Circuitos combinacionales. Álgebra de Boole

Salvador Marcos González
salvador.marcos@uah.es

Sistemas de numeración

Introducción

Un *Sistema de numeración* es una forma de representar cualquier cantidad numérica, de forma que una misma cantidad se puede escribir de muchas formas distintas, según sea el sistema de numeración utilizado. Así, el sistema utilizado normalmente por el hombre es el sistema decimal o de "base 10", mientras que el sistema usado internamente por las máquinas electrónicas actuales es el binario o de "base 2".

Casi todos los sistemas de numeración utilizados en la actualidad son de tipo "polinomial". Un sistema de numeración polinomial cumple las siguientes características generales:

- todo número es una expresión formada por un conjunto de símbolos, llamados "**dígitos**" o "cifras", cada uno con un valor propio, fijo y diferente del de los demás.
- la cantidad de dígitos distintos que se pueden usar en un determinado sistema de numeración constituye su "**base**".
- el valor de un número depende de dos factores: del valor de los dígitos que lo componen y de la posición de cada uno de ellos dentro del conjunto.
- cada posición del número tiene un valor intrínseco que aumenta de derecha a izquierda según potencias sucesivas de la base del sistema de numeración. Así, el dígito del extremo derecho es el de menor peso, y el dígito del extremo izquierdo es el de mayor significación.
- se verifica el llamado "**Teorema fundamental de la numeración**":

  El valor numérico en base 10 de un número expresado en cualquier otra base "b" se obtiene como suma de funciones potenciales de dicha base, según la siguiente expresión:

$$N_{(b} = a_n\, a_{n-1}.....a_2\, a_1\, a_0 \,/\, a_{-1}\, a_{-2}\, a_{-3}...=$$
$$a_n * b^n + a_{n-1} * b^{n-1} + ...+ a_2 * b^2 + a_1 * b^1 + a_0 * b^0 + a_{-1} * b^{-1} + a_{-2} * b^{-2} + a_{-3} * b^{-3} ...$$

En esta expresión, los coeficientes "a_i" son los digitos del número, y "b" es la base del sistema de numeración. Las potencias "b^i" son los valores intrínsecos de cada posición del número. El valor de la primera posición entera es siempre 1 (b^0).

El dígito "cero" (0) es el dígito de valor propio nulo. El valor de un número no se altera si se añaden ceros ala izquierda de la parte entera, o a la derecha de la parte decimal.

Los sistemas de numeración polinomiales más usados en la práctica son el Decimal (base 10), el Binario (Base 2), el Octal (base 8) y el Hexadecimal (Base 16). Un ejemplo de sistema no polinomial es el sistema de numeración romano.

El sistema binario

Es el sistema de **base 2**, y utiliza dos dígitos distintos, el 0 y el 1, de nominados normalmente con el nombre de "**bit**".

Es el sistema utilizado normalmente en las actuales máquinas electrónicas digitales. La razón de ello es que es muy facil diseñar sistemas físicos o electrónicos capaces de adoptar dos valores o posiciones distintas (0 y 1).

Un número binario estará formado por un conjunto de bits. El valor de cada posición del número aumenta de derecha a izquierda según potencias de 2. Las primeras potencias de dos son las siguientes:

Posición	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Valor	256	128	64	32	16	8	4	2	1
Posicional	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Con un número binario de "*n*" bits se pueden representar 2^n números distintos, desde el 0 hasta el 2^{n-1}. La tabla de los primeros 16 números binarios se muestra en la siguiente página.

decimal	binario
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
10	1 0 1 0
11	1 0 1 1
12	1 1 0 0
13	1 1 0 1
14	1 1 1 0
15	1 1 1 1

Para sumar dos números binarios, se comienza por la posición del extremo derecho, aplicando las 5 reglas de la suma de dos bits:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 (Acarreo *+1*)
1 + 1 *+1* = 1 (Acarreo *+1*)

Resolución de ejemplos

Ejemplo 1:

1 1 0 1 13
1 1 0 0 + 12 +
---
1 1 0 0 1 25

El sistema octal

Es el sistema de **base 8**, y utiliza ocho dígitos distintos: del 0 al 7. Se utiliza este sistema con frecuencia de bido a su facilidad de conversión con el sistema binario, lo cual hace que números binarios muy grandes se manejen más comodamente en octal. Esta facilidad de conversión se debe en última instancia a que la base 8 es potencia de 2 ($8=2^3$).

La conversión de binario a octal se realiza de la forma siguiente:

Se agrupan los bits del número binario, tanto enteros como decimales, en *grupos de 3* a partir del punto decimal. El valor en base 10 de cada uno de esos grupos da lugar a un dígito octal.

Por ejemplo: $100011101000'0101111_{(2} = 14350'274_{(8}$

1 4 3 5 0 2 7 4

La conversión de octal a binario es justamente el camino inverso (sustituir cada cifra octal por el grupo de 3 bits equivalente).

Para sumar dos números en octal, se comienza por la columna del extremo derecho. Cuando la suma en alguna columna sobrepase el valor de 7, se anota como resultado de dicha columna el valor de aquella suma menos 8 (la base del sistema), y se acarrea una unidad a la siguiente columna de mayor orden.

Resolución de ejemplos

Ejemplo 1:

7 3 1 ' 4 (8	473'5 (10
3 5 0 ' 2 (8 +	232'25 (10
1 3 0 1 ' 6 (8	705'75 (10

El sistema hexadecimal

Es el sistema de **base 16**, y utiliza 16 dígitos distintos: del 0 al 9 más las letras mayúsculas A,B,C,D,E y F que tienen como valores propios 10, 11, 12, 13, 14 y 15 respectivamente.

Es usado más frecuentemente que el Octal, y por la misma razón que éste, es decir, por su facilidad de conversión con el binario. La razón matemática de esto es que $16 = 2^4$.

La conversión de binario a hexadecimal se realiza ahora agrupando los bits *en grupos de 4* a partir del punto decimal. Cada uno de esos grupos da lugar a un dígito hexadecimal.

Por ejemplo: $1100100011100101 0'000011011_{(2} = 191CA'OD8_{(16}$

1 9 1 C A O D 8

La conversión de hexadecimal a binario consiste en sustituir cada dígito hexadecimal por el grupo de 4 bits equivalentes, según indica la tabla de los 16 primeros números:

decimal	binario	octal	hex
0	0 0 0 0	0	0
1	0 0 0 1	1	1
2	0 0 1 0	2	2
3	0 0 1 1	3	3
4	0 1 0 0	4	4
5	0 1 0 1	5	5
6	0 1 1 0	6	6
7	0 1 1 1	7	7
8	1 0 0 0	10	8
9	1 0 0 1	11	9
10	1 0 1 0	12	A
11	1 0 1 1	13	B
12	1 1 0 0	14	C
13	1 1 0 1	15	D
14	1 1 1 0	16	E
15	1 1 1 1	17	F

Cambios de base

Para pasar un número expresado en cualquier base a base 10, se aplica el *Teorema fundamental* de la numeración ya vista (suma de funciones potenciales de la base).

Para pasar un número entero de base 10 a cualquier otra base se realizan sucesivas divisiones enteras de dicho número por la base a la que se quiere cambiar, obteniéndose el resultado a partir del último cociente y todos los restos (las divisiones han de ser enteras, es decir, con un cociente y un resto).

Resolución de ejemplos

Ejemplo 1: Pasar $533_{(10}$ a binario y a hexadecimal.

$533_{(10} = 1000010101_{(2} = 215_{(16}$

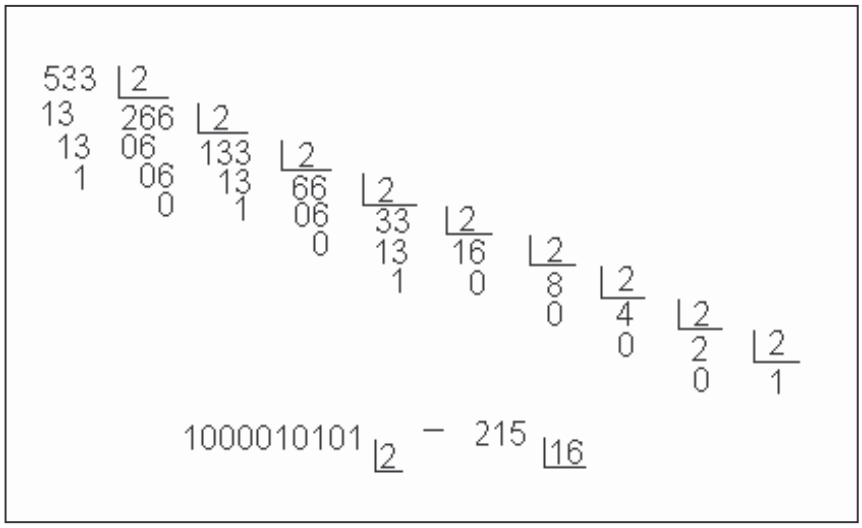


Figura 1. Ejemplo de paso de decimal a binario y hexadecimal

Si el número de partida en base 10 tiene parte decimal, ésta se cambia de base mediante multiplicaciones sucesivas por la base a la que se desea cambiar, obteniéndose el resultado a partir de la partes enteras de dichos productos.

Ejemplo 2: Pasar $533'56_{(10}$ a binario y hexadecimal

$533'566_{(10} = 1000010101'100011..._{(2} = 215'8F..._{(16}$

0'56 A 2 = 1'12	0'56 A 16 = 8'96
0'12 A 2 = 0'24	0'96 A 16 = 15'36
0'24 A 2 = 0'48	...
0'48 A 2 = 0'96	
0'96 A 2 = 1'92	
1'92 A 2 = 1'84	
...	

Los cambios de base entre octal y hexadecimal convienen realizarlos por intermedio del binario.

Los cambios de base entre dos bases extrañas cualesquiera hay que realizarlos a través de base 10.

Álgebra de Boole

Definición y postulados

Se define un Algebra de Boole (A,+,*) como todo conjunto de elementos capaces de adoptar dos valores, designados por 1 y 0, y entre los cuales están definidas dos operaciones: suma lógica (+) y producto lógico (*). Cada uno de dichos elementos recibe de variable lógica o binaria.

Todo Algebra de Boole cumple los siguientes postulados:

1.- Propiedad conmutativa: Dadas dos variables lógicas a,b 0 A (A= Algebra de Boole), se cumple...

a+b = b+a y a*b = b*a

2.- Propiedad distributiva: Dadas tres variables lógicas a,b,c 0 A se cumple...

a*(b+c) = a*b + a*c y a+(b*c) = (a+b)*(a+c)

3.- Elemento neutro: Existe un elemento neutro para cada una de las dos operaciones, designados por 0 para (+), 1 para (*). Así, dada la variable a A, dichos elementos cumplen las siguientes condiciones ....

a+0 = a y a*1 = a

4.- Elemento simétrico (complementario o inverso): Existe, para cada variable lógica a A, su complementaria o inversa (), definida para ambas operaciones (+) y (*), y tal que siempre se cumple...

a+ a= 1 y a* a= 0

Resolución de ejemplos

1º) Sea un Algebra de Boole A definida como conjunto de interruptores que pueden estar en dos posiciones: abierto (0), o cerrado (1). Sean las operaciones de (+) y (*) dos diferentes formas de asociar los interruptores: en paralelo (+) y en serie (*). Los cuatro postulados son aquí los siguientes:

a) Propiedad Conmutativa:

a+b = b+a a*b = b*a

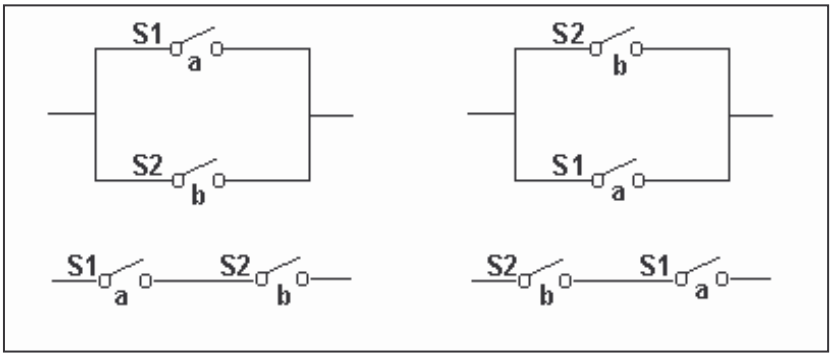
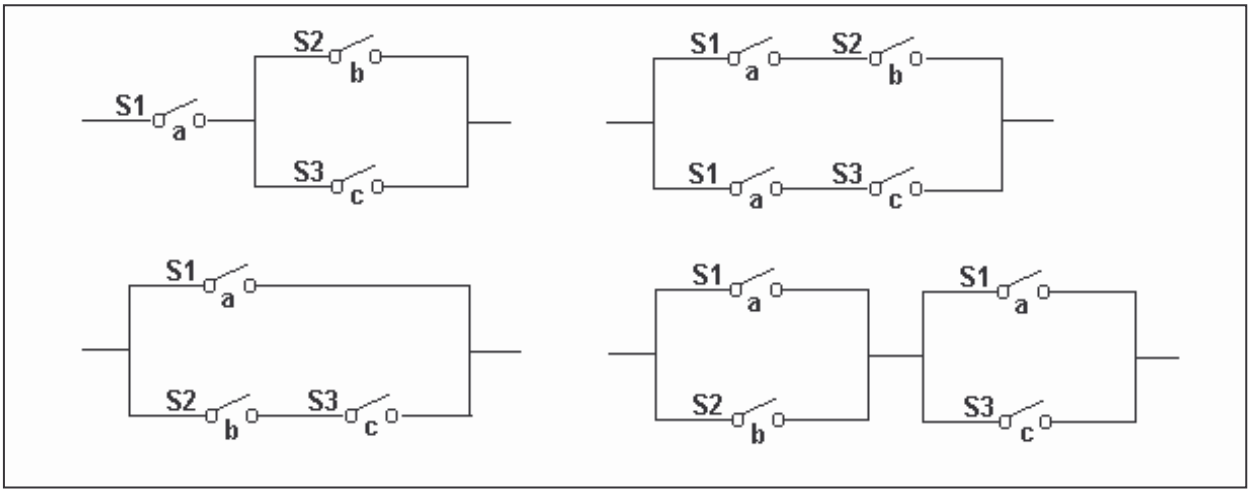


