
- $1_{10} - 7_{10} = 1_{10} + (10 - 7_{10}) = 1_{10} + 3_{10} = 4_{10}$ y se genera un acarreo

---

En el caso en que las magnitudes a restar contengan más de un dígito, los posibles acarreos que se generen en los dígitos deben sumarse a los substraendos contiguos a los que generaron dicho acarreo (al igual que ocurre en base 10). De aquí se pude deducir, pues, que $10000_9 - 1234_9 = 7655_9$.

Supóngase que se dispone de un número M expresado en base N con p dígitos (se supone también que M expresa una cantidad entera, aunque los resultados que se obtendrán son aplicables también a números con parte fraccionaria). El operador complemento a N de dicho número lo transforma en otro resultante de la operación aritmética de resta de la cantidad N^p con M_N. La cantidad N^p siempre resulta en un número con p+1 bits, en el que los p bits menos significativos son 0, y el más significativo es un 1. Por otro lado, la magnitud M, por definición, sólo dispone de p bits. Sean $d_{p-1}, ...d_1, d_0$, los p dígitos del número M, y $r_{p-1}, ...r_1, r_0$, los dígitos del resultado de aplicar el operador CaN al número M. Si d_0 es igual a 0, entonces $r_0 = 0-0 = 0$ y no se genera acarreo, Cy, hacia el dígito d_1 (Cy=0). Si d_0 es distinto de 0, entonces $r_0 = (N - d_0)$ y se tiene que Cy=1 que se suma a d_1. Continuando con este último caso, el siguiente bit del resultado, r_1, sería igual a $N - (d_1 + Cy) = N - 1 - d_1$ que, también, genera un acarreo hacia el siguiente dígito d_2. Para el resto de los dígitos se procede de forma idéntica. Obsérvese que la resta del último dígito del número M, d_{p-1} genera un acarreo que se anula con el bit p+1 de la cantidad N^p por lo que el resultado tiene p bits. En el caso en que el dígito d_0 hubiese sido 0, el resultado r_0 valdría 0 y no se generaría acarreo. Siguiendo con el dígito d_1, repetimos el proceso: si $d_1 = 0$, el resultado r_1 sigue siendo cero y, si no, $r_1 = N - d_1$. A partir de aquí, los dígitos r_i restantes se determinan mediante $r_i = N - 1 - d_i$

Ejemplos.

- $Ca10(1234_{10}) = 8766_{10}$
- $Ca8(220{,}23_8) = 557{,}55_8$
- $Ca4(11000_4) = 23000_4$
- $Ca2(11000_2) = 01000_2$
- $Ca2(10110_2) = 01010_2$

En el siguiente cuadro se ha representado el algoritmo que resume el modo de proceder para la obtención del CaN

1. Sea i=0, pdd0=0. (pdd0 significa primer dígito distinto de cero)
2. Si pdd0 = 0 y $d_i = 0$, entonces $r_i = 0$
   Si pdd0=0 y $d_i \neq 0$, entonces $r_i = N - d_i$ y pdd0=1
   Si pdd0=1 o $d_i \neq 0$, entonces $r_i = N - 1 - d_i$
3. i=i+1
4. Repetir 2 y 3 hasta que i sea igual a p

Supóngase que se dispone de un número M en base N, con p dígitos en la parte entera y q en la parte fraccionaria. El operador complemento a N-1 de dicho número genera otro resultante de la operación aritmética de resta de la cantidad $N^p - N^{-q}$ con M. Dicha cantidad, $N^p - N^{-q}$, resulta en un número con p bits en la parte entera, y q en la parte fraccionaria, en la que todos los dígitos toman el valor N-1.

Ejemplos.

- Para N=10, p=4 y q=1, la cantidad $N^p - N^{-q} = 10^4 - 10^{-1} = 10000 - 0{,}1 = 9999{,}9$
- Para N=9, p=4 y q=0, la cantidad $9^4 - 9^{-0} = 10000_9 - 1 = 8888_9$
- Para N=2, p=5 y q=0, la cantidad $2^4 - 2^{-0} = 100000_2 - 1 = 11111_2$

Sea $q = 0$, es decir, el número M no tiene parte fraccionaria y sean d_{p-1},..., d_1, d_0, los p dígitos del número M, y $r_{p-1}, ...r_1, r_0$ los dígitos del resultado de aplicar el operador CaN-1 al número M. El bit r_i del resultado se obtiene, de forma general para todos los dígitos ($\forall i$), como $N - 1 - d_i$.

Ejemplos.

- $Ca9(1234_{10}) = 8765_{10}$
- $Ca7(220{,}23_8) = 557{,}54_8$
- $Ca3(11000_4) = 22333_4$

- $Ca1(11000_2) = 00111_2$
- $Ca1(10110_2) = 01001_2$

En el siguiente cuadro se ha representado el algoritmo que resume el modo de proceder para la obtención del CaN-1

1. Sea i=0
2. $r_i = N - 1 - d_i$
3. i=i+1
4. Repetir 2 y 3 hasta que i sea igual a p

De mayor interés que la multitud de operadores CaN y $CaN - 1$ existentes para cualquier base N, son aquellos en que N=2, es decir, el operador Ca2 y Ca1 para números o magnitudes binarias. A continuación se muestran ejemplos de este tipo de operadores.

Ejemplos.

- $Ca2(11000_2) = 01000_2$
- $Ca2(10111_2) = 01001_2$
- $Ca2(10000_2) = 10000_2$
- $Ca1(11110_2) = 00001_2$
- $Ca1(10010_2) = 01101_2$
- $Ca1(10000_2) = 01111_2$

Se puede observar que el resultado del operador Ca1 se puede obtener directamente del número original solamente invirtiendo los bits, esto significa que, en las posiciones en la que el número tiene un 1, el resultado del operador tiene un 0, y en las que el número tiene un 0, el resultado tiene un 1.

Para el Ca2 el procedimiento parece algo más complejo. Empezando por el bit menos significativo del número, y procesando en cada paso el contiguo a su izquierda, el correspondiente bit del resultado es igual al del número, hasta el primer 1 (incluido este), a partir de este, los bits del resultado se invierten con respecto a los del número.

Propiedades de los operadores CaN y CaN-1

1. El complemento a N del complemento a N de un número M, es el propio número M. (Idém para el complemento a N-1)

$$CaN(CaN(M_N)) = M_N \tag{1.25}$$
$$CaN-1(CaN-1(M_N)) = M_N \tag{1.26}$$

Demostración:

Si CaN (M_N) = N^p - M^N, entonces CaN(CaN(M_N)) = CaN(N^p - M_N) = N^p - (N^p - M_N) = M_N

De idéntica forma se comprueba para el operador CaN-1().

Ejemplos.

- $Ca10(Ca10(1234_{10}) = Ca10(8796_{10}) = 1234_{10}$
- $Ca2(Ca2(1100_2) = Ca2(0100_2) = 1100_2$

2.

$$CaN-1(M_N) = CaN(M_N) - N^{-q} \tag{1.27}$$
$$CaN(M_N) = CaN-1(M_N) + N^{-q} \tag{1.28}$$

donde q es el número de dígitos de la parte fraccionaria del número M.

Demostración:

Como CaN ($M_N) = N^p - M_N$ entonces CaN-1 (M_N) = N^p -$N^{-q} - M_N$ = $(N^p - M_N) - N^{-q}$ = CaN(M_N) - N^-q

Si M no tiene parte fraccionaria, esto es, M es un número entero, q=0, las relaciones 1.28 se reducen a $CaN - 1(M_N) = CaN(M_N) - 1$

ó $CaN(M_N) = CaN - 1(M_N) + 1$. Estas expresiones nos permiten encontrar el resultado generado por un operador, a partir del otro, sin más que añadir o restar una unidad.

Ejemplos.

- $Ca10(1234_{10}) = 8766_{10} \rightarrow Ca9(1234_{10}) = 8766_{10} - 1 = 8765_{10}$
- $Ca10(220{,}23_{10}) = 779{,}77_{10} \rightarrow Ca9(220{,}23_{10}) = 220{,}23_{10} - 0{,}01 = 220{,}22_{10}$
- $Ca9(11000_{10}) = 88999_{10} \rightarrow Ca10(11000_{10}) = 88999_{10} + 1 = 89000_{10}$

Ejercicios propuestos

Problema 1.1 Represente en base 2,4,5,8,16 las siguientes magnitudes: (a) 17,1 ;(b) 3,45 y (c) 120,1

Problema 1.2 La ecuación de primer grado $5x + 122 = 244$ tiene su solución para $x = 14$. Determine en qué base de representación numérica está expresada dicha ecuación.

Problema 1.3 Expresar $\pm 27,3$ en las notaciones S-M, Ca1 y Ca2, reservando 6 bits para la parte entera y 3 bits para la parte fraccionaria. Muestre los resultados correspondientes a los casos anteriores a) truncando, y b) redondeando.

Problema 1.4 Considere la palabra 10111001. Interprete, si es posible, la información de dicha palabra según sea: número binario, representación signo-magnitud, representación en Ca1, representación en Ca2, código ASCII, código ASCCI con paridad par, código ASCII con paridad impar y código BCD.

Problema 1.5 Obtenga la lista de elementos de un código Gray de 6 bits. De dicha lista que código representa, si es posible, a los números 10, 45, 62 y 71.

Problema 1.6 Obtenga el Ca3, si es posible, de los siguientes números: 213_4, 120_3, 1450_9.

Problema 1.7 Representar el número $0,17_{10}$ en coma flotante con normalización por mantisa entera (redondeando si es necesario), reservando 8 bits para la mantisa y 4 para el exponente.

Problema 1.8 Representar $0,108_{10}$ en coma flotante, usando 9 bits para la mantisa, 5 bits para el exponente (incluyendo en ambos casos los bits de signo) y normalización por mantisa fraccionaria.

Problema 1.9

- Se quiere almacenar la cantidad -123×10^{-20} en octal codificado binario usando notación en coma flotante. Utilice el menor número de bits necesarios para la mantisa y el exponente.
- Represente la cantidad $-1,73 \times 10^{-3}$ en decimal codificado en binario usando la notación en coma flotante mediante normalización (1) entera

y (2) fraccionaria (en ambos casos considere 13 bits para la mantisa y 9 para el exponente incluyendo el bit de signo).

Capítulo 2

Álgebra de Conmutación

2.1. Postulados del Álgebra de Boole

El Álgebra de Boole es una estructura algebraica definida por dos operadores binarios: operador CRUZ ($+$) y operador PUNTO ($\bullet$) de tal forma que satisfacen los siguientes postulados:

1. **Postulado del cierre**

   (a) Si x, y son dos elementos del álgebra de Boole, entonces, el resultado obtenido de aplicar el operador CRUZ a ambos elementos también pertenece al álgebra.

   $$\text{Si } x, y \in B \text{ entonces } x + y \in B$$

   (b) Si x, y son dos elementos del álgebra de Boole, entonces, el resultado obtenido de aplicar el operador PUNTO a ambos elementos también pertenece al álgebra.

   $$\text{Si } x, y \in B \text{ entonces } x \bullet y \in B$$

2. **Postulado de los elementos identidad**

   (a) Un elemento de identidad con respecto al operador $+$ es designado por el símbolo 0 y cumple

$$x + 0 = 0 + x = x \text{ donde } x \in B$$

(b) Un elemento de identidad con respecto al operador $\bullet$ es designado por el símbolo 1 y cumple

$$x \bullet 1 = 1 \bullet x = x \text{ donde } x \in B$$

3. **Propiedad conmutativa**

   (a) Conmutatividad con respecto al operador $+$

   $$x + y = y + x$$

   (b) Conmutatividad con respecto al operador $\bullet$

   $$x \bullet y = y \bullet x$$

4. **Propiedad distributiva**

   (a) Distributividad con respecto al operador $+$

   $$x \bullet (y + z) = x \bullet y + x \bullet z$$

   (b) Distributividad con respecto al operador $\bullet$

   $$x + (y \bullet z) = (x + y) \bullet (x + z)$$

5. **Postulado del complemento** Para cada elemento $x \in B$, existe un elemento $\overline{x} \in B$(llamado complemento de x) tal que:

   (a) $x + \overline{x} = 1$

   (b) $x \bullet \overline{x} = 0$

6. Existen al menos dos elementos $x, y \in B$ de tal forma que $x \neq y$

En este álgebra, el significado de los operadores $+$,$\bullet$ son distintos de la aritmética clásica. Llama la atención el hecho de que aparezca la propiedad distributiva del operador $+$ sobre el $\bullet$. También, en los postulados no aparecen los inversos de los operadores $+$ y $\bullet$ y, sin embargo, se dispone de un operador nuevo como es el complemento. Los postulados del álgebra han sido listados a pares, parte (a) y parte (b). Una parte puede obtenerse a partir de la otra mediante el intercambio de los elementos unitarios (0 y 1) y los operadores binarios($+$ y $\bullet$). Esto se conoce como el **Principio de Dualidad**.

2.2. Álgebra de Boole bivalente o álgebra de conmutación

El álgebra de conmutación se obtiene haciendo que el conjunto B de elementos sean sólo dos, el 0 y el 1, y definiendo los operadores binarios $+$ (también llamado operador OR o suma lógica), y $\bullet$ (también llamado operador AND o producto lógico) y el operador complemento, de la forma presentada en las tablas 2.1 y 2.2.

Tabla 2.1: Operador NOT

x	$\overline{x}$
0	1
1	0

Tabla 2.2: Operador AND y OR

x	y	$x \bullet y$	$x + y$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

A continuación se demostrará que la estructura que se ha definido como álgebra de conmutación, cumple con los postulados del álgebra de Boole.

- *Cumplimiento del postulado P1.* El conjunto B formado, sólo, por los elementos 0 y 1 es cerrado. En la tabla 2.2 se puede ver que el resultado de $x + y$ o el de $x \bullet y$ es, también, 0 o 1.

- *Cumplimiento del postulado P2.* Se puede demostrar la igualdad $x + 0 = x$ escogiendo las filas de la tabla 2.2 en las que y sea 0. Se puede observar que el resultado $x + y$ para dichas filas coincide con el valor de x. De igual forma, para las filas en las que $y = 1$ en la tabla 2.2, se obtiene que $x \bullet 1 = x$.

- *Cumplimiento del postulado P3.* Si se intercambian las posiciones de las columnas x e y de la tabla 2.2 y se ordenan los valores asignados a los elementos y, x del mismo modo que el mostrado para los elementos x, y, se obtendrían los mismos resultados para las columnas $x + y$ e $x \bullet y$ y se probaría la propiedad conmutativa.

- *Cumplimiento del postulado P4.* Las propiedades distributivas se demostrarán mediante el uso de tablas. La tabla 2.3 demuestra el cumplimiento del postulado P4a. La primera columna contiene todas las combinaciones posibles asignadas a los elementos x, y, z, la segunda, tercera y quinta columnas contienen los resultados parciales de la operaciones $x \bullet y$, $x \bullet z$ e $y + z$ respectivamente. La cuarta y sexta columnas representan el segundo y primer miembros del postulado P4a y se comprueban que son iguales.

Tabla 2.3: Propiedad distributiva del operador AND sobre el operador OR

$x\ y\ z$	$x \bullet y$	$x \bullet z$	$x \bullet y + x \bullet z$	$y + z$	$x \bullet (y + z)$
000	0	0	0	0	0
001	0	0	0	1	0
010	0	0	0	1	0
011	0	0	0	1	0
100	0	0	0	0	0
101	0	1	1	1	1
110	1	0	1	1	1
111	1	1	1	1	1

Se procede de forma idéntica con el postulado P4b cuya demostración se muestra en la tabla 2.4.

Tabla 2.4: Propiedad distributiva del operador OR sobre el operador AND

$x\ y\ z$	$x + y$	$x + z$	$(x + y) \bullet (x + z)$	$y \bullet z$	$x + (y \bullet z)$
000	0	0	0	0	0
001	0	1	0	0	0
010	1	0	0	0	0
011	1	1	1	1	1
100	1	1	1	0	1
101	1	1	1	0	1
110	1	1	1	0	1
111	1	1	1	1	1

- *Cumplimiento del postulado P5.* De la definición del operador complemento (tabla 2.1) se pueden obtener los resultados que se recogen en la tabla 2.5 y que demuestran el postulado P5.

Tabla 2.5: Operador NOT

x	$\overline{x}$	$\overline{x} + x$	$\overline{x} \bullet x$
0	1	1	0
1	0	1	0

- *Cumplimiento del postulado P6.* El álgebra de conmutación ha definido dos elementos distintos, el 0 y el 1, que son los elementos identidad respecto al operador OR y AND respectivamente.

Además de los seis postulados, el álgebra de conmutación se acompaña de siete teoremas que se detallan y desarrollan a continuación. Nótese que los teoremas, que se construyen a partir de los postulados, heredan el principio de dualidad de estos y, por tanto, se listan en pares (a) y (b).

2.2.1. Teoremas del Álgebra de Conmutación

1. **Teorema de Idempotencia**
   - **(a)** $x + x = x$
   - **(b)** $x \bullet x = x$
2. **Teorema de los Elementos Dominantes**
   - **(a)** $x + 1 = 1$
   - **(b)** $x \bullet 0 = 0$
3. **Teorema Involutivo**
   $\overline{(\overline{x})} = x$
4. **Teorema de Absorción**
   - **(a)** $x + (x \bullet y) = x$
   - **(b)** $x \bullet (x + y) = x$
5. **Teorema del Consenso**
   - **(a)** $x + (\overline{x} \bullet y) = x + y$
   - **(b)** $x \bullet (\overline{x} + y) = x \bullet y$

6. **Teorema Asociativo**

   (a) $x + (y + z) = (x + y) + z$

   (b) $x \bullet (y \bullet z) = (x \bullet y) \bullet z$

7. **Leyes de Morgan**

   (a) $\overline{x + y} = \overline{x} \bullet \overline{y}$

   (b) $\overline{x \bullet y} = \overline{x} + \overline{y}$

   Leyes de Morgan generalizadas

   (a) $\overline{x + y + z \ldots} = \overline{x} \bullet \overline{y} \bullet \overline{z} \bullet \ldots$

   (b) $\overline{x \bullet y \bullet z \ldots} = \overline{x} + \overline{y} + \overline{z} + \ldots$

2.2.2. Demostraciones de los teoremas del álgebra de conmutación

Las demostraciones de los teoremas se pueden llevar a cabo de manera algebraica o mediante el uso de tablas. Las primeras tratan de obtener un miembro de la igualdad del enunciado del teorema aplicando, los postulados y teoremas que se hayan demostrado previamente, al otro miembro de la igualdad. Alternativamente, las tablas permiten verificar el teorema mediante la evaluación independiente de los dos miembros de la igualdad para toda asignación lógica de los elementos que lo componen.

Para la demostración de los teoremas del álgebra de conmutación se ha optado por una fórmula intermedia, por la que, algunos teoremas serán demostrados algebraicamente y, otros, por métodos tabulares. Se deja al lector, como ejercicio, la demostración tabular y algebraica de todos los teoremas.

- **Demostración del teorema T1**

  (a) $x + x = x$

  1. Por el postulado P2a aplicado al segundo miembro de la igualdad:
     $x = x + 0$.
  2. Se aplica el postulado P5b al elemento neutro 0 $x = x + 0 = x + x \bullet \overline{x}$

3. Se aplica la propiedad distributiva del operador $+$ sobre el operador $\bullet$ (postulado P4b). $x = x+0 = x+x\bullet\overline{x} = (x+x)\bullet(x+\overline{x})$
4. Por el postulado del complemento P5a $x = x+0 = x+x\bullet\overline{x} = (x+x)\bullet(x+\overline{x}) = (x+x)\bullet 1$
5. Finalmente, por el postulado P2b $x = x+0 = x+x\bullet\overline{x} = (x+x)\bullet(x+\overline{x}) = (x+x)\bullet 1 = x+x$

(b) $x \bullet x = x$

1. Por el postulado P2b $x = x \bullet 1$.
2. Aplicando el postulado P5a al elemento neutro 1 $x = x \bullet 1 = x \bullet (x+\overline{x})$
3. La propiedad distributiva del operador $\bullet$ sobre el operador $+$ (postulado P4a) nos permite transformar la expresión anterior en: $x = x \bullet 1 = x \bullet (x+\overline{x}) = x \bullet x + x \bullet \overline{x}$
4. Por el postulado del complemento P5a $x = x\bullet 1 = x\bullet(x+\overline{x}) = x \bullet x + x \bullet \overline{x} = x \bullet x + 0$
5. Y, finalmente, por el postulado P2a $x = x+0 = x+x\bullet\overline{x} = (x+x)\bullet(x+\overline{x}) = (x+x)\bullet 1 = x+x$

Obsérvese que para demostrar el apartado (b) de este teorema se han utilizado los postulados duales de los empleados para la demostración del apartado (a). De hecho, si los postulados presentan el principio de dualidad, lo teoremas, que se construyen a partir de los primeros, heredan esta característica. Con objeto de no fatigar al lector con excesivas demostracioens, a continuación se procederá a demostrar, sólamente, el apartado (a) de los restantes teoremas; el apartado (b), quedará automáticamente demostrado por efecto del principio de dualidad.

- **Demostración del teorema T2**

(a) $x + 1 = 1$

1. Se aplica el postulado P5a sobre el elemento neutro del primer miembro de la igualdad.
   $x + 1 = x + (x + \overline{x})$
2. Se aplica el teorema T6a, cuya validez se asumirá en este punto
   $x + 1 = (x + x) + \overline{x}$
3. Se aplica el teorema T1a $x + 1 = (x + x) + \overline{x} = x + \overline{x}$
4. Finalmente se usa el postulado P5a
   $x + 1 = (x + x) + \overline{x} = x + \overline{x} = 1$

(b) $x \bullet 0 = 0$ Queda demostrado por dualidad.

- **Demostración del teorema T3**

  $\overline{\overline{x}} = x$

  Para este teorema se utilizará la demostración por el método tabular. Como se observa en la tabla 2.6 se han utilizado tres columnas de las cuales, la primera y tercera, se corresponden con los dos miembros del teorema. Como el único elemento que aparece en el teorema es el x, tan sólo se necesitan dos filas en la tabla para recoger todas las combinaciones lógicas posibles que puede tomar dicho elemento. La validez del teorema qued demostrada por el hecho de que las columnas exteriores de la tabla 2.6 son iguales.

  Tabla 2.6: Demostración tabular del teorema T3

x	$\overline{x}$	$\overline{\overline{x}}$
0	1	**0**
1	0	**1**

- **Demostración del teorema T4**

  (a) $x + x \bullet y = x$

  1. Se aplica el postulado P2b al elemento x del primer miembro de la igualdad $x + y \bullet x = x \bullet 1 + x \bullet y$
  2. Ahora la propiedad distributiva o P4a. $x + y \bullet x = x \bullet 1 + x \bullet y = x \bullet (1 + y)$
  3. Según el teorema T2a $x + y \bullet x = x \bullet 1 + x \bullet y = x \bullet (1 + y) = x \bullet 1$
  4. Finalmente, por el postulado P2b $x + y \bullet x = x \bullet 1 + x \bullet y = x \bullet (1 + y) = x \bullet 1 = x$

  La demostración tabular del teorema requiere de una tabla como la 2.7 donde se han utilizado un conjunto de columnas intermedias necesarias para la evaluación del primer miembro de la igualdad. Obsérvese que se necesitan cuatro filas para recoger todos los posibles valores lógicos asignados a los elementos x e y

  (b) $x \bullet (x + y) = x$ Queda demostrado por dualidad.

- **Demostración del teorema T5**

Tabla 2.7: Demostración tabular del teorema T4

x	y	$y \bullet x$	$x + y \bullet x$
0	0	0	**0**
0	1	0	**0**
1	0	0	**1**
1	1	1	**1**

Tabla 2.8: Demostración tabular del teorema T5

x	y	$x + y$	$\overline{x}$	$\overline{x} \bullet y$	$x + \overline{x} \bullet y$
0	0	**0**	1	0	**0**
0	1	**1**	1	1	**1**
1	0	**1**	0	0	**1**
1	1	**1**	0	0	**1**

(a) $x + \overline{x} \bullet y = x + y$

La demostración tabular del teorema requiere de una tabla como la 2.8.

(b) $x \bullet (\overline{x} + y) = x \bullet y$ Queda demostrado por dualidad.

- **Demostración del teorema T6**

(a) $x + (y + z) = (x + y) + z$

Se usará el método tabular, por lo que se requiere representar todas las combinaciones lógicas posibles de los elementos x, y, z y los resultados parciales $(x + y)$ e $(y + z)$.

Tabla 2.9: Demostración tabular del teorema T6

x	y	z	$(y + z$	$x + (y + z)$	$(x + y)$	$(x + y) + z$
0	0	0	0	**0**	0	**0**
0	0	1	1	**1**	0	**1**
0	1	0	1	**1**	1	**1**
0	1	1	1	**1**	1	**1**
1	0	0	0	**1**	1	**1**
1	0	1	1	**1**	1	**1**
1	1	0	1	**1**	1	**1**
1	1	1	1	**1**	1	**1**

(b) $x \bullet (y \bullet z) = (x \bullet y) \bullet z$ Queda demostrado por dualidad.

- **Demostración del teorema T7**

(a) $\overline{x+y} = \overline{x} \bullet \overline{y}$

La tabla 2.10 comprueba la veracidad del teorema para toda combinación lógica de los elementos x e y.

Tabla 2.10: Demostración tabular del teorema T7

x	y	$x+y$	$\overline{x+y}$	$\overline{x}$	$\overline{y}$	$\overline{x} \bullet \overline{y}$
0	0	0	**1**	1	1	**1**
0	1	1	**0**	1	0	**0**
1	0	1	**0**	0	1	**0**
1	1	1	**0**	0	0	**0**

(b) $\overline{x \bullet y} = \overline{x} + \overline{y}$ Se demuestra por dualidad.

2.3. Variables y funciones binarias

Una *variable binaria* (también llamada lógica o de conmutación) es un símbolo (normalmente una letra con algún subíndice, o sin él) al cual se le puede asignar el valor lógico 0 o el valor lógico 1. Las letras x,y,z que se han utilizado para los enunciados de los postulados y teoremas del álgebra de conmutación, son ejemplos de variables binarias.

Una *función binaria* (de conmutación o lógica) de n variables es una regla que marca o asocia un valor binario (1 o 0) a cada una de las posibles combinaciones binarias de las n variables.

Ejemplo.

Para una función de dos variables (x, y), existen cuatro posibles combinaciones binarias $(x, y) = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$. Una función binaria concreta, $f(x, y)$, asocia a cada combinación (x, y) un 1 o un 0. En la tabla 2.11 se muestra un caso particular de función que para la combinación $(x, y) = (1, 1)$ asigna un 1 y, para las restantes combinaciones, asigna un 0.

La definición de función de conmutación se puede expresar de modo más formal.

Tabla 2.11: Ejemplo de función de conmutación de dos variables

x	y	**f**
0	0	0
0	1	0
1	0	0
1	1	1

Una función de conmutación de n variables es una aplicación del conjunto B^n en B

$$f : B^n \rightarrow B$$

donde $B^n = B_1 \times B_2 \times B_3 \ldots \times B_n$ es el producto cartesiano de los conjuntos de elementos del álgebra de conmutación y un elemento perteneciente a dicho producto cartesiano, $x \in B^n$, se compone de una n-tupla $(x_1, x_2, x_3..x_n)$.

La definición de una función de conmutación de n variables sugiere que esta pueda ser representada por una tabla de n+1 columnas, de las cuales las n primeras representan los valores de las variables binarias y, la última, el valor de la función para cada combinación. Esta tabla de combinaciones se denomina *tabla de verdad*.

Tabla 2.12: Tabla de verdad de una función genérica de n variables

$x_1x_2 \ldots x_n$	$f(x_1, x_2, \ldots, x_n)$
$00 \ldots 0$	$f(0, 0, \ldots, 0)$
$00 \ldots 1$	$f(0, 0, \ldots, 1)$
$\ldots$	$\ldots$
$11 \ldots 1$	$f(1, 1, \ldots, 1)$

Se dice que una función de conmutación es completa cuando ésta se encuentra definida (toma el valor 0 o 1) para toda combinación de entrada. Una función de conmutación es incompleta cuando existen combinaciones de entrada a las que no se les a asociado un valor concreto de la función. Por ahora, todas las funciones binarias que se presentarán son completas, se deja para el apartado 2.5, el estudio de las funciones incompletas.

Existen 2^{2^n} funciones de conmutación completas de n variables. Si para n variables existen 2^n combinaciones binarias posibles y una función puede asignar, o el 1 lógico o el 0, a cada combinación binaria de las variables, el número de funciones distintas es de 2^{2^n}. En las tablas 2.13 y 2.14 se muestran

las 16 funciones de conmutación distintas de 2 variables y a continuación se realiza una descripción de cada una de ellas.

Tabla 2.13: Tabla de verdad con las 16 funciones posibles de 2 variables.

xy	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0 0	0	0	0	0	0	0	0	0
0 1	0	0	0	0	1	1	1	1
1 0	0	0	1	1	0	0	1	1
1 1	0	1	0	1	0	1	0	1

Tabla 2.14: Tabla de verdad con las 16 funciones posibles de 2 variables (cont).

xy	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0 0	1	1	1	1	1	1	1	1
0 1	0	0	0	0	1	1	1	1
1 0	0	0	1	1	0	0	1	1
1 1	0	1	0	1	0	1	0	1

- La función F_0 asigna el valor 0 para toda combinación de (x, y). Se puede decir que $F_0 = 0$. Se denomina como la función constante cero.

- La función F_1 asigna el valor 1 para la combinación (x,y)=(1,1) y 0 para las restantes combinaciones. F_1 es la función AND y se puede expresar como $F_1 = x \bullet y$.

- La función F_2 asigna el valor 1 a la combinación (x,y)=(1,0) y 0 a las restantes. Algebraicamente puede expresarse como $F_2 = x \bullet \overline{y}$. Puede comprobarse que, al substituir cualquier combinación de valores lógicos de (x,y) en la expresión $x \bullet \overline{y}$, sólo, la (x,y)=(1,0), hace que el término producto valga 1, las restantes generan un resultado igual a 0.

- La función F_3 asigna el valor lógico 1 a aquellas combinaciones en las que la variable x vale 1 y 0 a las que x toma el valor 0. Se puede expresar esta función como $F_3 = x$ y se denomina la función transferencia de x.

- La función F_4, asigna el valor 1 a la entrada (x,y)=(0,1) y 0, a las restantes. Algebraicamente las podemos expresar como $F_4 = \overline{x} \bullet y$.

- La función F_5 asigna el valor 1 lógico a todas aquellas combinaciones en las que la variable y toma el valor 1, y 0 en aquellas en las que y vale 0. Por tanto F_5 se denomina la función transferencia de y y se expresa como $F_5 = y$.

- La función F_6, asigna el valor 1 para (x,y)=(1,0) y (0,1) y 0 para las restantes. Algebraicamente se puede expresar como $F_6 = \overline{x} \bullet y + x \bullet \overline{y}$. Substituyendo todas las combinaciones binarias de (x,y) en la expresión anterior se puede comprobar su validez. Esta función se denomina la función EXOR (Exclusive-OR) de x e y, se simboliza, también, como $x \oplus y$.

- La función F_7 asigna el valor 0 a la combinación (x,y)=(0,0), y 1 a las restantes. Se puede expresar como $F_7 = x + y$, y es la función OR.

- La función F_8, asigna el valor 1 a la combinación (x,y)=(0,0), y 0 a las restantes. Nótese que, en comparación con F_7, esta función hace su inversa o su complemento. Se puede expresar como $F_8 = \overline{x + y}$ y es la función NOR (Negate-OR).

- La función F_9, asigna el valor 1 a las combinaciones (x,y)=(0,0) y (1,1) y 0 a las restantes. Nótese que, en comparación con F_6, la función EXOR, esta función implementa su complemento. Se puede expresar como $F_9 = \overline{x} \bullet \overline{y} + x \bullet y$, es la función NEXOR (Negate-EXOR).

- La función F_{10} asigna el valor 1 para aquellas entradas en las que y toma el valor 0, y asigna el 0 para las que y toma el valor 1. Por tanto esta es la función complemento de y (o función NOT de y) que se representa como $F_{10} = \overline{y}$. Notése que esta función es la versión complementada de F_5.

- La función F_{11} asigna el valor 0 para la combinación de (x,y)=(0,1) y 1 para las restantes. Nótese que F_{11} es la versión complementada de F_4, por tanto $F_{11} = \overline{F_4} = \overline{\overline{x} \bullet y} = \overline{\overline{x}} + \overline{y} = x + \overline{y}$.

- La función F_{12} asigna el valor 1 para aquellas entradas en las que x toma el valor 0, y asigna el 0 para las que x toma el valor 1. Esta es la función complemento de x (o función NOT de x) que se representa como $F_{12} = \overline{x}$.

- La función F_{13} asigna el valor 0 para la combinación (x,y)=(1,0) y 1 para las restantes. Nótese que F_{13} es la versión complementada de F_2, por tanto $F_{13} = \overline{F_2} = \overline{x \bullet \overline{y}} = \overline{x} + y$.

- La función F_{14} asigna el valor 0 para la combinación (x,y)=(1,1) y 0 para las restantes. Obsérvese que F_{14} es la versión complementada de F_1, la función AND, y por tanto puede expresarse como $F_{14} = \overline{x \bullet y}$. Se denomina función NAND (Negate-AND).

- La función $F_{15} = 1$, es la función constante igual a 1.

2.4. Expresiones de conmutación

Una expresión de conmutación de n variables consiste en un número finito de constantes (0, 1) y variables conectados por los operadores $(+)$,$(\bullet)$ *y* $()$*de forma que* $(+)$ *y* $(\bullet)$ *no pueden estar adyacentes nunca.*

Ejemplos.

$x + y \bullet \overline{z}$

$a \bullet [\overline{(a + b \bullet c)} + b]$

$\overline{0} \bullet [\overline{(a + 1 \bullet c)}]$

Cada expresión de conmutación de n-variables describe una única función de conmutación de n-variables. Esto es, si para una expresión de conmutación de n variables, se substituyen las n-variables por los 2^n valores posibles que pueden tomar éstas y se opera matemáticamente haciendo uso de los postulados y teoremas del álgebra de conmutación se obtendrá, como resultado de cada substitución, un 0 o un 1. En definitiva, se está describiendo una función de conmutación.

Dos expresiones de conmutación A y B se dicen equivalentes (y se representa A=B) sii (si y sólo si) ellas describen la misma función de conmutación. Si para todas las combinaciones posibles de las variables de entradas, la expresión A evalúa lo mismo que la B, ambas son equivalentes. Dicho de otra forma, la expresión A describe una función de conmutación que puede representarse mediante su tabla de verdad y del mismo modo, la expresión B describe una función que, también, es representable en una tabla de verdad. Si ambas tablas de verdad son idénticas, las expresiones A y B son equivalentes.

Ejemplo.

Sean f y g dos funciones de conmutación descritas por las expresiones $f = a + b \bullet \overline{c}$ y $g = (a + b) \bullet (a + \overline{b} + \overline{c})$. Para determinar si estas son equivalentes entre sí, se representará la tabla de verdad de f y g (ver tabla 2.15). Obsérvese

que la función f toma el valor 1 siempre que la variable a sea 1 ($f = 1+b\bullet\overline{c} = 1$) o bien si las variables b y c toman los valores 1 y 0 respectivamente ($f = a+1\bullet\overline{0} = a+1\bullet 1 = a+1 = 1$). Entonces, f evalúa 1 para las combinaciones en las variables $(a,b,c) = (0,1,0),(1,0,0),(1,0,1),(1,1,0),(1,1,1)$, para cualquier otra combinación, f evalúa 0. Por otro lado, la función g toma el valor de 0 si las variables a y b son 0 a la vez ($g = (0+0)\bullet(0+\overline{0}+\overline{c}) = 0\bullet(0+1+\overline{c}) = 0$) o si las tres variables (a,b,c) toman los valores (0,1,1) respectivamente ($g = (0+1)\bullet(0+\overline{1}+\overline{1}) = 1\bullet(0+0+0) = 1\bullet 0 = 0$). Por tanto g toma el valor 0 para las entradas $(a,b,c) = (0,0,0),(0,0,1)\ y\ (0,1,1)$, para cualquier otra combinación, g toma el valor 1. Observando la tabla 2.15, se comprueba que las expresiones asociadas a las funciones f y g son equivalentes.

Tabla 2.15: Tabla de verdad de las funciones $f = a+b\bullet\overline{c}$ y $g = (a+b)\bullet(a+\overline{b}+\overline{c})$

a	**b**	**c**	**f**	**g**
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

2.4.1. Expresiones normales

Se define un literal como una variable binaria que aparece complementada o sin complementar en una expresión de conmutación

Se llama término producto (o conjunción) a un literal o producto de literales.

Se llama término suma (o disyunción) a un literal o suma de literales.

Una expresión de conmutación representada por un simple término producto o suma de ellos se llama expresión normal disyuntiva o suma de productos.

Un expresión de conmutación representada por un simple término suma o producto de ellos se llama expresión normal conjuntiva o producto de sumas.

Ejemplos.

1. La expresión $(a + b) \bullet (a + \overline{b} + \overline{c})$ tiene tres variables, cinco líterales y dos términos suma $(a + b)$ y $(a + \overline{b} + \overline{c})$. Se puede concluir que es una expresión en producto de sumas o normal conjuntiva.

2. La expresión $a + b \bullet \overline{c}$ tiene tres variables, tres literales y dos términos producto a y $b \bullet \overline{c}$, por tanto es una expresión normal disyuntiva o suma de productos.

3. La expresión $a \bullet [\overline{(a + b \bullet c)} + b]$, no es una expresión normal. En primer lugar, el operador complemento no aparece sólo sobre las variables y no existen claramente ni términos productos ni términos suma.

2.4.2. Expresión en suma de mintérminos o forma canónica disyuntiva

La expresión normal disyuntiva se define como un término producto o suma de ellos. Por tanto la expresión 2.1 es un ejemplo que satisface dicha definición pero que, también, tiene la característica de que todas las variables de la función f, aparecen una vez en cada término producto (complementadas o sin complementar). Un término producto con esta propiedad se llama **mintérmino o producto estándar**. Una expresión consistente sólo de mintérminos en la que no aparezcan dos iguales se dice que está en **forma canónica disyuntiva**, o simplemente, en forma de **suma de mintérminos**.

$$f(a, b, c) = a \bullet b \bullet c + a \bullet \overline{b} \bullet \overline{c} + \overline{a} \bullet \overline{b} \bullet c \tag{2.1}$$

Para n variables existen 2^n mintérminos, ya que cada variable en el término producto puede aparecer complementada o sin complementar. La lista de los mintérminos de 2 variables aparece en la tabla 2.16

La lista de los mintérminos de 3 variables aparece en la tabla 2.17

Tabla 2.16: Lista de mintérminos de dos variables

$\overline{x_1} \bullet \overline{x_2}$
$\overline{x_1} \bullet x_2$
$x_1 \bullet \overline{x_2}$
$x_1 \bullet x_2$

Tabla 2.17: Lista de mintérminos de tres variables

$\overline{x_1} \bullet \overline{x_2} \bullet \overline{x_3}$
$\overline{x_1} \bullet \overline{x_2} \bullet x_3$
$\overline{x_1} \bullet x_2 \bullet \overline{x_3}$
$\overline{x_1} \bullet x_2 \bullet x_3$
$x_1 \bullet \overline{x_2} \bullet \overline{x_3}$
$x_1 \bullet \overline{x_2} \bullet x_3$
$x_1 \bullet x_2 \bullet \overline{x_3}$
$x_1 \bullet x_2 \bullet x_3$

Dada una lista completa de los mintérminos de n-variables, si a cada una de las n-variables se le asigna el valor 0 o 1, entonces sólo un mintérmino de la lista tomará el valor 1 y los otros el 0.

Ejemplos. (para 3 variables, usando la tabla 2.17)

Para la entrada (1 1 0) sólo toma el valor 1 el mintérmino $x_1 \bullet x_2 \bullet \overline{x_3}$, mientras que los otros mintérminos, para esta misma entrada, evalúan 0.

Para la entrada (0 0 1) sólo toma el valor 1 el minérmino $\overline{x_1} \bullet \overline{x_2} \bullet x_3$, mientras que los otros mintérminos, para esta misma entrada, evalúan 0.

A partir de una combinación de entrada, se puede obtener, directamente, el mintér-mino de la lista para el que dicha combinación de entrada evalúa 1. El mintérmino a escoger debe tener las variables complementas para aquellas entradas que son 0 y sin complementar para las que son 1. Por ejemplo, para la entrada $(x_1, x_2, x_3) = (1, 0, 1)$ el mintérmino que evalúa 1 es aquel en el que la variable x_1 aparece sin complementar (por ser $x_1 = 1$), x_2 aparece complementada (por ser $x_2 = 0$) y x_3 aparece sin complementar (por ser $x_3 = 1$), o sea, el mintérmino ($x_1 \bullet \overline{x_2} \bullet x_3$).

Toda función de conmutación puede expresarse en forma canónica de min-

términos . De hecho, si se escogen los mintérminos asociados a las combinaciones de entrada que hagan que la función valga 1 y realizando la suma lógica de estos, se obtendría una expresión que describiría a la función.

Ejemplo. Sea la función f cuya tabla de verdad esta representada por 2.18

Tabla 2.18: Tabla de verdad de una función de tres variables.

x	y	z	f
0	0	0	0
0	0	1	0
0	**1**	**0**	**1**
0	**1**	**1**	**1**
1	0	0	0
1	**0**	**1**	**1**
1	1	0	0
1	**1**	**1**	**1**

En la tabla 2.18 aquellas filas de la tabla correspondientes a combinaciones de entrada que hacen $f = 1$ se han destacado en negrita. Los mintérminos asociados a esas entradas son $\overline{x} \bullet y \bullet \overline{z}$ para 010, $\overline{x} \bullet y \bullet z$ para 011, $x \bullet \overline{y} \bullet z$ para 101 y $x \bullet y \bullet z$ para 111. Por tanto la suma de estos mintérminos (expresión 2.2) describe la función f representada en la tabla.

$$f = \overline{x} \bullet y \bullet \overline{z} + \overline{x} \bullet y \bullet z + x \bullet \overline{y} \bullet z + x \bullet y \bullet z \tag{2.2}$$

Para comprobar que la ecuación 2.2 representa la función dada por la tabla 2.18 nos basamos en la propiedad que dice que dada una combinación de entrada, sólo existe un mintérmino, de todos los posibles, que tomará el valor 1 para dicha combinación, mientras que los restantes toman el valor 0. En la expresión 2.2 no aparece, por ejemplo, el mintérmino $\overline{x} \bullet \overline{y} \bullet \overline{z}$, que es el único que tomaría el valor 1 para la combinación de entrada $(x, y, z) = (0, 0, 0)$. Esto implica que, para dicha combinación, la función f, evalúa 0. Este mismo razonamiento se extiende a las entradas $(x, y, z) = (0, 0, 1), (1, 0, 0), (1, 1, 0)$ cuyos mintérminos asociados no aparecen en la expresión 2.2. Algo diferente ocurre para la entrada $(x, y, z) = (0, 1, 0)$, cuyo mintérmino, $\overline{x} \bullet y \bullet \overline{z}$, sí aparece en la expresión de f. Si se evalúa la expresión de f para esta nueva entrada, el mintérmino $\overline{x} \bullet y \bullet \overline{z}$ toma el valor 1 y gracias al teorema de los elementos dominantes T2.a la función f también vale 1. Este razonamiento, se hace ex-

tensivo a las restantes entradas asociadas a los mintérminos que aparecen en la expresión que define la función, para las que esta última toma el valor 1.

La expresión en suma de mintérminos de una función de conmutación es única. No pueden existir dos expresiones canónicas distintas que describan la misma función de conmutación ya que, al ser distintas, deben diferir en, al menos, un mintérmino. Este mintérmino provocaría que la expresión que lo contenga tome el valor 1 para una combinación de entrada que aplicada a la otra expresión, que no lo tiene, hace que esta última tome el valor 0. Si para una combinación de entrada una expresión evalúa diferente que una segunda expresión, las funciones descritas por ambas, no pueden ser idénticas.

Toda expresión de conmutación completa puede ser descrita en forma canónica disyuntiva o suma de mintérminos . Para conseguirlo se deben seguir una serie de pasos que se enumeran a continuación.

1. Si se supone que una expresión puede tener cualquier forma $\overline{(a+b)}$ o $\overline{(a \bullet b)}$. Nos interesa que los operadores complemento aparezcan sólo sobre las variables y no sobre productos o sumas de ellos. Desarrollamos dichas expresiones aplicando las leyes de Morgan (Teorema T7a) las veces que sean necesarias.

2. A continuación se aplica la propiedad distributiva del operador $\bullet$ sobre el operador $+$ (Postulado 4a) y se simplifican los términos $x \bullet \overline{x} = 0$ (Postulado 5b).

   Con los dos pasos anteriores, se obliga a que la expresión resultante sea de tipo normal. Para conseguir los mintérminos cada término producto de la suma debe contener todas las variables de la función.

3. Se multiplican aquellos términos productos que no contengan todas las variables de la función por la expresión $\Pi(x_i+\overline{x_i})$ (siendo x_i las variables ausentes del término) y se aplica, de nuevo, la propiedad distributiva del operador $\bullet$ sobre el operador $+$.

4. Por último, se simplifican aquellos mintérminos que aparezcan repetidos aplicando el teorema T1a.

Ejemplo. Pasar a forma de mintérminos la expresión $f(a,b,c) = \overline{[(a \bullet b)+c]}$

- Paso 1. Se aplica Ley de Morgan.
  $f(a,b,c) = \overline{[(a \bullet b) + c]} = \overline{[a \bullet b]} \bullet \overline{c} = (\overline{a} + \overline{b}) \bullet \overline{c}$
- Paso 2. Postulado distributivo.
  $f(a,b,c) = (\overline{a} + \overline{b}) \bullet \overline{c} = \overline{a} \bullet \overline{c} + \overline{b} \bullet \overline{c}$
- Paso 3. Se añaden las variables que faltan a los términos productos y se aplica el postulado distributivo.
  $f(a,b,c) = \overline{a} \bullet \overline{c} + \overline{b} \bullet \overline{c} = \overline{a} \bullet \overline{c} \bullet (b + \overline{b}) + \overline{b} \bullet \overline{c} \bullet (a + \overline{a}) = \overline{a} \bullet b \bullet \overline{c} + \overline{a} \bullet \overline{b} \bullet \overline{c} + a \bullet b \bullet c + a \bullet \overline{b} \bullet c$
- Paso 4. Eliminar mintérminos repetidos.
  $f(a,b,c) = \overline{a} \bullet b \bullet \overline{c} + \overline{a} \bullet \overline{b} \bullet \overline{c} + a \bullet b \bullet c + a \bullet \overline{b} \bullet c$

2.4.2.1. Notación m

La *Notación m* introduce una manera simplificada de representar los mintérminos de una función. En la tabla 2.19 se han representado la lista de mintérminos de tres variables y la correspondiente notación simplificada.

Tabla 2.19: Lista de los mintérminos de tres variables, entradas binarias y decimales asociadas y notación m reducida.

$\overline{x_1} \bullet \overline{x_2} \bullet \overline{x_3}$	0	0	0	0	m_0
$\overline{x_1} \bullet \overline{x_2} \bullet x_3$	0	0	1	1	m_1
$\overline{x_1} \bullet x_2 \bullet \overline{x_3}$	0	1	0	2	m_2
$\overline{x_1} \bullet x_2 \bullet x_3$	0	1	1	3	m_3
$x_1 \bullet \overline{x_2} \bullet \overline{x_3}$	1	0	0	4	m_4
$x_1 \bullet \overline{x_2} \bullet x_3$	1	0	1	5	m_5
$x_1 \bullet x_2 \bullet \overline{x_3}$	1	1	0	6	m_6
$x_1 \bullet x_2 \bullet x_3$	1	1	1	7	m_7

Ejemplos. Obtenga las expresiones algebraicas de los siguientes mintérminos, suponiendo que, para todos los casos, se utilizan 4 variables.

- m_4. El valor binario de la entrada asociada, usando cuatro bits, es 0100, por tanto $m_4 = \overline{x_1} \bullet x_2 \bullet \overline{x_3} \bullet \overline{x_4}$

- m_14. El valor binario de la entrada asociada, usando cuatro bits, es 1110, por tanto $m_{14} = x_1 \bullet x_2 \bullet x_3 \bullet \overline{x_4}$

Una función de conmutación de n variables puede expresarse de forma resumida utilizando la Notación m.

Ejemplo. La función $f = \overline{x_1} \bullet \overline{x_2} \bullet \overline{x_3} + \overline{x_1} \bullet x_2 \bullet \overline{x_3} + x_1 \bullet x_2 \bullet \overline{x_3}$ puede expresarse, de forma más simplificada, sustituyendo la expresión de cada mintérmino por el correspondiente símbolo en notación m (ver tabla 2.19). Entonces $f = m_0 + m_2 + m_6$, que a su vez se puede expresar como $f = \sum m(0, 2, 6) = \sum(0, 2, 6)$.

2.4.3. El primer teorema de expansión

Se lista en dos apartados.

1. *Cualquier función de conmutación completa de n variables puede expresarse de la forma dada por la ecuación 2.3, donde las funciones* $f(0, x_2, .. \ x_n)$ *y* $f(1, x_2, ..x_n)$ *se denominan funciones residuo de* f *para* $x_1 = 0$ *y* $x_1 = 1$ *respectivamente.*

$$f(x_1, x_2, \ldots, x_n) = \overline{x_1} \bullet f(0, x_2, \ldots, x_n) + x_1 \bullet f(1, x_2, \ldots, x_n) \qquad (2.3)$$

La validez del apartado (a) del primer teorema de expansión se hace extensiva a las restantes variables, $x_2, x_3 ... x_n$, y su demostración es muy simple: substituyendo en ambos miembros de la igualdad de 2.3 $x_1 = 0$ en primer lugar, y, posteriormente $x_1 = 1$, se puede comprobar que se satisface la igualdad.

Este teorema permite encontrar una expresión de n variables que describa una función de conmutación, también de n variables, a partir de las expresiones de los residuos de la dicha función.

Ejemplo. Sea $f(a, b, c) = a \bullet [\overline{(a + b \bullet c)} + b]$

La función residuo $f(0, b, c)$ se obtiene haciendo $a = 0$ en la expresión asociada a la función del ejemplo. Como se puede deducir, $f(0, b, c) = 0$.

De igual forma, la función residuo $f(1,b,c)$ se obtiene haciendo $a=1$ en dicha expresión. $f(a,1,c)=1 \bullet [\overline{(1+b \bullet c)}+b]=b$.

Según el apartado (a) del primer teorema de expansión, $f(a,b,c)=\overline{a} \bullet f(0,b,c)+a \bullet f(1,b,c)=\overline{a} \bullet 0+a \bullet b=a \bullet b$.

---

2. *Toda función de conmutación completa de n variables puede escribirse de la forma dada por la expresión 2.4 donde $f(i)$ es el valor que toma la función f para la entrada decimal i, y $m_i(x_1,x_2,..,x_n)$ el mintérmino asociado a dicha entrada i.*

$$f(x_1,x_2,\ldots,x_n)=\sum_{i=0}^{2^n-1} f(i) \bullet m_i(x_1,x_2,\ldots,x_n) \tag{2.4}$$

Dicho de otra forma, cualquier función de conmutación completa de n variables puede expresarse como la suma del producto de lo que evalúa la función f para cada entrada por el mintérmino asociado a dicha entrada.

A continuación se demostrará este apartado del primer teorema de expansión, para el caso concreto de funciones de tres variables, sin que esto suponga una merma en la generalidad del mismo.

Sea $f(x_1,x_2,x_3)$ una función de conmutación completa de tres variables. Por el apartado 1 del primer teorema de expansión, la función de conmutación se puede desarrollar por la expresión 2.5

$$f(x_1,x_2,x_3)=\overline{x_1} \bullet f(0,x_2,x_3)+x_1 \bullet f(1,x_2,x_3) \tag{2.5}$$

A su vez, los residuos de 2.5 son funciones de dos variables que, también, son susceptibles de desarrollarse según el apartado 1 del primer teorema de expansión (ecuación 2.6).

$$\begin{aligned} f(x_1,x_2,x_3) \quad &= \overline{x_1} \bullet [\overline{x_2} \bullet f(0,0,x_3)+x_2 \bullet f(0,1,x_3)] \\ &\quad +x_1 \bullet [\overline{x_2} f(1,0,x_3)+x_2 \bullet f(1,1,x_3)] \\ &= \overline{x_1} \bullet \overline{x_2} \bullet f(0,0,x_3)+\overline{x_1} \bullet x_2 \bullet f(0,1,x_3) \\ &\quad +x_1 \bullet \overline{x_2} \bullet f(1,0,x_3)+x_1 \bullet x_2 \bullet f(1,1,x_3) \end{aligned} \tag{2.6}$$

Las funciones de una variable $f(0,0,x_3)$, $f(0,1,x_3)$, $f(1,0,x_3)$ y $f(1,1,x_3)$ que aparecen en 2.6, a su vez, pueden desarrollarse según 1 y que sustituidas en 2.6 permiten expresar $f(x_1,x_2,x_3$ según ecuación 2.7.

$$\begin{aligned} f(x_1,x_2,x_3) &= \overline{x_1}\ \overline{x_2}\ \overline{x_3} f(0,0,0) + \overline{x_1}\ \overline{x_2}\ x_3 f(0,0,1) \\ &+ \overline{x_1}\ x_2\ \overline{x_3} f(0,1,0) + \overline{x_1}\ x_2\ x_3 f(0,1,1) \\ &+ x_1\ \overline{x_2}\ \overline{x_3} f(1,0,0) + x_1\ \overline{x_2}\ x_3 f(1,0,1) \\ &+ x_1\ x_2\ \overline{x_3} f(1,1,0) + x_1\ x_2\ x_3 f(1,1,1) \\ &= \sum_{i=0}^{2^3-1} m_i(x_1,x_2,x_3) f(i) \end{aligned} \tag{2.7}$$

El segundo apartado del primer teorema de expansión da una forma simple, pero compacta, de expresar cualquier función de conmutación de n variables.

Ejemplo. Sea f una función de conmutación completa de tres variables descrita por la tabla de verdad 2.20

Tabla 2.20:

x	y	z	f
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

El segundo apartado del primer teorema de expansión permite expresar la función f según la ecuación 2.8

$$\begin{aligned} f(x_1,x_2,x_3) &= \overline{x_1}\ \overline{x_2}\ \overline{x_3} \bullet 0 + \overline{x_1}\ \overline{x_2}\ x_3 \bullet 1 \\ &+ \overline{x_1}\ x_2\ \overline{x_3} \bullet 1 + \overline{x_1}\ x_2\ x_3 \bullet 0 \\ &+ x_1\ \overline{x_2}\ \overline{x_3} \bullet 1 + x_1\ \overline{x_2}\ x_3 \bullet 0 \\ &+ x_1\ x_2\ \overline{x_3} \bullet 0 + x_1\ x_2\ x_3 \bullet 1 \\ &= \overline{x_1}\ \overline{x_2}\ x_3 + \overline{x_1}\ x_2\ \overline{x_3} + x_1\ \overline{x_2}\ \overline{x_3} + x_1\ x_2\ x_3 \end{aligned} \tag{2.8}$$

Puede observarse que f viene expresada como la suma de aquellos mintérminos asociados a las entradas para las que la función vale 1 en la tabla 2.20

2.4.4. Expresión en suma de maxtérminos o forma canónica conyuntiva

La expresión normal conjuntiva se define como un término suma o producto de ellos. Por tanto la expresión 2.9 es un ejemplo que satisface dicha definición pero que, también, tiene la característica de que todas las variables de la función f, aparecen una vez en cada término suma (complementadas o sin complementar). Un término suma con esta propiedad se llama **maxtérmino o suma estándar**. Una expresión consistente sólo de maxtérminos en la que no aparezcan dos iguales se dice que está en **forma canónica cconjuntiva**, o simplemente, en forma de **producto de sumas**.

$$f(a,b,c) = (a+b+c) \bullet (a+\overline{b}+\overline{c}) \bullet (\overline{a}+\overline{b}+c) \tag{2.9}$$

Para n variables existen 2^n maxtérminos, ya que cada variable en el término suma puede aparecer complementada o sin complementar. La lista de los maxtérminos de 2 variables aparece en la tabla 2.21

Tabla 2.21: Lista de maxtérminos de dos variables

$x_1 + x_2$
$x_1 + \overline{x_2}$
$\overline{x_1} + x_2$
$\overline{x_1} + \overline{x_2}$

La lista de los maxtérminos de 3 variables aparece en la tabla 2.22

Tabla 2.22: Lista de maxtérminos de tres variables

$x_1 + x_2 + x_3$
$x_1 + x_2 + \overline{x_3}$
$x_1 + \overline{x_2} + x_3$
$x_1 + \overline{x_2} + \overline{x_3}$
$\overline{x_1} + +x_2 + x_3$
$\overline{x_1} + x_2 + \overline{x_3}$
$\overline{x_1} + \overline{x_2} + x_3$
$\overline{x_1} + \overline{x_2} + \overline{x_3}$

Dada una lista completa de los maxtérminos de n-variables, si a cada una de las n-variables se le asigna el valor 0 o 1, entonces sólo un maxtérmino de la lista tomará el valor 0 y los otros el 1.

Ejemplo. (para 3 variables, usando la tabla 2.22)

Para la entrada (1 1 0) sólo toma el valor 0 el maxtérmino $\overline{x_1} + \overline{x_2} + x_3$, mientras que los otros maxtérminos, para esta misma entrada, evalúan 1.

Para la entrada (0 0 1) sólo toma el valor 0 el maxtérmino $x_1 + x_2 + \overline{x_3}$, mientras que los otros maxtérminos, para esta misma entrada, evalúan 0.

A partir de una combinación de entrada se puede obtener, directamente, el maxtér-mino de la lista para el que dicha combinación de entrada evalúa 0. El maxtérmino a escoger debe tener las variables complementas para aquellas entradas que son 1 y sin complementar para las que son 0. Por ejemplo, para la entrada $(x_1, x_2, x_3) = (1, 0, 1)$ el maxtérmino que evalúa 1 es aquel en el que la variable x_1 aparece complementada (por ser $x_1 = 1$), x_2 aparece sin complementar (por ser $x_2 = 0$) y x_3 aparece complementada (por ser $x_3 = 1$), o sea, el maxtérmino ($\overline{x_1} + x_2 + \overline{x_3}$).

Toda función de conmutación puede expresarse en forma canónica de maxtérminos . De hecho, si se escogen los maxtérminos asociados a las combinaciones de entrada que hagan que la función valga 0 y realizando el producto lógico de estos, se obtendría una expresión que describiría a la función.

Ejemplo. Sea la función f cuya tabla de verdad está representada por 2.23

Tabla 2.23: Tabla de verdad de una función de tres variables.

x	y	z	f
0	**0**	**0**	**0**
0	**0**	**1**	**0**
0	1	0	1
0	1	1	1
1	**0**	**0**	**0**
1	0	1	1
1	**1**	**0**	**0**
1	1	1	1

Se han destacado en negrita aquellas filas de la tabla 2.23 que se corresponden a combinaciones de entrada que hacen $f = 0$. Los maxtérminos asociados a esas entradas son $x + y + z$ para 000, $x + y + \overline{z}$ para 001, $\overline{x} + y + z$ para 100

y $\overline{x}+\overline{y}+z$ para 110. Por tanto el producto de dichos maxtérminos (expresión 2.10) describe la función f representada.

$$f = (x+y+z) \bullet (x+y+\overline{z}) \bullet (\overline{x}+y+z) \bullet (\overline{x}+\overline{y}+z) \tag{2.10}$$

Para comprobar que la ecuación 2.10 representa la función dada por la tabla 2.23 nos basamos en la propiedad que dice que dada una combinación de entrada, sólo existe un maxtérmino, de todos los posibles, que tomará el valor 0 para dicha combinación, mientras que los restantes toman el valor 1. En la expresión 2.10 no aparece, por ejemplo, el maxtérmino $\overline{x}+\overline{y}+\overline{z}$, que es el único que tomaría el valor 0 para la combinación de entrada $(x,y,z)=(1,1,1)$. Esto implica que, para dicha combinación, la función f, evalúa 1. Este mismo razonamiento se extiende a las entradas $(x,y,z)=(0,1,0),(0,1,1),(1,0,1)$ cuyos maxtérminos asociados no aparecen en la expresión 2.10. Algo diferente ocurre para la entrada $(x,y,z)=(0,0,0)$, cuyo maxtérmino, $x+y+z$, sí aparece en la expresión de f. Si se evalúa la expresión de f para esta nueva entrada, el maxtérmino $x+y+z$ toma el valor 0 y gracias al teorema de los elementos dominantes T2.b la función f también vale 0. Este razonamiento, se hace extensivo a las restantes entradas asociadas a los maxtérminos que aparecen en la expresión que define la función, para las que esta última toma el valor 0.

La expresión en producto de maxtérminos de una función de conmutación es única. No pueden existir dos expresiones canónicas distintas que describan la misma función de conmutación ya que, al ser distintas, deben diferir en, al menos, un maxtérmino. Este maxtérmino provocaría que la expresión que lo contenga tome el valor 0 para una combinación de entrada que aplicada a la otra expresión, que no lo tiene, hace que esta última tome el valor 1. Si para una combinación de entrada una expresión evalúa diferente que una segunda expresión, las funciones descritas por ambas, no pueden ser idénticas.

Toda expresión de conmutación completa puede ser descrita en forma canónica conjuntiva o producto de maxtérminos . Para conseguirlo se deben seguir una serie de pasos que se enumeran a continuación.

1. Si se supone que una expresión puede tener cualquier forma $\overline{(a+b)}$ o $\overline{(a \bullet b)}$. Nos interesa que los operadores complemento aparezcan sólo sobre las variables y no sobre productos o sumas de ellos. Desarrolla-

mos dichas expresiones aplicando las leyes de Morgan (Teorema T7b) las veces que sean necesarias.

2. A continuación se aplica la propiedad distributiva del operador $+$ sobre el operador $\bullet$ (Postulado 4b) y se simplifican los términos $x + \overline{x} = 1$ (Postulado 5a).

   Con los dos pasos anteriores, se obliga a que la expresión resultante sea de tipo normal. Para conseguir los maxtérminos cada término suma del producto debe contener todas las variables de la función.

3. Se suman a aquellos términos suma que no contengan todas las variables de la función la expresión $\sum(x_i \bullet \overline{x_i})$ (siendo x_i las variables ausentes del término) y se aplica, de nuevo, la propiedad distributiva del operador $+$ sobre el operador $\bullet$.

4. Por último, se simplifican aquellos maxtérminos que aparezcan repetidos aplicando el teorema T1b.

Ejemplo. Pasar a forma de maxtérminos la expresión $f(a,b,c) = \overline{[(a \bullet b) + c]}$

- Paso 1. Se aplica Ley de Morgan.

  $f(a,b,c) = \overline{[a \bullet b]} \bullet \overline{c} = (\overline{a} + \overline{b}) \bullet \overline{c}$

- Paso 2. Postulado distributivo.

  No es necesario aplicarlo porque la expresión está en forma de producto de sumas.

- Paso 3. Se añaden las variables que faltan a los términos suma y se aplica el postulado distributivo.

  $f(a,b,c) = (\overline{a} + \overline{b} + c \bullet \overline{c}) \bullet (c + a \bullet \overline{a} + b \bullet \overline{b}) = (\overline{a} + \overline{b} + c) \bullet (\overline{a} + \overline{b} + \overline{c}) \bullet (a + b + c) \bullet (a + \overline{b} + c) \bullet (\overline{a} + b + c) \bullet (\overline{a} + \overline{b} + c)$

- Paso 4. Eliminar maxtérminos repetidos.

  $f(a,b,c) == (\overline{a} + \overline{b} + c) \bullet (\overline{a} + \overline{b} + \overline{c}) \bullet (a + b + c) \bullet (a + \overline{b} + c) \bullet (\overline{a} + b + c)$

2.4.4.1. Notación M

La *Notación M* introduce una manera simplificada de representar los maxtérminos de una función. En la tabla 2.24 se han representado la lista de maxtérminos de tres variables y la correspondiente notación simplificada.

Tabla 2.24: Lista de los maxtérminos de tres variables, entradas binarias y decimales asociadas y notación M reducida.

$x_1 + x_2 + x_3$	0	0	0	0	M_0
$x_1 + x_2 + \overline{x_3}$	0	0	1	1	M_1
$x_1 + \overline{x_2} + x_3$	0	1	0	2	M_2
$x_1 + \overline{x_2} + \overline{x_3}$	0	1	1	3	M_3
$\overline{x_1} + x_2 + x_3$	1	0	0	4	M_4
$\overline{x_1} + x_2 + \overline{x_3}$	1	0	1	5	M_5
$\overline{x_1} + \overline{x_2} + x_3$	1	1	0	6	M_6
$\overline{x_1} + \overline{x_2} + \overline{x_3}$	1	1	1	7	M_7

Ejemplos. Obtenga las expresiones algebraicas de los siguientes maxtérminos, suponiendo que, para todos los casos, se utilizan 4 variables.

- M_4. El valor binario de la entrada asociada, usando cuatro bits, es 0100, por tanto $M_4 = x_1 + \overline{x_2} + x_3 + x_4$
- M_14. El valor binario de la entrada asociada, usando cuatro bits, es 1110, por tanto $M_{14} = \overline{x_1} + \overline{x_2} + \overline{x_3} + x_4$

Una función de conmutación de n variables puede expresarse de forma resumida utilizando la Notación M.

Ejemplo. La función $f = (\overline{x_1} + \overline{x_2} + \overline{x_3}) \bullet (\overline{x_1} + x_2 + \overline{x_3}) \bullet (x_1 + x_2 + \overline{x_3})$ puede expresarse, de forma más simplificada, sustituyendo la expresión de cada maxtérmino por el correspondiente símbolo en notación M (ver tabla 2.24). Entonces $f = M_7 \bullet M_5 \bullet M_1$, que a su vez se puede expresar como $f = \Pi M(1, 5, 7) = \Pi(1, 5, 7)$.

2.4.5. El segundo teorema de expansión

Se lista en dos apartados.

1. *Cualquier función de conmutación completa de n variables puede expresarse de la forma dada por la ecuación 2.11, donde las funciones* $f(0, x_2, ..x_n)$ *y* $f(1, x_2, ..x_n)$ *se denominan funciones residuo de* f *para* $x_1 = 0$ *y* $x_1 = 1$ *respectivamente.*

$$f(x_1, x_2, \ldots, x_n) = [x_1 + f(0, x_2, \ldots, x_n)] \bullet [\overline{x_1} + f(1, x_2, \ldots, x_n)] \quad (2.11)$$

La validez del apartado (a) del segundo teorema de expansión se hace extensiva a las restantes variables, $x_2, x_3 ... x_n$, y su demostración es muy simple: substituyendo en ambos miembros de la igualdad de 2.11 $x_1 = 0$ en primer lugar, y, posteriormente $x_1 = 1$, se puede comprobar que se cumple la igualdad.

Este teorema permite encontrar una expresión de n variables que describa una función de conmutación, también de n variables, a partir de las expresiones de los residuos de la dicha función.

Ejemplo. Sea $f(a, b, c) = a \bullet [\overline{(a + b \bullet c)} + b]$

La función residuo $f(0, b, c)$ se obtiene haciendo $a = 0$ en la expresión asociada a la función del ejemplo. Como se puede deducir, $f(0, b, c) = 0$. De igual forma, la función residuo $f(1, b, c)$ se obtiene haciendo $a = 1$ en dicha expresión. $f(a, 1, c) = 1 \bullet [\overline{(1 + b \bullet c)} + b] = b$.

Por el apartado (a) del segundo teorema de expansión, $f(a, b, c) = [a + 0] \bullet [\overline{a} + b] = a \bullet b$.

---

2. *Toda función de conmutación completa de n variables puede escribirse de la forma dada por la expresión 2.12 donde* $f(i)$ *es el valor que toma la función* f *para la entrada decimal* i*, y* $M_i(x_1, x_2, .., x_n)$ *el maxtérmino asociado a dicha entrada* i*.*

$$f(x_1, x_2, \ldots, x_n) = \Pi_{i=0}^{2^n - 1} [f(i) + M_i(x_1, x_2, \ldots, x_n)] \quad (2.12)$$

Dicho de otra forma, cualquier función de conmutación completa de n variables puede expresarse como el producto de los términos suma formados por lo que evalúa la función f para cada entrada y el maxtérmino asociado a dicha entrada.

A continuación se demostrará este apartado del segundo teorema de expansión, para el caso concreto de funciones de tres variables.

Sea $f(x_1, x_2, x_3)$ una función de conmutación completa de tres variables. Por el apartado 1 del primer teorema de expansión, la función de conmutación se puede desarrollar por la expresión 2.13

$$f(x_1, x_2, x_3) = [x_1 + f(0, x_2, x_3)] \bullet [\overline{x_1} + f(1, x_2, x_3)] \tag{2.13}$$

A su vez, los residuos de 2.13 son funciones de dos variables que, también, son susceptibles de desarrollarse según el apartado 1 del segundo teorema de expansión (ecuación 2.14).

$$\begin{aligned} f(x_1, x_2, x_3) \quad &= [x_1 + (x_2 + f(0, 0, x_3) \bullet (\overline{x_2} + f(0, 1, x_3)] \\ &\quad [\overline{x_1} + (x_2 + f(1, 0, x_3) \bullet (\overline{x_2} + f(1, 1, x_3)] \\ &= [x_1 + x_2 + f(0, 0, x_3)] \bullet [x_1 + \overline{x_2} + f(0, 1, x_3)] \\ &\bullet [\overline{x_1} + x_2 + f(1, 0, x_3)] \bullet [\overline{x_1} + \overline{x_2} + f(1, 1, x_3)] \end{aligned} \tag{2.14}$$

Las funciones de una variable $f(0, 0, x_3)$, $f(0, 1, x_3)$, $f(1, 0, x_3)$ y $f(1, 1, x_3)$ que aparecen en 2.14, a su vez, pueden desarrollarse según 1 y que sustituidas en 2.14 permiten expresar $f(x_1, x_2, x_3$ según ecuación 2.15.

$$\begin{aligned} f(x_1, x_2, x_3) \quad &= [x_1 + x_2 + x_3 + f(0, 0, 0)] \bullet [x_1 + x_2 + \overline{x_3} + f(0, 0, 1)] \\ &\bullet [x_1 + \overline{x_2} + x_3 + f(0, 1, 0)] \bullet [x_1 + \overline{x_2} + \overline{x_3} + f(0, 1, 1)] \\ &\bullet [\overline{x_1} + x_2 + x_3 + f(1, 0, 0)] \bullet [\overline{x_1} + x_2 + \overline{x_3} + f(1, 0, 1)] \\ &\bullet [\overline{x_1} + \overline{x_2} + x_3 + f(1, 1, 0)] \bullet [\overline{x_3} + \overline{x_2} + \overline{x_3} + f(1, 1, 1)] \\ &= \Pi_{i=0}^{2^3-1} [M_i(x_1, x_2, x_3) + f(i)] \end{aligned} \tag{2.15}$$

El segundo apartado del segundo teorema de expansión da una forma simple, pero compacta, de expresar cualquier función de conmutación de n variables.

Ejemplo. Sea f una función de conmutación completa de tres variables descrita por la tabla de verdad 2.20

El segundo apartado del segundo teorema de expansión permite expresar la función f según la ecuación 2.16

$$\begin{aligned} f(x_1,x_2,x_3) &= [x_1+x_2+x_3+0]\bullet[x_1+x_2+\overline{x_3}+1] \\ &\bullet[x_1+\overline{x_2}+x_3+1]\bullet[x_1+\overline{x_2}+\overline{x_3}+0] \\ &\bullet[\overline{x_1}+x_2+x_3+1]\bullet[\overline{x_1}+x_2+\overline{x_3}+0] \\ &\bullet[\overline{x_1}+\overline{x_2}+x_3+0]\bullet[\overline{x_3}+\overline{x_2}+\overline{x_3}+1] \\ &= [x_1+x_2+x_3]\bullet[x_1+\overline{x_2}+\overline{x_3}] \\ &\bullet[\overline{x_1}+x_2+\overline{x_3}]\bullet[\overline{x_1}+\overline{x_2}+x_3] \end{aligned} \tag{2.16}$$

Puede observarse que f viene expresada como el producto de aquellos maxtérminos asociados a las entradas para las que la función vale 0 en la tabla 2.20

2.4.6. Relación entre las formas canónicas

Se ha visto que cualquier función de conmutación puede describirse como suma de mintérminos o como producto de maxtérminos. Una expresión en suma de mintérminos identifica para qué entradas f toma el valor 1, siendo evidente que, para las restantes entradas, f vale 0. De igual forma, una expresión en producto de maxtérminos identifica para qué entradas f toma el valor 0, por lo que, para las otras, f vale 1. Pues bien, si f es una función que viene expresada en suma de mintérminos, entonces f puede expresarse, de forma directa, como producto de aquellos maxtérminos asociados a las entradas no contenidas en la expresión dada en suma de mintérminos.

Ejemplo.

$$f=\sum(0,1,2,7,10,14,15)=\pi(3,4,5,6,8,9,11,12,13)$$

De igual forma, una función g expresada como producto de maxtérminos puede representarse de forma directa como suma de aquellos mintérminos asociados a las entradas no contenidas en la expresión dada en producto de maxtérminos.

Ejemplo.

$$g = \pi(0, 3, 5, 7) = \sum(1, 2, 4, 6)$$

2.4.7. Complemento de una función de conmutación

El complemento del mintérmino asociado a una entrada es igual al maxtérmino de dicha entrada. Para comprobar esta afirmación basta con aplicar el teorema de Morgan T-7a a un mintérmino dado.

$$\overline{m_i} = M_i \tag{2.17}$$

El complemento del maxtérmino asociado a una entrada es igual al mintérmino de dicha entrada.

$$\overline{M_i} = m_i \tag{2.18}$$

Ejemplo. El mintérmino, m_2, y el maxtérmino, M_2, asociados a tres variables de entrada, se expresan, respectivamente, como $\overline{x} \bullet y \bullet \overline{z}$ y $x + \overline{y} + z$. El complemento del mintérmino m_2 es igual a $\overline{m_2} = \overline{\overline{x} \bullet y \bullet \overline{z}} = \overline{\overline{x}} + \overline{y} + \overline{\overline{z}} = x + \overline{y} + z = M_2$. De igual forma, el complemento del maxtérmino M_2 se representa como $\overline{M_2} = \overline{x + \overline{y} + z} = \overline{x} \bullet \overline{\overline{y} \bullet \overline{z}} = \overline{x} \bullet y \bullet \overline{z} = m_2$

Sea f una función de conmutación completa de n variables expresada como la suma de los mintérminos asociados a las entradas que hacen f igual a 1. La función complemento de f, esto es, $\overline{f}$, se expresa como el producto de los

maxtérminos asociados a aquellas entradas que hacen f igual a 1. De hecho, si $f = \sum m_i$, la función $\overline{f} = \overline{\sum m_i} = \Pi\overline{m_i} = \Pi\ M_i$.

Sea f una función de conmutación completa de n variables expresada como el producto de los maxtérminos asociados a las entradas que hacen f igual a 0. La función complemento de f, esto es, $\overline{f}$, se expresa como la suma de los mintérminos asociados a aquellas entradas que hacen f igual a 0. De igual forma que en párrafo anterior, si $f = \Pi M_i$, la función $\overline{f} = \overline{\Pi M_i} = \sum \overline{M_i} = \sum m_i$.

2.5. Funciones de conmutación incompletas

Las funciones de conmutación incompletas son aquellas que no están definidas para todo el conjunto de combinaciones de entradas. En la tabla 2.25 se ha representado una función de conmutación incompleta. Obsérvese que para las entradas en las que f no está definida se ha puesto una marca (entradas 3 y 5)

Tabla 2.25: Tabla de verdad de una función incompleta de tres variables.

x	y	z	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	-
1	0	0	0
1	0	1	-
1	1	0	0
1	1	1	1

Las funciones de conmutación completas tienen un dominio que abarca todas las combinaciones posibles de las variables de la función, en cambio, las funciones incompletas, al excluir algunas combinaciones de entradas, tienen un dominio menor. Para describir matemáticamente una función incompleta es necesario introducir una función que nos indique el dominio de la misma. Esta función se denomina función **inespecificación** o función **no importa** y se representa por $d(x, y, \ldots)$. Dicha función describe cuáles son las combinaciones de entradas para las que no está definida la función de conmutación.

Ejemplos.

- $F(x, y, z) = \sum(0, 1, 7) + d(3, 5)$.

  La función F toma el valor 1 para las entradas 0,1, y 7, no está definida para las entradas 3 y 5, y toma el valor 0 para las restantes entradas, o sea, las 2, 4 y 6.

- $G(x, y, z) = \Pi(2, 4, 6) \bullet d(3, 5)$.

  La función G, toma el valor 0 para las entradas 2,4, y 6, no está definida para las entradas 3 y 5 y toma el valor 1 para las restantes.

Obsérvese la forma de acompañamiento de la función inespecificación. Para suma de mintérminos aparece en suma y para el producto de maxtérminos, aparece en producto. Esto, en sí, no es más que un criterio de presentación puesto que la **función inespecificación sólo indica para qué entradas no está definida la función**, se exprese ésta como suma de mintérminos o como producto de maxtérminos.

2.6. Representación de funciones

Las funciones de conmutación pueden ser representadas usando, fundamentalmente, los cuatro métodos que se listan a continuación:

1. Expresión de conmutación
2. Tabla de verdad
3. Mapa de Karnaugh (K-mapa)
4. Representación simbólica

Algunas de estas formas, como expresiones de conmutación y tablas de verdad ya se han estudiado con anterioridad y no serán repetidas aquí.