Figura 3.1. Propiedad conmutativa

b) Propiedad Distributiva:

a*(b+c) = a*b + a*c a+(b*c) = (a+b) * (a+c)



c) Elementos Neutros: 0 = Conmutador siempre abierto
1 = Conmutador siempre cerrado

a+0 = a a*1 = a

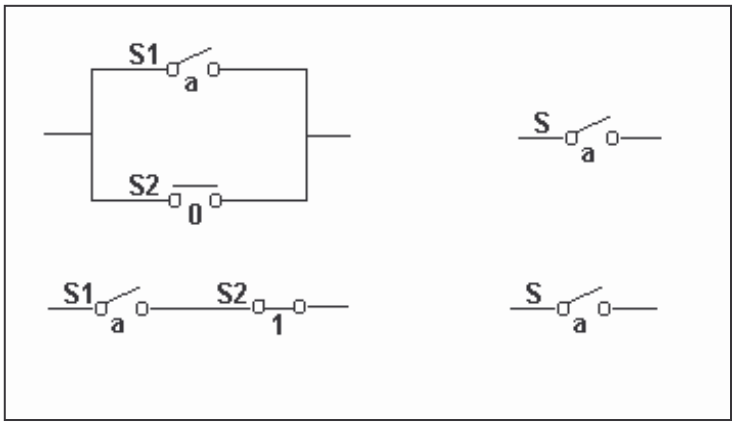


Figura 3.3. Elementos neutros

d) Elementos Simétricos: Se define el elemento simétrico a de un interruptor a como otro interruptor tal que, cuando a está abierto, a está cerrado, y viceversa.

a+ a= 1 y a* a= 0

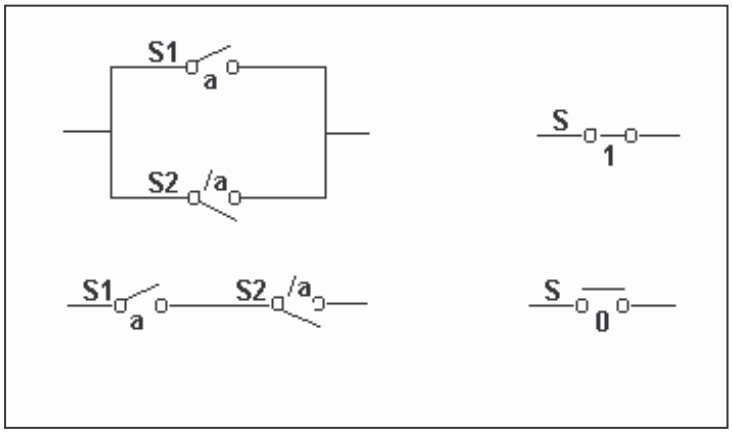


Figura 3.4. Elementos simétricos

2º) Sea A el conjunto de las proposiciones (frases aseverativas) que pueden tomar dos valores: verdadero y falso. Sean las dos operaciones (+), (*) dos tipos de conjunciones que unen dos proposiciones: "o" (+, V, OR) e "y" (*, , AND), definidas de la forma siguiente:

a OR b es verdadero si alguna de las proposiciones a, b es verdadera, y falsa si las dos son falsas.

a AND b es verdadero si las dos proposiciones a, b son verdaderas, y falsa si alguna es falsa.

Este conjunto posee también estructura de Algrebra de Boole.

Teoremas del Algebra de Boole

Son 7 los Teoremas fundamentales de un Algebra de Boole, y se demuestran todos a partir de los cuatro postulados anteriores.

1.- Ley de Dualidad: Cualquier expresión o identidad en un Algebra de Boole tiene su expresión dual que se obtiene intercambiando (+) por (*) y 0 por 1.

Por ejemplo:

a+0 = a a*1 = a

a+ a= 1 a* a= 0

a*(b+c) = (a*b) + (a*c) a+(b*c) = (a+b) * (a+c)

2.- Se cumplen, para toda variable a 0 A, las dos condiciones siguientes:

a+1 = 1 a*0 = 0

Para demostrar esto:

1 = a+a = a + (a * 1) = (a+a) * (a+1) = 1 * (a+1) = a+1

0 = a* a= a * (a+ 0) = (a*a) + (a*0) = 0 + (a*0) = a*0

Con estas dos propiedades y las de los Elementos Neutros, podemos establecer las llamadas "Tablas de Verdad" de las operaciones suma lógica y producto lógico:

Suma lógica:
la suma lógica es igual que la suma decimal excepto en el último caso.

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

Producto lógico:
el producto lógico coincide exactamente con el producto decimal.

a	b	a*b
0	0	0
0	1	0
1	0	0
1	1	1

3.- Ley de Idempotencia: Se cumple, para toda variable a 0 A, lo siguiente...

a+a =a y a*a = a

Para demostrarlo:
a = a+0 = a + (a*) = (a+a) * (a+) = (a+a)*1 = a+a
a = a*1 = a * (a+) = (a*a) + (a*) = (a*a)+0 = a*a

4.- Ley de Absorción: Para todo par de variables a,b 0 A, se cumple ...

a+(a*b) = a y a*(a+b) = a

Demostración:
a = 1*a = (1+b) * a = (1*a) + (b*a) = a +(a*b)
a = 0+a = (0*b) + a = (0+a) * (b+a) = a *(a+b)

5.- Ley asociativa: Dadas tres varibles lógicas a,b,c 0 A, se cumple...

a+(b+c) = (a+b)+c y a*(b*c) = (a*b)*c

Demostración práctica:

c b a	b+c	a+(b+c)	a+b	(a+b)+c
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

6.- Ley de la doble negación o Ley Involutiva: Para toda variable a 0 A, se cumple...

a = a

7.- Leyes de Morgan: Para todo par de variables lógicas a,b 0 A, se cumple...

a+b = a*b y a*b =a+b

Para demostrar esto:

a+b = a*b Si demostramos que siempre se cumple (a+b) + (a*b) = 1 y, (a+b) * (a*b) = 0, es porque (a+b) y (a*b) son siempre opuestos, o sea, porque (a+b) y (a*b) son siempre iguales.

(a+b) * (a*b) = [(a+b)+a] * [(a+b)+b] = (1+b) * (1+a) = 1*1 = 1

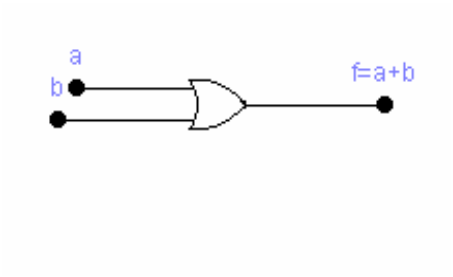
(a+b) * (a*b) = [a*(a*b)] + [b*(a*b)] = (0*b) + (0*a) = 0+0 = 0

Estas dos leyes son muy importantes, ya que permiten pasar de expresiones en sumas lógicas a expresiones equivalentes en productos lógicos.

Puertas lógicas

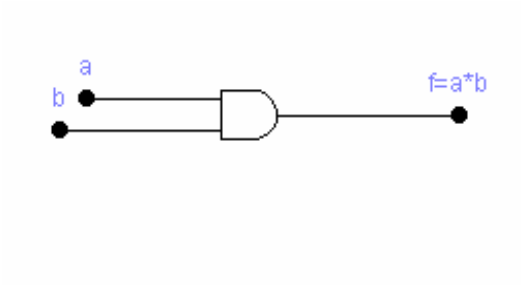
Anteriormente se vio el ejemplo del álgebra de los conmutadores, cuya realización práctica se llevó a cabo mediante relés para construir los primeros ordenadores de la historia. El avance de la tecnología electrónica ha llevado a la realización física de otros elementos, las "puertas lógicas", que configuran también un álgebra de Boole. En este caso las variables binarias son señales eléctricas de tensión alta (H,1) o baja (L,0). Las puertas básicas son las correspondientes a las tres operaciones lógicas básicas: suma, producto y complementación.

1.- Puerta **OR**: Salida 1 si hay algún 1 en las entradas.



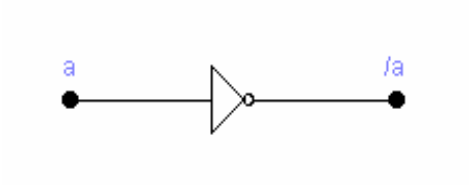
a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

2.- Puerta **AND**: Salida 1 si todas las entradas son 1.



a	b	a*b
0	0	0
0	1	0
1	0	0
1	1	1

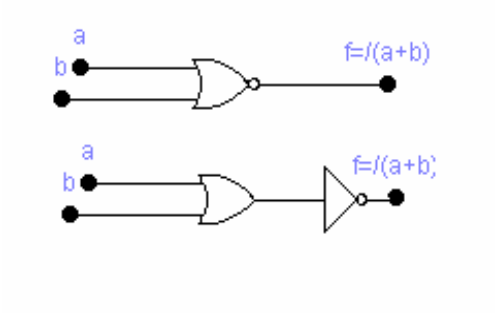
3.- Puerta **NOT**:



a	a
0	1
0	0

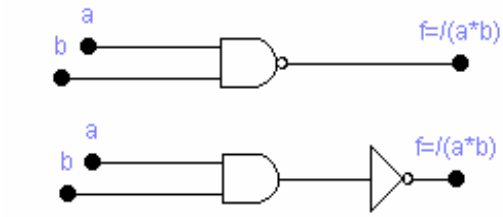
Mediante combinaciones de estas se obtienen otras dos de muy amplio uso:

4.- Puerta **NOR**:



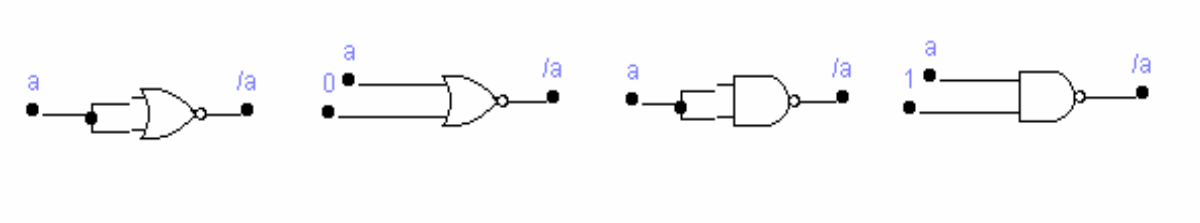
a	b	a+b
0	0	1
0	1	0
1	0	0
1	1	0

5.- Puerta **NAND**:



a	b	a*b
0	0	1
0	1	1
1	0	1
1	1	0

Las puertas más utilizadas son la NOT, NOR y NAND. A su vez, las puertas NOR y NAND pueden funcionar como puertas inversoras, conectando sus entradas apropiadamente:



Resolución de ejemplos

Representar con puertas lógicas la operación compuesta (función): **c+(b*a)**

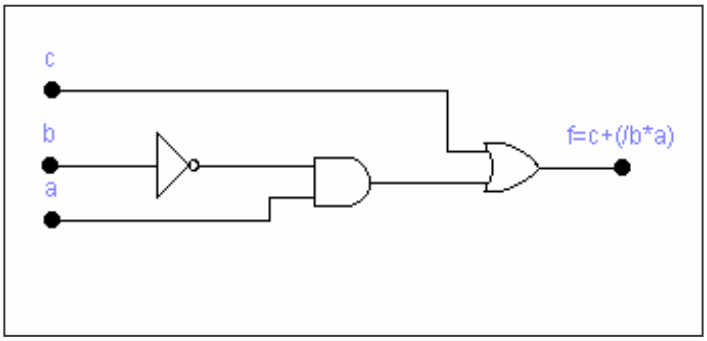


Figura 3.4. Representación de f=c+(/b*a)

Otra forma se obtendría aplicando la ley distributiva con lo que la función quedaría de la siguiente forma: **c+(b*a) = (c+b) * (c+a)**

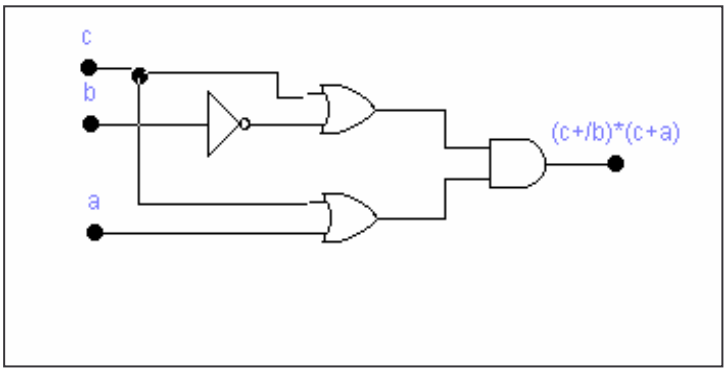


Figura 3.5. Representación de f=(c+/b)*(c+a)

En este caso es preferible la primera forma, ya que necesita menos puertas y por lo tanto el coste del circuito es menor.

Simbología:

En una puerta lógica, una entrada con un circuito significa entrada invertida (a través de un inversor), e igualmente, una salida con circuito significará normalmente salida a través de un inversor.

Funciones en un álgebra de Boole

Una función es una expresión de variables boolianas o binarias unidas por las operaciones lógicas suma, producto y complementación. Se representa por f(...c,b,a), y un ejemplo puede ser el siguiente... f_1 (c,b,a) = a+ cb + cba.

Una función se puede considerar como una forma de expresar el funcionamiento de un circuito electrónico a base de puertas lógicas, en el que las variables a,b,c,... son las señales binarias de entrada, y la función f(...c,b,a) es la señal binaria de salida.

Cualquier término de la función en que aparezcan todas las variables de que depende la función se llama "término canónico", y aquella función formada exclusivamente por términos canónicos recibe el nombre de "función canónica". En el ejemplo anterior, el sumando "cba" es término canónico.

Existe la "función dual" de cualquier función f(..c,b,a), obtenida a partir de ella cambiando productos por sumas y viceversa (ley de dualidad). La función dual del ejemplo anterior será... f_2(c,b,a) = a*(c+b) * (c+b+a), diferente de f_1(c,b,a). El término (c+b+a) es también canónico.

Un término canónico de la forma cba (producto de variables) se llama **MINTERM**, y uno de la forma c+b+a (suma de variables) se llama **MAXTERM**.

Una función de tres variables puede tener hasta 2^3 minterms o maxterms diferentes. Los posibles términos minterm para una función f(c,b,a) son...

MINTERM	MAXTERM	nombre binario	nombre decimal
c b a	c + b + a	0 0 0	0
c b a	c + b + a	0 0 1	1
c b a	c + b + a	0 1 0	2
c b a	c + b + a	0 1 1	3
c b a	c + b + a	1 0 0	4
c b a	c + b + a	1 0 1	5
c b a	c + b + a	1 1 0	6
c b a	c + b + a	1 1 1	7

Para dar nombre a cada uno de ellos se utiliza su configuración binaria correspondiente, o bien su equivalente decimal. Dicha configuración binaria se obtiene asignando a las variables complementadas el valor 0, y a las no complementadas el valor 1.

La variable "a" representa la cifra binaria de menor orden; después le siguen "b" y "c".

Con este convenio se puede nombrar o especificar cualquier función canónica dada por sus tréminos minterm o maxterm:

f_3 (c,b,a) = cba + cba + cba = Σ_3 (5, 2, 3)

101 010 011

5 2 3

f_4 (c,b,a) = (c+b+a) * (c+b+a) * (c+b+a) = Π_3 (5, 2, 3)

1 0 1 0 1 0 0 1 1

5 2 3

Esta forma de representar funciones es muy util para la posterior simplificación óptima de las mismas, pero sólo es aplicable a funciones canónicas (formada por términos minterm o maxterm). Debemos encontrar la forma de expresar cualquier función no canónicas mediante su equivalente canónico.

Existe una "regla práctica" para pasar una función cualquiera a forma de minterms, y es la siguiente:

“Se multiplica cada término no canónico de la función por la suma de la variable o variables que en él falten en forma (directa + complementada)”. Como ejemplo:

f_5 (c,b,a) = cb + ca = cb (a+a) + ca (b+b) = cba + cba + cba = Σ_3 (7, 6, 3, 1)

f_6 (c,b,a) = cba + cb + c = cba + cb(a+a) + c(b+b) (a+a) =

= cba + ~~cba~~ + cba + cba + cba + cba + cba =

= Σ_3 (7, 6, 3, 2, 1, 0)

Si obtenemos dos términos iguales eliminamos uno, ya que, según la ley de Idempotencia, es x+x = x.

Otra forma de conseguir esto es a partir del "Teorema fundamental" de funciones de un Algebra de Boole, que dice que cualquier función se puede escribir de la forma

f(c,b,a) = c * f(1,b,a) + c * f(0,b,a)

o bien,
f(c,b,a) = b * f(c,1,a) + b * f(c,0,a)
f(c,b,a) = a * f(c,b,1) + a * f(c,b,0)

Este teorema también se cumple en la forma dual:

f(c,b,a) = [c+f(0,b,a)] * [c + f(1,b,a)]

La demostración de este teorema, a partir de su primera expresión es,

f(c,b,a) = c * f(1,b,a) + c * f(0,b,a)

Para c = 0 f(0,b,a) = ~~0 * f(1,b,a)~~ + 1 * f(0,b,a) = f(0,b,a)
Para c = 1 f(1,b,a) = 1 * f(1,b,a) + ~~0 * f(0,b,a)~~ = f(1,b,a)

Como se cumple para c=0 y c=1 se cumple siempre.
Operando:
f(c,b,a) = cba f(111) + cba f(110) + cba f(101) + cba f(100) +
+ cba f(011) + cba f(010) + cba f(001) + cba f(000)

Esto significa que cualquier función f(c,b,a) se puede representar en forma de suma de minterms multiplicados por los coeficientes binarios resultantes de aplicar la función a la configuración binaria correspondiente a cada minterm.

Para expresar cualquier función en forma de maxterms se utiliza la expresión dual:

f(c,b,a) =[c+b+a+f(000)] [c+b+a+f(001)] [c+b+a+f(010)] [c+b+a+f(011)] [c+b+a+f(100)] [c+b+a+f(101)] [c+b+a+f(110)] [c+b+a+f(111)]

Resolución de ejemplos

Ejemplo 1: f_5 (c,b,a) = cb + ca

f_5 (000) = 00 + 10 = 0 f_5 (011) = 1 f_5 (110) = 1
f_5 (001) = 00 + 11 = 1 f_5 (100) = 0 f_5 (111) = 1
f_5 (010) = 01 + 10 = 0 f_5 (101) = 0

f_5 (c,b,a) = cba1 + cba1 ~~+ cba0~~ ~~+ cba0~~ + cba1 ~~+ cba0~~ + cba1 ~~+ cba0~~ = cba + cba + cba + cba = Σ_3 (7, 6, 3, 1)

f_5 (c,b,a) = [c+b+a+0] ~~[c+b+a+1]~~ [c+b+a+0] ~~[c+b+a+1]~~ [c+b+a+0][c+b+a+0] ~~[c+b+a+1]~~ ~~[c+b+a+1]~~ = Θ_3 (7, 6, 3, 1)

Existe una "regla práctica" para pasar una función canónica dada en forma de minterms a su expresión en forma de maxterms, y es la siguiente:

Se observa sobre la función minterm cuales son los términos que faltan, y se obtiene el "complemento a 7" de dichos términos (a 7 si son tres las variables). Estos complementos son los términos maxterm de la función.

"Complemento a 7" significa cantidad que falta hasta 7 en un determinado número. En general, siendo "n" el número de variables, se ha de hallar el complemento a 2^n-1 de los términos minterm que faltan.

La transformación inversa (de una función cánonica maxterm a su equivalente minterm) es idéntica a ésta.

Ejemplo 2: f_5 (c,b,a) = cb + ca = Σ_3 (7, 6, 3, 1)

Términos que faltan : 0, 2, 4 y 5
Complemento a 7 : 7, 5, 3 y 2
f_5 (c,b,a) = Θ_3 (7, 5, 3, 2) =(c+b+a) (c+b+a) (c+b+a) (c+b+a)

Proceso inverso: f_5 (c,b,a) = cb + ca = ϑ_3 (7, 5, 3, 2)

Términos que faltan: 0, 1, 4 y 6
Complemento a 7: 7, 6, 3 y 1
f_5 (c,b,a) = 3_3 (7, 6, 3, 1) = cba + cba + cba + cba

Ejemplo 3: f_6 (d,c,b,a) = dca + cb + dcba

f_6 (d,c,b,a) = dc(b+b)a +(d+d)cb (a+a) + dcba =
= dcba + dcba + dcba + dcba + dcba + dcba + dcba =
= 3_4 (7, 5, 9, 8, 1, 0, 10)

Términos que faltan: 2, 3, 4, 6, 11, 12, 13, 14, 15
Complemento a 15: 13, 12, 11, 9, 4, 3, 2, 1, 0
f_6 (d,c,b,a) = ϑ_4 (13, 12, 11, 9, 4, 3, 2, 1, 0)

Ejemplo 4: f_7 (c,b,a) = b+a + cba + (c+b)a

f_7 (c,b,a) = ba + cba + (cb)+a = (c+c)ba + cba + cb + a =
= (c+c)ba + cba + cb(a+a) + (c+c)(b+b)a =
= cba + cba + cba + cba + ~~cba~~ + ~~cba~~ + ~~cba~~ + cba + ~~cba~~ = cba + cba + cba + cba + cba + cba =
= 3_3 (0, 1, 2, 4, 6)

Términos que faltan: 3, 5, 7
Complemento a 7: 4, 2, 0
f_7 (c,b,a) = ϑ_3 (0, 2, 4) = (c+b+a) (c+b+a) (c+b+a)

Tablas de verdad

Es otra forma de representar una función lógica, y sirve para obtener el desarrollo en forma canónica de la misma. Consiste en escribir todas las posibles combinaciones de las "n" variables (un total de 2^n) y anotar los valores que toma la función para cada una.

Por ejemplo la función f_5 (c,b,a) = cb + ca

	c b a	F_5
0	0 0 0	0
1	0 0 1	1
2	0 1 0	0
3	0 1 1	1
4	1 0 0	0
5	1 0 1	0
6	1 1 0	1
7	1 1 1	1

Para expresar la función en forma de minterms tomamos las combinaciones para las cuales la función vale 1, obteniendo de ellas los términos canónicos minterm mediante el convenio normal (valor 1 = variable directa, valor 0 = variable invertida).

Para expresar la función en forma de términos maxterms, tomamos las combinaciones para las cuales la función vale 0, obteniendo de ellas los términos maxterm mediante el convenio unvertido (valor 0 = variable directa, valor 1 = variable inversa).

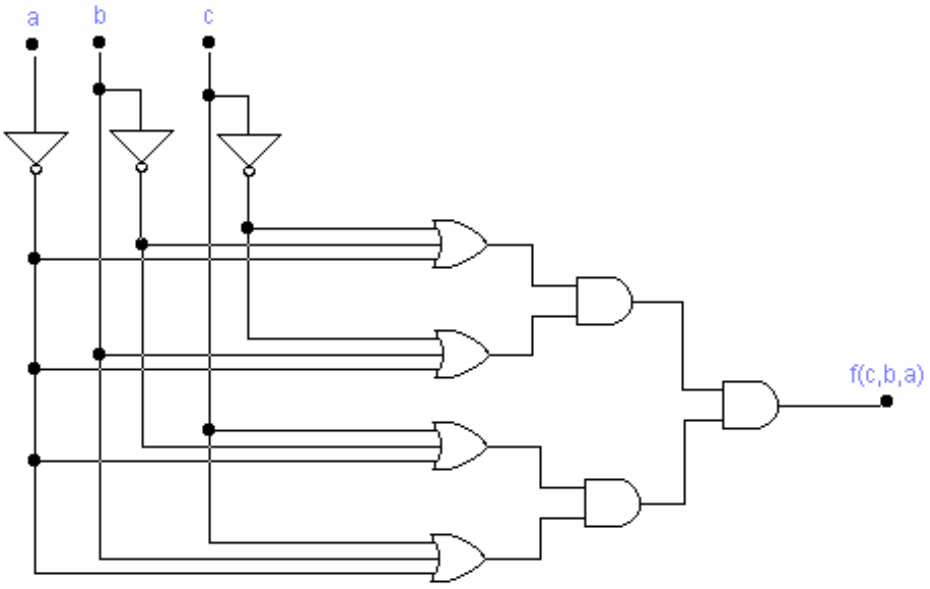
Resolución de ejemplos

Diseñar en forma de minterms y maxterms un circuito en puertas lógicas con tres variables de entrada, tal que, si la combinación binaria de entrada representa un número decimal "par", el circuito lo detecte (da salida 1).

f(c,b,a) = cba + cba + cba + cba = 3_3 (0, 2, 4, 6)

f(c,b,a) = (c+b+a) (c+b+a) (c+b+a) (c+b+a) = ϑ_3 (6, 4, 2, 0)

	c b a	F_5
0	0 0 0	1
1	0 0 1	0
2	0 1 0	1
3	0 1 1	0
4	1 0 0	1
5	1 0 1	0
6	1 1 0	1
7	1 1 1	0