2.6.1. Mapa de Karnaugh (K-mapa)

El mapa de Karnaugh es un diagrama hecho de cuadros, cada uno de los cuales está asociado una entrada y su contenido representa el valor que toma la función para dicha entrada. Su estructura depende del número de variables de la función. A continuación se representan los K-mapas de 2,3,4 y 5 variables.

2.6.1.1. Mapas de 2 variables

Esta formado por cuatro cuadros organizados en dos filas y dos columnas. Los nombres de las variables de entrada se sitúan en la parte superior izquierda del mapa y separadas por una línea inclinada (ver figura 2.1). Los valores que puede tomar la variable x_1 se sitúan en la cabecera de las columnas del K-mapa, mientras que los asociados a la otra variable, x_2, se sitúan a la izquierda de las filas del mapa. El nombre de la función representada se sitúa, habitualmente, debajo del mapa.

La entrada asociada a un determinado cuadro del K-mapa está formada por los valores lógicos situados en la cabecera de la columna y fila a las que pertenece el cuadro. Por ejemplo, el cuadro situado en la parte inferior derecha del K-mapa tiene la entrada asociada $x_1x_2 = (11)$, el del cuadro situado en la parte inferior izquierda $x_1x_2 = (01)$, el de la parte superior derecha, $x_1x_2 = (10)$ y el de la parte superior izquierda la $x_1x_2 = (0,0)$. En el interior de cada cuadro o celda del K-mapa se puede representar, en decimal, la entrada asociada al mismo.

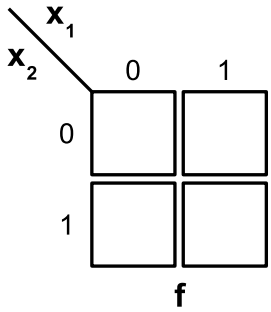


Figura 2.1. K-mapa de dos variables

Ejemplo. Representa en un K-mapa la función $f = x_1 + x_2$. Esta función toma el valor 0 para la entrada $x_1x_2 = (0, 0)$ y 1 para las restantes.

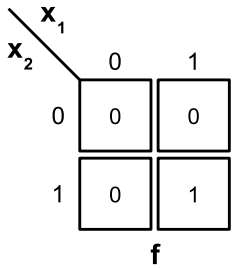


Figura 2.2. Representación de la función $f = x_1 + x_2$

2.6.1.2. Mapas de 3 variables

Están formados por ocho cuadros organizados en cuatro columnas y dos filas. Los nombres de las variables se sitúan en la parte superior izquierda separadas por una línea inclinada. En la parte superior, y horizontalmente, se sitúan todas las combinaciones posibles de las variables x_1x_2 y en la parte izquierda, y verticalmente, las dos combinaciones asociadas a la variable x_3. Es muy importante destacar el hecho de que para las combinaciones de las variables x_1x_2 se ha seguido una codificación de tipo Gray, gracias a la cual, la cabecera de la columna de la izquierda tiene el código 00 y, las siguientes, el 01, el 11 y, el 10. En el próximo capítulo se estudiará las ventajas de este tipo de codificación.

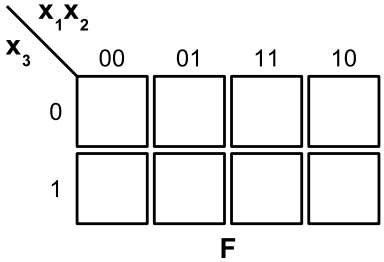


Figura 2.3. K-mapa de tres variables

Ejemplo. Representar la función $f = \sum(0, 4, 6)$

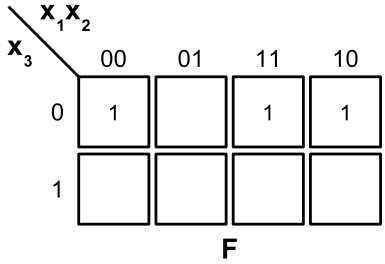


Figura 2.4. Representancián en K-mapa de una función de tres variables

2.6.1.3. Mapas de 4 variables

Están formados por 16 cuadros agrupados en cuatro filas y cuatro columnas. Los nombres de las variables se sitúan de forma idéntica a los casos anteriores y siguiendo la codificación Gray tanto en horizontal como en vertical.

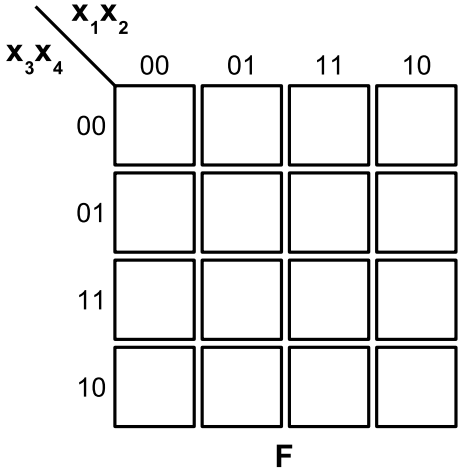


Figura 2.5. K-mapa de cuatro variables

Ejemplo. Representa la función $f = \sum(1, 2, 3) + d(5, 10)$

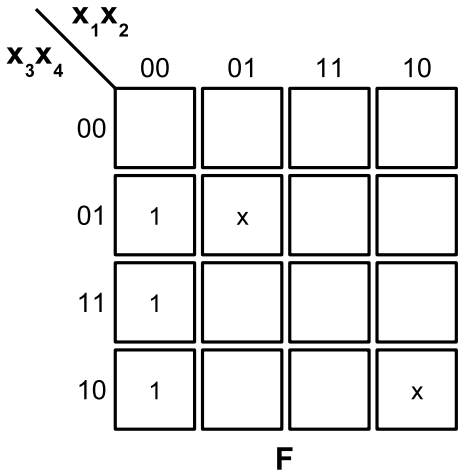


Figura 2.6. Representación de la función $f = \sum(1, 2, 3) + d(5, 10)$

2.6.1.4. Mapas de 5 variables

Están formados por 32 cuadros organizados en ocho columnas y cuatro filas. Los valores asociadas a las entradas que figuran horizontalmente y verticalmente están ordenados siguiendo la codificación Gray de tres bits y 2 bits respectivamente.

2.6.1.5. Propiedad de adyacencia de los K-mapas

La propiedad más importante que tienen los K-mapas se deriva de su construcción. Al utilizar el código Gray para numerar o identificar las filas y columnas del mapa, cada celda o cuadro del mismo heredará la propiedad de aquel por la cual las entradas asociadas a los cuadros adyacentes en horizontal o en vertical a uno dado, solo difieren en un bit.

Sea A_1 la entrada (o grupo de bits) asociada a un cuadro concreto del K-mapa y sean A_2, A_3,... las entradas (o grupos de bits) asociadas a todos los

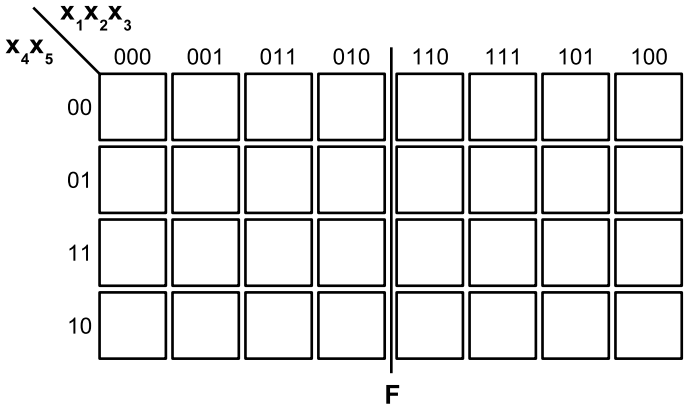


Figura 2.7. K-mapa de cinco variables

cuadros adyacentes al primero en cualquier dirección (horizontal o vertical), entonces, el grupo de bits de las entradas A_2, A_3,..., sólo difieren del de la entrada A_1 en un único bit. Expresado de otra forma, si alteramos un único bit del grupo que forma la entrada A_1 asociada a un cuadro, el nuevo grupo de bits tiene asociado un cuadro del K-mapa que es adyacente horizontalmente o verticalmente al de la entrada A_1.

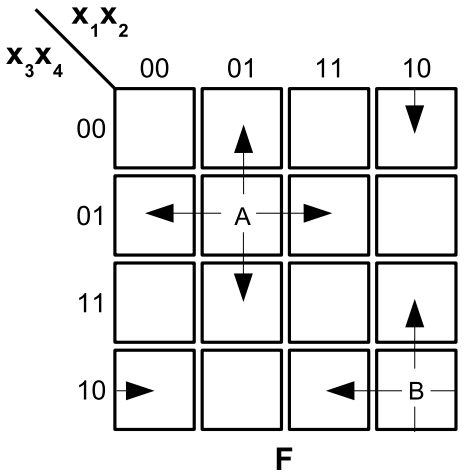


Figura 2.8. Representación de las celdas adyacentes a los cuadros A y B

En la figura 2.8 se ha representado un K-mapa de 4 bits. El cuadro con la letra A, tiene una entrada asociada igual a $(x_1, x_2, x_3, x_4) = (0101)$. Los cuadros

adyacentes al A, son los que contienen las puntas de flechas y tienen entradas asociadas (0100),(0001),(1101), (0111). Puede observarse que estas últimas sólo difieren de la primera en un bit. Las cuatros entradas asociadas a las casillas adyacentes de A se pueden obtener a partir de (0101) modificando un bit cada vez.

La propiedad de adyacencia se lleva más allá de los cuadros interiores al K-mapa. Así los cuadros exteriores tienen adyacencias que se continúan en los cuadros más alejados de él contenidos en la misma fila y columna. Por ejemplo, en la figura 2.8, el cuadro B, cuya entrada asociada es $(x_1, x_2, x_3, x_4) = (1010)$, tiene como cuadros adyacentes aquellos cuyas entradas asociadas son (1000),(1011),(1110),(0010)

En un K-mapa de n variables, cualquier cuadro del mismo tiene exactamente n cuadros adyacentes. En la figura 2.9 se han representado los cincos cuadros adyacentes a uno dado en un K-mapa de 5 variables.

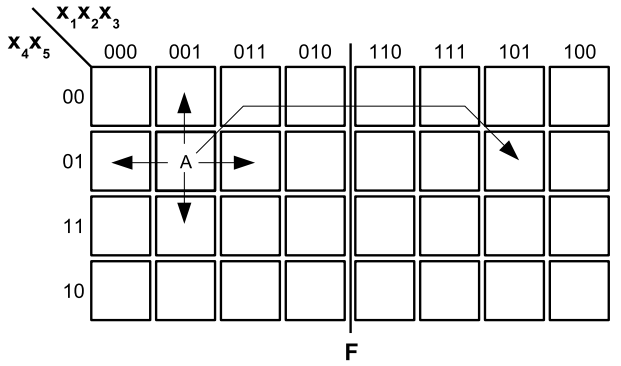


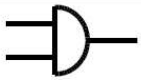
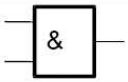
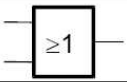
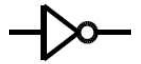
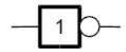
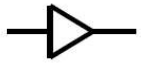
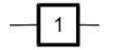
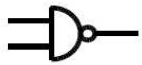
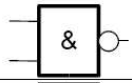
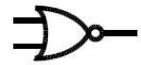
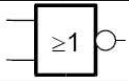
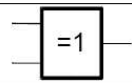
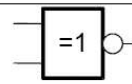
Figura 2.9. Representación de las celdas adyacentes al cuadro A en un K-mapa de 5 variables

Los mapas con cinco variables pueden considerarse constituidos por dos mitades divididas por un eje que separa la columna cuyo código es 010 de la del código 110 y que, individualmente, actúan o tienen las propiedades de sendos mapas de cuatro variables. Cuatro de las celdas adyacentes a una dada, se encuentran en la misma mitad del K-mapa que contiene dicha celda, mientras que, la quinta, se sitúa en la misma fila de la otra mitad, pero en la columna simétricamente distanciada con respecto al eje que divide el mapa, de la celda original.

2.6.2. Representación simbólica

Existen símbolos gráficos que representan las funciones lógicas más comunes AND, OR, NOT, etc. En la tabla 2.26 se muestran los símbolos lógicos más importantes, utilizando la representación clásica y la moderna (símbolo IEEE).

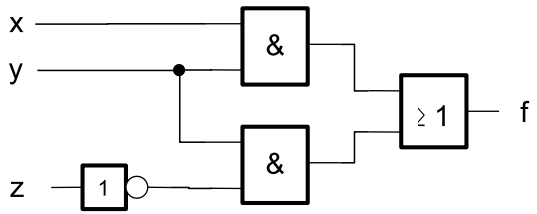
Tabla 2.26: Tabla de operadores lógicos clásicos y normalizados IEEE

Nombre	Símbolo gráfico	Símbolo IEEE
AND		
OR		
NOT		
BUFFER o Seguidor		
NAND		
NOR		
EXOR		
NEXOR o XNOR		

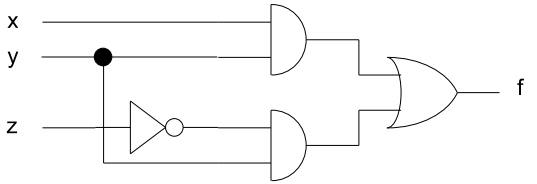
La representación simbólica de las funciones de conmutación se consigue mediante gráficos que representan estos símbolos interconectados.

Ejemplo.

Sea la función de conmutación $f = x \bullet y + \overline{z} \bullet y$, su representación simbólica se muestra en la fígura 2.10:



(a) IEEE



(b) Clásico

Figura 2.10. Representación simbólica de la función $f = x \bullet y + \overline{z} \bullet y$, empleando los componentes de la tabla 2.26.

2.7. Primitivas lógicas: conjuntos completos

*Un **conjunto de operadores** lógicos se considera **completo** si con ellos y las variables binarias oportunas, se puede definir cualquier función de conmutación completa.*

Como se ha visto en apartados anteriores, una función de conmutación completa puede representarse en una suma de mintérminos o en un producto de maxtérminos. Para la primera representación, se necesitan el operador AND y NOT para generar cualquier mintérmino y el operador OR para la suma de estos. Para la segunda representación, se necesitan los operadores OR y NOT para generar cualquier maxtérmino y el operador AND para el producto de estos. En ambos casos, para definir una función de conmutación completa se necesitan los operadores lógicos AND, OR y NOT, por lo que el conjunto de estos tres operadores constituye un conjunto completo. Existen otros mu-

chos conjuntos completos de operadores, algunos de los cuales se mostrarán a continuación.

Si las variables de una función de conmutación se disponen o todas complementadas o todas sin complementar, entonces, dichas variables están en **raíl simple**. En cambio, si las variables de una función de conmutación se disponen tanto complementadas como sin complementar, entontes, dichas variables están en **raíl doble**. Si las variables se disponen en doble raíl no se necesita la utilización de operadores NOT para generar mintérminos o maxtérminos, pero si ellas están en raíl simple, el uso de operadores NOT, es indispensable para la generación de cualquier mintérmino o maxtérmino. Atendiendo a la definición anterior, el conjunto de operadores formado por (AND,OR) constituye un conjunto completo si las variables se disponen en doble raíl.

Otros grupos completos de operadores son los formados por (AND, NOT), (OR, NOT), (NAND) o (NOR). Para comprobar si un grupo de operadores es completo basta con demostrar si, con los operadores de dicho grupo, se pueden construir el conjunto (AND, NOT y OR) que, se sabe, es completo.

- Demostración de que (AND,NOT) forma un conjunto completo

  Si con el conjunto (AND, NOT), se puede construir el operador OR, entonces, dicho conjunto, es completo.

  Sea el operador AND(x,y) = $x \bullet y$, el operador OR(x,y)= $x+y$ y el operador NOT(x) = $\overline{x}$. El operador OR(x,y) se puede contruir mediante la relación de operadores OR(x,y)=NOT(AND(NOT(x),NOT(Y))). En efecto, NOT(AND(NOT(x), NOT(y))) = $\overline{\overline{x} \bullet \overline{y}} = \overline{\overline{x}} + \overline{\overline{y}} = x + y$.

- Demostración de que (OR,NOT) forma un conjunto completo

  Se procede de forma similar a la del apartado anterior.

  El operador que falta puede construirse según la relación AND(x,y) = NOT(OR(NOT(x), NOT(y))). De hecho NOT(OR(NOT(x),NOT(y))) = $\overline{\overline{x} + \overline{y}} = \overline{\overline{x}} \bullet \overline{\overline{y}} = x \bullet y$

- Demostración de que (NAND) forma un conjunto completo

  Si del operador NAND se consigue generar el conjunto de operadores (AND,NOT), entonces sería completo. Sea NAND(x,y) = $\overline{x \bullet y}$

  -NOT El operador NOT(x) = NAND(x,x). De hecho NAND(x,x)= $\overline{x \bullet x} = \overline{x}$

-AND El operador AND(x,y) = NOT(NAND(x,y)). De hecho NOT(NAND(x, y)) = $\overline{\overline{x \bullet y}} = x \bullet y$

El conjunto formado por el operador (NAND) es completo.

- Demostración de que (NOR) forma un conjunto completo

  Tan sólo es necesario demostrar que con NOR se puede construir OR y NOT.

  -NOT El operador NOT(x) = NOR(x,x). De hecho NOR(x,x)= $\overline{x + x} = \overline{x}$

  -OR El operador OR(x,y) = NOT(NOR(x,y)). De hecho NOT(NOR (x, y)) = $\overline{\overline{x + y}} = x + y$

  El conjunto formado por el operador (NOR) es completo.

Ejercicios propuestos

Problema 2.1 Demuestre todos los teoremas del álgebra de conmutación mediante el uso de los postulados del álgebra.

Problema 2.2 Demuestre con el álgebra de conmutación que $xy + \overline{x}z + yz = xy + \overline{x}z$.

Problema 2.3 Exprese en producto de maxtérminos la fórmula de conmutación siguiente: $f = \overline{x} \bullet \overline{[y \bullet \overline{(\overline{z} \bullet x)}]} + x \bullet z$

Problema 2.4 Represente las siguientes funciones de conmutación en tablas de verdad y K-mapas del número de variables que sean necesarios.

1. $f = \sum(1, 2, 6, 7, 12)$
2. $g = \pi(0, 2, 3, 4, 6, 7, 8, 10, 11, 12)d(1, 5)$
3. $h = \sum(3, 4, 7) + d(1)$

Problema 2.5 Obtenga las expresiones en suma de mintérminos y producto de maxtérminos de las funciones de conmutación representadas en la tabla 2.27

Tabla 2.27:

x	y	z	f	g
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	-	1
1	0	0	0	0
1	0	1	-	0
1	1	0	1	0
1	1	1	1	1

Problema 2.6 Sea $f(x, y, z) = \sum(0, 1, 4, 5, 7)$. Construya las expresiones en suma de mintérminos y producto de máxternimos de los residuos $f(0, y, z)$ y $f(1, y, z)$.

Problema 2.7 Verificar que se cumple $f(x,y,z) = \overline{x} \bullet f(0,y,z) + x \bullet f(1,y,z)$ para $f(x,y,z) = \Pi(0,3,7)$.

Problema 2.8 Dadas las funciones $f = \sum(0,1,3,7,12)$ y $g = \overline{\Pi(1,15)}$ calcular $\overline{f}g$ y expresarla en suma de mintérminos.

Problema 2.9 Dibujar los circuitos asociados a las funciones representadas en el problema 2.5. Utilizar simbología clásica e IEEE.

Problema 2.10 Construya una puerta XNOR de cuatro entradas usando puertas XNOR y NOR de dos entradas.

Problema 2.11 Justifica si el operador $\Delta(x,y) = x\overline{y}$ es funcionalmente completo si se disponen de un 0 y de un 1.

Capítulo 3

Análisis y Diseño de Circuitos Combinacionales

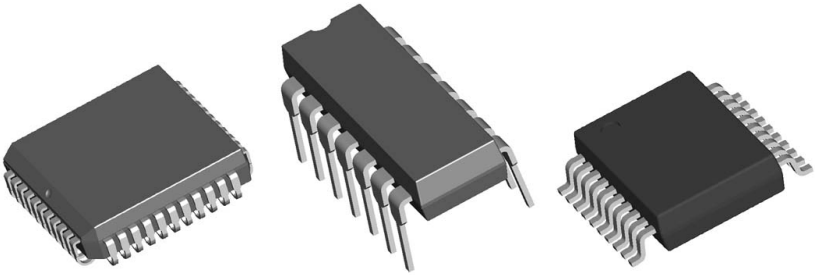
3.1. Puertas y Familias Lógicas

3.1.1. Puertas Lógicas

Las funciones de conmutación estudiadas en el capítulo 2 pueden ser descritas por el conjunto de operadores AND, OR y NOT y variables de entrada. Estos operadores lógicos pueden implementarse físicamente mediante circuitos electrónicos que, en lugar de manejar valores lógicos como el 0 y el 1, utilizan niveles de tensión (voltaje) o corriente (intensidad). *Los circuitos que implementan las operaciones lógicas básicas se denominan PUERTAS LÓGICAS.*

Las puertas lógicas contienen componentes electrónicos como transistores, diodos, etc. y estos se pueden encontrar de forma discreta (distinguibles a simple vista) o integrada en un sustrato de silicio que se ubica en el interior de un chip (circuito integrado o CI). La forma más habitual es la última ya que la primera ocupa gran espacio en comparación con el CI. Los circuitos integrados pueden tener diversos encapsulados: DIP, PLCC, SOJ, etc tal y como aparece en la figura 3.1. Además, los CI presentan otras ventajas:

- Bajo coste
- Bajo consumo
- Alta fiabilidad
- Alta velocidad de operación
- Reducido número de conexiones externas



(a) PLCC (b) DIP (c) SSOP

Figura 3.1. Algunos de los encapsulados con los que se pueden presentar los circuitos integrados.

Se puede establecer una cladifiación de los CI atendiendo a la escala de integración, que representa una medida del número de componentes (según algunos autores) o de puerta lógicas (según otros) integrados en el mismo chip.

Esta clasificación puede ser:

- SSI (Small Scale of Integration). Pequeña escala de integración. Comprende aquellos CI que contienen un número menor a 10 puertas lógicas o menos de 64 componentes.
- MSI (Medium Scale of Integration). Escala de integración mediana. Incluye aquellos CI que contienen un número de puertas inferior a 100 o un número de componentes menor a 2K.
- LSI (Large Scale of Integration). Gran escala de integración. Incluye aquellos CI que contienen un número de puertas inferior 10000 o ún número de componentes menor a 64K.

- VLSI (Very Large Scale of Integration). Escala de integración muy grande y comprende aquellos CI con un número menor de 100000 puertas o con menos de 2M componentes.

- ULSI (Ultra Large Scale of Integration). Escala de integración ultra grande. Aquí se incluyen todos los circuitos con un número mayor de 100000 puertas o más de 2M componentes.

3.1.2. Familias Lógicas

Una misma puerta lógica puede diseñarse internamente con componentes electrónicos diferentes provocando que las características eléctricas de un mismo operador lógico sean distintas. Así, por ejemplo, los valores de tensión e intensidad para las entradas y salida de una puerta AND fabricada con una tecnología pueden diferir de los de otra puerta AND que se haya fabricado con una tecnología diferente. No obstante, en ambos casos, la función lógica que implementa la puerta es la misma.

El conjunto de todos los componentes lógicos que han sido fabricados utilizando la misma tecnología se denomina FAMILIA LÓGICA. En la tabla 3.1 se muestran las principales familias lógicas agrupadas en dos grandes ramas: bipolar y MOS, que dan su nombre por el tipo de transitor que utilizan. Las familias más importantes son la CMOS y la TTL.

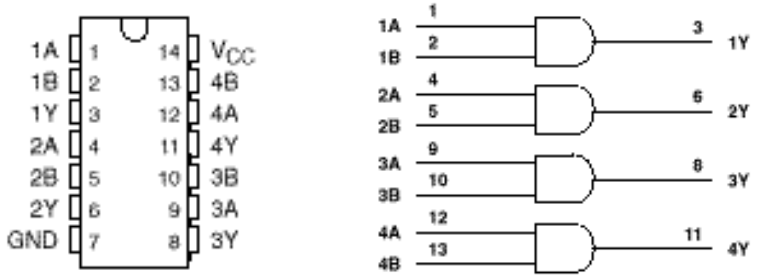
Tabla 3.1: Principales familias lógicas

$Bipolar$	MOS
TTL	pMOS
ECL	nMOS
I^2L	CMOS

Cada familia, a su vez, se divide en una gran variedad de subfamilias o series lógicas. Por ejemplo, en la familia TTL se pueden encontrar las series: 74(TTL estándar), 74H (High Speed), 74L (Low Power), 74S (Schottky), 74LS (Low power Schottky), 74AS (Advanced Schottky), 74ALS (Advanced Low Power Schotkky), 74F (Fast), etc.; y para la familia CMOS las series: 74HC (High Speed CMOS), 74HCT (High Speed CMOS, TTL compatible), 74AC (Advanced CMOS), 74ACT (Advanced CMOS, TTL compatible), 74FCT (Fast CMOS, TTL compatible), etc.

Dentro de cada serie se utiliza una numeración para identificar a un tipo de puerta lógica. Esta numeración se mantiene, incluso, para otras series lógicas. Por ejemplo, el C.I. 74LS00 contiene 4 puertas NAND, el C.I. 74LS04 contiene 6 puertas NOT, el C.I. 74LS08 contiene 4 puertas AND, todas ellas con tecnología TTL Low Schottky. Por su parte, el C.I. 74HC00 contiene 4 puertas NAND, el 74HC04 seis puertas NOT, ambas, con tecnología High Speed CMOS.

La figura 3.2 muestra el patillaje del circuito integrado 74AS08 y las funciones lógicas de las puertas que contiene. Puede observarse la existencia de terminales adicionales a los propios operadores, como son V_{cc} y GND, denominados alimentación y tierra respectivamente, y que suministran la tensión e intensidad necesarias para el funcionamiento del chip. El términal GND se conecta al nivel de tensión bajo (0 lógico), mientras que V_{cc} a 5V (3,3V) o nivel de tensión alto (1 lógico).



(a) Encapsulado (b) Función lógica

Figura 3.2. Descripción del circuito 74AS08.

La documentación técnica de una determinada puerta lógica incluye la tabla de verdad (*function table*) de la misma.

Como se puede observar en la figura 3.3, la tabla de función no contiene ni el valor lógico 0, ni el 1. El fabricante muestra los niveles de tensión de entrada y salida, H para el nivel alto, L para el nivel bajo y X para cualquier nivel de tensión (alto o bajo). Si las entradas A y B tienen un nivel alto de tensión, la salida genera un nivel alto de tensión; si la entrada A o la B tienen un nivel bajo de tensión, L, independientemente del nivel de tensión de la otra entrada, X, la salida mostrará un nivel bajo, L. Obsérvese que el comportamiento de la puerta se corresponde con el de una puerta AND, tan sólo se tienen que

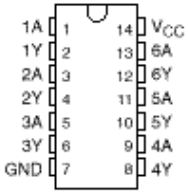
FUNCTION TABLE
(each gate)

INPUTS		OUTPUT
A	B	Y
H	H	H
L	X	L
X	L	L

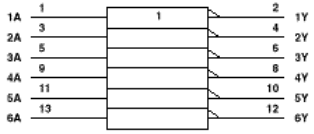
Figura 3.3. Tabla de función de cada puerta lógica del circuito 74AS08

sustituir los valores de tensión alto y bajo, por unos y ceros. Si se usa **lógica positiva**, el nivel de tensión alto representa al 1 lógico (H=1), y el nivel de tensión bajo, al 0 lógico (L=0). Si se usa **lógica negativa**, el nivel de tensión alto es el 0 lógico(H=0), y el nivel de tensión bajo, el 1 lógico (L=1)

En la figura 3.4 se muestran diferentes circuitos junto con sus tablas de función.



(a) Encapsulado

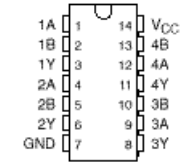


(b) Simbólos

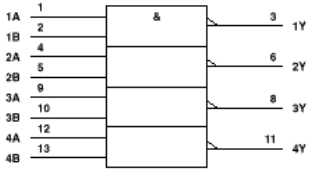
FUNCTION TABLE
(each inverter)

INPUT A	OUTPUT Y
H	L
L	H

(c) Tabla de función



(d) Encapsulado



(e) Simbólos

FUNCTION TABLE
(each gate)

INPUTS		OUTPUT
A	B	Y
H	H	L
L	X	H
X	L	H

(f) Tabla de función

Figura 3.4. Descripción del circuito 74AS04 (HEX NOT) y 74AS00 (QUAD NAND).

3.1.3. Características eléctricas de las puertas lógicas

Los fabricantes de CI incorporan, en la documentación técnica de los chips, las especificaciones eléctricas y temporales de los mismos. En las tablas 3.2 y 3.3 se muestran partes de una hoja de datos de la puerta lógica comercial (74AS04) y de su versión militar (54AS04), que recogen algunos de los parámetros eléctricos más importantes para el diseño electrónico. A continuación se explicarán algunos de estos parámetros.

Tabla 3.2: Condiciones de trabajo recomendadas para el 74AS04

recommended operating conditions

		SN54AS04			SN74AS04			UNIT
		MIN	NOM	MAX	MIN	NOM	MAX	
V_{CC}	Supply voltage	4.5	5	5.5	4.5	5	5.5	V
V_{IH}	High-level input voltage	2			2			V
V_{IL}	Low-level input voltage			0.8			0.8	V
I_{OH}	High-level output current			−2			−2	mA
I_{OL}	Low-level output current			20			20	mA
T_A	Operating free-air temperature	−55		125	0		70	°C

Tabla 3.3: Características eléctricas del 74AS04

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER	TEST CONDITIONS		SN54AS04			SN74AS04			UNIT
			MIN	TYP§	MAX	MIN	TYP§	MAX	
V_{IK}	V_{CC} = 4.5 V,	I_I = −18 mA			−1.2			−1.2	V
V_{OH}	V_{CC} = 4.5 V to 5.5 V,	I_{OH} = −2 mA	V_{CC} −2			V_{CC} −2			V
V_{OL}	V_{CC} = 4.5 V,	I_{OL} = 20 mA		0.35	0.5		0.35	0.5	V
I_I	V_{CC} = 5.5 V,	V_I = 7 V			0.1			0.1	mA
I_{IH}	V_{CC} = 5.5 V,	V_I = 2.7 V			20			20	µA
I_{IL}	V_{CC} = 5.5 V,	V_I = 0.4 V			−0.5			−0.5	mA
I_O¶	V_{CC} = 5.5 V,	V_O = 2.25 V	−30		−112	−30		−112	mA
I_{CCH}	V_{CC} = 5.5 V,	V_I = 0		3	4.8		3	4.8	mA
I_{CCL}	V_{CC} = 5.5 V,	V_I = 4.5 V		14	26.3		14	26.3	mA

§ All typical values are at V_{CC} = 5 V, T_A = 25°C.
¶ The output conditions have been chosen to produce a current that closely approximates one half of the true short-circuit output current, I_{OS}.

3.1.3.1. Tensión o voltaje de alimentación

Se designa como V_{CC} (*Supply voltage*) y recoge los valores mínimo, máximo y nominal, referidos a tierra (GND), que se necesitan en el terminal V_{CC} para que el CI funcione. En la tabla 3.2 y para el integrado ($74AS04$), la tensión nominal de alimentación es de 5V, aunque el fabricante da un margen de valores que aseguran el buen funcionamiento del circuito ($[4{,}5V, 5{,}5V]$). Fuera

del citado margen, el circuito puede no operar o, incluso, deteriorarse.

3.1.3.2. Función de transferencia

La representación del voltaje de salida de la puerta lógica frente al voltaje de entrada no suele ser suministrado por el fabricante, pero nos sirve para entender algunos parámetros eléctricos que sí aparecen en las tablas de características.

Si a una puerta de tipo inversor como el 74AS04, se conecta en su entrada un generador de tensión que pueda variar entre cero y cinco voltios y, en su salida, un medidor, la representación gráfica de los valores que da el medidor para cada tensión generada en su entrada, tendría la forma que se muestra en la figura 3.5.

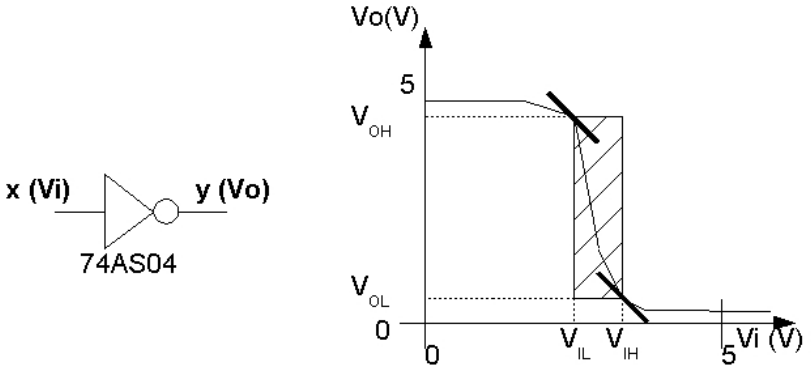


Figura 3.5. Inversor y su función de transferencia.

Como se aprecia en la figura 3.5, para tensiones de entrada bajas (por ejemplo 0 V), la salida muestra un nivel alto de tensión ($\approx 5V$); para tensiones más altas de entrada (p.e. 5V), la salida presenta un nivel bajo de tensión ($\approx 0V$)y para tensiones intermedias existe una región de transición. Salvo para la región de transición, el comportamiento de la puerta se corresponde con el de un inversor lógico, así, para un nivel bajo de tensión (0 lógico) en la entrada, la salida presenta un nivel alto de tensión (1 lógico) y si la entrada tiene un nivel alto (1 lógico), la salida presenta un nivel bajo (0 lógico). No se deben restringir los valores de tensión asociados al cero y uno lógicos a cantidades concretas, sino, más bien, a intervalos de valores para los que el modo de operación del CI se corresponda con el del operador lógico que implementa. Por

lo que muestra la figura 3.5 los límites de esos intervalos se establecen a partir de los puntos de la curva de transferencia $V_o - V_i$ que tiene pendiente -1. Las coordenadas de estos puntos definen los siguientes parámetros:

- V_{IH} Es la mínima tensión de entrada que se considera como un 1 lógico.
- V_{IL} Es la máxima tensión de entrada que se considera como un 0 lógico.
- V_{OH} Es la mínima tensión de salida que se considera como un 1 lógico.
- V_{OL} Es la máxima tensión de salida que se considera como un 0 lógico.

Cualquier tensión de entrada comprendida en el rango $V_i \in [0, V_{IL}]$ es un 0 lógico y genera una tensión de salida comprendida en el rango $V_o \in [V_{OH}, V_{CC}]$ que se considera un 1 lógico.

Cualquier tensión de entrada comprendida en el rango $V_i \in [V_{IH}, V_{CC}]$ es un 1 lógico y genera una tensión de salida comprendida en el rango de $V_o \in [0, V_{OL}]$ que se considera un 0 lógico.

Por último, cualquier tensión de entrada comprendida entre $V_i \in [V_{IL}, V_{IH}]$ genera una tensión de salida comprendida entre $V_o \in [V_{OH}, V_{OL},]$ al que no se le asocia ningún valor lógico (zona de transición).

Para el CI 74AS04, cuyas hojas de características aparecen en la tabla 3.3, los valores tipicos de estos parámetros son: $V_{IL} = 0{,}8V$, $V_{IH} = 2V$, $V_{OH} = 3V$ para $V_{CC} = 5V$ y $V_{OL} = 0{,}35V$ (valor típico). Se destaca el hecho de que los niveles de tensión para los valores lógicos del 0 y del 1 varía de la entrada a la salida. En la figura 3.6 se han dibujado esos rangos.

En la tabla 3.4 se muestra como varían estos parámetros de una serie a otra.

Tabla 3.4: Parámetros V_{IH}, V_{IL},V_{OH} y V_{OL} de varias series lógicas.

	74	**74LS**	**74ALS**	**74F**	**74HC**	**74HCT**	**74AC**
V_{IH}	1,8V	2,2V	2,2V	2V	3,15V	2V	3,25V
V_{IL}	0,6V	0,8V	0,9V	0,8V	1,35V	0,8V	1,65V
V_{OH}	2,2V	2,9V	2,9V	2,7V	3,98V	3,98V	3,94V
V_{OL}	0,2V	0,5V	0,6V	0,5V	0,26V	0,26V	0,36V

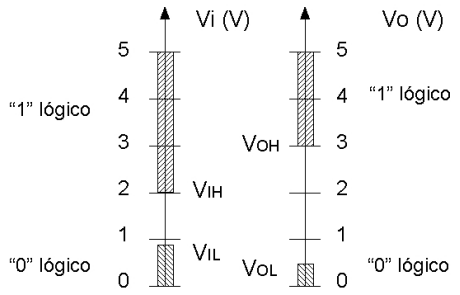


Figura 3.6. Intervalos de tensiones asignadas para un cero y uno lógicos en entrada y salida del inversor 74AS04.

3.1.3.3. Margénes de ruido

En la sección anterior nos encontramos con que los rangos para los unos y ceros lógicos varían dependiendo de si estos son de entrada o salida. Además, para la salida, dichos rangos son más restrictivos que para la entrada. Esta característica no es una casualidad sino una necesidad. En la figura 3.7 se han representado dos inversores conectados en serie (la salida del primer inversor con la entrada del segundo) mediante un cable que es susceptible de recibir voltajes inducidos externos (ruido) que se suman al nivel de tensión que genera el primer inversor. Es deseable que el conjunto de los dos inversores trabaje adecuadamente, a pesar de la existencia de ruido externo, siempre que éste se mantenga por debajo de un cierto nivel.

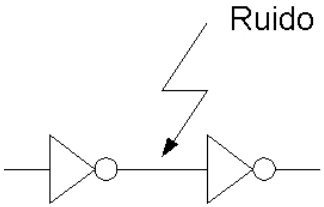


Figura 3.7. Dos inversores en cascada cuya interconexión está sometida a ruido externo.

Si el primer inversor genera un 0 lógico como una tensión dada por V_{OL} y a ésta, se le suma cierto voltaje de ruido, V_N, para que el segundo inversor interprete adecuadamente la entrada como un 0 lógico, su V_{IL} debe ser lo suficientemente grande como para que $V_{OL} + V_N \leq V_{IL}$. Igualmente, si el primer inversor genera un 1 lógico como una tensión dada por V_{OH}, a la que se le suma el ruido, V_N, el receptor debe interpretar correctamente el 1 lógico si $V_{IH} \leq V_{OH} - V_N$

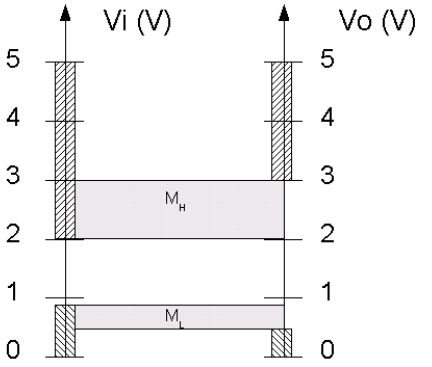


Figura 3.8. Márgenes de ruido.

El margen de ruido representa la máxima amplitud de ruido medida en voltios, bajo la cual una puerta lógica puede funcionar correctamente. Existe un margen para el nivel alto, M_H, o margen de ruido superior y otro para el nivel bajo, M_L o margen de ruido inferior.

- $M_H = V_{OH} - V_{IH}$. Representa la máxima tensión de ruido admisible que no alteraría la interpretación de un uno lógico en la entrada de una puerta lógica.

- $M_L = V_{IL} - V_{OL}$. Representa la máxima tensión de ruido admisible que no alteraría la interpretación de un cero lógico en la entrada de una puerta lógica.

El márgen de ruido, M, de una serie lógica se define como el menor valor entre M_H y M_L. Para el circuito integrado $74AS04$, $M_H = 1V$ y $M_L = 0{,}45V$, por consiguiente, su márgen de ruido es de $M = 0{,}45V$. En la tabla 3.5 se han representado los márgenes de ruido de algunas familias lógicas. Obsérvese que los circuitos CMOS son los que mayor margen de ruido disponen.

Tabla 3.5: Márgenes de ruido de algunas series lógicas.

	74	**74LS**	**74ALS**	**74F**	**74HC**	**74HCT**	**74AC**
M	0,4V	0,3V	0,3V	0,3V	0,83	0,54V	0,29V

3.1.3.4. Corrientes de entrada y salida

Los terminales o pines de los circuitos integrados presentan intensidades (corrientes) que a su vez pueden ser de entrada hacia el interior del circuito (en cuyo caso se consideran positivas) o de salida hacia el exterior del mismo (corrientes negativas).

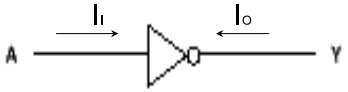


Figura 3.9. Intensidades de entrada y salida de un inversor.

El diseñador electrónico debe tener en cuenta las intensidades de entrada y salida de cada puerta lógica a la hora de interconectarla con otras puertas o con componentes analógicos como LEDS, resistencias, etc. Los fabricantes de suelen suministrar los siguientes parámetros:

- $I_{IH(max)}$ = Es la máxima intensidad que circula por la entrada de la puerta lógica cuando esta tiene un uno lógico.
- $I_{IL(max)}$ = Es la máxima intensidad que circula por la entrada de la puerta lógica cuando esta tiene un cero lógico.
- $I_{OH(max)}$ = Es la máxima intensidad que circula por la salida de la puerta lógica cuando esta tiene un uno lógico.
- $I_{OL(max)}$ = Es la máxima intensidad que circula por la salida de la puerta lógica cuando esta tiene un cero lógico.

La tabla 3.6 recoge los parámetros de intensidad de algunas de las familias lógicas más importantes. Obsérvese que los dispositivos con tecnología CMOS presentan menores intensidades que los de tecnología TTL.

3.1.3.5. Abanico de salida Fan-out

El fan-out define el número máximo de puertas lógicas que pueden conectarse a una puerta dada y se expresa en U.L. (unidades lógicas).

Tabla 3.6: Tabla con los parámetros $I_{IH(max)}$, $I_{IL(max)}$, $I_{OH(max)}$ e $I_{OL(max)}$ de algunas series lógicas.

	74	**74LS**	**74ALS**	**74F**	**74HC**	**74HCT**	**74AC**
I_{IH}	40μA	20μA	20μA	20μA	0,1μA	0,1μA	0,1μA
I_{IL}	-1,6mA	-0,360mA	-0,2mA	-0,6mA	-0,1mA	-0,1mA	-0,1mA
I_{OH}	-0,4mA	-0,4mA	-0,4mA	-1mA	-4mA	-4mA	-24mA
I_{OL}	16mA	8mA	8mA	20mA	4mA	4mA	24mA

Supóngase que, a la salida de un inversor del tipo 74AS04, se conectan un número N de inversores del mismo tipo. Supóngase, también, que la salida del primer inversor (A) es un uno lógico, tal como muestra en la figura 3.10, donde, además, se han representado los sentidos reales de corriente.

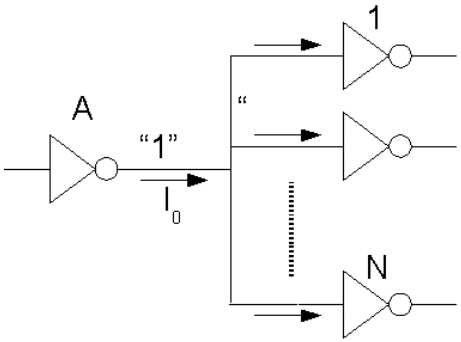


Figura 3.10. Conjunto de N inversores conectados a uno dado que genera un 1 lógico en su salida.

Cuando la puerta A genera un uno lógico en su salida, la intensidad máxima que ésta puede suministrar (por ser negativa) es de $I_{OH} = -2mA$. Por otro lado, los inversores numerados desde el 1 hasta el N, si tienen un uno lógico en sus entradas, reciben, cada uno, una intensidad hacia el interior igual a $I_{IH} = 20\mu A$. Por tanto, el número máximo, N, de puertas que se pueden conectar a una dad que genera un 1 lógico en su salida sería de: $N \times I_{IH} = I_{OH}$. Para el 74AS04, $N = 100$.

La figura 3.11 muestra los sentidos de las corrientes para el supuesto en el que el inversor A genera un cero lógico en su salida. La intensidad máxima que la puerta A es capaz de absorber es de $I_{OL} = 20mA$. Las puertas 1,2, ..N, que tienen un cero lógico en sus entradas son capaces de suministrar un máximo de $N \times I_{IL} = N \times 0{,}5mA$. Por tanto, el número máximo de inversores que pueden conectarse en esta situación al 74AS04 es de $N = 20mA/0{,}5mA =$

40.

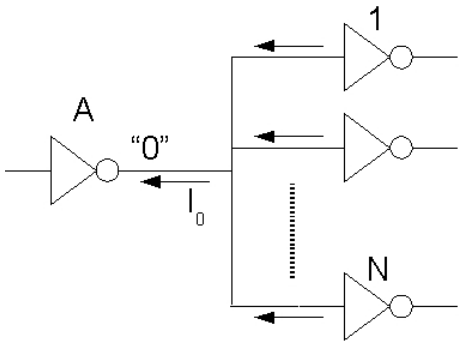


Figura 3.11. Conjunto de N inversores conectados a uno dado que genera un 0 lógico en su salida.

El fan-out es valor más restrictivo de N para los dos supuestos anteriores y que, para la familia 74AS, es de 40 U.L. En la tabla 3.7 se muestran los fan-outs de algunas series lógicas.

Tabla 3.7: Fan-out de algunas series lógicas.

74	**74LS**	**74ALS**	**74F**	**74HC**	**74HCT**	**74AC**
10 UL	20 UL	20 UL	33 UL	40 UL	40 UL	24 UL

3.1.3.6. Disipación de potencia

Es la potencia disipada por el circuito integrado y se mide, normalmente, en mW. Un parámetro asociado es la intensidad I_{cc} o corriente que pasa a través del pin de alimentación V_{cc}. Al igual que las intensidades que circulan por los terminales de las puertas lógicas cambian en función de los niveles lógicos de estos, la intensidad que se suministra por el terminal V_{cc} también lo hace. El fabricante muestra dos valores de I_{cc}: I_{cch} e I_{ccl}, que se corresponden con las intensidades cuando las salidas del circuito integrado están a uno lógico y a cero lógico respectivamente. Tomando el valor medio de I_{cch} e I_{ccl} se puede obtener una buena aproximación del consumo real del integrado.

$$Potencia = V_{cc}\frac{I_{cch} + I_{cch}}{2}$$

3.1.4. Características temporales

Algunos parámetros de gran importancia que miden la **capacidad de respuesta** de los circuitos integrados digitales son los tiempos de propagación o tiempos de retraso.

Figura 3.12. Tiempos de subida y bajada

- t_{LH} = Es el tiempo que tarda la salida de la puerta en pasar desde un nivel bajo de tensión a un nivel alto (tiempo de subida).
- t_{HL} = Es el tiempo que tarda la salida de la puerta en pasar desde un nivel alto de tensión a un nivel bajo (tiempo de bajada).

Los tiempos t_{LH} y t_{HL} se miden desde 10 % al 90 % de la señal de salida.

Aparte de los tiempos de subida y bajada, que definen la duración de la transición en la respuesta de la puerta, se tiene el tiempo de retardo,t_p que mide cuánto tardaría una transición en la entrada de una puerta en reflejarse en su salida. Existen dos modalidades:

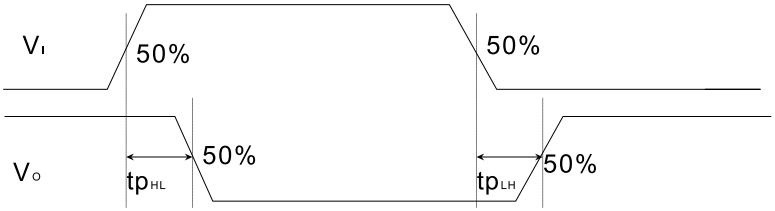


Figura 3.13. Tiempos de propagación

- $t_{p_{LH}}$ = Mide el tiempo transcurrido desde que se produce una transición en la entrada de la puerta, y la salida de la misma comienza a sufrir una transición desde el nivel bajo al nivel alto.

- t_{pHL} = Mide el tiempo transcurrido desde que se produce una transición en la entrada de la puerta, y la salida de la misma comienza a sufrir una transición desde el nivel alto al nivel bajo.

Tal y como se presenta en la figura 3.13, estos tiempos se miden desde el instante en que la señal de entrada (V_i) alcanza el 50 % de su valor máximo, hasta el instante en que la señal de salida (V_o) alcanza, también, el 50 %.

Para el 74AS04, el fabricante da los valores que se resumen en la tabla 3.8

Tabla 3.8: Características temporales de la puerta 74AS04 según suministra su fabricante.

switching characteristics (see Figure 1)

PARAMETER	FROM (INPUT)	TO (OUTPUT)	V_{CC} = 4.5 V to 5.5 V, C_L = 50 pF, R_L = 500 Ω, T_A = MIN to MAX†				UNIT
			SN54AS04		SN74AS04		
			MIN	MAX	MIN	MAX	
t_{PLH}	A	Y	1	6	1	5	ns
t_{PHL}			1	4.5	1	4	

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.

En la mayoría de las situaciones no es necesario diferenciar entre t_{pHL} y t_{pHL} por lo que se da un único valor de propagación t_p que se obtiene a partir del valor medio de los dos valores anteriores.

$$t_p = \frac{t_{pHL} + t_{pHL}}{2}$$

Tabla 3.9: Tiempos de transición y propagación de algunas series lógicas.

	74	**74LS**	**74ALS**	**74F**	**74HC**	**74HCT**	**74AC**
t_{LH}	15ns	13ns	10ns	2.5ns	7ns	7ns	-
t_{HL}	10ns	3ns	6ns	2.5ns	7ns	7ns	-
t_{pLH}	11ns	8ns	4ns	3.7ns	9ns	12ns	5ns
t_{pHL}	7ns	8ns	4ns	3.2ns	9ns	12ns	4.4ns

3.2. Análisis de circuitos combinacionales

Un circuito combinacional es un circuito digital cuyas salidas, en un instante determinado y sin considerar los tiempos de propagación de las puertas, son función, exclusivamente, de la ***combinación*** *de valores binarios de las entradas del circuito en ese mismo instante*. En el circuito combinacional, para cada posible combinación de valores de entrada sólo existe un único valor binario de salida.

El análisis de un circuito combinacional trata de dar el conjunto de funciones lógicas representadas por el mismo o/y su tabla de verdad o/y su K-mapa. En definitiva, definir y expresar, de forma ordenada, los valores que toman las salidas del circuito para todas las posibles combinaciones de valores en sus entradas.

Ejemplo. Analice el circuito de la figura 3.14, constituido por un único chip de puertas NAND.

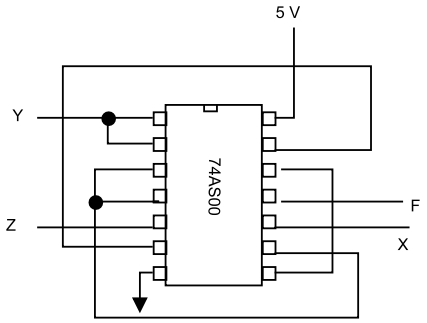
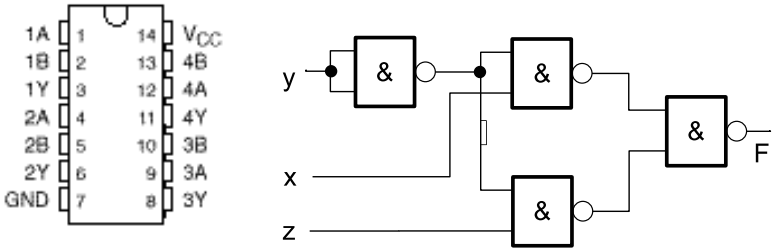


Figura 3.14. Esquema de un circuito digital construido con un único chip 74AS00

A partir del encapsulado del 74AS00 (que se muestra en la figura 3.15.a) y sabiendo que éste está formado por cuatro puertas NAND cuyas entradas están marcadas con las letras A,B y, su salida, con la Y, se puede construir el esquema de circuito (o representación simbólica) que se muestra en la figura 3.15.b.

De la representación inicial mostrada en la figura 3.14, se han eliminado los terminales de alimentación que, aunque son necesarios para que las puertas lógicas del circuito integrado operen correctamente en implementaciones



(a) (b)

Figura 3.15. Encapsulado del 74AS00 (a) y circuito equivalente al de la figura 3.14(b).

reales, no afectan a la descripción de como, el conjunto del cableado y puertas utilizadas, funciona.

A partir de la representación simbólica se obtiene, de forma más fácil, la expresión, y/o la tabla de verdad, que describe el comportamiento de la salida F según los valores de las variables de entrada x,y,z. A modo de ejemplo, se sugiere que el lector obtenga, directamente, la tabla de verdad del circuito de la figura 3.15b . Para conseguirlo, se necesitan determinar los valores lógicos de todos los nodos[1] del esquema para cada combinación de entrada (tabla 3.10).

Tabla 3.10: Tabla de verdad del circuito sometido a análisis.

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Si el lector ha seguido la sugerencia de obtener la tabla de verdad directamente del esquemático del circuito, habrá comprobado que el tiempo invertido en ello es largo y puede imaginar que, si el circuito hubiera tenido cuatro o más variables y/o más puertas lógicas, dicho tiempo sería, aún, mucho mayor.

[1] Entiéndase por nodo a cada extremo (o terminal) de un componente en el esquema.

El análisis de un circuito que viene representado por el conexionado entre los pines de los encapsulados de los chips que lo forman, requiere el conocimiento del patillaje de todos los encapsulados, convirtiendo el análisis en un proceso tedioso que no añade ningún valor a la descripción funcional del circuito. Asimismo, en dichos circuitos aparecen terminales de alimentación y tierra que son fundamentales para que la implementación física funcione, pero que no influyen en el comportamiento lógico de los mismos. Por todas estas razones, los circuitos que se analizarán, a partir de ahora, sólo contienen los símbolos lógicos de las puertas que lo forman y las interconexiones entre ellos.

Se pueden suministrar unos criterios a seguir en el análisis de circuitos combinacionales.

1. Etiquetar o señalar con símbolos las salidas de las puertas que son función de las variables de entrada. Obtener la función de Boole para cada una de estas puertas.
2. Etiquetar con símbolos las salidas de las puertas que son función de puertas etiquetadas o/y variables de entrada.
3. Repetir 2 hasta que se llegue a la salida del circuito. Obtener las funciones de Boole
4. Por sustitución repetida de las etiquetas, obtener la expresión algebraica de la salida en función de las variables de entrada

---

Ejemplo. Analizar el circuito de la figura 3.16

- Paso1: Se han etiquetado la salida del inversor, A, y de la puerta OR, B.
  $A = \overline{a}$
  $B = b + c$
- Paso 2 y 3: Se etiquetan las restantes puertas hasta llegar a F.
  $C = A \bullet b$
  $D = \overline{B \bullet d}$
  $F = C + D$

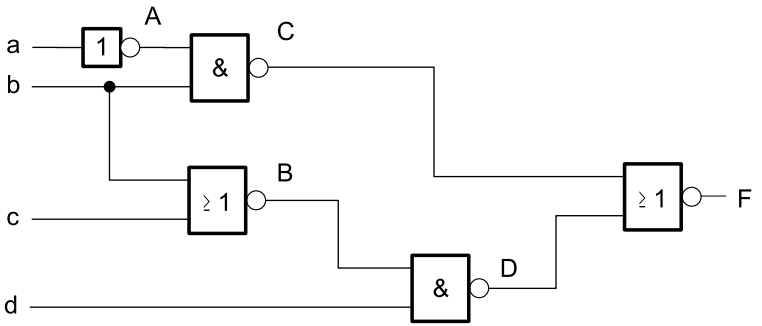


Figura 3.16. Circuito sometido a análisis.

- Paso 4: Se sustituyen en F las etiquetas, de forma sucesiva, hasta conseguir la expresión final.
  $F = A \bullet b + \overline{B \bullet d} = \overline{a} \bullet b + \overline{B} + \overline{d} = \overline{a} \bullet b + \overline{b} \bullet \overline{c} + \overline{d}$

3.2.1. Análisis temporal de circuitos combinacionales

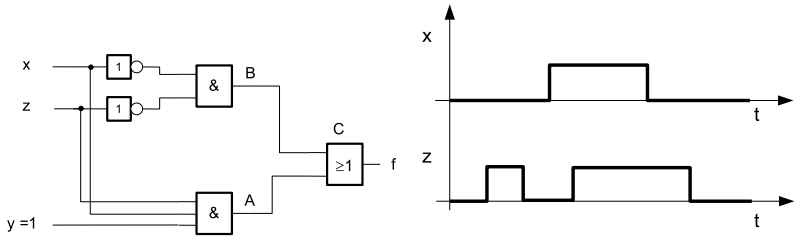
El análisis temporal de la salida (o salidas) de un circuito es una representación de la evolución en el tiempo de los niveles de tensión de dicha salida, para una evolución determinada en las entradas del mismo. El análisis temporal puede abordarse desde diferentes modos que se mostrarán con el siguiente ejemplo.

Ejemplo.

Dadas las secuencias temporales de las variables x,y,z para el circuito de la figura 3.17, se desea obtener la evolución temporal de la salida del mismo.

El análisis lógico del circuito de la figura 3.17 permite expresar $f = \overline{x}\ \overline{z} + x\ y\ z$ cuya tabla de verdad se muestra en la tabla 3.11

En la evolución temporal de las entradas x,y,z que se dan en la figura 3.17.b, se identifican aquellos periodos o intervalos de tiempo para los que las tres entradas simultáneamente permanezcan constantes. La salida del circuito para cada intervalo identificado se obtiene haciendo uso de la tabla 3.11



(a) Circuito (b) Secuencia temporal entradas x, z

Figura 3.17. Esquema y secuencias temporales de entrada del circuito sometido a análisis.

Tabla 3.11: Tabla de verdad del circuito de la figura 3.17a.

x	y	z	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

con los valores de las entradas x,y,z para dicho intervalo (figura 3.18).

También se podría obtener la evolución temporal de la salida sin necesidad de analizar previamente el circuito y obtener su tabla de verdad. A partir de las entradas x, y, z se procede a representar la evolución temporal de cada uno de los nodos del circuito hasta llegar a la salida f. En la figura 3.19 se muestra este método. Obsérvese que éste requiere de un número mayor de representaciones temporales.

En el análisis temporal de este ejemplo no se han contemplado los retrasos o tiempos de propagación de las puertas lógicas t_p, de modo que, cuando existe un cambio de valor en la entrada, se ha supuesto que las puertas lógicas trasladan ese cambio a sus salidas inmediatamente. El análisis temporal de circuitos que contienen puertas lógicas con t_p no nulo, precisa de una resolución nodo a nodo y en la que, los cambios en las entradas de las puertas lógicas provocan transiciones en sus salidas un tiempo t_p posterior. En la fi-

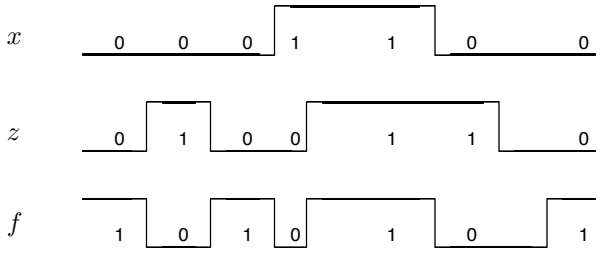


Figura 3.18. Análisis temporal del circuito de la figura 3.17 utilizando la tabla 3.11 para y=1.

gura 3.20 se muestra el procedimiento de resolución. Obsérvese, por ejemplo, que cuando la entrada x pasa de valer 0 a 1, la salida del inversor, $\overline{x}$, pasa de 1 a 0 después de un cierto tiempo.

3.2.1.1. Azares

Al hacer el estudio temporal de la salida de algunos circuitos en los que se contemplan los fenómenos de retrasos en las puertas, puede aparecer discrepancias (a modo de pulsos estrechos) en la salida de los mismos con respecto al estudio lógico. Estos pulsos se denominan azares.

Ejemplo. Supóngase que las puertas NAND, OR y AND del circuito de la figura 3.21 presentan retardos de 20ns, 10ns y 1ns respectivamente. Un estudio lógico del circuito nos indica que $f = ab = \overline{(1 \bullet A)}(0 + A) = \overline{A}A = 0$. No obstante, un análisis temporal del mismo, en el que se incluyan los tiempos de retraso de las puertas lógicas generan la aparición de un pequeño pulso transitorio a la salida. Dicho pulso se conoce como azar.

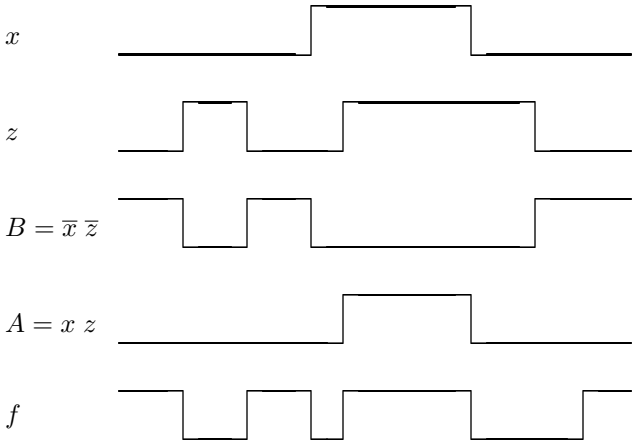


Figura 3.19. Análisis temporal del circuito de la figura 3.17 sin necesidad de obtener la tabla de verdad y para el que se deben determinar la evolución en los puntos A y B del circuito.

3.3. Diseño de circuitos combinacionales

El diseño de circuitos combinacionales trata el problema inverso al análisis: a partir de una especificación inicial, se desean determinar las ecuaciones boolenas (o tabla de verdad) que satisfagan dicha especificación y, de éstas, el esquema del circuito.

Ejemplo. Se desea diseñar un sistema de aviso muy simple para un coche, que debe operar del siguiente modo:

- Si el motor está apagado y las puertas abiertas, sonará una alarma
- Si el motor está encendido y el freno de mano está puesto, también sonará la alarma.

En el diseño de este sistema de aviso, hay dos partes diferenciadas: por un lado la electrónica de medida, que comprende todos los componentes ne-

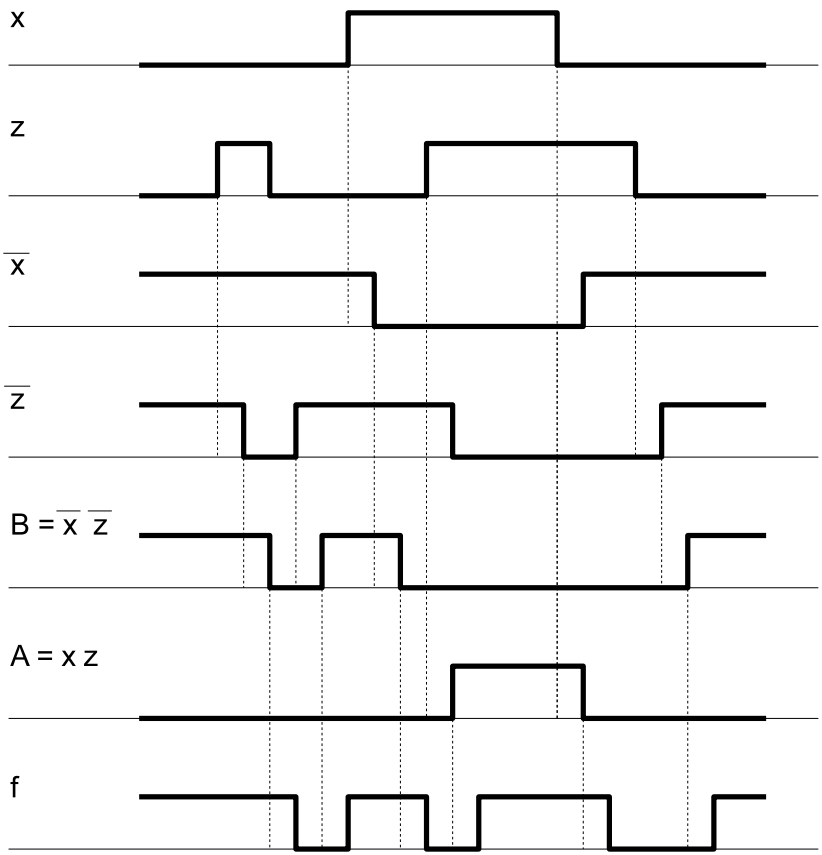


Figura 3.20. Análisis temporal del circuito de la figura 3.17 para puertas lógicas reales con un tiempo de retraso de t_p.

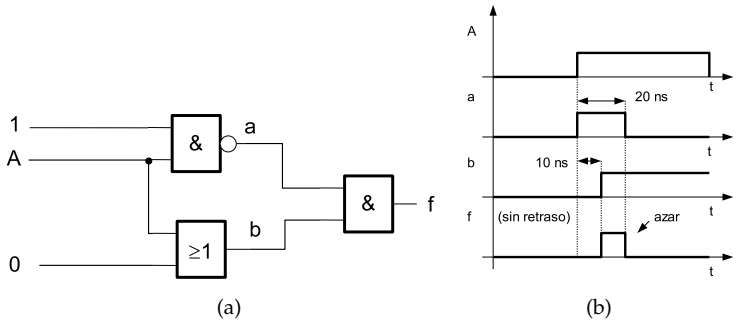


Figura 3.21. Representación de un azar para un circuito real: (a) Circuito (b) Secuencias temporales en los nodos

cesarios para la detección y medición de los parámetros físicos, así como la conversión de las respuestas de estos a unos niveles de tensión adecuados; por otro lado, la electrónica "de decisión" que activa la alarma cuando se cumplen las condiciones expresadas en las especificaciones y que será el objeto de este capítulo.

Independientemente de los sensores utilizados para la detección de si alguna puerta está abierta o no, si el freno de mano está puesto o no, si el motor está encendido o no, podemos suponer que estos parámetros pueden ser tratados como variables binarias que, en función de los valores lógicos que toman, indican el estado asociado al parámetro. Por ejemplo, sea f la variable binaria asociada al estado del freno de mano. Si $f = 1$ se puede suponer que el freno esta puesto y si f=0 que no lo está. Por tanto, sean f,e,p tres variables binarias que indican:

- f: freno de mano. Toma el valor 1 si está puesto y 0 en caso contrario.
- p: Puerta. Toma el valor 1 si alguna de las puertas del coche están abiertas y 0 cuando todas las puertas están cerradas.
- e: Encendido. Toma el valor 1 si el motor está en marcha, 0 si está parado.

El circuito a diseñar, recibe estas tres variables de entrada y genera una señal de salida que debe activar, o no, una alarma. Es evidente que esta alarma puede considerarse también como una señal binaria, A, que toma dos valores posibles: Si A=1 , la alarma se activa, si A=0, la alarma está en silencio.

Llegado a este punto, se puede decir que el diseño del circuito digital se reduce a encontrar la relación existente entre la salida digital A y las variables binarias de entrada f,p,e y para ello, primero, en una tabla de verdad, representamos todas las combinaciones de entrada y, segundo, determinamos el valor que toma la salida A para cada una de ellas.

Tabla 3.12: Tabla de verdad del circuito de alarma.

f	p	e	A
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Según la primera de las dos especificaciones del problema, si el motor está apagado (e=0) y las puertas están abiertas ($p = 1$), entonces se activa la alarma $A = 1$. Todas las combinaciones de entrada del tipo $(f, p, e) = (-, 1, 0)$, donde el guión representa cualquier valor lógico para f, hacen $A = 1$. De la segunda especificación, si el motor está encendido ($e = 1$)y el freno de mano puesto ($f = 1$) también sonará la alarma, se tiene que, para las combinaciones de entrada $(f, p, e) = (1, -, 1)$, la salida $A = 1$.

A partir de la tabla de verdad 3.12, se obtiene la expresión en suma de mintérminos 3.1.

$$A = f \bullet p \bullet e + f \bullet p \bullet \overline{e} + f \bullet \overline{p} \bullet e + \overline{f} \bullet p\overline{e} \tag{3.1}$$

Y de la ecuación 3.1, el circuito equivalente representado en la figura 3.22.

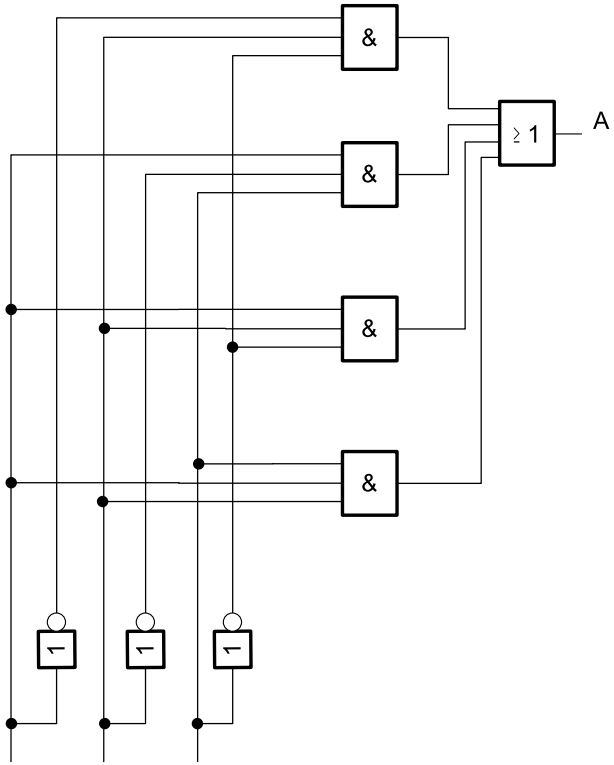


Figura 3.22. Circuito del sistema de aviso para el coche.

3.3.1. Minimización de circuitos combinacionales: Ideas generales

Uno de los objetivos del diseño lógico es el desarrollo óptimo del circuito para lo que deben considerarse algunos factores en la evaluación del producto final:

- Coste: Depende del valor de los componentes, el coste de construcción de las puertas y el mantenimiento del producto final.
- Fiabilidad: Para conseguir circuitos fiables pueden seguirse dos estrategias diferentes no excluyentes la una de la otra:
  1. Utilización de componentes más fiables.
  2. Utilización elementos redundantes que entren en funcionamiento cuando algunos componentes fallen.
- Tiempo de conmutación: Este factor mide la rapidez de respuesta del circuito a los cambios en los valores lógicos de sus entradas.

Todos estos factores deben tenerse en cuenta a la hora de diseñar un circuito combinacional. Por desgracia, no existe ningún método sistemático que englobe a todos ellos, por lo que normalmente es, la experiencia del diseñador, la que consigue la obtención de un circuito con un coste, fiabilidad y tiempo de conmutación razonables con los requisitos del cliente. No obstante, en el caso de que algunos factores se consideren de mayor peso que otros, si se pueden aportar unos criterios o métodos sistemáticos para conseguir el circuito de conmutación óptimo.

En primer lugar, asumimos que el retraso de conmutación debe ser mínimo y se sabe que este depende de la tecnología empleada y del número de niveles de puertas lógicas que tenga el circuito final. Si todos los componentes son de la misma tecnología, el retraso de propagación depende, entonces, del número de niveles. Como toda función de conmutación puede representarse en forma canónica, cualquier circuito puede realizarse usando, como mínimo, dos niveles de puertas lógicas (siempre que las variables de entrada se dispongan en doble raíl) y, por tanto, no será posible disminuir más el tiempo de retraso.

En segundo lugar, supongamos que el coste total de los componentes es el factor de peso para el diseño de circuitos de conmutación. El *coste en puertas*(si asumismos constante el valor monetario de cada una de ellas) es proporcional al número de puertas necesarias para la realización del circuito.

En la figura 3.23 se muestra un circuito equivalente al de la figura 3.22 que representa su misma función de conmutación (se deja al lector la comprobación). Para el circuito de la figura 3.23 se han empleado cuatro puertas AND, una puerta OR y tres puertas NOT, para obtener el complemento de algunas variables de entrada (si el sistema hubiera dispuesto de doble raíl, no hubieran sido necesarios los inversores). El coste en puertas es de ocho para el caso de variables en simple raíl o de cinco si estas están en doble raíl. Para el circuito de la figura 3.23, se han utilizado dos puertas AND, una OR y un inversor. Por consiguiente, en simple raíl, el coste total es de cuatro puertas y, en doble raíl, de tres puertas.

Como los circuitos de las figuras 3.23 y 3.22 son representaciones simbólicas de expresiones algebraicas, es obvio que, a partir de estas, sin necesidad de dibujar el circuito equivalente, se puede obtener el coste de ellos. A partir de una expresión en suma de productos (producto de sumas), el coste en puertas si las variables están en doble raíl será igual al número de términos productos (términos sumas) más una unidad. Al resultado anterior se le suman el número de variables (que no de literales) que aparecen complementadas si se desea el coste en simple raíl.

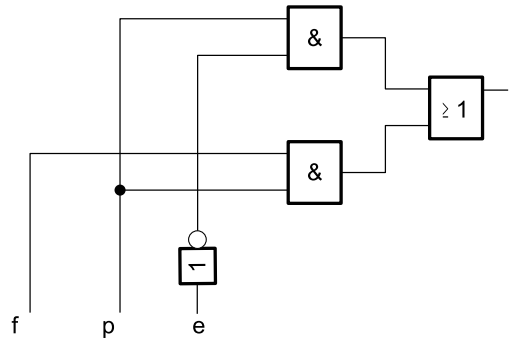


Figura 3.23. Circuito equivalente al de la figura 3.22.

Otro factor asociado al coste es el de interconexión, que recoge el número de conexiones a realizar entre las variables, las puertas lógicas y la salida. Se determina como la suma del número de literales de la expresión y del coste en

puertas.

El problema de encontrar una expresión de conmutación que satisface algún criterio de optimización se denomina MINIMIZACIÓN. A continuación se desarrollará técnicas de diseño mínimo en redes de dos niveles de puertas lógicas con salida simple. Se definirá una expresión de conmutación mínima como aquella expresión que representa una función de conmutación que se puede construir con el menor número de puertas e interconexiones posibles.

Antes de desarrollar las técnicas para obtener funciones de conmutación mínimas es útil destacar algunas propiedades de las funciones.

3.3.2. Implicantes primas y expresiones irredundantes

Consideremos dos funciones de n variables: f_1 y f_2. Se dice que la función f_1 implica a f_2 si no existe asignación de valores a las n variables de entrada que hagan f_1 igual a 1 y f_2 igual a 0. Esto es, f_2 se hará 1 para aquellas entradas que hacen f_1 igual 1 y para las entradas en que f_2 toma el valor 0, f_1 también vale 0. Cuando f_1 implica a f_2 se puede decir de forma equivalente que f_2 incluye o cubre a f_1.

La definición anterior referida a funciones también es extensible a términos, ya que estos, en definitiva, describen funciones de conmutación.

Ejemplos.

- Sean $f_1(x,y,z) = xy + yz$ y $f_2(x,y,z) = x\ y + y\ z + \overline{x}\ z$ dos funciones de conmutación completas representadas en la tabla 3.13. Según se aprecia, f_1 implica a f_2, porque todas las entradas que hacen 1 a f_1, también hacen 1 a f_2 (obsérvese que para las entradas en las que f_2 vale 0, también lo vale f_1).

- Sean $f_3(x,y,z) = (x+y)(y+z)(\overline{x}+z)$ y $f_4(x,y,z) = (x+y)(y+z)$ dos funciones de conmutación completas representadas en la tabla 3.14. Se puede observar que f_3 implica a f_4 ya que todas las entradas que hacen que f_3 valga uno, también lo hacen con f_4.

Tabla 3.13: Tabla de verdad de las funciones f_1 y f_2

x	**x**	**z**	f_1	f_2
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Tabla 3.14: Tabla de verdad de las funciones f_3 y f_4

x	**x**	**z**	f_3	f_4
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	0	1
1	1	1	1	1

Un término t_1 se dice subsuma de otro término t_2, sii los literales del término t_2 son también literales del término t_1. Expresado de forma equivalente, los literales de t_2 deben estar contenidos en t_1. Si t_1 es subsuma de t_2 se dice también que t_2 cubre a t_1.

Ejemplos.

- Sean los términos $t_1 = x\ \overline{y}\ \overline{z}$ y $t_2 = x\ \overline{z}$. El término t_1 es subsuma de t_2 porque los literales de t_2 están contenidos en t_1.
- Sean los términos $t_3 = x + \overline{y} + \overline{z}$ y $t_4 = x + \overline{z}$.El término t_3 es subsuma de $t4$ porque los literales de t_4 están contenidos en t_3.

3.3.2.1. Implicantes

Se dice que un término producto es una ***implicante*** *de una función f, si el término producto implica a la función sobre su dominio.*

Ejemplo. Sea la función de conmutación f(x,y,z)=$\sum m(0,1,3,5) + d(2,6)$. Se desea determinar si los términos $t_1 \equiv \overline{x}$, $t_2 \equiv \overline{y}\ z$ y $t_3 \equiv y\ z$, son implicantes de la función f.