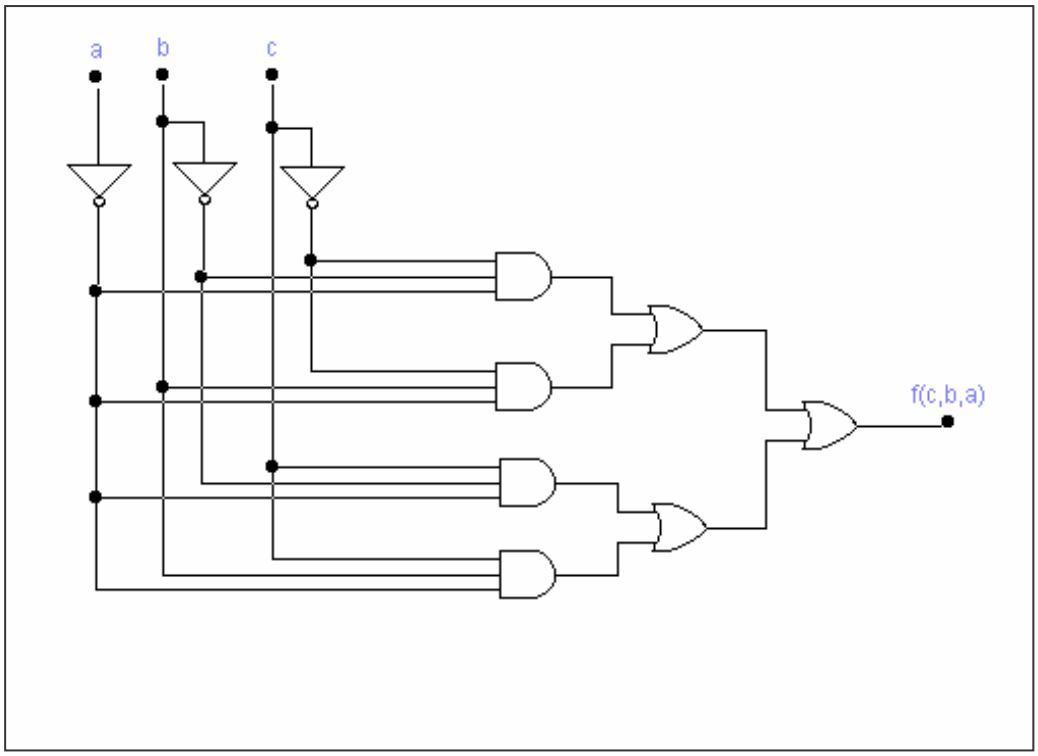


Figura 3.6. Implementación

Estos dos circuitos no son los óptimos para resolver el problema, ya que, efectuando la simplificación de la función f, obtendríamos como resultado...

f(c,b,a) = a

El cicuito óptimo es:

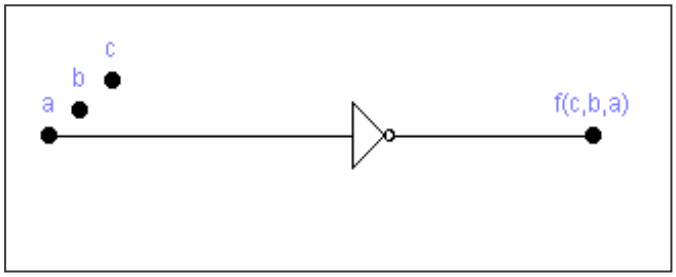
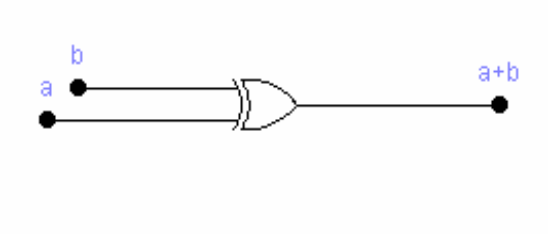


Figura 3.7. Circuito óptimo

Funciones lógicas importantes

1.- Función **OR-EXCLUSIVA (.)**

Es la función que verifica la siguiente tabla de verdad:

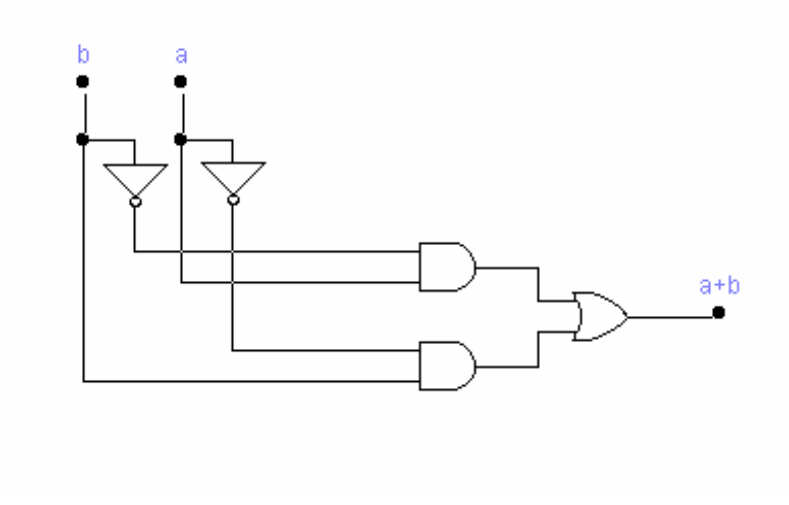


a	b	a⊕b
0	0	0
0	1	1
1	0	1
1	1	0

Esta función está construida en la práctica en forma de una puerta lógica especial (puerta OR-EXCLUSIVA), cuyo símbolo se muestra en la figura.

La función OR-Exclusiva vale 1 cuando hay un número impar de variables de entrada a 1, y vale 0 cuando dicho número es par. Este criterio es aplicable tanto a puertas OR-Exclusiva de dos entradas como a puertas con mayor número de entradas.

Su equivalente con puertas AND y OR es el siguiente:

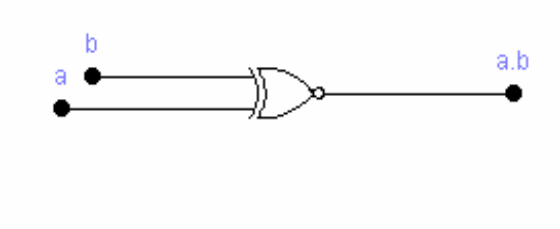


La función OR-EX de 2 entradas cumple también las siguientes propiedades:

a⊕b = a⊕b = a⊕b = a⊕b

2.- Función **NOR-EXCLUSIVA**

Es la inversa de la OR-Exclusiva, y su tabla de verdad y símbolo son los siguientes:



a	b	aΘb
0	0	1
0	1	0
1	0	0
1	1	1

Esta función vale 1 cuando hay un número par de "unos" en las entradas (considerando el cero par). No suele encontrarse como puerta lógica individual, sino que se obtiene con una OR-Exclusiva más una puerta inversora.

La función NOR-EX de 2 entradas cumple a⊕b = a⊕b = a⊕b

Realización de funciones en puertas NAND Y NOR

Cualquier función booleana puede realizarse usando exclusivamente puertas NAND O NOR. Para transformar la expresión de una función lógica cualquiera a otra equivalente en puertas NAND (a□b) o NOR (a+b), se aplican las leyes de Morgan y la doble complementación:

a+b+c+... = a.b.c...

a.b.c ... = a+b+c...

Resolución de ejemplos

Por ejemplo:

Realizar en puertas NAND las funciones

f_1 (d,c,b,a) = cb + ba + dca y
f_2(d,c,b,a) = (c+b)*(b+a)*(d+c+a)
f_1 = cb + ba + dca = cb . ba . dca
f_2 = (c+b) (b+a) (d+c+a) = cb . ba . dca

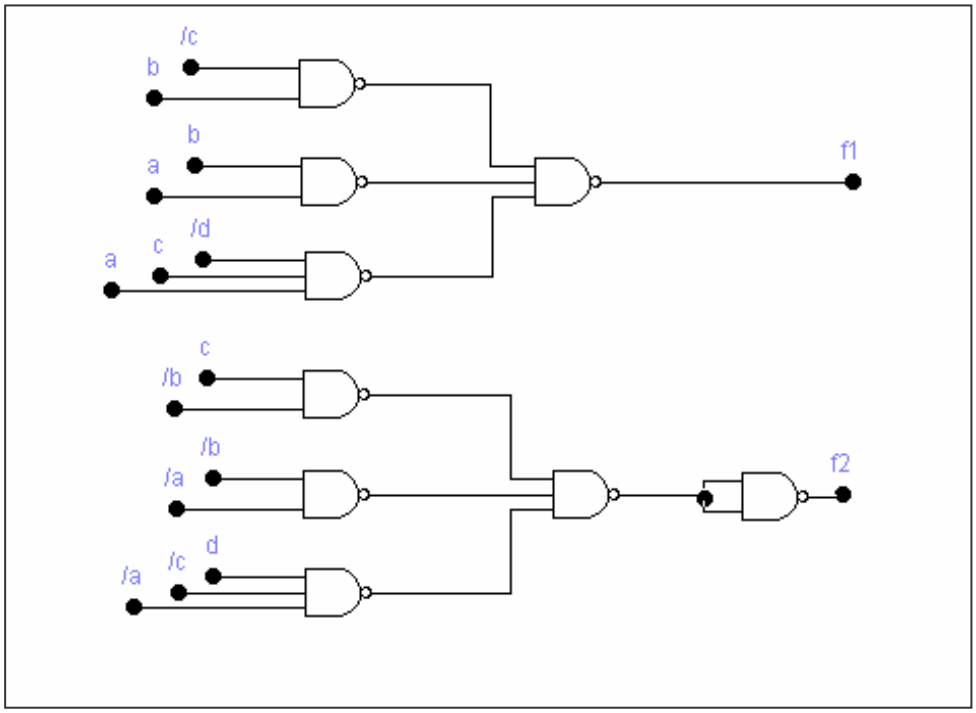


Figura 3.8. Circuito de puertas NAND

Otro ejemplo: Realizar en puertas NOR las funciones

f_1 (d,c,b,a) = cb + ba + dca y
f_2(d,c,b,a) = (c+b)*(b+a)*(d+c+a)

f_1 = cb + ba + dca = (c+b) + (b+a) + (d+c+a)
f_2 = (c+b) (b+a) (d+c+a) = (c+b) + (b+a) + (d+c+a)

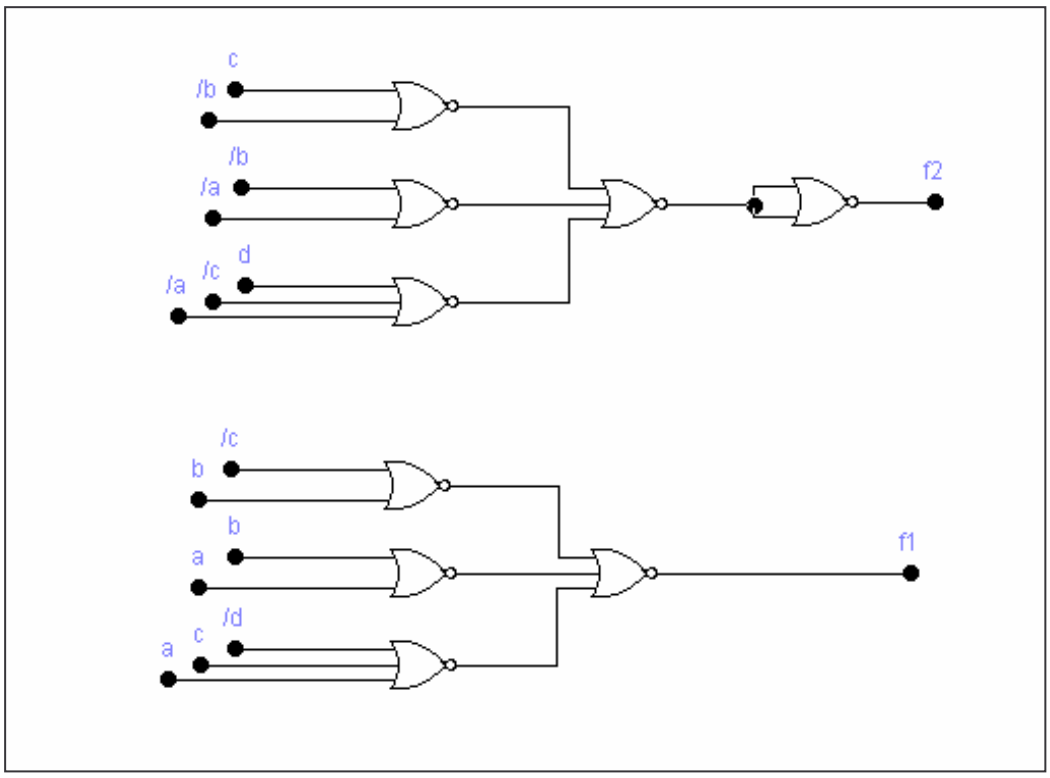


Figura 3.9. Circuito de puertas NOR

Tercer ejemplo: Realizar en puertas NAND y NOR la función...

f3 = (c,b,a) = c(b+a) + cba + c + b

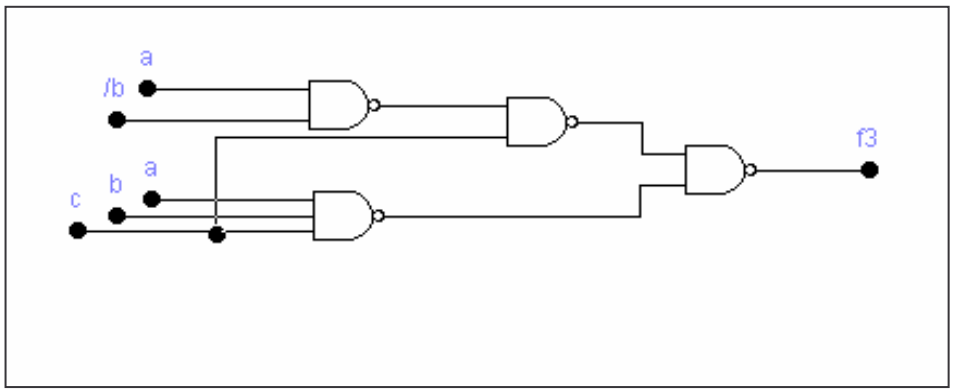


Figura 3.10. Circuito con puertas NAND

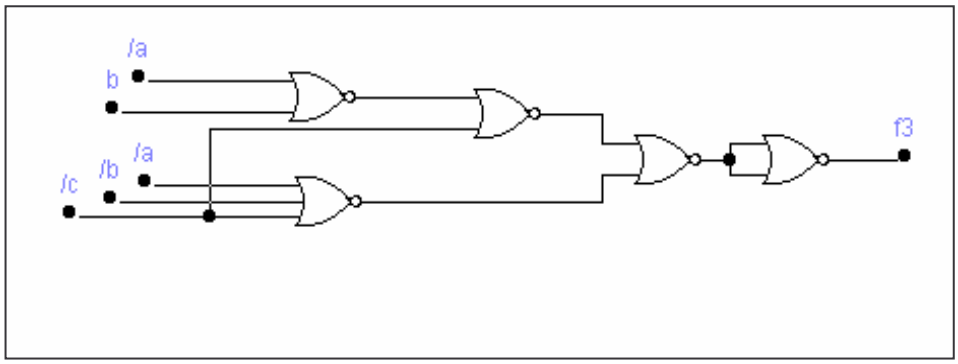


Figura 3.11. Circuito con puertas NOR

Se entiende por "nivel" de un circuito lógico al máximo número de puertas que debe atravesar cualquiera de las variables de entrada hasta alcanzar la salida. A mayor nivel, más lento es el circuito en su respuesta, ya que cada puerta introduce un retardo en la propagación de las señales.

En el ejemplo f_3 hemos obtenido dos circuitos, uno de nivel 4 y otro de nivel 5.

Cualquier función lógica puede realizarse con un circuito de nivel 2 (el correspondiente a la función canónica equivalente), pero se suele aumentar el nivel del circuito para conseguir reducir el número de puertas o el número de entradas en cada puerta, aunque se pierda velocidad de respuesta.

Tecnologías de fabricación de puertas lógicas

Las puertas lógicas son actualmente los elementos básicos fundamentales que constituyen todo tipo de circuitos y máquinas digitales. Hemos visto hasta ahora el comportamiento externo de las puertas más importantes. Veremos ahora cual ha sido la evolución seguida en el modo de realizarlas o fabricarlas en la practica.

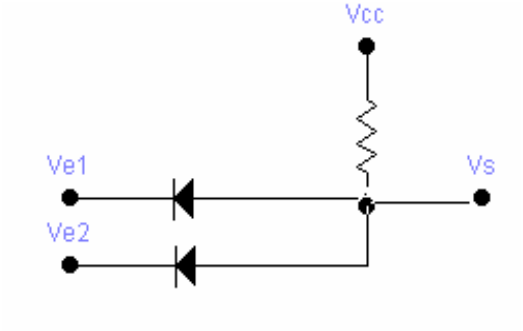
Las características de funcionamiento de las puertas lógicas han ido mejorando rápidamente, a medida que se producían nuevos descrubimientos o avances en el campo de la Electrónica y aparecían nuevas "familias" de puertas lógicas. Entre dichas características, las mas importantes son las siguientes:

- Tiempo de retardo: Intervalo de tiempo desde que se aplica una conmutación en las entradas de una puerta hasta que se produce la conmutación completa en la salida.
- Fan-out: Número máximo de entradas de puerta a las que puede atacar la salida de una puerta determinada.
- Fan-in: Número máximo de entradas que puede tener una puerta de una determinada familia.
- Disipación: Potencia eléctrica media consumida por una puerta en régimen estacionario.

Familias lógicas

Antes del descubrimiento de las válvulas de vacío y del nacimiento de la Electrónica, las funciones lógicas booleanas eran realizadas mediante "relés" (interruptores mecánicos operados eléctricamente mediante un electroimán).

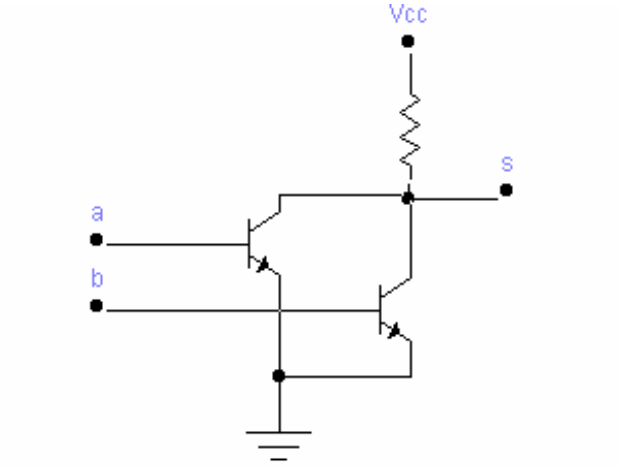
Con la aparición de los diodos se ideó una nueva forma de realizar las puertas lógicas, es decir, surgió una primera tecnología o "familia lógica" cuyo circuito básico era el indicado en la figura:



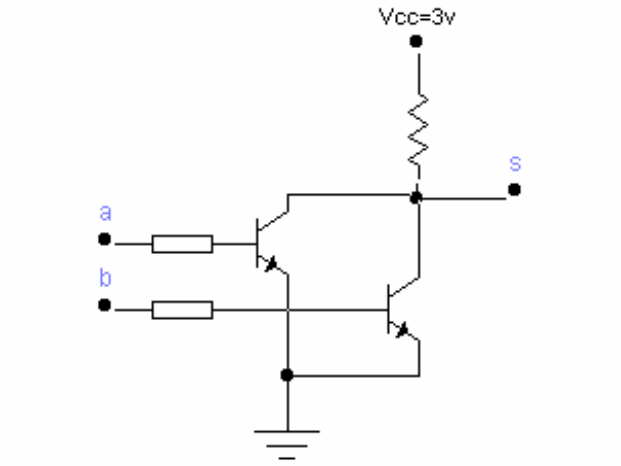
Ve1	Ve2	Vs
0	0	0
0	1	0
1	0	0
1	1	1

Esta familia presentaba problemas de todo tipo, debido a que los diodos no eran ideales.

Con la invención del Transistor aparecieron las primeras familias lógicas comerciales: la DCTL (Lógica de Transistores acoplados directamente) y la RTL (Lógica Resistor-Transistor).



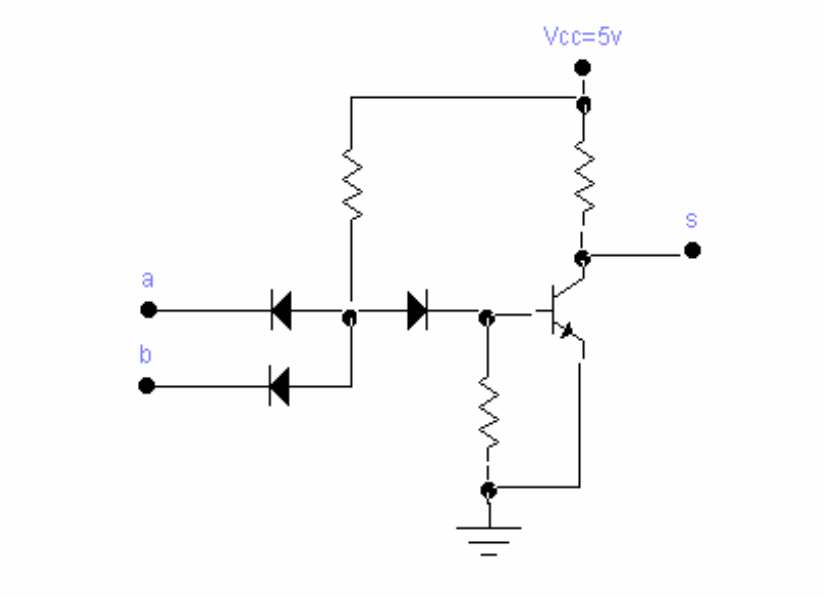
A	B	S
0	0	0
0	1	0
1	0	0
1	1	1



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

Estas familias tuvieron poca difusión, ya que tenían en general malas características en cuanto a velocidad de conmutación, influencia de ruido externo, disipación de potencia, "fan-out", etc.

La primera familia lógica de uso generalizado fue la DTL (Lógica Diodo-Transistor), que luego sería sustituida por la TTL, compatible con ella. La puerta básica es la siguiente:

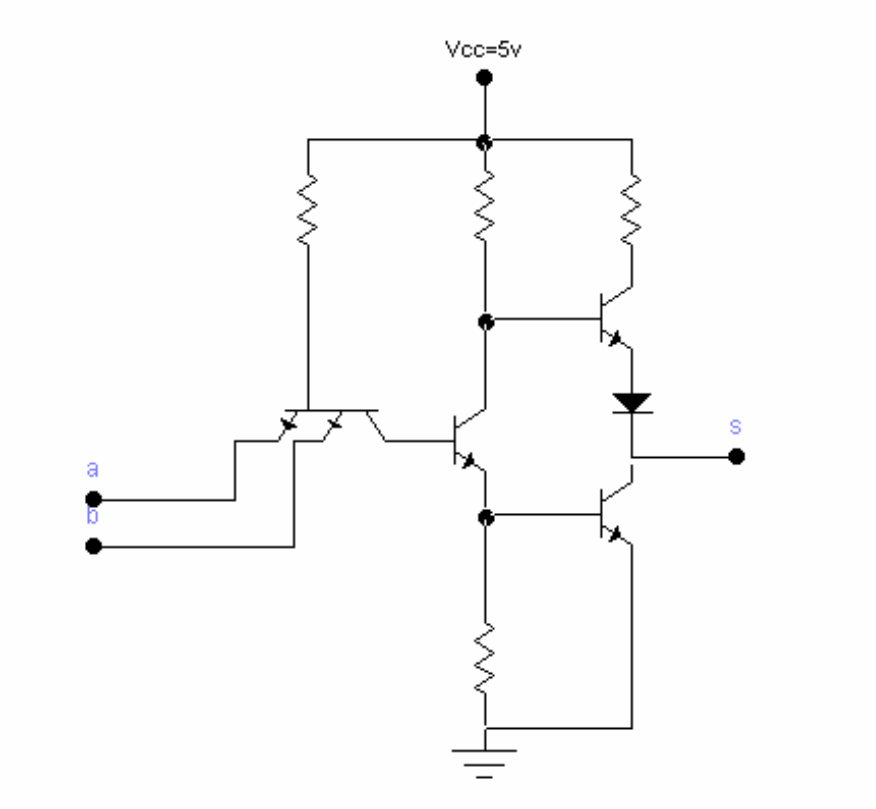


A	B	S
0	0	1
0	1	1
1	0	1
1	1	0

Esta familia tenía como ventajas una gran variedad de puertas disponibles, baja generación de ruido y buen "fan-out" (8), pero su velocidad de conmutación era relativamente baja (tiempo de retardo de 25 nsg.), y presentaba poca inmunidad al ruido externo.

Apareció después la familia TTL (Lógica Transistor-Transistor) la más utilizada actualmente en niveles de integración medios. Presentaba como ventajas la compatibilidad de interconexión con DTL, una gran variedad de puertas y circuitos lógicos, buena inmunidad al ruido, buen "fan-out" (8), menor coste de fabricación que DTL, menor tiempo de retardo (10 nsg.) y menor potencia de disipación (10 mW). Esta familia constituye actualmente la serie de integrados xx74... o xx54.., y trabaja normalmente con niveles lógicos de 0 y 5 voltios, al igual que DTL.

La puerta básica en TTL es la NAND, cuyo circuito interno es el siguiente:



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

De este circuito se desprende que una entrada dejada al aire en TTL equivale a un "1" lógico, ya que el emisor correspondiente de entrada no conduce.

Existen varias subfamilias derivadas de la TTL:

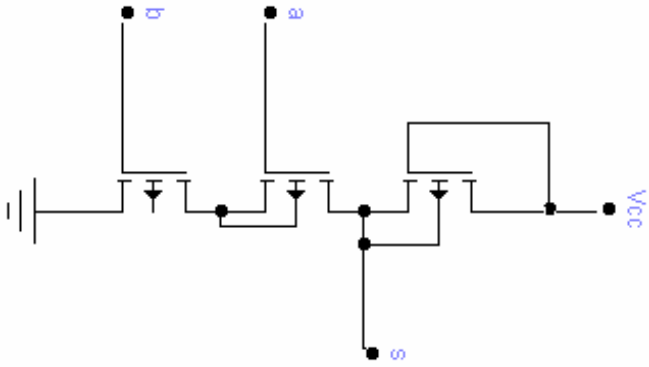
- STTL (TTL Schottky, serie XX74S...), construida a base de transistores Schottky de conmutación rápida, cuya característica fundamental es su gran velocidad de conmutación (retardo de 3 nsg.).

- HTTL (High-speed TTL, serie XX74H...), TTL de alta velocidad.
- LPTTL (Low-power TTL, serie XX74L...), TTL de baja disipación.

- LSTTL (Low-power Schottky TTL, serie XX74LS...), TTL Schottky de baja disipación, la más moderna y la de mejores características.

Otra familia utilizada actualmente, aunque con menor difusión (sólo en aplicaciones científicas y militares), es la ECL (Lógica de emisores acoplados). Es la familia más rápida de todas (retardo de 1 nsg.), pero presenta el inconveniente de que no tiene una gran variedad de ciruitos y además no es compatible con otras familias, por lo que necesita circuitos adaptadores especiales para interconectar con ellas.