Se ha representado en la tabla 3.15 la función ejemplo con los términos t_1, t_2 y t_3. En concreto, el término t_1 es implicante de la función f puesto que, para todas las entradas que hacen t_1 igual a 1 (excluyendo aquellas que se encuentran fueran del dominio de f), la función f se hace 1. Del mismo modo, el término t_2 también es implicante. En cambio, el término producto t_3 no es implicante puesto que para la entrada $(x, y, z) = (1, 1, 1)$ dicho término vale 1 y la función f toma el valor de 0.

Tabla 3.15: Tabla de verdad de la función $f = \sum(0,1,3,5) + d(2,6)$ y los términos $t_1 \equiv \overline{x}$, $t_2 \equiv \overline{y}\ z$ y $t_3 \equiv yz$.

x	**y**	**z**	**f**	t_1	t_2	t_3
0	0	0	1	1	0	0
0	0	1	1	1	1	0
0	1	0	-	1	0	0
0	1	1	1	1	0	1
1	0	0	0	0	0	0
1	0	1	1	0	1	0
1	1	0	-	0	0	0
1	1	1	0	0	0	1

3.3.2.2. Implicantes primas

Una implicante de la función f que no sea subsuma de otra implicante de la misma funcion que contenga menos literales, se denomina implicante prima.

Una implicante prima es un término producto que implica a la función en su dominio y del que, si quitamos algún literal de dicho producto, el término resultante no implica a la función.

Ejemplo.

Siguiendo con la función $f(x, y, z) = \sum m(0, 1, 3, 5) + d(2, 6)$ del ejemplo anterior, se desea determinar si, las implicantes $t_1 \equiv \overline{x}$ y $t_2 \equiv \overline{y}\ z$, son primas.

- El término $t_1 \equiv \overline{x}$ representa una implicante prima de la función f. A menos que la función f sea la función constante igual a 1 (que no lo es), al término t_1 no se le puede quitar ningún literal, por tanto, no es subsuma de otro término que implique a la función en su dominio.
- Si al término $t_2 \equiv \overline{y}z$ le quitamos un literal nos quedan dos posibles términos productos: $t_{2_1} \equiv \overline{y}$ y $t_{2_2} \equiv z$, que se han representado en la tabla 3.16. En concreto, el término t_{2_1} no es una implicante de la función porque para la entrada $(x, y, z) = (1, 1, 1)$ el término toma el valor 1 y la función el 0. Del mismo modo, el término t_{2_2} tampoco es implicante de la función: analice la entrada $(x, y, z) = (1, 1, 1)$. En consecuencia, el término t_2, es una implicante prima.

3.3.2.3. Suma irredundante o expresión disyuntiva irredundante

Se sabe que el coste asociado a una expresión que describe una función de conmutación disminuye conforme se eliminan literales de la fórmula. Con las implicantes primas se están consiguiendo términos productos que impliquen a la función con el menor número de literales posible. Entonces, una expresión normal disyuntiva mínima estará formada por una suma de implicantes primas.

Tabla 3.16: Tabla de verdad de la función $f = \sum(0,1,3,5) + d(2,6)$ y los términos $t_{2_1} \equiv \overline{y}$ y$t_{2_2} \equiv z$.

x	**y**	**z**	**f**	t_{2_1}	t_{2_2}
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	-	1	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	-	1	0
1	1	1	0	1	1

Una suma irredundante es una expresión normal disyuntiva que describe una función de conmutación que cumple:

1. Todo término producto en la expresión es implicante prima.
2. Ningún término producto puede ser eliminado de la expresión sin que cambie la función descrita por la fórmula.

Ejemplos.

- Sea la función de conmutación $f(x,y,z) = \sum(0,1,3,5) + d(2,6)$. Determinar si la expresión $f(x,y,z) = \overline{x} + \overline{y}\ z$ es una suma irredundante.

  Se ha demostrado anteriormente que los términos $t_1 \equiv \overline{x}$ y $t_2 \equiv \overline{y}\ z$ son Implicantes Primas y se puede comprobar que, la suma de estas, evaluadas en aquellas entradas para las que la función está definida, representan a la función f (suma de las columnas t_1 y t_2 de la tabla 3.15). Si seretira alguna de las implicantes primas de la suma, la expresión resultante no representa a la función f. Por todo esto, se puede afirmar que $f(x,y,z) = \overline{x} + \overline{y}\ z$ es una suma irredundante y no existe ninguna otra expresión de menor coste que implemente dicha función.

- Supongamos que la expresión $f(a,b,c) = \overline{a}\ \overline{b} + \overline{b}\ \overline{c} + \overline{a}\ c$ está compuesta por todos los implicantes primos de f. Se trata de determinar si esa suma de implicantes primas es irredundante. En la tabla 3.17 se han representado la función f junto con cada una de las implicantes primas

que aparecen en la expresión. Se puede observar que, con la suma de las dos implicantes primas representadas en las columnas de la derecha de la tabla, hubiera bastado para definir a la función f. Por consiguiente, la suma irredundante estaría formada por: $f = \overline{b}\,\overline{c} + \overline{a}c$.

Tabla 3.17: Tabla de verdad de la función $f(a,b,c) = \overline{a}\,\overline{b} + \overline{b}\,\overline{c} + \overline{a}\,c$, junto con los términos productos que la componen.

a	**b**	**c**	**f**	$\overline{a}\,\overline{b}$	$\overline{b}\,\overline{c}$	$\overline{a}\,c$
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	0	0	0	0
0	1	1	1	0	0	1
1	0	0	1	0	1	0
1	0	1	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	0	0

3.3.2.4. Obtención de las implicantes primas de una función

Las implicantes primas de una función se obtienen a partir de los mintérminos de la misma siguiendo un procedimiento exhaustivo que se describe con el siguiente ejemplo.

Ejemplo.

Encontrar todos las implicantes primas de la función $f = \sum(0, 1, 3, 5, 6)$+d(2).

- A partir del mintérmino $m_0 = \overline{x}\ \overline{y}\ \overline{z}$, que es implicante de la función f, se obtienen las siguientes subsumas: $\{\overline{x}\ \overline{y}\}$, $\{\overline{x}\ \overline{z}\}$, $\{\overline{y}\ \overline{z}\}$,$\{\overline{x}\}$, $\{\overline{y}\}$ y $\{\overline{z}\}$ que se han representado en la tabla 3.18. Como puede observarse ,algunas subsumas no son implicantes de la función ($\{\overline{y}\ \overline{z}\}$, $\{\overline{y}\}$ y $\{\overline{z}\}$) y de las restantes subsumas, la $\{\overline{x}\}$ es la que implica a la función con el menor número de literales, o sea, la implicante prima.

Tabla 3.18: Tabla de verdad de la función $f = \sum(0,1,3,5,6) + d(2)$ junto con los términos subsumas del mintérmino m_0.

x	y	z	f	$\overline{x}\,\overline{y}$	$\overline{x}\,\overline{z}$	$\overline{y}\,\overline{z}$	$\overline{x}$	$\overline{y}$	$\overline{z}$
0	0	0	1	1	1	1	1	1	1
0	0	1	1	1	0	0	1	1	0
0	1	0	-	0	1	0	1	0	1
0	1	1	1	0	0	0	1	0	0
1	0	0	0	0	0	1	0	1	1
1	0	1	1	0	0	0	0	1	0
1	1	0	1	0	0	0	0	0	1
1	1	1	0	0	0	0	0	0	0

- Los mintérminos m_1 y m_3 son subsumas de la implicante prima $\{\overline{x}\}$, por lo que, a partir de ellos, no se obtendrá ninguna implicante adicional.
- Del mintérminо $m_5 = x\overline{y}z$ se obtienen las subsumas $\{x\ \overline{y}\}$, $\{x\ z\}$, $\{\overline{y}\ z\}$,$\{x\}$, $\{\overline{y}\}$ y $\{z\}$ que se han representado en la tabla 3.19. Obsérvese que el único término subsuma que implica a la función es el $\{\overline{y}\ z\}$ y, por consiguiente, implicante prima.

Tabla 3.19: Tabla de verdad de la función $f = \sum(0,1,3,5,6) + d(2)$ junto con los términos subsumas del mintérmino m_5.

x	y	z	f	$x\,\overline{y}$	$x\ z$	$\overline{y}\ z$	x	$\overline{y}$	z
0	0	0	1	0	0	0	0	1	0
0	0	1	1	0	0	1	0	1	1
0	1	0	-	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	1
1	0	0	0	1	0	0	1	1	0
1	0	1	1	1	1	1	1	1	1
1	1	0	1	0	0	0	1	0	0
1	1	1	0	0	1	0	1	0	1

- Del mintérmino $m_6 = xy\overline{z}$ se obtienen las subsumas $\{xy\}$, $\{x\ \overline{z}\}$, $\{y\ \overline{z}\}$, $\{x\}$, $\{y\}$ y $\{\overline{z}\}$ y que se han representado en la tabla 3.20. La única subsuma que implica a la función es $\{y\ \overline{z}\}$ y, por tanto, implicante prima.

A partir de los mintérminos de la función se han obtenido tres implicantes primas: $\{y\ \overline{z}\}$, $\{\overline{y}\ z\}$ y $\{\overline{x}\}$. Se deja al lector la comprobación de que la suma irredundante es: $f(x,y,z) = \overline{x} + y\ \overline{z} + \overline{y}\ z$.

Tabla 3.20: Tabla de verdad de la función $f = \sum(0,1,3,5,6) + d(2)$ junto con los términos subsumas del mintérmino m_6.

x	**y**	**z**	**f**	$x\ y$	$x\ \overline{z}$	$y\ \overline{z}$	x	y	$\overline{z}$
0	0	0	1	0	0	0	0	0	1
0	0	1	1	0	0	0	0	0	0
0	1	0	-	0	0	1	0	1	1
0	1	1	1	0	0	0	0	1	0
1	0	0	0	0	1	0	1	0	1
1	0	1	1	0	0	0	1	0	0
1	1	0	1	1	1	1	1	1	1
1	1	1	0	1	0	0	1	1	0

3.3.3. Implicadas primas y productos irredundantes

- Un término suma se dice implicada de una función f, si la función implica al término en el dominio de aquella. En otras palabras, una implicada es un término suma que, para los valores de entrada que pertenecen al dominio de la función f y que hacen que dicho término valga cero, la función f también vale cero.
- Una implicada prima de una función es una implicada cuyas subsumas no son, a su vez, implicadas de la misma función con un menor número de literales.

Ejemplo.

Para la función $f(x,y,z) = \Pi(0,1,3,5)d(2,6)$, determine si los términos $t_4 \equiv x + y$, $t_5 \equiv x$, $t_6 \equiv y$ y $t_7 \equiv y + \overline{z}$, son implicadas de f y, a su vez, cuales de ellas son primas.

El término $t_4 \equiv x+y$ es implicada de la función, porque para las entradas $(x,y,z) = \{(001),(000)\}$, t_4 evalúa 0 y la función también. El término $t_5 \equiv x$ es implicada de la función porque para las entradas $(x,y,z) = \{(011),(001),(000)\}$ evalúa 0 y la función también. El término $t_6 \equiv y$ no es implicada de la función porque para la entrada 4 el término vale 0 y la función 1. Por último, el término $t_7 \equiv y + \overline{z}$ es implicada porque para las entradas $(x,y,z) = \{(001),(101)\}$ evalúa 0 y la función también.

- La implicada $t_4 \equiv x + y$ es subsuma de los términos $t_5 \equiv x$ y $t_6 \equiv y$. Como t_5 es implicada de la función entonces, t_4, no es implicada prima.

Tabla 3.21: Tabla de verdad de la función $f(x,y,z) = \Pi(0,1,3,5)d(2,6)$, junto con los términos sumas $t_4 \equiv x+y$, $t_5 \equiv x$, $t_6 \equiv y$ y $t_7 \equiv y+\overline{z}$.

x	**y**	**z**	**f**	t_4	t_5	t_6	t_7
0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0
0	1	0	-	1	0	1	1
0	1	1	0	1	0	1	1
1	0	0	1	1	1	0	1
1	0	1	0	1	1	0	0
1	1	0	-	1	1	1	1
1	1	1	1	1	1	1	1

- A la implicada t_5 no se le puede retirar ningún literal más (a no ser que la función fuera la constante 0), por lo que es implicada prima.
- La implicada $t_7 \equiv y+\overline{z}$ es subsuma de los términos $t_6 \equiv y$, que no es ni siquiera implicada, y $t_9 \equiv \overline{z}$ que tampoco es implicada de la función como el lector podrá comprobar. Por tanto, t_7, es implicada prima.

- Un producto irredundante es una expresión normal conjuntiva que describe una función de conmutación que cumple:

  1. Todo término suma en la expresión es implicada prima.
  2. Ningún término suma puede ser eliminado de la expresión sin que cambie la función descrita por la fórmula.

Ejemplo. Encuentre el producto irredundante o producto mínimo para la función $f(x,y,z) = \Pi(0,1,3,5)d(2,6)$ representada en la tabla 3.21, .

Esta función tiene dos implicadas primas: t_5 y t_7 por lo que función puede expresarse como $f(x,y,z) = x(y+\overline{z})$, siendo este el producto mínimo. El producto mínimo asegura que la función que lo representa está expresada con el menor número de literales posibles, y por tanto, el circuito asociado, tiene un coste mínimo.

- Las implicadas primas de una función de conmutación determinada se obtienen a partir de los maxtérminos de dicha función, siguiendo un procedimiento similar al descrito en el apartado de las implicantes.

Ejemplo.

Obtenga todas las implicadas primas de la función $f = \Pi(0,1)d(2,5)$.

- El maxtérmino $M_0 = x + y + z$, que es implicada de la función f, tiene como subsumas: $\{x + y\}$,$\{x + z\}$,$\{y + z\}$,$\{x\}$,$\{y\}$ y $\{z\}$, las cuales, junto con la función f, se han representado en la tabla 3.22. Puede observarse que los términos $\{x+y\}$ y $\{x+z\}$ son implicadas primas de la función.

Tabla 3.22: Tabla de verdad de la función $f = \Pi(0,1)d(2,5)$ junto con los términos subsumas del maxtérmino M_0.

x	**y**	**z**	**f**	$x+y$	$x+z$	$y+z$	x	y	z
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	0	0	1
0	1	0	-	1	0	1	0	1	0
0	1	1	1	1	1	1	0	1	1
1	0	0	1	1	1	0	1	0	0
1	0	1	-	1	1	1	1	0	1
1	1	0	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1

- Las subsumas del maxtérmino $M_0 = x + y + \overline{z}$ son: $\{x + y\}$,$\{x + \overline{z}\}$,$\{y + \overline{z}\}$,$\{x\}$,$\{y\}$ y $\{\overline{z}\}$ y se han representado en la tabla 3.23. Las implicadas primas resultantes son: $\{x + y\}$ y $\{y + \overline{z}\}$

Tabla 3.23: Tabla de verdad de la función $f = \Pi(0,1)d(2,5)$ junto con los términos subsumas del maxtérmino M_1.

x	**y**	**z**	**f**	$x+y$	$x+\overline{z}$	$y+\overline{z}$	x	y	$\overline{z}$
0	0	0	0	0	1	1	0	0	1
0	0	1	0	0	0	0	0	0	0
0	1	0	-	1	1	1	0	1	1
0	1	1	1	1	0	1	0	1	0
1	0	0	1	1	1	1	1	0	1
1	0	1	-	1	1	0	1	0	0
1	1	0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	0

En este ejemplo se han obtenido tres implicadas primas: $\{y+\overline{z}\}$, $\{x+z\}$ y $\{x + y\}$. El lector podrá comprobar que el producto irredundante es

$f(x, y, z) = x + y.$

3.3.4. Método de simplificación mediante K-mapa

El proceso de minimización del coste de un circuito combinacional pasa por encontrar la expresión irredundante que dicho circuito implementa. A continuación se muestra cómo, a partir del K-mapa, se pueden obtener implicantes (implicadas) de la función representada en él, qué criterios se deben seguir para escoger aquellas implicantes (implicadas) que sean primas, descartando las que no los son, y cómo se puede conseguir la suma (producto) irreduntante a partir de éstas últimas.

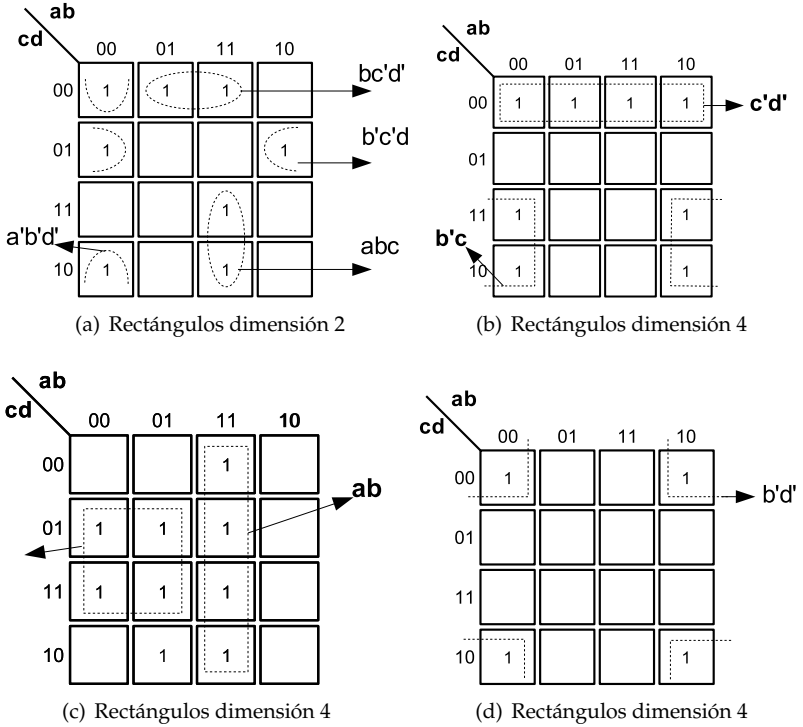
3.3.4.1. Obtención de implicantes o implicadas a partir de un K-mapa.

En un K-mapa, cada celda contiene el valor binario que toma la función de conmutación para la entrada asociada a dicha celda. Es decir, cada celda puede representar los mintérminos (si contiene un 1) o los maxtérminos (si contiene un 0) de la función. Aquellas celdas que contengan unos se denominan *1-cells*, y las que contengan ceros, *0-cells*.

La importancia del K-mapa radica en que este facilita la obtención de las implicantes (implicadas). Un conjunto de *1-cells*(*0-cells*) que formen un rectángulo de dimensiones $2^a \times 2^b$, es descrito por un término producto (suma) de n - a - b variables, donde n representa el número de variables de la función (o dimensión del K-mapa). La dimensión de un rectángulo mide el número de unos (o ceros) que contiene. Por ejemplo, un rectángulo de dimensiones $2^0 \times 2^0$ de *1-cells* está representado por $2^0 \times 2^0 = 1 \times 1 = 1$ *1-cell*, que en este caso se corresponde con un mintérmino. Si la función tiene cuatro variables, el mintérmino, o el rectángulo de *1-cells*, puede ser descrito por $4 - 0 - 0 = 4$ variables. En el caso en el que un rectángulo de *0-cells* tenga las dimensiones $2^1 \times 2^0$, entonces, dicho rectángulo contiene $2^1 \times 2^0 = 2 \times 1 = 2$ *0-cells*. Para este último ejemplo, si el número de variables es, igualmente de cuatro, el rectángulo puede ser descrito, matemáticamente, por un término suma de $4 - 1 - 0 = 3$ variables.

En las figuras 3.24 y 3.25 se han representado K-mapas de 4 variables en

el que pueden verse ejemplos de rectángulos de *1-cells* de dimensión 2, 4 y 8 junto con las expresiones producto que los describen.



(a) Rectángulos dimensión 2

(b) Rectángulos dimensión 4

(c) Rectángulos dimensión 4

(d) Rectángulos dimensión 4

Figura 3.24. Diferentes tipos de rectángulos de dimensión 2 y 4 en K-mapas de 4 variables.

Para ayudar a comprender el hecho de que un rectángulo de *1-cells* es descrito por un término producto, se analizarán los siguientes casos:

- En el K-mapa de la figura 3.26 se ha representado una función de conmutación $f(a,b,c,d) = \sum(5,13) = \overline{a}\ b\overline{c}\ d + a\ b\ \overline{c}\ d$.

  Obsérvese que, los dos mintérminos, ocupan casillas adyacentes en el K-mapa, porque sus entradas asociadas sólo difieren en un bit y, debido a esto, la expresión de la función se puede reducir eliminando el literal que cambia en ambos mintérminos, o sea, $\overline{a}\ b\ \overline{c}\ d + a\ b\ \overline{c}\ d = (a+\overline{a})b\ \overline{c}\ d = b\ \overline{c}\ d$.

  Cualquier rectángulo formado por dos *1-cell* adyacentes puede descri-

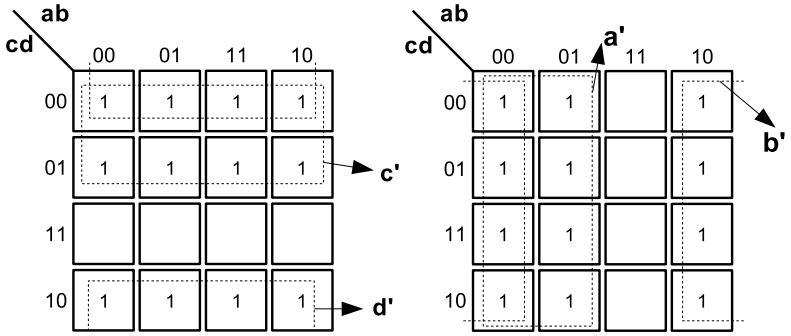


Figura 3.25. Diferentes tipos de rectángulos de dimensión 8 en K-mapas de 4 variables.

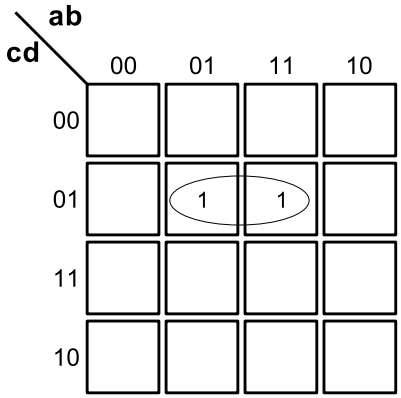


Figura 3.26. K-mapa de cuatro variables con un rectángulo de dos *1-cells*.

birse mediante un término producto que contiene un literal menos que el de los mintérminos individuales asociados a cada *1-cell*. Recuérdese que, el K-mapa, fue construido para que dos celdas fueran adyacentes si sus entradas asociadas difieren, exactamente, en un bit. Como una *1-cell* describe un mintérmino, los mintérminos de dos celdas adyacentes difieren exactamente en un literal. La propiedad $p_1\ p_2 + p_1\ \overline{p_2} = p_1$ puede aplicarse en este caso. Entonces, un par de celdas adyacentes, representarán un término producto con una variable eliminada (la que cambia).

A partir del K-mapa, se puede obtener directamente la expresión del término producto que define el rectángulo de dos *1-cells*. En primer lugar se escogen los literales asociados a las entradas que no cambian para los dos *1-cells*, de forma que si la entrada es 0, su literal asociado aparece complementado y si es 1, sin complementar.

Ejemplos.

En la figura 3.24.a se han representado diferentes rectángulos de dimensión dos en un K-mapa de cuatro variables. Para cada rectángulo se ha representado su expresión asociada. Asimismo se propone al lector que haga el proceso inverso, es decir, a partir de un término producto de tres literales, obtenga la representación, en K-mapa de cuatro variables, de dicho término producto.

- En el K-mapa de la figura 3.27 se ha representado la función $f(a,b,c,d)$ = $\sum$ (2, 3, 10, 11) = $\overline{a}\ \overline{b}\ c\ \overline{d} + \overline{a}\ \overline{b}\ c\ d + a\ \overline{b}\ c\ d + a\ \overline{b}\ c\ \overline{d}$ que forma un rectángulo de cuatro *1-cells*.

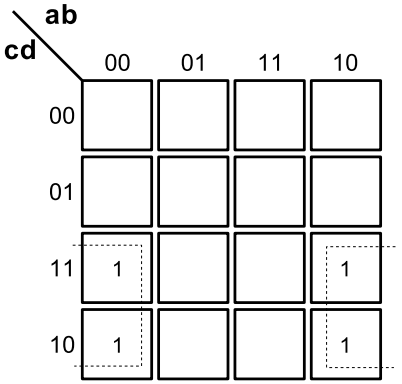


Figura 3.27. K-mapa de cuatro variables con un rectángulo formado por cuatro *1-cells*.

Los mintérminos m_2 y m_3 son adyacentes y, por tanto, $\sum(2,3)$ =

$\overline{a}\ \overline{b}\ c\ \overline{d} + \overline{a}\ \overline{b}\ c\ d = \overline{a}\ \overline{b}\ c$. Del mismo modo, los mintérminos m_{10} y m_{11} son adyacentes y $\sum(10, 11) = a\ \overline{b}\ c\ d + a\overline{b}\ c\ \overline{d} = a\ \overline{b}\ c$. Por tanto, la función f se reduce a $f = \overline{a}\ \overline{b}\ c + a\ \overline{b}\ c$. Nótese que los términos $\overline{a}\ \overline{b}\ c$ y $a\ \overline{b}\ c$ describen dos rectángulos de dos *1-cells*que, también, son adyacentes, por lo que se puede aplicar, de nuevo, la reducción de literales: $f(a, b, c, d,) = \overline{a}\ \overline{b}\ c + a\ \overline{b}\ c = \overline{b}\ c$. Entonces, un rectángulo de cuatro *1-cells* puede describirse mediante un término producto en el que se han eliminado dos variables de la función.

A partir del K-mapa se puede obtener directamente la expresión del término producto que define el rectángulo de cuatro *1-cells*. En primer lugar se escogen los literales asociados a las entradas que no cambian para los cuatro *1-cells*, de forma que si la entrada es 0, su literal asociado aparece complementado y si es 1, sin complementar.

El proceso de obtención de la expresión asociada a los rectángulos, puede extenderse a rectángulos de mayor tamaño o a rectángulos formados por *0-cells*. En este último caso la expresión que las representa se corresponde con un término suma en el que los literales asociados a las entradas que son cero aparecen sin complementar y los que sus entradas son 1, complementados (figura 3.28).

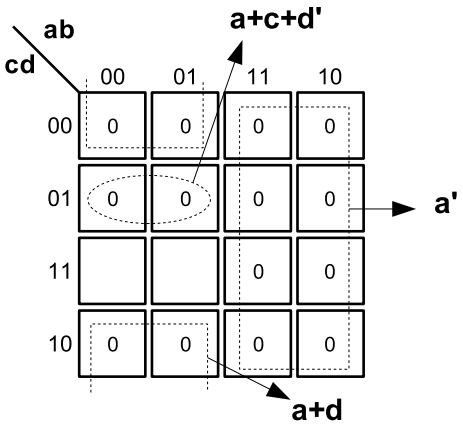


Figura 3.28. K-mapa de cuatro variables con rectángulos de *0-cells*.

3.3.4.2. Obtención de las implicantes primas y sumas irredundantes por el método del K-mapa.

A partir de un K-mapa, las implicantes de la función que se representa en él, se determinan mediante las expresiones asociadas a los rectángulos de *1-cells* de dicha función. En la figura 3.29 se ha representado las expresiones de tres implicantes de la función (aunque existen más implicantes: los cuatro mintérminos y los dos rectángulos de 1-cells horizontales).

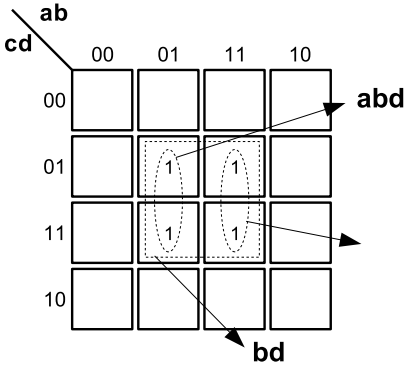


Figura 3.29. Búsqueda de implicantes primas en un K-mapa de cuatro variables.

Nuestro interés debe centrarse en encontrar implicantes primas. Como se observa en la figura 3.29, las expresiones de las tres implicantes son: $\overline{a}bd$, abd y bd. Analizando estas expresiones, se observa que, las dos primeras, son subsumas de la tercera, por consiguiente aquellas no pueden ser primas (Nótese, que cualquiera de los cuatro mintérminos o rectángulos horizontales de *1-cells*, son subsumas, también, de la implicante bd). Todo esto nos permite afirmar que, para la función representada en la figura 3.29, la única implicante prima es la expresada por el término bd. En resumen, las implicantes primas vienen representadas en el K-mapa por rectángulos de *1-cells* con el mayor área posible. Todas aquellos rectángulos de *1-cells* contenidos en otros de mayor dimensión, no representan a implicantes primas. Se puede seguir el siguiente algoritmo para la búsqueda de implicantes primas.

1. Marcar todos los mintérminos de la función porque éstos son las primeras implicantes (Implicante de orden 0, con dimensión igual a uno).
2. Marcar todas las **parejas de mintérminos adyacentes** y eliminar las marcas de los mintérminos contenidos en estas parejas (Implicantes de orden 1, con dimensión igual a dos). Sea n=1.
3. Marcar todas las **parejas de implicantes de orden n que sean adyacentes** y eliminar las marcas de las implicantes contenidas en estas parejas (Implicantes de orden m=n+1, con dimensión igual a 2^m). Hacer n=n+1
4. Repetir el apartado anterior hasta que no se encuentren más implicantes de orden superior.
5. Todas las implicantes que queden marcadas al final del proceso, sean del orden que sean, son implicantes primas de la función

Ejemplo. Encontrar todas las implicantes primas de la función $F = \sum(0, 4, 5, 6, 7, 9, 10, 14, 15)$.

En la figura 3.30 se muestra el K-mapa de cuatro variables que contiene todos los mintérminos de la función F y que han sido marcados con una circunferencia. Estos son los implicantes de orden cero.

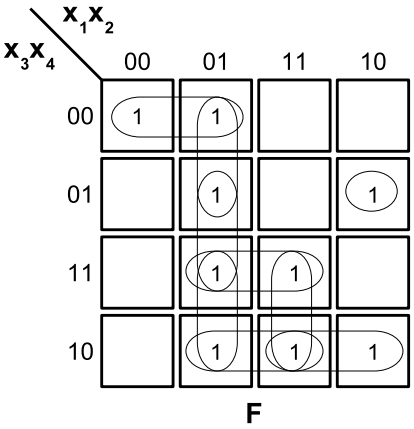


Figura 3.30. Implicantes de orden 0 de la función $F = \sum(0, 4, 5, 6, 7, 9, 10, 14, 15)$.

A partir de los mintérminos, se buscan todas las implicantes de orden uno que engloban dos mintérminos adyacentes. Al final del proceso se desmarcan todas aquellas implicantes de orden inferior que estén contenidas en las de orden uno (figura 3.31.a).

A partir de las implicantes de orden uno, se buscan todas las implicantes de orden dos que engloban a dos de las anteriores que sean adyacentes. Al final del proceso se desmarcan todas aquellas implicantes de orden inferior que estén contenidas en las de orden dos (figura 3.31.b).

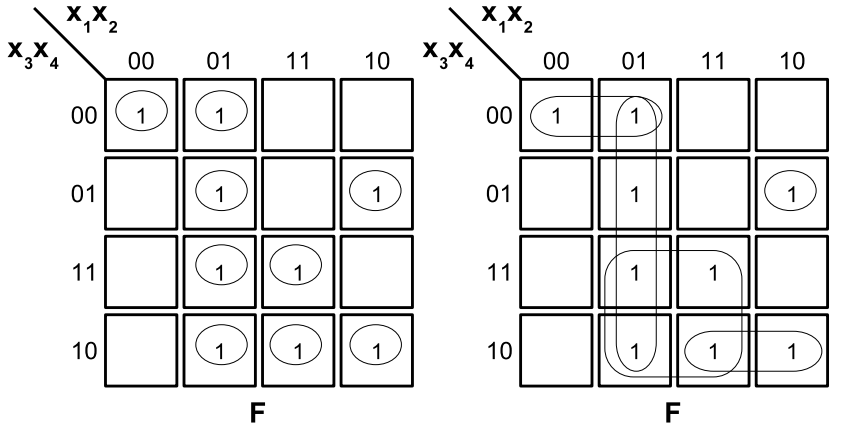


Figura 3.31. En (a) se representan todas las implicantes de orden 1. Se han eliminado aquellas implicantes de orden inferior englobadas en las de orden superior. A partir de (a) se llega a (b) buscando todas las implicantes de orden 2. De igual forma, se han eliminado todas las implicantes de menor orden que hayan sido englobadas por las de orden superior. Puesto que no hay posibilidad de encontrar más implicantes de orden superior a 2, (b) representa todas las implicantes primas de la función $F = \sum(0, 4, 5, 6, 7, 9, 10, 14, 15)$.

Todas las implicantes resultantes en el K-mapa, al final del proceso, son implicantes primas. Para este ejemplo son: $\overline{x_1}\ \overline{x_3}\ \overline{x_4}$, $\overline{x_1}\ x_2$,$x_2\ x_3$, $x_1\ x_3\ \overline{x_4}$ y $x_1\ \overline{x_2}\ \overline{x_3}\ x_4$.

Una vez obtenida la lista de todas las implicantes primas, cabe determinar la suma irredundante. A continuación se mostrarán numerosos ejemplos que nos ayudarán a presentar los diferentes criterios que se deben seguir.

Ejemplos.

1. El K-mapa de la figura 3.32 representa una función, f, de tres variables, junto con las dos implicantes primas que posee.

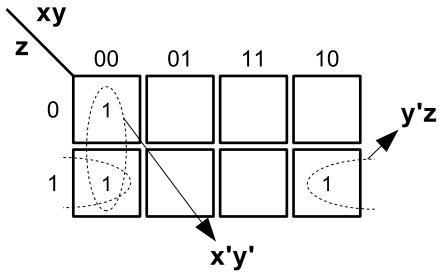


Figura 3.32. Representación de la función $F = \sum(0, 1, 5)$.

La fórmula irredundante es una suma de implicantes primas en la que ninguna de ellas pueda retirarse de la suma y que, la expresión resultante, siga implementando la misma función de conmutación. En este ejemplo, las dos implicantes primas deben participar en la expresión final, por consiguiente:

$$f(x, y, z) = \overline{x}\ \overline{y} + \overline{y}\ z$$

2. En el K-mapa de cuatro variables de la figura 3.33 se ha representado una función F junto con las tres implicantes primas que posee: $b\ d$,$\overline{a}\ d$ y $\overline{a}\ \overline{b}$.

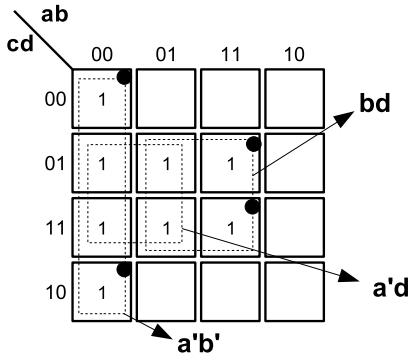


Figura 3.33. Representación de la función $F = \sum(0, 1, 2, 3, 5, 7, 13, 15)$.

En una primera aproximación, se puede construir la suma irredundante como $F(a, b, c, d) = b\ d + \overline{a}\ d + \overline{a}\ \overline{b}$. No obstante, ésta no consituye la suma de menor coste, puesto que podemos retirar el término producto

$\overline{a}\ d$ sin que, la función descrita por la expresión resultante, cambie. No ocurre lo mismo si son, las otras implicantes, las que se retiran.

Existen implicantes primas que son **esenciales** y deben aparecen en la suma irredundante y, otras, que no lo son y que, en muchos casos, podrán no aparecer en la expresión final. Mediante el K-mapa pueden determinarse, de forma muy sencilla, las implicantes que son esenciales. Para ello se procede a buscar todas aquellas *1-cells* que sólo están cubiertas por una única implicante prima. Dichas casillas se denominan *1-cells distinguidas* o, simplemente, *1 distinguido*. Cualquier implicante prima que englobe uno o más *1-cells distinguidas* es una implicante prima esencial. De hecho, si una función toma el valor uno para una entrada en la que sólo un término producto de la expresión final evaluaría como uno, dicho término producto no debe ser eliminado. En el K-mapa de la figura 3.33 se han marcado, con un punto, todas las *1-cells distinguidas*. Las implicantes primas $b\ d$ y $\overline{a}\ d$ son esenciales. Puede imaginarse que si alguna de ellas no estuviera en la expresión final, no habría forma de que la función, que dicha expresión describa, evaluara como 1 para las entradas (0,2,13,15). Entonces, como mínimo, la función $F(a,b,c,d) = b\ d + \overline{a}\ d$. La otra implicante prima, $\overline{a}\ \overline{b}$, que no es esencial, cubre unos 1-cells que ya están cubiertos por la suma de las implicantes primas esenciales anteriores, por tanto, no se necesita en la expresión final. La suma irredundante es:

$$F(a,b,c,d) = b\ d + \overline{a}\ d$$

3. En el K-mapa de la figura 3.34 se ha representado una función de cuatro variables junto con sus cinco implicantes primas de las cuales, cuatro, son esenciales: las de orden uno.

   Por lo visto en el ejemplo anterior, los términos asociados a las implicantes primas esenciales deben aparecer, obligatoriamente, en la expresión irredundante y, estos, son suficientes para describir la función de conmutación porque cubren todos los mintérminos del K-mapa. Por tanto la suma irredundante es:

$$F(a,b,c,d) = \overline{b}\ \overline{c}\ \overline{d} + \overline{a}\ \overline{b}\ d + b\ \overline{c}\ d + \overline{a}\ b\ \overline{d}$$

4. En los dos ejemplos anteriores, la expresión mínima en suma de productos estaba consituida, exclusivamente, por implicantes primas esenciales. En este ejemplo se ilustrará que esta situación, siempre, no es posible. En la figura 3.35 se ha representado una función F de tres variables junto con sus implicantes primas de las cuales, dos, son esenciales.

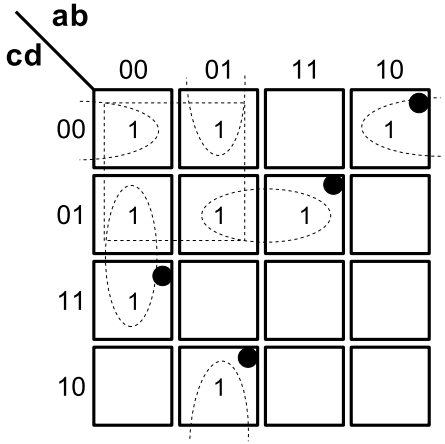


Figura 3.34. Representación de la función $F = \sum(0, 1, 3, 4, 5, 6, 8, 13)$.

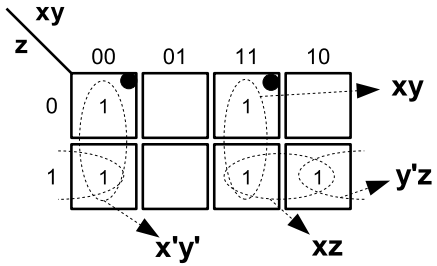


Figura 3.35. Representación de la función $F = \sum(0, 1, 5, 6, 7)$.

La expresión, $F(x, y, z) = \overline{x}\ \overline{y} + x\ y$, formada con la suma de las implicantes primas esenciales no permite cubrir todos los unos de la función F, faltaría el asociado a la entrada cinco. Para incluir, este último, en la expresión final, existen dos opciones: añadir la implicante $\overline{y}\ z$ o la implicante $x\ z$. En ambos casos, el coste de la expresión resultante es el mismo y cualquiera de las dos soluciones sería correcta. Suma irredundante:

Solución 1: $F(x, y, z) = \overline{x}\ \overline{y} + x\ y + x\ z$

Solución 2: $F(x, y, z) = \overline{x}\ \overline{y} + x\ y + \overline{y}\ z$

Como reglas generales a seguir: si un mintérmino es cubierto por varias implicantes, se deberá escoger aquella que consiga cubrir, junto con dicho mintérmino, el mayor número de *1-cells* posibles. En el caso de que existan varias implicantes que, además de cubrir el mismo número de mintérminos adicionales, tienen diferentes costes, se escogerá la de menor.

5. En el K-mapa de la figura 3.36 se ha representado una función F de cuatro variables. Salvo la implicante prima esencial bd, que cubre los mintérminos $\sum(5, 7, 13, 15)$, las restantes *1-cells* están cubiertas por un par de implicantes primas. Para empezar, $F(a, b, c, d) = b\ d + \ldots$.

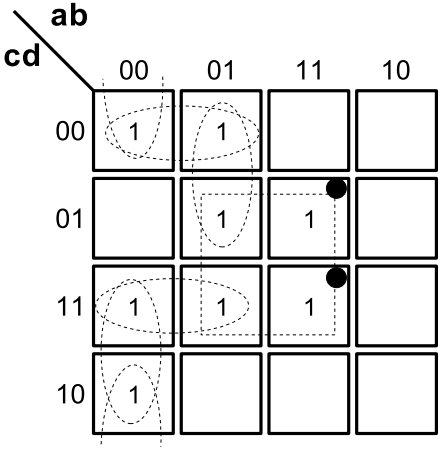


Figura 3.36. Representación de la función $F = \sum(0, 2, 3, 4, 5, 7, 13, 15)$.

Ahora, se tendrán que buscar las implicantes primas que cubran los mintérminos de la función que no son abarcados por el producto $b\ d$. Para ello, se comenzará por el mintérmino m_4 que se encuentra cubierto por las implicantes $\overline{a}\ b\ \overline{c}$ y $\overline{a}\ \overline{c}\ \overline{d}$. La primera implicante, $\overline{a}\ b\ \overline{c}$, cubre, además

de a dicho mintérmino, al m_5, el cual ya estaba cubierto por la implicante $b\ d$, mientras que la implicante prima $\overline{a}\ \overline{c}\ \overline{d}$, además de mintérmino m_4 cubre al m_0 que aún está sin cubrir. Como ambas implicantes tienen el mismo coste (igual número de literales), la mejor elección es la ofrecida por la implicante $\overline{a}\ \overline{c}\ \overline{d}$, por tanto, $F(a,b,c,d) = b\ d + \overline{a}\ \overline{c}\ \overline{d} + \ldots$.

Todavía nos quedan *1-cells* por cubrir, pero parece evidente que la mejor solución de todas viene dada por la implicante $\overline{a}\ \overline{b}\ c$. La expresión en suma de productos de menor coste es:

$$F(a,b,c,d) = bd + \overline{a}\ \overline{c}\ \overline{d} + \overline{a}\ \overline{b}\ c$$

Si una implicante prima, A, tiene un coste menor o igual que otra implicante prima, B, y los mintérminos que son cubiertos por B (excluyendo a aquellos que pertenecen a otras implicantes primas que deben aparecer en la suma irredundante) están contenidos en los de A, entonces la implicante prima B puede eliminarse del conjunto de implicantes primas candidatas a pertenecer a la expresión irredundante final. Otras circunstancias, como que la implicante prima A tenga mayor coste que la B o/y que los mintérminos de B (con la citada exclusión) no estén totalmente incluidos en A, no permiten la eliminación de B.

6. En este ejemplo se ilustra el procedimiento que se debe seguir para conseguir la expresión mínima de una función cuando ésta no tiene implicantes primas esenciales. En el K-mapa de la figura 3.37 se han representado las implicantes primas de una función de cuatro variables y en la que no existen *1-cells* distinguidos.

   En casos como este se deberá empezar por escoger una implicante prima de entre todas las posibles que cubran un mintérmino determinado. A partir de la implicante prima escogida, se deberá encontrar la expresión de menor coste asociada a la elección hecha. Este procedimiento se repetirá para las demás implicantes que contengan al mintérmino de partida. De todas las expresiones obtenidas, la de menor coste, constituye la suma irredundante. Como es obvio, se recomienda buscar un mintérmino de partida que esté cubierto por el menor número de implicantes primas.

   Para el ejemplo que nos ocupa, se puede observar en la figura 3.37 que todos los mintérminos de la función están cubiertos por dos implicantes primas, por lo que, cualquiera de ellos, nos sirve para iniciar el procedimiento. Escogemos, por ejemplo, el mintérmino m_0, que está cubierto por la implicante prima $\overline{b}\ \overline{c}$ o por la implicante prima $\overline{b}\ \overline{d}$. Como alguna de las dos implicantes anteriores deben aparecer en la suma irredundante, se comenzará suponiendo que es la primera, $\overline{b}\ \overline{c}$, la que debe estar.

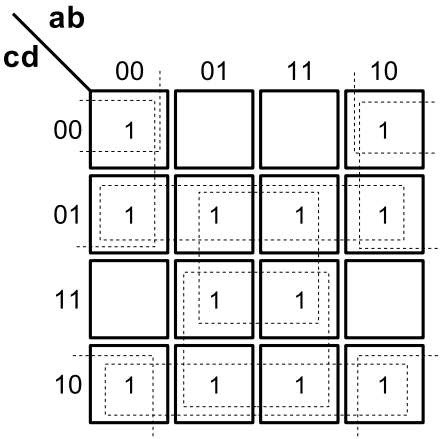


Figura 3.37. Representación de la función $F = \sum(0, 1, 2, 5, 6, 7, 8, 9, 10, 13, 14, 15)$.

a) $F(a, b, c, d) = \overline{b}\ \overline{c} + ..$ La figura 3.38 muestra los mintérminos de la función F que se encuentran cubiertos por la implicante prima $\overline{b}\ \overline{c}$. De las restantes implicantes primas, se deberán escoger aquellas que permitan cubrir los mintérminos que no aparecen sombreados.

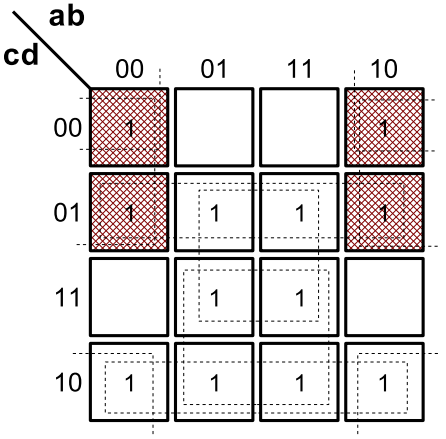


Figura 3.38. Representación de la función $F = \sum(0, 1, 2, 5, 6, 7, 8, 9, 10, 13, 14, 15)$ en la que se han sombreado los cuadros cubiertos por la expresión $F(a, b, c, d) = \overline{b}\ \overline{c} + ..$

Existen dos posibles coberturas para el mintérmino m_2, la determinada por la implicante prima $c\ \overline{d}$ o por la $\overline{b}\ \overline{d}$. La primera, $c\ \overline{d}$, además de al propio mintérmino m_2, cubre a los mintérminos m_6, m_{10} y m_{14}. En cambio, la implicante prima $\overline{b}\ \overline{d}$, adicionalmente, tan

sólo cubre al mintérmino m_{10} y tiene el mismo coste que la primera. Por consiguiente, la mejor opción está dada por $c\ \overline{d}$, que se añade a la expresión algebraica de la función $F(a,b,c,d) = \overline{b}\ \overline{c} + c\ \overline{d} + \ldots$, y que cubren los mintérminos que se han sombreado en la figura 3.39.

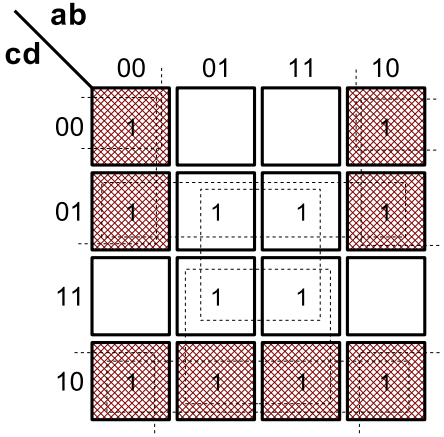


Figura 3.39. Representación de la función $F = \sum(0,1,2,5,6,7,8,9,10,13,14,15)$ en la que se han sombreado los cuadros cubiertos por la expresión $F(a,b,c,d) = \overline{b}\ \overline{c} + c\ \overline{d} + \ldots$.

Para los mintérminos que quedan por cubrir y de las implicantes primas que pueden usarse para su cubrimiento, la bc es la mejor opción. Por consiguiente, la expresión de menor coste que se obtiene bajo la primera suposición es:

$$F(a,b,c,d) = \overline{b}\ \overline{c} + c\ \overline{d} + b\ c$$

b) Si se hubiera escogido a la implicante prima $\overline{b}\ \ \overline{d}$ para cubrir al mintérmino m_0, entonces, la función debería expresarse como:

$$F(a,b,c,d) = \overline{b}\ \overline{d} + \ldots$$

. En la figura 3.40 se han sombreado los mintérminos que son cubiertos por esa implicante prima.

Ahora, el mintérmino m_1 puede ser cubierto por la implicante prima $\overline{c}\ d$ o por la implicante prima $\overline{b}\ \overline{c}$. La primera de ellas, además de cubrir al mintérmino m_1, cubre a los m_5,m_9,y m_{13}; mientras que la segunda, además del mintérmino m_1, sólo cubre al m_9. Ambas implicantes tienen igual coste, pero la primera cubre más unos que la

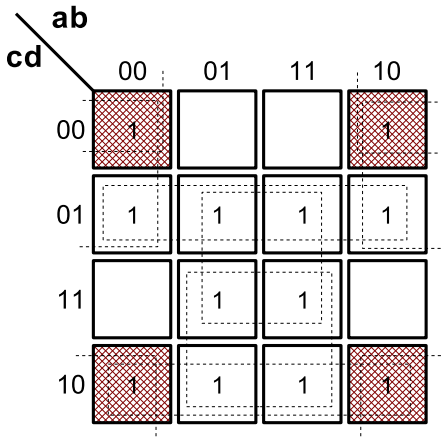


Figura 3.40. Representación de la función $F = \sum(0, 1, 2, 5, 6, 7, 8, 9, 10, 13, 14, 15)$ en la que se han sombreado los cuadros cubiertos por la expresión $F(a, b, c, d) = \overline{b}\,\overline{d} + \ldots$

segunda, por tanto, ésta última se puede eliminar y deja, a $\overline{c}\,d$ como la única implicante prima que puede representar al mintérmino m_1, $F(a, b, c, d) = \overline{b}\,\overline{d} + \overline{c}\,d + ..$

De las implicantes primas restantes y de los mintérminos que quedan por cubrir, se deduce que la mejor opción viene dada por la implicante $b\,d$. Por tanto, la expresión final queda como sigue:

$$F(a, b, c, d) = \overline{b}\,\overline{d} + c\,\overline{d} + b\,d$$

Comparando los costes de las dos posibles soluciones obtenidas se deduce que, ambas, son sumas irredundantes.

3.3.4.3. Obtención de las implicadas primas y productos irredundantes mediante el K-mapa.

Se sigue el mismo procedimiento que para las implicantes primas y sumas irredundantes, pero ahora, se buscan rectángulos de *0-cells* que vienen expresados mediante términos suma.

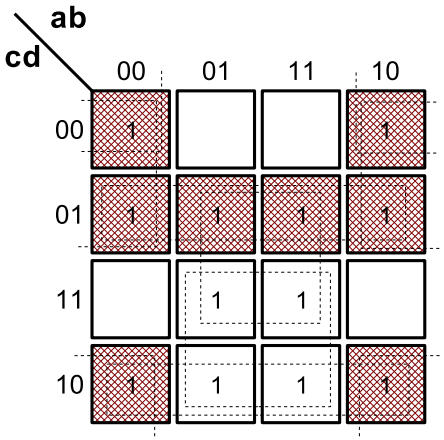


Figura 3.41. Representación de la función $F = \sum(0, 1, 2, 5, 6, 7, 8, 9, 10, 13, 14, 15)$ en la que se han sombreado los cuadros cubiertos por la expresión $F(a, b, c, d) = \overline{b}\ \overline{d} + \overline{c}\ d + ..$

Ejemplo.

Obtener el producto mínimo de la función cuyo K-mapa está representado en la figura 3.42.

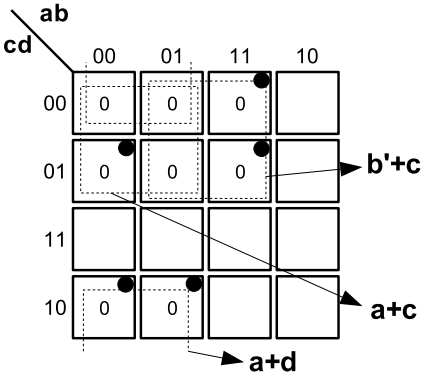


Figura 3.42. Representación de la función de conmutación $F = \Pi(0,\ 1,\ 2,\ 4,\ 5,\ 6,\ 12,\ 13)$ junto con las expresiones de sus implicadas primas.

Existen tres implicadas primas que son esenciales por contener ceros distinguidos. El producto irredundante es:

$$f(a, b, c, d) = (a + d)(a + c)(\overline{b} + c)$$

3.3.4.4. Minimización de funciones de conmutación incompletas

En la figura 3.43 se ha representado el K-mapa de la siguiente función de conmutación incompleta: $F = \sum(5, 7, 15) + d(13)$

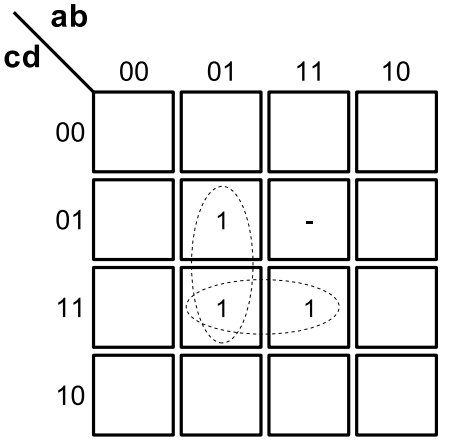


Figura 3.43. K-mapa de la función $f = \sum(5, 7, 15) + d(13)$.

Parece razonable buscar las implicantes primas asociadas a los *1-cells*. Para este ejemplo, la función F vendría dada por la relación:

$$F(a, b, c, d) = \overline{a}\ b\ d + b\ c\ d$$

No obstante, se sabe que una inespecificación, para un circuito digital, se corresponde con una condición de entrada que no sucede nunca y que, por eso, no se necesita definir cuál sería su salida correspondiente. Podemos valernos de las entradas no definidas para conseguir una reducción mayor de la expresión lógica. En concreto, para el ejemplo de la figura 3.43, si se toma la inespecificación como si fuera un 1 lógico, la expresión final resultante sería $F(a, b, c, d) = b\ d$, de menor coste que la obtenida a partir de los mintérminos exclusivamente. Si se realizara un test de funcionamiento, en el que se observa la salida del circuito para todas las entradas **definidas** posibles (excluyendo la 13), a los circuitos que implementen las funciones $F(a, b, c, d) = b\ d$ y $F(a, b, c, d) = \overline{a}\ b\ d + b\ c\ d$, se vería que ambos funcionan de manera idéntica, pero, el primero, como es obvio, tiene un coste menor.

La obtención de la expresión mínima en s.p. (suma de productos) o p.s. (producto de sumas), pasa por dos fases: obtención de las implicantes (implicadas) primas y cubrimiento mínimo de los *1-cells*(*0-cells*). Para las funciones de conmutación incompletas, se utilizarán las entradas no definidas como si fueran 0 o 1, a conveniencia, para formar los rectángulos de *1-cells* (o de *0-cells*) de mayor área posible y, así, conseguir implicantes (implicadas) primas más reducidas. En la fase de cubrimiento mínimo, se descartarán aquellas implicantes primas que sólo cubran inespecificaciones, sólo se utilizarán aquellas que, al menos, cubran a un mintérmino.

Ejemplo.

Obtener la expresión mínima, en suma de productos, de la función $F = \sum(0, 2, 3, 4, 12, 13) + d(5, 7, 10, 11)$.

Las implicantes primas de F que contienen, al menos, un mintérmino de la función, se han dibujado en el K-mapa de la figura 3.44.

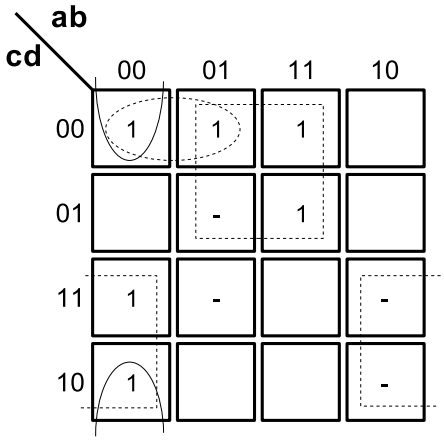


Figura 3.44. K-mapa de la función $F = \sum(0, 2, 3, 4, 12, 13) + d(5, 7, 10, 11)$.

La inespecificación asociada a la entrada 5, tomada como 1 lógico, permite obtener la implicante prima $b\ \overline{c}$. Del mismo modo, las inespecificaciones 10 y 11, tomadas como 1 lógico, permiten generar la implicante prima $\overline{b}\ c$ y la inespecificación 7 la implicante prima $\overline{a}\ c\ d$.

Ahora se estudia el cubrimiento. La implicante prima $b\overline{c}$ es esencial. Por

otro lado, el mintérmino m_0 está cubierto por dos implicantes primas de igual coste: $\overline{a}\ \overline{c}\ \overline{d}$ y $\overline{a}\ \overline{b}\ c$. La mejor de las dos es la $\overline{a}\ \overline{b}\ c$ porque, además del mintérmino m_0 cubre al m_2. Por último, para cubrir el mintérmino m_3, están las implicantes primas $\overline{b}c$ y $\overline{a}cd$, que no cubren ningún mintérmino adicional y cuyos costes permiten escoger a la primera de las dos. La expresión final irredundante es:

$$F(a,b,c,d) = b\,\overline{c} + \overline{a}\,\overline{b}\,c + \overline{b}\,c$$

3.3.4.5. Minimización de funciones de conmutación de 5 variables

El procedimiento de minimización mediante el K-mapa ya ha sido ampliamente estudiado y es aplicable a mapas de cualquier tamaño. La única novedad que puede plantear la simplificación en mapas de 5 variables es el hecho de la obtención de las implicantes (implicadas) primas, ya que la formación de los rectángulos de *1-cell*(*0-cell*) es algo más compleja.

Se puede considerar que el mapa de 5 variables está constituido por un K-mapa de cuatro variables (K_1) al que se le ha añadido, por la derecha, un nuevo K-mapa de cuatro variables (K_2). La formación de las implicantes (implicadas) primas se realiza en cada parte (K_1 y K_2) por separado, de igual manera a la estudiada con anterioridad, y en conjunto, buscando aquellos rectángulos que ocupen posiciones adyacentes en ambas mitades. Los rectángulos que cumplan estos requisitos, constituyen una implicante (implicada) prima.

Ejemplos.

1. El K-mapa de la figura 3.45 muestra todos las implicantes primas asociadas a la función $F = \sum(0,5,6,7,8,13,15,16,20,21,22,23,24,29,31)$. Usando aquellas que son esenciales, se obtiene $F = \overline{c}\,\overline{d}\,\overline{e} + \overline{a}\,b\,\overline{c} + c\,e + \overline{b}\,c\,d$.

2. El K-mapa de la figura 3.46 muestra todas las implicadas primas asociadas a la función $F = \Pi(1,3,5,7,9,10,11,16,17,18,19,20,21,22,23,25,26,27)$. Todas ellas son primas, por consiguiente, $F = (\overline{a}+b)(b+\overline{e})(c+\overline{e})(\overline{b}+c+\overline{d})$.

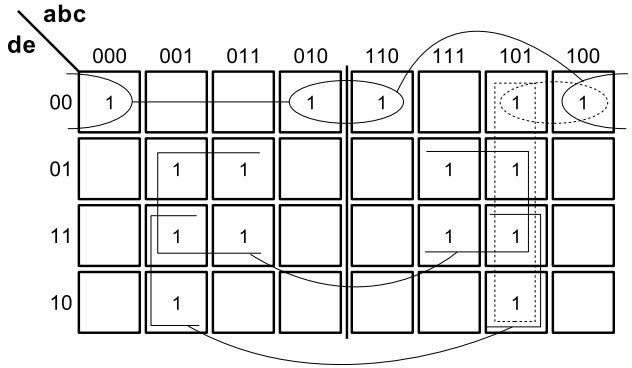


Figura 3.45. K-mapa de la función $F = \sum(0, 5, 6, 7, 8, 13, 15, 16, 20, 21, 22, 23, 24, 29, 31)$.

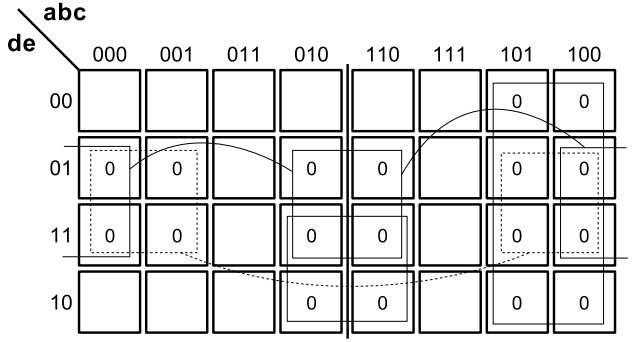
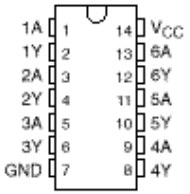


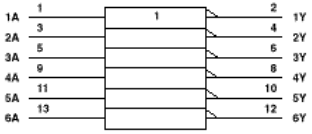
Figura 3.46. K-mapa de $F = \Pi(1, 3, 5, 7, 9, 10, 11, 16, 17, 18, 19, 20, 21, 22, 23, 25, 26, 27)$.

Ejercicios propuestos

Problema 3.1 Describa los terminales y funciones del esquema mostrado en la figura 3.3.4.5. Asimismo, explique los parámetros eléctricos representados en la tabla 3.24 y calcule los márgenes de ruido de este dispositivo.



(a) Encapsulado



(b) Simbólos

FUNCTION TABLE
(each inverter)

INPUT A	OUTPUT Y
H	L
L	H

(c) Tabla de función

Tabla 3.24:

	MIN	NOM	MAX
V_{cc}	4,5V	5V	5,5V
V_{IH}	2V		
V_{IL}			0,8V
V_{OH}	4,8V		
V_{OL}		0,35V	0,5V
$t_{p_{LH}}$	1ns		5ns
$t_{p_{HL}}$	1ns		4ns
$Fanout$			20

Problema 3.2 Analizar el circuito de la figura 3.47 y obtenga la expresión mínima en suma de productos.

Problema 3.3 Obtenga todas las implicantes primas de la función $f = \Pi(6, 7, 8, 12, 13)d(0, 5, 10)$

Problema 3.4 Obtenga la expresión óptima en producto de sumas que implementa la función:

$$F = \sum(1, 2, 3, 4, 5, 7, 8,) + d(6, 10, 14)$$

Problema 3.5 Para el circuito de la figura 3.48, implemente el circuito G de menor coste, para que $F(w, x, y, z) = \sum(0, 1, 2, 3, 4, 7, 8, 11)$.

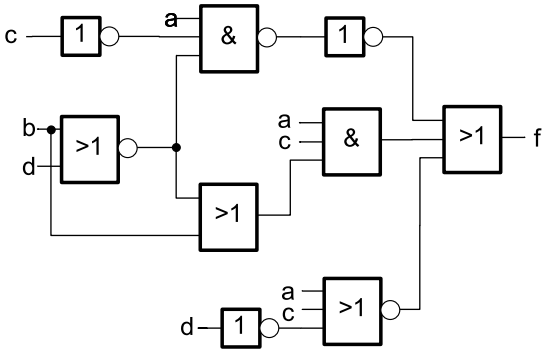


Figura 3.47.

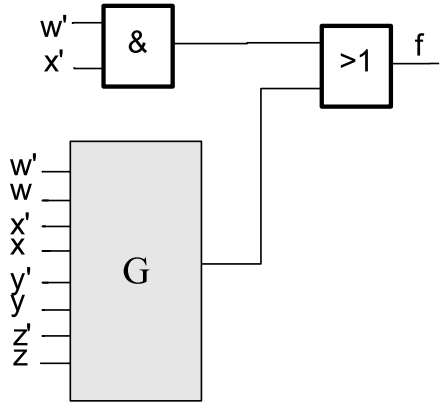


Figura 3.48.

Problema 3.6 Se dispone de puertas F de dos entradas, tal que $F(a,b) = a \bullet b + \overline{a}$: Se pide:

1. Implementar con puertas F y las constantes lógicas 0 y 1, las puertas AND, OR y NOT. ¿Forman las puertas F un conjunto completo de operadores?.
2. Analizar el circuito de la figura 3.49. Obtener la expresion canónica de sumas de productos de la salida Z.

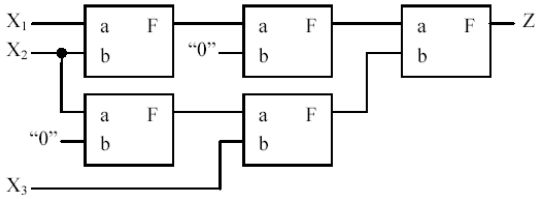


Figura 3.49.

3. Implementar con puertas F y una puerta NAND (que no se podrá utilizar como inversor) la función W = $\sum(2, 4, 6)$

Problema 3.7 Dibujar el mapa de Karnaugh de 6 variables ($abcdef$) e indicar cómo se podrían obtener todos los implicantes de cuatro literales que contengan, simultáneamente, los mintérminos m_{42} y m_{10}.

Problema 3.8 A un depósito (figura 3.50) acceden cuatro canalizaciones de líquido, cada una de las cuales es capaz de suministrar un caudal determinado. Cada una de estas canalizaciones es controlada por una electroválvula cuyo estado (abierto o cerrado) depedende de una variable binaria. Llamemos a, b, c, d a las variables binarias que controlan dichas electroválvulas. Estas variables binarias son generadas por un sistema de control de acceso al depósito. Las cuatro canalizaciones de salida, también están controladas por electroválvulas, pero el caudal que cada una de ellas es capaz de evacuar es diferente al de las de entrada. Se pide diseñar el circuito lógico sólo con NAND para el gobierno de las electroválvulas de salida, de tal forma que el caudal total de entrada sea igual al de salida, y teniendo en cuenta que nunca podrán estar abiertas más de dos electroválvulas de entradas.

Problema 3.9 Para poner en marcha un motor se requieren tres interruptores a, b ,c de tal forma que el funcionamiento del mismo se produzca únicamente

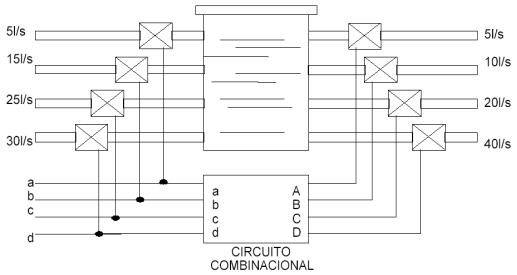


Figura 3.50.

en las siguientes condiciones:

- cuando esté cerrado solamente c.
- cuando estén cerrados simultáneamente a y c y no lo esté b.
- cuando estén cerrados simultáneamente a y b y no lo esté c.

Se pide: (a) construir la tabla de verdad, (b) implementar el circuito de mando mínimo con puertas NAND de dos entradas y (c)implementar el circuito de mando mínimo con puertas NOR de dos entradas

Capítulo 4

Subsistemas combinacionales

4.1. Circuitos integrados MSI/LSI

Las puertas lógicas y circuitos integrados estudiados en el tema anterior, sólo implementan funciones lógicas elementales, AND, OR, EXOR, etc. Son demasiado simples para poder realizar cualquier circuito integrado con un nivel de complejidad medio usando pocos componentes. Para ello, se usan circuitos integrados de mayor nivel de integración que incluyen funciones lógicas algo más complejas que la simple AND,OR, etc y que pueden resolver los problemas de dimensionado y cantidad de componentes del circuito resultante.

Todos aquellos circuitos integrados que poseen una complejidad mayor a la simple puerta lógica se denominan subsistemas combinacionales. En general, consideraremos al subsistema combinacional como un circuito multientrada y multisalida, esto es, dispone de n líneas de entrada, y genera m funciones lógicas de salida, figura 4.1.

Estas líneas de entrada y salida pueden clasificarse en:

- líneas de datos
- líneas de control

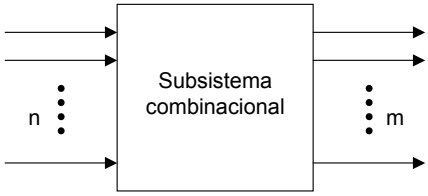


Figura 4.1.

Las primeras se corresponden con aquellas líneas de entrada o salida del circuito por las que entran los bits de datos al circuito y salen los bits del resultado generado por el mismo. Estas líneas pueden ser activas en alto (en cuyo caso un 1 es activación y un 0 desactivación) o en bajo (en cuyo caso un 0 significa activación y un 1 desactivación). Las líneas que son activas en bajo se representan con una "burbuja" (aunque en muchas ocasiones aparece también una especie de triángulo), mientras que las líneas activas en alto, carecen de ella.

Las líneas de control pueden ser, también, de entrada y salida y sirven para controlar o generar códigos necesarios para el funcionamiento del circuito. Una de las líneas de control más usadas es el habilitador (*Enable*), figura 4.2.

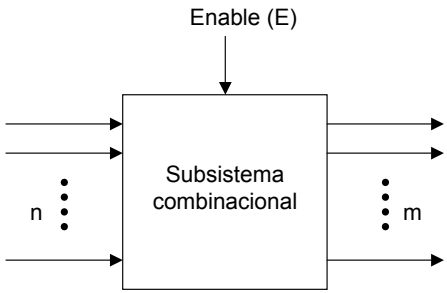


Figura 4.2.

La habilitación de un subsistema a través de la línea de *Enable* permite el funcionamiento, o no, del subsistema, y puede obtenerse cuando dicha línea de control tenga un nivel alto (un 1 lógico) o cuando tenga un nivel bajo (un 0 lógico). En el primer caso, se dice que el *Enable* es activo en alto, mientras que para el segundo, el *Enable* es activo en bajo, figura 4.3.

Podemos encontrar subsistemas que tengan múltiples entradas de habili-

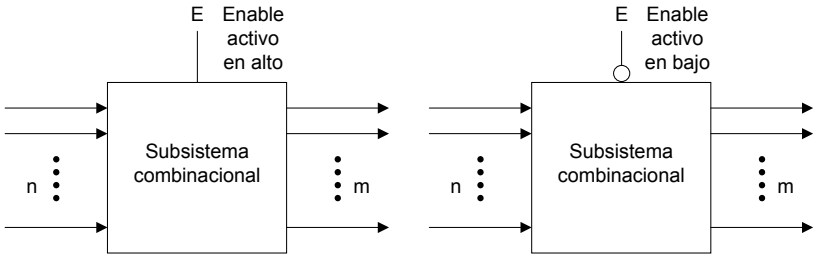


Figura 4.3.

tación, donde algunas de ellas son activas en alto y, otras, activas en bajo. Esta situación implica que el subsistema se encontrará habilitado cuando cada uno de los enables individuales tenga simultáneamente el nivel (alto o bajo) apropiado para su habilitación. Para el ejemplo siguiente, mostrado en la figura 4.4, E_1 y E_2 deben tomar el valor 0, y E_3, el valor 1, para que el subsistema se habilite y funcione adecuadamente.

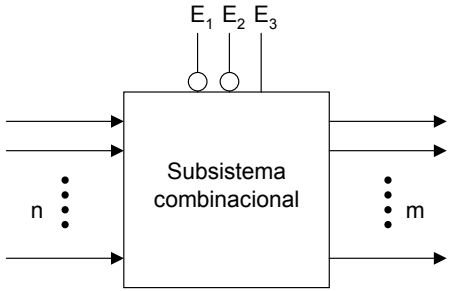


Figura 4.4.

Podemos clasificar los subsistemas combinacionales en aquellos que son de propósito específico y los que son de propósito general. Los primeros realizan unas funciones de conmutación muy concretas y no pueden modificarse o utilizarse para generar otra función distinta a la que realizan. Los segundos, pueden utilizarse para realizar cualquier función de conmutación.

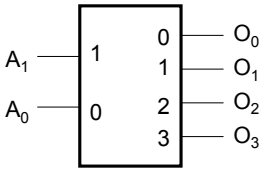
4.2. Subsistemas combinacionales de propósito específico

4.2.1. Decodificadores

El decodificador es un circuito integrado de n entradas de datos y m salidas de datos, donde $m < 2^n$. Cuando se cumple la igualdad, se dice que el decodificador es completo. Los decodificadores se especifican de la siguiente manera: DEC n:m o DEC de n a m.

El propósito de un decodificador es generar los 2^n mintérminos o maxtérminos asociados a las n variables de entrada, por tanto, el funcionamiento del mismo se deriva de esta propiedad, es decir, sólo hay una salida activa para cada combinación de entrada.

La tabla de verdad para un decodificador de 2 a 4 con salidas activas en alto se muestra en la figura 4.5.



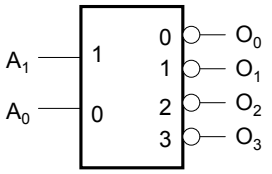
A_1	A_0	O_0	O_1	O_2	O_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Figura 4.5.

Las salidas tienen las siguientes expresiones lógicas, que son los mintérminos de las variables A_1, A_0:

$$\begin{aligned} O_0 &= \bar{A}_1 \cdot \bar{A}_0 \\ O_1 &= \bar{A}_1 \cdot A_0 \\ O_2 &= A_1 \cdot \bar{A}_0 \\ O_3 &= A_1 \cdot A_0 \end{aligned}$$

En el caso de que las salidas del decodificador de 2 a 4 fueran activas en bajo, figura 4.6, todas las salidas estarían a 1 lógico salvo la que estuviera seleccionada, que estaría a 0 lógico.



A_1	A_0	O_0	O_1	O_2	O_3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

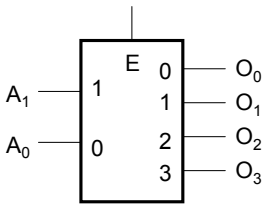
Figura 4.6.

Las salidas tienen las siguientes expresiones lógicas, que son los distintos maxtérminos de las entradas A_1, A_0:

$$\begin{aligned} O_0 &= A_1 + A_0 \\ O_1 &= A_1 + \bar{A}_0 \\ O_2 &= \bar{A}_1 + A_0 \\ O_3 &= \bar{A}_1 + \bar{A}_0 \end{aligned}$$

Por lo visto anteriormente, se obtienen mintérminos cuando las salidas del decodificador son activas en alto y maxtérminos cuando estas son activas en bajo.

Se pueden encontrar decodificadores con señales de habilitación sencillas o múltiples, tanto activas en alto como en bajo. En la figura 4.7 se representa la tabla de verdad de un decodificador de 2 a 4 con salidas activas en alto y con señal de *Enable* activa en alto.



E	A_1	A_0	O_0	O_1	O_2	O_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

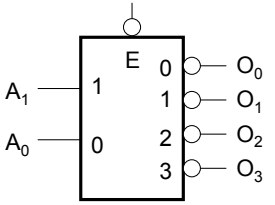
Figura 4.7.

Las salidas, ahora, tienen la siguiente expresión lógica:

$$\begin{aligned} O_0 &= \bar{A}_1 \cdot \bar{A}_0 \cdot E \\ O_1 &= \bar{A}_1 \cdot A_0 \cdot E \\ O_2 &= A_1 \cdot \bar{A}_0 \cdot E \\ O_3 &= A_1 \cdot A_0 \cdot E \end{aligned}$$

Podemos decir que la expresión de salida de un decodificador con salidas activas en alto que posee entrada de habilitación también activa en alto equivale a $O_i = m_i \cdot E$, donde m_i es el mintérmino asociado a la salida i. Si la entrada de habilitación fuese activa en bajo, la expresión de la salida i del decodificador sería $Oi = m_i \cdot \bar{E}$.

En aquellos decodificadores cuyas salidas son activas en bajo, la expresión lógica para cada una de las salidas, en el caso de que se posea línea de *Enable* activa en alto, sería $O_i = M_i + \bar{E}$, y si el *Enable* es activo en baja, $O_i = M_i + E$. La figura 4.8 representa las salidas de un decodificador de 2 a 4 con salidas activas en baja, y señal de *Enable* también activa en baja.



E	A_1	A_0	O_0	O_1	O_2	O_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Figura 4.8.

En el caso de que el decodificador tenga múltiples entradas de habilitación, la expresión de la salida i, se obtendría de forma similar a la mostrada en el siguiente ejemplo, figura 4.9. Disponemos de un decodificador de 3 a 8, con salidas activas en bajo y tres líneas de habilitación (E_1, E_2 y E_3) de las cuales, dos, son también activas en bajo.

Sustituimos los tres habilitadores por uno sólo, E, activo en alta (aunque podría asumirse, igualmente, que este fuese activo en baja). Bajo esta nueva situación, la salida i del decodificador tiene la siguiente expresión lógica, $O_i = M + \bar{E}$. Ahora, sólo necesitamos relacionar el valor de E, con los enables individuales (E_1, E_2, E_3). Si $E = 1$ el decodificador está habilitado. De forma

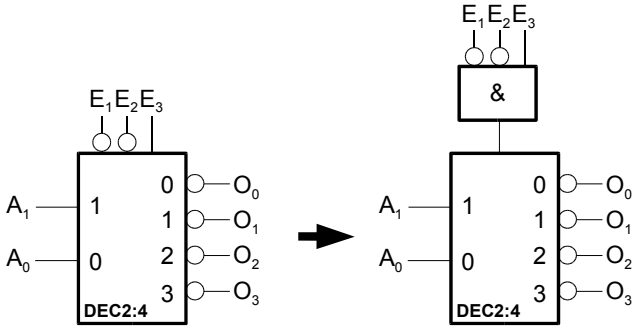


Figura 4.9.

equivalente, los valores de (E_1, E_2, E_3) para que el decodificador se encuentre habilitado son $(0, 0, 1)$ respectivamente. Por tanto:

$$E = \bar{E_1} \cdot \bar{E_2} \cdot E_3$$

Substituyendo esta expresión en O_i, obtenemos:

$$O_i = M_i + E_1 + E_2 + \bar{E_3}$$

4.2.1.1. Aplicaciones de los decodificadores

Se sabe que con n bits pueden formarse 2^n combinaciones distintas. Pues bien, un decodificador puede servirnos para indicar cuál de esos 2^n combinaciones se encuentran en la entrada.

Puesto que un decodificador genera en sus salidas los mintérminos o maxtérminos asociados a las variables de entrada, entonces podemos implementar funciones de conmutación si le añadimos las puertas necesarias para generar la suma de mintérminos o el producto de maxtérminos adecuado.

Ejemplo. Implementar con un decodificador las siguientes funciones de conmutación:

$$\begin{aligned} F_1 &= \sum(0, 3, 6) \\ F_2 &= \prod(1, 3, 4, 6) \end{aligned}$$

Solución 1. Supóngase que el decodificador tiene las salidas activas en alto. En este caso, el decodificador genera mintérminos. Necesitamos un decodificador de 3 a 8, ya que hay que generar, en principio, hasta el mintérmino 6. Por otro lado, la función F_2 viene expresada como producto de maxtérminos, por tanto, antes de implementarla, debemos expresarla como suma de mintérminos. Esto es:

$$F_2 = \prod (1, 3, 4, 6) = \sum (0, 2, 5, 7)$$

Y el circuito resultante es el de la figura 4.10.

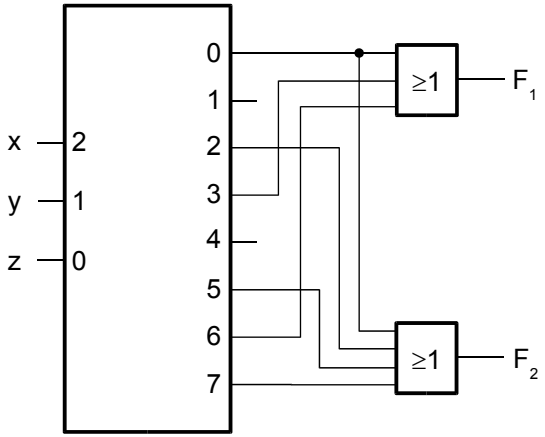


Figura 4.10.

Solución 2. Supóngase, ahora, que el decodificador tiene las salidas activas en bajo. En este caso, el decodificado genera maxtérminos. Convertimos F_1 a producto de maxtérminos.

$$F_1 = \sum (0, 3, 6) = \prod (1, 2, 4, 5, 7)$$

Y el circuito resultante es el de la figura 4.11.

Obsérvese que, en ambos casos, se han implementado las mismas funciones de conmutación de dos formas distintas: suma de mintérminos (decodificador con salidas activas en alto y puertas OR) y producto de maxtérminos (decodificador con salidas activas en bajo y puertas AND).

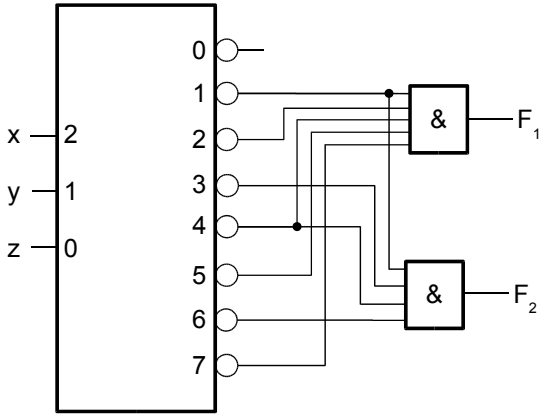


Figura 4.11.

4.2.1.2. Asociación de decodificadores

Los decodificadores pueden asociarse entre sí, para construir decodificadores que dispongan de mayor número de líneas de entrada. Como ejemplo, figura 4.12, a continuación se muestra la estructura de un decodificador de 3 a 8 con línea de E activa en alta, usando, exclusivamente, decodificadores de 2 a 4 también con línea de E activa en alta.

La línea E se conecta a la entrada de habilitación del decodificador 1, que a su vez controla los habilitadores 2 y 3. Si $E = 0$, el habilitador 1 pone todas sus salidas a 0, lo que provoca que los decodificadores 2 y 3 estén inhabilitados y todas las salidas O_i a cero lógico. Si $E = 1$, el decodificador 1 está habilitado y, ahora, la entrada A_2 controla la habilitación de los decodificadores 2 y 3. Si $A_2 = 0$, se habilita el decodificador 2 mientras que el 3 permanece inhabilitado. Si $A_2 = 1$, se habilita el decodificador 3 mientras que el 2 permanece inhabilitado. En ningún caso, ambos decodificadores (2 y 3) se encuentran simultáneamente habilitados. Las entradas A_1, A_0 controlan, para los decodificadores 2 y 3, cuál, de las cuatro salidas de cada uno se activa en cada momento. Obsérvese que, gracias a la entrada A_2, sólo un decodificador está activo y por tanto, en cada momento, sólo una, de las ocho salidas, está activa cuando $E = 1$.

Como ejemplo práctico, se va a analizar la documentación que ofrece un fabricante de un decodificador comercial. En la figura 4.13 se muestra el símbolo del decodificador 74LS138. En el símbolo gráfico se aprecia que todas las

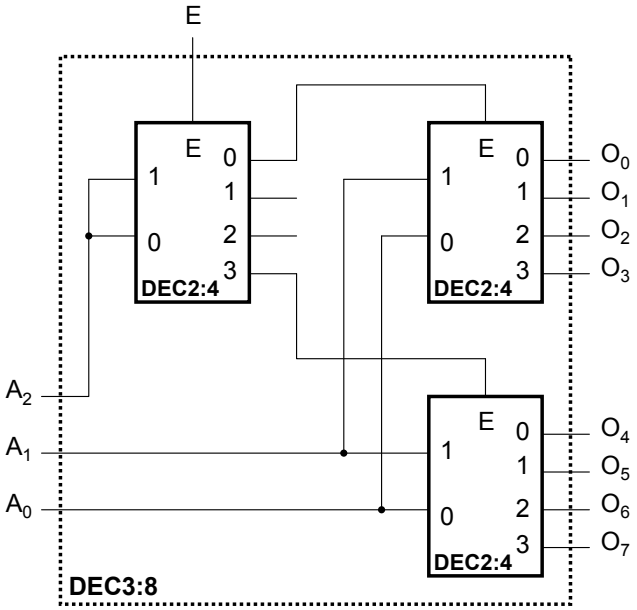

E
E
1
0
0
1
2
3
DEC2:4
E
1
0
0
1
2
3
DEC2:4
O0
O1
O2
O3
A2
A1
A0
E
1
0
0
1
2
3
DEC2:4
O4
O5
O6
O7
DEC3:8

Figura 4.12.

líneas tienen sobreimpresas un número que identifica el pin correspondiente del circuito integrado. Algunas de estas líneas tiene una especia de triángulo para informar que son activas en bajo. También se aprecian tres zonas en el símbolo: la derecha, que se corresponde con las líneas de salida activas en bajo numeradas, en el interior desde 0 a 7, y designadas, en el exterior, como $Y0-7$; la parte superior izquierda, formada por tres líneas de entrada, numeradas en el interior como 1, 2, 4 y designadas por A, B y C respectivamente; y la parte inferior izquierda, formada por tres líneas designadas como $G1$, $G\bar{2}A$ y $G\bar{2}B$, que entran en el decodificador en una región especifica que se ha destacado internamente mediante un rectángulo nombrado por EN (habilitación) y con el símbolo & en su interior. Estas tres últimas entradas son los habilitadores del decodificador, dos de los cuales son activos en baja, $G\bar{2}A$ y $G\bar{2}B$, y el otro activo en alto, $G1$, que se tienen que combinar, a modo de función AND, para conseguir la habilitación del decodificador, esto es, $G1 = 1$, $G2\bar{A} = 0$ y $G\bar{2}B = 0$. Para las líneas de entrada A, B y C se observa que se han situado los pesos asociados a cada una de ellas, 4 para la C, 2 para la B, y 1 para la A (en lugar de 2, 1, 0 como se mostrado a lo largo de este apartado), por lo que la entrada C es la más significativa y la A la menos significativa.

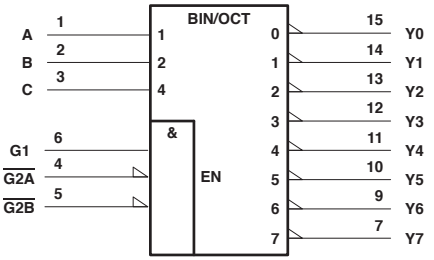


Figura 4.13.

El comportamiento del decodificador se puede seguir en la tabla de función representada en la figura 4.14. Para cualquier combinación de $(G1, G\bar{2}A, G\bar{2}B)$ que no sea (H, L, L), las salidas están en nivel alto para cualquier combinación de la entradas (C, B, A). Si $(G1, G\bar{2}A, G\bar{2}B) = (H, L, L)$, entonces una de las ocho salidas se activa dependiento de la combinación de entradas (C, B, A), por ejemplo si $(C, B, A) = (H, H, L)$ o equivalentemente $(C, B, A) = (1, 1, 0)$, la salida 6 se activa, L, mientras que las restantes están inactivas, H.

En la figura 4.15 se muestra la estructura interna de este decodificador.

FUNCTION TABLE

INPUTS						OUTPUTS							
ENABLE			SELECT										
G1	$\overline{G2A}$	$\overline{G2B}$	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	L	H	H	H	H	H	L	H	H	H	H
H	L	L	H	L	L	H	H	H	H	L	H	H	H
H	L	L	H	L	H	H	H	H	H	H	L	H	H
H	L	L	H	H	L	H	H	H	H	H	H	L	H
H	L	L	H	H	H	H	H	H	H	H	H	H	L

Figura 4.14.

Se observa que las entradas de selección pasan a una especie de inversores con dos salidas cuya función es la de generar las versiones complementadas y sin complementar de dichas entradas. Después existe un conjunto de puertas NAND que permiten generar los maxtérminos asociados a las variables (C, B, A) con la opción de inhabilitación procedente de una puerta AND que recoge las entradas de habilitación.

4.2.2. Codificadores

Los codificadores son circuitos integrados que realizan la función inversa de un decodificador. En él existe un conjunto de entradas, de las cuales sólo una puede estar activa en cada momento, y un conjunto de salidas que codifican el valor de la entrada activa en ese momento. No hay relación directa entre el número de entradas con el de salidas, aunque generalmente 2^n entradas producen n salidas. En este último caso se dice que el codificador es completo.

La tabla de verdad mostrada en la figura 4.16 representa el funcionamiento de un codificador de 4 entradas a binario natural, en el que tanto las entradas como las salidas son activas en alto.

Internamente, los codificadores se construyen con puertas OR. La figura 4.17 muestra la estructura interna del codificador del ejemplo anterior. Esto

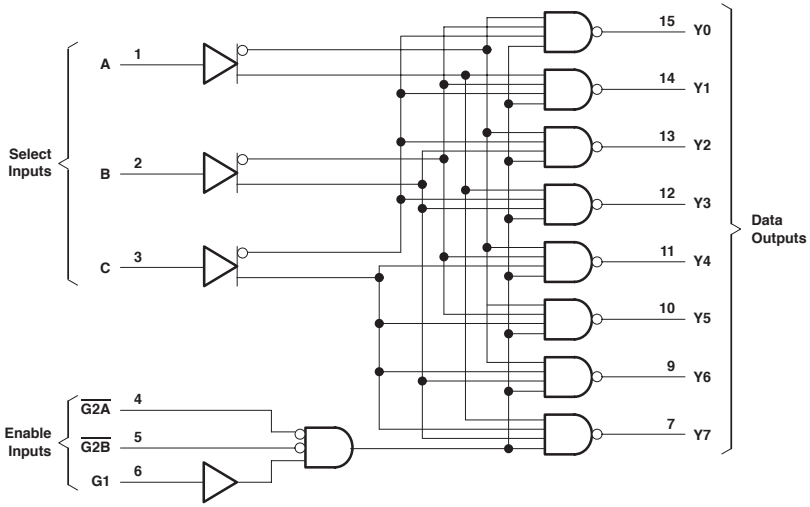
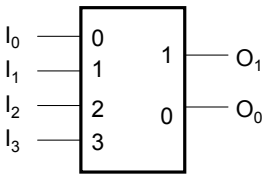


Figura 4.15.



I_0	I_1	I_2	I_3	O_1	O_0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

Figura 4.16.

puede deducirse directamente de la tabla de verdad, donde se puede ver que, por ejemplo, O_1 toma el valor 1 cuando I_2 o I_3 son 1, y O_0 toma el valor 1 cuando I_1 o I_2 también lo son.

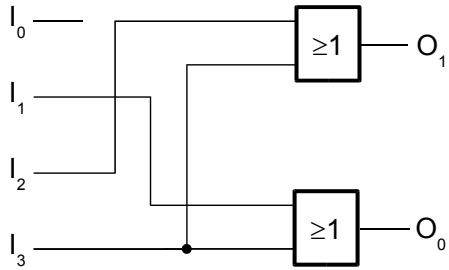
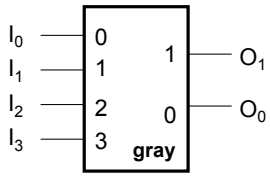


Figura 4.17.

Existen otros tipos de codificadores, como el Gray, representado en la siguiente tabla.



I_0	I_1	I_2	I_3	O_1	O_0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	1
0	0	0	1	1	0

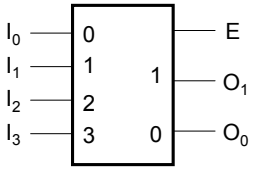
Figura 4.18.

Su estructura interna viene determinada por dos puertas OR:

$$\begin{aligned} O_0 &= I_1 + I_2 \\ O_1 &= I_2 + I_3 \end{aligned}$$

Existe otro tipo de codificador, denominado **codificador de prioridad**, en el cual puede existir más de una entrada activa simultáneamente, o ninguna entrada activa. Las salidas siguen funcionando de modo similar, pero en este caso codificando, de entre todas las entradas activas simultáneamente, aquella que posea mayor peso. Como ejemplo, a continuación ilustramos el funcionamiento de un codificador binario completo de prioridad de 4 entradas activas en alto y salidas activas en alto, figura 4.19. Este codificador posee una salida adicional, E, que actúa para indicar si existe o no alguna entrada activa y, de

esta forma, diferenciar cuándo el codificador está codificando el valor 0, de cuando no está codificando nada porque no existe ninguna entrada activa.



I_0	I_1	I_2	I_3	E	O_1	O_0
1	0	0	0	0	0	0
x	1	0	0	0	0	1
x	x	1	0	0	1	0
x	x	x	1	0	1	1
0	0	0	0	1	0	0

Figura 4.19.

Analizaremos ahora el funcionamiento de un circuito codificador comercial como es el 74178. Este codificador de prioridad tiene 8 entradas de datos más una entrada de habilitación, EI, todas ellas activas en bajo, y 3 salidas de datos activas en bajo, A_2, A_1 y A_0 que codifican las entradas anteriores, más dos salidas de control, E_0 y GS. El símbolo del codificador 74178 se ha representado en la figura 4.20 y su tabla de funcionamiento, en la figura 4.21.

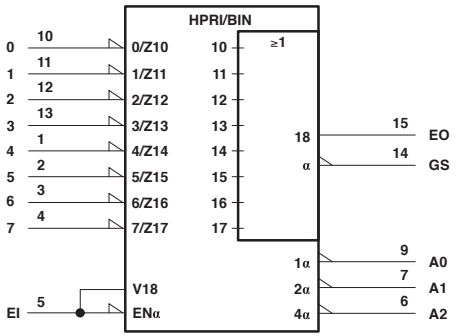


Figura 4.20.

El funcionamiento es como sigue: si la entrada EI tiene un nivel alto, el codificador está inhabilitado, por lo que todas las salidas, independientemente del valor de las entradas de datos, están inactivas. Si EI tiene un nivel bajo, el codificador está habilitado y, ahora, en función de los valores de las entradas $(0, \ldots, 7)$, se activarán las salidas (A_2, A_1, A_0), mostrando el código asociado a la entrada activa de mayor peso activa. Si, por ejemplo, la entrada activa de mayor peso es la 6, es decir, esta entrada tiene un 0 (L), la entrada 7 un 1

FUNCTION TABLE

INPUTS									OUTPUTS				
EI	0	1	2	3	4	5	6	7	A2	A1	A0	GS	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

Figura 4.21.

(H), y las restantes, 0 a 5, no importan, las salidas deben codificar el valor de la entrada 6, pero como estas son activas en baja, $(A_2, A_1, A_0) = (0, 0, 1)$, es decir, (L, L, H).

La salida E_0 se activa (es decir, vale 0, L) si el codificador está habilitado($EI = 0$) y no existe ninguna entrada activa ($0-7 = H-H$). Por otro lado, la salida GS está activa (vale 0, L) si el codificador está habilitado y hay alguna entrada activa en $0 - 7$.

4.2.3. Convertidores de código

Son circuitos integrados que transforman una palabra de un código a una palabra perteneciente a otro código. En definitiva, son traductores de código. Pueden existir de varios tipos, por ejemplo:

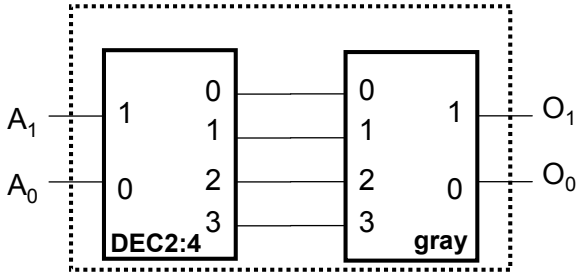
- binario/gray
- gray/binario
- BCD/gray
- BCD/7-segmentos

En la figura 4.22 se muestra la tabla de verdad de un convertidor binario/gray de dos bits.

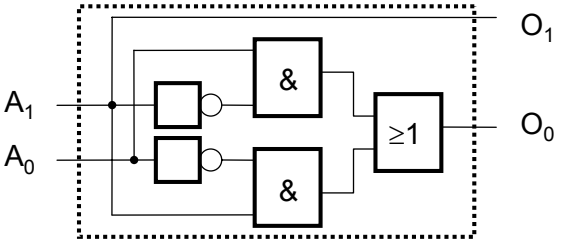
A_1	A_0	O_1	O_0
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Figura 4.22.

Los convertidores pueden diseñarse a nivel de puertas lógicas o, directamente, usando una asociación entre un decodificador y un codificador. En la figura 4.23 se han representado un convertidor de código de dos bits de binario a Gray, realizado, primero, mediante una asociación entre decodificador y codificador y, segundo, a nivel de puertas lógicas. Esta última realización se alcanza con la obtención de las expresiones mínimas obtenidas de la tabla de verdad del codificador a implementar.



(a)



(b)

Figura 4.23.

Un tipo de convertidor muy utilizado es el convertidor de BCD a código 7-segmentos, figura 4.24. El código 7-segmentos es utilizado para iluminar los distintos segmentos de un display númerico.

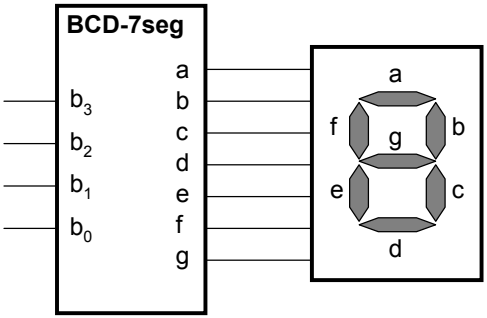


Figura 4.24.

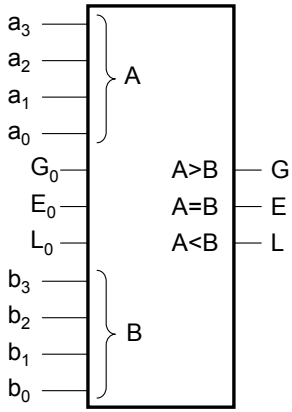
Si se desea visualizar el número 0 en el display, el segmento g debe estar apagado (entrada $g = 0$) y los restantes, encendidos (1). Si se desea visualizar el 8, todos los segmentos deben estar encendidos (1). Se puede deducir fácilmente que la tabla de verdad de la figura 4.25 se corresponde con la de dicho convertidor.

b_3	b_2	b_1	b_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	0	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1

Figura 4.25.

4.2.4. Comparadores de magnitud

Un comparador de magnitud de n bits es un circuito integrado que permite comparar dos números de n bits. Podemos asumir que este dispositivo tiene $2^n + 3$ entradas y 3 salidas. En la figura 4.26 se representa la estructura de un comparador de 4 bits, donde se aprecian las 8 entradas de datos correspondientes a los dos números binarios A y B, de cuatro bits cada uno, 3 salidas (G, E, L) y tres entradas adicionales (G_0, E_0, L_0).



A B	G E L
A>B	1 0 0
A=B	G_0 E_0 L_0
A<B	0 0 1

Figura 4.26.

La salida G indica que el número A es mayor que el B, por tanto se activa cuando $A > B$; la salida L indica que el número A es menor que B, por tanto se activa cuando $A < B$, y la salida E indica que los números son iguales. Se deduce del funcionamiento descrito que el comparador no puede tener simultáneamente activas varias de sus tres salidas. Esta propiedad será heredada por las entradas (G_0, E_0, L_0), por lo que estas sólo pueden tener un 1 lógico repartido entre las tres. Volviendo a la descripción funcional del comparador, la salida G toma el valor 1 lógico cuando $A > B$ y 0 en cualquier otro caso y la salida L toma el valor 1 lógico cuando $A < B$, y 0 en cualquier otro caso. No obstante, cuando los números A y B son iguales, existe una diferencia substancial. En este caso las tres salidas, (G, E, L) toman los valores lógicos de las entradas (G_0, E_0, L_0). Por tanto, si queremos que la salida E se active cuando $A = B$, debemos introducir en las entradas (G_0, E_0, L_0) los valores lógicos $(0, 1, 0)$ respectivamente. Este modo de funcionamiento permite

la asociación de comparadores, para poder realizar comparaciones a números que tengan una mayor cantidad de bits.

La tabla mostrada en la figura 4.27 resume el funcionamiento del comparador de magnitud de 4 bits 74LS85.

TRUTH TABLE

COMPARING INPUTS				CASCADING INPUTS			OUTPUTS		
A_3,B_3	A_2,B_2	A_1,B_1	A_0,B_0	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$	$O_{A>B}$	$O_{A<B}$	$O_{A=B}$
$A_3>B_3$	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
$A_3=B_3$	$A_2=B_2$	$A_1=B1$	$A_0>B_0$	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
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	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

Figura 4.27.

4.2.4.1. Asociación de comparadores

De las formas posibles de asociación de comparadores de magnitud para construir comparadores con capacidad de proceso de números que contengan más bits, presentaremos la más elemental: la asociación en serie. Veremos el proceso de asociación con un ejemplo.

Se dispone de comparadores de magnitud de 4 bits, y se desea construir un comparador que permita comparar dos números A y B de 12 bits. La solución a este problema se presenta en la figura 4.28.

Los 4 bits más significativos de los dos números A y B, se introducen en las entradas del comparador 1. Si la magnitud asociada a los cuatro bits a_{11-8} es mayor que la asociada a los cuatro bits b_{11-8}, entonces $A > B$ y se activa

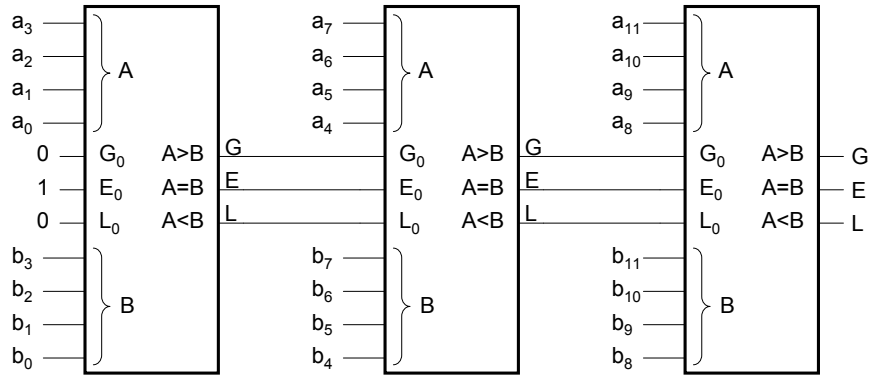


Figura 4.28.

la salida G; si la magnitud asociada a los cuatro bits a_{11-8} es menor que la asociada a los bits b_{11-8}, entonces $A < B$ y se activa la salida L. En el caso en que las magnitudes asociadas a los bits a_{11-8} y b_{11-8} sean iguales, las salidas (G, E, L) toman el valor de sus entradas (G_0, E_0, L_0), las cuales dependen a su vez de las salidas (G, E, L) del comparador 2. Este nuevo comparador tiene, en sus entradas de datos, los bits a_{7-4} y b_{7-4} de los números A y B. Repitiendo el proceso anterior veremos que la salida G del comparador 1 se activará en este caso si $a_{11-8} = b_{11-8}$ y $a_{7-4} > b_{7-4}$; la salida L si $a_{11-8} = b_{11-8}$ y $a_{7-4} < b_{7-4}$; y en el caso en que también las magnitudes asociadas a los bits a_{7-4} y b_{7-4} sean iguales, las salidas (G, E, L) del comparador 1 son iguales a las entradas (G_0, E_0, L_0) del comparador 2, las cuales están controladas a su vez por las salidas (G, E, L) del comparador 3. Este último comparador realiza las mismas funciones que sus antecesores pero con los bits a_{3-0} y b_{3-0} de los dos números. En este caso, las entradas (G_0, E_0, L_0) de este comparador tienen las constantes $(0, 1, 0)$ respectivamente, por lo que si ocurre una situación de igualdad entre las magnitudes asociadas a los 4 bits menos significativos de ambos números A y B, la salida E se pondrá a 1 lógico.

4.2.5. Demultiplexor

Es un circuito MSI que dispone de n líneas de control de entrada y $2^n + 1$ líneas de datos. 2^n líneas de datos son de salida (también llamadas canales) frente a una única línea de entrada. Se pueden designar mediante DEMUX

1:m, donde m es el número de canales. Recordemos que los canales deben ser una potencia del número 2. Otra forma de designarlos es DEMUX de n entradas de control. El símbolo usado para representar al demultiplexor es el mostrado en la figura 4.29.

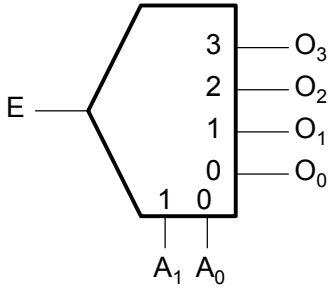


Figura 4.29.

El funcionamiento de un demultiplexor puede entenderse mediante su símil mecánico. En la figura 4.30 se ha representado un sistema interruptor consistente en una línea de entrada y 4 líneas de salida (canales). El interruptor está construido sobre un eje sobre el que puede girar y provocar la conexión de la entrada con uno de los cuatro canales de salida. El canal de salida escogido para la conexión es determinado por las dos líneas de control.

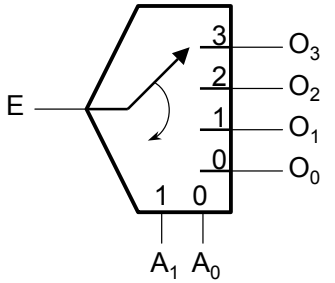


Figura 4.30.

En efecto, un demultiplexor es un circuito digital que permite "pasar" el valor lógico de la entrada a un canal de salida seleccionado por las líneas de control. Un canal de salida seleccionado muestra el valor lógico de la entrada, mientras que un canal de salida no seleccionado genera el valor lógico 0. La

tabla de verdad de la figura 4.31 muestra el funcionamiento de un demultiplexor de 4 canales.

E	A_1	A_0	O_0	O_1	O_2	O_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Figura 4.31.

Es fácil comparar el funcionamiento de este dispositivo con el de un decodificador con entrada de habilitación. En este último dispositivo cuando la entrada de habilitación es 0, todas sus salidas son cero, mientras si la entrada de habilitación es 1, se activará la salida correspondiente a la entradas situadas en el decodificador. Esta similitud provoca que no exista diferenciación entre algunos decodificadores y los demultiplexores, los cuales se suelen presentar como un mismo circuito.

4.3. Subsistemas de propósito general

4.3.1. Multiplexor

Es un dispositivo que consta de 2^n entradas de datos (también llamados canales), n entradas de selección de canal y una salida de datos. Pueden ser designados mediante MUX $2^n : 1$, o MUX de n líneas de control. Destacamos el hecho de que los canales y las entradas de selección de canal están relacionados mediante potencia de 2. La figura 4.32 representa el símbolo gráfico de un multiplexor de cuatro canales (MUX 4:1).

El multiplexor realiza la función inversa a la del demultiplexor. La salida z muestra el valor lógico existente en el canal de entrada que es seleccionado por las entradas A_1 y A_0. El símil mecánico del multiplexor puede verse en la

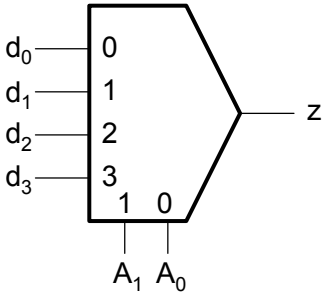


Figura 4.32.

figura 4.33. Las entradas de selección de canal fijan la posición del interruptor mecánico y, por tanto, la salida muestra el valor del canal seleccionado.

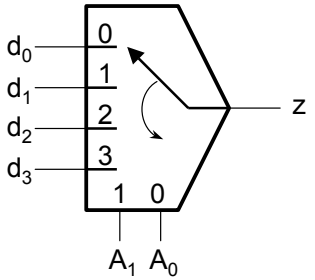


Figura 4.33.

La tabla de verdad simplificada del multiplexor de cuatro canales y su estructura interna se representan en la figura 4.34.

De la tabla se deduce que la expresión lógica del multiplexor es:

$$Z = d_0\bar{A}_1\bar{A}_0 + d_1\bar{A}_1A_0 + d_2A_1\bar{A}_0 + d_3A_1A_0$$

Se puede comprobar que la salida de un multiplexor se corresponde a una suma de los mintérminos asociados a las entradas de selección, los cuales están multiplicados por los valores de los canales asociados.

Los multiplexores pueden disponer, además, de una línea de habilitación denominada normalmente ***strobe***, figura 4.35. Estas líneas pueden ser acti-

A_1	A_0	z
0	0	d_0
0	1	d_1
1	0	d_2
1	1	d_3

Figura 4.34.

vas en alto o en bajo. Cuando están activas, el multiplexor opera de la forma descrita con anterioridad. Si están inactivas, la salida del multiplexor es un 0 lógico.

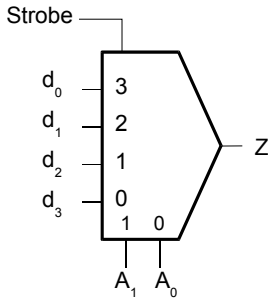


Figura 4.35.

4.3.1.1. El multiplexor como generador de funciones

El teorema de expansión de Shannon estudiado en el tema 3 indica que cualquier función de conmutación completa de n variables puede ser expresada como:

$$F(x_1, \ldots, x_n) = \sum f(i) \cdot m_i(x_1, \ldots, x_n)$$

La expresión anterior se asemeja a la salida de un multiplexor de n entradas de selección:

$$Z = \sum d_i \cdot m_i(A_1, \ldots, A_n)$$

Por tanto, si suponemos que las entradas de selección de canal $A_1, \ldots, A_n$

son las variables de la función $x_1, \ldots, x_n$ y en los canales situamos las constantes 0 y 1 correspondientes a los valores lógicos que toma la función de conmutación para cada condición de entrada asociada a las variables $d_i = f(i)$, la salida z se corresponderá con la implementación de dicha función.

Ejemplo 1. Usando un multiplexor de 8 canales, implementar la función de conmutación $f = \sum(0, 4, 5, 7)$. La solución se muestra en la figura 4.36.

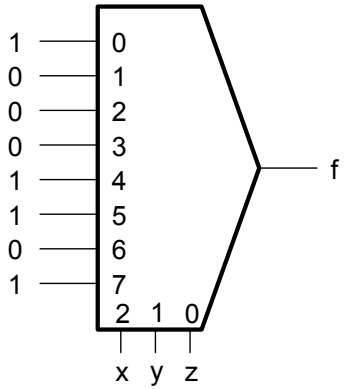


Figura 4.36.

El método para la implementación de funciones de conmutación con multiplexores se hace evidente en primera instancia, ya que escogemos un MUX con tantas entradas de selección como variables tenga la función. Cada una de las variables se conecta a una entrada de selección. Ahora bien, cada combinación de entrada que tome las variables de la función, dicha función de conmutación toma un valor lógico (0 ó 1). Este valor se sitúa en el canal que se encuentra actualmente seleccionado para esa combinación de entrada.

No obstante, en la mayoría de las ocasiones las funciones se pueden implementar con multiplexores de menor número de canales o bien, el número de canales del multiplexor es un parámetro que fija el problema, por lo que no siempre podremos aplicar el método de diseño enseñado. Para el caso general, se siguen una pautas que mostraremos en los siguientes ejemplos.

Ejemplo 2. Diseñar la función de conmutación $f = \sum(0, 4, 5, 7)$ usando multiplexores de 4 canales.

En primer lugar, de las tres variables que tiene la función, se escogen dos para ser conectadas en las entradas de selección de canal del multiplexor (p.e.: x, y). Como la salida del multiplexor debe corresponderse con la función de conmutación, entonces:

$$F(x, y, z) = d_0\bar{x}\bar{y} + d_1\bar{x}y + d_2x\bar{y} + d_3xy$$

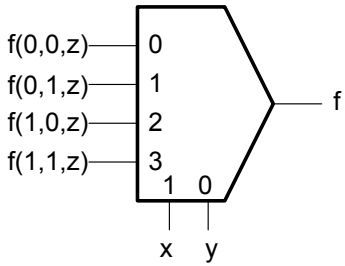


Figura 4.37.

Por lo que los canales deben tener asociados las "subfunciones" de f resultantes de extraer las variables x e y. Estas funciones se denominan **funciones residuo**, figura 4.37. El K-mapa de la figura 4.38 representa las cuatro funciones resultantes de este ejemplo y sus expresiones lógicas.

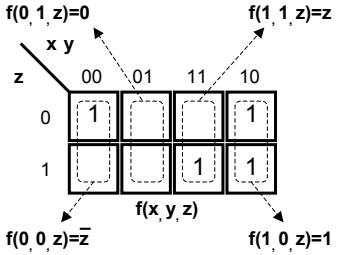


Figura 4.38.

Por tanto, y suponiendo que las variables están disponibles en doble raíl, la solución al problema sería la ilustrada en la figura 4.39.

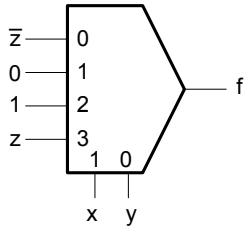


Figura 4.39.

Ejemplo 3. Diseñar la función de conmutación $f = \sum(0, 4, 5, 7)$ usando multiplexores de 2 canales.

Se procede de modo similar al ejemplo anterior. En primer lugar escogemos de las tres variables de la función la que se conectará a la entrada de selección de canal del multiplexor, por ejemplo, x. La salida del multiplexor debe corresponderse con la función de conmutación, figura 4.40.

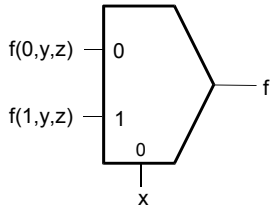


Figura 4.40.

Por lo que los canales 0 y 1 tienen las funciones residuo $f(0, y, z)$ y $f(1, y, z)$ respectivamente. El K-mapa de la figura 4.41 representa los residuos de dicha función y sus expresiones lógicas.

En este caso, las funciones residuo no son una simple variable (complementada o sin complementar), ni las constantes 0 ó 1, sino que son expresiones algebraicas que deben implementarse usando otros multiplexores. Por tanto procedemos con esta técnica de diseño, pero ahora con las funciones residuo. Por ejemplo, la función residuo $f(0, y, z)$ de dos variables, va a ser generada por un nuevo multiplexor de 2 canales cuya entrada de selección de canal va a ser, por ejemplo, y, tal como aparece en la figura 4.42. Entonces la salida de este nuevo multiplexor será:

$$f(0, y, z) = d_0\bar{y} + d_1 y$$

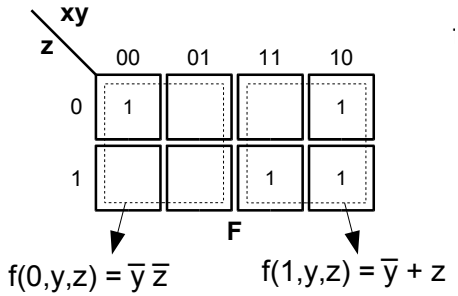


Figura 4.41.

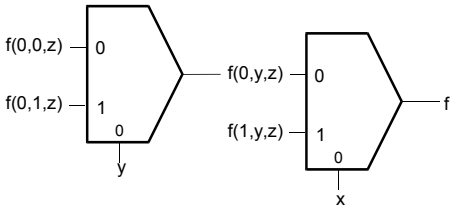


Figura 4.42.

Por lo tanto, los canales 0 y 1 de este nuevo multiplexor, están ligados a las funciones residuo $f(0,0,z)$ y $f(0,1,z)$ respectivamente. El K-mapa de la figura 4.43 representa la función $f(0,y,z)$, y sus dos residuos con sus expresiones lógicas.

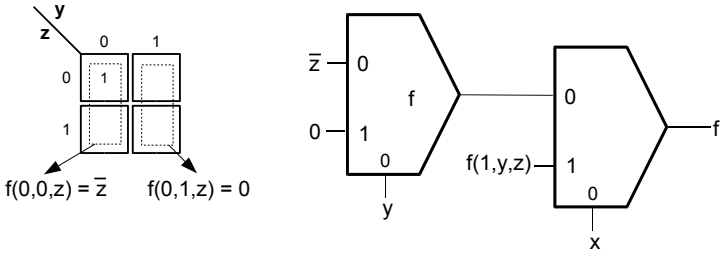


Figura 4.43.

Repitiendo el proceso para el segundo residuo, $f(1,y,z)$, para el que, igualmente extraemos la variable y, obtendremos:

$$f(1,y,z) = d_0\bar{y} + d_1 y$$

y de ahí el circuito de la figura 4.44.

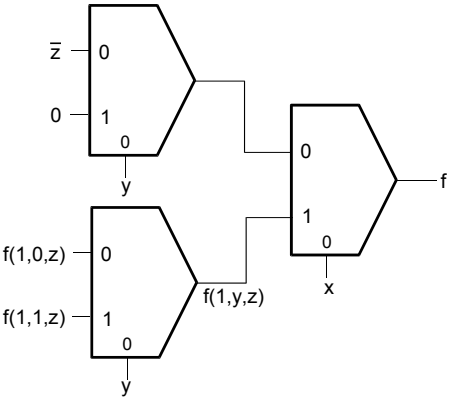


Figura 4.44.

Los canales 0 y 1 del último multiplexor se corresponden con los residuos $f(1,0,z)$ y $f(1,1,z)$ respectivamente. El K-mapa de la figura 4.45 representa la función $f(1,y,z)$, y sus dos residuos con sus expresiones lógicas.

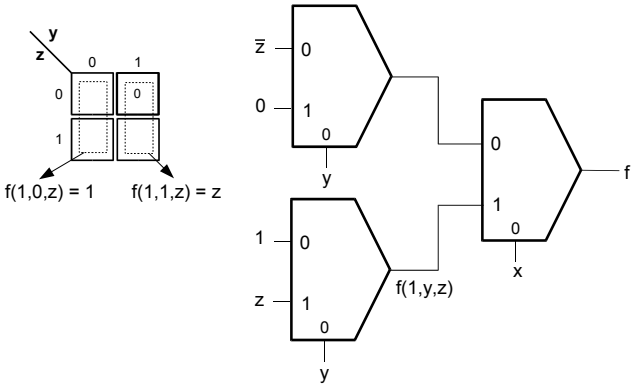


Figura 4.45.

Este mismo ejemplo se podría resolver de forma que el resultado tuviera un coste menor, esto es, un número menor de multiplexores. Supongamos que en el primer paso, en lugar de escoger la variable x, escogemos la z. En este caso,

$$f(x,y,z) = d_0\bar{z} + d_1 z$$

cuya implementación comenzaría con el multiplexor de la figura 4.46.

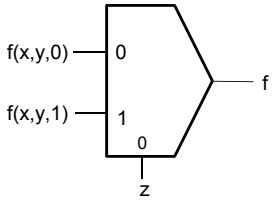


Figura 4.46.

Así, los residuos a colocar en los canales, y sus expresiones lógicas son las mostradas en la figura 4.47.

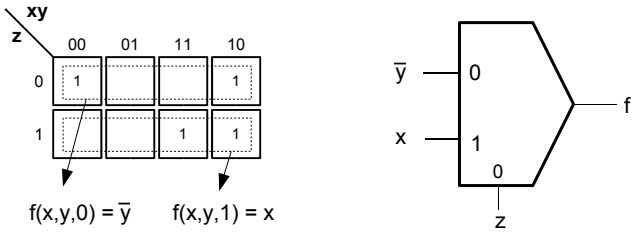


Figura 4.47.

4.3.1.2. Asociación de multiplexores

El proceso de asociación consiste en poder construir multiplexores de mayor número de canales a partir de multiplexores con menor número de canales. Veamos el proceso de asociación con un ejemplo. Supongamos que se desea construir un multiplexor de 8 canales usando multiplexores de 2 canales.

La expresión de salida del multiplexor de 8 canales es la siguiente:

$$\begin{aligned} Z(A_2, A_1, A_0) &= d_0\bar{A}_2\bar{A}_1\bar{A}_0 + d_1\bar{A}_2\bar{A}_1A_0 + d_2\bar{A}_2A_1\bar{A}_0 + d_3\bar{A}_2A_1A_0 + \\ &\quad d_4A_2\bar{A}_1\bar{A}_0 + d_5A_2\bar{A}_1A_0 + d_6A_2A_1\bar{A}_0 + d_7A_2A_1A_0 \end{aligned}$$

Pero esta función debe ser generada por multiplexores de 2 canales. Por tanto, seguimos los mismos pasos que para la implementación de funciones.

Usamos un multiplexor de 2 cuya salida es la función Z, y la línea de selección de canal es A_2, tal como aparece en la figura 4.48.

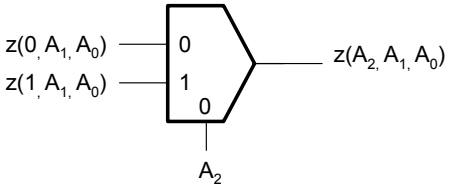


Figura 4.48.

La función residuo $Z(0, A_1, A_0) = d_0\bar{A}_1\bar{A}_0 + d_1\bar{A}_1A_0 + d_2A_1\bar{A}_0 + d_3A_1A_0$ y la función residuo $Z(1, A_1, A_0) = d_4\bar{A}_1\bar{A}_0 + d_5\bar{A}_1A_0 + d_6A_1\bar{A}_0 + d_7A_1A_0$. Estas nuevas funciones residuo pueden implementarse usando sendos multiplexores de 2 canales cuyas líneas de selección de canal están controladas por la variable A_1, figura 4.49.

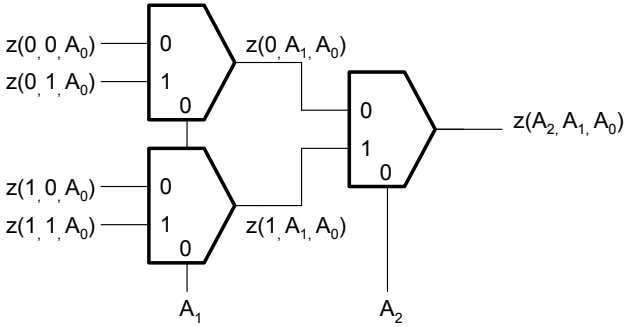


Figura 4.49.

Los cuatro residuos resultantes pueden implementarse, a su vez, con cuatro multiplexores de dos canales:

$$\begin{aligned} Z(0,0,A_0) &= d_0\bar{A}_0 + d_1A_0 \\ Z(0,1,A_0) &= d_2\bar{A}_0 + d_3A_0 \\ Z(1,0,A_0) &= d_4\bar{A}_0 + d_5A_0 \\ Z(1,1,A_0) &= d_6\bar{A}_0 + d_7A_0 \end{aligned}$$

El circuito resultante se muestra en la figura 4.50.

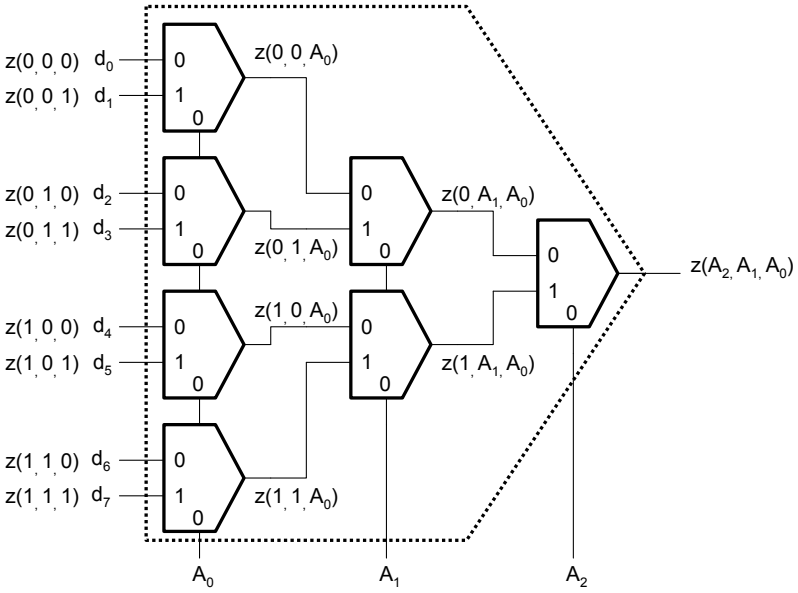

z(0, 0, 0) d0
z(0, 0, 1) d1
z(0, 1, 0) d2
z(0, 1, 1) d3
z(1, 0, 0) d4
z(1, 0, 1) d5
z(1, 1, 0) d6
z(1, 1, 1) d7
z(0, 0, A0)
z(0, 1, A0)
z(1, 0, A0)
z(1, 1, A0)
z(0, A1, A0)
z(1, A1, A0)
z(A2, A1, A0)
A0
A1
A2

Figura 4.50.

4.3.2. ROM (Read Only Memory)

Son memorias en las que sólo se puede leer la información que previamente se programó. La información que contienen no se pierde aunque sea desconectada la alimentación del circuito. La ROM consta de n entradas que, en conjunto, se denominan **bus de direcciones**, y de m salidas de datos que, juntas, forman el **bus de datos**. Adicionalmente tienen algunas señales de control de las cuales comentaremos sólo una de ellas. La ROM se identifica determinando su capacidad de almacenamiento. Esta capacidad de almacenamiento se mide normalmente en bits y está relacionada con las líneas de direcciones n, y las lineas de datos m, mediante la siguiente expresión:

$$Capacidad(bits) = 2^n \times m$$

Por tanto, una ROM de 8x2, es una ROM de 16 bits que tiene 3 líneas en su bus de direcciones y 2 líneas en el de datos. De igual forma, una ROM de 256x8 bits es una ROM que tiene 8 líneas en su bus de direcciones y 8 en su bus de datos.

Internamente, podemos considerar que la ROM está formada por un conjunto de celdas (que contienen un 1 o un 0) organizadas en 2^n filas y m columnas. De forma que cuando se selecciona una fila (para una determinada combinación de valores en el bus de direcciones), los bits contenidos en las celdas de dicha fila, salen por el bus de datos. Para el ejemplo de la figura 4.51, si se selecciona la fila 3, $A_2 - A_0 = 011$, la salida $D_3 - D_0 = 1110$, si se selecciona la fila 6, $A_2 - A_0 = 110$, la salida $D_3 - D_0 = 0001$.

Una ROM de $2^n \times m$, está formada por un decodificador de m líneas de entrada, $2^n \times m$ fusibles o interconexiones y m puertas OR. La figura 4.52 esquematiza una ROM de 4x2.

El decodificador, en función de la combinación de entrada aplicada al bus de direcciones activa una fila (esto es, pone un 1 lógico a la salida de la fila 0, 1, 2 ó 3). Los fusibles permiten la interconexión de las salidas del decodificador con las entradas de las puertas OR, las cuales generan cada una de las líneas del bus de datos. Un fusible puede estar fundido o sin fundir. En el primer caso, se elimina la interconexión entre la fila y la columna correspondientes. Esto provoca que dicha columna introduzca un 0 lógico a la entrada de la OR, gracias a la resistencia de *pull-down*. En el segundo caso, el fusible sin fundir,

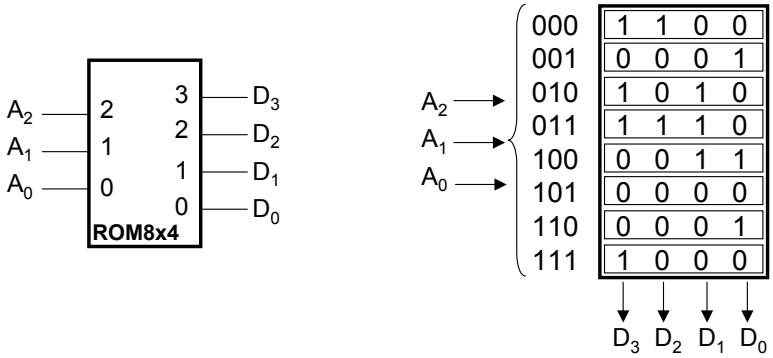


Figura 4.51.

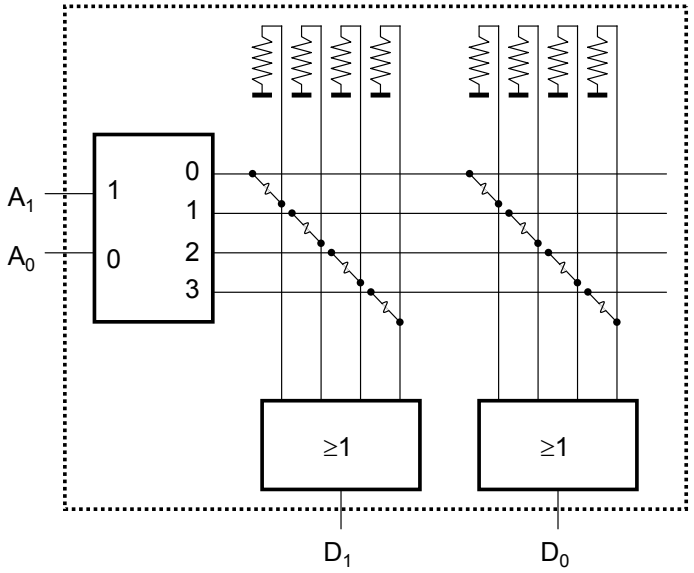


Figura 4.52.

se establece conexión entre la fila y la columna; y, en el supuesto que dicha fila esté activa (1 lógico), la columna correspondiente también se activa, provocando que se active la salida asociada del bus de datos. Podemos intuir que la existencia de un fusible fundido o sin fundir provoca el almacenamiento de un 0 o un 1 respectivamente.

Analicemos el siguiente ejemplo, donde aparecen fusibles fundidos y sin fundir. Si las entradas de la ROM son $A_1A_0 = 00$, sólo la salida 0 del decodificador tiene un 1, mientras que las restantes tienen un 0. El 1 lógico de la salida 0 del decodificador pasa a la salida D_1 de la ROM, mientras que por la salida D_0 sale un 0 lógico por tener el fusible correspondiente fundido. Si las entradas de la ROM son $A_1A_0 = 01$, sólo la salida 1 del decodificador tiene un 1, el cual no tiene camino físico de salida hacia el bus de datos. Por tanto $D_1D_0 = 00$. Si $A_1A_0 = 10$, la salida 2 del decodificador tiene un 1, el cual provoca que $D_0 = 1$, mientras que ahora $D_1 = 0$. Por último, si $A_1A_0 = 11$, las salidas $D_1D_0 = 11$, por tener intactos los fusibles correspondientes. El resultado se muestra en la figura 4.53.

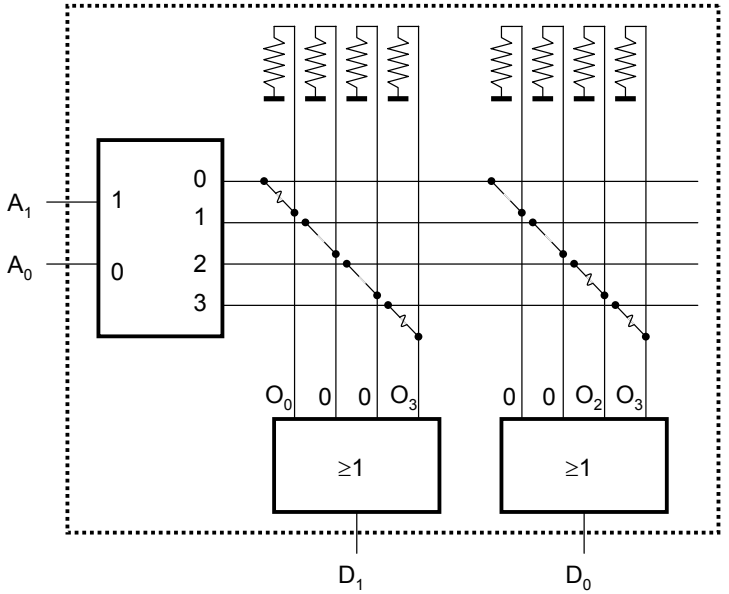


Figura 4.53.

Si hacemos una representación esquemática de la ROM anterior, y de su

contenido, nos queda lo que aparece en la figura 4.54.

A_1	A_0	D_1	D_0
0	0	1	0
0	1	0	0
1	0	0	1
1	1	1	1

Figura 4.54.

Si comparamos el contenido obtenido de la ROM con la estructura de fusibles, veremos fácilmente que un 1 es equivalente a un fusible sin fundir y un 0 a un fusible fundido.

Una forma más simple de representar la estructura interna de la ROM aparece en la figura 4.55. Se han substituido las múltiples líneas de conexión vertical de cada OR, por una simple línea y la existencia de un fusible por una aspa $\times$. De esta forma, si existe una aspa entre una línea horizontal y la vertical, indica que en esa posición hay almacenado un 1, y si no existe aspa en el cruce, hay almacenado un 0.

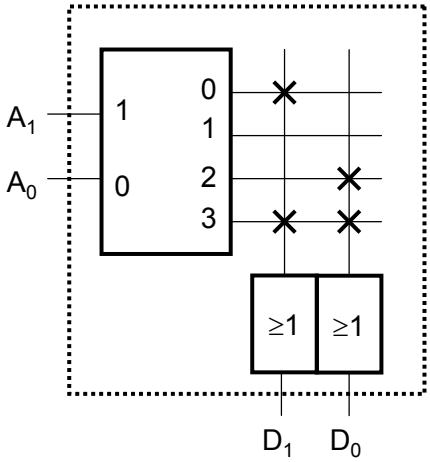


Figura 4.55.

Los dispositivos ROM tienen líneas de control que permiten su habilitación, lectura, etc. De estas líneas sólo mostraremos CS, ***Chip Select***, cuya fun-

ción es similar a la de la entrada *enable* en los decodificadores y puede aparecer activa en alto o en bajo. En el caso de que CS esté inactivo, la ROM esta inhabilitada, pasa a un modo de bajo consumo (*standby*) y sus salidas se ponen en **alta impedancia**. Si CS está activo, la ROM está habilitada y opera del modo descrito con anterioridad.

Las salidas en alta impedancia son debidas a la existencia de unos dispositivos llamados ***buffers* triestado** situados entre las salidas de las puertas OR (internas a la ROM) y las salidas físicas del bus de datos, figura 4.56.

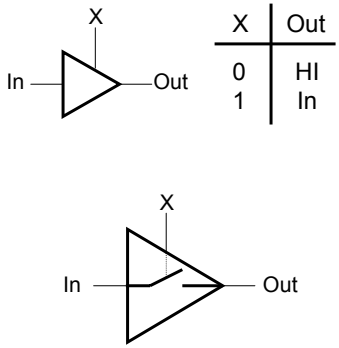


Figura 4.56.

El estado de alta impedancia consiste en una desconexión lógica de la salida. Podemos asumir que el buffer triestado es una especie de interruptor que cuando está cerrado, $x = 1$, la salida es igual a la entrada, y cuando esta abierto, $x = 0$, la salida se ha desconectado físicamente de la entrada.

En una ROM, los buffers triestado están controlados por la línea CS, según se muestra en la figura 4.57.

4.3.2.1. La ROM como generador de funciones de conmutación

La implementación de funciones de conmutación, en el caso de suma de mintérminos, requiere de la existencia mínima de una puerta OR, y de las puertas AND necesarias para la realización de los productos. Una ROM permite ambas cosas. Por un lado existe un decodificador que permite generar todos los mintérminos asociados a las entradas de direcciones; y por otro lado

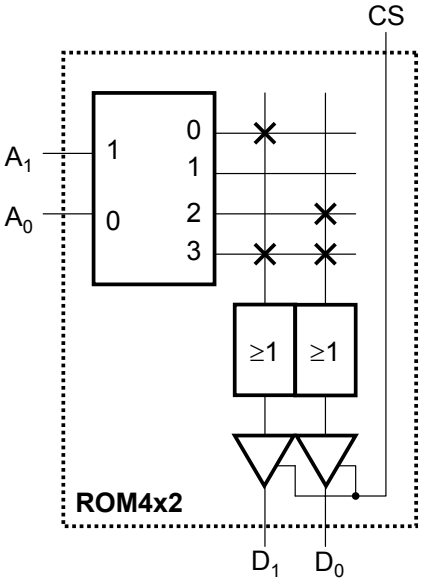


Figura 4.57.

tenemos una matriz programable que permite seleccionar aquellos mintérminos que pasarán a la entrada de la OR. En resumen, una ROM de $2^n \times m$ bits permite generar m funciones de conmutación de n variables cada una.

Implementar una o varias funciones en una ROM equivale a indicar cuál debe ser el contenido de la misma, o lo que es equivalente, dar su **tabla de programación**. La tabla de programación muestra los bits almacenados en cada dirección de memoria.

Ejemplo 1. Implementar las funciones $f = \sum(1,3,5,7)$ y $g = \sum(0,2,4,5)$ haciendo uso de una ROM.

Como mínimo la ROM debe tener la capacidad de 8x2 bits. El número 2 indica que tiene dos salidas, ya que vamos a programar dos funciones de conmutación f y g; y el número 8 indica que el bus de direcciones tiene las 3 líneas necesarias para conectar las tres variables de las dos funciones f y g. La tabla de programación debe indicar, para cada dirección, cuál debe ser el contenido de la ROM, para que la salidas se correspondan con las funciones f y g, figura 4.58. En primer lugar haremos que la variable x se conecte a la línea A_2 del bus de direcciones, la variable y a la línea A_1 y la variable z a la línea A_0. Asimismo, la salida D_1 será la responsable de generar la función f, mientras que la D_0, la función g.

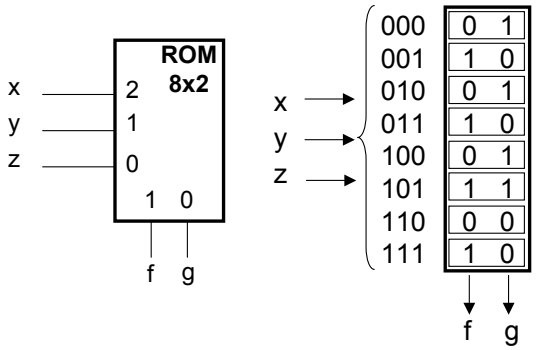


Figura 4.58.

En general, podemos decir que generar la tabla de programación de una ROM que implementa funciones de conmutación se reduce a algo tan simple

como mostrar la tabla de verdad de dichas funciones. Es común que la tabla de programación de una ROM, cuando esta dispone de muchas líneas de entrada y salida, se exprese de forma más reducida usando el código hexadecimal. Así, por ejemplo, la tabla de programación del ejemplo anterior sería la mostrada en la figura 4.59.

POS[$]	CONT[$]
0	1
1	2
2	1
3	2
4	1
5	3
6	0
7	2

Figura 4.59.

4.3.2.2. Asociación de ROM

La asociación de ROM busca la construcción de ROM de mayor capacidad a partir de dispositivos ROM de menor capacidad. La obtención de una mayor capacidad puede conseguirse de tres formas distintas: aumentando la capacidad de direccionamiento (incrementar n, m fija); aumentando el tamaño del bus de datos (n fija, incrementar m); y aumentando tanto la capacidad de direccionamiento como el bus de datos (incrementar n y m). Es evidente que este última posibilidad se obtiene de mezclar las dos anteriores. Por eso, estudiaremos las dos primeras posibilidades mediante el empleo de ejemplos.

Ejemplo 1. Se dispone de ROM de 8x2 y se desea construir una ROM de 8x4.

Estamos en el caso de aumentar el número de bits del bus de datos. El diseño resultante puede verse en la figura 4.60, donde se han interconectado juntas las líneas correspondientes a los buses de direcciones. Por otro lado, el conjunto de líneas formadas a partir de las dos líneas de datos que dispone

cada dispositivo suman las cuatro líneas de datos necesarias para la ROM de 3x4.

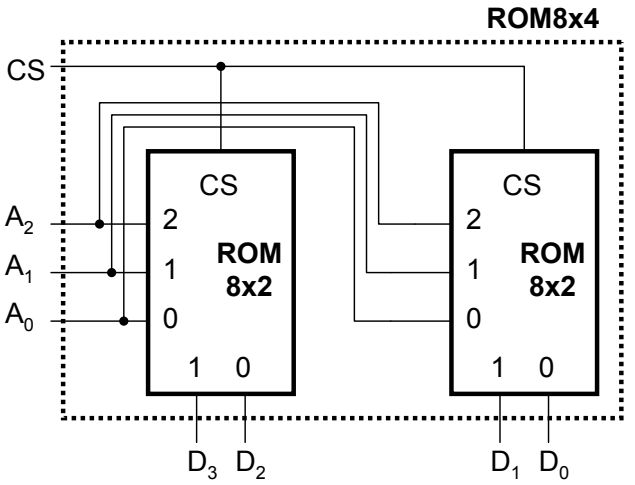


Figura 4.60.

Ejemplo 2. Se dispone de ROMs de 8x2, y se desea construir una ROM de 16x2.

La figura 4.61 muestra una posible solución a esta asociación. Como puede verse las líneas del bus de direcciones de las dos ROMs se han interconectado. La línea que falta, A_3, se conecta a los CS de las dos ROM, entre las que se conecta un inversor. De esta forma, si $A_3 = 0$, se encuentra activa la ROM de la derecha, y esta, en funcion de A_{2-0}, coloca su contenido en el bus de datos. Si $A_3 = 1$, está habilitada la ROM de la izquierda y, esta, en funcion de A_{2-0} coloca su contenido en el bus de datos. Hay que destacar el hecho de que los buses de datos de ambas ROM han sido interconectados. Esto es posible porque las ROM disponen de alta impedancia.

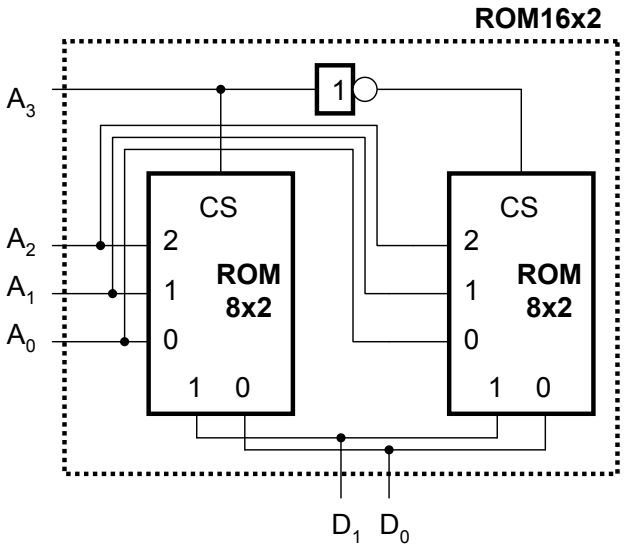


Figura 4.61.

4.3.3. PLDs (Programmable Logic Devices)

Los dispositivos lógicos programables son circuitos integrados que permiten la programación de la función de conmutación deseada. En estos dispositivos podemos encontrar dos planos o matrices (arrays): el plano AND, que genera los términos productos necesarios; y el plano OR que genera la suma de los términos productos resultantes del plano anterior. Estrictamente una ROM es un PLD que dispone de estos dos planos, de los cuales el plano AND no es programable mientras que el OR sí lo es. De los dispositivos que vamos a estudiar: la PAL se caracteriza porque el plano programable es el AND, mientras que el OR es fijo; y la PLA se caracteriza porque ambos planos son programables.

4.3.3.1. PAL (Programmable Array Logic)

Los dispositivos PAL son unos circuitos programables que poseen el plano AND programable, y el plano OR fijo. La figura 4.62 muestra la estructura de una PAL elemental de tres entradas, tres salidas, capaz de realizar 3 términos productos para cada salida.

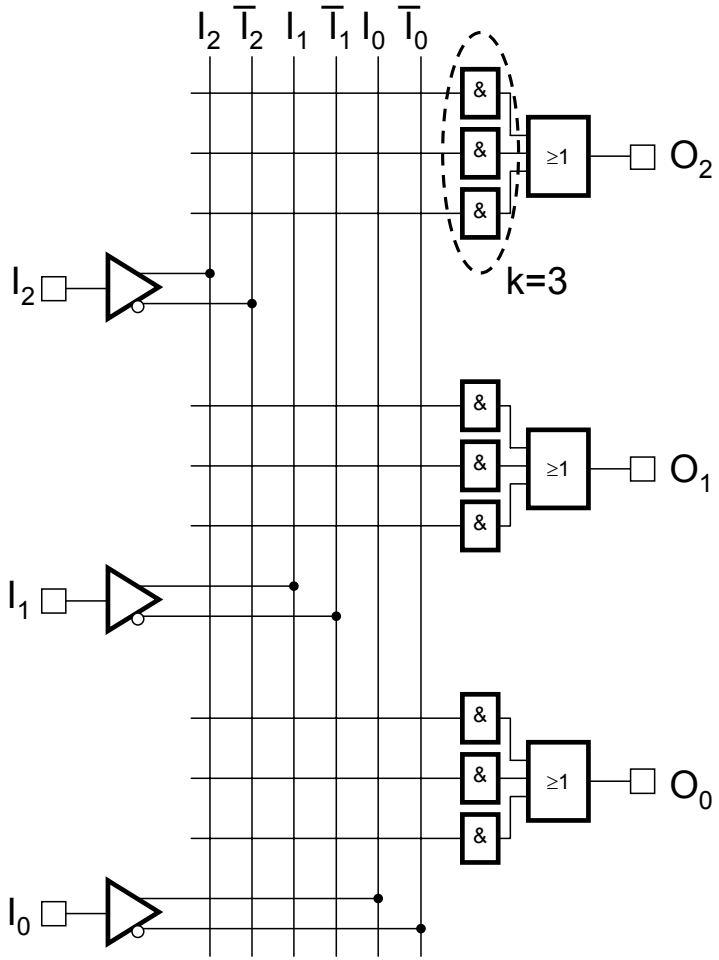

I2 $\overline{I_2}$ I1 $\overline{I_1}$ I0 $\overline{I_0}$
&
&
&
≥1
O2
k=3
I2
&
&
&
≥1
O1
I1
&
&
&
≥1
O0
I0

Figura 4.62.

Este dispositivo permite programar 3 funciones de conmutación distintas de 3 variables máximo, con la limitación de que dichas funciones no contengan más de 3 términos producto cada una.

Analicemos con más detalle el circuito PAL anterior. En primero lugar, se observa que las entradas tienen unos buffers con doble salida cuya función es la de introducir, en el plano AND programable, las variables de entrada en doble raíl. Si x es el valor de entrada de dicho buffer, a salida sin burbuja toma el valor x, mientras que la salida con burbuja, $\bar{x}$.

Los valores en doble raíl de todas las entradas se disponen en las columnas 1, 2, 3, 4, 5, 6. Además, Cada una de estas columnas puede tener interconexión con las filas 7, 8, 9, 10, 11, 12, 13, 14, 15. Cada línea horizontal (que representa una fila) es una simplificación del conjunto de cables o líneas que entrarían en cada puerta AND de la PAL. Nótese, en el caso del ejemplo, que para que una puerta AND pueda generar un término producto cualquier, es necesario que dicha puerta tenga en sus entradas todas las variables disponibles, con o sin complementar, en este caso, 6, $I_2, I_1, I_0, \bar{I}_2, \bar{I}_1, \bar{I}_0$. La figura 4.63 muestra en detalle las interconexiones de entrada de la puerta AND de la fila 7.

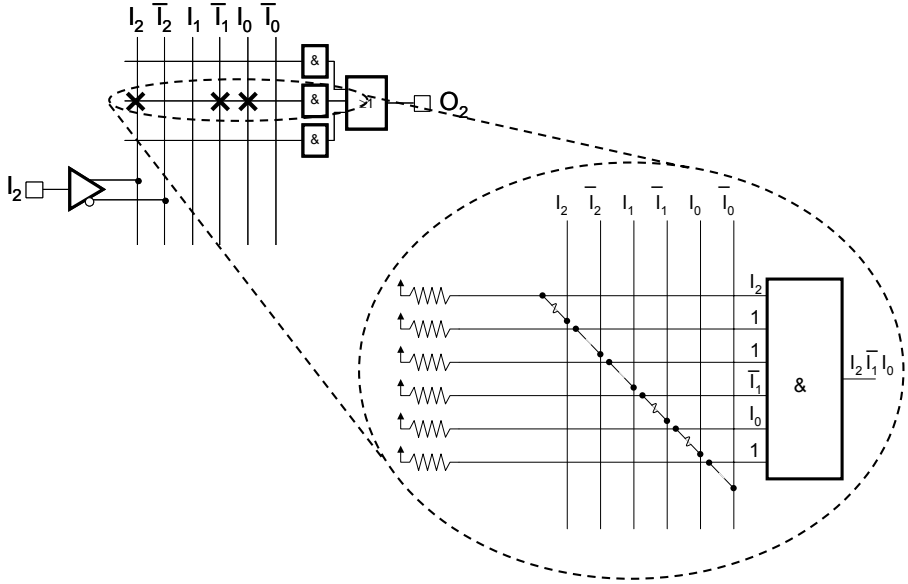


Figura 4.63.

Al igual que en la ROM, cada entrada de la AND dispone de una resistencia en *pull-up* ("tirada arriba") con el objeto de que si un fusible es fundido, y por tanto eliminada la conexión entra la fila y columna correspondiente, dicha entrada tenga un 1 lógico.

Seguimos, pues, adoptando el criterio simplificador mostrado en la ROM y por tanto, la existencia de interconexión será representada por una aspa $\times$. La figura 4.64 muestra un ejemplo en el que se está generando el producto $x \cdot y$. Como puede apreciarse, sobre las columnas se han especificado cuales son las variables de entrada asociadas a las mismas

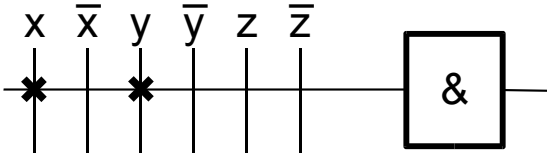


Figura 4.64.

La PAL estudiada, aunque simple, muestra de forma fácil la estructura de las mismas. Las PALs comerciales pueden tener más pines de entrada para usar como variables, más salidas para conseguir programar un número mayor de funciones de conmutación y también un mayor número de puertas AND asociadas a cada OR, con el objeto de poder implementar más términos producto en cada función. Además, podemos encontrar PALs comerciales con estructuras de salidas más complejas. Analizaremos el caso de la PAL de tres entradas y tres salidas mostrada en la figura 4.65.

La estructura de salida para cada O_i dispone de un inversor triestado inversor que está controlado por la salida de una nueva puerta AND cuyas entradas pueden ser programadas. Este inversor triestado cuando está habilitado genera a su salida el complemento del valor lógico de entrada, y si no está habilitado, su salida se encuentra en alta impedancia. Además, en la estructura se observa una realimentación de la salida del inversor triestado hacia el interior del plano programable. Esta realimentación, combinada con la presencia del inversor triestado, presenta dos ventajas claras. En primer lugar, si el inversor triestado está inhabilitado, este se encuentra en alta impedancia, por lo que la salida O_i puede ser utilizada como entrada. Esto permite que la estructura PAL pueda ser utilizada para programar 3 funciones de 3 variables máximo o 2 funciones de 4 variables (usando una salida como entrada) o 1 función de 5 variables (usando dos salidas como entrada). La segunda ventaja

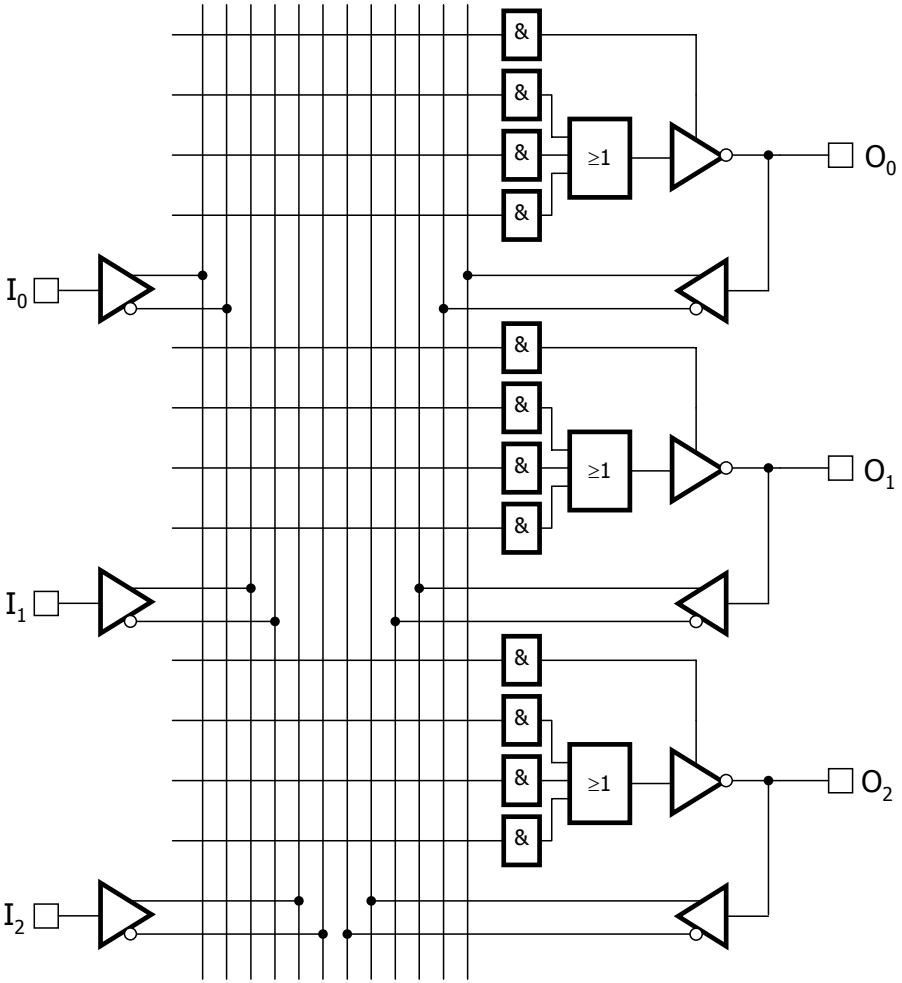


Figura 4.65.

se corresponde con la posibilidad de utilizar esta realimentación para formar funciones de conmutación que representen sumas de más de 3 términos producto. En efecto, si el inversor triestado está habilitado, su salida representa una función de conmutación que será suma de algunos términos producto. Esta suma de productos, gracias a la realimentación, se dispone en el array programable, por lo que puede ser usada por otras puertas AND, asociadas a otras salidas, para generar expresiones de conmutación con un mayor número de términos producto.

Ejemplo. Usando la primera estructura PAL, implementar las funciones de conmutación siguientes $f = \sum(0, 2, 4, 7)$, $g = \sum(0, 1, 2, 6, 7)$, $h = \sum(6, 7)$.

Sabemos que esa estructura PAL permite implementar un máximo de tres funciones de conmutación de tres variables y con tres términos producto. Encontrar la solución consiste en buscar que las funciones anteriores puedan ser descritas como una suma con un máximo de tres términos producto y en programar las conexiones internas, sin que sea necesario minimizar la función.