Finalmente, existen algunas familias muy extendidas en la actualidad (tanto o más que TTL), de fácil compatibilidad con TTL y construidas a base de transistores unipolares MOSTFET; son las familias P-MOS (con transistores de canal P), N-MOS (con transistores de canal N) y C-MOS (MOST Complementarios, cana N y P simultáneamente), esta última la de mayor difusión en niveles de integración media.

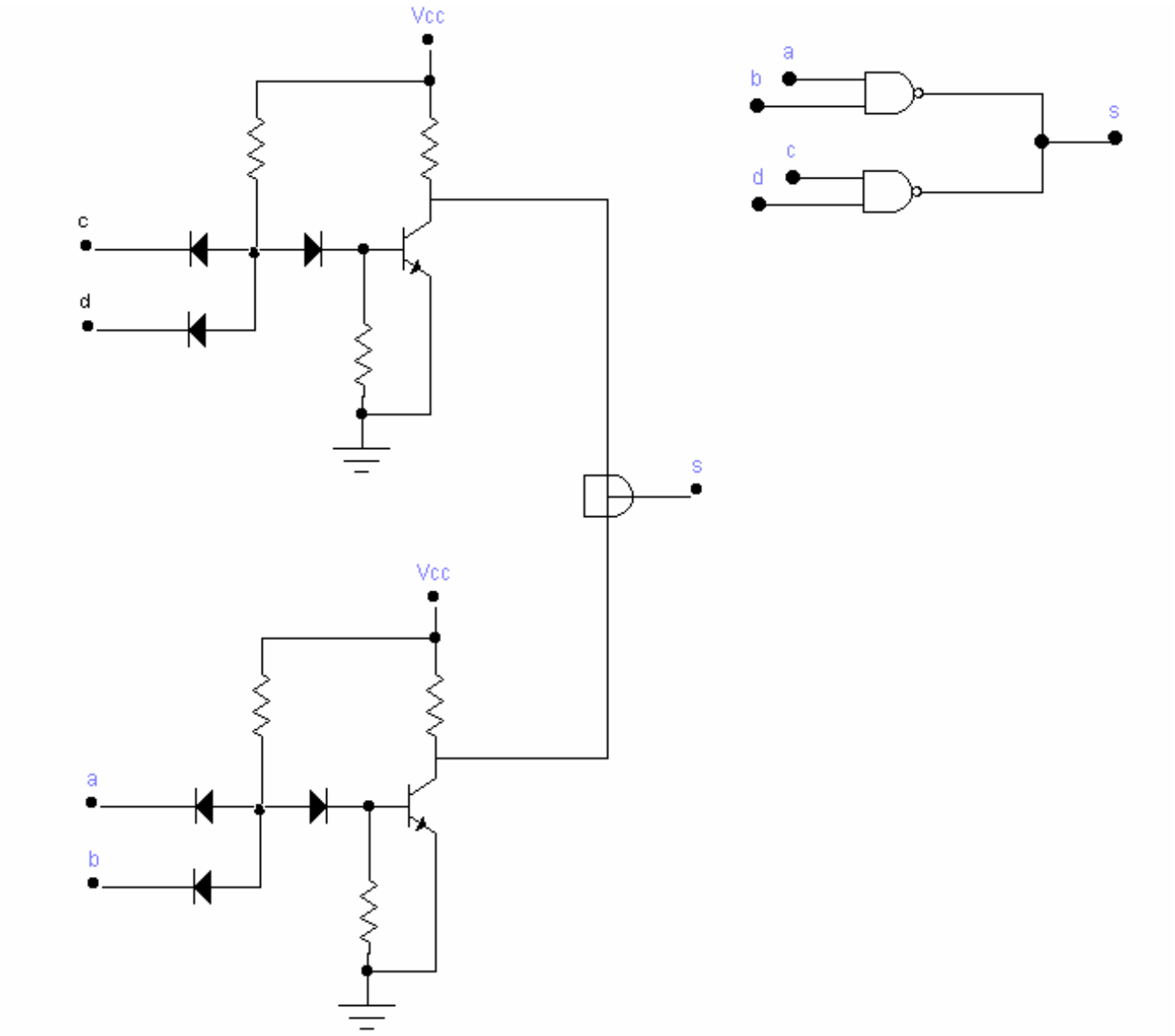


A	B	S
0	0	1
0	1	1
1	0	1
1	1	0

Presentan como ventajas una disipación de potencia ínfima, un "fan-out" muy elevado (debido a la elevada Ze de los MOSTFET), una tensión de alimentación variable (de 3v a 18v) y una gran densidad de integración a bajo precio. Sin embargo, su velocidad de conmutación es baja, y no son directamente conectables con TTL por lo general. Estas familias constituyen las series XX40... y XX41...

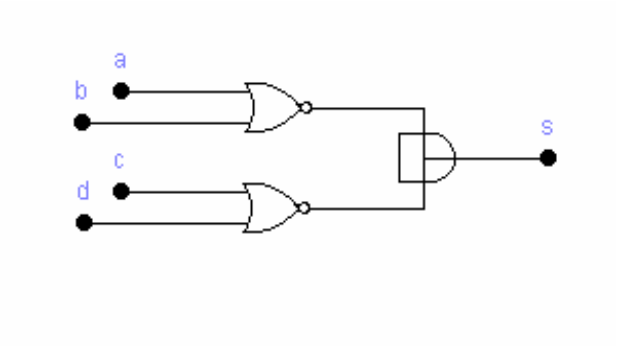
Conexión entre salidas de puertas lógicas

Las familias lógicas DCTL, RTL y DTL permiten el llamado "cableado lógico de salida", es decir, permiten la conexión directa entre las salidas de varias puertas lógicas, según el esquema de la figura:



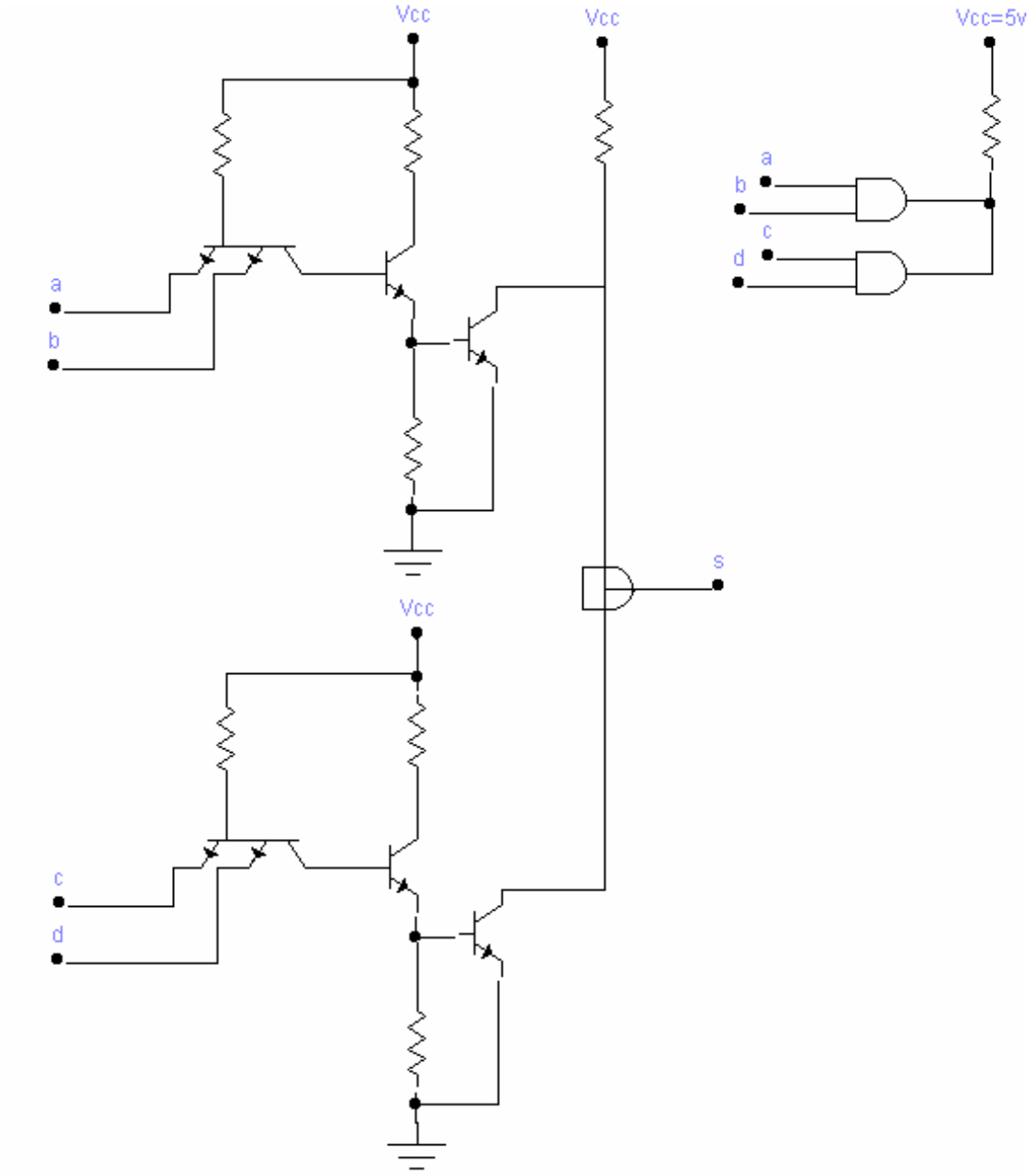
La función resultante es la AND entre las salidas de dichas puertas, ya que si cualquiera de ellas vale 0 (TRT saturado), la salida conjunta también pasará a nivel 0. Esta función recibe el nombre de función "AND cableada".

Si esta función se realiza con puertas NOR, lo que se obtiene es una expansión de dichas puertas NOR para conseguir otra NOR con más entradas:



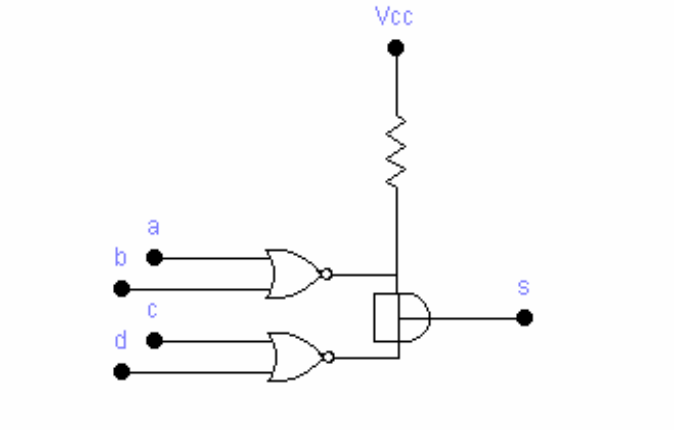
Sin embargo, esta función no puede realizarse con las salidas de puertas tipo TTL normal, ya que se produce un cortocircuito a través de los transistores de salida de dichas puertas.

Para solucionar esto se han ideado las puertas TTL "con colector abierto", cuyo esquema interno es el indicado en la siguiente figura.



Son puertas TTL que necesitan la conexión de una resistencia exterior entre la salida o salidas unidas y la alimentación positiva. El valor de esta resistencia es un compromiso entre velocidad de conmutación, "fan-out" y disipación, y suele ser pequeño (Rext = 1K).

Estas puertas permiten conseguir la función "AND cableada" en TTL, y, al igual que antes, si se realiza con puertas NOR, obtendremos una sencilla expansión de estas:



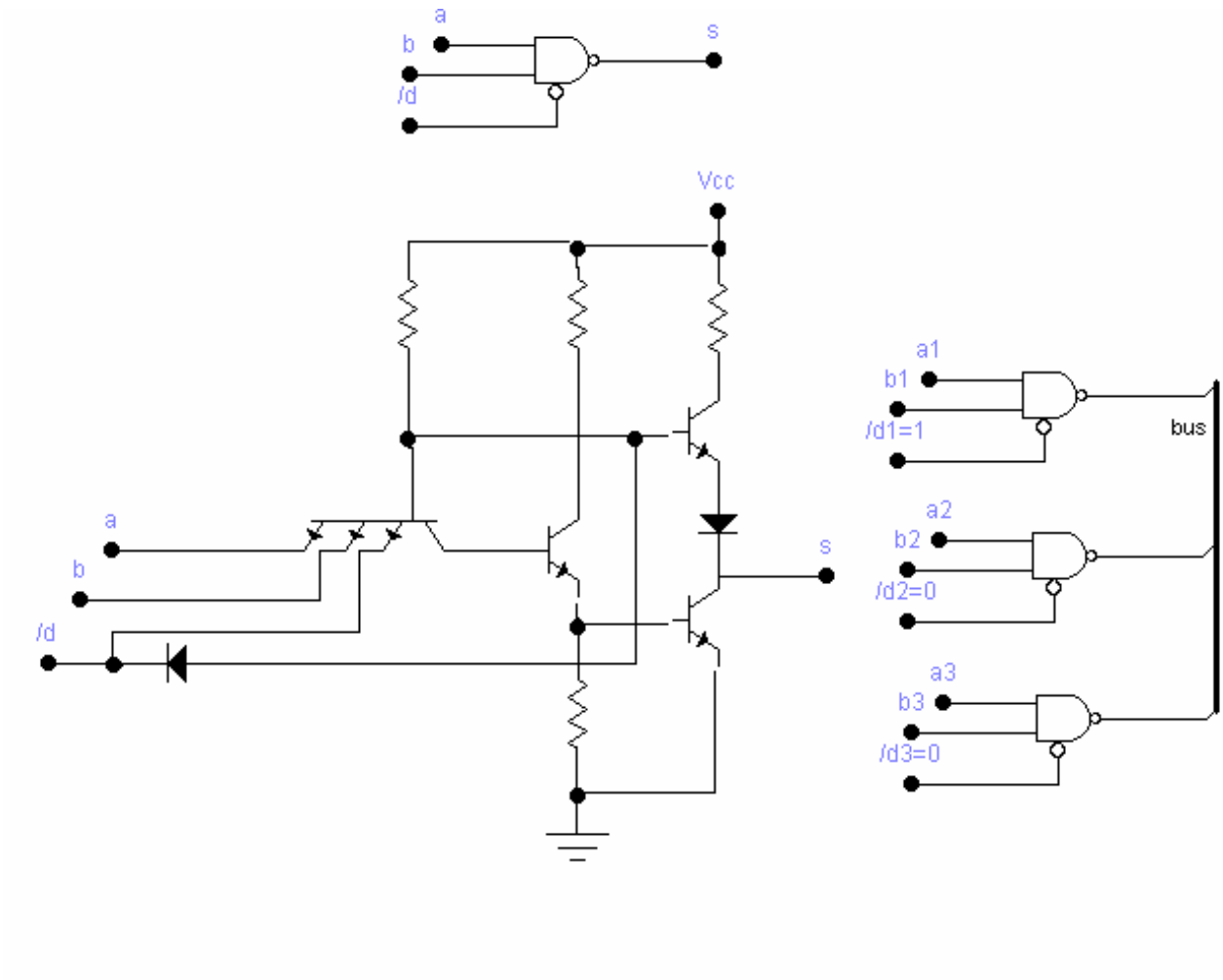
Otra solución al problema de la conexión en paralelo de las salidas de puertas TTL, es el uso de las puertas TTL "de tres estados" (Tristate", consistentes en una puerta standard a la que se añade una entrada especial de "disable" o "inhibición" (entrada D), activa a nivel bajo.

Mientras esta entrada no se active, la puerta funciona normalmente, con los dos niveles de salida 0 y 1 (0v y 5v). Cuando la entrada D es activada, la salida de la puerta pasa a un tercer estado de "alta impedancia", equivalente a un circuito abierto, con lo cual ya no se producirá cortocircuito si esta salida está conectada a otras salidas de puerta.

Cuando se conectan puertas "tristate" con salidas unidas, sólo una de ellas tendrá su entrada de Disable a 1 (aquella cuya información queremos que pase a la salida), estando las demás con D = 0, es decir, con salida aislada.

El circuito interno y esquema de una puerta NAND Tristate es el siguiente:

d	a	b	s
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0
0	X	X	c*a


a
b
/d
s
Vcc
a
b
/d
s
a1
b1
/d1=1
bus
a2
b2
/d2=0
a3
b3
/d3=0

Circuitos combinacionales

Introducción

Existen dos típos generales de circuitos lógicos: combinacionales y secuenciales. Los **circuitos combinacionales**, son aquellos cuyas salidas en un determinado instante, son función exclusivamente del valor de las entradas en ese instante. Sin embargo, en los **circuitos secuenciales**, las salidas obtenidas en cada momento dependen del valor de las entradas y también del valor de esas mismas salidas en el momento anterior (las salidas dependen del tiempo o momento en que sean tomadas).

Veremos en este tema cómo se diseña un circuito combinacional sencillo.
Los circuitos combinacionales se pueden dividir en dos tipos:

a) Sistemas unifuncionales: Tienen una sola función de salida.
b) Sistemas multifuncionales: Tienen varias funciones de salida.

A su vez, una función puede ser "completa" (su valor está determinado para todas las posibles combinaciones de las variables de entrada) o "incompleta" (existen algunas combinaciones de entrada para las cuales el valor de la función es indeterminado).

Para obtener un circuito combinacional óptimo, se sigue el proceso general siguiente:

1.- Dado el enunciado del problema, establecemos su "tabla de verdad".

2.- A partir de esta tabla, obtenemos la función canónica en minterms o en maxterms.

3.- A continuación simplificamos dicha función, bien en forma algebráica (aplicando teoremas y postulados del Algrebra de Boole) o bien mediante la aplicación de métodos tabulares sencillos (métodos de Karnauh o de McCluskey).

4.- Finalmente, realizamos la función simplificada mediante las oportunas puertas lógicas.

Simplificación de las funciones lógicas simples

Toda función canónica dada por sus términos minterm o maxterm debe ser simplificada, con el fin de utilizar un menor número de puertas lógicas en su realización.

Ahora bien, una función puede ser simplificada y reducida de muchas formas. La mejor será aquella que de lugar a un circuito con el menor número posible de puertas lógicas y con el menor número posible de entradas en cada puerta.

El método básico de simplificación de funciones es el "**método algebraico**", consistente en aplicar directamente la propiedad distributiva a los términos de la función, eliminando variables.

Por ejemplo:

f_1 (d,c,b,a) = dcba + dcba = dcb (a+a) = dcb * 1 = dcb

f_2 (d,c,b,a) = (d+c+b+a) (d+c+b+a) = d+c+b+aa = d+c+b+0 = d+c+b

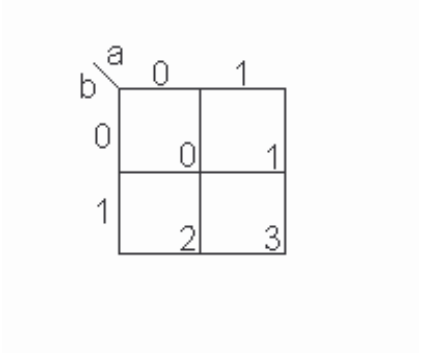
Sin embargo, pocas veces viene expresada la función de forma que sea fácilmente aplicable a este método.

Para conseguir de una forma fácil la mejor simplificación de una función, suelen utilizarse dos posibles métodos tabulares: el método de Karnaugh o el método de McCluskey.

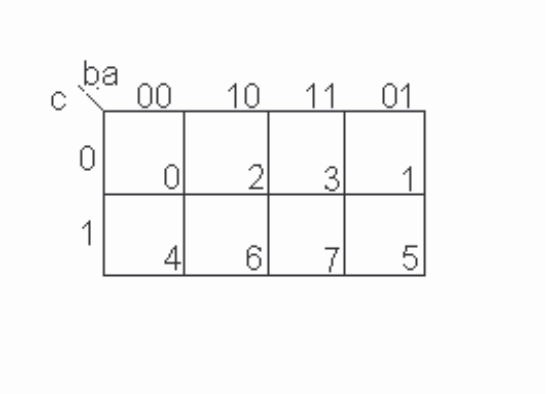
Método de Karnaugh Es un método tabular gráfico que se basa en los llamados "*mapas de Karnaugh*", consistentes en una tabla de cuadros, cada uno de los cuales representa un término canónico. Estos cuadros están distribuidos de tal modo que cualquiera dos de ellos contiguos físicamente, corresponden a términos canónicos adyacentes.

Dos términos canónicos son "*adyacentes*" cuando sus respectivas configuraciones binarias difieren entre sí en un único bit. Se pueden definir también como aquellos términos a los que se les puede aplicar la propiedad distributiva para simplificar una variable.

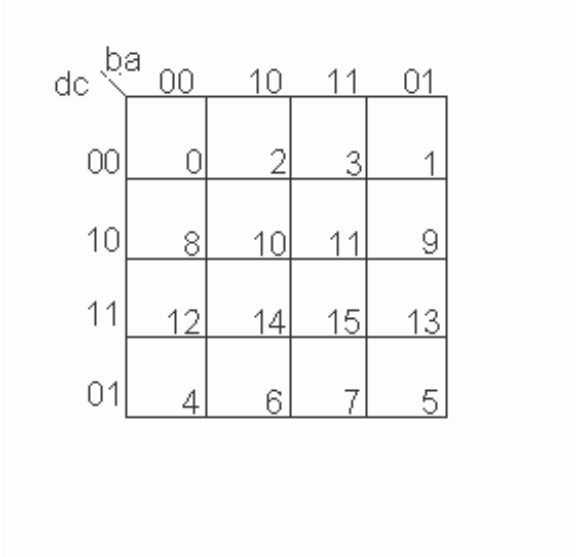
Mapa de Karnaugh para funciones de 2 variables:



Mapa de Karnaugh para funciones de 3 variables:



Mapa de Karnaugh para funciones de 4 variables:

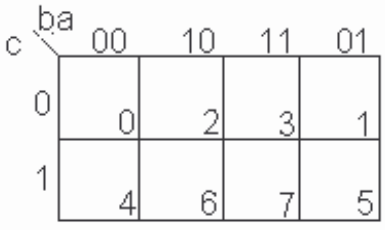


En los mapas de 3 y 4 variables se verifica que los cuadros opuestos en los extremos de una misma fila o columna también representan términos canónicos adyacentes.

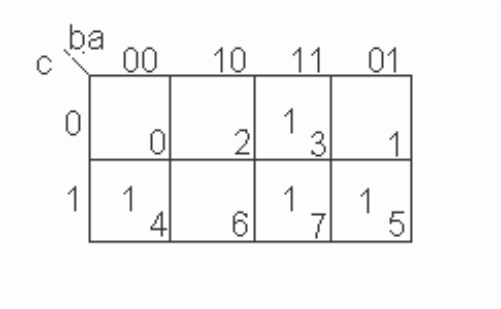
El procedimiento de simplificación mediante Mapas de Karnaugh, se indica a continuación, con la ayuda de un ejemplo que consiste en simplificar la función canónica...

f_3 (c,b,a) = 3_4 (3,4,5,7) = cba + cba + cba+ cba

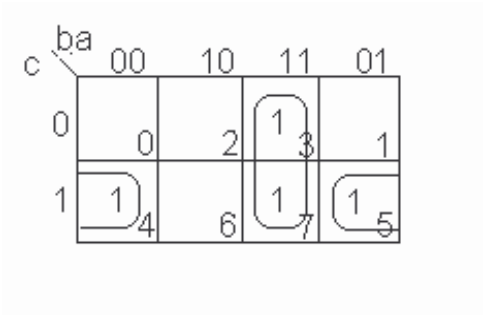
1) Se dibuja el mapa adecuado para la función a simplificar (2,3, ó 4 variables). Esta función puede venir dada en forma de minterms o maxterms.



2) Se escribe un "1" en los cuadros correspondientes a los minterm de la función, o un "0" si los términos son maxterm.



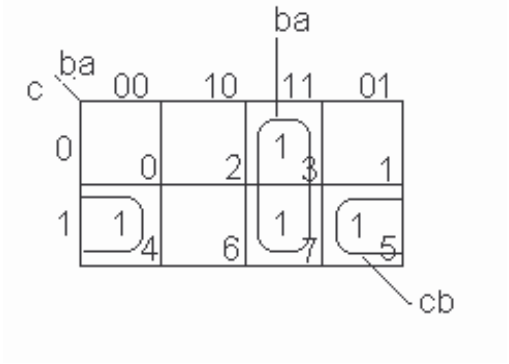
3) Se agrupan mediante una curva cerrada los grupos de dos "1" ó "0" adyacentes que no puedan formar grupos de cuatro.



4) Se agrupan también, si los hay, los grupos de cuatro "1" ó "0" adyacentes que no puedan formar grupos de 8, y los grupos de ocho que no puedan formar grupo de 16, etc. (No existen en el ejemplo).

5) Cada uno de los grupos así obtenidos da lugar a un término simplificado, mediante el siguiente criterio:

En cada grupo desaparece la variable o variables cuyo valor es 0 en la mitad de los cuadros del grupo, y 1 en la otra mitad. Las variables que permanecen son tomadas como "no negadas" si su valor es 1 en todo el grupo de cuadros, y

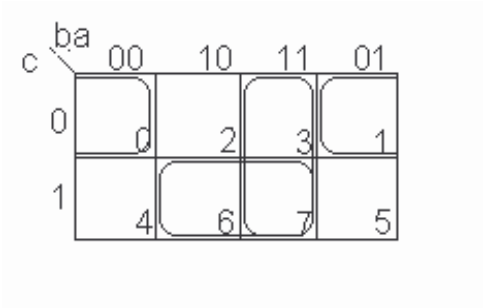


como "negada" si su valor es 0.