En primer lugar, la función h tiene dos mintérminos, por lo que no hay que hacer nada más:

$$h(x, y, z) = xy\bar{z} + xyz$$

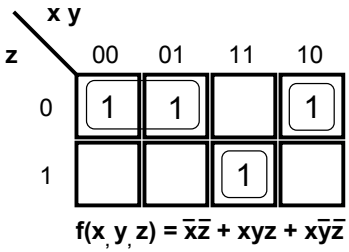
La función f y la función g tienen 4 y 5 mintérminos o términos producto, por lo que se obtendrán sus K-mapas para poder simplificarlas, figura 4.66.

Las expresiones para f y g son:

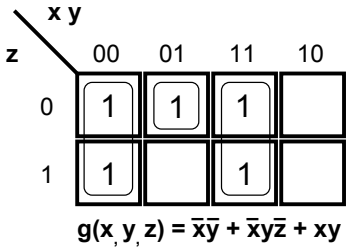
$$\begin{aligned} f(x, y, z) &= \bar{x}\bar{z} + xyz + x\bar{y}\bar{z} \\ g(x, y, z) &= \bar{x}\bar{y} + \bar{x}y\bar{z} + xy \end{aligned}$$

Las tres expresiones anteriores se programarían en la PAL de la forma que muestra la figura 4.67.

Sólo destacar el hecho de que para generar la función h, que sólo tiene dos términos productos, y por tanto sólo se usan dos puertas AND, hay que inutilizar la tercera puerta AND (fila 15). Para que esta puerta AND genere un 0 lógico a su salida, debemos realizar el producto lógico $x \cdot \bar{x} = 0$. En caso contrario, si no hubiera aparecido ninguna aspa en la fila 15, la AND generaría



$f(x, y, z) = \bar{x}\bar{z} + xyz + x\bar{y}\bar{z}$



$g(x, y, z) = \bar{x}\bar{y} + \bar{x}y\bar{z} + xy$

$h(x, y, z) = xy\bar{z} + xyz$

Figura 4.66.

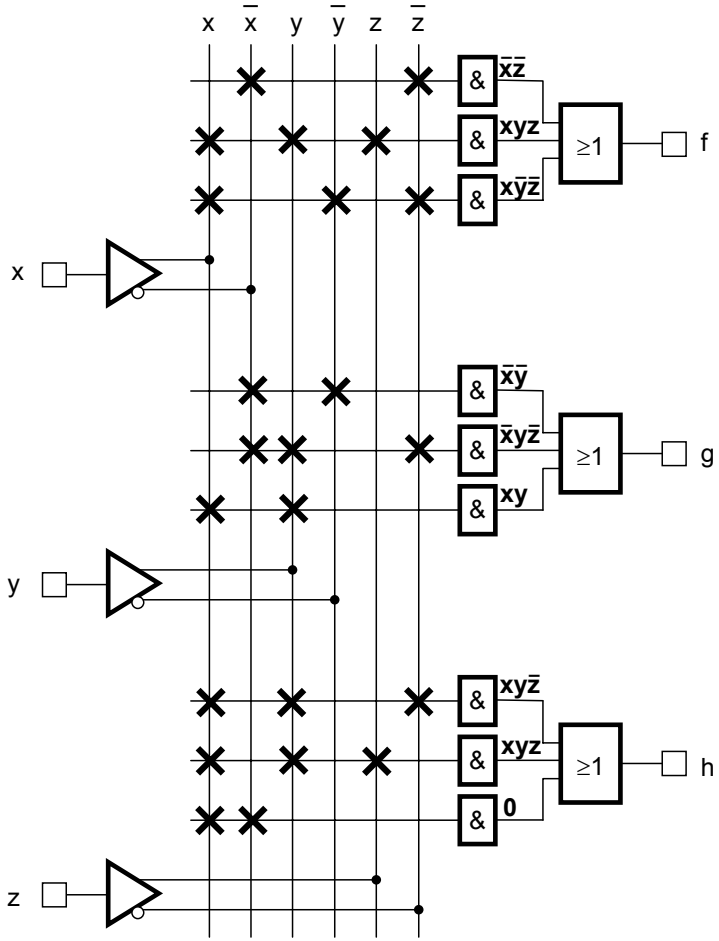


Figura 4.67.

un 1 como salida y la función h sería $h = 1 + xyz + xy\bar{z} = 1$, lo cual sería incorrecto.

4.3.3.2. PLA (Programmable Logic Array)

Este dispositivo tiene tanto el plano AND como el OR totalmente programables. La figura 4.68 muestra la estructura de una PLA de 3 entradas y 3 salidas.

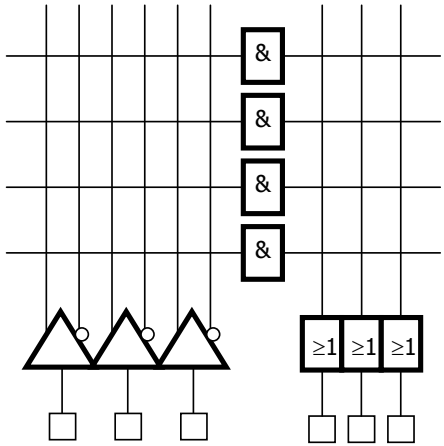


Figura 4.68.

La PLA de la figura 4.68 permite programar un máximo de tres funciones de conmutación distintas de tres variables cada una. Sólo pueden utilizarse para estas tres funciones un máximo de 4 puertas AND en conjunto, esto es, las tres funciones deben de expresarse con un máximo de cuatro términos producto.

Una de las ventajas principales que tiene este dispositivo frente a la PAL, reside en la posibilidad de compartir términos productos. Un mismo término producto puede usarse para generar cualquiera de las tres funciones de salida, mientras que en una PAL si n funciones de salida tienen el mismo término producto, este debe ser generado n veces distintas, una por cada función.

Ejemplo. . Implementar las siguientes funciones en la PLA anterior:

$$\begin{aligned} F &= \bar{x} + y \\ G &= \bar{x}y + x\bar{z} \\ H &= x\bar{z} + \bar{x} + y \end{aligned}$$

La solución se muestra en la figura 4.69

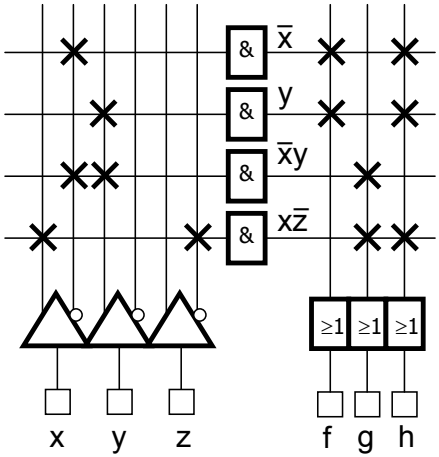


Figura 4.69.

Ejercicios propuestos

Problema 4.1 Utiliza la PAL de la figura 4.70 para implementar, siempre que sea posible: (a) un codificador de 2:4; (b) un decodificador 2:4 con entrada de Enable activo en nivel bajo; (c) un multiplexor de 4:1 y (d) un comparador de 2 bits sin entradas en cascada.

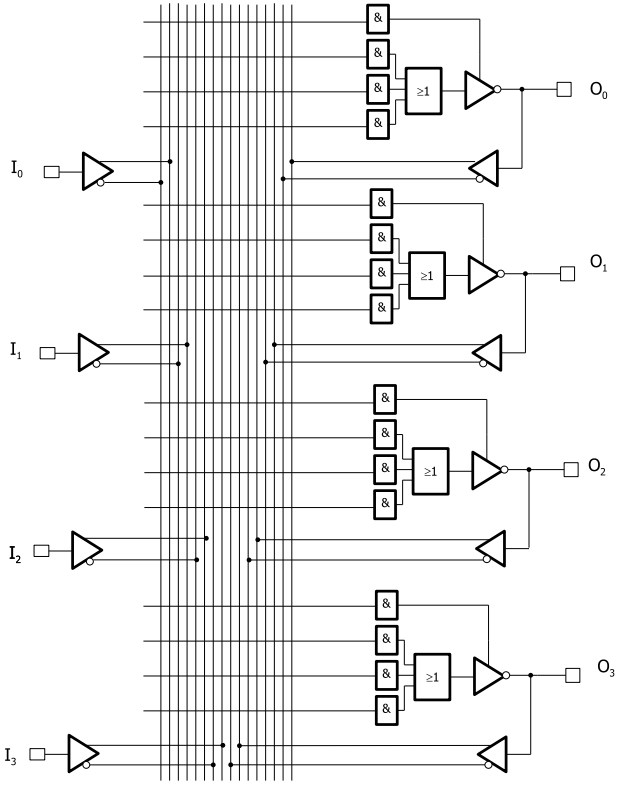


Figura 4.70.

Problema 4.2 a) Diseñe, a nivel de puertas lógicas, un comparador de magnitud de 1 bit, que disponga de dos entradas A_i y B_i y tres salidas G ($A_i > B_i$), E($A_i = B_i$) y L ($A_i < B_i$).

b Utilizando cuatro comparadores de un bit, del tipo diseñado en el apartado anterior y puertas lógicas, diseñe un comparador de magnitud de 4 bits con las mismas características.

Problema 4.3 Implemente la función $f = \sum(0, 5, 7, 10, 13, 15)$ haciendo uso de:

a) Un DEC 3:8 con salidas activas en alto y un máximo de 6 puertas NAND.

b) Un DEC 3:8 con salidas activas en bajo, una puerta AND con el menor número de entradas posibles y un máximo de dos puertas OR de dos entradas.

c) MUX's de 4 canales (máximo 5).

d) Una PLA lo más simple posible.

Problema 4.4

(a) Analice el circuito de la figura 4.71.

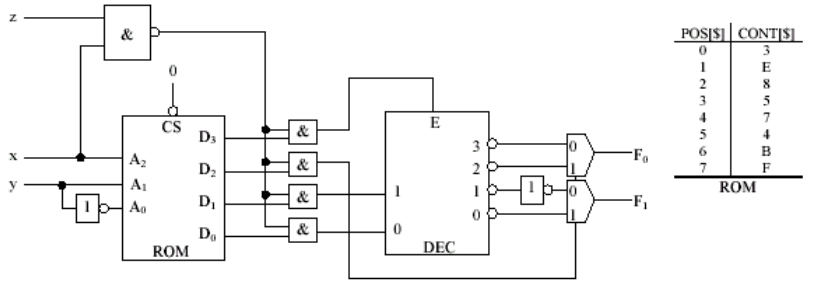


Figura 4.71.

(b)Rediséñelo usando una PLA.

Problema 4.5 Se desea usar un comparador de magnitudes de 4 bits, pero no disponemos de él. En su lugar se tiene dos memorias ROM de 16X3 y de un número inferior a 10 puertas lógicas. Diseñe el comparador con estos componentes.

Problema 4.6 Diseñar con puertas un circuito cuya salida sea el resto de la división entera de un número A de tres bits entre un número B de dos bits. El número B nunca puede ser cero.

Problema 4.7 Implemente la función $f = \sum(1, 2, 4, 6, 9, 12, 15, 22, 25, 27) + d(0, 31)$ utilizando un MUX's 1:4.

Problema 4.8 Implemente las funciones $f = \sum(1 ,3, 5, 6, 7, 9, 12, 15)$ y $g = \Pi(9, 4, 6, 8, 12, 13, 14)$ utilizando una ROM con 8 posiciones de memoria y MUX's de 2:1.

Capítulo 5

Aritmética binaria y circuitos aritméticos

5.1. Aritmética binaria

5.1.1. Aritmética binaria básica

Las operaciones de suma, resta, multiplicación y división para números binarios no difieren de las operaciones para aritmética decimal. Sólo hay que considerar que la base de numeración es la 2, y por tanto se generan acarreos cuando se alcance este valor.

Suma binaria. Sean A y B dos números de n bits. El resultado de la suma $S = A + B$ se puede obtener de la siguiente forma[1]:

[1] La función $\mathrm{LSB}(x)$ devuelve el bit menos significativo de x; la función $\mathrm{MSB}(x)$ devuelve el bit más significativo de x.

1. Sea C_i el acarreo (*carry*) generado al hacer la suma $A_i + B_i$
2. Sea $i = 0$ y $C_i = 0$
3. $S_i = \text{LSB}(A_i + B_i + C_i)$
4. $C_{i+1} = \text{MSB}(A_i + B_i + C_i)$
5. Incrementa i
6. Repite desde 3 mientras que $i < n$

Ejemplos:

$$
\begin{array}{rccccc}
C_{out} = & 1\,\curvearrowleft & 1\,\curvearrowleft & 0\,\curvearrowleft & 0\,\curvearrowleft & & \leftarrow C_i \\
 & & 1 & 1 & 0 & 1 & \leftarrow A_i \\
 & & 1 & 1 & 0 & 0 \;+ & \leftarrow B_i \\
 & 1 & 1 & 0 & 0 & 1 & \leftarrow S_i
\end{array}
$$

$$
\begin{array}{rccccc}
C_{out} = & 1\,\curvearrowleft & 1\,\curvearrowleft & 1\,\curvearrowleft & 1\,\curvearrowleft & & \leftarrow C_i \\
 & & 1 & 1 & 1 & 1 & \leftarrow A_i \\
 & & 0 & 0 & 0 & 1 \;+ & \leftarrow B_i \\
 & 1 & 0 & 0 & 0 & 0 & \leftarrow S_i
\end{array}
$$

Resta binaria. Sean A y B dos números de n bits. El resultado de la resta $S = A - B$ se puede obtener de la siguiente forma:

1. Sea C_i el acarreo (*borrow*) generado al hacer la resta $A_i - B_i$
2. Sea $i = 0$ y $C_i = 0$
3. Si $A - i \geq B_i + C_i$, entonces $R_i = A_i - (B_i + C_i)$ y $C_{i+1} = 0$; en caso contrario $R_i = \text{LSB}((B_i + C_i) - A_i)$ y $C_{i+1} = 1$.
4. Incrementa i
5. Repite desde 3 mientras que $i < n$

Ejemplos:

$$
\begin{array}{rccccc}
 & & 1 & 0 & 0 & 1 & \leftarrow A_i \\
C_{out} = & 0\,\curvearrowleft & 1\,\curvearrowleft & 1\,\curvearrowleft & 0\,\curvearrowleft & & \leftarrow C_i \\
 & & 0 & 0 & 1 & 1 \;- & \leftarrow B_i \\
 & & 0 & 1 & 1 & 0 & \leftarrow R_i
\end{array}
$$

$$
\begin{array}{rcccccl}
 & & 1 & 0 & 0 & 1 & \leftarrow A_i \\
C_{out} = & 1\,\text{↰} & 1\,\text{↰} & 1\,\text{↰} & 0\,\text{↰} & & \leftarrow C_i \\
 & 1 & 1 & 1 & 0 & - & \leftarrow B_i \\
\hline
 & 1 & 0 & 1 & 1 & & \leftarrow R_i
\end{array}
$$

Multiplicación binaria. La multiplicación binaria sigue las mismas reglas que la correspondiente a la base diez, salvo que la suma final se realiza en binario. **Ejemplos**:

$$
\begin{array}{cccccccc}
 & & 1 & 0 & 0 & 1 & & \leftarrow A \\
 & & & 1 & 0 & 1 & \times & \leftarrow B \\
\cline{3-7}
 & & 1 & 0 & 0 & 1 & & \\
 & 1 & 0 & 0 & 1 & & & \\
\cline{2-7}
 & 1 & 0 & 1 & 1 & 0 & 1 & \leftarrow P
\end{array}
$$

$$
\begin{array}{ccccccccc}
 & & & 1 & 1 & 0 & 1 & & \leftarrow A \\
 & & & 1 & 0 & 1 & 1 & \times & \leftarrow B \\
\cline{4-8}
 & & & 1 & 1 & 0 & 1 & & \\
 & & 1 & 1 & 0 & 1 & & & \\
 & 1 & 1 & 0 & 1 & & & & \\
\cline{2-8}
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \leftarrow P
\end{array}
$$

División binaria. Ejemplos:

$$
\begin{array}{cccccc|ccc}
1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & - & & & \multicolumn{3}{c}{1\ 0\ 1\ 1} \\
\cline{1-3}
 & & 1 & 1 & 1 & & & & \\
 & & 1 & 0 & 0 & - & & & \\
\cline{3-6}
 & & & 1 & 1 & 1 & & & \\
 & & & 1 & 0 & 0 & - & & \\
\cline{4-7}
 & & & 0 & 1 & 1 & & &
\end{array}
$$

$$
\begin{array}{cccccc|ccc}
1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
 & 1 & 1 & 0 & - & & \multicolumn{3}{c}{1\ 1\ 0} \\
\cline{2-4}
 & 0 & 1 & 1 & 0 & & & & \\
 & & 1 & 1 & 0 & - & & & \\
\cline{3-6}
 & & & & 0 & 1 & & &
\end{array}
$$

Las operaciones aritméticas básicas (suma y resta) en cualquier tipo de notación numérica deben generar un resultado adecuado al tipo de notación. Para estas operaciones binarias básicas dispondremos de circuitos que permiten realizar sumas y restas (o sea, sumadores y restadores). Se demostrará en los siguientes apartados que las notaciones **complemento a 1** (Ca1) y **complemento a 2** (Ca2) son las que pueden presentar un menor coste electrónico debido que estas permiten realizar las operaciones de resta y suma sólo con circuitos sumadores.

5.1.2. Aritmética en notación signo-magnitud

Para poder obtener el resultado de la suma de dos números expresados en notación signo-magnitud (S-M) se deben seguir las siguientes reglas:

(a) Si ambos números tienen el mismo signo, la magnitud del resultado se corresponde con la suma de las magnitudes de los números. Además, el bit de signo del resultado es el mismo que el de cualquiera de los sumandos.

(b) Si los números tienen distinto signo, la magnitud del resultado se determina calculando la diferencia entre la magnitud mayor y menor de los dos números. El bit de signo se corresponde con el del número que tenga mayor magnitud.

La figura 5.1 ilustra el procedimiento de suma en signo-magnitud mediante un ejemplo con números de distinto signo.

Desde el punto de vista del diseño lógico, un circuito que permita operaciones aritméticas básicas con números en notación S-M debe tener:

- un sumador binario
- un restador binario
- un comparador de magnitud
- lógica de comparación de los bits de signo

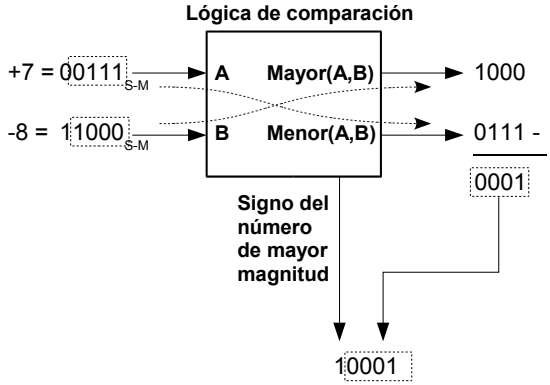


Figura 5.1.

5.1.3. Aritmética en complemento a 2

La representación de números binarios con signo en notación complemento a 2 (Ca2) eliminan la necesidad de utilizar circuitos restadores para la realización de las operaciones aritméticas básicas. En complemento a 2, un número negativo se obtiene aplicando el operador Ca2 al módulo de dicho número (más un cero en la posición más significativa). Este operador da como resultado otro número que se obtiene aplicando la siguiente fórmula:

$$\mathrm{Ca2}(M) = 2^n - M$$

La forma de representación de un número negativo en Ca2, lleva implícita la operación de resta. Si se desea, por ejemplo, realizar la resta binaria entre dos números A y B, basta con expresar B en Ca2 (esto genera una magnitud igual a $2^n - B$) y sumárselo al número A (el resultado sería $2^n + A - B$). Como se puede apreciar, el resultado se compone de la cantidad 2^n y del valor correcto de la diferencia de A y B, por lo que muchos de los circuitos que permiten realizar las operaciones aritméticas básicas se basan, exclusivamente, en un sumador binario.

En la aritmética en Ca2 deben considerarse algunas situaciones para conseguir el resultado correcto. En todas las situaciones siguientes se realizarán operaciones de sumas con dos números. La operación aritmética de resta se consigue sumando dos números de distinto signo. Se analizarán los siguientes casos:

(a) **Se dispone de dos números binarios A y B positivos**. La suma binaria de dos números A y B positivos expresados en Ca2 genera el resultado correcto también expresado en Ca2. **Ejemplo**:

$$\begin{array}{rccccc} \boxed{C_{out}=0} & 0 & 1 & 0 & 0 & & \leftarrow acarreos \\ & 0 & 0 & 1 & 0 & 1 & \leftarrow A=+5 \\ & 0 & 0 & 1 & 1 & 0 & + \leftarrow B=+6 \\ \hline & 0 & 1 & 0 & 1 & 1 & \leftarrow S=+11 \checkmark \end{array}$$

Puede darse la situación paradójica en que al sumar dos números positivos, el resultado sea un número negativo (bit más significativo igual a 1). En este caso se dice que se ha generado un **desbordamiento** u ***overflow***. El desbordamiento se produce cuando la magnitud del resultado no puede ser expresada con el número de bits que tienen los operandos. **Ejemplo**:

$$\begin{array}{rccccc} \boxed{C_{out}=0} & 1 & 1 & 0 & 0 & & \leftarrow acarreos \\ & 0 & 1 & 1 & 0 & 0 & \leftarrow A=+12 \\ & 0 & 1 & 1 & 0 & 1 & + \leftarrow B=+13 \\ \hline & 1 & 1 & 0 & 0 & 1 & \leftarrow S=-7 \text{ ✘} \end{array}$$

El desbordamiento se soluciona de forma sencilla añadiendo más bits a la representación de los números.

$$\begin{array}{rcccccc} \boxed{C_{out}=0} & 0 & 1 & 1 & 0 & 0 & & \leftarrow acarreos \\ & 0 & 0 & 1 & 1 & 0 & 0 & \leftarrow A=+12 \\ & 0 & 0 & 1 & 1 & 0 & 1 & + \leftarrow B=+13 \\ \hline & 0 & 1 & 1 & 0 & 0 & 1 & \leftarrow S=+25 \checkmark \end{array}$$

(b) **Se dispone de dos números binarios A y B negativos**. La suma de dos números A y B negativos expresados en Ca2 genera el resultado correcto, también expresado en Ca2, más un acarreo final. **Ejemplo**:

$$\begin{array}{rccccc} \boxed{C_{out}=\not{1}} & 1 & 0 & 1 & 0 & & \leftarrow acarreos \\ & 1 & 1 & 0 & 1 & 1 & \leftarrow A=-5 \\ & 1 & 1 & 0 & 1 & 0 & + \leftarrow B=-6 \\ \hline & 1 & 0 & 1 & 0 & 1 & \leftarrow S=-11 \checkmark \end{array}$$

El acarreo final debe ser despreciado en Ca2 para obtener el resultado correcto.

Al igual que en el apartado anterior, se puede generar un desbordamiento si al sumar dos números negativos, el resultado es positivo. **Ejemplo**:

$\boxed{C_{out} = \not{1}}$	0	0	0	0			$\leftarrow acarreos$
	1	0	1	0	0		$\leftarrow A = -12$
	1	0	0	1	1	+	$\leftarrow B = -13$
	0	0	1	1	1		$\leftarrow S = +7$ ✘

La solución es idéntica a la expuesta anteriormente, se añaden más bits a la representación.

$\boxed{C_{out} = \not{1}}$	1	0	0	0	0		$\leftarrow acarreos$
	1	1	0	1	0	0	$\leftarrow A = -12$
	1	1	0	0	1	1 +	$\leftarrow B = -13$
	1	0	0	1	1	1	$\leftarrow S = -25$ ✔

Justificación. Si A y B son números negativos, entonces en Ca2 expresan las cantidades:

$$2^n - |A|$$
$$2^n - |B|$$

donde n es el número de bits de A y B; $|A|$ y $|B|$ son sus magnitudes (valores absolutos), respectivamente. La suma en Ca2 de estas cantidades debe generar como resultado la cantidad:

$$2^n - (|A| + |B|)$$

pero el resultado la suma binaria es:

$$(2^n - |A|) + (2^n - |B|) = [2^n - (|A| + |B|)] + 2^n$$

De la suma binaria sobra el término 2^n para obtener el resultado correcto en Ca2. Obsérvese que si los números A y B tienen n bits, el término 2^n tiene $n + 1$ bits, siendo el bit más significativo un 1, y los restantes n bits, $00\ldots0$. Teniendo además en cuenta que el sumando $2^n - (|A| + |B|)$ es menor que 2^n (ya que tanto $|A|$ como $|B|$ son negativos) y por tanto, cabe en n bits, es claro que el sumando 2^n es responsable de la generación de un acarreo de salida C_{out} en la suma binaria.

(c) **Se dispone de dos números binarios A y B de distinto signo**. Supongamos que A es negativo y B es positivo. Es evidente que la suma de dos números de distinto signo puede generar un resultado positivo o negativo en función de las magnitudes de los mismos. En primer lugar se asumirá que la magnitud de A es menor o igual que la magnitud de B.

(c.1) $|A| \leq |B|$. **Ejemplo**:

$$
\begin{array}{lccccccl}
\boxed{C_{out} = \not{1}} & 1 & 1 & 0 & 0 & & & \leftarrow acarreos \\
 & 1 & 0 & 1 & 0 & 0 & & \leftarrow A = -12 \\
 & 0 & 1 & 1 & 0 & 1 & + & \leftarrow B = +13 \\
\hline
 & 0 & 0 & 0 & 0 & 1 & & \leftarrow S = +1 \checkmark
\end{array}
$$

Se observa que el resultado de la suma binaria incluye un acarreo de salida que se debe eliminar para obtener el resultado correcto en Ca2.

(c.2) $|A| > |B|$. **Ejemplo**:

$$
\begin{array}{lccccccl}
\boxed{C_{out} = 0} & 0 & 0 & 0 & 0 & & & \leftarrow acarreos \\
 & 1 & 0 & 0 & 1 & 1 & & \leftarrow A = -13 \\
 & 0 & 1 & 1 & 0 & 0 & + & \leftarrow B = +12 \\
\hline
 & 1 & 1 & 1 & 1 & 1 & & \leftarrow S = -1 \checkmark
\end{array}
$$

En este caso, la suma binaria obtiene directamente el resultado correcto, sin que se haya generado un acarreo.

Justificación. Si A es un número negativo, entonces en Ca2 expresa la cantidad:

$$2^n - |A|$$

Al ser B positivo, es trivial que $B = |B|$.
Si $|A| \leq |B|$, entonces el resultado debe ser un número positivo que en Ca2 se expresaría como:

$$|B| - |A|$$

Pero la suma de las cantidades $2^n - |A|$ con $|B|$ genera el resultado:

$$2^n + (|B| - |A|)$$

es decir, obtenemos dos sumandos: el término $|B| - |A|$, que es el resultado esperado en Ca2, y el término 2^n, que no es más que el acarreo de salida, y que debe despreciarse.
En cambio si $|A| > |B|$ el resultado correcto en Ca2 es un número negativo que debe expresarse como:

$$2^n - (|A| - |B|)$$

Ésta es la cantidad que se obtiene mediante la suma binaria de ambos números, en la que no se genera acarreo puesto que $|A| - |B|$ es una cantidad positiva que se resta a 2^n, de forma que el resultado cabe en n bits.

La tabla 5.1 resume todas las posibilidades desarrolladas anteriormente.

Tabla 5.1: Suma en complemento a 2

Signos	Valores	A+B en Ca2 (resultado esperado)	A+B en binario (resultado obtenido)	Corrección a realizar
A ≥ 0 B ≥ 0	A = |A| B = |B|	|A| + |B|	|A| + |B|	Ninguna
A < 0 B ≥ 0	A = 2^n - |A| B = |B|	*si* |A| > |B|: 2^n – (|A| - |B|) *si* |A| ≤ |B|: |B| - |A|	2^n - |A| + |B| = { 2^n – (|A| - |B|) ; (|B| - |A|) + 2^n }	Ninguna **Despreciar acarreo**
A ≥ 0 B < 0	A = |A| B = 2^n - |B|	*si* |A| ≥ |B|: |A| - |B| *si* |A| < |B|: 2^n – (|B| - |A|)	2^n + |A| - |B| = { (|A| - |B|) + 2^n ; 2^n – (|B| - |A|) }	**Despreciar acarreo** Ninguna
A < 0 B < 0	A = 2^n - |A| B = 2^n - |B|	2^n – (|A| + |B|)	[2^n – (|A| + |B|)] + 2^n	**Despreciar acarreo**

Conclusiones:

- La aritmética en Ca2 permite la realización de sumas y restas utilizando, exclusivamente, un sumador binario.
- El resultado correcto de la operación aritmética se obtiene despreciando el acarreo de salida si éste se genera.

5.1.3.1. Resta en complemento a 2

Muchos fabricantes de microprocesadores que usan la notación Ca2 para la representación de números con signo en sus máquinas incluyen circuitos sumadores y restadores en las unidades de cálculo de los mismos, para la realización de las operaciones aritméticas. Esto implica que para restar dos números A y B no es necesario calcular previamente el Ca2 de uno de ellos y añadírselo al otro, sino que directamente se usa el restador. Se pueden distinguir los siguientes casos:

(a) A **y** B **tienen el mismo signo**. Hay dos posibilidades:

(a.1) Si $A > B$, entonces $R = A - B$ es positivo y no se genera acarreo (*borrow*) de salida $C_{out} = 0$. **Ejemplos**:

	1	1	0	1		$\leftarrow A = -3$
$\boxed{C_{out} = 0}$	0	0	0			$\leftarrow acarreos$
	1	1	0	0	$-$	$\leftarrow B = -4$
	0	0	0	1		$\leftarrow R = +1$ ✔

	0	1	0	0		$\leftarrow A = +4$
$\boxed{C_{out} = 0}$	0	1	1			$\leftarrow acarreos$
	0	0	1	1	$-$	$\leftarrow B = +3$
	0	0	0	1		$\leftarrow R = +1$ ✔

(a.2) Si $A < B$, entonces el resultado de la resta es negativo y por tanto debe estar representado como el complemento a 2 de la diferencia $B - A$, generándose un acarreo de salida, $C_{out} = 1$. **Ejemplos**:

$$\begin{array}{rcccccl} & 1 & 1 & 0 & 0 & & \leftarrow A = -4 \\ \boxed{C_{out} = \cancel{1}} & 1 & 1 & 1 & & & \leftarrow acarreos \\ & 1 & 1 & 0 & 1 & - & \leftarrow B = -3 \\ \hline & 1 & 1 & 1 & 1 & & \leftarrow R = -1 \checkmark \end{array}$$

$$\begin{array}{rcccccl} & 0 & 0 & 1 & 1 & & \leftarrow A = +3 \\ \boxed{C_{out} = \cancel{1}} & 1 & 0 & 0 & & & \leftarrow acarreos \\ & 0 & 1 & 0 & 0 & - & \leftarrow B = +4 \\ \hline & 1 & 1 & 1 & 1 & & \leftarrow R = -1 \checkmark \end{array}$$

Nótese que al restar dos números con el mismo signo, es imposible que se genere desbordamiento.

(b) A **es positivo y** B **es negativo**. El resultado obtenido debe ser positivo y se genera acarreo de salida $C_{out} = 1$. **Ejemplo**:

$$\begin{array}{rcccccl} & 0 & 0 & 1 & 1 & & \leftarrow A = +3 \\ \boxed{C_{out} = \cancel{1}} & 1 & 0 & 0 & & & \leftarrow acarreos \\ & 1 & 1 & 0 & 0 & - & \leftarrow B = -4 \\ \hline & 0 & 1 & 1 & 1 & & \leftarrow R = +7 \checkmark \end{array}$$

Pueden aparecer errores por desbordamiento en este caso, puesto que el resultado final debe ser positivo e igual a la suma de las magnitudes de los números A y B y, ésta, podría no ser representable con el número de bits de dichos números. Como solución a este problema, se debe ampliar el número de bits de A y B. **Ejemplos**:

$$\begin{array}{rcccccl} & 0 & 1 & 0 & 1 & & \leftarrow A = +5 \\ \boxed{C_{out} = \cancel{1}} & 0 & 0 & 0 & & & \leftarrow acarreos \\ & 1 & 1 & 0 & 0 & - & \leftarrow B = -4 \\ \hline & 1 & 0 & 0 & 1 & & \leftarrow R = -7 \text{✘} \end{array}$$

$$\begin{array}{rccccccl} & 0 & 0 & 1 & 0 & 1 & & \leftarrow A = +5 \\ \boxed{C_{out} = \cancel{1}} & 1 & 0 & 0 & 0 & & & \leftarrow acarreos \\ & 1 & 1 & 1 & 0 & 0 & - & \leftarrow B = -4 \\ \hline & 0 & 1 & 0 & 0 & 1 & & \leftarrow R = +9 \checkmark \end{array}$$

(c) A **es un número negativo y** B **es positivo**. El resultado obtenido debe ser negativo y no se genera acarreo final $C_{out} = 0$. **Ejemplo**:

$$\begin{array}{rrrrrrl} & 1 & 1 & 0 & 0 & - & \leftarrow A=-4 \\ \boxed{C_{out}=0} & 0 & 1 & 1 & & & \leftarrow acarreos \\ & 0 & 0 & 1 & 1 & & \leftarrow B=+3 \\ \hline & 1 & 0 & 0 & 1 & & \leftarrow R=-7 \checkmark \end{array}$$

También es posible en este caso la aparición de desbordamientos. El resultado final debe ser un número negativo con magnitud igual a la suma de las magnitudes de los números A y B. Si ésta no puede ser representada con el número de bits iniciales de A y B, se debe ampliar el número de bits de la representación de los mismos. **Ejemplos**:

$$\begin{array}{rrrrrrl} & 1 & 1 & 0 & 0 & - & \leftarrow A=-4 \\ \boxed{C_{out}=0} & 1 & 1 & 1 & & & \leftarrow acarreos \\ & 0 & 1 & 0 & 1 & & \leftarrow B=+5 \\ \hline & 0 & 1 & 1 & 1 & & \leftarrow R=+7 \times \end{array}$$

$$\begin{array}{rrrrrrrl} & 1 & 1 & 1 & 0 & 0 & - & \leftarrow A=-4 \\ \boxed{C_{out}=0} & 0 & 1 & 1 & 1 & & & \leftarrow acarreos \\ & 0 & 0 & 1 & 0 & 1 & & \leftarrow B=+5 \\ \hline & 1 & 0 & 1 & 1 & 1 & & \leftarrow R=-9 \checkmark \end{array}$$

5.1.4. Aritmética en complemento a 1

Al igual que se hizo con la aritmética en Ca2, en Ca1 deben considerarse algunas situaciones especiales:

(a) **Se dispone de dos números A y B positivos**. Este caso es idéntico al del Ca2 puesto que los números positivos se expresan del mismo modo.

(b) **Se dispone de dos números A y B negativos**. **Ejemplo**:

$$\begin{array}{rrrrrrrl} \boxed{C_{out}=1} & 1 & 0 & 0 & 0 & & & \leftarrow acarreos \\ & 1 & 1 & 0 & 1 & 0 & & \leftarrow A=-5 \\ & 1 & 1 & 0 & 0 & 1 & + & \leftarrow B=-6 \\ \hline & 1 & 0 & 1 & 1 & 1 & & \leftarrow S=-12 \times \end{array}$$

En el ejemplo anterior se observa que la suma binaria de dos números negativos expresados en Ca1 genera un resultado más un acarreo final. El resultado generado no expresa el valor correcto de la suma en Ca1, ya que le falta una unidad.

$\boxed{C_{out}=1}$	1	0	0	0			$\leftarrow acarreos$
	1	1	0	1	0		$\leftarrow A=-5$
	1	1	0	0	1	+	$\leftarrow B=-6$
	1	0	0	1	1		
					1	+	
	1	0	1	0	0		$\leftarrow S=-11$ ✔

En Ca1, el acarreo generado se desprecia y además se debe añadir una unidad al resultado obtenido mediante la suma binaria para conseguir el valor correcto.

Al igual que en Ca2, también se puede producir un error de desbordamiento en Ca1 si el resultado de la operación excede de los límites de la representación.

Justificación. Si A y B son números negativos (sin parte fraccionaria), entonces en Ca1 expresan la cantidad:

$$2^n - |A| - 1$$
$$2^n - |B| - 1$$

donde n es el número de bits de A y B; $|A|$ y $|B|$ son sus magnitudes (valores absolutos), respectivamente.
El resultado esperado de la suma en Ca1 de A y B debe ser:

$$2^n - (|A| + |B|) - 1$$

Sin embargo, la suma binaria de estas cantidades genera el siguiente resultado:

$$(2^n - |A| - 1) + (2^n - |B| - 1) = [2^n - (|A| + |B|) - 1] + 2^n - 1$$

Para obtener el resultado esperado hay que cancelar el término $2^n - 1$, lo cual se consigue desechando el acarreo final C_{out} (que equivale al sumando 2^n) y añadiendo una unidad al resultado de la suma binaria.

(c) **Se dispone de dos números A y B de distinto signo**. Supongamos que A es negativo y B es positivo. La suma de dos números de distinto signo puede generar un resultado positivo o negativo en función de las magnitudes de los mismos, por tanto, distiguimos los siguientes casos:

(c.1) $|A| < |B|$. **Ejemplo**:

$\boxed{C_{out} = 1}$	1	1	1	1			$\leftarrow acarreos$
	1	0	0	1	1		$\leftarrow A = -12$
	0	1	1	0	1	+	$\leftarrow B = +13$
	0	0	0	0	0		
					1	+	
	0	0	0	0	1		$\leftarrow S = +1$ ✔

Se observa que para obtener el resultado esperado, es necesario desechar el acarreo de salida y añadir una unidad al resultado obtenido de la suma binaria de ambos números.

(c.2) $|A| \geq |B|$. **Ejemplo**:

$\boxed{C_{out} = 0}$	0	0	0	0			$\leftarrow acarreos$
	0	1	1	0	0		$\leftarrow A = +12$
	1	0	0	1	0	+	$\leftarrow B = -13$
	1	1	1	1	0		$\leftarrow S = -1$ ✔

Se observa que la suma binaria proporciona directamente el resultado correcto, sin necesidad de correcciones.

Justificación. Si A es un número negativo, entonces en Ca1 representa la cantidad:

$$2n - |A| - 1$$

Trivialmente, al ser B positivo, $B = |B|$.
Si $|A| < |B|$, entonces el resultado debe ser un número positivo que en Ca1 se expresaría como:

$$B - A$$

Pero la suma de las cantidades $2^n - |A| - 1$ y $|B|$ genera el resultado:

$$2^n + (|B| - |A|) - 1$$

del que hay que despreciar el término 2^n, que es el acarreo final, y sumar una unidad para obtener el resultado esperado.
En cambio si $|A| \geq |B|$ el resultado es un número negativo que debe expresarse como:

$$2^n - (|A| - |B|) - 1$$

De hecho, esta es la cantidad que se obtiene por la suma binaria de ambos números, pero ahora no se genera acarreo puesto que $|A| - |B|$ es una cantidad positiva que se resta a $2^n - 1$. No es necesaria ninguna corrección.

La tabla 5.2 resume todas las posibilidades desarrolladas anteriormente.

Tabla 5.2: Suma en complemento a 1

Signos	Valores	A+B en Ca2 (resultado esperado)	A+B en binario (resultado obtenido)	Corrección a realizar
A ≥ 0 B ≥ 0	A = |A| B = |B|	|A| + |B|	|A| + |B|	Ninguna
A < 0 B ≥ 0	A = 2^n - |A| - 1 B = |B|	*si* |A| ≥ |B|: 2^n – (|A| - |B|) – 1 *si* |A| < |B|: |B| - |A|	2^n - |A| + |B| - 1 = { 2^n – (|A| - |B|) – 1 ; (|B| - |A|) + **2^n – 1** }	Ninguna **Despreciar acarreo, sumar 1**
A ≥ 0 B < 0	A = |A| B = 2^n - |B| - 1	*si* |A| > |B|: |A| - |B| *si* |A| ≤ |B|: 2^n – (|B| - |A|) – 1	2^n + |A| - |B| - 1 = { (|A| - |B|) + **2^n – 1** ; 2^n – (|B| - |A|) – 1 }	**Despreciar acarreo,sumar 1** Ninguna
A < 0 B < 0	A = 2^n - |A| - 1 B = 2^n - |B| - 1	2^n – (|A| + |B|) – 1	2^n - (|A| + |B|) – 1 + **2^n – 1**	**Despreciar acarreo, sumar 1**

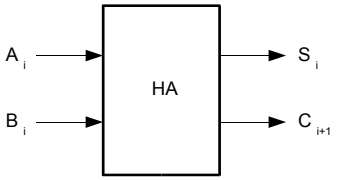
Conclusiones:

- La aritmética en Ca1 permite la realización de sumas y restas utilizando, exclusivamente, un sumador binario.
- Para obtener el resultado correcto de la operación aritmética, se debe despreciar el acarreo que se genere de la suma binaria de los dos números en Ca1, y añadir una unidad al resultado de dicha suma.

5.2. Circuitos aritméticos básicos

5.2.1. Circuitos semisumador y sumador completo

Un **semisumador** o **sumador medio** (HA, *Half Adder*), es un circuito combinacional con dos entradas A_i y B_i y dos salidas binarias S_i y C_{i+1}. La salida S_i representa el resultado de la suma aritmética de las entradas A_i y B_i y la salida C_{i+1}, el acarreo. El símbolo correspondiente se ha representado en la figura 5.2. La tabla de verdad adjunta resume el funcionamiento del semisumador para cualquier combinación de entradas.



Entradas	Salidas
A_iB_i	$C_{i+1}S_i$
0 0	0 0
0 1	0 1
1 0	0 1
1 1	1 0

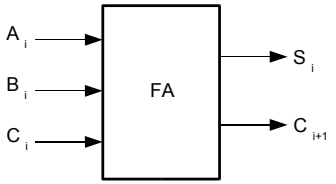
Figura 5.2.

De la tabla de verdad se pueden obtener, directamente, las expresiones lógicas de las salidas C_{i+1} y S_i, en función de las dos entradas A_i y B_i. Dichas expresiones se muestran a continuación:

$$\begin{aligned} S_i &= A_i \oplus B_i \\ C_{i+1} &= A_i B_i \end{aligned}$$

El **sumador completo** o **sumador total** (FA, *Full Adder*) es un circuito combinacional con tres entradas, A_i, B_i y C_i y dos salidas S_i y C_{i+1}. La salida S_i

representa la suma binaria de las tres entradas y, C_{i+1},el acarreo. El símbolo del sumador completo y su tabla de verdad se han representado en la figura 5.3.



Entradas	Salidas
$A_i B_i C_i$	$C_{i+1} S_i$
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

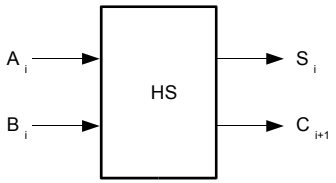
Figura 5.3.

Las funciones binarias de las salidas del sumador completo se obtienen simplificando a partir de la tabla de verdad y son las siguientes:

$$\begin{aligned} S_i &= A_i \oplus B_i \oplus C_i \\ C_{i+1} &= A_i B_i + A_i C_i + B_i C_i \end{aligned}$$

5.2.2. Circuitos semirrestador y restador completo

Un **semirestador** o **restador medio** (HS, *Half Subtracter*) es un circuito combinacional con dos entradas, A_i y B_i y dos salidas R_i y C_{i+1}. La salida R_i representa el resultado de la resta aritmética de las entradas A_i y B_i, es decir, $A_i - B_i$, y la salida C_{i+1}, el acarreo de la resta ("me llevo uno"). El símbolo correspondiente se ha representado en la figura 5.4. La tabla adjunta muestra los valores de salida para toda combinación de entrada.



Entradas	Salidas
$A B_i$	$C_{i+1} R_i$
0 0	0 0
0 1	1 1
1 0	0 1
1 1	0 0

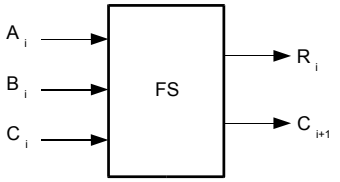
Figura 5.4.

De la tabla anterior se obtienen, directamente, las siguientes expresiones

para las salidas.

$$\begin{aligned} R_i &= A_i \oplus B_i \\ C_{i+1} &= \bar{A}_i B_i \end{aligned}$$

Un **restador completo** (FS, *Full Subtracter*) es un circuito combinacional con tres entradas A_i, B_i y C_i y dos salidas R_i y C_{i+1}. La salida R_i representa el resultado de la resta aritmética de las entradas A_i, B_i y C_i ($A_i - B_i - C_i$) y la salida C_{i+1}, el arrastre. El símbolo y la tabla de verdad del restador completo se muestran en la figura 5.5.



Entradas			Salidas	
A_i	B_i	C_i	C_{i+1}	S_i
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

Figura 5.5.

Simplificando las funciones de salida representadas en la tabla de verdad del restador completo, obtenemos las siguientes expresiones:

$$\begin{aligned} R_i &= A_i \oplus B_i \oplus C_i \\ C_{i+1} &= \bar{A}_i B_i + \bar{A}_i C_i + B_i C_i \end{aligned}$$

5.2.3. Sumadores y restadores de n bits

Los sumadores y restadores anteriores pueden combinarse para formar circuitos sumadores y restadores de números de n bits.

La suma de dos números binarios de n bits A ($A_{n-1} \ldots A_0$) y B ($B_{n-1} \ldots B_0$) genera un resultado, también de n bits, S ($S_{n-1} \ldots S_0$) y un acarreo C_n. El conjunto formado por el acarreo y los n bits de S ($C_n S_{n-1} \ldots S_0$) forman el resultado correcto de la suma. En la figura siguiente se muestra el procedimiento para la realización de la suma de dos números A y B de n bits cada uno.

C_n ↰	C_{n-1} ↰	...	C_{i+1} ↰	C_i ↰	...	C_2 ↰	C_1 ↰		
	A_{n-1}	...	...	A_i	...	A_2	A_1	A_0	
	B_{n-1}	...	...	B_i	...	B_2	B_1	B_0	+
C_n	S_{n-1}	...	...	S_i	...	S_2	S_1	S_0	

Para conseguir un circuito sumador de n bits, no hay más que entender el proceso de suma de dos números de n bits. Los bits A_0 y B_0 se suman para formar el resultado S_0, y si se genera acarreo, C_1, éste se suma en la siguiente etapa (los siguientes bits). Se repite el proceso para el bit 1: se suman A_1 y B_1 con el acarreo (si hay) de la etapa anterior (C_1), y esto genera la suma S_1 y un posible acarreo, C_2, etc. Este procedimiento se repite sucesivamente con todos los bits de los números A y B.

En general, para cada etapa i-ésima ($i = 0, \ldots, n-1$), se suman los bits A_i, B_i y C_i para dar el resultado S_i y se genera el acarreo C_{i+1} para la siguiente etapa.

Es evidente que el circuito que consigue esto es un sumador completo a excepción de la etapa $i = 0$, que no recibe ningún acarreo de entrada, y que puede implementarse con un semisumador. En la figura 5.6 se muestra la estructura de un sumador de n bits construido con n sumadores de 1 bit.

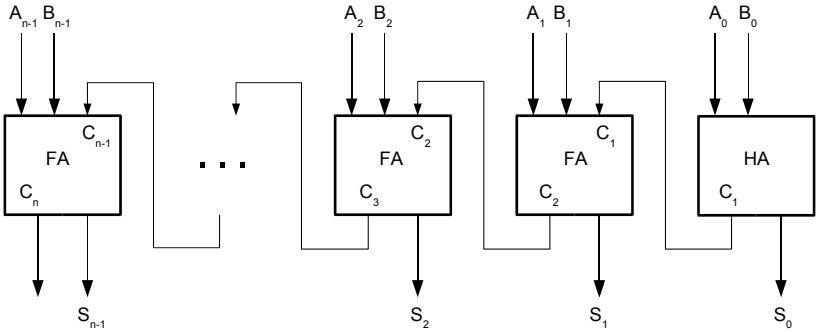


Figura 5.6.

Los sumadores de n bits pueden representarse como bloques funcionales que disponen de $2n$ entradas (números A y B), generan n salidas (resultado S) y un acarreo, C_n ó C_{out}, figura 5.7. Este tipo de sumador también se denomina **sumador paralelo**.

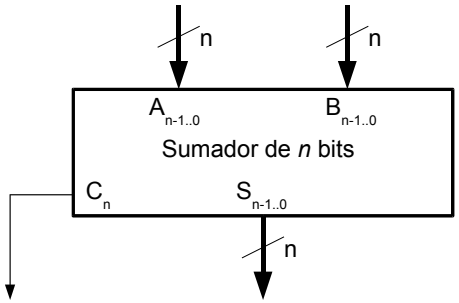


Figura 5.7.

En la siguiente figura se representa el diagrama de conexiones y el símbolo lógico del sumador binario de 4 bits 74LS283. Se aprecia que además del acarreo de salida CO (C4) existe un acarreo de entrada CI (C0).

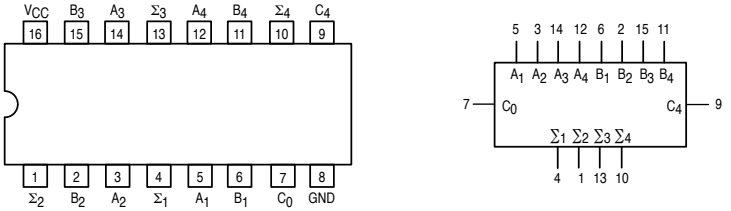


Figura 5.8.

Para poder añadir un posible acarreo inicial, C_0, la primera etapa del sumador de n bits, cuando $i = 0$, debe estar constituida por un sumador completo de 1 bit, en lugar del semisumador que se presentó con anterioridad. Las salidas Σ_{4-1} muestran la suma de los números A, B más el acarreo de entrada ($\Sigma = A + B + CI$). La incorporación de un acarreo de entrada a un sumador de n bits, permite **asociaciones de sumadores** con el fin de obtener circuitos sumadores que permitan operar con datos con un número mayor de bits. En la figura 5.9 se representa una asociación de sumadores de 4 bits para sumar dos números de 12 bits. Obsérvese que el sumador que calcula S_{3-0}, debe tener su entrada de acarreo igual a 0, no así los demás que deben contar con los acarreos de los sumadores anteriores para generar el resultado correcto.

Todo lo desarrollado anteriormente para los sumadores se puede aplicar en el caso de la operación de resta binaria, para obtener restadores de n bits. La figura 5.10 muestra un restador de dos números de n bits.

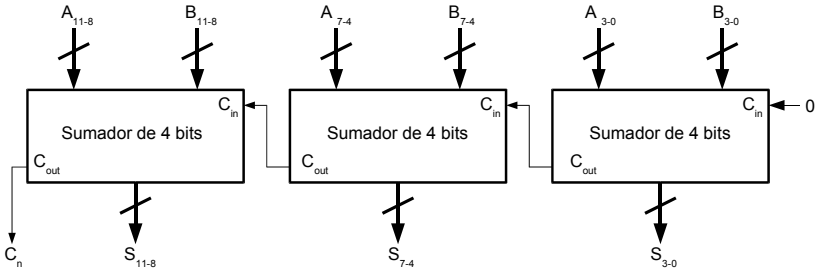


Figura 5.9.

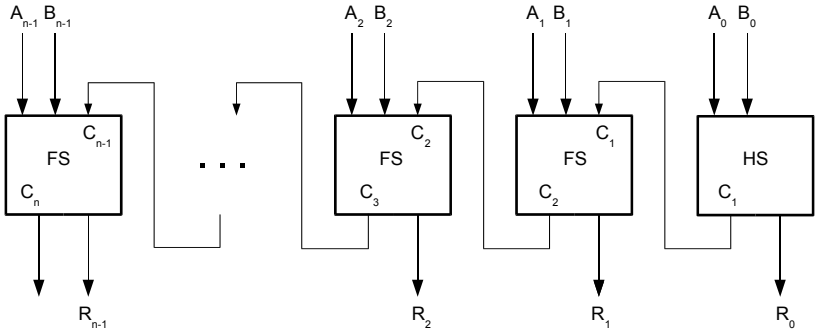


Figura 5.10.

5.2.4. Circuito sumador-restador

La operación aritmética de la resta puede implementarse con una suma si los números se introducen en complemento a 1 o en complemento a 2. También se sabe que el complemento a 1 de un número A de n bits es otro número B de n bits que se obtiene aplicando el operador NOT a cada uno de los bits del número A, esto es, $B_i = \bar{A}_i$ para $i = 0, \ldots, n-1$. Asimismo, el complemento a 2 de un número A de n bits se puede obtener de forma simple sin más que sumarle una unidad al complemento a 1 de dicho número, esto es, $\mathrm{Ca2}(A) = \mathrm{Ca1}(A) + 1$.

En este apartado, se diseñará un circuito sumador/restador, utilizando las propiedades de las notaciones Ca1 y Ca2. Este circuito debe disponer de una señal de control $\mathrm{RESTAR}/\overline{\mathrm{SUMAR}}$ que especifica qué tipo de operación debe realizarse (si $\mathrm{RESTAR}/\overline{\mathrm{SUMAR}} = 0$, suma, si $\mathrm{RESTAR}/\overline{\mathrm{SUMAR}} = 1$, resta). El núcleo del circuito sumador/restador de n bits, es un sumador de n bits que dispone, en sus entradas, los números A y B en caso de la operación de suma, o los números A y $\mathrm{Ca1}(B)$ (o $\mathrm{Ca2}(B)$) en caso operación de resta, tal como se muestra en la figura 5.11.

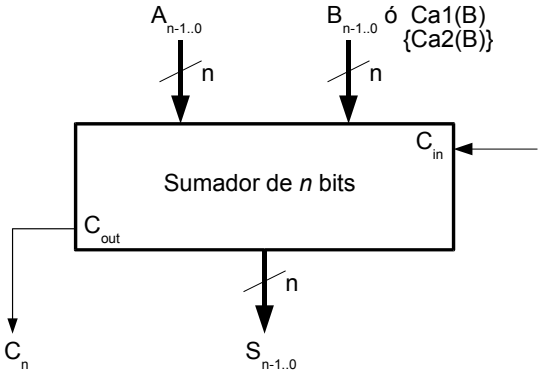


Figura 5.11.

Por tanto, la señal de control $\mathrm{RESTAR}/\overline{\mathrm{SUMAR}}$ que rige el funcionamiento del circuito tan sólo debe generar el Ca1 (O Ca2) del número B, o dejar pasar dicho número tal cual a las entradas del sumador. Se necesita, pues, diseñar un circuito capaz de realizar la operación complemento. Comenzaremos por el caso más simple, el Ca1.

En la figura 5.12 se ha representado una puerta EXOR de dos entradas denominadas A y CONTROL, y una salida S.

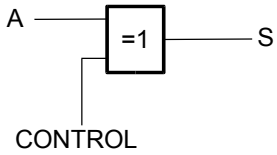


Figura 5.12.

Si la entrada CONTROL se encuentra a 0 lógico, la salida S tomará el valor 0 si la entrada A vale 0, y 1 si ésta vale 1. Es decir, se cumple que $S = A$. En cambio, si la entrada CONTROL se encuentra a 1 lógico, la salida $S = \overline{A}$. Se puede comprobar, que una puerta EXOR de dos entradas se comporta como un circuito complementador de 1 bit, gobernado por la entrada CONTROL. Si la entrada CONTROL está a 0, el bit de entrada A pasa a la salida tal cual; si la entrada CONTROL está a 1, el bit de entrada A pasa a la salida complementado.

Si este complementador de 1 bit se repite n veces, se obtendrá un circuito complementador a 1, tal como muestra la figura 5.13.

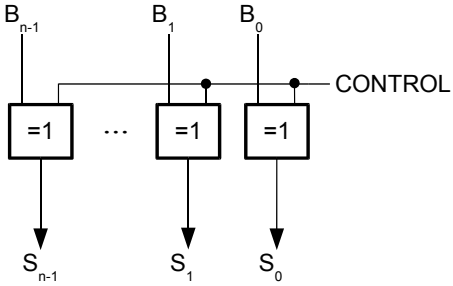


Figura 5.13.

La **aritmética en Ca1** exige que al resultado de la suma o resta, se debe añadir una unidad, en caso de generarse acarreo, para obtener el resultado correcto. El circuito sumador/restador en complemento a 1 se muestra en la figura 5.14.

Si se desea trabajar en **aritmética en Ca2**, habrá que realizar algunas modificaciones al circuito sumador/restador anterior. En primer lugar, el circuito

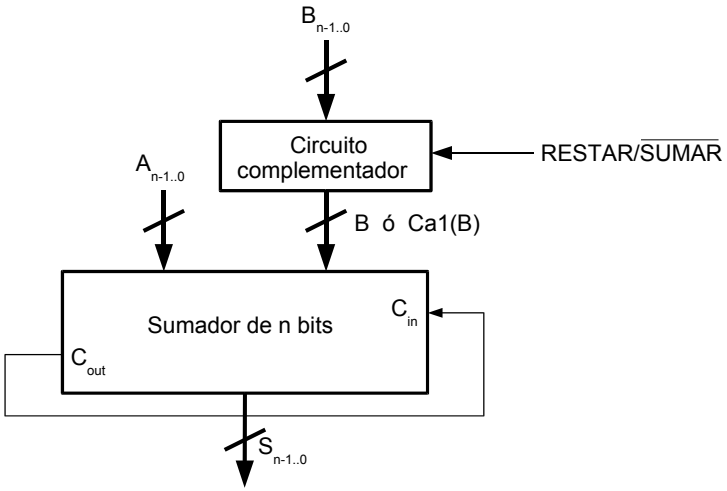


Figura 5.14.

complementador sólo permite generar el Ca1 del número B, y se sabe que el $\mathrm{Ca2}(A) = \mathrm{Ca1}(A) + 1$, por lo que hay que sumar 1, al resultado final. Por otro lado, la aritmética en Ca2 elimina los posibles acarreos que se generen. El circuito que realiza esta tarea se muestra en la figura 5.15.

Si $\mathrm{RESTAR}/\overline{\mathrm{SUMAR}} = 0$, el complementador deja pasar el número B tal cual, y el sumador genera la suma $A+B$. Si $\mathrm{RESTAR}/\overline{\mathrm{SUMAR}} = 1$. El complementador genera el $\mathrm{Ca1}(B)$ y el sumador, $A+\mathrm{Ca1}(B)+1 = A+\mathrm{Ca2}(B) = A-B$ en Ca2.

5.2.5. Sumador BCD: aritmética decimal

La importancia de la aritmética decimal es de tal grado que en múltiples ocasiones es deseable que los sistemas digitales operen con datos decimales (codificados en binario, BCD). En este apartado se construirá un circuito capaz de sumar dos números decimales codificados (dos números BCD) y generar el resultado también en BCD. Este sumador será modular, por lo que conexiones en cascada de sumadores BCD permitirán la suma de números con más dígitos. Es evidente que el núcleo de este sumador BCD está formado por un sumador binario de 4 bits, capaz de sumar los bits de los números BCD. La cuestión importante es la de generar un resultado también en BCD.

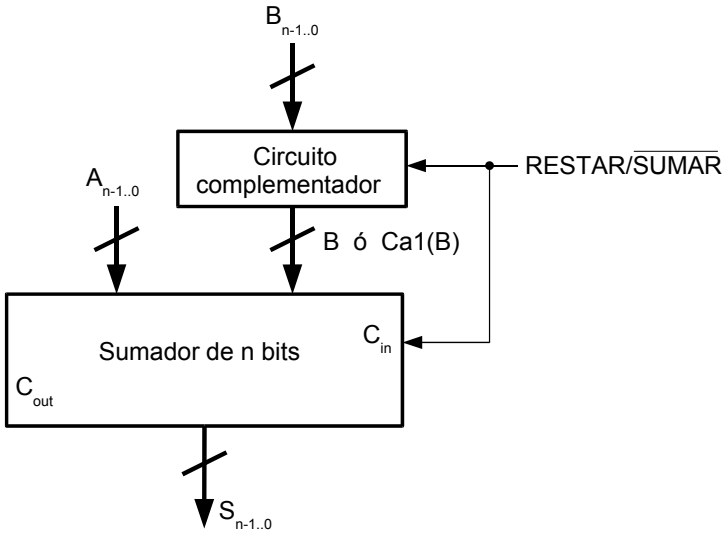


Figura 5.15.

Algunos aspectos deben ser considerados en la aritmética BCD. En la suma de dos digitos BCD con sumadores binarios de 4 bits, pueden darse los siguientes casos:

(a) **El resultado es un número BCD y no existe arrastre**:

$$\begin{array}{cccccl} & 0 & 1 & 0 & 1 & = 5 \\ & 1 & 1 & 0 & 0 + & = 4 \\ \hline & 1 & 0 & 0 & 1 & = 9 \checkmark \end{array}$$

(b) **El resultado no es un número BCD**:

$$\begin{array}{cccccl} 1 & 0 & 0 & 1 & & = 9 \\ 0 & 1 & 1 & 0 & + & = 6 \\ \hline 1 & 1 & 1 & 1 & & \text{no es BCD} \times \end{array}$$

(c) **El resultado es un número BCD y se genera arrastre**:

$$\begin{array}{cccccl} & 1 & 0 & 0 & 1 & = 9 \\ & 1 & 0 & 0 & 0 + & = 8 \\ \hline 1 & 0 & 0 & 0 & 1 & = 11 \times \end{array}$$

Para el caso a el resultado generado por el sumador binario es un número BCD y es correcto. Para el caso b, el resultado es correcto pero no está expresado en BCD. Para el caso c el resultado binario generado es correcto, pero no lo es si se interpreta en BCD (11_{BCD} para el ejemplo, debiendo ser 17_{BCD}).

Los casos b y c pueden resolverse si se le suma la magnitud 0110_2 (6_{10}) al resultado obtenido por el sumador binario.

Para el ejemplo del caso b, $1001_2 + 0110_2$ ($9_{10} + 6_{10}$) debe generar un resultado en BCD de dos dígitos, $1\,0101_2$ (15_{10}). En BCD esto se traduce en un bit de acarreo y la cantidad 0101_2.

$$\begin{array}{cccccc} & 1 & 0 & 0 & 1 & = 9 \\ & 0 & 1 & 1 & 0 \;+ & = 6 \\ \hline & 1 & 1 & 1 & 1 & \\ & 0 & 1 & 1 & 0 \;+ & \\ \hline 1 & 0 & 1 & 0 & 1 & = 15 \;\checkmark \end{array}$$

El caso c se resuelve del mismo modo: sumando 0110_2 a lo obtenido se consigue el resultado correcto (17_{BCD} en lugar de 11_{BCD}).

$$\begin{array}{cccccc} & 1 & 0 & 0 & 1 & = 9 \\ & 1 & 0 & 0 & 0 \;+ & = 8 \\ \hline 1 & 0 & 0 & 0 & 1 & \\ & 0 & 1 & 1 & 0 \;+ & \\ \hline 1 & 0 & 1 & 1 & 1 & = 17 \;\checkmark \end{array}$$

El sumador BCD debe estar compuesto, pues, de dos sumadores binarios de 4 bits. Uno de ellos suma los dígitos BCD tal cuales, y el otro, en caso de detectarse los casos b o c debe añadir la cantidad 0110_2 al resultado. Es evidente que se debe acompañar de cierta circuitería que permita la detección de los casos b y c con el objeto de decidir si es necesario efectuar la corrección o no.

El caso b se obtiene con un detector de números BCD. Este es un circuito combinacional muy básico que activa su salida si sus entradas no son un

número BCD (cualquier valor de 4 bits mayor que 9). También puede diseñarse este detector con un comparador de magnitudes.

El caso c) se detecta simplemente cuando se pone a 1 el acarreo de salida del sumador binario.

Si se da uno cualquiera de los casos b o c, o sea, si la salida del detector es 1 o la salida de acarreo del sumador es 1, se debe sumar 0110_2 al resultado obtenido por el primer sumador. En caso contrario, no hay que sumar nada. En la figura 5.16 se representa el sumador BCD.

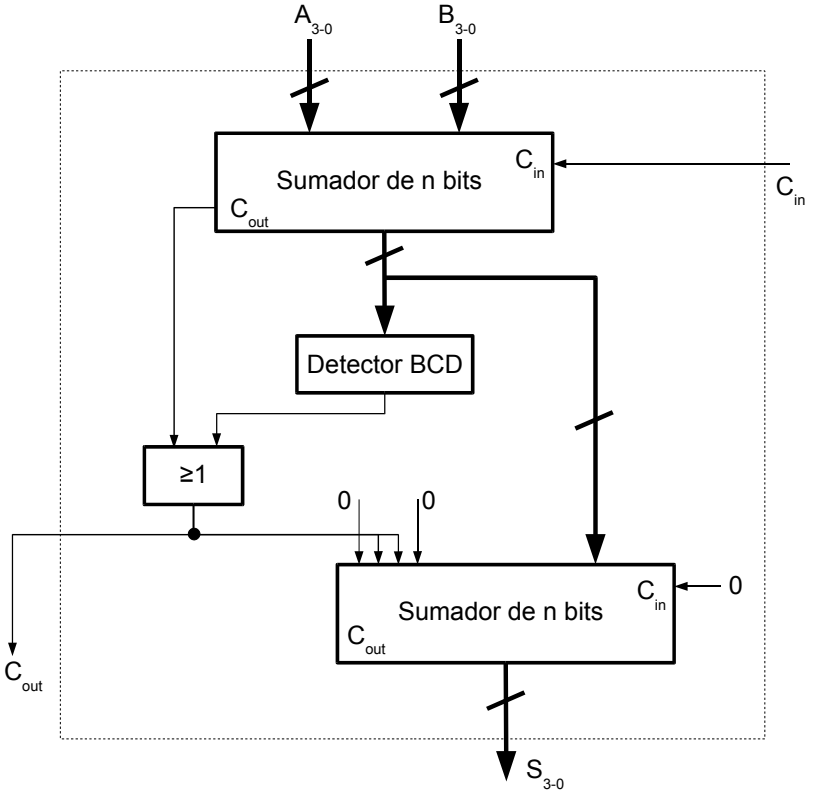


Figura 5.16.

5.3. Unidad aritmético-lógica

Una unidad aritmético-lógica o **ALU** (*Arithmetic-Logic Unit*) de n bits es un dispositivo combinacional que acepta dos palabras de entrada A y B, de n bits cada una y genera un resultado de n bits (además de cierta información como acarreo, desbordamiento, etc.) procedente de la realización de alguna operación aritmética o lógica identificada por unas señales se selección.

La figura 5.17 muestra dos posibles símbolos que representan a una ALU de 4 bits capaz de realizar ocho operaciones aritmético-lógicas. La entrada S_2 distingue entre operaciones lógicas y aritméticas, mientras que las entradas S_1, S_0 eligen, dentro de cada tipo, la función a realizar. A y B son las entradas y F la salida. C_{in} y C_{out} son los acarreos de entrada y salida para las operaciones aritméticas.

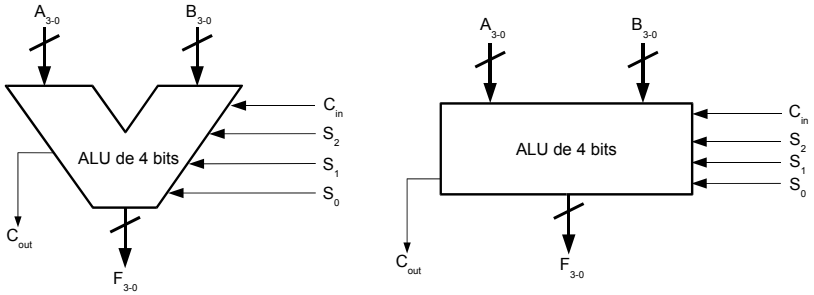


Figura 5.17.

En esta sección se pretende diseñar una ALU, además de mostrar una ALU comercial.

El diseño de una ALU se realiza en tres etapas: diseño del circuito aritmético, diseño del circuito lógico y unión de las partes anteriores.

5.3.1. Diseño del circuito aritmético

El componente básico del circuito aritmético de una ALU de n bits es un sumador paralelo de n bits. A modo de ejemplo realizaremos el diseño para $n = 4$ bits. Si se controlan las entradas de dicho sumador, que llamaremos X

e Y, su salida S puede generar distintas funciones aritméticas:

(a) Si las entradas del sumador son A y $00\ldots0$, figura 5.18, se pueden obtener funciones de **transferencia**, $F = A$ si $C_{in} = 0$, o de **incremento**, $F = A + 1$ si $C_{in} = 1$.

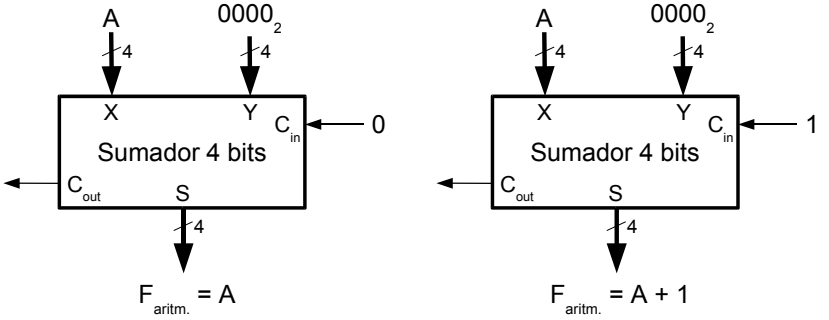


Figura 5.18.

(b) Si las entradas del sumador son A y B (los números de entrada de la ALU), figura 5.19, se pueden obtener funciones de **suma**: $F = A + B$ si $C_{in} = 0$, o $F = A + B + 1$ si $C_{in} = 1$.

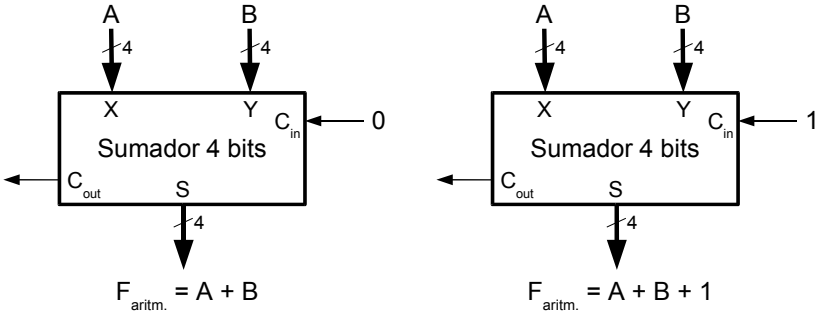


Figura 5.19.

(c) Si las entradas del sumador son A y $\bar{B}$ (el complemento de los bits del número B, es decir, $\mathrm{Ca1}(B)$), figura 5.20, se pueden obtener operaciones de **resta en Ca1**, $F = A + \bar{B} = A + \mathrm{Ca1}(B)$ si $C_{in} = 0$, o de **resta en Ca2**, $F = A + \bar{B} + 1 = A + \mathrm{Ca2}(B)$ si $C_{in} = 1$.

(d) Si las entradas del sumador son A y $11\ldots1$, figura 5.21, se pueden obtener funciones de **transferencia**, $F = A$ si $C_{in} = 1$, o de **decremento**,

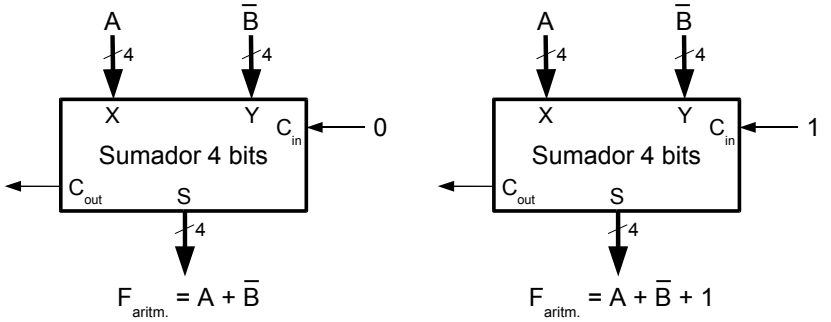


Figura 5.20.

$F = A - 1$ si $C_{in} = 0$.

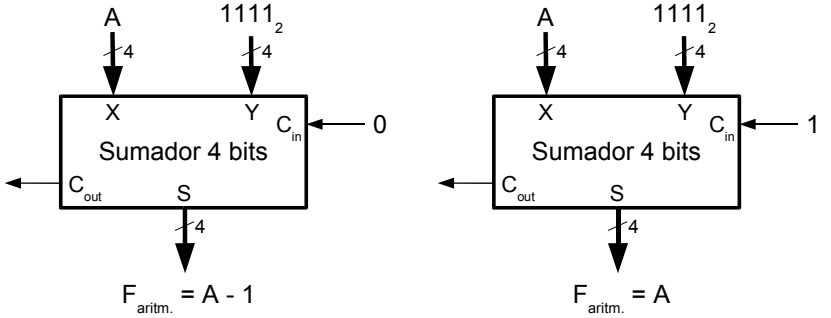


Figura 5.21.

De lo expuesto anteriormente se observa que la entrada X del sumador completo siempre se conecta con la entrada del número A de la ALU, mientras que la entrada Y del sumador, de forma alternativa, se conecta a la entrada B, a $\bar{B}$, a $00\ldots0$ y a $11\ldots1$. Esto da un total de cuatro combinaciones aritméticas posibles (recordamos que sólo se dispone de dos líneas de selección de operación aritmética) aunque si consideramos el acarreo de entrada, el número de operaciones aritméticas posibles es mayor.

Se debe diseñar un circuito combinacional que, en función de S_1 y S_0, permita seleccionar qué llega a la entrada Y del sumador completo y, con ello, determine qué operación aritmética realizará éste. La tabla de verdad de la figura 5.22 muestra el comportamiento de Y_i en función de B_i, S_1 y S_0.

S_1	S_0	Y_i
0	0	0
0	1	B_i
1	0	$\overline{B_i}$
1	1	1

Figura 5.22.

La ecuación de Y_i será:

$$Y_i = B_i\bar{S}_1S_0 + \bar{B}_iS_1\bar{S}_0 + S_1S_0 = B_iS_0 + \bar{B}_iS_1$$

que corresponde al circuito de la figura 5.23.

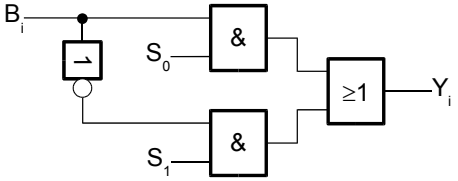


Figura 5.23.

La etapa i-ésima de la parte artimética de la ALU sería, por tanto, la ilustrada en la figura 5.24.

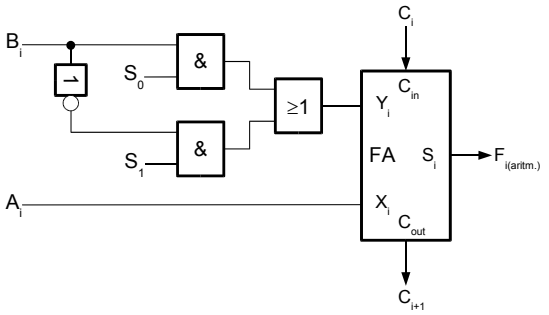


Figura 5.24.

El acarreo de salida nos puede dar una información muy importante, véase la tabla 5.3.

Tabla 5.3:

S_1	S_0	C_{in}	$F=$	Operación	C_{out}
0	0	0	A	Transferir A	$C_{out}=0$ siempre
0	0	1	$A+1$	Incrementar A	Si $A=2^n-1$, $C_{out}=1$ y $F=0$
0	1	0	$A+B$	Sumar $A+B$	Si $A+B\geq 2^n$, $C_{out}=1$ y desbordamiento
0	1	1	$A+B+1$	Incrementar $A+B$	Si $A+B\geq 2^n-1$, $C_{out}=1$ y desbordamiento
1	0	0	$A+\bar{B}$	Restar $A-B$ en Ca1	Si $A>B$, $C_{out}=1$ Si $A\leq B$, $C_{out}=0$
1	0	1	$A+\bar{B}+1$	Restar $A-B$ en Ca2	Si $A\geq B$, $C_{out}=1$ Si $A<B$, $C_{out}=0$
1	1	0	$A-1$	Decrementar A	Si $C_{out}=0$, $A=0$
1	1	1	A	Transferir A	$C_{out}=1$ siempre

5.3.2. Diseño del circuito lógico

Supongamos que las operaciones lógicas a realizar son: A AND B, A OR B, A EXOR B y NOT A, donde A y B son las palabras de entrada en la ALU[2]. La selección de una operación u otra se hará, al igual que en la parte aritmética, dependiendo del valor que tomen las entradas S_1 y S_0. Arbitrariamente se ha tomado la asignación siguiente: 00 para OR, 01 para EXOR, 10 para AND y 11 para NOT.

La etapa i-ésima del circuito lógico aparece en la figura 5.25. Ésta permite calcular las distintas operaciones lógicas sobre los bits A_i y B_i de entrada a la ALU. Dado que estas operaciones son a nivel de bit, las etapas del circuito lógico son independientes entre sí, es decir, no necesitan interconexiones entre ellas. Tan sólo comparten las líneas de selección de operación, S_1 y S_0.

5.3.3. Unión de los circuitos aritmético y lógico

Para construir la ALU se han de interconectar los dos circuitos que la forman, aritmético y lógico. Las variables S_1 y S_0 serán comunes para las dos partes, y será S_2 la que seleccione una u otra. La siguiente figura representa una posible solución para la etapa i-ésima de una ALU.

[2]Se entiende que A AND B (o cualquier otra operación lógica) genera un resultado F, donde cada bit, $F_i = A_i \text{AND} B_i$.

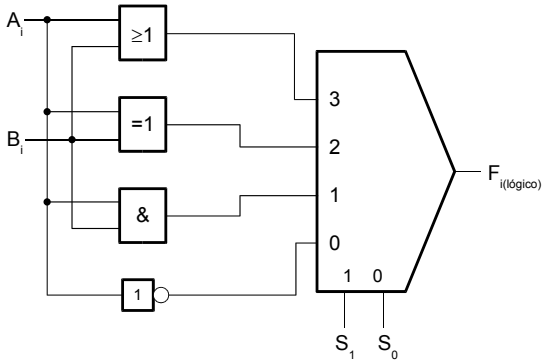


Figura 5.25.

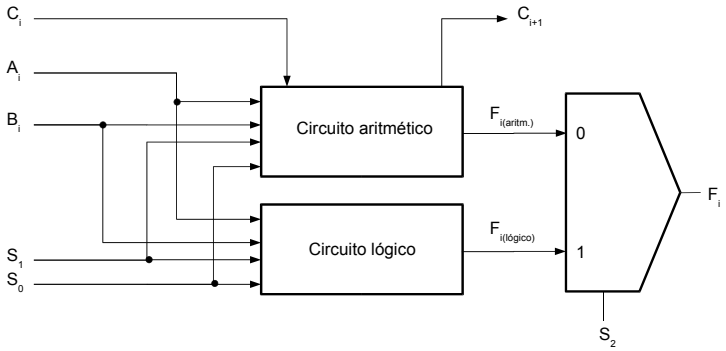
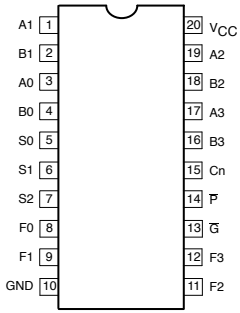


Figura 5.26.

Combinando n etapas, obtenemos una ALU de n bits. La única interconexión entre la etapa i-ésima y la siguiente es el acarreo C_i (parte aritmética). En la primera etapa, $i = 0$, el acarreo C_0 sirve como acarreo de entrada C_{in} de la ALU para las operaciones aritméticas; análogamente, la última etapa, $i = n-1$ genera el acarreo C_n, que se corresponde con el acarreo de salida C_{out} de la ALU.

5.3.4. Descripción de una ALU comercial

Las siguientes figuras ilustran el diagrama de conexionado y la tabla de operaciones de la ALU 74F381.



PINS	DESCRIPTION
A0 – A3	A operand inputs
B0 – B3	A operand inputs
S0 – S2	Function select inputs
Cn	Carry input
$\overline{P}$	Carry Propagate ouptut (active-Low)
$\overline{G}$	Carry Generate ouptut (active-Low)
F0–F3	Outputs

Figura 5.27. titulo

FUNCTION SELECT TABLE

SELECT			OPERATING MODE
S0	S1	S2	
L	L	L	Clear
H	L	L	B minus A
L	H	L	A minus B
H	H	L	A Plus B
L	L	H	A $\oplus$ B
H	L	H	A + B
L	H	H	AB
H	H	H	Preset

H = High voltage level
L = Low voltage level

Figura 5.28. titulo

Obsérvese que la operación aritmética de suma aparece como PLUS para distinguirla de la operación lógica OR, que aparece expresada mediante ope-

rador +. Asimismo, la operación aritmética de resta aparece como MINUS.

Ejercicios propuestos

Problema 5.1 Defina qué es el Overflow y cómo se detecta en operaciones de números con signo y sin signo. Ponga ejemplos.

Problema 5.2 Se desea diseñar un circuito que permita convertir un número binario de 5 bits en su correspondiente en BCD. (Ej.: $10111_2 = 23_{10} = 0010\ 0011_{BCD}$). Base su diseño, exclusivamente en sumadores de 4 bits y comparadores de magnitud de 4 bits.

Problema 5.3

- Construir la ALU de 4 bits especificada por la tabla de operación 5.4 y para lo que se disponen de sumadores de 1 bit, puertas y multiplexores 2:1. Asimismo, la ALU debe disponer de entrada y salida de acarreo para su conexión en cascada. Expresar el símbolo del subsistema de esta ALU.

Tabla 5.4:

S_1	S_0	$F =$
0	0	$2A$
0	1	$1111_{(2}$
1	0	$A + B$
1	1	$A - B$ en Ca1

- A partir del subsistema diseñado, construir una ALU de 12 bits.
- Dados dos números de 12 bits A y B, indique cómo puede realizarse la operación $F = A - B$ en Ca2 usando la ALU del apartado anterior.

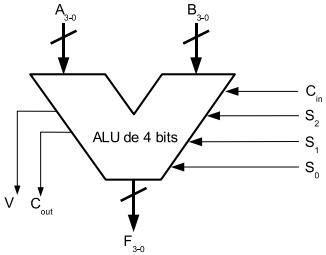
Problema 5.4 Se dispone de una ALU de 4 bits (figura 5.29)que realiza las operaciones que se muestran en la tabla adjunta.

a) Determine los valores lógicos que toman las salidas V,C_{out} y F_{3-0} para todos los posibles valores S_2 S_1 S_0, si A_{3-0} = 1000, B_{3-0} =1010 y C_{in}=1.

b) Diseñe el circuito aritmético de la ALU.

$S_2\ S_1\ S_0$	$F =$
0 0 0	$A + B + C_{in}$
0 0 1	$A + Ca1(B) + C_{in}$
0 1 0	$A + C_{in}$
0 1 1	$A + 2^4 - 1 + C_{in}$
1 0 0	AND
1 0 1	OR
1 1 O	EXOR
1 1 1	NOT A

(a) Tabla de opreración



(b) ALU

Figura 5.29.

Problema 5.5 Se desea diseñar un circuito combinacional (ver figura 5.30) que reciba un número de tres bits, A_{2-0}, y una entrada de control X_i, y genera el Ca1 o el Ca2 del número de entrada A_{2-0} en función de X_i ($X_i = 0$ genera Ca2 y $X_i = 1$, el Ca1). Además el circuito debe disponer de una salida adicional, X_o, que se pone a 1 cuando existe un 1 en alguna de las entradas X_i, A_{2-0}, y, 0, para cualquier otro caso.

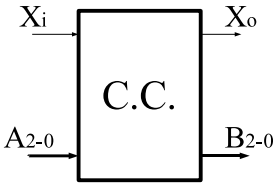


Figura 5.30.

a) Implementa el circuito correspondiente usando, exclusivamente, multiplexores de cuatro canales.

b) Determina cuál es la función que realiza el circuito de la figura 5.31.

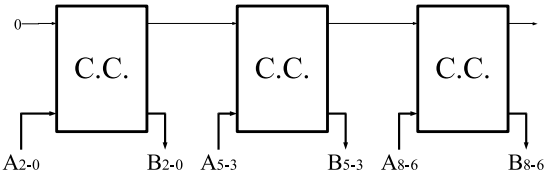


Figura 5.31.

Problema 5.6 Un circuito restador binario de 8 bits se va a utilizar para comparar dos números, A y B, de 8 bits. Este restador binario, además de generar el resultado de la diferencia $R = A - B$, dispone de salidas especializadas

que dan cierta información adicional: salida N, signo del resultado, toma el valor lógico del bit de signo del resultado (R_7); salida Z, resultado cero, se pone a 1, si el resultado R es igual a 0; salida C, arraste, es el bit de *borrow de la resta; salida V, overflow en Ca2, esta salida se pone a 1 cuando existe sobrecapacidad en operaciones con números en complemento a 2 (Ca2).*

a) Justifique que $V = \overline{A_7} \bullet B_7 \bullet R_7 + A_7 \bullet \overline{B_7} \bullet \overline{R_7}$

b) Si las entradas A y B del restador expresan números en notación Ca2 demuestre que si $[Z + (N \oplus V)] = 0$, entonces $A > B$.

c) Si A y B representan números sin signo, demuestre que si $C = 0$, entonces $A \geq B$.

Problema 5.7 Diseñar una Unidad Aritmético-Lógica (ALU) que permita operar con dos números binarios (a y b) de ocho bits, pudiendo realizar las siguientes operaciones:

a) aritméticas: (1) sumar sumar ($a+b$) con y sin acarreo; (2) decrementar a; (3) incrementar a ; (4) transferir a; (5) restar ($a-b$); (6) restar ($a-b-1$).

b) lógicas: (1) $\overline{a \otimes b}$; (2) $\overline{a \bullet b}$; (3) $\overline{a+b}$ y (4) $\overline{a}$.

Problema 5.8 Se desea contruir un circuito que admita como entradas tres números A, B y C de n bits y una salida, z, que se pone a a 1 si el número B es el más cercano al A y a 0 en caso contrario.

Problema 5.9

a) Construir un sumador de dos números de un dígito BCD cada uno y que muestre el resultado, también, en BCD. Se deberán tener en cuenta los acarreos de entrada y salida.

b) A partir del sumador del apartado anterior, construya un sumador de dos números de 4 dígitos BCD.

Problema 5.10

(a) Realice las siguientes multiplicaciones en aritmética binaria: (1) $110_2 \times 11_2$; (2) $110_2 \times 110_2$.

(b) Diseñe un multiplicador que permita realizar el producto aritmético de un número, A, de tres bits, con otro, B, de dos bits, ambos sin signo.Utilizar un sumador de 4 bits y puertas AND de dos entradas. Dibujar el símbolo del subsistema construido.

(c) A partir de un sumador de 4 bits y un sumador FA de un bit, construya un sumador de 5 bits.

(d)Diseñe un multiplicador par dos números, A y B, ambos sin signo y de tres bits. Para ello utilice el multiplicador construido en el apartado (b) el sumador del (c) y puertas AND de dos entradas.

Capítulo 6

Análisis y diseño de circuitos secuenciales

6.1. Introducción

Hasta ahora, los circuitos digitales estudiados disponen de un conjunto de salidas que para un instante t,(salvo tiempos de retraso asociados a las puertas lógicas que los constituyen) toman un valor que es, siempre, combinación de los valores de las entradas en ese mismo instante t. De esta forma, si las entradas cambian, las salidas también cambian y, para una combinación de entradas dada, las salidas siempre toman el valor asociado a dicha combinación. Estos circuitos son **combinacionales**.

La mayoría de los circuitos digitales difieren de estos circuitos combinacionales en la incorporación de elementos capaces de almacenar información, de forma que las salidas de dichos circuitos se comportan de forma más compleja. En este caso, las salidas son función de la combinación de entradas y de los datos almacenados en la memoria, los cuales, a su vez, pueden cambiar de valor en función de las entradas. Estos nuevos circuitos deben ser descritos mediante una lógica **secuencial**, pues aquí ocurre que una misma combinación de entradas puede generar distintas salidas ya que los datos almacenados en memoria pueden ser diferentes en cada caso, y estos datos almacenados dependen del historial de entradas o, dicho de otra forma, de la *secuencia* de

entradas que pudiera haber tenido dicho circuito.

En la figura 6.1 se representa el diagrama de bloques de un circuito secuencial.

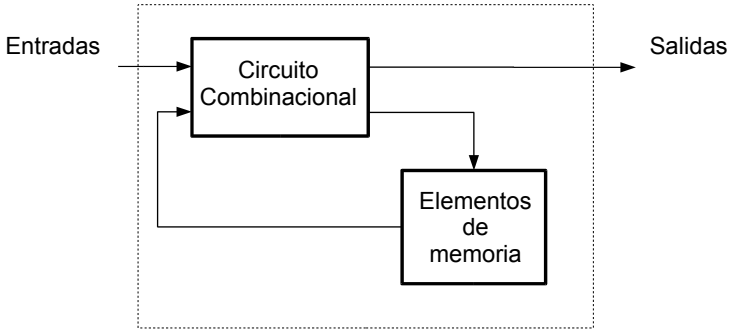


Figura 6.1.

Un **circuito secuencial** (en adelante, CS) dispone de un bloque puramente secuencial, constituido por elementos de memoria donde se almacenan bits, y de un bloque combinacional, que recibe las entradas del CS y los valores almacenados en los elementos de memoria y genera las salidas del CS y los próximos valores a almacenar en los elementos de memoria.

Se justifica pues, el interés de comenzar el estudio de los CS por los elementos de memoria elementales, es decir, los capaces de almacenar un bit. Básicamente, podemos decir que estos elementos de memoria están constituidos por puertas lógicas que disponen de **realimentaciones** de sus salidas hacia sus entradas. La figura 6.2 muestra un ejemplo de elemento de memoria constituido por una puerta OR con una única realimentación de su salida hacia una de sus dos entradas.

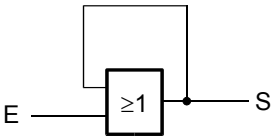


Figura 6.2.

Si $E = 0$ e inicialmente $S = 0$, la salida seguirá siendo 0. Si $E = 1$, la salida $S = 1$, pero si en este caso, E vuelve a pasar a 0, la salida seguirá siendo 1.

Este es un ejemplo muy básico de un elemento de memoria que ha sido capaz de almacenar un 1 de forma perpetua. En general, para que un elemento de memoria que almacene un bit sea útil, debe tener capacidad de almacenar un 1 o un 0, el tiempo que se precise, y disponer de los terminales adecuados para la escritura del 1 y del 0. Estos elementos de memoria se denominan **biestables**.

6.2. Biestables

6.2.1. Biestable SR realizado con puertas NOR

La figura 6.3 muestra la estructura del biestable SR-NOR que pasaremos a analizar.

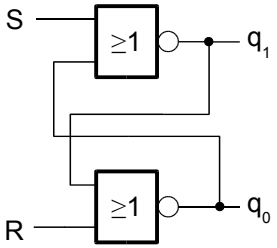


Figura 6.3.

Este biestable tiene dos entradas, S y R, y dos salidas q_1 y q_2. La entrada S (que viene de *Set*, "puesta a 1") cuando se activa provoca la escritura de un 1 lógico en el biestable; si se activa la entrada R (que viene de *Reset*, "puesta a 0") se provoca el borrado del biestable, es decir, la escritura de un 0 lógico; si ninguna de estas entradas están activas, el biestable almacena el valor (1 o 0) escrito anteriormente. Es evidente, que S y R no deben estar activas simultáneamente. Las salidas q_1 y q_2, deben contener, de alguna manera, el valor almacenado en el biestable.

Las salidas q_1 y q_2 dependen de S, R y de sí mismas, también. Si asumimos que las puertas NOR introducen un cierto tiempo de propagación t_p, podemos distinguir los valores q_1, q_2 de entrada de las puertas con los valores q_1, q_2 que tomarán las salidas de las mismas pasado un tiempo igual al tiempo de

propagación. Denotaremos por Q_1, Q_2, a los valores futuros de las salidas (una vez transcurrido el tiempo de propagación), y por q_1, q_2 a sus valores actuales (y que por tanto, son aplicados a las entradas de las puertas NOR). De esta forma:

$$\begin{aligned} Q_1 &= \overline{S + q_2} = \overline{S}\,\overline{q_2} \\ Q_2 &= \overline{R + q_1} = \overline{R}\,\overline{q_1} \end{aligned}$$

Si representamos estas expresiones en un K-mapa, obtenemos lo representado en la figura 6.4.

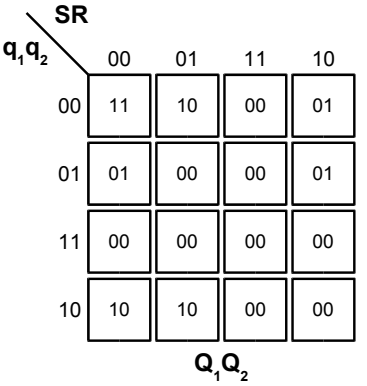


Figura 6.4.

A partir de esta tabla podemos conocer cómo funciona dicho biestable y lo haremos analizando la evolución temporal de las salidas q_1, q_2 para una secuencia de sus entradas S, R, tal como se muestra en la figura 6.5.

Suponemos que inicialmente q_1, q_2 toman los valores 1 y 0, respectivamente. En el instante $t = 0$, $SR = 00$, y $q_1q_2 = 10$, según el K-mapa, $Q_1Q_2 = 10$, esto es, los próximos valores de las salidas del biestable coinciden con los valores actuales. Esto define una situación o **estado** estable que contiuará en el tiempo mientras que las entradas SR no cambien. Así alcanzamos el instante $t = t_1$, donde la entrada S se pone a 1 lógico (pulso de *Set*). Para esta nueva situación, el K-mapa del biestable indica que los próximos valores a tomar por las salidas (transcurrido el tiempo t_p de propagación de las puertas NOR) será 00. Pero esta situación es inestable, puesto que una vez que las salidas pasen a tomar estos valores, vuelven a introducirse como entradas, las cuales generan unas nuevas salidas. Esto es, para $SR = 10$ y $q_1q_2 = 00$, las nuevas

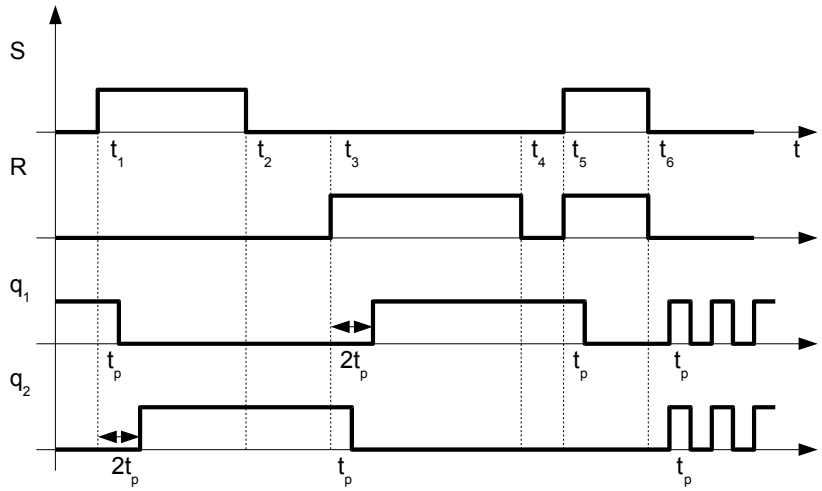


Figura 6.5.

salidas son $Q_1Q_2 = 01$. Estos nuevos valores que se han establecido en un tiempo igual a $2 \times t_p$ se mantienen estables mientras que las entradas SR no cambien. Así se llega al instante $t = t_2$, en el que la entrada S se pone a 0 lógico. Las próximas salidas para $SR = 00$ y $q_1q_2 = 01$ son $q_1q_2 = 01$, por tanto tenemos también una situación de estabilidad que se mantendrá mientras SR no cambie. Así llegamos a $t = t_3$, donde se inicia un pulso de *Reset*, $R = 1$. Las próximas salidas para $SR = 01$ y $q_1q_2 = 01$ son $Q_1Q_2 = 00$ que se generan en el instante $t = t_3 + t_p$. Estas nuevas salidas pasan a ser entradas y generan unas nuevas salidas que serán, para $SR = 01$ y $q_1q_2 = 00$, $Q_1Q_2 = 10$ a partir de $t = t_3 + 2t_p$. A partir de aquí se alcanza una nueva situación de estabilidad que permanecerá mientras que SR no cambien. En $t = t_4$, $SR = 00$, mientras que $q_1q_2 = 10$, nuevamente tenemos una situación estable, por lo que las salidas no cambian de valor.

Es hora de recordar la funcionalidad de las entradas SR. La línea S provoca la escritura de un 1 en el biestable, la línea R la escritura de un 0, y si ninguna está activa, el biestable mantiene el valor almacenado. Si observamos el diagrama temporal durante el intervalo $[t_1, t_2]$, donde se genera pulso de *Set*, la salida q_2 se pone a 1, mientras que q_1 a 0. Parece razonable, pues, comenzar a intuir que q_2 debe ser la salida del biestable que muestra el valor almacenado. Si observamos el diagrama temporal durante el intervalo $[t_3, t_4]$, pulso de Reset, observamos que q_2 pone a 0, mientras que q_1 se pone a 1. Además, si $SR = 00$, intervalos $[0, t_1]$, $[t_2, t_3]$ y $[t_4, t_5]$, las salidas no cambian de valor

manteniendo el bit que previamente se almacenó mediante un pulso de *Set* o *Reset*.

Sigamos analizando el resto del diagrama temporal a partir del instante t_5. En este caso se generan simultanemante un pulso de *Set* y *Reset*, que tienen igual duración. Cómo ya se mencionó anteriormente, no tiene sentido forzar a un dispositivo a escribir un 1 y un 0 simultáneamente. Aquí mostraremos una justificación más de que estas entradas deben estar prohibidas. Si $SR = 11$, los próximos valores de las salidas q_1q_2 son 00, y está situación se mantiene estable mientras que SR no cambien. En el instante $t = t_6$, ambas entradas pasan a 00. En este caso, los próximos valores de las salidas son 11; nuevamente estas salidas pasan a ser entradas, por lo que generan unas nuevas salidas, 00; que nuevamente pasan a ser entradas y generan las salidas 11; y así sucesivamente. El biestable ha entrado en una situación oscilatoria que dista mucho del comportamiento funcional que se requiere de él.

La circunstancia analizada anteriormente no es la única que podría provocar un comportamiento oscilatorio no deseado de las salidas del biestable. De hecho esta situación se genera siempre que en algún momento q_1q_2 sean 00 y las entradas SR también lo sean. Si observamos con detenimiento la evolución temporal de q_1q_2, observamos que cuando se generaban pulsos de *Set* o *Reset*, q_1q_2 tomaban los valores 00 durante un periodo corto de tiempo antes de conseguir el valor adecuado. Esto hace pensar que si se generan pulsos de *Set* o *Reset* excesivamente cortos, el biestable también podría malfuncionar.

Por todo lo expuesto anteriormente, cabe hacer las siguientes restricciones funcionales al biestable SR:

- Las entradas $SR = 11$ quedan prohibidas.
- La duración de los pulsos de *Set* y *Reset* debe ser superior a $2 \times t_p$ (tiempo necesario para que las salidas q_1q_2 tomen un valor estable adecuado).

Si estas restricciones son satisfechas, podemos decir que las salidas del biestable nunca tomarán los valores 00, ni tampoco 11, ya que estos últimos se obtienen a partir de los primeros. Además podemos afirmar que el biestable tiene una salida q que muestra el valor almacenado en el mismo, que se corresponde con q_2, y una línea $\bar{q}$ que muestra el valor complementado, y que se corresponde con q_1. Por todo esto pasamos a una descripción más simplificada del K-mapa del biestable SR, figura 6.6.

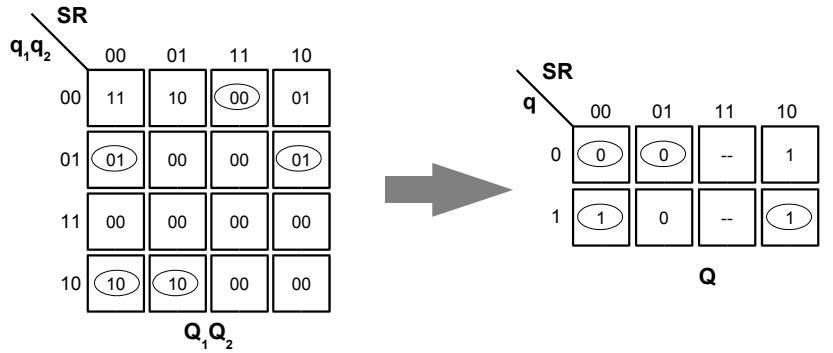


Figura 6.6.

Con un círculo se han representado aquellas situaciones que producen estabilidad, esto es, cuando el proximo valor de la salida coincide con el de la entrada.

El símbolo gráfico del biestable SR es el representado en la figura 6.7.

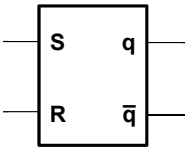


Figura 6.7.

6.2.2. Biestable SR-NAND

Cualquier circuito que se pueda construir con puertas NOR, puede realizarse también con puertas NAND. En la figura 6.8 se muestra el proceso de obtención de un biestable, el NAND, a partir del SR-NOR.

Aplicando la ley de Morgan, $\overline{x+y} = \overline{x}\,\overline{y}$, sobre la parte (a) de la figura, se obtiene la parte (b). Si ahora asociamos la puerta AND con el inversor, obtenemos una NAND, parte (c), donde destacamos que ahora la salida q del biestable se corresponde con la salida superior, mientras que la $\bar{q}$ es la inferior. Asociar los inversores con las entradas SR, es equivalente a suponer que estas son activas en bajo, parte (d).

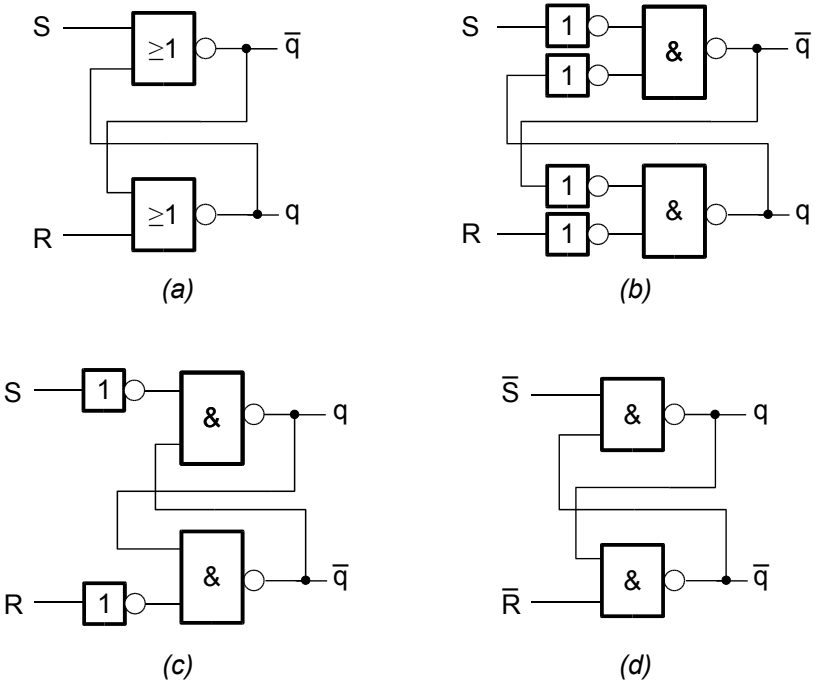


Figura 6.8.

La tabla funcional del SR-NAND es la mostrada en la figura 6.9.

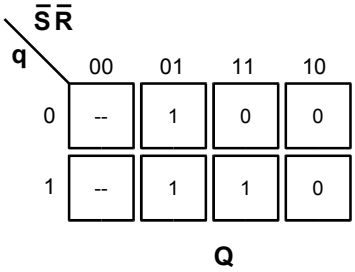


Figura 6.9.

6.2.3. Biestables síncronos

Existen dos tipos de biestables, los **asíncronos** y los **síncronos**. Los que hemos visto hasta ahora son asíncronos, es decir, no tienen ningún tipo de mecanimo que permita su sincronización. La sincronización de un biestable viene dada por una línea adicional, denominada línea de reloj, cuya función es la de indicar al biestable cuándo puede cambiar de estado, es decir, cuándo puede cambiar sus salidas. En el biestable SR-NOR estudiado, cuando las entradas cambian, el biestable puede cambiar de estado, y esta situación se repite siempre que cambien las entradas ya que no existe ninguna línea adicional que pueda indicarle cuándo el biestable puede cambiar y cuándo no. La necesidad de biestables síncronos se hace patente en sistemas secuenciales reales los cuales pueden generar transiciones inesperadas (azares) que si tienen la duración suficiente pueden provocar que los biestables del CS cambien de estado o que estos entren en un funcionamiento caótico (oscilaciones). La utilización de biestables síncronos minimiza estos problemas, ya que la señal de reloj inhibiría el funcionamiento del biestable durante el tiempo donde estas transiciones inesperadas tienen mayor presencia y habilitaría el funcionamiento del mismo cuando hubiese pasado el "peligro" asociado a estos azares.

Cuando un biestable síncrono está habilitado se dice que se ha producido el **disparo** del mismo. La señal de reloj es la que genera, de forma periódica, dichos disparos. En la mayoría de las ocasiones, la señal de reloj no es más que una señal binaria, periódica, cuadrada, de unos y ceros que se van alternando en el tiempo.

Existen varios tipos de biestables síncronos: **disparados por nivel**, ***master-slave*** (amo-esclavo), y **disparados por flanco**. A continuación estudiaremos cada uno de ellos, utilizando, siempre como base, el biestable SR.

6.2.3.1. Biestables disparados por nivel

La figura 6.10 muestra la estructura de un biestable SR-NOR disparado por nivel. Este está constituido por un biestable SR-NOR (biestable asíncrono), dos puertas AND cuyas salidas controlan las entradas SR del biestable asincrono y cuyas entradas son S y R, y la señal de reloj clk.

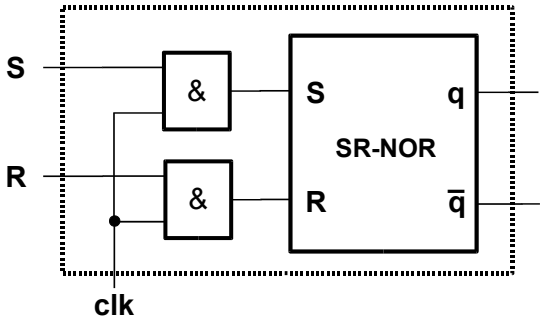


Figura 6.10.

El funcionamiento del mismo es el siguiente. Si $clk = 0$, las salidas de las puertas AND son cero, y por tanto el biestable asíncrono tiene como entradas 00, por lo que no cambia de estado. Este estado permanece inalterado mientras que clk siga siendo 0, independientemente de los valores que tomen las entradas SR. Ahora bien si $clk = 1$, las entradas SR actúan directamente sobre las entradas del biestable asíncrono, y este cambiará de estado en función de los valores que tomen dichas entradas y durante todo el tiempo en que la señal de reloj esté a 1 lógico.

Hay dos tipos de biestables disparados por nivel. Aquellos que son disparados por nivel alto y los que lo son por nivel bajo. En nuestro ejemplo, el biestable es disparado por nivel alto. Fácilmente podríamos haber construido el disparado por nivel bajo, sin más que añadir a la estructura mostrada un inversor en la línea de la señal de reloj.

Un diseño alternativo del biestable disparado por nivel es el que se muestra en la figura 6.11 , el cual ha sido construido a partir del SR-NAND. En este caso, las puertas AND han sido substituidas por puertas NAND.

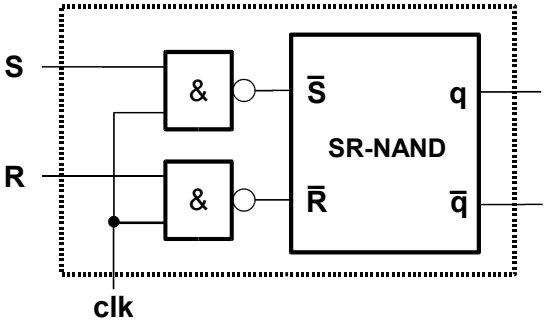


Figura 6.11.

En el caso en que $clk = 0$, las salidas de las puertas NAND son 1, por lo que el biestable asíncrono no cambia de estado. Si $clk = 1$, las entradas SR pasan a controlar las entradas del biestable asíncrono de forma invertida. Por tanto si $SR = 10$, las entradas del biestable asíncrono son 01, por lo que se genera la puesta a 1; si $SR = 01$, las entradas del biestable asíncrono valen 10, por lo que se genera la puesta a 0. En general, podemos decir que, independientemente de la realización interna del biestable SR asíncrono (NAND o NOR), el símbolo y la funcionalidad del biestable SR disparado por nivel son los mismos.

En la figura 6.12 se muestran los simbolos asociados al biestable SR disparado por nivel alto, (a), y al disparado por nivel bajo, (b):

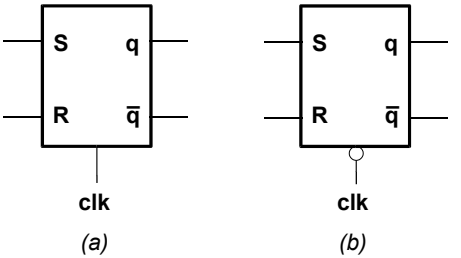


Figura 6.12.

6.2.3.2. Biestable *Master-Slave* (Amo-Esclavo)

A veces resulta insuficiente, para determinadas aplicaciones, el uso de los biestables disparados por nivel. En muchas ocasiones es interesante que los cambios del biestable se realicen en instantes muy determinados de la señal de reloj, en concreto, en los flancos de subida o bajada de la misma. Una primera solución a esta situación la presenta el biestable *Master-Slave*, en el apartado siguiente estudiaremos una segunda solución, plasmada en lo que se denominan biestables disparados por flanco.

En la figura 6.13 se representa la estructura de un biestable *Master-Slave* del tipo SR. Puede apreciarse que este biestable está formado, internamente, por dos biestables SR disparados por nivel. El situado a la izquierda de la página es el biestable amo (*Master*), cuyas entradas son las entradas SR del conjunto *Master-Slave*. Las salidas del biestable esclavo (*Slave*) son las salidas del conjunto *Master-Slave* y tiene como entradas las salidas del biestable amo. Obsérvese que el biestable amo está disparado por nivel alto, mientras que el segundo biestable, el esclavo, está disparado por nivel bajo. Otra configuración alternativa a la mostrada sería aquella en la que el biestable amo es disparado por nivel bajo y el esclavo por nivel alto. En definitiva, para un biestable *Master-Slave*, en cada semiciclo de la señal de reloj, sólo uno de los dos biestables puede estar activo, mientras que en el semiciclo siguiente, sólo se encontrará activo aquel biestable que en el primer semiciclo estaba inactivo.

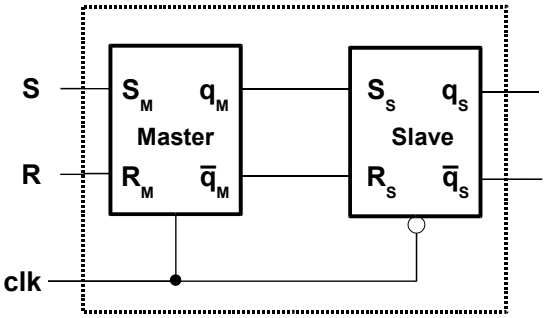


Figura 6.13.

Siguiendo la estructura definida en la figura anterior, podemos apreciar que el biestable amo está activo cuando la señal de reloj está en alto, mientras que el biestable esclavo se activa cuando la señal de reloj está en bajo.

Por tanto, si clk toma un 1 lógico, el biestable esclavo está inactivo, y sus salidas q, $\bar{q}$ no cambian de valor durante todo este semiciclo, independientemente de los cambios que se produzcan en las entradas $S_S R_S$. En cambio, el biestable Amo está activado, y las salidas de q, $\bar{q}$ para este biestable pueden cambiar en función de los valores de las entradas $S_M R_M$ para este semiciclo.

Si clk toma el valor 0 lógico, el biestable amo ahora está inactivo. Sus salidas no cambian de valor, aunque cambien las entradas $S_M R_M$, y mantienen los valores que se alcanzaron al final del semiciclo en el que clk valía 1 lógico. Por otro lado, ahora las salidas del biestable esclavo puede cambiar de valor ya que este se encuentra habilitado. Los valores futuros de q, $\bar{q}$, dependen de las entradas $S_S R_S$ en este semiciclo. Estas entradas son las salidas del biestable amo, el cual está inactivo y por tanto se mantienen constantes mientras clk sea 0 lógico. Como la salida q del amo se conecta a la entrada S del esclavo y la salida $\bar{q}$ del amo a la entrada R del esclavo, entonces las únicas posibilidades para el esclavo son $S_S R_S = 10$ o $S_S R_S = 01$, esto es, escribir un 1 o escribir un 0, respectivamente.

Nótese que las salidas del biestable *Master-Slave* pueden cambiar desde el instante en que el biestable esclavo comienza a estar activo, esto es, en la transición de reloj de 1 lógico a 0 lógico (flanco de bajada), y no vuelve a cambiar de valor hasta que llegue una nueva transición de bajada.

En la figura 6.14 se muestra la evolución temporal de las salidas de los biestables del conjunto *Master-Slave* para una secuencia de entradas determinada. Partimos de la situación en la que los dos biestables del conjunto *Master-Slave* están tienen un 0 lógico.

En el intervalo de tiempo $[0, t_1]$, clk tiene un 0 lógico, por tanto biestable amo inactivo, biestable esclavo activo. El biestable amo, tiene su salida q igual a 0 lógico, por tanto, en este intervalo, esta salida se mantiene con este valor a pesar de que la entrada S se pone a 1 lógico. El biestable esclavo está activo y tiene como entrada S la salida q del amo y como entrada R la salida $\bar{q}$ del amo, por tanto, se escribe un 0 lógico en el biestable esclavo que dura todo este tiempo.

- Intervalo $[t_1, t_2]$. Biestable esclavo inactivo, por tanto mantiene el 0 lógico del intervalo anterior y biestable amo activo (a continuación sólo nos referiremos al biestable amo). Durante este intervalo, R permanece a 0 mientras que S toma el valor 1, pasa a 0, y vuelve a ser 1 lógico. Por

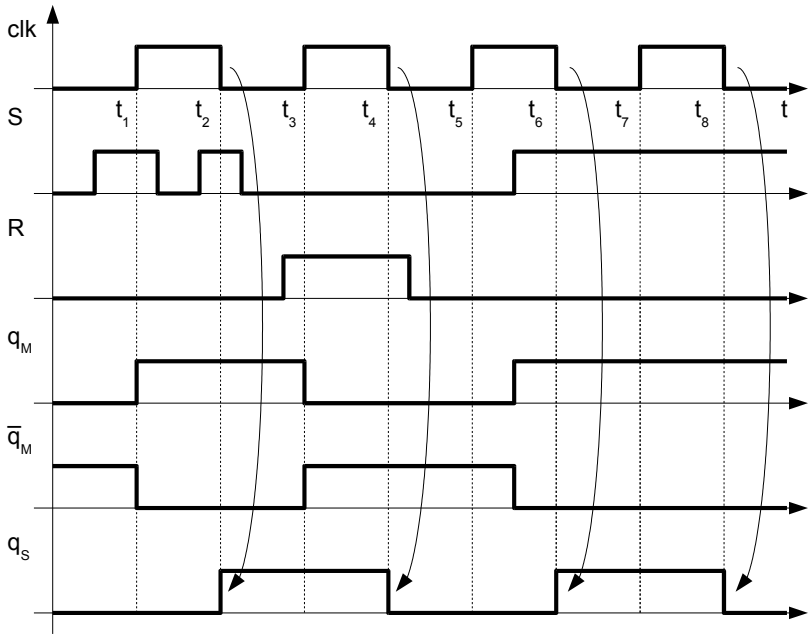


Figura 6.14.

tanto las entradas de este biestable son $SR = 10$, que pone la salida del biestable a 1, $SR = 00$, que mantiene el valor almacenado, y $SR = 10$, que vuelve a poner la salida q del biestable a 1. Entonces, durante este semiciclo, la salida q permanece a 1 mientras que $\bar{q}$ lo hace a 0.

- Intervalo $[t_2, t_3]$. Biestable amo inactivo, su salida q se mantiene a 1 lógico; biestable esclavo activo. Este último pone los valores de salida que se corresponden con las entradas $SR = 10$, esto es un 1 lógico.

- Intervalo $[t_3, t_4]$. Biestable esclavo inactivo, su salida q se mantiene a 1 lógico, y biestable amo activo. Durante este intervalo, las entradas SR del biestable amo toman los valores 01 respectivamente, por tanto la salida q del biestable amo se pone a 0 lógico durante todo este intervalo.

- Intervalo $[t_4, t_5]$. Biestable amo inactivo, su salida q se mantiene a 0 lógico, biestable esclavo activo. Las entradas SR de este último son ahora 01, respectivamente, por lo que la salida q del esclavo se pone a 0 lógico.

- Intervalo $[t_5, t_6]$. Biestable esclavo inactivo, su salida q se mantiene a 0 lógico, biestable amo activo. Durante este intervalo R vale 0 y S pasa de valer 0 a tomar el valor 1 lógico. Las entradas SR del amo en este intervalo son, primero, 00, y después 10. Para $SR = 00$, el biestable amo mantiene el valor almacenado, esto es, un 0 lógico (valor del intervalo $[t_4, t_5]$), mientras que para $SR = 10$, la salida q del biestable amo pasa a ser 1 lógico.

- Intervalo $[t_6, t_7]$. Biestable amo inactivo, mantiene su salida q a 1 lógico, biestable esclavo activo. En este caso, las entradas SR del esclavo son 10, por tanto, la salida q del esclavo se pone a 1 lógico.

- Intervalo $[t_7, t_8]$. Biestable esclavo inactivo, mantiene su salida q a 1 lógico, biestable amo activo. Para este intervalo S toma el valor 1, mientras que R el 0, por tanto se escribe un 1 lógico.

- Intervalo $[t_8, \ldots]$. Biestable amo inactivo, mantiene su salida a 1 lógico, biestable esclavo activo. La salida q del esclavo se pone a l lógico.

Como se puede apreciar en la figura anterior, la salida del *Master-Slave*, Q_S, sólo cambia en los flancos de bajada de la señal de reloj, manteniéndose constante durante el resto del periodo de reloj. Podíamos haber diseñado un *Master-Slave* cuya salida cambiase en los flancos de subida del reloj. Para ello

el biestable amo debería ser disparado por nivel bajo y el esclavo por nivel alto.

En la figura 6.15 se representan los símbolos de los biestables *Master-Slave* con cambio de salida en flanco de subida, (a), y en flanco de bajada, (b):

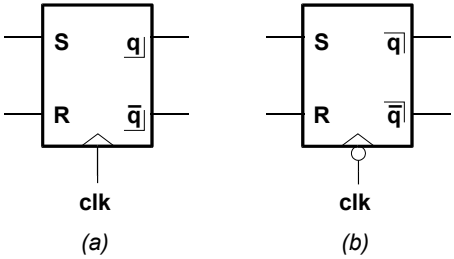


Figura 6.15.

6.2.3.3. Biestable disparado por flanco

Este tipo de biestable presenta un modo de funcionamiento similar al biestable *Master-Slave*, en el sentido de que la salida q del biestable sólo puede cambiar en los flancos de subida o bajada de la señal de reloj, permaneciendo estable el resto del periodo. En la figura 6.16 se representa los símbolos lógicos de un biestable SR disparado por flanco de subida, (a), o por flanco de bajada, (b).

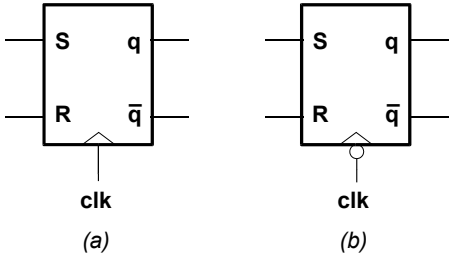


Figura 6.16.

La principal diferencia con el funcionamiento del *Master-Slave* estriba en que en el momemto del flanco activo (bajada o subida) se leen las entradas (en este caso SR) que determinarán el próximo valor de q para el ciclo de reloj

siguiente. Aclaremos el funcionamiento del biestable con el ejemplo ilustrado en el cronograma de la figura 6.17. En él se ha representado la evolución de las salida q de un biestable SR disparado por flanco de bajada (o negativo) para una secuencia de entradas determinada. Suponemos que el valor inicial, para $t = 0$, del biestable SR es un 1 lógico.

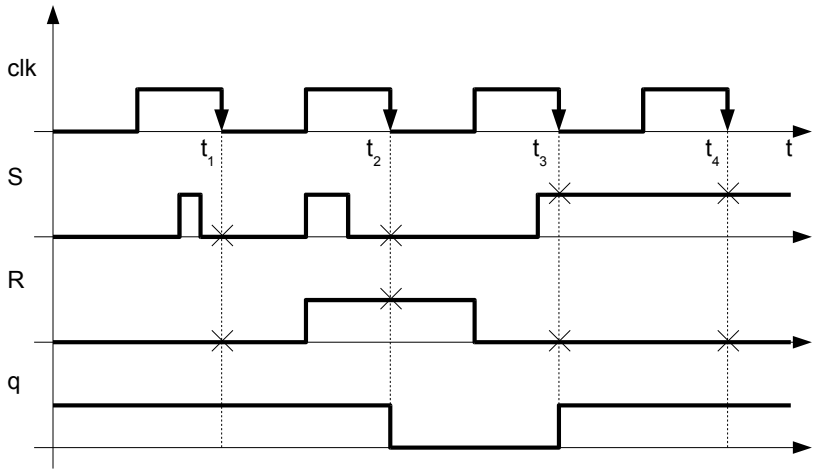


Figura 6.17.

- Intervalo $[0, t_1]$. No se genera ningún flanco de bajada durante este periodo de reloj, por lo que la salida q se mantiene a 1 lógico.

- Instante t_1. Se genera un flanco de bajada. Las entradas SR para $t = t_1$ son 00 respectivamente. Por tanto el periodo siguiente de reloj, el biestable mantiene el 1 lógico.

- Instante t_2. Se genera flanco de bajada. Las entradas SR en este instante son 01, respectivamente, por lo que la salida q se pone a 0 lógico. Este 0 lógico se mantiene hasta t_3.

- Instante t_3. Las entradas SR son 10, la salida q del biestable se pone a 1 lógico hasta que se genere un nuevo flanco de bajada.

- Instante t_4. Las entradas SR son 10, la salida q del biestable sigue a 1 lógico.

Como puede verse en el cronograma anterior, las entradas SR pueden estar a 1 lógico simultáneamente (intervalo $[t_1, t_2]$). En este tipo de biestable estas entradas están permitidas mientras que no se encuentren en el momento que se genere un flanco activo.

Para incidir más en las diferencias funcionales del biestable disparado por flanco y el biestable *Master-Slave*, se ha representado, en la figura 6.18, un cronograma que muestra la evolución de las salidas de estos biestables para una secuencia de entradas de SR determinada. Suponemos que, inicialmente, el estado de ambos biestables es desconocido.

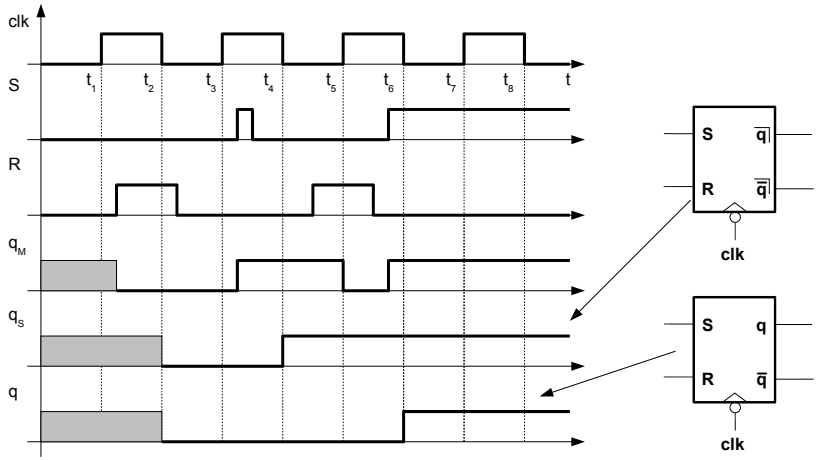


Figura 6.18.

Para el biestable disparado por flanco:

- Inicio. El valor de q es desconocido hasta que llegue el primer flanco de bajada de la señal de reloj.
- Instante t_2. Las entradas SR para t_2 son 01 respectivamente. El biestable pone su salida q a 0 lógico hasta el instante t_4.
- Instante t_4. Las entradas SR son 00. El biestable mantiene el valor anterior de q, esto es, 0 lógico, hasta el siguiente flanco.
- Instante t_6. Las entradas SR son 10. El biestable pone su salida q a 1 lógico hasta el siguiente flanco.

- Instante t_8. Las entradas SR son 10. El biestable pone su salida q a 1 lógico hasta el siguiente flanco.

Para el biestable *Master-Slave*:

- Intervalo $[0, t_1]$. Biestable Amo inactivo, Esclavo activo. Como la salida q del amo es desconocida, y este esta inactivo, mantiene este valor desconocido durante todo este intervalo. El biestable esclavo está activo, pero como las entradas son desconocidas, el valor de salida del mismo también es desconocido.

- Intervalo $[t_1, t_2]$. Biestable Esclavo inactivo, mantiene el valor desconocido, biestable amo activo. Durante este intervalo las entradas del amo son $SR = 00$ y $SR = 01$. Para el primer caso, $SR = 00$, el biestable amo mantiene el estado desconocido, mientras que para $SR = 01$, la salida q pasa a ser 0 lógico.

- Intervalo $[t_2, t_3]$. Biestable amo inactivo, mantiene su salida q a 0 lógico, biestable esclavo activo. Las entradas de este último son $SR = 01$, por tanto, pone su salida q a 0 lógico durante este intervalo.

- Intervalo $[t_3, t_4]$. Biestable esclavo inactivo, mantiene su salida q a 0 lógico, biestable amo activo. Durante este periodo, las entradas son $SR = 00$, después $SR = 10$, y finalmente $SR = 00$. Para el primer caso, $SR = 00$, el biestable mantiene el dato almacenado, o sea, 0, para $SR = 10$, el biestable amo pone su salida q a 1 lógico, y para $SR = 00$, mantiene el 1 lógico.

- Intervalo $[t_4, t_5]$. Biestable amo inactivo, mantiene su salida q a 1 lógico, biestable esclavo activo. La salida q de este último se pone a 1 lógico.

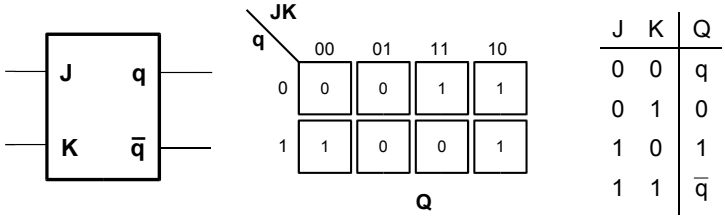
- Intervalo $[t_5, t_6]$. Biestable esclavo inactivo, mantiene su salida q a 1 lógico, biestable amo activo. Durante este intervalo las entradas del amo pasan de ser $SR = 01$, a $SR = 00$ y finalmente $SR = 10$. Para el primer caso, $SR = 01$, la salida q del amo se pone a 0 lógico; para el segundo caso, $SR = 00$, la salida $q = 0$ se mantiene, y para $SR = 10$, la salida q vuelve a ponerse a 1 lógico.

- Intervalo $[t_6, t_7]$. Biestable amo inactivo, mantiene su salida q a 1 lógico, biestable esclavo activo. La salida q del esclavo se pone a 1 lógico.

Como puede apreciarse en las formas de onda de salida de ambos biestables, aunque estos funcionan de forman similar, sus resultados no son idénticos. Es fácil apreciar que el pulso de *Set* del intervalo $[t_3, t_4]$ es totalmente transparente para el biestable disparado por flanco, pero no para el *Master-Slave*, ya que este pulso es capturado por el biestable amo y transferido al esclavo.

6.2.4. Otros biestables

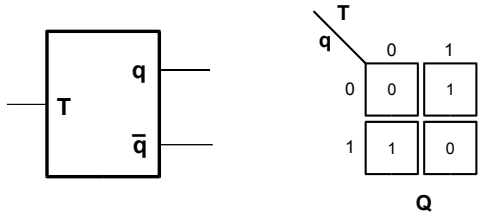
Biestable JK. (figura 6.19) Es similar al biestable SR, pero tiene la ventaja de que la entrada $JK = 11$ no está prohibida (la entrada J es similar a la S y la K a la R).



J	K	Q
0	0	q
0	1	0
1	0	1
1	1	$\overline{q}$

Figura 6.19.

Biestable T. (figura 6.20) Es igual al JK cuando unimos sus dos entradas.



T	Q
0	q
1	$\overline{q}$

Figura 6.20.

Biestable D. (figura 6.21) Almacena el bit que llega por su única entrada D.

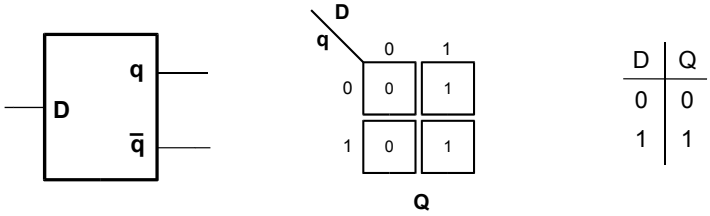


Figura 6.21.

6.2.5. Realización de biestables a partir de otros

En este apartado daremos las nociones generales para construir un biestable cualquiera a partir de uno dado. Haremos dos ejemplos.

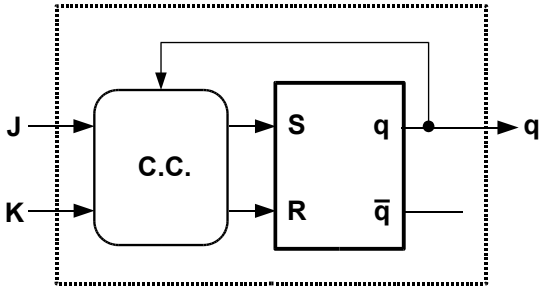
Ejemplo 1. Construir un biestable JK a partir de un SR.

Esto es, disponemos de un biestable SR y queremos construir una estructura de forma que tenga dos terminales que hagan las veces de las entradas JK, y una salida q que cambie en función de JK tal como define la tabla de transición del biestable JK.

Usaremos el biestable SR como elemento de memoria, naturalmente, y la idea de diseño se base en encontrar los valores de SR adecuados para facilitar las transiciones definidas en el biestable JK en función de estas entradas, J y K. Partimos de la estructura representada en la figura 6.22. Deseamos diseñar un circuito combinacional (CC) que, en función de las entradas J, K, y el valor del biestable, q, determine las entradas SR adecuadas para que el conjunto opere como un biestable JK.

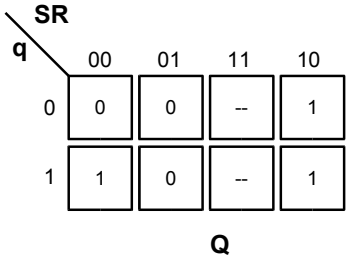
Para ello, partimos de la tabla de transición del biestable SR, de la cual se obtiene la tabla de excitación, figura 6.23. Esta nueva tabla muestra qué valores de entrada SR son necesarios para generar las cuatro transiciones posibles en todo biestable: de 0 a 0, de 0 a 1, de 1 a 0 y de 1 a 1.

A continuación determinamos la estructura del CC a partir de un K-mapa que defina su funcionamiento. Si $JK = 00$ y $q = 0$, el biestable JK a diseñar tiene que generar como estado futuro $Q = 0$, por tanto, el CC que tiene como entradas $JK = 00$ y $q = 0$ debe generar las salidas SR para que el biestable SR pase de 0, valor presente, a 0 como valor futuro. Esto es, según la tabla de



Biestable JK

Figura 6.22.



$q \rightarrow Q$	S	R
$0 \rightarrow 0$	0	x
$0 \rightarrow 1$	1	0
$1 \rightarrow 0$	0	1
$1 \rightarrow 1$	x	0

Figura 6.23.

excitación, $SR = 0x$.

Si $JK = 11$ Y $q = 0$, el biestable JK a diseñar tiene que generar comoe stado futuro $Q = 1$, por tanto, el CC debe generar las salidas SR apropiadas para que el biestable SR pase de valor 0, presente, a 1, valor futuro. La tabla de excitación nos indica que estos valores son $SR = 10$.

Si iteramos este procedimiento para las restantes casillas obtenemos el K-mapa de la figura 6.24 para el CC.

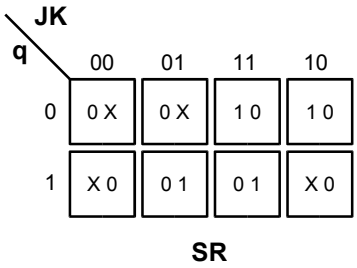


Figura 6.24.

Las expresiones para S y R son:

$$\begin{aligned} S &= J\,\bar{q} \\ R &= K\,q \end{aligned}$$

El cicuito resultante es el representado en la figura 6.25.

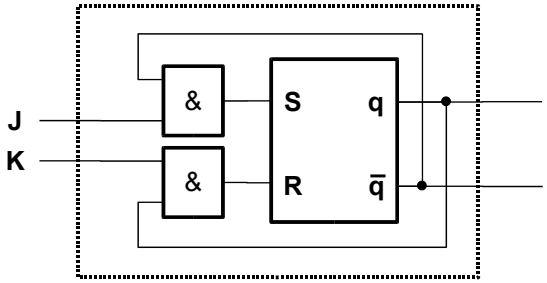


Figura 6.25.

Ejemplo 2. Obtener el biestable SR a partir de un T.

Se parte de la estructura mostrada en la figura 6.26, y se repiten los pasos anteriores.

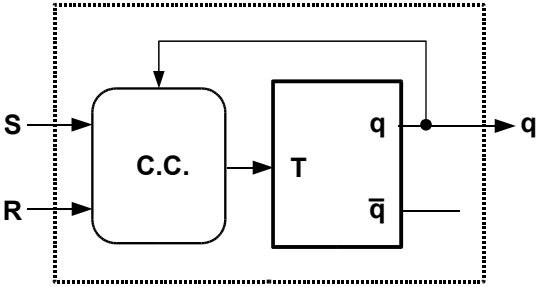
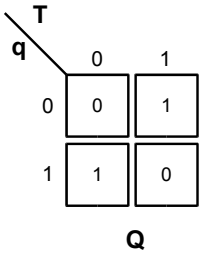


Figura 6.26.

En primer lugar, se obtiene la tabla de excitación del biestable T, figura6.27.



q → Q	T
0 → 0	0
0 → 1	1
1 → 0	1
1 → 1	0

Figura 6.27.

A continuación determinamos la estructura del CC a partir de un K-mapa que defina su funcionamiento. Para $SR = 00$ y $q = 0$, el biestable SR debe tener como valor futuro $Q = 0$, por tanto el CC que recibe dichas entradas debe generar como salida $T = 0$, para que el biestable pase de estado presente 0 a estado futuro 0. Si $SR = 01$ y $q = 1$, el estado futuro del SR es 0, por tanto el biestable T debe de pasar de estado actual 1 a estado 0, por lo que el CC debe generar salida $T = 1$ para estas entradas. Si procedemos para todas las posibles combinaciones de SR y teniendo en cuenta que $SR = 11$ están

inespecificadas para el CC, obtenermos el K-mapa de la figura 6.28.

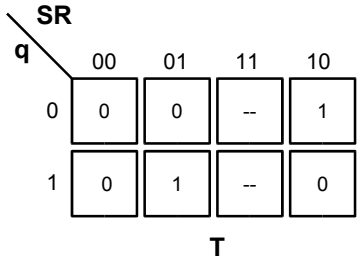


Figura 6.28.

De donde obtenemos que:

$$T \;=\; R\,q \;+\; S\,\bar{q}$$

6.2.6. Entradas asíncronas de los biestables

Son unas entradas adicionales que disponen algunos biestables síncronos. Estas entradas permiten la puesta a 1 o a 0 del biestable síncrono sin necesidad de esperar a la llegada del nivel activo o el flanco activo de la señal de reloj. Estas son las entradas de *Preset*, Pr, y *Clear*, Cl. La primera sirve para poner a 1 la salida q del biestable y la segunda para ponerla a 0. Estas entradas pueden ser activas en bajo o en alto. En la figura 6.29 se ha representado un biestable T disparado por flanco de bajada con enetradas asíncronas Pr y Cl activas en bajo.

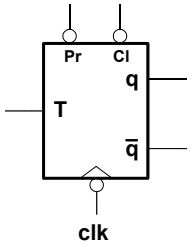


Figura 6.29.

Cuando las entradas de Pr y Cl están a 1 lógico, el biestable funciona de forma normal a la descrita en apartados anteriores, las salida q cambiará en función de los valores que tome la entrada T en los flancos de bajada de clk. Si $Pr = 0$, y $Cl = 1$, se produce una puesta a 1 asíncrona, esto es, la salida q del biestable se pone a 1 lógico desde el momento en que $Pr = 0$, independientemente de la entrada de reloj y la entrada T. Además esta salida se mantiene mientras que Pr siga valiendo 0. Si $Cl = 0$ y $Pr = 1$, ocurre lo mismo que lo descrito anteriormente pero la salida q se pone a 0. Las entradas $Cl = Pr = 0$ se prohíben.

En la figura 6.30 se ha representado la evolución temporal para el biestable anterior para una secuencia de entradas de Pr, Cl y T. En este dibujo no se ha representado la señal periódica de reloj, simplemente se han simbolizado los flancos activos con las líneas punteadas verticales.

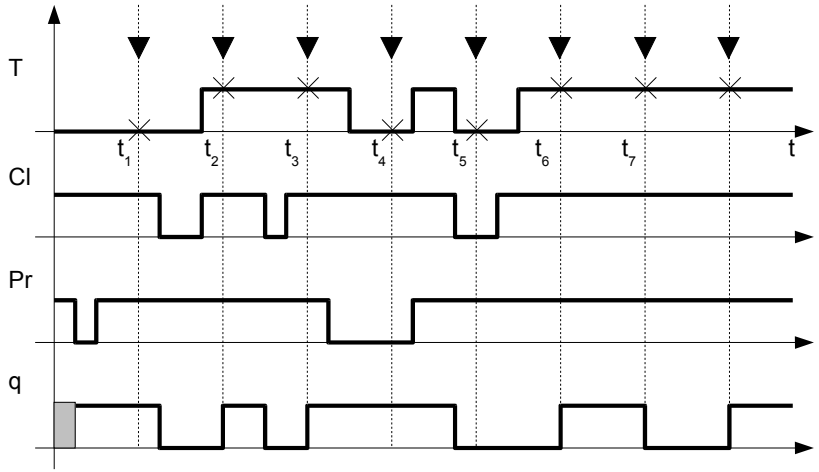


Figura 6.30.

Inicialmente el estado del biestable es desconocido.

- Intervalo $[0, t_1]$. Hasta el instante A, el valor de q es desconocido. En este instante, se produce un pulso de *Preset*, que pone la salida q a 1 lógico. Esto se mantiene hasta t_1, donde llega un flanco activo. En el instante t_1, ninguna entrada asíncrona está activa, y la entrada T, que está a 0 lógico determina que el próximo valor de q será el 1 lógico.

- Intervalo $[t_1, t_2]$. Se produce un pulso de *Clear*, que pone la salida q a 0. Este 0 lógico se mantiene hasta t_2, instante en el que llega un flanco de bajada de clk. Como la entrada T en este instante está a 1, y el valor de q actual es 0, el próximo valor de q será un 1 lógico.

- Intervalo $[t_2, t_3]$. Se produce un pulso de *Clear*, que nuevamente pone la salida q a 0 lógico hasta que llegue el siguiente flanco de reloj. En t_3, la entrada T vale 1, y el q actual es 0, por lo que el q próximo es 1.

- Intervalo $[t_3, t_4]$. Existe un pulso de *Preset* que deja la q a 1 lógico. En t_4, existe un flanco activo, pero el *Preset* sigue a 0. Las entradas asíncronas tienen prioridad sobre T, por lo que sigue la salida a 1.

- Intervalo $[t_4, t_5]$. La salida se mantiene a 1 hasta el punto D, donde se genera un pulso de *Clear*, que pone la salida q a 0 lógico. Este pulso de *Clear* coincide con el flanco activo que llega en el instante t_5, por lo que este no tiene efecto.

- Intervalo $[t_5, t_6]$. La salida q se mantiene a 0 durante este intervalo. A partir de aquí, ninguna entrada asíncrona se activa, por lo que los cambios de q vienen determinados por los valores de T en los flancos activos. En t_6 como $T = 1$ y $q = 0$, el próximo valor es $q = 1$.

- Intervalo $[t_6, t_7]$. La salida q se mantiene a 1 durante este intervalo. En t_7 llega un flanco activo que provoca que para $T = 1$ y $q = 1$, el próximo valor de q sea 0.

6.3. Análisis de circuitos secuenciales síncronos

6.3.1. Autómatas de Mealy y Moore

Vimos en la introducción del capítulo el esquema general de un circuito secuencial o máquina secuencial, constituido por un bloque combinacional y un conjunto de biestables, que conforman los elementos de memoria. A partir de ahora nos centraremos en un tipo determinado de circuito secuencial: la **máquina secuencial síncrona**. Este tipo de máquina cumple las siguientes condiciones:

- Todos los biestables son del tipo disparado por flanco y todos son o de subida o de bajada
- Todos los biestables reciben la misma señal de reloj

Con esto conseguimos que todos los elementos de memoria de la máquina secuencial cambien simultáneamente. Aquí aparece el concepto de **estado** de una máquina secuencial. Éste se puede definir como el *conjunto de valores almacenados en los biestables durante un ciclo de reloj determinado*. Así, una máquina secuencial que tenga un biestable, tiene dos estados posibles, representados por los valores 0 y 1 que pueden ser almacenados en dicho biestable. Si la máquina secuencial tiene dos biestables, existen 4 estados distintos, correspondientes a las cuatro combinaciones posibles de valores almacenables en los dos biestables, 00, 01, 10 y 11. En general, si disponemos de n biestables, la máquina tiene un total de 2^n estados posibles distintos. En una máquina o circuito secuencial síncrono, por tanto, el paso de un estado a otro sólo se puede producir al llegar el flanco de disparo de los biestables por la señal de reloj.

Existen dos tipos de circuitos secuenciales síncronos (en adelante, CSS):

Máquina o autómata de *Moore*. Todo aquel CSS cuyas salidas sólo dependen del estado actual, es decir, de los valores almacenados en los biestables.

$$Z_k = Z_k(q_1, q_2, \ldots)$$

Por tanto, las salidas de una máquina de *Moore* cambian de valor cuando lo hacen las salidas q_i de los biestables, esto es, en los flancos *activos* (ascendentes o descendentes, según el tipo de disparo de reloj de los biestables), mientras que en el resto del ciclo permanecen constantes. La estructura en bloques de una máquina de Moore se muestra en la figura 6.31.

Máquina o autómata de *Mealy*. Todo aquel CSS cuyas salidas sean función de los estados de los biestables y las entradas de la máquina.

$$Z_k = Z_k(x_1, x_2, \ldots, q_1, q_2, \ldots)$$

Por tanto, las salidas de una máquina de *Mealy* pueden cambiar cuando lo hagan las salidas q_i de los biestables o cuando lo hagan las entradas

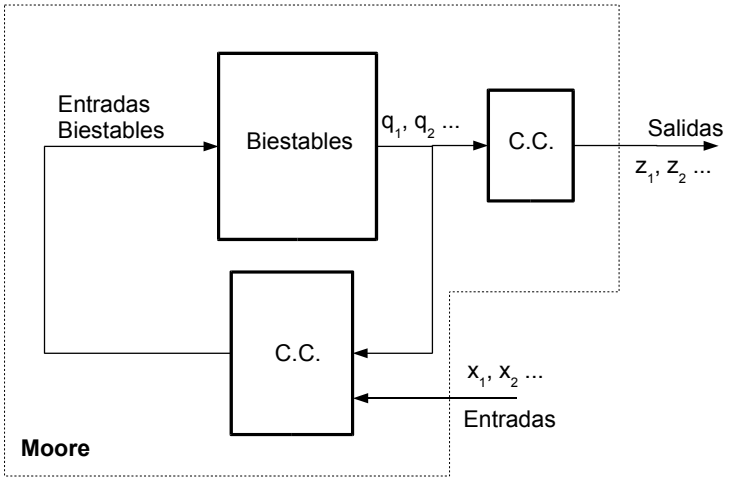


Figura 6.31.

al circuito. Como estas últimas pueden cambiar en cualquier instante, sin depender de la señal de reloj, las salidas de este autómata pueden hacer lo propio en cualquier momento. La estructura en bloques de una máquina de Moore se muestra en la figura 6.32.

6.3.2. Análisis de circuitos secuenciales síncronos

El proceso de análisis de CSS, de forma similar al análisis de circuitos combinacionales, persigue obtener una descripción total del mismo. Si recordamos la estructura de un CSS, veremos que por un lado tenemos un conjunto de salidas que dependen, en el caso más general, de las entradas y los estados de la máquina, y por otro lado tenemos un conjunto de biestables cuyo estado puede cambiar en función del estado actual y de las entradas.

El proceso de análisis pretende obtener una descripción completa a través de una tabla o diagrama de estados, donde se especifiquen los valores de salida y la evolución de estados en función de las entradas y estados presentes. Dicho proceso se va realizar en los siguientes pasos:

(a) **Obtención de las ecuaciones de salida y ecuaciones de excitación**. A partir del esquema del circuito se obtienen las expresiones booleanas de

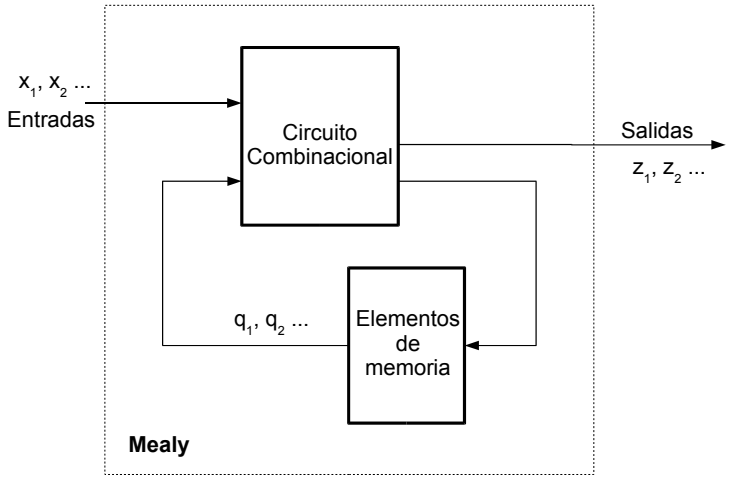


Figura 6.32.

las salidas del mismo y de las entradas de cada uno de los biestables que lo constituyen (ecuaciones de excitación).

(b) **Tabla de excitación y salida**. La tabla de excitación y salida es la representación en forma de K-mapa de las ecuaciones de excitación y salida.

(c) **Tabla de transición**. Es la representación en K-mapa de los próximos valores, Q, que toman cada uno de los biestables del circuito en función de los valores actuales, q, y de las entradas del circuito.

(d) **Tabla de estados / Diagrama de estados**. La tabla de estados se obtiene a partir de la tabla de transición sin más que asignarle un nombre a cada conjunto de valores de los biestables. El diagrama de estados es una representación gráfica de la tabla de estados en los que éstos son representados como círculos, y los cambios de estado o transiciones, como flechas que unen los círculos. La notación usada se muestra en la figura 6.33.

Ejemplo. Analizar el circuito de la figura 6.34.

Se trata de un CSS, ya que cumple los requisitos reseñados anteriormente, con dos biestables SR disparados por flanco de bajada, una entrada x y una salida Z. El circuito posee, pues, un total de cuatro estados distintos. Además

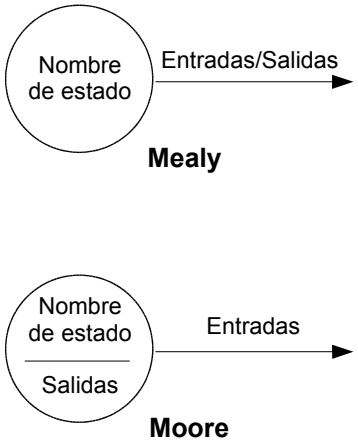


Figura 6.33.

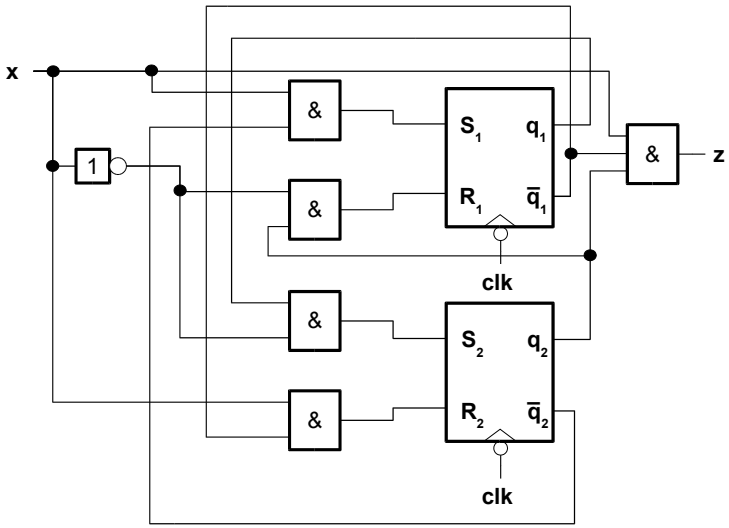


Figura 6.34.

se trata de una máquina de *Mealy*, porque la salida z es función de la entrada x y de los valores q de los biestables (si fuese de *Moore* sólo dependería de los valores q).

Ecuación de salida:

$$z = x\,\overline{q_1}\,q_2$$

Ecuaciones de excitación:

$$\begin{aligned} S_1 &= x\,\overline{q_2} \\ R_1 &= \overline{x}\,q_2 \\ S_2 &= \overline{x}\,q_1 \\ R_2 &= x\,\overline{q_1} \end{aligned}$$

Tabla de excitación (figura 6.35). Se representan en un K-mapa las ecuaciones de excitación anteriores, procurando colocar en vertical los valores almacenados en los biestables, y en horizontal, las entradas.

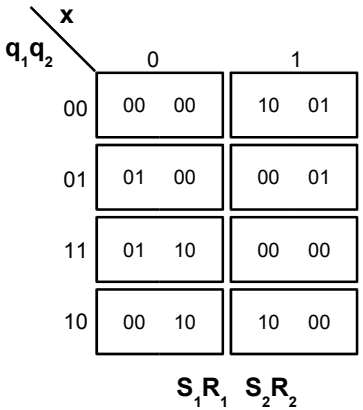


Figura 6.35.

Tabla de salida (figura 6.36). Se representa la ecuación de salida en un K-mapa siguiendo los criterios de la tabla de excitación.

Tabla de transición (figura 6.37). Éste es el paso más complejo de todos los del análisis, pues hay que combinar el funcionamiento de cada biestable del

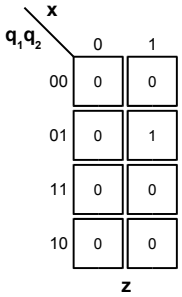


Figura 6.36.

circuito con la tabla de excitación. Por ejemplo, para $q_1q_2 = 00$ y $x = 0$ las entradas del primer biestable son $S_1R_1 = 00$, esto implica que el valor futuro Q_1 será el mismo que el presente q_1, o sea, 0. De igual forma, las entradas S_2R_2 para el segundo biestable son 00, por lo que el valor Q_2 es el mismo que q_2, o sea, 0. Si $q_1q_2 = 11$ y $x = 0$, las entradas a los biestables son $S_1R_1 = 01$ y $S_2R_2 = 10$, por lo que el primer biestable se pone su proximo valor $Q_1 = 0$ y el segundo biestable $Q_2 = 1$. Este proceso debe repetirse para todos los biestables.

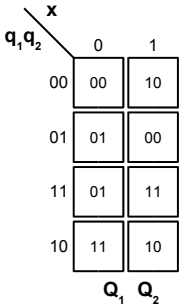


Figura 6.37.

Tabla de estados/salidas (figura 6.38). A cada uno de los valores de q_1q_2 le asignamos un nombre. Por ejemplo a $q_1q_2 = 00$, A; $q_1q_2 = 01$, B, $q_1q_2 = 10$, C y $q_1q_2 = 11$, D. Al aplicar esta asignación de nombres a la tabla de transición y de estados, se obtiene la tabla de estados/salidas.

Diagrama de estados (figura 6.39). Este punto es opcional. Simplemente se translada la tabla de estados a una representación gráfica.

S \ x	0	1
A	A, 0	C, 0
B	B, 0	A, 1
D	B, 0	D, 0
C	D, 0	C, 0

NS, z

Figura 6.38.

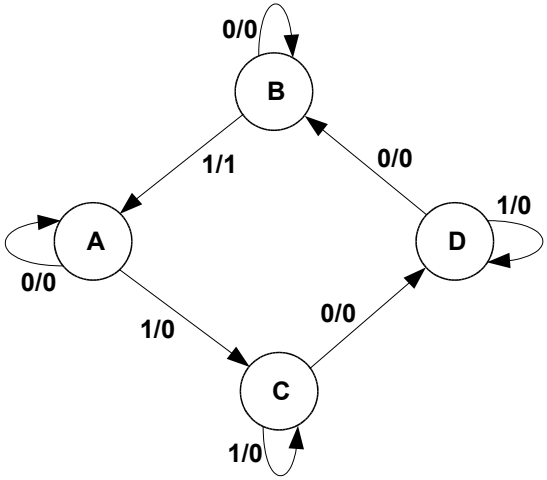


Figura 6.39.

6.4. Síntesis de circuitos secuenciales síncronos

6.4.1. Proceso de síntesis

El proceso de síntesis o diseño de un circuito secuencial síncrono conlleva un recorrido inverso al de análisis. Esto es, a partir de la especificación formal del funcionamiento de la máquina secuencial, obtendremos, en primer lugar, el diagrama de estados o tabla de estados que modele su funcionamiento y, a partir de aquí, para obtener el circuito secuencial, se realizarán los pasos correspondientes hasta obtener las ecuaciones de excitación y salida. Estos pasos a seguir quedan resumidos en el siguiente esquema.

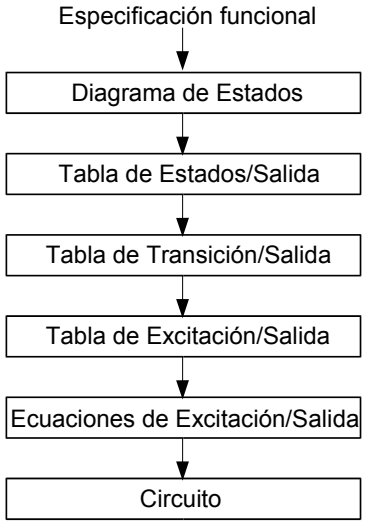


Figura 6.40.

El paso más difícil y sobre el que no hay metodología a aplicar es el primero, obtener un diagrama de estados que modele el funcionamiento de la máquina secuencial. Una vez obtenido éste, los siguientes pasos son más o menos "mecánicos". No obstante, en el proceso de diseño, es deseable que el circuito resultante tenga un coste reducido. Esto implica que tendremos que aplicar algunos métodos de simplificación que permitan obtener un circuito óptimo. Así, cuando se tenga la tabla de estados (paso 2), aplicaremos un método de simplificación que busque la minimizacíon del número de esta-

dos y por consiguiente la reducción del número de biestables que emplee la máquina secuencial y, además, se dictarán las reglas necesarias para obtener una buena asignación de la tabla de estados, con el objeto de que la tabla de transición y tabla de excitación resultantes provoquen que las ecuaciones de excitación contengan el menor número de literales posibles (paso 3). En primer lugar se realizarán un par de ejemplos, en los que el objetivo principal es iniciar al lector a la obtención de los diagramas de flujo. Más adelante se estudiarán las técnicas de simplificación necesarias para la obtención del circuito secuencial óptimo.

Ejemplo 1. Se pide diseñar un circuito secuencial síncrono que genere periódicamente la secuencia $0, 1, 1, 1$.

Este circuito no dispone de ninguna entrada de datos (salvo el reloj que es necesario en todos los CSS), por lo que se trata de un autómata de *Moore*. En segundo lugar, cada bit de la salida tiene la duración de un ciclo de reloj, esto es, en el primer ciclo de reloj, la salida $Z = 0$, en el siguiente, $Z = 1$, en el siguiente, $Z = 1$, en el siguiente, $Z = 1$, en el siguiente, $Z = 0$ (volviendo a empezar la secuencia), y así se repite la secuencia sucesivamente.

A continuación se desarrolla, paso a paso, el proceso de síntesis de este CSS.

(a) **Diagrama de estados**. Es fácil comprender que este CSS necesita de, al menos, cuatro estados distintos que se recorren de forma consecutiva, tal como se muestra en la figura 6.41. Cada estado permanece "activo" en un ciclo de reloj. Así en el primer ciclo de reloj, el estado activo es A, donde la salida generada es 0, en el siguiente ciclo de reloj, el estado activo es el B, cuya salida es 1; en el siguiente ciclo de reloj pasamos al estado C, cuya salida es, nuevamente, 1; en el siguiente ciclo de reloj se alcanza el estado D, que pone la salida Z a 1; en el siguiente ciclo de reloj pasamos de nuevo al estado A, cuya salida es $Z = 0$, etc. Podemos comprobar que el diagrama de estados modela de forma adecuada este generador de secuencia.

(b) **Tabla de estados/salida**. Se obtiene directamente a partir del diagrama anterior. La tabla de estados/salida está representada en la figura 6.42.

(c) **Tabla de transición**. A partir del diagrama de estados/salida, mediante una asignación cualquiera, obtenemos la tabla de transición, figura 6.43.

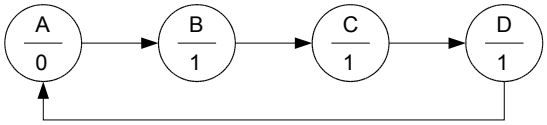


Figura 6.41.

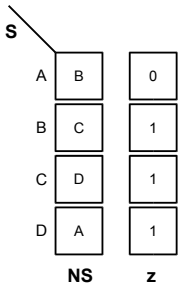


Figura 6.42.

Esta asignación debe contemplar el hecho de que al ser un una máquina de 4 estados, necesitamos dos biestables, q_1, q_2. Pues bien, el estado A será aquel en que q_1, q_2 valgan 00 respectivamente; el estado B, 01; el estado C, 10; y el estado D, 11.

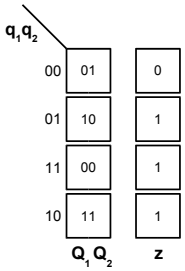


Figura 6.43.

(d) **Tabla de excitación/salida**. Para realizar este paso hay que elegir el tipo de biestable que se va a utilizar en el circuito. Supongamos que utilizamos biestables JK. La figura 6.44 muestra la tabla de transición del biestable JK:

La tabla de excitación se obtiene como se indica a continuación. Para la primera fila de la tabla de transición $q_1, q_2 = 00$, los próximos valores Q_1, Q_2 son 01 respectivamente. La transición efectuada por las salidas del primer biestable es $0 \rightarrow 0$, por lo que sus entradas deben ser $J_1, K_1 = 0x$, mientras que la del segundo es $0 \rightarrow 1$, por lo que sus entradas deben

q → Q	J	K
0 → 0	0	x
0 → 1	1	x
1 → 0	x	1
1 → 1	x	0

Figura 6.44.

ser $J_2, K_2 = 1x$. Repitiendo este proceso para el resto de la tabla, se obtendrá la tabla de excitación, figura 6.45.

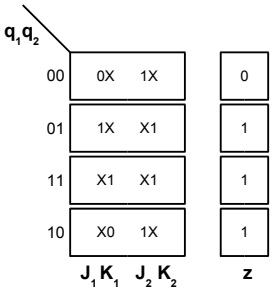


Figura 6.45.

(e) **Ecuaciones de excitación/salida**. A partir de la tabla de excitación/salida anterior, se obtienen las siguientes expresiones:

$$\begin{aligned} Z &= q_1 + q_2 \\ J_1 &= q_2 \\ K_1 &= q_2 \\ J_2 &= 1 \\ K_2 &= 1 \end{aligned}$$

(f) **Circuito**. Por último, se representa el circuito resultante, figura 6.46.

Ejemplo 2. Se desea diseñar un circuito secuencial síncrono que sea capaz de detectar la secuencia de entrada $1, 1, 1$.

El circuito que se ha de diseñar tiene una entrada, X, por donde se introducen una secuencia de bits y una salida, Z, que se activa cuando del conjunto de bits de entrada se ha detectado la secuencia $1, 1, 1$.

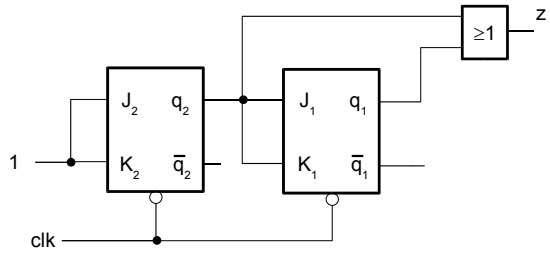


Figura 6.46.

Para resolver este problema (y similares) debemos hacer las siguientes consideraciones. La primera de ellas hace referencia a la duración de los bits de entrada y es de carácter general para todo este tipo de problemas. Estos bits deben tener la misma duración uno que otro, es decir, si el primer bit de entrada por X tiene una duración de t segundos, el siguiente y sucesivos deben tener también la misma duración. Además la duracíon t debe estar en concordancia con la capacidad o rapidez que tenga el circuito secuencial en procesar los bits de entrada. Como esta rapidez viene determinada por la frecuencia o periodo de la señal de reloj, la duración de los bits de entrada debe ser igual a este periodo. La segunda consideración es de carácter particular para los CSS que se diseñan para detectar secuencias de entradas, y consiste en si estos permiten solapamiento de secuencia o no. En la siguiente figura se ha ilustrado la situación de solapamiento. Se ha representado en el tiempo la evolución de una posible secuencia de entradas y, debajo, se ha representado la salida del circuito detector para el caso en que exista solapamiento (a) y para el caso en que no existe solapamiento (b):

	Tiempo →																				
X:	0	0	1	1	0	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	…
(a) Z:	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1	1	0	0	0 …
(b) Z:	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0 …

Se puede observar para el caso (a) y (b) que si se han recibido tres 1 consecutivos por la entrada X, la salida Z se pone a 1. Para el caso (a), en la ráfaga de 6 unos consecutivos se aprecia que la salida Z se mantiene a 1 desde el tercer 1 detectado hasta el final de la ráfaga. Esto es debido al solapamiento, en cuyo caso la salida se activa siempre que los tres últimos bits recibidos por el circuito han sido 1s. Para el caso (b), no existe solapamiento, puesto que una vez detectado la secuencia de tres unos, la máquina secuencial se inicializa para buscar una nueva secuencia de tres unos, al final de los cuales activa su salida Z.

El caso más habitual es el de considerar que existe solapamiento, por tanto resolveremos el problema incorporando este requisito.

Aquí se plantean dos alternativas importantes de diseño, realizar el circuito como un autómata de *Moore* o como uno de *Mealy*. En este primer ejemplo, resolveremos el circuito de ambas formas destacando las diferencias y ventajas de cada esquema.

(a) **Como autómata de *Moore***. El primer paso es obtener el diagrama de estados que modele el funcionamiento del detector de secuencia. En primer lugar buscaremos la "columna vertebral" del diagrama de estados, planteándo cuántos estados son necesarios y la funcionalidad de cada uno para, después, completar con las distintas combinaciones de entradas.

- Estado A: estado inicial donde se espera la recepción del primer 1 por la entrada X. Este estado "memoriza" que no se ha recibido ningún 1 y en él se genera la salida $Z = 0$.
- Estado B: estado que "memoriza" que se ha recibido un 1 y genera $Z = 0$.
- Estado C: Estado que "memoriza" que ya se han recibido dos 1s consecutivos por la entrada X y en el que se genera salida $Z = 0$.
- Estado D: estado que "memoriza" que los tres últimos bits recibidos son 1s. La salida generada en este estado es $Z = 1$.

La "columna vertebral" de este diagrama de estados se representa en la figura 6.47.

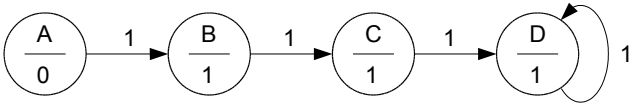


Figura 6.47.

Ahora se deben completar las transiciones que faltan. Por ejemplo, si en el estado A, se recibe un 0 por la entrada X, es lógico que el próximo estado sea el propio A. Si en el estado B, se recibe un 0, pasamos al estado A, lo mismo que si se recibe un 0 en el C o en D. El diagrama definitivo se muestra en la figura 6.48.

A partir del diagrama de estados, se obtiene la tabla de estados/salida, figura 6.49.

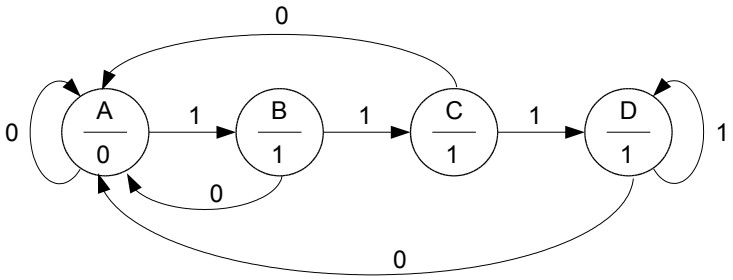


Figura 6.48.

S \ x	0	1	z
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1
	NS		z

Figura 6.49.

Si realizamos la siguiente asignación de estados para q_1, q_2: $A \rightarrow 00$, $B \rightarrow 01$, $C \rightarrow 10$ y $D \rightarrow 11$, obtenemos la tabla de transición de la figura 6.50.

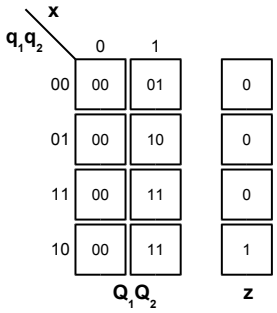


Figura 6.50.

Si para la realización escogemos biestables JK, la tabla de excitación/salida resultante es la mostrada en la figura 6.51.

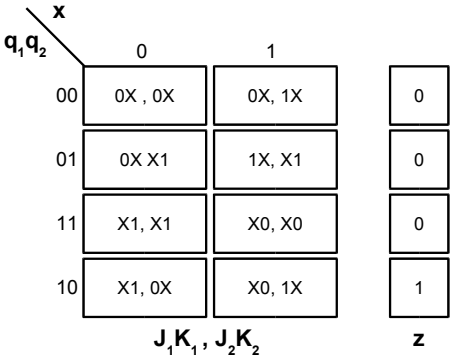


Figura 6.51.

Y de la tabla de excitación/salida, obtenemos las ecuaciones de excitación y de salida:

$$\begin{aligned} Z &= q_1\,\overline{q_2} \\ J_1 &= X\,q_2 \\ K_1 &= \overline{X} \\ J_2 &= X \\ K_2 &= \overline{X} + \overline{q_1} \end{aligned}$$

(b) **Como autómata de *Mealy***. Si analizaramos la respuesta de este circui-

to detector de secuencia diseñado como autómata de *Moore* (a partir del diagrama de estados), observaríamos que la salida $Z = 1$ se activa un ciclo de reloj posterior a la recepción del tercer 1 de la secuencia. En la siguiente figura se ha representado, para una cierta secuencia de entradas, la evolución de estados de la máquina y la salida correspondiente a cada estado:

	$Tiempo(T) \rightarrow$												
$T:$	1	2	3	4	5	6	7	8	9	10	11	12	$\ldots$
$X:$	0	0	1	1	1	0	0	1	1	1	1	0	$\ldots$
$S:$	A	A	A	B	C	D	A	A	B	C	D	D	$\ldots$
$Z:$	0	0	0	0	0	1	0	0	0	0	1	0	$\ldots$

Supongamos que inicialmente está activo el estado A, tiempo $T = 1$. La entrada en ese ciclo de reloj es 0, por tanto el próximo estado es A y salida 0, según el diagrama de estados. En $T = 2$, tenemos estado A, entrada $X = 0$, por tanto, salida $Z = 0$, y próximo estado A. En $T = 3$, tenemos estado A, y $X = 1$, por lo que el próximo estado es B. En $T = 4$, tenemos estado B, y entrada $X = 1$, por lo que el próximo estado es C. En $T = 5$ se recibe el tercer 1 de la secuencia a detectar, pero el estado en este ciclo de reloj es C cuya salida asociada es 0. No obstante, el próximo estado es el D. En $T = 6$, el estado actual ya es D cuya salida es $Z = 1$, y dado que se recibe un 0 por la entrada X, el próximo estado es A.

Como se aprecia en la figura anterior, la salida Z se activa a partir del siguiente ciclo de reloj en el que se detectó el último bit de la secuencia $1, 1, 1$. Si el diseño del circuito se hubiera basado en una máquina de *Mealy*, la detección (activación de la salida Z) se hubiese realizado en el mismo ciclo de reloj en el que se recibe el tercer 1 consecutivo. En efecto, en un circuito de *Mealy*, la salida depende del estado y de la entrada, por tanto, si en el estado C (diagrama *Moore* anterior), al que se llega cuando se han recibido dos unos consecutivos, la entrada X es un 1, inmediatamente el circuito reconoce la secuencia a detectar y activa su salida $Z = 1$. Si en este estado C, la entrada hubiese sido $X = 0$, la salida sería 0 sin mayor problema. En cambio en un esquema *Moore* en donde la salida sólo depende del estado, es necesario asegurarse de que se han recibido los tres unos antes de activar la salida, por eso es obligatorio un cuarto estado, el D. Un CSS basado en una estructura de tipo *Mealy* emplea un menor número de estados que una estructura de tipo *Moore* que resuelva el mismo problema.

Para el ejemplo que nos ocupa, sólo se necesitan tres estados. La función de cada estado es:

- Estado A: Estado inicial que "memoriza" que no se han recibido ningún 1.
- Estado B: Estado al que se llega cuando se recibió un 1 en el ciclo de reloj anterior. Por tanto memoriza que se ha recibido un 1.
- Estado C: Estado que "memoriza" que se han recibido dos o más 1s consecutivos.

La "columna vertebral" del diagrama de estados de la máquina se ha representado en la figura 6.52.

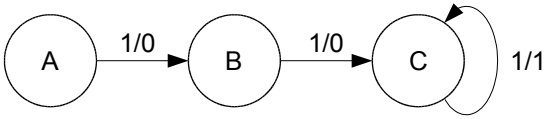


Figura 6.52.

Finalmente se completa el diagrama de estados con las transiciones que faltan, figura 6.53

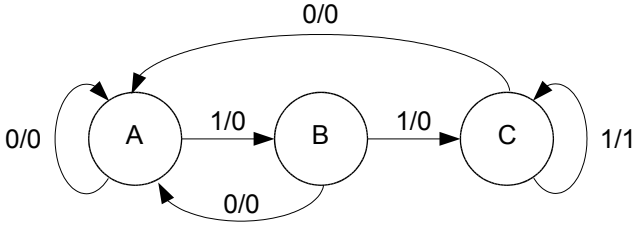


Figura 6.53.

A partir de aquí se obtiene la tabla de estados/salida, mostrada en la figura 6.54.

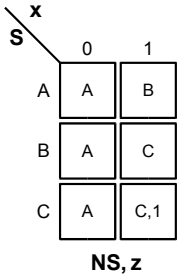


Figura 6.54.

Mediante la asignacion $A \rightarrow 00$, $B \rightarrow 01$ y $C \rightarrow 10$, obtenemos la tabla de transición/salida, figura 6.55.

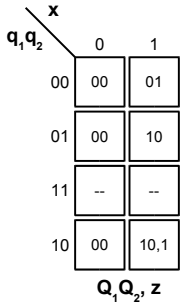


Figura 6.55.

Si utilizamos biestables de tipo D, el contenido de la tabla de excitación/salida sería igual que el de la tabla de transición (ya que en este tipo de biestables se verifica que $D = Q$), figura 6.56.

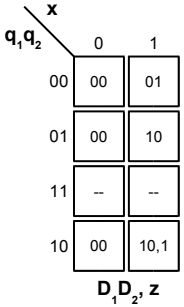


Figura 6.56.

Y las ecuaciones de excitación y salida resultantes son:

$$\begin{aligned} Z &= X\, q_1\, \overline{q_2} \\ D_1 &= X\, q_1 + X\, q_2 \\ D_2 &= X\, \overline{q_1}\, \overline{q_2} \end{aligned}$$

6.4.2. Minimización de tablas de estado

En este apartado estudiaremos una técnica que va a permitir simplificar eliminar aquellos estados innecesarios o redundantes de la tabla de estados. Reduciendo el número de estados, equivale a disminuir el número de biestables que necesita la máquina secuencial y, por tanto, reducir el coste de la misma.

Estados idénticos. Se dicen que dos estados (p, q) son idénticos si cumplen las siguientes condiciones:

(a) Los próximos estados de p para cualquier entrada son los mismos que los próximos estados de q para las mismas entradas. Esto se puede expresar como $\mathrm{NS}(p, x_i) = \mathrm{NS}(q, x_i)\ \forall x_i$

(b) Las salidas de p y q para todas las entradas son idénticas: $Z(p, x_i) = Z(q, x_i)\ \forall x_i$

Es razonable pensar que si en una tabla de estados encontramos algunos estados que tienen idénticas salidas y se comportan de forma exacta (coinciden sus próximos estados), los estados repetidos o redundantes pueden ser eliminados sin que se afecte al funcionamiento de la máquina secuencial.

Ejemplo 1. Consideremos la tabla de estados/salida mostrada en la figura 6.57.

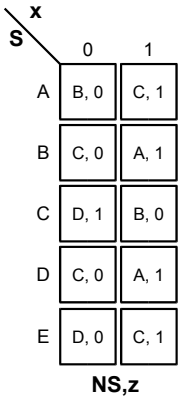


Figura 6.57.

Analizando dicha tabla se deduce que los estados B y D son idénticos, ya que tienen las mismas salidas y próximos estados para toda entrada x. Por tanto, uno de los estados puede ser eliminado, por ejemplo, el D. Hay que tener especial cuidado al obtener la nueva tabla reducida, ya que al desaparecer el estado D, las referencias que existieran a este estado eliminado deben cambiarse hacia el estado equivalente, el B. La tabla reducida queda tal como aparece en la figura 6.58.

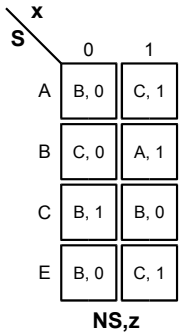


Figura 6.58.

Nuevamente, en la tabla anterior, existen estados idénticos, en este caso A y E. Uno de ellos puede ser eliminado, por ejemplo, el estado E. La tabla resultante se muestra en la figura 6.59.

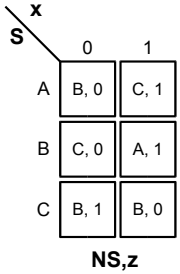


Figura 6.59.

En esta nueva tabla, ya no existen más estados idénticos.

Hemos comprobado en el ejemplo anterior que la búsqueda y eliminación de los estados equivalentes ha reducido una tabla de estados de cinco estados, y por tanto implementada físicamente por tres biestables, a una tabla de estados equivalente reducida con sólo tres estados, y por consiguiente, implementada con dos biestables.

No obstante, este método de reducción no es siempre eficaz, puesto que no nos asegura que la tabla reducida sea mínima. Pueden encontrarse el caso de tablas de estados que no tengan estados idénticos y que no sean tablas mínimas. Para asegurar la obtención de la tabla mínima se desarrollará el método de "eliminación de estados redundantes por pares equivalentes".

Par equivalente. Se dicen que dos estados p y q forman un par equivalente si cumplen las siguientes condiciones:

(a) Los próximos estados de p y q para cada entrada forman un par equivalente. Esto es, $\mathrm{NS}(p, x_i)$ y $\mathrm{NS}(q, x_i)$ son equivalentes $\forall x_i$

(b) Las salidas de p y q para todas las entradas, son idénticas: $Z(p, x_i) = Z(q, x_i)\ \forall x_i$

La definición de un par equivalente no es en sí una definición, puesto que para definir al par equivalente, una de las condiciones es que los proximos

estados sean tambien equivalentes. Esta definición *recursiva* no nos proporciona un método que nos permita determinar si un par dado es equivalente, sin embargo, sí podemos "darle la vuelta" para decir si un par no es equivalente, o lo que es lo mismo, si es *incompatible*.

Par incompatible. Se dicen que dos estados p, q forman un par incompatible si se cumplen *al menos una* de los dos condiciones siguientes:

(a) Los próximos estados de p y q para alguna entrada forman un par incompatible. Esto es, $\mathrm{NS}(p, x_i)$ y $\mathrm{NS}(q, x_i)$ son incompatibles para alguna entrada x_i NS(p,xi)

(b) Existe alguna entrada para la cual las salidas de p y q son diferentes: $Z(p, x_i) \neq Z(q, x_i)$ para alguna x_i

La parte más útil de la definición de par incompatible viene dada por la segunda condición, que nos permite determinar directamente un conjunto inicial de pares que no pueden ser equivalentes. A partir de ahí se pueden deducir todas las equivalencias posibles entre estados, como se muestra en el siguiente ejemplo.

Ejemplo 2. En la tabla de estados de la figura 6.60, podemos ver que cualquier pareja de estados que incluya al estado F forma un par incompatible, puesto que este estado tiene salida $Z = 1$ para $X = 0$, mientras que los restantes estados tienen salida $Z = 0$, para dicha entrada: (A, F) es incompatible, (B, F) es incompatible, (C, F) es incompatible, y así sucesivamente.

Ahora se plantea la cuestión de si la pareja (C, D) forman un par equivalente. Para ello, hay que determinar si cumplen las condiciones dadas en la definición. La condición (b) es satisfecha por esta pareja, puesto que las salidas de C y D para toda entrada X son 0. La condición (a) es más difícil de demostrar. En primer lugar debemos determinar las parejas de próximos estados de C y D para todas las entradas. Para $X = 0$, C tiene como próximo estado G, y D tiene a H, mientras que si $X = 1$, C tiene como próximo estado a E, mientras que D tiene a F. Por tanto las parejas de próximos estados son (G, H) para $X = 0$ y (E, F) para $X = 1$.

Entonces (C, D) formarán un par equivalente si (G, H) y (E, F) son equivalentes. Para saber si (G, H) y (E, F) son equivalentes habría que repetir el

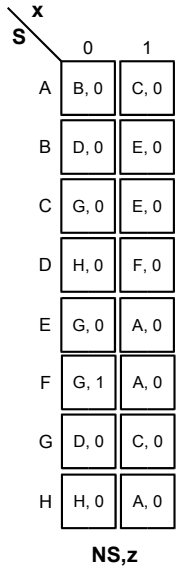


Figura 6.60.

mismo proceso que con (C, D), es decir, determinar si estos estados tienen las mismas salidas y, si es así, comprobar si los próximos estados de (G, H) y (E, F) son equivalentes. Se puede comprobar que una de las parejas de próximos estados de (C, D) es el par (E, F), que es incompatible según se vió anteriormente. Por tanto, la pareja (C, D) también es incompatible.

Como puede intuirse, el proceso que determina si un par es equivalente o incompatible no es complejo, pero sí tedioso, debido al número elevado de iteraciones que se ha de realizar. Si a esto le añadimos que además hay que determinar la equivalencia de todas las parejas posibles de estados, el proceso se hace muy poco agradable. No obstante, a continuación se describe un método que simplifica la obtención de los pares equivalentes.

6.4.2.1. Método de eliminación de estados redundantes por pares equivalentes

6.4.2.1.1. Fundamentos

El método de búsqueda de los pares equivalentes pretende determinar cúales son las parejas de estados equivalentes entre sí. Las parejas de estados que son equivalentes entre sí tienen un compartamiento similar por lo que pueden simplificarse o fusionarse en un único estado.

El concepto de par equivalentes es más global que el del par idéntico, de hecho se puede demostrar que si un par es idéntico, también cumple que es equivalente, pero si un par es equivalente, no tiene por qué ser idéntico. En la figura 6.61 se muestra el diagrama de estados de una máquina que tiene dos estados p y q que son idénticos.

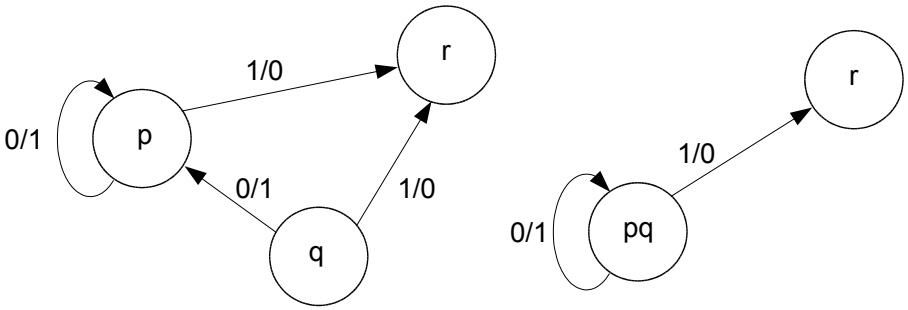


Figura 6.61.

Es razonable pensar que no se necesitan dos estados distintos que se comporten de la misma forma, por tanto se puede reducir hasta tener una única estado, el estado pq. En la figura 6.62 se han representado dos estados, que no son idénticos, pero que se comportan de forma idéntica.

Las salidas de p y q son iguales para todo posible valor de entrada (a la que designamos, en adelante, como X). El próximo estado de p y q para $X = 1$ es r, pero los próximos estados de p y q para $X = 0$ son ellos mismos. Ahora se plantea la siguiente cuestión, ¿es necesario que la máquina secuencial tenga dos estados distintos, que generan las mismas salidas, tengan el mismo próximo estado para $X = 1$, y para $X = 0$ tengan el mismo comportamiento, esto

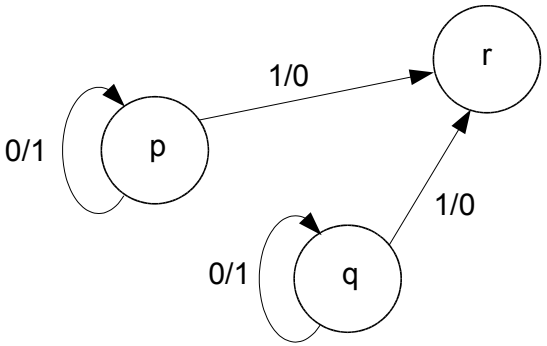


Figura 6.62.

es, se quedan en ellos mismos?. La respuesta es que no. En efecto, con un estado equivalente a p y q hubiera bastado. Este estado equivalente genera las mismas salidas que p y q, tienen a r como próximo estado para $X = 1$ y a sí mismo para $X = 0$. No hace falta tener dos estados distintos para ello. Por tanto p y q se pueden aunar en un mismo estado. Evidentemente los estados p y q no son idénticos, pero intuitivamente es claro que son equivalentes entre sí. El diagrama de estados simplificado se muestra en la figura 6.63.

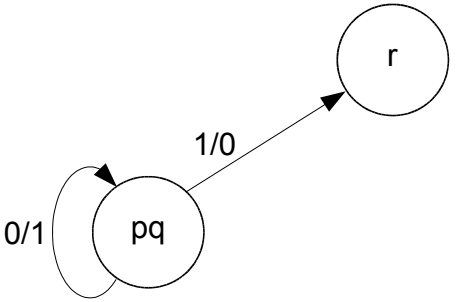


Figura 6.63.

A continuación se verán algunos ejemplos que nos ayuden a comprender la filosofía de reducción por el método de los pares equivalentes y partiendo de la premisa de que todo estado es compatible consigo mismo.

Ejemplo 1. En la figura 6.64 se muestra parte del diagrama de estados de una máquina secuencial. Se desea seber si la pareja (p, q) forman un par equivalente.

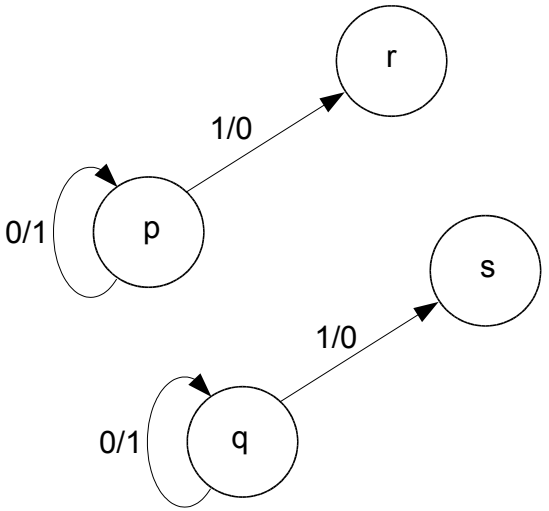


Figura 6.64.

Para este ejemplo, las salidas de p y q son idénticas para toda combinación de entradas. Por otro lado, $\mathrm{NS}(p,1) = r$ y $\mathrm{NS}(q,1) = s$ y $\mathrm{NS}(p,0) = p$ y $\mathrm{NS}(q,0) = q$. Por tanto, p y q forman un par equivalente si las parejas (p,q) para $X = 0$ y (s,r) para $X = 1$, son equivalentes entre sí. Es de destacar el hecho de que para demostrar que (p,q) son equivalentes hay de nuevo que averiguar si (p,q) son equivalentes. Esta situación se resuelve de forma fácil, de hecho no hay nada que nos indique que p y q no puedan ser equivalentes, por tanto, siempre supondremos, hasta que se demuestre lo contrario, que una pareja de estados es equivalente. Sólo nos queda, pues, comprobar si r y s son estados equivalentes. Supongamos que sí. En este caso, estos estados pueden simplificarse a un único estado, como se muestra en la figura 6.65.

Y esta nueva situación no es diferente a la mostrada anteriormente: los estados p y q son equivalentes y se pueden aunar a un mismo estado, figura 6.66.

En el caso en que r y s no fueran equivalentes, p y q tampoco lo serían, no pudiendo estos simplificarse en un estado, situación mostrada en la figura 6.67, ya que si se unen para formar un único estado, para $X = 1$, es imposible distinguir cuándo hay que dirigirse al estado r y cuándo al estado s.

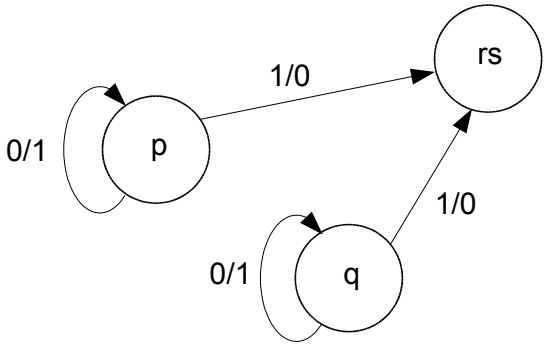


Figura 6.65.

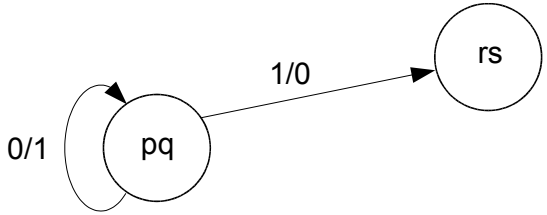


Figura 6.66.

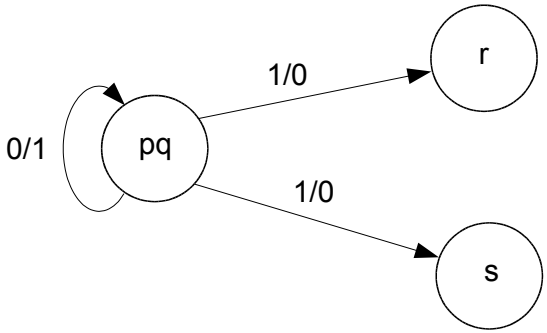


Figura 6.67.

Ejemplo 2. La figura 6.68 muestra una parte del diagrama de estados de una máquina secuencial. Se desea saber si los estados p y q son equivalentes.

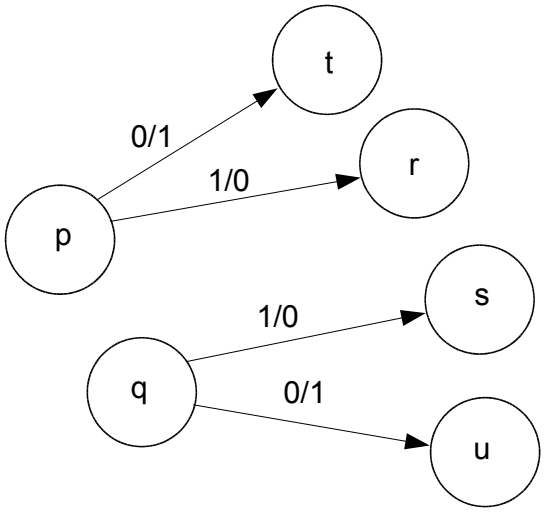


Figura 6.68.

Las salidas de p y q son las mismas para cualquier entrada. Por otro lado, los próximos estados de p y q son: para $X = 0$, (t, u); y para $X = 1$, (r, s). La pareja (p, q) será equivalente si lo son las parejas (t, u) y (r, s). Si éstas son equivalentes, la pareja (t, u) forma un estado reducido, al igual que la pareja (r, s), figura 6.69.

De aquí se ve que p y q forma un par equivalente (en este caso son idénticos) y por tanto se pueden simplificar en un mismo estado, como se muestra en la figura 6.70.

Si (t, u) ó (r, s) fueran incompatibles, (p, q) no podrían simplificarse a un único estado.

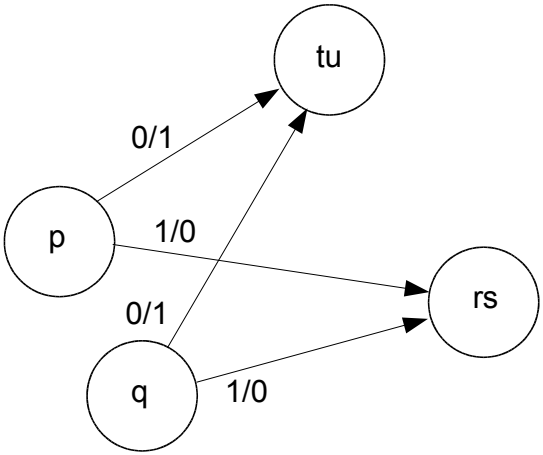


Figura 6.69.

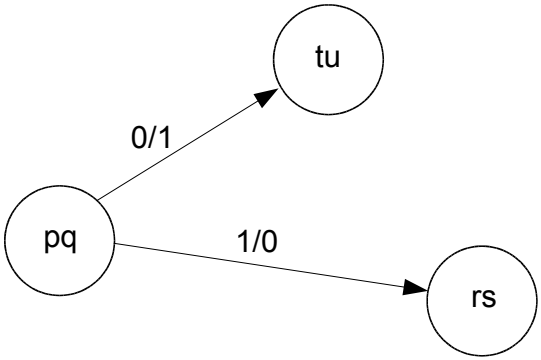


Figura 6.70.

6.4.2.1.2. Tabla de implicación

Para encontrar todos los pares equivalentes se construye la **tabla de implicación** a partir de la tabla de estados. Se trata de una tabla en forma de escalera en la que cada estado se puede relacionar con todos los demás. En la siguiente figura se muestra, a modo de ejemplo, una tabla de implicación construida a partir de una tabla de estados que poseía cinco estados. En la tabla de implicación se relacionan todas las parejas que se puedan formar usando estos cinco estados: $(A, B); (A, C); (B, E); \ldots$ En total, son 10 parejas que dispondrán, cada una, de una casilla en la tabla. En la figura 6.71 se ha marcado la casilla que relaciona la pareja (A, D).

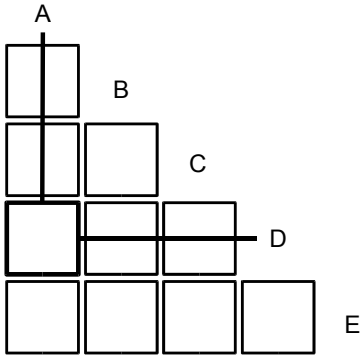


Figura 6.71.

Cada casilla de la tabla de implicación incluirá un aspa, $\times$, si la pareja de estados asociada a dicha casilla es incompatible. En caso contrario, la casilla contendrá la información de los próximos estados de la pareja asociada a dicha casilla para toda condición de entrada. Siguiendo el ejemplo anterior, supongamos que los estados (A, D) son incompatibles (porque las salidas son distintas): en tal caso, la casilla correspondiente aparecerá marcada con un aspa, figura 6.72.

Supongamos que las salidas de los estados (A, D) son idénticas (todavía no sabemos si son compatibles o no), y supongamos que los próximos estados de A y D para $X = 0$ son D y E, respectivamente, mientras que para $X = 1$ son A y B. En tal caso, la casilla (A, D) se rellena según se muestra en la figura 6.73.

Una vez que la tabla está rellena, y partiendo de las parejas cuya casilla

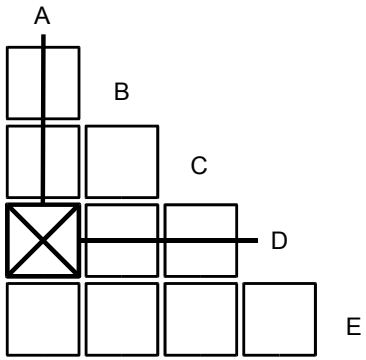


Figura 6.72.

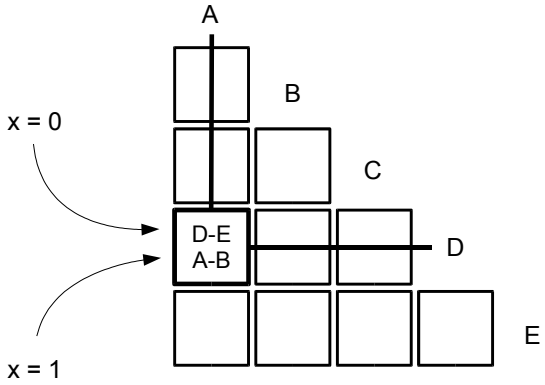


Figura 6.73.

contiene un aspa (son incompatibles), se van determinando qué otras parejas son incompatibles. El método se ilustra con el siguiente ejemplo.

Ejemplo. En la figura 6.74 se muestra la tabla de implicación asociada a una tabla de 8 estados.

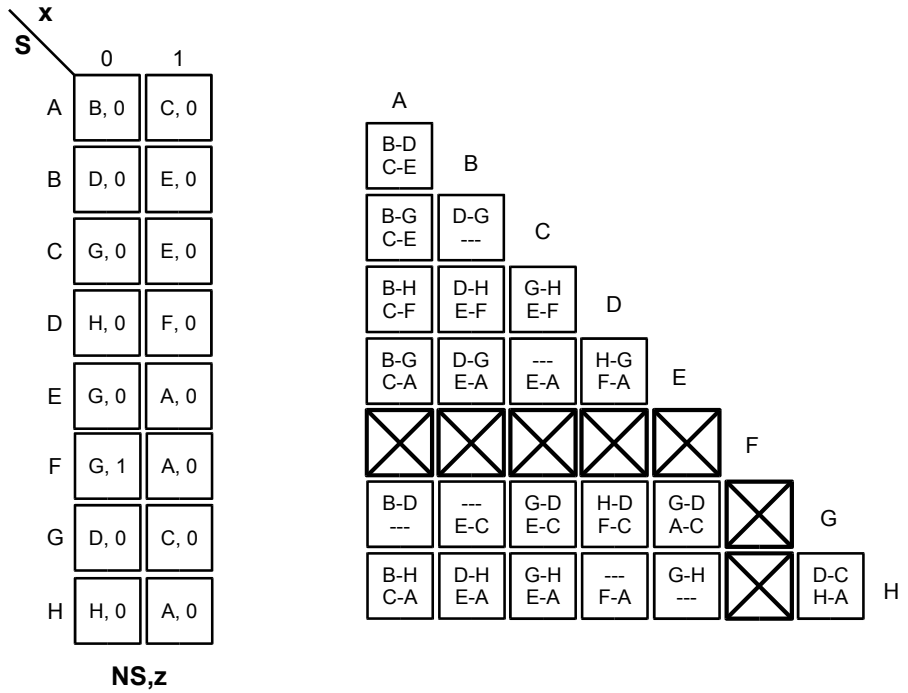


Figura 6.74.

Se puede observar en la tabla de implicación que existen en algunas casillas el símbolo—. Por ejemplo, la casilla asociada a la pareja (A, G), tiene como próximos estados para $X = 0$, la pareja (B, D) y para $X = 1$, la pareja (C, C). Como un estado es compatible consigo mismo, sólo nos bastaría determinar si (B, D) son equivalentes, para averiguar si (A, G) lo son.

Usando la tabla de implicación iremos desechando aquellas parejas de estados que son incompatibles. Por ejemplo la pareja (A, D), tiene como próximos estados, las parejas (B, H) y (C, F), como (C, F) es incompatible, (A, D)

también lo es. Marcamos con un aspa esta casilla. Procediendo con las restantes casillas, llegamos a la situación mostrada en la figura 6.75.

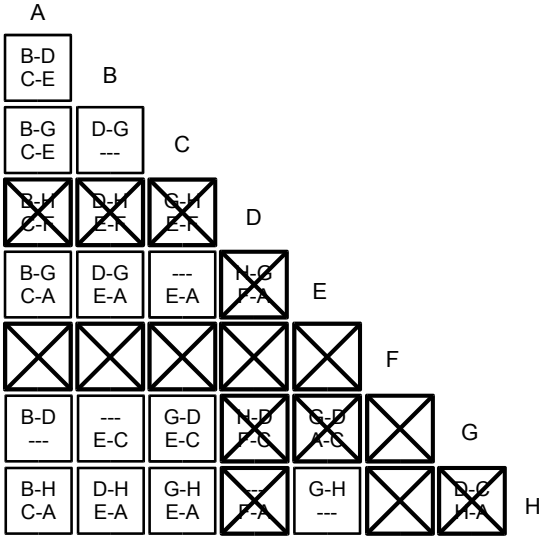


Figura 6.75.

Aquí han aparecido nuevos estados incompatibles. Realizaremos un nuevo recorrido por todas las casillas para buscar más estados incompatibles. De hecho se realizarán tantos recorridos por la tabla de implicación hasta que no aparezca ningún nuevo estado incompatible. Para el caso del ejemplo, el resultado final es el representado en la figura 6.76.

A partir de esta tabla, se obtendrá la lista de todos los estados que son equivalentes entre sí.

6.4.2.1.3. Obtención de los pares equivalentes

El método de obtención de pares equivalentes se ilustra siguiendo el ejemplo anterior.

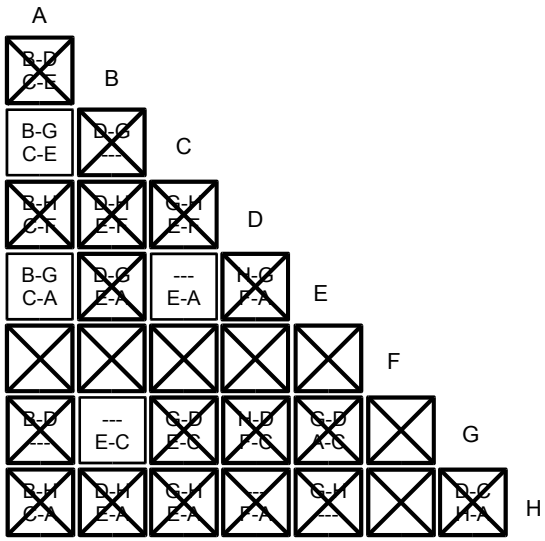


Figura 6.76.

Ejemplo. La lista de pares equivalentes se construye utilizando una tabla de dos columnas en la que, primero, situamos todos los estados en la parte izquierda, comenzando por el último estado de la tabla de implicación, que se sitúa en la parte superior, y terminando por el primer estado de la tabla de implicación en la parte inferior.

H	
G	
F	
E	
D	
C	
B	
A	

De arriba hacia abajo, cada estado de la parte izquierda se compara con la lista de estados de la parte derecha. Para el primero de ellos, el estado H, no hay lista con la que comparar, por lo que se añade, sin más ,al lado derecho.

H	(H)
G	
F	
E	
D	
C	
B	
A	

Una vez que un estado se compara con el de la lista a su derecha, la nueva lista generada es heredada por siguiente estado a comparar. Para el ejemplo anterior, el estado H lo pasamos a la fila del G.

H	(H)
G	(H)
F	
E	
D	
C	
B	
A	

A continuación se compara G con el estado H mediante la tabla de implicación. Si G y H son equivalentes, estos se unen en un único estado (GH), pero si son incompatibles, se forman dos estados independientes $(G)(H)$, como en realidad ocurre. Esta lista de dos estados es heredada por el estado F.

H	(H)	
G	(H)	(G)
F	(H)	(G)
E		
D		
C		
B		
A		

El estado F es incompatible con los estados H y G, por lo que se forma el conjunto $(H)(G)(F)$ que, nuevamente, se pasa al estado E.

H	(H)				
G	(H)	(G)			
F	(H)	(G)	(F)		
E	(H)	(G)	(F)		
D					
C					
B					
A					

El proceso se repite de forma similar hasta el estado C, debido al carácter incompatible de los estados H, G, F, E, D.

H	(H)				
G	(H)	(G)			
F	(H)	(G)	(F)		
E	(H)	(G)	(F)	(E)	
D	(H)	(G)	(F)	(E)	(D)
C	(H)	(G)	(F)	(E)	(D)
B					
A					

La comparación entre la lista de estados y el estado C, muestra que este es equivalente al estado E, dando lugar al estado unificado (CE).

H	(H)				
G	(H)	(G)			
F	(H)	(G)	(F)		
E	(H)	(G)	(F)	(E)	
D	(H)	(G)	(F)	(E)	(D)
C	(H)	(G)	(F)	(E)	(D)
B	(H)	(G)	(F)	(CE)	(D)
A					

De la comparación de la lista con el estado B, surge un nuevo par equivalente (BG).

H	(H)				
G	(H)	(G)			
F	(H)	(G)	(F)		
E	(H)	(G)	(F)	(E)	
D	(H)	(G)	(F)	(E)	(D)
C	(H)	(G)	(F)	(E)	(D)
B	(H)	(G)	(F)	(CE)	(D)
A	(H)	(BG)	(F)	(CE)	(D)

Por último, el estado A es equivalente a los estados C y E, que, como a su vez, son equivalentes entre sí, formando el par (CE), dan lugar al estado (ACE).

H	(H)				
G	(H)	(G)			
F	(H)	(G)	(F)		
E	(H)	(G)	(F)	(E)	
D	(H)	(G)	(F)	(E)	(D)
C	(H)	(G)	(F)	(E)	(D)
B	(H)	(G)	(F)	(CE)	(D)
A	(H)	(BG)	(F)	(CE)	(D)
	(H)	(BG)	(F)	(ACE)	(D)

La lista de estados simplificada es $(H)\,(BG)\,(F)\,(ACE)\,(D)$.

De la lista de estados resultantes deducimos que se ha conseguido reducir a cinco estados una tabla de ocho estados. La tabla reducida se obtiene a partir de la original sabiendo que los estados G y B son el mismo y los estados A, C y E también son el mismo. Llamaremos h al estado de la tabla reducida que procede de H, b al estado de la tabla reducida que procede de la unión de (B, G), a al estado que procede de la unión (A, C, E), f del F y d del D. La tabla reducida se muestra en la figura 6.77.

6.4.3. Asignación de estados

Como ya sabemos, la asignación de estados consiste en asociar a un estado S una combinación binaria $q_{n-1}, q_{n-2}, \ldots, q_0$, donde n es el número de biesta-

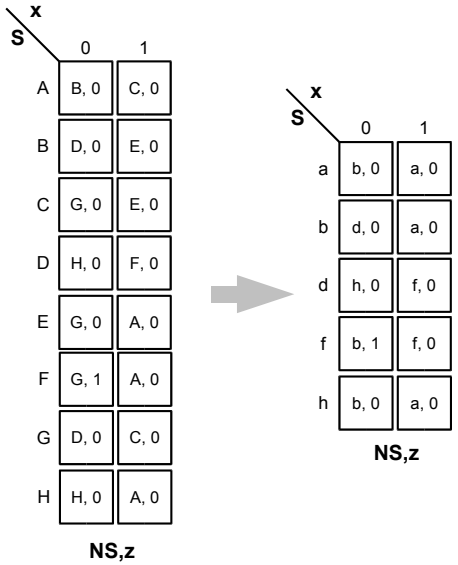


Figura 6.77.

bles del CSS, es decir, esa combinación será la que presenten las salidas de los biestables cuando el CSS se encuentre en el estado S.

La asignación que se escoja repercutirá en la complejidad de la parte combinacional del diseño resultante: por ello es deseable disponer de métodos que nos ayuden a encontrar buenas soluciones.

6.4.3.1. Método exhaustivo

Se basa en el barrido sistemático de todas las posibilidades; por ello, con este método podemos encontrar la solución más óptima.

Las etapas correspondientes del procedimiento de diseño del CSS deberán ser repetidas tantas veces como asignaciones posibles existan.

El problema que presenta este procedimiento es que es muy costoso porque el número de asignaciones posibles es muy elevado incluso para CSSs con pocos estados, tal como revela la tabla 6.1.

Tabla 6.1:

m	r	N
2	1	1
3	2	3
4	2	3
5	3	140
6	3	420
7	3	840
8	3	840
9	4	10810800
10	4	75675600
11	4	454053600

m = número de estados
r = número mínimo de biestables
N = número de asignaciones específicas

Para circuitos de sólo 2 estados, no existe ninguna elección posible, y por ello, no aparece el problema de la asignación. Para 3 y 4 estados, lo más sencillo es intentar las tres asignaciones posibles y ver cuál produce el circuito más económico. Para más de 4 estados, este método resulta demasiado costoso o incluso impracticable. En estos casos es preferible usar la técnica basada en las reglas de adyacencia, aunque, como veremos, no siempre proporcionan la mejor solución.

En un principio puede resultar extraño que para dos estados, A y B, exista una única asignación posible, cuando claramente podemos encontrar dos:

- $(A, B) = (0, 1)$
- $(A, B) = (1, 0)$

Sin embargo, hay que tener en cuenta que el resto de las asignaciones se pueden obtener a partir de las asignaciones que hemos calificado de "específicas" en la tabla anterior, bien complementando algunas variables, bien cambiando el orden de las mismas. Lo importante es que ninguno de esos cambios afecta a la forma de cualquier función booleana, y por ello, carecen de relevancia en lo que se refiere a coste.

En el caso de dos estados, por ejemplo, claramente una de las asignaciones es el complemento de la otra, por lo que, desde el punto de vista del coste, son

Tabla 6.2:

Estado	Asignación 1	Asignación 2	Asignación 3
A	00	00	01
B	01	11	11
C	11	01	00

Tabla 6.3:

Estado	Asignación 1	Asignación 2	Asignación 3
A	00	00	01
B	01	11	11
C	11	01	00
D	10	10	10

idénticas. En el caso de 3 ó 4 estados, las asignaciones específicas pueden ser las que se muestran en las tablas 6.2 y 6.3, respectivamente.

El resto de asignaciones posibles se obtendría complementando una o ambas variables de estado y/o cambiando el orden de las mismas.

Por ejemplo, veamos cómo una cierta asignación de tres estados, escogida al azar, puede transformarse en alguna de las que aparecen en la tabla 6.2. Consideremos la asignación $(A, B, C) = (10, 01, 00)$, que no aparece en la tabla. Si complementamos la primera variable, obtenemos la asignación $(A, B, C) = (00, 11, 10)$; si además ahora cambiamos de orden las variables, obtendremos la segunda asignación de dicha tabla, $(A, B, C) = (00, 11, 01)$.

Ejemplo. Para ilustrar el método exhaustivo, partimos de la tabla de estados de la figura 6.78.

Se trata simplemente de repetir los pasos de diseño para las tres asignaciones específicas diferentes que aparecen en la tabla 6.3. No tiene sentido probar más asignaciones, ya que como hemos indicado, el resto de asignaciones se pueden reducir a tan sólo éstas tres.

No detallaremos aquí todos los pasos que hay que seguir, sino que directamente diremos cuáles son las ecuaciones de excitación y salida, y compararemos los resultados en lo que se refiere a coste.

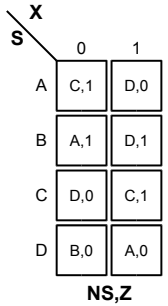


Figura 6.78.

- Asignación 1:

$$\begin{aligned} D_1 &= \bar{q_1}\bar{q_0} + q_1 q_0 + X q_0 \\ D_0 &= \bar{X}\bar{q_0} + X q_1 q_0 \\ Z &= X q_0 + \bar{X}\bar{q_1} \end{aligned}$$

  que se pueden implementar con 2 puertas de 3 entradas y 7 puertas de 2 entradas (compartiendo lo términos comunes).

  Total: 9 puertas / 20 entradas.

- Asignación 2:

$$\begin{aligned} D_1 &= \bar{X}\bar{q_1}q_0 + \bar{X}q_1\bar{q_0} + X\bar{q_1}\bar{q_0} + X q_1 q_0 \\ D_0 &= \bar{X}\bar{q_0} + X\bar{q_1}q_0 \\ Z &= X q_0 + q_1 q_0 + \bar{X}\bar{q_1}\bar{q_0} \end{aligned}$$

  que se pueden implementar con 1 puerta de 4 entradas, 7 puertas de 3 entradas y 4 puertas de 2 entradas.

  Total: 12 puertas / 33 entradas.

- Asignación 3:

$$\begin{aligned} D_1 &= \bar{X}q_0 + X\bar{q_0} \\ D_0 &= \bar{q_1} + X\bar{q_0} \\ Z &= \bar{X}\bar{q_0} + q1\bar{q_0} + X\bar{q_1}q_0 \end{aligned}$$

  que se pueden implementar con 2 puertas de 3 entradas y 6 puertas de 2 entradas.

  Total: 8 puertas / 18 entradas.

A tenor de los resultados obtenidos, seleccionaríamos la asignación 3 ya que es la que implica un menor coste.

¿Cómo se obtiene el número de asignaciones específicas? Dado el número de estados, m, de un CSS, determinamos el mínimo número de biestables, r, necesarios para diseñarlo (mediante la relación $2^{r-1} < m \leq 2^r$, ya conocida). A cada uno de los m estados tenemos que asignarles una combinación binaria, distinta cada vez, de las 2^r posibles. Esta asignación se corresponde con el modelo matemático de las variaciones sin repetición de 2^r elementos tomados de m en m, luego habrá:

$$\frac{2^r!}{(2^r-m)!}$$

maneras de realizarla. Como ya se ha comentado, muchas de las asignaciones coinciden en coste, concretamente aquellas que resultan de complementar algunas variables (en total 2^r posibilidades) y/o de alterar del orden de las mismas (en total $r!$ posibilidades), por lo cual el número de asignaciones específicas, N, será:

$$N = \frac{1}{r!}\frac{1}{2^r}\frac{2^r!}{(2^r-m)!} = \frac{(2^r-1)!}{r!(2^r-m)!}$$

6.4.3.2. Método de las adyacencias

Es especialmente útil si el diseño se implementa con biestables tipo D, aunque también da buenos resultados con biestables JK.

El método de las adyacencias se usa cuando tenemos CSSs de más de 4 estados, en otro caso es preferible el método exhaustivo. Antes de enunciar las reglas a seguir y explicar de qué forma se aplican, definimos el concepto de estados adyacentes, que aparece frecuentemente en ellas.

Estados adyacentes. Aquellos estados cuyos códigos binarios se diferencian en el valor de un sólo bit se dice que son adyacentes.

Por ejemplo: en un CSS construido con tres biestables, los estados representados con los códigos 010 y 110 son adyacentes porque sólo difieren en el valor de un bit, en este caso, el primero.

Las reglas de adyacencia, en orden de prioridad, son las siguientes:

Regla 1a. Deben ser adyacentes aquellos estados cuyos próximos estados coincidan para todas las combinaciones de entrada. En el ejemplo de la figura 6.79, los estados (P, Q) deben ser adyacentes según esta regla.

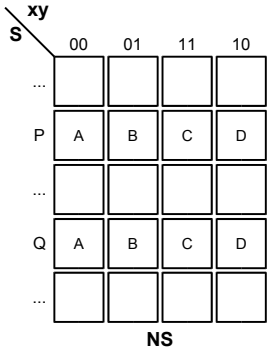


Figura 6.79.

Regla 1b. Deben ser adyacentes aquellos estados cuyos próximos estados coincidan para algunas combinaciones de entrada, teniendo mayor prioridad aquellas adyacencias en los que haya mayor número de coincidencias. En el ejemplo de la figura 6.80, analizando cada combinación de entrada (es decir, columna a columna, figura 6.81):

- Para la combinación $xy = 00$: P y Q comparten como estado siguiente a A, luego (P, Q) deben ser adyacentes;
- Para la combinación $xy = 01$: P y Q comparten como estado siguiente a B, luego (P, Q) deben ser adyacentes;
- Para la combinación $xy = 11$: Q y R comparten como estado siguiente a H, luego (Q, R) deben ser adyacentes;
- Para la combinación $xy = 10$: los siguientes estados no coinciden, luego no se extrae ninguna adyacencia.

Dado que se repite la adyacencia (P, Q), ésta tendrá mayor prioridad que (Q, R) en caso de conflicto.

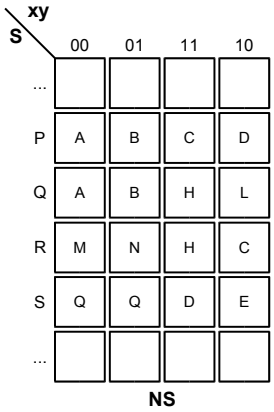


Figura 6.80.

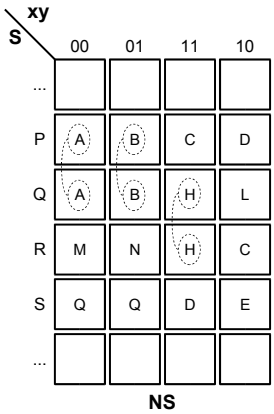


Figura 6.81.

Regla 2. Deben ser adyacentes aquellos estados que sean próximos estados de un mismo estado. En el ejemplo de la figura 6.82, (A, B, C, D) deben ser adyacentes.

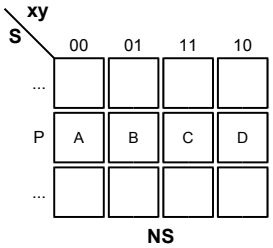


Figura 6.82.

Regla 3a. Deben ser adyacentes aquellos estados cuyas salidas coincidan para todas las combinaciones de entrada. En el ejemplo de la figura 6.83, (P, Q) deben ser adyacentes.

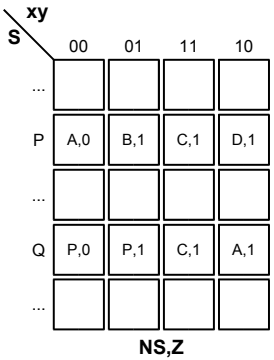


Figura 6.83.

Regla 3b. Deben ser adyacentes aquellos estados cuyas salidas coincidan para algunas combinaciones de entrada, teniendo mayor prioridad aquellas adyacencias en las que haya mayor número de coincidencias. En el ejemplo de la figura 6.84, analizando cada combinación de entrada (es decir, columna a columna, figura 6.85):

- Para la combinación $xy = 00$: Q y R comparten el valor de salida 1, luego (Q, R) deben ser adyacentes;
- Para la combinación $xy = 01$: P y R comparten el valor de salida 1, luego (P, R) deben ser adyacentes;

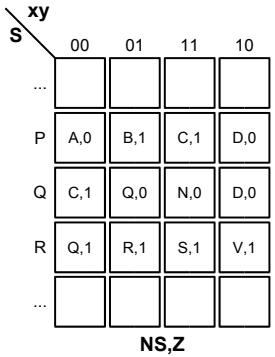


Figura 6.84.

- Para la combinación $xy = 11$: P y R comparten el valor de salida 1, luego (P, R) deben ser adyacentes;
- Para la combinación $xy = 10$: P y Q comparten el valor de salida 0, luego (P, Q) deben ser adyacentes.

Dado que se repite la adyacencia (P, R), ésta tendrá mayor prioridad que (Q, R) y (P, Q) en caso de conflicto.

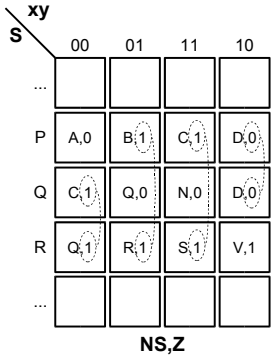


Figura 6.85.

En el caso de que el circuito tenga muchas salidas, y no demasiados estados, es conveniente adoptar el siguiente orden de aplicación: 3a, 3b, 1a, 1b, 2.

En la mayoría de los casos, las reglas darán resultados conflictivos, en el sentido de que si hacemos adyacentes ciertos estados según cierta regla, es

muy posible que otra regla no se pueda aplicar. Sin embargo, una asignación hecha en base a una determinada regla no debe modificarse para satisfacer una regla de menor prioridad.

El problema con estos conflictos se reduce entonces a los debidos a adyacencias con una misma prioridad: no sabemos a priori cuál de ellas será la que nos lleve al mejor resultado. Si alguna de esas adyacencias vuelve a aparecer en otras reglas de menor prioridad, puede ser conveniente considerarla de mayor importancia que las otras adyacencias, pero no necesariamente se dará esta situación. Esta falta de determinación a la hora de escoger las adyacencias es la que nos lleva a considerar las reglas sólo como una guía.

Por último, cabe destacar que las reglas 1a, 1b y 2 ayudan a mejorar las ecuaciones de excitación, y las reglas 3a y 3b, las ecuaciones de salida.

Para ilustrar cómo deben aplicarse las reglas anteriores, se desarrollan dos ejemplos.

Ejemplo 1. Consideremos en primer lugar la tabla de estados de la figura 6.86. Se trata, por supuesto de encontrar una asignación de estados aceptable, mediante las reglas de adyacencia.

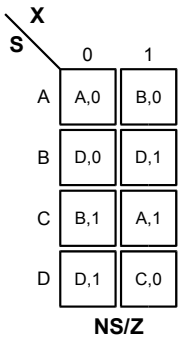


Figura 6.86.

La **regla 1a** dice que deberían ser adyacentes aquellos estados cuyo próximo estado coincide para todas las combinaciones de valores de entrada. En la tabla de estados que nos ocupa, no se da el caso, ya que los pares de siguientes

estados, $A-B$, $D-D$, $B-A$ y $D-C$, no se repiten.

$$\textbf{Regla 1a}: \ -$$

La **regla 1b** dice que deberían ser adyacentes aquellos estados para los que coincida al menos uno de sus próximos estados, para una misma combinación de entradas. Para $X=1$, los estados B y D coinciden en su próximo estado. Por ello, deberían ser adyacentes.

$$\textbf{Regla 1b}: \ (B,D)$$

La **regla 2** dice que deberían ser adyacentes aquellos estados que son estados siguientes a un mismo estado. Para el estado A: deben ser adyacentes (A,B). Para el B, la regla no se aplica ya que los próximos estados son el mismo. Para el C, tenemos que hacer adyacentes (A,B) de nuevo: esta adyacencia aparece 2 veces, así que tendrá mayor importancia que otras adyacencias de la misma regla. Lo anotamos escribiendo $(A,B)[2]$. Para D, la adyacencia recomendada es (C,D).

$$\textbf{Regla 2}: \ (A,B)[2]; \ (C,D)$$

La **regla 3a** dice que deberían ser adyacentes aquellos estados cuyas salidas coincidan para todas las combinaciones de las entradas. En este caso, no coinciden ninguna de las combinaciones de salida de cada estado.

$$\textbf{Regla 3a}: \ -$$

La **regla 3b** dice que deberían ser adyacentes aquellos estados cuyas salidas coincidan para, al menos, una de las combinaciones de las entradas. Cuando $X=0$, los estados (A,B) tienen $Z=0$; (C,D) tienen $Z=1$. Cuando $X=1$, son (A,D) quienes hacen $Z=0$; y (B,C), $Z=1$.

$$\textbf{Regla 3b}: \ (A,B); \ (C,D); \ (A,D); \ (B,C)$$

Ahora hay que usar estas adyacencias para efectuar la asignación de estados. Sobre un k-mapa de tantas variables q_i como biestables tenga el circuito se distribuyen los distintos estados intentando cumplir el mayor número posible

de adyacencias, atendiendo también a sus prioridades. Hay que recordar que en un k-mapa las casillas contiguas horizontalmente o verticalmente corresponden a combinaciones binarias adyacentes.

El CSS del ejemplo dispone de 4 estados, luego serán necesarios 2 biestables para implementarlo. Las variables de estado serán, por tanto: q_1, q_0. El k-mapa sobre el que se hace la asignación, tendrá la siguiente forma mostrada en la figura 6.87.

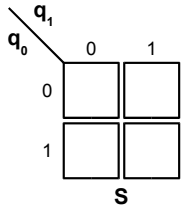


Figura 6.87.

Tras hacer varias pruebas, llegamos a la solución de la figura 6.88, que verifica las adyacencias de mayor importacia: (B, D), regla 1b; (A, B), regla 2 (dos veces) y regla 3b; y (C, D), regla 2 y regla 3b. Hay que tener en cuenta, tras lo expuesto en la discusión sobre el método exhaustivo, que cualquier asignación que resulte de complementar una o ambas variables o de cambiar el orden de las mismas, resulta en una nueva asignación que mantiene las mismas adyacencias, por lo que, en cuanto a coste final de la solución, son idénticas. El resto de adyacencias, (A, D) y (B, C), no puede hacerse cumplir

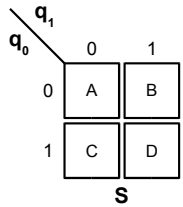


Figura 6.88.

con esta asignación. Sin embargo, esas adyacencias son de menor prioridad, así que no vamos a cambiar la asignación que hemos escogido para hacerlas cumplir.

Ejemplo 2. Consideremos la tabla de estados de la figura 6.89.

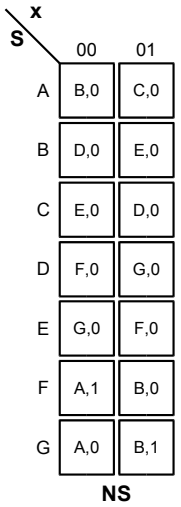


Figura 6.89.

Para aplicar la **regla 1a**, buscamos estados cuyos próximos estados coincidan para todas las combinaciones de entrada. Los únicos estados que lo verifican son F y G.

$$\textbf{Regla 1a}: \ (F, G)$$

La **regla 1b** es similar a la 1a, salvo que no exige una coincidencia de los próximos estados para todas las combinaciones de entrada. Es evidente que los estados que verifican el enunciado de la regla 1a también verifican la regla 1b: dado que hemos asignado una prioridad más alta para las adyacencias propuestas por la regla 1a, no es necesario volver a repetirlas. En nuestro ejemplo, la regla 1b no propone nuevas adyacencias.

$$\textbf{Regla 1b}: \ -$$

Según la **regla 2**, debemos hacer adyacentes entre sí todos los estados que comparten la característica de ser próximo estado de uno determinado. En este caso, aparecen adyacencias repetidas, por lo que éstas tendrán mayor prioridad.

$$\textbf{Regla 2}: \ (B, C); (D, E)[2]; (F, G)[2]; (A, B)[2]$$

La **regla 3a** es análoga a la regla 1a, pero referida a las las salidas en lugar de a los próximos estados: debemos hacer adyacentes aquellos estados cuyas salidas coincidan para todas las combinaciones de entrada, lo que, en el ejemplo, ocurre para los estados A, B, C, D y E.

$$\textbf{Regla 3a}: \ (A, B, C, D, E)$$

Al igual que en el caso anterior, la **regla 3b** es análoga a la regla 1b, pero referida a las las salidas en lugar de a los próximos estados: debemos hacer adyacentes aquellos estados cuyas salidas coincidan para al menos una combinación de entrada; en el ejemplo, para $X = 0$ otendríamos que (A, B, C, D, E, G) deben ser adyacentes, y para $X = 1$, (A, B, C, D, E, F).

$$\textbf{Regla 3b}: \ (A, B, C, D, E, G); (A, B, C, D, E, F)$$

El CSS del ejemplo dispone de 7 estados, luego serán necesarios 3 biestables para implementarlo. Las variables de estado serán, por tanto, q_2, q_1 y q_0. El k-mapa sobre el que haremos la asignación de estados será el mostrado en la figura 6.90 .

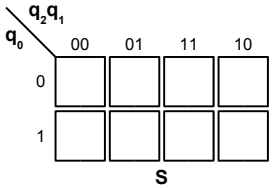


Figura 6.90.

Hay muchas maneras de colocar los estados; tomemos, por ejemplo, la asignación de la figura 6.91:

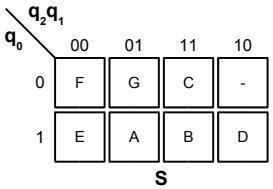


Figura 6.91.

Esta asignación verifica todas las adyacencias obtenidas con las reglas 1a, 1b y 2. Las reglas 3a y 3b no podemos usarlas, ya que el número de estados que

aparecen en cada adyacencia no es potencia de 2, y por lo tanto no podemos encontrar un lazo que los cubra a todos.

Ejercicios propuestos

Problema 6.1 Representa la secuencia de salida q, q' de un biestable SR asíncrono para las siguientes secuencias de entrada (figura 6.92).

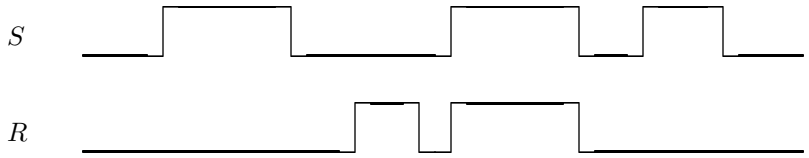


Figura 6.92.

Problema 6.2 Dibujar el cronograma de la salida q de un biestable T disparado por flanco de subida y con entradas asíncronas de Reset y Preset cuyo estado inicial es desconocido y para las ecuencia de señales dada en la figura 6.93.

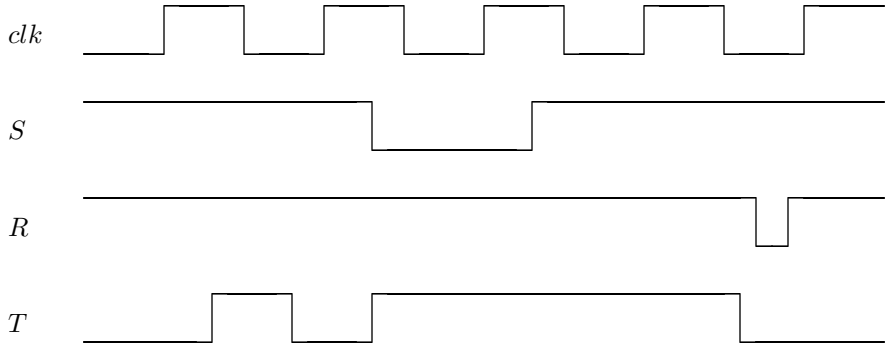


Figura 6.93.

Problema 6.3 La figura 6.94 representa un circuito secuencial con una entrada X y una salida Z. Se pide:

a)Analizar dicho circuito.

b) Obtener la evolución de la salida Z para la secuencia de entrada de la figura 6.95, suponiendo que el estado inicial de los biestables es $q_1 q_0 = 00$

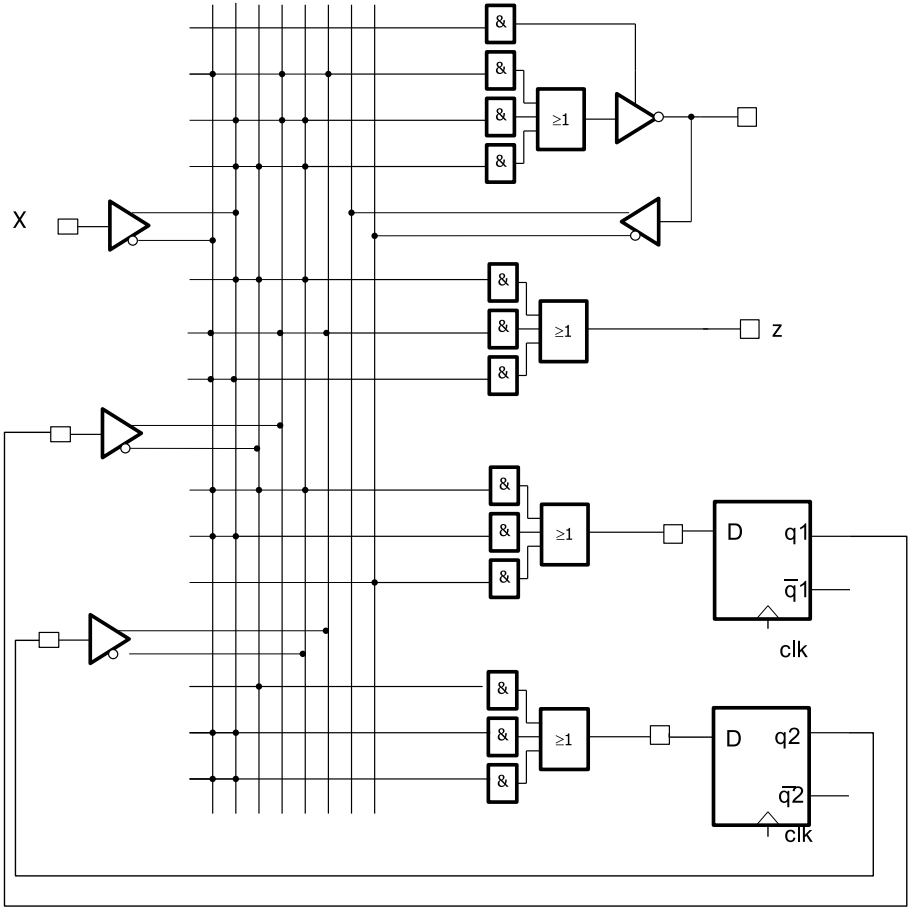


Figura 6.94.

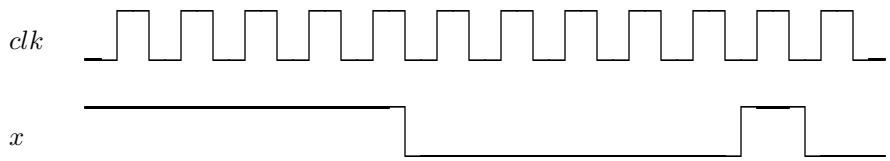


Figura 6.95.

Problema 6.4 Diseñar un circuito secuencial síncrono que reciba en serie por su única entrada números de cuatro bits (del menos significativo al más significativo) y cuya misión sea calcular el complemento a 2 de dichos números. Implemente el circuito con biestables JK.

Problema 6.5 Considere el biestable MN cuya tabla de funcionamiento se muestra en la tabla 6.4

Tabla 6.4:

M	N	Q
0	0	0
0	1	1
1	0	$\overline{q}$
1	1	prohibido

Constrúyalo utilizando biestable D y puertas lógicas y justifique si, con este tipo de biestable, se puede implementar cualquier circuito secuencial síncrono.

Problema 6.6 Diseñe una máquina secuencial que genere, periódicamente, los 7 primeros números primos. Utilice como máximo 3 biestables de tipo T y puertas lógicas.

Problema 6.7 Se desea diseñar un circuito que permita detectar la secuencia 101 siempre que esta venga sin errores. Para ello, por una entrada X se envían, sincronizados con la señal de reloj, grupos de cuatro bits. Los tres primeros bits de cada grupo se corresponden con la secuencia a detectar, mientras que el cuarto, es un bit de paridad impar. El circuito dispone de una salida Z que se activa (1 lógico) coincidiendo con el cuarto bit de entrada de cada grupo si los tres primeros bits del mismo se corresponden con la secuencia y el bit de paridad recibido es correcto. En cualquier otro caso, la salida Z se encuentra

a 0. Obtenga las ecuaciones de salida y excitación para el circuito detector suponiendo que se dispone de biestables D y puertas lógicas.

Problema 6.8 Obtenga la forma de onda de la salida de un biestable SR (a) de tipo Master-Slave con cambio de salida en flanco descendente; (b) disparado por flanco ascendente; suponiendo que las entradas se presentan la secuencia que se muestra en 6.96

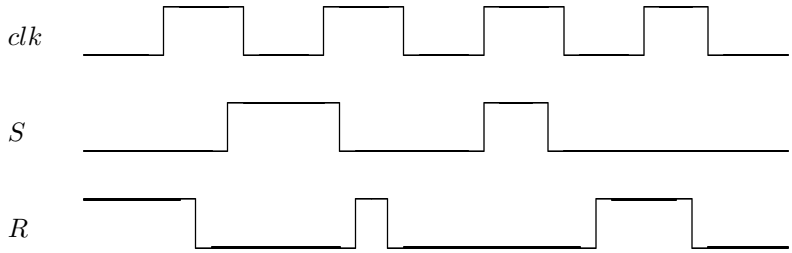


Figura 6.96.

Problema 6.9 Se desea obtener un circuito secuencial síncrono con dos entradas, X e Y, y una salida Z. Por las entradas X e Y se reciben, de forma sincronizada, números de tres bits sin signo, bit a bit, empezando por el más significativo. La salida Z se debe activar coincidiendo con el tercer bit de entrada si el número X es mayor que el Y. Se dispone de tres biestables para el diseño (un JK, un T y un D), obtenga la tabla de excitación y salida del circuito secuencial.

Problema 6.10 Diseñe un circuito secuencial síncrono con una entrada X por la que se reciben en serie números de 4 bits (primero llega el bit más significativo), y una salida Z que debe ponerse a 1 (coincidiendo con el último bit de cada número) si el número recibido es par (es decir, múltiplo de 2). Use biestables JK y puertas lógicas.

Problema 6.11 Se desea obtener un circuito secuencial síncrono con dos entradas, X e Y, y una salida Z. Por las entradas X e Y se reciben, de forma sincronizada, números de cuatro bits, bit a bit, empezando por el más significativo. La salida Z se debe activar coincidiendo con el cuarto bit de entrada si el número X es igual que el Y. Base su diseño en biestables D.

Problema 6.12 Diseñe un circuito secuencial síncrono con una entrada X y una salida Z, que se activa cuando por la entrada X se han detectado las secuencias 111, 110, 101, 000, 001 y 010 sin solapamiento. Base su diseño en biestables y puertas lógicas.

Capítulo 7

Subsistemas secuenciales

7.1. Introducción

Existen dispositivos comerciales que implementan funciones secuenciales de propósito especifico y general. Estos dispositivos son de uso muy común y un diseño a nivel de biestables y puertas lógicas de estos elementos provocaría que el dimensionado del circuito global que los contiene sería excesivamente grande. De estos dispositivos de uso frecuente estudiaremos los contadores, registros y los dispositivos lógicos programables.

7.2. Contadores

Un contador módulo k es un circuito digital capaz de contar k sucesos distintos. Estos dispositivos no son otra cosa que circuitos secuenciales cuyos cambios de estado se producen, evidentemente, a ritmo de su señal de reloj, y en el que cada estado "memoriza" un valor de cuenta. Por tanto, un contador módulo-k, tiene k estados de cuentas distintos, desde el 0, hasta el $k-1$. Estos dispositivos podrán incrementarse o no en función de los valores lógicos que tomen ciertas señales de control que se estudiarán más adelante. Inicialmente se supondrá que el contador siempre está contando, por lo que cada valor de cuenta se corresponde con un ciclo de reloj. En la figura 7.1 se muestra la

evolución de los estados de cuenta de un contador módulo k. Se observa que a partir del ciclo k de reloj, estado de cuenta $k-1$, el contador vuelve a su estado inicial (estado de cuenta 0). Esto es una característica de todos los contadores.

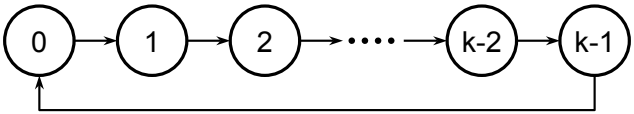


Figura 7.1.

Un contador módulo-k, tiene k estados, y éstos se pueden implementar con n biestables, existiendo una relación entre k y n determinada por la expresión:

$$2^{n-1} < k \leq 2^n$$

Así un contador módulo 8 se puede construir usando 3 biestables, uno de módulo 16 con cuatro biestables, y así sucesivamente. Normalmente los contadores se construyen usando el menor número de biestables posibles y si estos tienen un módulo que es potencia de 2, suelen identificarse también como **contadores de n bits**.

Si las salidas del contador son la codificación binaria del estado de cuenta y el número de estados es una potencia de 2, este suele denominarse **divisor de frecuencia**. En la figura 7.2 se ha representado la salida de un contador módulo 8 (contador de 3 bits o divisor de frecuencia de 3 bits).

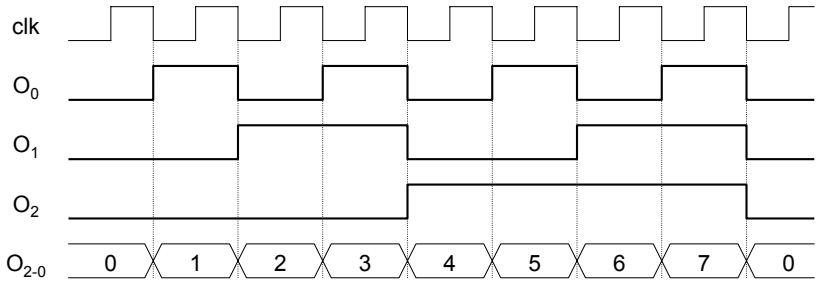


Figura 7.2.

En las siguientes secciones se presentarán dos posibles estructuras internas que permiten el funcionamiento de los contadores (estructura asíncrona y estructura síncrona) y se estudiarán las líneas de control y salidas más comunes

que, comercialmente, disponen estos dispositivos.

7.2.1. Contadores síncronos

Un contador tiene que estar formado internamente por biestables, cada uno de los cuales dispone de su entrada de reloj. Un contador síncrono es un circuito cuyos biestables internos son disparados por flanco y todos reciben la misma señal de reloj. Un contador síncrono, pues, cumple con las condiciones de una máquina secuencial síncrona.

Se puede determinar su estructura interna de dos formas distintas:

- Se obtiene el diagrama de estados correspondiente y a partir de este el circuito.
- Usando un procedimiento no sistemático, pero que se utiliza frecuentemente en la síntesis de circuitos complejos.

Se deja al lector la obtención de la estructura interna del contador según el primer método; aquí se describirá el segundo.

Si analizamos las señales de salida, figura 7.3 del contador módulo-8 observamos que:

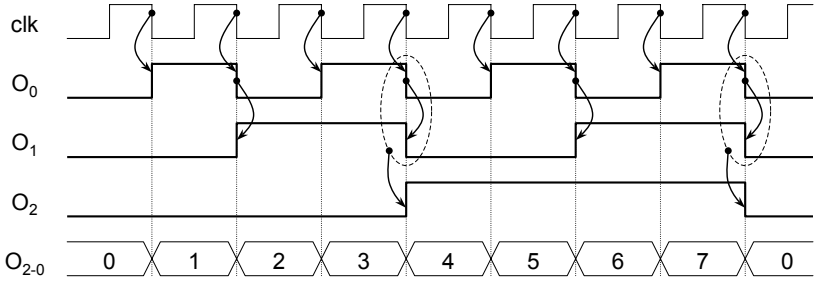


Figura 7.3.

- O_0: cambia de valor con cada suceso de entrada (1 ciclo de clk)
- O_1: cambia de valor con cada suceso de O_0 (1 ciclo de O_0)

- O_2: cambia de valor con cada suceso de O_1 (1 ciclo de O_1)

En particular:

- O_0: cambia de valor con cada flanco de bajada de clk
- O_1: cambia de valor en los flancos de bajada de clk si $O_0 = 1$
- O_2: cambia de valor en los flancos de bajada de clk si $O_0 = O_1 = 1$

Si se hace corresponder la salida O_i del contador con el valor q_i del biestable interno, se tiene que el biestable q_0 debe cambiar de valor, generando $0, 1, 0, 1, \ldots$, en cada flanco de bajada de clk; el biestable q_1 debe cambiar de estado en el flanco de bajada de clk si q_0 es igual a 1, en caso contrario, q_1 mantiene su valor; y el biestable q_2 debe cambiar de estado si $q_1 = q_2 = 1$, y en caso contrario mantiene su valor.

Esta descripción nos hace pensar que el tipo de biestable que mejor encaja para el diseño de los contadores es el biestable T, ya que éste, si su entrada es 0, provoca que el biestable no cambie de valor y si su entrada es 1, provoca el cambio del valor almacenado.

Por tanto, si usamos biestables tipo T, tenemos que:

$$\begin{aligned} T_0 &= 1 \\ T_1 &= q_0 \\ T_2 &= q_1\, q_0 \end{aligned}$$

En general, si diseñamos un contador de n bits, la expresión de la entrada del biestable j (donde $j = 1, 2, 3, \ldots n-1$) sería

$$T_j = q_0\, q_1\, \ldots\, q_{j-1}$$

En las figuras 7.4 y 7.5 se muestran dos posibles diseños para la estructura de un contador síncrono de n bits

En los siguientes apartados se estudiarán las señales de control y salida más comunes que acompañan a los contadores.

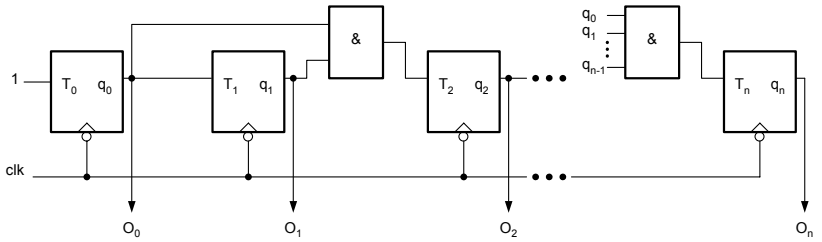


Figura 7.4.

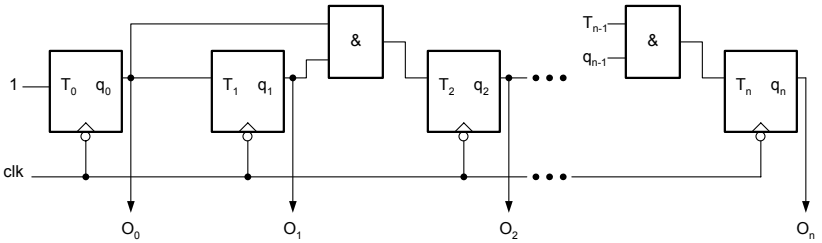


Figura 7.5.

7.2.1.1. *Reset* (*Clear*) y *Preset*

La línea de *Reset* o *Clear* permite inicializar el contador a su estado de cuenta 0 cuando es activada. La línea de *Preset* permite inicializar el contador a su valor de cuenta más alto (aquel en que todos los biestables están a 1). Estas líneas pueden ser activas en alto o en bajo, y existen dos modalidades desde el punto de vista funcional: asíncrona y síncrona.

Supongamos que la línea *Clear* de un contador es asíncrona. Esto implica que inmediatamente despúes a la activación de dicha señal, el contador pasa al estado de cuenta 0, independientemente de la señal de reloj. Por el contrario, si el *Clear* se activase, pero tuviese un modo de funcionamiento síncrono, el contador se pondría en el estado de cuenta 0 cuando llegase un flanco activo de la señal de reloj. En resumen, un *Clear* asíncrono no depende de clk para su funcionamiento, mientras que un síncrono sí. Esto es igualmente aplicable para la línea de *Preset*.

A continuación se estudiará cómo se implementan en el contador estas funciones.

***Clear* o *Reset* asíncrono**. De forma general diremos que todas las líneas de

control que operen de modo asíncrono lo hacen usando las entradas asíncronas de los biestables. El contador está formado por biestables tipo T. Supongamos que estos tienen entradas de Cl y Pr. Implementar un *Clear* asíncrono en el contador es equivalente a unir dicha entrada de *Clear* a todas las entradas Cl de los biestables que forman parte del contador. De esta manera, cuando *Clear* se active, todos los biestables se ponen a 0 de forma asíncrona y por consiguiente el contador pasa al estado de cuenta 0. En la figura 7.6 se ha representado la estructura de la celda i de un contador de n bits, y en la figura 7.7 la posible respuesta funcional de la salida del mismo para la situación de activación de la línea *Clear*, que este caso es activa en baja.

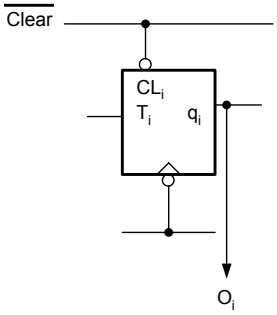


Figura 7.6.

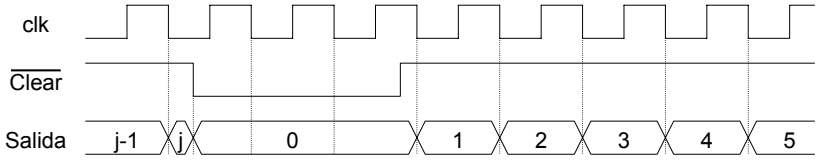


Figura 7.7.

Se observa en el cronograma que la salida se pone a 0 inmediatamente después de la activación de la línea de *Clear* (segundo ciclo de reloj) y se mantiene así hasta que *Clear* vuelva a valer 1 (cuarto ciclo de reloj), a partir del cual, el contador vuelve a inciar su ritmo de cuenta.

***Clear* o *Reset* síncrono**. En el *Clear* o *Reset* síncrono no se utilizan las entradas asíncronas de los biestables. Se trata en este caso de introducir por la entrada T de cada uno de los biestables el valor adecuado para que cuando se reciba un flanco activo en clk, cada uno de los biestables se ponga a 0. Hay que tener presente que el contador debe incrementarse cuando no se active el

Clear, por lo que las entradas T deben tener los valores ya determinados en el incio del apartado.

En primer lugar, hay que determinar qué se debe introducir a la entrada del biestable i para que su salida q_i pase a 0 en el siguiente ciclo de reloj. Supongamos que la salida $q_i = 0$, por tanto la entrada del biestable i debe ser $T_i = 0$ para que en el siguiente ciclo se mantenga el 0. En cambio si la salida del biestable $q_i = 1$, la entrada T_i debe ser un 1 lógico, para que en el siguiente ciclo, la salida q_i pase a valer 0. En resumen, si queremos hacer un *Clear*, la entrada del biestable T_i debe ser igual a q_i. Si todos los biestables del contador disponen de esta entrada, el contador pasará al estado de cuenta 0 cuando se reciba el flanco activo de reloj.

La figura 7.8 muestra la estructura de la celda básica de un contador que permite un *Clear* síncrono. La entrada del biestable i está conectada a la salida de un multiplexor de dos canales controlado por la señal *Clear* (activa en baja). De esta forma, si $Clear = 0$, el canal 0 del multiplexor (es decir, q_i) pasa a la entrada T_i, y se produce el efecto deseado. Si $Clear = 1$, el canal 1 del multiplexor (es decir, $q_i\, T_{i-1}$) pasa a la entrada T_i, y esto implica que el contador se incrementa (modo de funcionamiento normal).

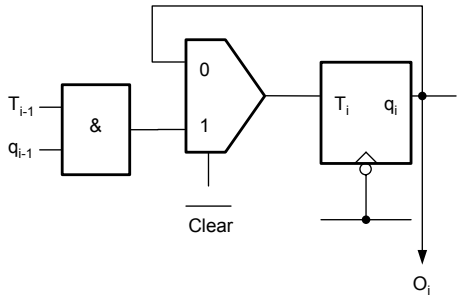


Figura 7.8.

En el cronograma de la figura 7.9 se ve claramente el funcionamiento del *Clear* síncrono, el cual no se produce realmente hasta la recepción de un flanco activo en clk.

***Preset* asíncrono**. Es idéntico al *Clear* asíncrono salvo que las entradas asíncronas de los biestables a usar es Pr. La etapa i-ésima del contador es la mostrada en la figura 7.10.

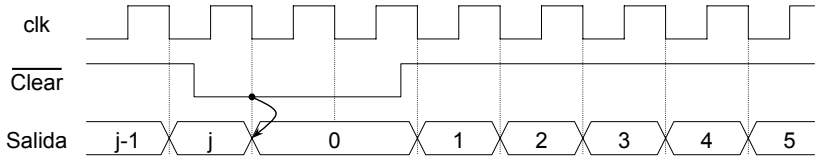


Figura 7.9.

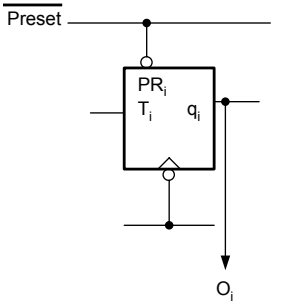


Figura 7.10.

De esta forma, cuando se activa la señal de *Preset* del contador, todos los biestables se ponen a 1 de forma asíncrona, poniendo al contador en el valor más alto de cuenta, es decir, en el estado $k-1$ (si el contador es módulo-k), figura 7.11.

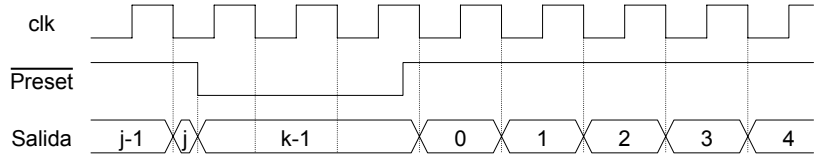


Figura 7.11.

***Preset* síncrono**. Es similar al *Clear* síncrono, pero en este caso pretendemos poner a 1 todos los biestables. Si $q_i = 1$, T_i debe ser 0, para que en el siguiente ciclo se mantenga el valor de q_i, en cambio si $q_i = 0$, T_i debe ser 1, para que $Q_i = 1$ en el siguiente ciclo de reloj. Por tanto $T_i = \overline{q_i}$, tal como se muestra en la figura 7.12. El cronograma correspondiente se ha representado en la figura 7.13.

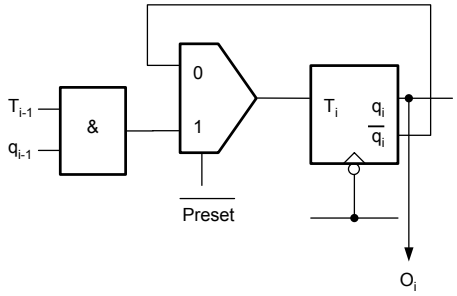


Figura 7.12.

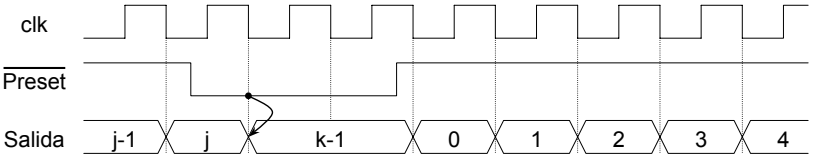


Figura 7.13.

7.2.1.2. Inhibición

Es una línea de control (activa en alta o en baja) cuya misión es detener el proceso de cuenta del contador. Esto implica que mientras que la línea esté activa, aunque se reciban flancos activos por la señal de reloj del contador, este no modifica su estado de cuenta. Podemos encontrar varios diseños alternativos para implementar la función de inhibición del contador. Uno de ellos es el mostrado por la figura 7.14, donde la entrada de reloj del contador pasa a través de una puerta AND antes de distribuirse por las entradas de reloj de cada uno de los biestables que forma parte de dicho contador. La otra entrada de la AND está formada por la linea de control $\overline{Inh}$. Si ésta vale 1, el reloj se distribuye a todos los biestables, pero sin $\overline{Inh}$ vale 0, se distribuye un 0 a los biestables, por lo que éstos no pueden cambiar de estado al no recibirse flancos por sus entradas de reloj.

No obstante este tipo de implementación, que se suele denominar **asíncrona**, presenta problemas a la hora de activar la inhibición de cuenta, ya que en función del nivel de la señal de reloj, se pueden producir incrementos no deseados en el valor de cuenta del contador, tal como se muestra en la figura 7.15. En el ejemplo anterior, los biestables del contador son disparados por flanco de bajada y la señal de inhibición es activa en bajo. Llamemos clk_2 a la señal de reloj que reciben los biestables del contador, esto es, la salida de la

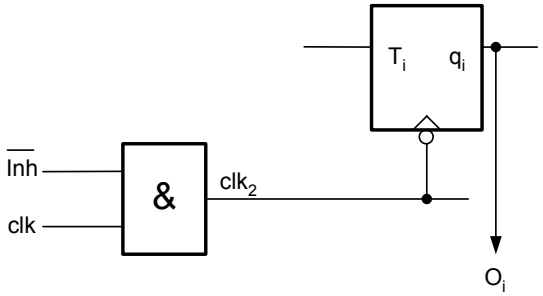


Figura 7.14.

puerta AND. Si $\overline{Inh} = 1$, $clk_2 = clk$, funcionamiento normal. Si suponemos que $clk = 1$ (nivel alto de reloj, y entonces $\overline{Inh} = 0$, lo lógico es que el contador, desde ese mismo momento, no debe de cambiar de estado, pero lo que ocurre es que clk_2 pasa de valer 1 a valer 0, esto es, genera un flanco de bajada que provoca que los biestables cambien de estado incrementando la cuenta del contador.

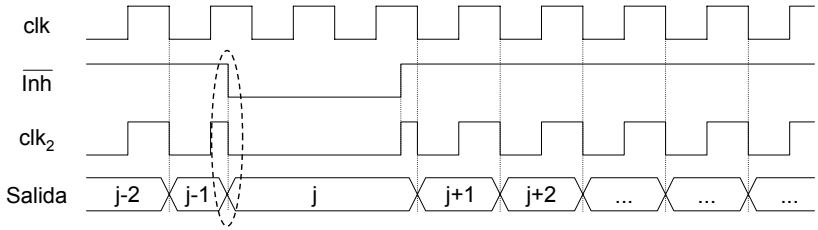


Figura 7.15.

La forma más habitual de implementar la función de inhibición es la que se describirá a continuación y que suele denominarse como **síncrona**. Aquí suponemos que la señal de reloj del contador llega, sin obstáculos, a todos los biestables que lo forman. Por tanto, se trata, ahora, de determinar qué deben tener las entradas T_i de los biestables del contador para que este mantenga el estado de cuenta. Que el contador mantega su valor de cuenta es equivalente a que cada uno de los biestables que lo forman mantengan su bit cuando $\overline{Inh}$ está activo. Evidentemente la solución es simple, $T_i = 0$ si $\overline{Inh}$ está activa. La figura 7.16 muestra la estructura de la celda básica de un contador que ha implementado este tipo de inhibición cuando esta es activa en bajo.

Como se aprecia en la figura 7.16, si $\overline{Inh} = 1$, la entrada T_i se correponderá con otras funciones a realizar por el biestable: *clear*, cuenta, etc.

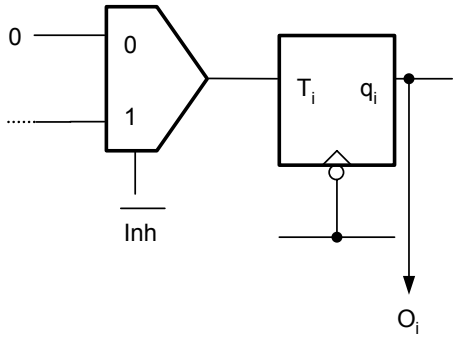


Figura 7.16.

En la figura 7.17 se representa el cronograma para la inhibición síncrona.

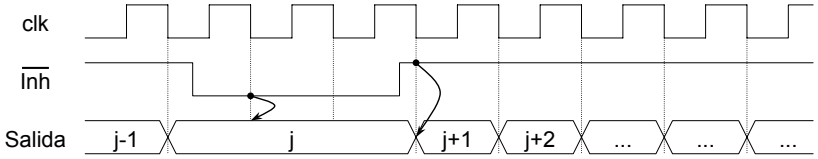


Figura 7.17.

7.2.1.3. *Load* (Carga en paralelo)

En determinadas aplicaciones es interesante cargar al contador con un valor de cuenta inicial. Esto se consigue con la línea de *Load*. Evidentemente el contador debe de disponer de algunas líneas de entrada adicionales por las que se introduce el dato a cargar, en caso de que se proceda con esta operación. Existen tantas líneas de datos como bits tenga el contador. Cada línea de datos contiene un bit del dato que será cargado en un biestable del contador. La línea de $Load$ puede ser activa en alta o en baja y ser síncrona o asíncrona.

***Load* asíncrono**. Esto se consigue mediante la activación de las señales asíncronas de Pr y Cl de cada uno de los biestables del contador. Reduzcamos el problema a una sóla etapa y supongamos que *Load* es activo en baja. Si $Load = 0$ y el bit a escribir en la etapa i es 0, se deben activar las entradas $Cl_i = 0$ y $Pr_i = 1$. Si $Load = 0$ y el bit a escribir es 1, entonces $Cl_i = 1$, y $Pr_i = 0$. Si $Load = 1$, $Cl_i = Pr_i = 1$. Si representamos estos valores en un

K-mapa y simplificamos, obtenemos las siguientes expresiones para Cl_i y Pr_i:

$$\begin{aligned} Cl_i &= Load + D_i \\ Pr_i &= Load + \overline{D_i} \end{aligned}$$

El circuito equivalente es el representado en la figura 7.18:

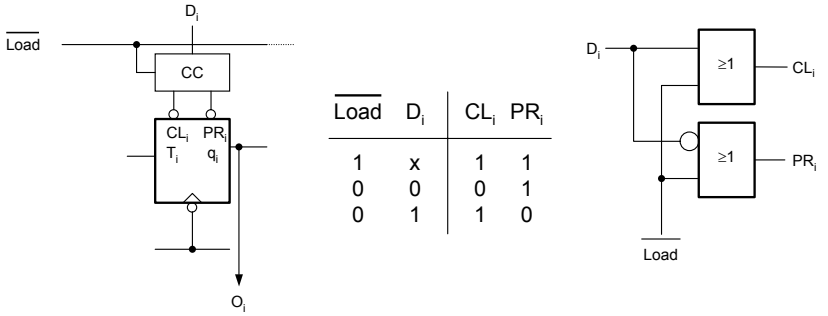


Figura 7.18.

La figura 7.19 ilustra el funcionamiento de un contador que ha implementado la operación de *Load* asíncrono. Como se observa, en el momento que se activa *Load*, se produce la carga del dato, simbolizado como el número N.

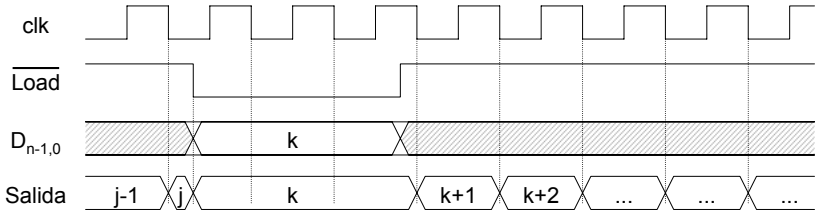


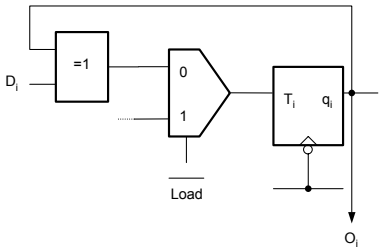
Figura 7.19.

***Load* síncrono**. Se trata de determinar cuál debe ser la entrada del biestable T_i para que este se cargue con el valor del dato D_i de entrada. Si suponemos que $Load$ es activo en baja, tenemos que cuando $Load = 0$, si además $D_i = 0$ y $q_i = 0$, entonces T_i debe ser 0; sin embargo, si $D_i = 0$ y $q_i = 1$, T_i debe ser 1 para forzar en el siguiente ciclo de reloj que la salida se ponga a 0. De forma equivalente si $D_i = 1$ y $q_i = 1$, entonces T_i debe ser 0; pero si $D_i = 1$ y $q_i = 0$, entonces T_i debe ser 1. En resumen, para que q_i sea igual al dato D_i que queremos cargar debemos comparar q_i con D_i, si los dos son iguales, T_i debe

ser 0 para no realizar ningún cambio, pero si D_i y q_i son distintos, T_i debe ser 1 para que en el siguiente ciclo, q_i tome el valor del bit a cargar.

$$T_i = q_i \oplus D_i$$

La figura 7.20 muestra la estructura de la etapa i-ésima de un contador que incorpora la carga síncrona.



$\overline{Load}$	D_i	q_i	T_i
1	x	...	...
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0

Figura 7.20.

En la figura 7.21 se muestra un cronograma donde se observa el funcionamiento de este tipo de operación.

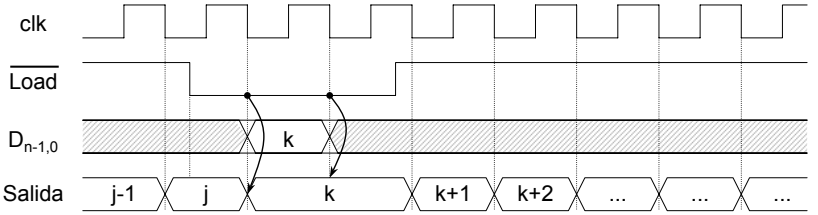


Figura 7.21.

7.2.1.4. Contadores reversibles (*Up/Down*)

Los contadores estudiados hasta ahora sólo tienen capacidad de cuenta ascendente, existen otros que tienen capacidad de cuenta descendente o incluso ambas (ascendente y descendente). Estos últimos son los contadores reversibles. En primer lugar se determinará la estructura de un contador descendente, y posteriomente se unirá con la del ascendente, introduciendo la señal de control *Up/Down*, que controla el sentido de la cuenta.

La figura 7.22 muestra la salida de un contador descendente de módulo 8.

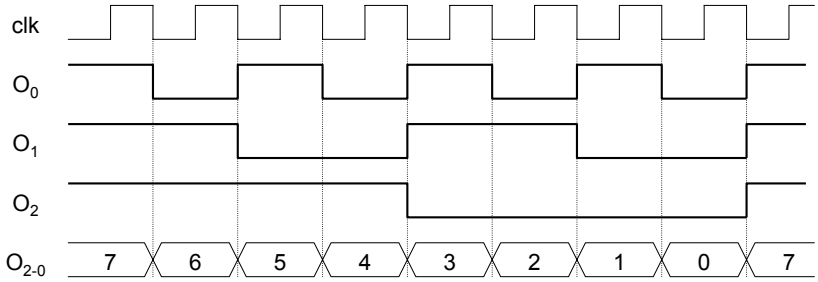


Figura 7.22.

Para obtener las ecuaciones de excitación de los biestables, procedemos de forma similar al caso del contador ascendente.

- O_0: cambia de valor con cada flanco de bajada de clk, luego $T_0 = 1$.
- O_1: cambia de valor en los flancos de bajada de clk si $O_0 = 0$, por tanto $T_1 = \overline{q_0}$.
- O_2: cambia de valor en los flancos de bajada de clk si $O_0 = O_1 = 0$, $T_2 = \overline{q_1}\,\overline{q_0}$.

Si en lugar de un contador módulo 8, disponemos de un contador de n bits, la expresión de la entrada del biestable i-ésimo de dicho contador sería:

$$T_i = \overline{q_0}\,\overline{q_1}\,\ldots\overline{q_{i-1}}\;(i = 1, 2, \ldots, n-1)$$

Se puede comprobar que la expresión es muy similar a la del ascendente salvo que aquí, los términos van complementados.

De forma equivalente, la expresión anterior se puede escribir como

$$T_i = \overline{q_{i-1}}\,T_{i-1}$$

Ya estamos en condiciones de diseñar el contador reversible. Estos disponen de una señal de control $UP/\overline{DOWN}$, que indica el sentido de la cuenta.

Si $UP/\overline{DOWN}$ está a 1, cuenta ascendete, si $UP/\overline{DOWN}$ está a 0, cuenta descendente.

Por tanto la entrada T_i de cada biestable del contador debe tener la siguiente expresión:

$$T_i = q_{i-1}\, T_{i-1}\, UP/\overline{DOWN} \; + \; \overline{q_{i-1}}\, T_{i-1}\, \overline{UP/\overline{DOWN}}$$

El circuito de la etapa i-ésima se muestra en la figura 7.23. Para ilustrar el funcionamiento de la entrada $UP/\overline{DOWN}$ se ha confeccionado el cronograma de la figura 7.24.

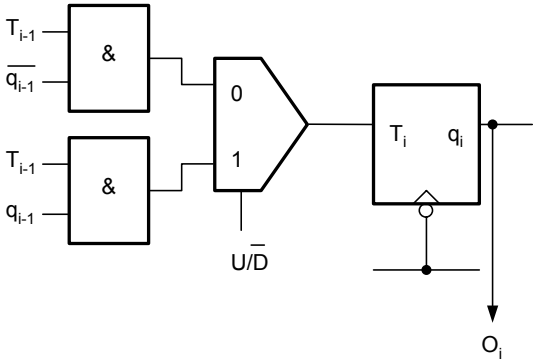


Figura 7.23.

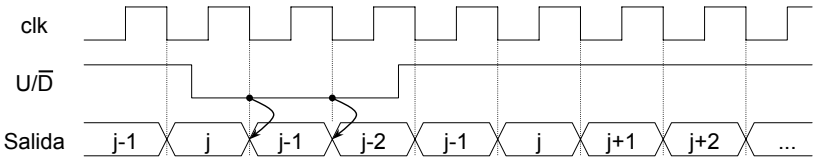


Figura 7.24.

7.2.1.5. Líneas de Carry y Borrow

Son salidas del contador que informan de la llegada al último estado de cuenta del mismo. Para contadores ascendentes, se usa la salida *Carry* (CY) y para descendentes *Borrow* (BW). La salida CY se pone a 1 cuando el contador

ascendente alcanza su estado de cuenta más alto. Por ejemplo, si el contador es de módulo 8, la salida CY se pone a 1 cuando el contador llega al estado 7, tal como se puede ver en la figura 7.25.

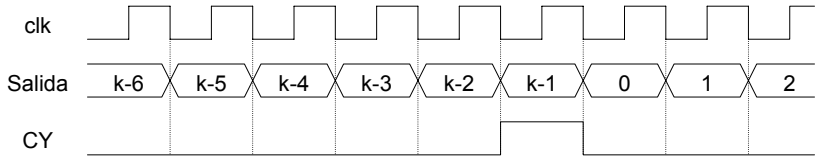


Figura 7.25.

En cambio la señal BW se activa cuando el contador descendente alcanza su estado de cuenta más bajo, el 0, según se muestra en la figura 7.26.

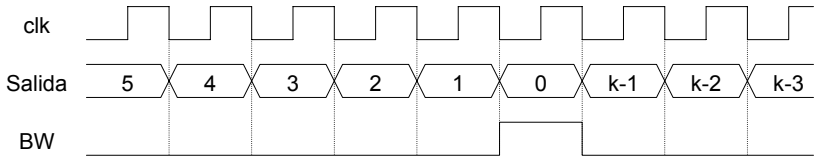


Figura 7.26.

Los contadores reversibles disponen de una señal de salida *Terminal Count* (TC) que se pone a 1 si se alcanza el estado de cuenta más alto si el contador tiene el modo ascendente (CY) o se activa en el estado de cuenta 0 si el contador está programado como descendente (BW).

7.2.2. Contadores asíncronos

Otro diseño alternativo al contador módulo 8 del principio del apartado 7.2.1 es el que se muestra en la figura 7.27.

Según se desprende de las formas de onda de salida del contador:

- O_0: cambia en los flancos de bajada de clk, por lo que el biestable 0 tiene su entrada T igual a 1 y su entrada de reloj igual a la señal de reloj clk.
- O_1 es a O_0 como O_0 es a clk. Esto es, si O_0 fuese considerado como una señal de reloj, entonces la entrada del biestable T puede estar a 1. Con

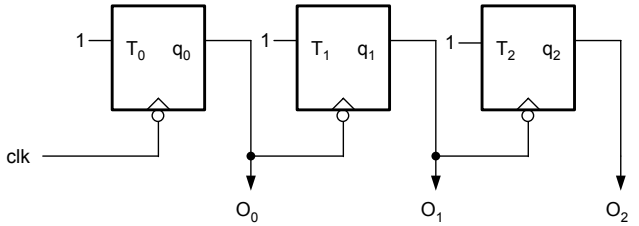


Figura 7.27.

esto conseguimos que este biestable siempre cambie en los flancos de bajada de O_1.

- De igual forma, O_2 cambia en los flancos de bajada de O_1, por lo que la entrada de reloj del biestable 2 es O_1 y su entrada T es un 1.

Este tipo de diseño alternativo se denomina también **contador de rizado** o ***ripple-counter***, por la especie de rizo que hace la conexión de la salida de un biestable a la entrada de reloj del siguiente biestable. La estructura de este contador es más simple que la del síncrono, por no necesitar de puertas lógicas adicionales, sin embargo presenta algunos inconvenientes, como la velocidad de operación (que es menor en este caso) y la aparición de estados de cuentas fantasmas. Analicemos este último aspecto. Consideremos la situación realista de que los biestables del contador de rizado anterior tienen un tiempo de propagación no nulo, igual a t_p. Como se muestra en la figura 7.28, el contador va a pasar del estado de cuenta 3 al 4. Al usarse la salida de un biestable como reloj del siguiente, los retrasos van acumulándose de una etapa a otra, por lo que la salida de cuenta verdadera, tarda un tiempo, en este caso, igual a $3\,t_p$. En el intervalo de tiempo comprendido entre 0 y $3\,t_p$ se han producido cuentas transitorias erróneas.

Los contadores asíncronos comerciales disponen de señales de control como *Reset*, *Load*, etc. todas ellas con funcionamiento asíncrono (ya descrito en el apartado anterior).

7.2.3. Contadores con módulo diferente a una potencia de 2

Es habitual que en la práctica se necesiten contadores con un módulo diferente a la potencia de dos (en el mercado es muy común encontrarse con

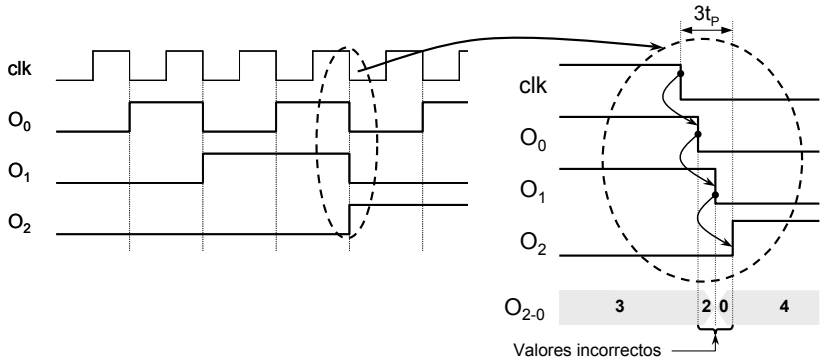


Figura 7.28.

contadores módulo 10 por la importancia que tiene esta base de numeración). Si se desea construir un contador con un módulo distinto a lo que se ofrece comercialmente, tenemos dos posibilidades:

- Diseñarlo con biestables y puertas, como si se tratara de una máquina secuencial síncrona
- Usando contadores y puertas. Disponiendo de un contador con un módulo mayor del que se desea diseñar, y puertas lógicas, podemos hacer que este se comporte contando sólo aquellos estados de interés.

Aquí se desarrollará el segundo método; remitimos al lector al tema 7 para el diseño según el primero.

Ejemplo. Se desea construir un contador módulo 10, que cuente desde el 0 hasta el 9, usando un contador módulo 16 y puertas lógicas.

Es lógico que el contador módulo 16 usado para implementar el módulo 10 debe interrumpir su cuenta cuando llega al estado de cuenta 9. Esto es, el contador módulo 16 pasaría, después del 9, al estado de cuenta 10, pero debemos obligarle a que, en lugar del 10, pase al estado de cuenta 0. Esto es posible si el contador dispone de línea de *Clear*. De alguna manera, esta línea debe de activarse cada vez que sea necesario para forzar el paso del estado de cuenta 9 al 0. Existen dos alternativas de diseño, en función de que la línea de *Clear* tenga un modo de funcionamiento asíncrono o síncrono. Analizaremos las dos situaciones.

***Clear* síncrono (activo en bajo)**. Aquí se plantea diseñar un circuito combinacional que en función del estado de cuenta del contador módulo 16 active la señal de *Clear*, Cl. Como esta es síncrona, debe activarse en el estado de cuenta 9, para que el próximo estado, en el siguiente flanco activo de clk, sea 0. Véase la figura 7.29.

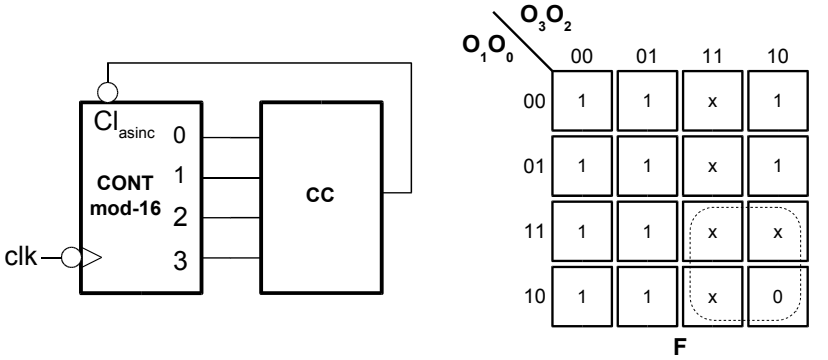


Figura 7.29.

Por tanto, $Cl = \overline{O_3\, O_0}$.

En la figura 7.30 se muestra un cronograma donde se aprecia el funcionamiento del conjunto contador-puerta. Se observa que en el estado de cuenta 9, la señal Cl se pone a 0, lo que obliga a que el contador se ponga a 0 en el siguiente ciclo de reloj.

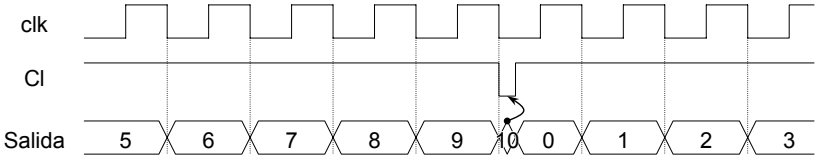


Figura 7.30.

***Clear* asíncrono (activo en bajo)**. El procedimiento con el clear asíncrono es similar que con el síncrono, salvo que este caso la señal de clear debe activarse en el estado de cuenta 10. Una vez que el circuito combinacional detecta la llegada de este estado, activa la señal de *Clear*, Cl, lo que provoca que inmediatamente el contador se ponga a 0 (sin esperar el flanco activo de clk), figura 7.31

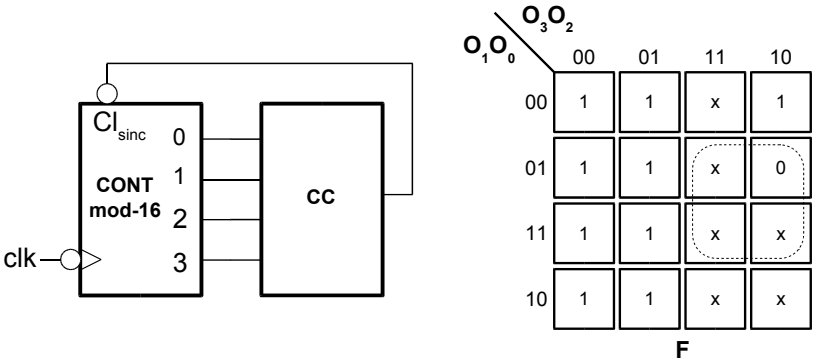


Figura 7.31.

Por tanto, $Cl = \overline{O_3\, O_1}$.

La figura 7.32 muestra un cronograma del funcionamiento de este contador, donde se ve que el estado de cuenta 10 aparece durante un tiempo pequeño, el necesario para la ejecución de la operación *Clear*.

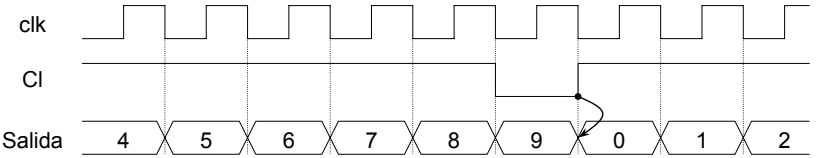


Figura 7.32.

7.2.4. Contadores de anillo y conmutado en cola

La figura 7.33 muestra la estructura de un **contador en anillo** de módulo 4. Éste está constituido por 4 biestables, conectados entre sí de modo que la salida de uno es la entrada del siguiente y así sucesivamente hasta llegar al último biestable, cuya salida se reintroduce por la entrada del primero. Esto da idea del nombre de contador en anillo. En este contador, el estado de cuenta j, viene determinado por un 1 en la salida O_j (mientras que el resto está a 0).

La línea $Init$ sirve para inicializar el circuito, de forma que cuando esta vale 0, los biestables 0, 1, 2 y 3 toman, de forma asíncrona, los valores 1, 0, 0 y 0 respectivamente. La puesta a 1 de $Init$ sirve como indicador para la puesta en

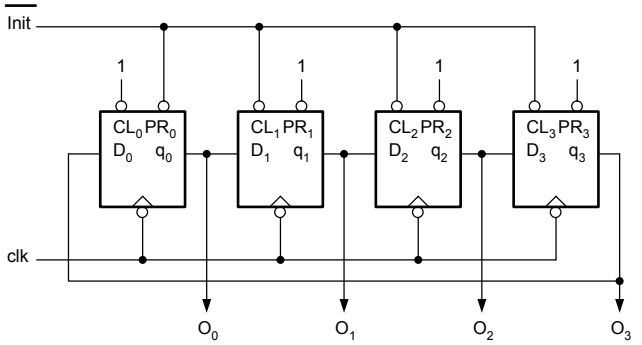


Figura 7.33.

marcha del contador. La figura 7.34 muestra el cronograma de funcionamiento de este contador.

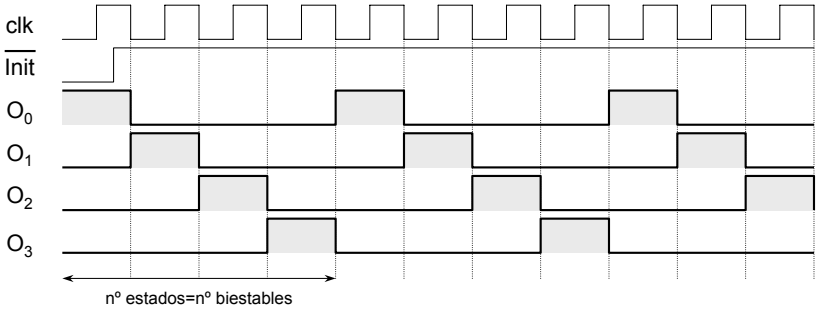


Figura 7.34.

En el ciclo de reloj posterior a la inicialización, el biestable 1 captura el 1 que tiene el biestable 0, mientras que éste, junto con los biestables 2 y 3, se ponen a 0. En el siguiente ciclo, el biestable 2 captura el 1, los restantes están a 0, y así sucesivamente. En definitiva, existe un único 1 que en cada ciclo de reloj va pasando de un biestable a otro.

Una modificación al contador anterior, la constituye el **anillo de Johnson** o **contador conmutado en cola**, el cual permite un número mayor de estados. En concreto, si este nuevo contador tiene n biestables, el número de estados de cuenta es de $2\,n$. Sin embargo, este contador, a diferencia del de anillo, no muestra de forma evidente el estado de cuenta. La figura 7.35 muestra la estructura de un contador de anillo conmutado en cola o contador de Johnson.

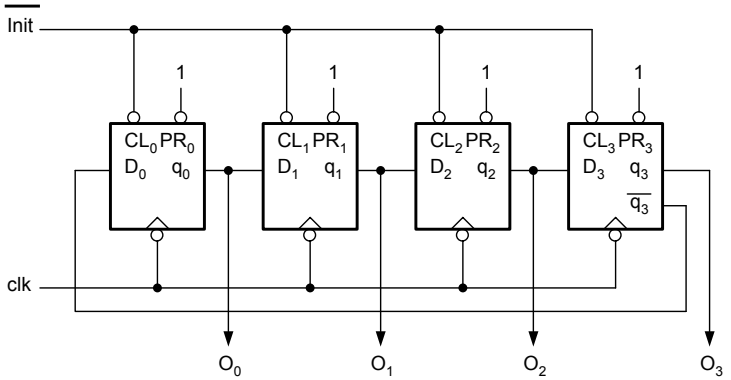


Figura 7.35.

Son dos las diferencias con el contador en anillo:

1. La inicialización. Todos los biestables del contador se inicializan con 0.

2. Las entrada de cada biestable está conectada con la salida del biestable anterior, salvo el primero, cuya entrada es la salida complementada del último biestable, el que está en la cola (de ahí el nombre).

Las salidas de este contador se muestram en la figura 7.36. Si inicialmente todos los biestables están a 0, en el siguiente ciclo de reloj, el biestable 0, se carga con un 1, ya que su entrada es el complemento del contenido del biestable situado en cola. En el siguiente ciclo de reloj este 1 pasa al biestable 1, mientras que el biestable 0 sigue cargándose con un 1 (mientras que $q_3 = 0$). Este proceso se repite hasta que por fin, todos los biestables están a 1. En el siguiente ciclo de reloj, el biestable 0 se carga con un 0 (al ser $q_3 = 1$). A continuación este 0 pasa al biestable 1, en el siguiente ciclo de reloj, mientras que el biestable 0 sigue cargándose con un 0. Esto se va repitiendo hasta que finalmente todos los biestables se encuentran a 0, el punto de partida. A partir de aquí se repite toda la secuencia. Se ve claramente, que todos los estados posibles son 8, justamente el doble del número de biestables.

Figura 7.36.

7.3. Registros

Un registro de n bits es un dispositivo que tiene capacidad de almacenar n bits. Internamente están formados por biestables, tantos como bits sea capaz de almacenar el registro. Normalmente estos dispositivos son síncronos siendo los biestables D los más usados para la implementación interna.

En cuanto a las operaciones básicas que se realizan sobre los registros destacamos fundamentalmente dos: **escritura** (W) y **lectura** (R), aunque habitualmente se pueden encontrar registros que incorporan operaciones como *Clear*.

Los registros se pueden clasificar en función de cómo se lean o escriban los bits, así podemos encontrar:

- registros con entrada serie (entrada hace referencia a escritura) y salida serie (salida hace referencia a lectura)
- registros con entrada serie y salida paralelo
- registros con entrada paralelo y salida serie
- registros con entrada paralelo y salida paralelo.

Si disponemos de un registro de n bits, diremos que si éste tiene entrada serie, entonces el registro sólo tiene una línea de entrada de datos por la que, uno tras otro, se introducen los n bits que serán almacenados por el registro. Si un registro de n bits se dice que tiene entrada paralelo, entonces dicho registro dispone de n líneas de entrada, una por cada bit, por la que se introducen

simultáneamente los n bits al registro. Estos conceptos de entrada serie y entrada paralelo son igualmente aplicables para la salida serie y salida paralelo. Un registro de n bits tiene salida serie, implica que tiene una única línea de salida por la que, uno tras otro, van saliendo los n bits almacenados en el registro. Si un registro de n bits tiene salida paralelo, implica que dispone de n salidas, una por cada bit, de forma que simultáneamente se leen todos los bits del registro.

Todos los registros que tengan algún modo de funcionamiento serie para lectura o para escritura se denominan registros de desplazamiento (*shift registers*). Podemos encontrar dos tipos de registros de desplazamiento en función del sentido de movimiento de los bits: izquierda o derecha.

7.3.1. Registro de entrada serie y salida serie

La estructura básica de un registro de 4 bits de entrada serie y salida serie se muestra en la figura 7.37.

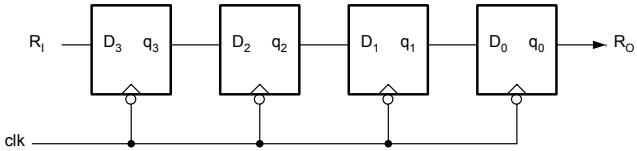


Figura 7.37.

Para todos los registros de desplazamiento es necesario que los biestables D que lo constituyen sean disparados por flanco. Sólo se salva de esta regla el registro de entrada paralelo y salida paralelo, el cual suele estar formado por biestables disparados por nivel.

Se observa en el registro de la figura 7.37, que los biestables están conectados de forma que la salida de cada uno se corresponde con la entrada del biestable situado inmediatamente a su derecha. Asimismo, la salida del biestable situado a la derecha del conjunto se corresponde con la salida del registro serie, mientras que la entrada para el biestable situado a la izquierda del conjunto se correponde con la entrada de datos del registro.

Para que el registro opere de forma adecuada, los bits de entrada deben

estar sincronizados con la señal de reloj, esto es, un bit de entrada por cada ciclo de reloj. El procedimiento de escritura–lectura se ilustra en la figura 7.38. El primer bit de entrada, B_0, se escribirá en el biestable D_3 en el primer flanco activo de reloj. En el segundo flanco activo, el bit B_0 pasa el biestable D_2 al tiempo en que el segundo bit de entrada, B_1, se escribe en D_3, sobreescribiendo su valor anterior (B_0). Del mismo modo se escriben los bits B_2 y B_3 en el registro. Al finalizar el proceso de escritura, los biestables D_{3-0} contienen a los bits B_{3-0}.

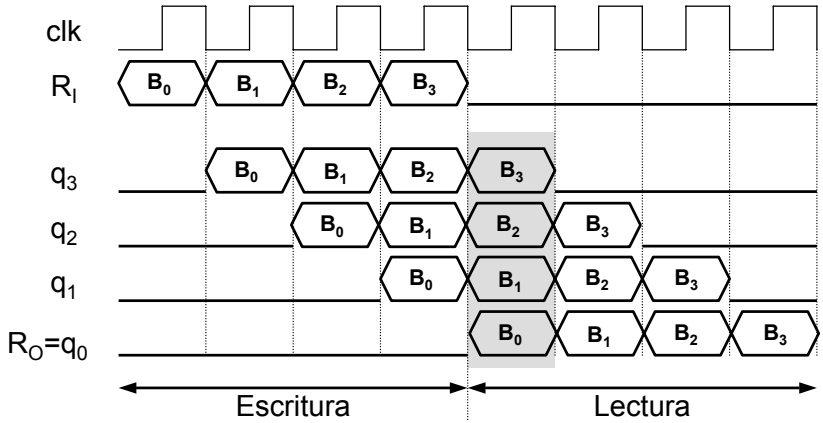


Figura 7.38.

Para la lectura serie el procedimiento es similar. Sólo hay que tener en cuenta que para leer el primer bit almacenado, el B_0, no es necesario esperar ningún ciclo de reloj, ya que la salida q del último biestable (y por tanto el bit B_0), se encuentra conectada con la salida del registro.

Por todo lo visto, podemos decir que el registro de entrada serie y salida serie presentado en este apartado es un registro de desplazamiento a derecha. El registro de desplazamiento a izquierda se hubiera construido de forma similar sin más que conectar la salida de un biestable con la entrada del biestable situado a su izquierda. La entrada del registro sería en este caso D_0, y la salida, q_3.

7.3.2. Registro con entrada serie y salida paralelo

La figura 7.39 muestra la estructura básica de un registro de entrada serie y salida paralelo de 4 bits. Al igual que en el apartado anterior, éste es un registro de desplazamiento a derecha:

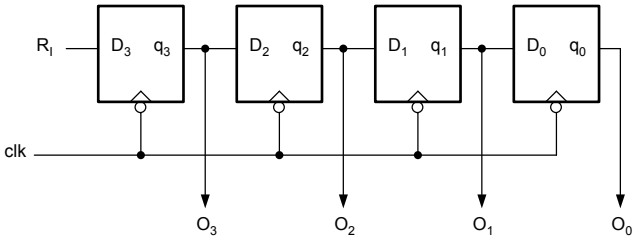


Figura 7.39.

El procedimiento de escritura es el mismo que el descrito en el apartado anterior. En cuanto a la lectura, se hace evidente que el registro no depende de la señal de reloj para mostrar su contenido, éste siempre aparece en las líneas de salida.

7.3.3. Registro con entrada paralelo y salida serie

La figura 7.40 ilustra la estructura básica de un registro de estas características.

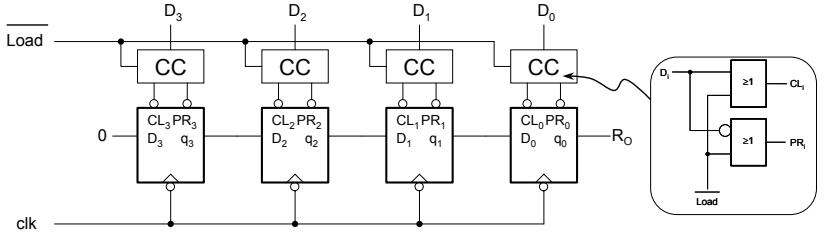


Figura 7.40.

El procedimiento de lectura es igual que el del registro serie–serie, para lo que se hace necesario que los biestables esten conectados entre sí, salida de uno con la entrada del siguiente (en este ejemplo, el registro de desplazamien-

to es nuevamente a la derecha). Aquí se modifica el proceso de escritura, que se realiza cuando la línea $Load$ está activa.

En la escritura en paralelo, todos los bits de entrada A_i, se cargarán simultáneamente en los biestables q_i. En la figura se ha escogido un procedimiento de carga asíncrona, usando las entradas Cl y Pr de los biestables. Se deja al lector el diseño de la estructura para un procedimiento de carga o escritura síncrona. Se ha diseñado un circuito combinacional (CC), formado por dos puertas NAND y un inversor, que controla las líneas de Cl y Pr de cada biestable en función del bit a escribir, A_i, y de la señal de control $Load$. Cuando ésta última está a 0, las entradas Cl y Pr están a 1, no ocurre nada. Si $Load = 1$, entonces si $A_i = 0$, se activa Cl y si $A_i = 1$, se activa Pr, esto es, escritura asíncrona de un 0 o un 1, respectivamente. Este mecanismo de escritura es el mismo para todos los biestables del registro.

7.3.4. Registro con entrada paralelo y salida paralelo

La figura 7.41 ilustra la estructura de un registro con entrada y salida en paralelo de 4 bits.

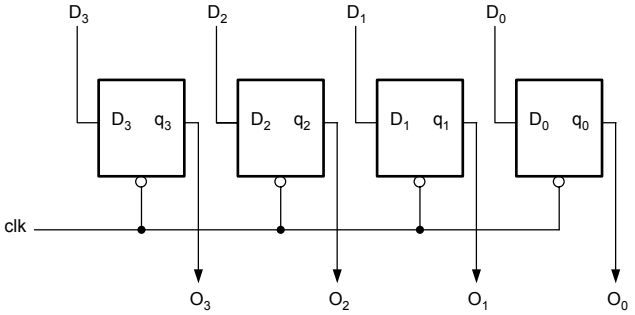


Figura 7.41.

Como se observa, los biestables son del tipo disparado por nivel (bajo, en el circuito mostrado en la figura) en lugar de disparado por flanco. Esto suele ser habitual en este tipo de registros, denominados también como ***latch***, aunque no se descarta la posibilidad de que se puedan diseñar con biestables disparados por flanco.

Si la señal clk vale 0, los biestables capturan simultáneamente todos los bits de entrada. Si clk vale 1, los biestables mantienen la información capturada que muestran siempre por sus líneas de salida.

7.3.5. Registro Universal

Un registro universal es aquel que tiene todas las formas de lectura y escritura posibles, tanto en serie como en paralelo.

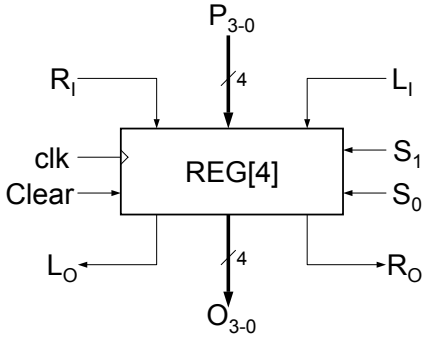
Ejemplo. Diseñar un registro universal de 4 bits que tenga las siguientes operaciones: desplazamiento a la derecha, desplazamiento a la izquierda, carga en paralelo y puesta a cero.

De las cuatro operaciones anteriores, las de desplazamiento a derecha y a izquierda, forzosamente tienen que ser síncronas. La carga (*Load*) y la puesta a cero (*Clear)* pueden ser síncronas o asíncronas. Supongamos que la carga es síncrona, y la puesta a cero, asíncrona. Tenemos un total de 4 operaciones a realizar más la de inhibición[1] que debe estar presente en cualquier diseño de registro ya que ésta es la que consiste en mantener la información memorizada. Está claro que para implementar la operación asíncrona se necesita que los cuatro biestables que forman parte del registro tengan entradas asíncronas, en este caso de Cl, la cual se conecta directamente con la entrada de $Clear$ del registro. Las otras cuatro operaciones síncronas se codifican en dos líneas de control, S_1 y S_0, tal como se muestra en la siguiente figura 7.42.

Para la escritura o carga en paralelo, el registro ha de disponer de cuatro líneas de entrada, P_{3-0}. Para la lectura en paralelo, el registro dispone de cuatro líneas de salida, O_{3-0}. R_I y R_O son las líneas de entrada y salida, respectivamente, para la operación de desplazamiento a la derecha (R viene de *Right*, derecha). L_I y L_O son las líneas de entrada y salida, respectivamente, para la operación de desplazamiento a la izquierda (L viene de *Left*, izquierda). Por último, es necesaria la señal de reloj, clk.

Si $S_1\,S_0 = 00$, desplazamiento a la derecha, las entradas de los cuatro biestables del registro deben ser (teniendo en cuenta que D_3 el registro situado

[1] También denominada NOP (*No OPeration*)



S_1	S_0	Operación
0	0	SHR
0	1	Load
1	0	Inhibición (NOP)
1	1	SHL

Figura 7.42.

más a la izquierda y D_0 el situado a más la derecha):

$$\begin{aligned} D_3 &= R_I \\ D_2 &= q_3 \\ D_1 &= q_2 \\ D_0 &= q_1 \\ R_O &= q_0 \end{aligned}$$

Si $S_1\,S_0 = 01$, carga en paralelo, las entradas deben ser los valores que tengan las entradas en paralelo del registro, P_{3-0}:

$$\begin{aligned} D_3 &= P_3 \\ D_2 &= P_2 \\ D_1 &= P_1 \\ D_0 &= P_0 \end{aligned}$$

Si $S_1\,S_0 = 10$, inhibición, las entradas de los biestables coincidir con sus propios valores almacenados, para que éstos no cambien:

$$\begin{aligned} D_3 &= q_3 \\ D_2 &= q_2 \\ D_1 &= q_1 \\ D_0 &= q_0 \end{aligned}$$

Si $S_1\,S_0 = 11$, desplazamiento a la izquierda, las entradas de los biestables

deben ser:

$$\begin{aligned} L_O &= q_3 \\ D_3 &= q_2 \\ D_2 &= q_1 \\ D_1 &= q_0 \\ D_0 &= L_I \end{aligned}$$

En cualquier caso, las salidas O_{3-0} son los valores de los biestables q_{3-0}.

Combinando los casos anteriores, obtenemos las siguientes ecuaciones:

$$\begin{aligned} D_3 &= \bar{S}_1\bar{S}_0R_I + \bar{S}_1S_0B_3 + S_1\bar{S}_0q_3 + S_1S_0q_2 \\ D_2 &= \bar{S}_1\bar{S}_0q_3 + \bar{S}_1S_0B_2 + S_1\bar{S}_0q_2 + S_1S_0q_1 \\ D_1 &= \bar{S}_1\bar{S}_0q_2 + \bar{S}_1S_0B_1 + S_1\bar{S}_0q_1 + S_1S_0q_0 \\ D_0 &= \bar{S}_1\bar{S}_0q_1 + \bar{S}_1S_0B_0 + S_1\bar{S}_0q_0 + S_1S_0L_I \end{aligned}$$

las cuales se pueden implementar según se muestra en la figura 7.43.

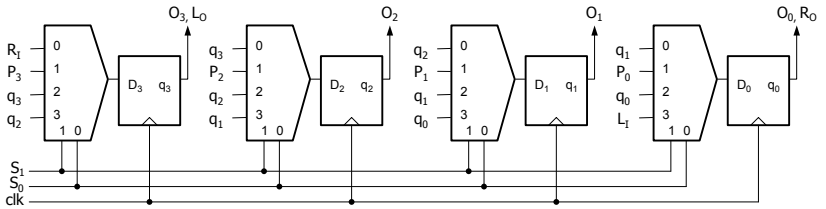


Figura 7.43.

7.4. PLDs secuenciales

En el capítulo 4 se estudiaron los dispositivos lógicos programables (PLD, *Programmable Logic Device*) de tipo combinacional, esto es, PLA y PAL. Los PLDs secuenciales se basan en las mismas estructuras que aquéllos, con la novedad de que incorporan biestables cuyas entradas y salidas están conectadas a la matriz programable. Gracias a ello, usando PLDs secuenciales podemos implementar funciones de conmutación tanto combinacionales como

secuenciales con un número máximo de estados que depende de la cantidad de biestables disponibles en el PLD.

A continuación, y a modo de caso ilustrativo, se estudia detalladamete la PAL secuencial 22V10, figura 7.44. Ésta dispone de once entradas fijas I_{1-11}, una entrada especial I_0 que, además de alimentar a la matriz programable, sirve de señal de reloj a los biestables de la PAL, y diez líneas que pueden ser programadas como entradas o salidas I/O_{0-9}. Por supuesto, también incorpora las entradas de alimentación, V_{CC} y GND.

La diferencia fundamental respecto de una PAL combinacional es la estructura de salida, que incluye, como novedad, los siguientes componentes:

- un biestable tipo D, disparado por flanco, con entradas asíncronas de *Clear*, AR, y *Preset*, SP, programables en la matriz;
- un multiplexor de cuatro canales que determina la salida de la macrocelda al pin I/O correspondiente, cuyas entradas de selección S_1 y S_0 son programables;
- un multiplexor de dos canales que controla la realimentación desde la macrocelda a la matriz programable, cuya entrada de selección está compartida con la línea S_1 del multiplexor de cuatro canales.

Todo ello permite que las etapas de salida, que reciben el nombre de **macroceldas** (*macrocells*) sean muy versátiles, pudiendo funcionar en cuatro modos distintos según se programen las entradas de selección de los multiplexores. La estructura de la macrocelda de la PAL 22V10 se muestra en la figura 7.45.

Si, por ejemplo, programamos una macrocelda de modo que las entradas de selección S_1 y S_0 (nótese que S_1 es compartida por los dos multiplexores) queden conectadas al nivel lógico 0 (manteniendo intactos los fusibles correspondientes, lo cual se expresaba colocando un aspa en la interconexión), dicha macrocelda quedará configurada de forma que el biestable tendrá su entrada D conectada a la matriz programable, y su salida Q aparecerá complementada en el pin I/O correspondiente, tal como se muestra en la figura 7.46.

Los cuatro modos de operación de la macrocelda se ilustran en la figura 7.47, y corresponden a las cuatro combinaciones posibles que pueden presentar las líneas programables S_1 y S_0 de la macrocelda:

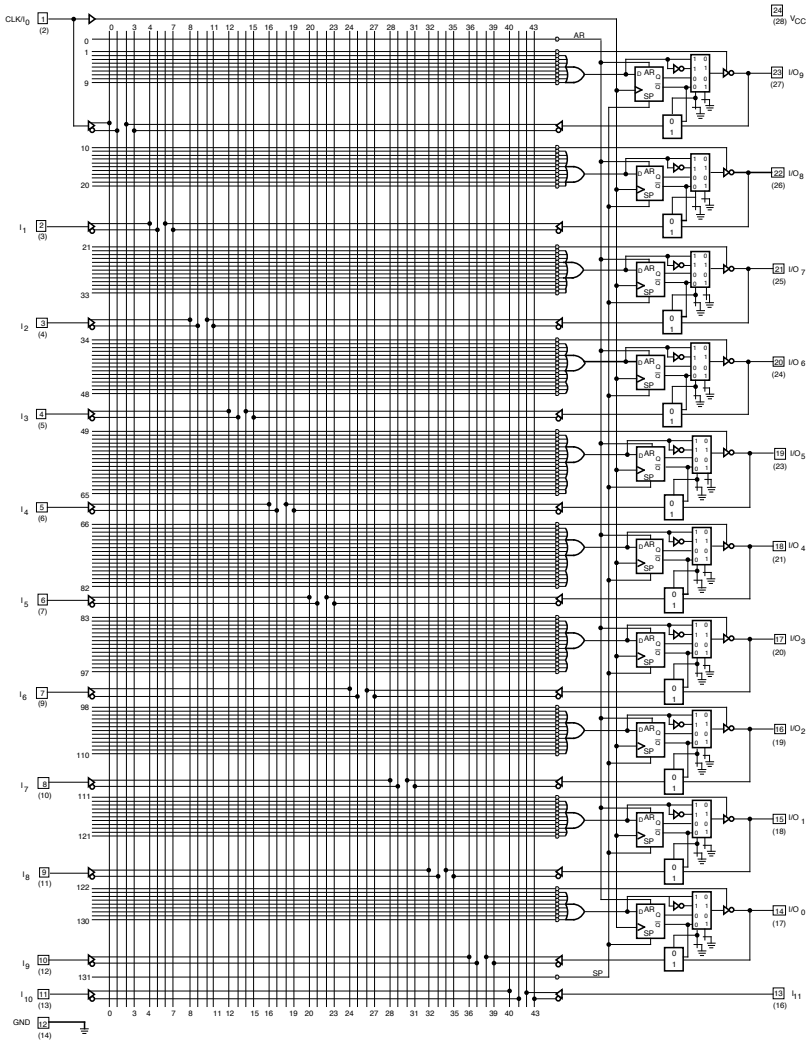
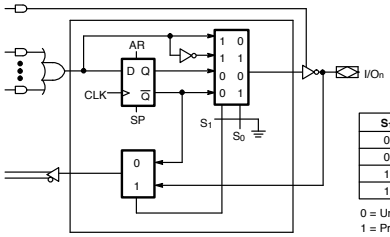

CLK/I0
I1
I2
I3
I4
I5
I6
I7
I8
I9
I10
GND
AR
SP
VCC
I/O9
I/O8
I/O7
I/O6
I/O5
I/O4
I/O3
I/O2
I/O1
I/O0
I11

Figura 7.44.



S_1	S_0	Output Configuration
0	0	Registered/Active Low
0	1	Registered/Active High
1	0	Combinatorial/Active Low
1	1	Combinatorial/Active High

0 = Unprogrammed fuse
1 = Programmed fuse

Figura 7.45.

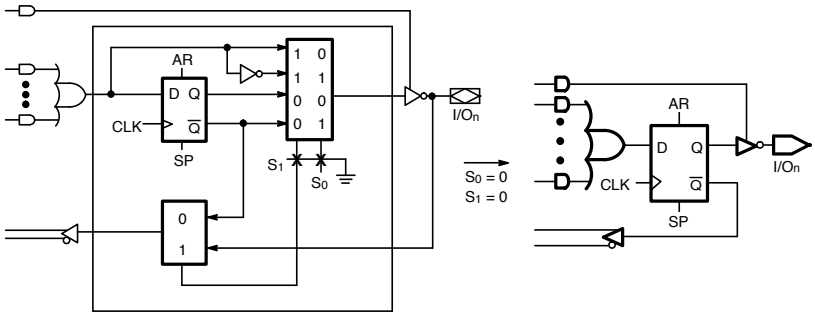


Figura 7.46.

(a) ***Registered/Active LOW***: la macrocelda opera en modo secuencial (*registered*), esto es, el biestable está conectado a la salida, que, en este caso, está complementada (*active low*);

(b) ***Registered/Active HIGH***: la macrocelda opera en modo secuencial y la salida no está complementada (*active high*);

(c) ***Combinatorial/Active LOW***: la macrocelda opera en modo combinacional (*combinatorial*), esto es, no se usa el biestable. En este caso, la salida está complementada;

(d) ***Combinatorial/Active HIGH***: la macrocelda opera en modo combinacional; la salida no está complementada.

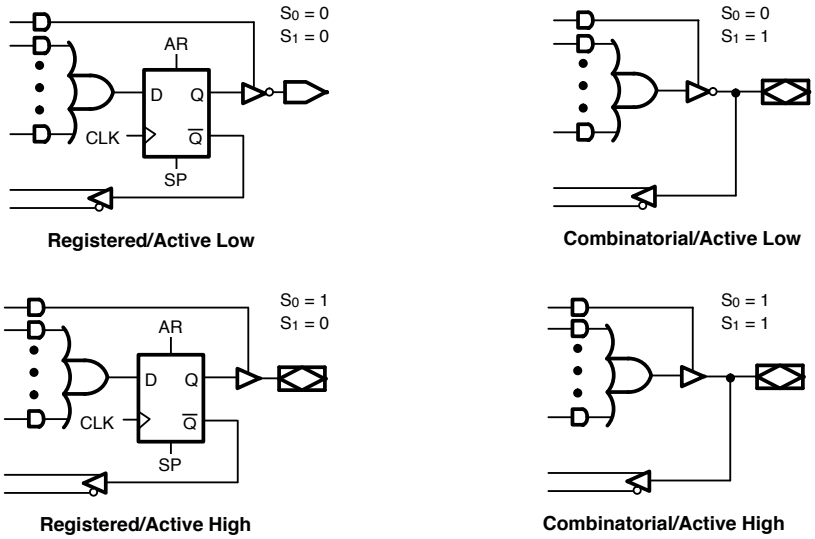


Figura 7.47.

Al igual que en una PAL combinacional, si el inversor triestado que interconecta la macrocelda con el pin I/O correspondiente está programado de forma que su salida está en alta impedancia, la macrocelda queda desconectada del pin I_O y éste puede ser usado como entrada adicional.

Ejercicios propuestos

Problema 7.1 Diseñar un circuito secuencial síncrono que genere la secuencia periódica 100101 por su salida Z. Base su diseño en un registro de desplazamiento y puertas lógicas.

Problema 7.2 Diseñe la etapa típica de un registro universal de 8 bits. Base su diseño en biestables D y MUX 2:1.

Problema 7.3 Diseñar la etapa típica de un registro de desplazamiento bidireccional con carga en paralelo síncrona y reset asíncrono. Utilice biestables T y multiplexores 2:1.

Problema 7.4

a)Utilizando un contador ascendente síncrono de módulo 16 que , además, dispone de operación de reset asíncrona y carga síncrona, y puertas lógicas, diseñe un contador que cuente de : 1) de 0 a 6; 2) de 7 a 15; 3) de 4 a 10.

b) Diseñe la etapa típica del contador con biestables T y multiplexores.

Problema 7.5

(a) Diseñar un contador ascendente de módulo 16 que disponga de entradas de cuenta y puesta a 0, ambas síncronas y activas en alto. Para ello utilice exclusivamente los siguientes subsistemas:

(b) Explique qué modificaciones tendría que hacer sobre el diseño del apartado (a) para que la cuenta se incrementase de 2 en 2 (0, 2, 4, 6, ...). ¿Cúantos estados distintos tendría el contador resultante?.

(c) Explique qué modificaciones tendría que hacer sobre el diseño del apartado (a) para que la cuenta se incrementase de 3 en 3 (0, 3, 6, 9, ...). ¿Cúantos estados distintos tendría el contador resultante?.

(d) Obtenga un contador módulo 15 a partir del contador de módulo 16 diseñado en el apartado (a).

Problema 7.6 Diseñe un circuito digital que a partir de una señal de reloj de 24 horas de duración, muestre en displays de 7 segmentos, el número del mes. Se

entiende que cada mes tiene 30 días y el primer mes es el 1 (Enero) y Diciembre el 12. Base su diseño en contadores módulo-32, módulos combinacionales y puertas.

Problema 7.7 Un determinado edificio dispone de sólo dos puertas de acceso, una de entrada y otra de salida. En cada puerta se ha situado un sensor que genera un pulso activo en alta con una duración de 1μ s, cada vez que una persona pasa por la puerta. Usando como base un contador de 12 bits que se incrementa cada vez que se una persona entra al edificio y se decrementa cuando otra sale, diseñe un circuito secuencial con una entrada, X, y dos salidas, Z_1, Z_0, de forma que la salida Z_1 se activa si, para $X = 0$ en el edificio hay más de 500 personas, o si para $X = 1$ en el edificio hay más de 1000 personas. En cualquier caso, Z_0, se activa siempre que no hay nadie en el interior del edificio. El circuito a diseñar debe disponer de una entrada adicional, R, que ponga a cero el contador. Diseñe el circuito usando los siguientes componentes, un contador de 12bits, comparadores de magnitud de 4 bits, multiplexores de 2 canales y puertas lógicas.