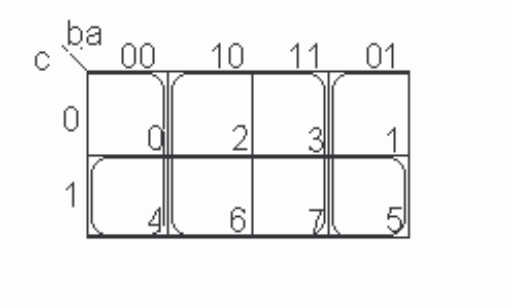
La función f_3 simplificada es ... f_3 (c,b,a) = cb + ba

Las formas de los posibles grupos en un mapa de 3 variables son las siguientes:

Grupos de 2

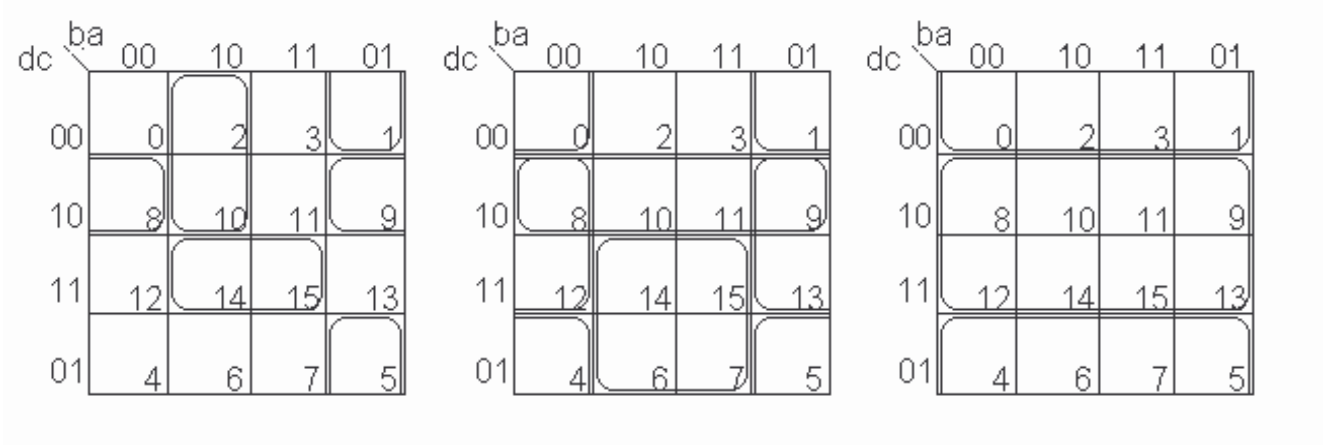


Grupos de 4



Las formas de los posibles grupos en un mapa de 4 variables son las siguientes:

Grupos de 2 ***Grupos de 4*** ***Grupos de 8***



Los cuadros con "1" ó "0" que no tengan ningún otro adyacente igual, representan minterms o maxterms respectivamente, que no pueden simplificarse y no se modifican en el resultado de la simplificación.

El número de unos o ceros en cada grupo debe ser siempre una potencia de 2, de modo que en un grupo de 2^n unos o ceros, se eliminan n variables de las que forman los términos canónicos.

La agrupación de cuadros debe ser tal que el número de grupos sea el mínimo posible, y cada grupo sea lo mayor posible. Un mismo "1" o "0" de un cuadro puede pertenecer a más de un grupo a la vez.

Si el método de simplificación se aplica a una función canónica Minterm, el resultado vendrá dado como suma de productos de variables. Si, por el contrario, se aplica a una función Maxterm, el resultado será un producto de sumas.

En general, si se desea una función simplificada al máximo, conviene realizar su simplificación en minterm y en maxterm, para ver cual es el resultado más sencillo.

Resolución de ejemplos

Ejemplo: f_4 (d,c,b,a) = 3_4 (1,3,7,8,11,13,15)

Simplificando por minterm, es...

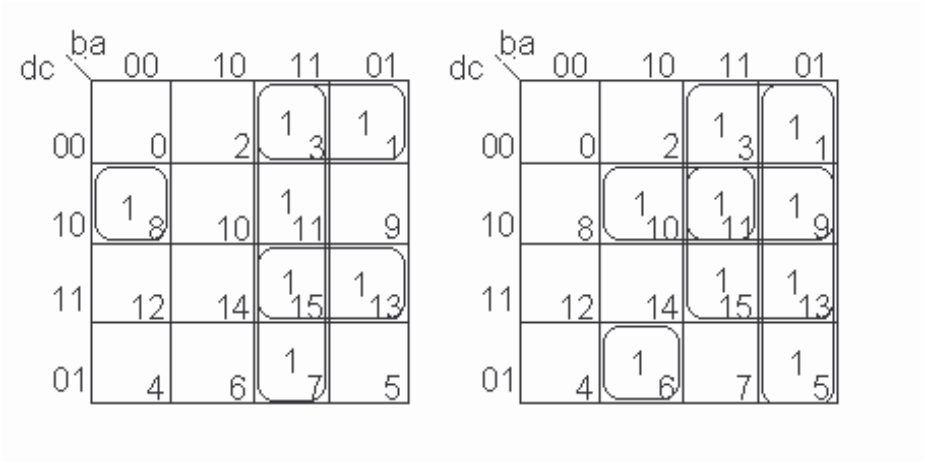
f_4 (d,c,b,a) = dcba + dca + dca + ba = dcba + a(d⊕c+b)

Pasando a maxterms...

f_4 (d,c,b,a) = ϑ_4 (1,3,5,6,9,10,11,13,15)

Simplificando por maxterm es...

f_4 (d,c,b,a) = (d+a)(c+a)(d+c+b)(b+a)(d+c+b+a) = (dcb+a)(d+c+b)(d+c+b+a)



Es preferible la primera simplificación, ya que se realiza con 5 puertas, y la segunda con 6.

Para simplificar funciones f(e,d,c,b,a) de 5 variables mediante Karnaugh, se hace uso de dos mapas de Karnaugh para 4 variables, uno para e = 0 (siendo e la variable de mayor peso) y otro para e = 1. El primer mapa (e=0) servirá para

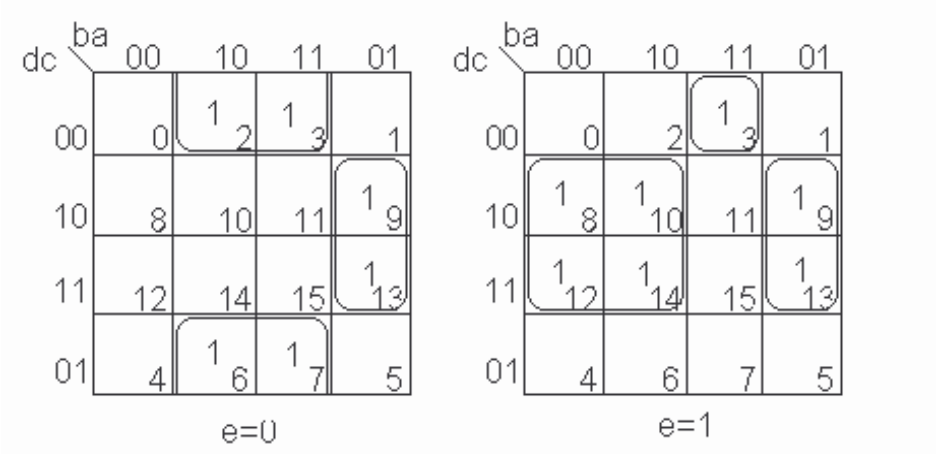
representar los téminos canónicos comprendidos entre 0 y 15, y el segundo (e=1) para los términos entre 16 y 31.

El criterio de adyacencia entre cuadros en cada mapa es el mismo que con 4 variables, pero ahora también existe adyacencia entre cuadros con posición idéntica en ambos mapas.

Resolución de ejemplos

Ejemplo:

f_5 (e,d,c,b,a) = 3_5 (2,3,6,7,9,13,19,24,25,26,28,29,30)



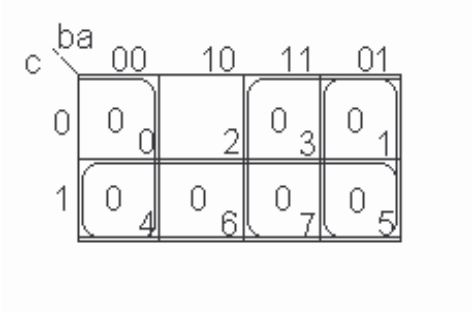
Simplificando es...
Grupo (2,3,6,7) ..... edb
Grupo (9,13,25,29) .... dba
Grupo (24,26,28,30) ... eda
Grupo (3,19) ..... dcba

f_5 (e,d,c,b,a) = edb + dba + eda + dcba

Las funciones de 6 variables se simplifican mediante 4 mapas de Karnaugh de 4 variables, existiendo adyacencia entre cuadros con posición idéntica en dos mapas contiguos, o con posición idéntica en los 4 mapas a la vez.

Resolución de ejemplos

A) f_7 (c,b,a) = ϑ_3 (0,1,3,4,5,6,7)



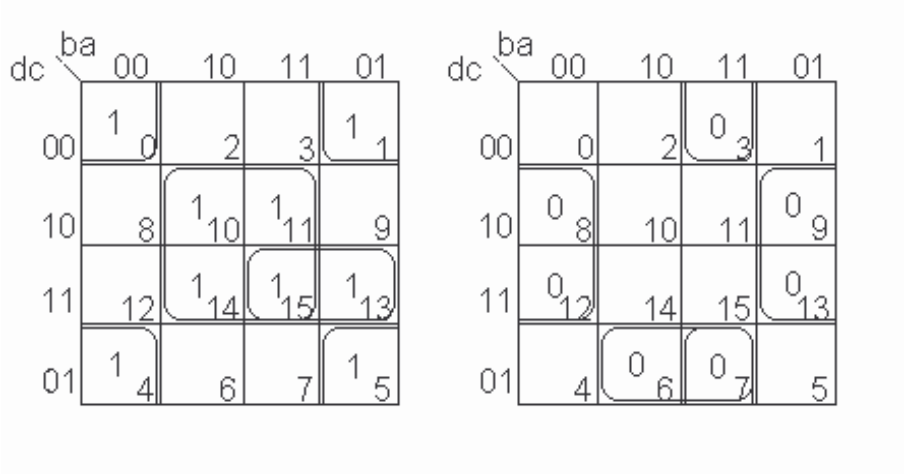
Simplificando por maxterms...

f_7 (c,b,a) = c*b*a

Pasando f_7 a minterms, se llega al mismo resultado:

f_7 (c,b,a) = cba

B) f_8 (d,c,b,a) = 3_4 (0,1,4,5,10,11,13,14,15)



Simplificando por minterms, es...

f_8 (d,c,b,a) = db+ db + dca = (d⊕b) + dca

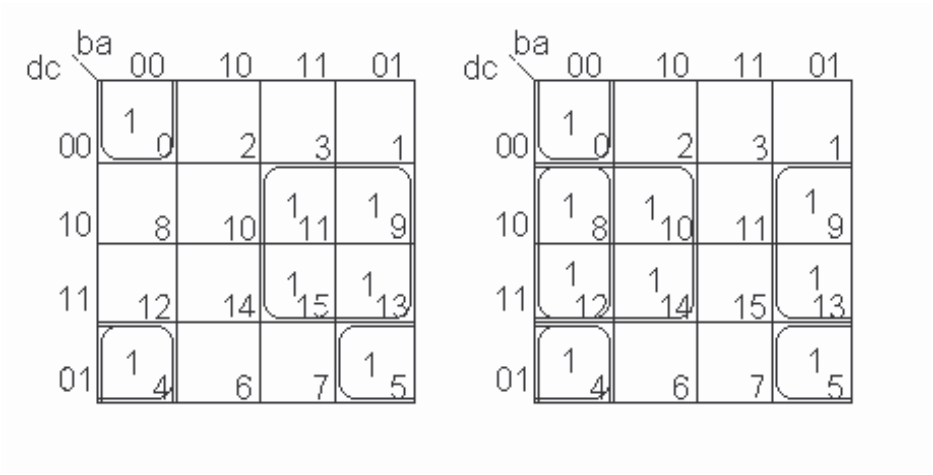
Pasando a maxterms y simplificando ...

f_8 (d,c,b,a) = ϑ_4 (13,12,9,8,7,6,3)

f_8 (d,c,b,a) = (d+b) (d+c+b) (d+b+a) = (d+b+ca) (d+b)

Es mejor la primera solución.

C) f_9 (e,d,c,b,a)= 3_5 (0,4,5,9,11,13,15,16,20,21,24,25,26,28,29,30)



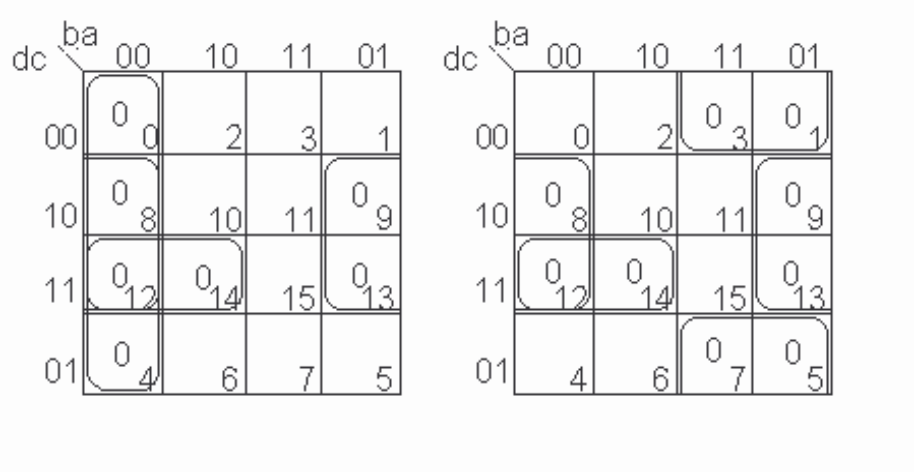
e = 0 ***e = 1***

Simplificando por minterms ...

f_9 (e,d,c,b,a)= eda + dba + dcb + eda + edb

Pasando a maxterms, es...

f_9 (e,d,c,b,a)= ϑ_5 (30,29,28,25,24,23,21,19,17,14,13,12,9,8,4,0)



e = 0 ***e = 1***

Simplificando...

f_9 (e,d,c,b,a)= (d+b) (e+b+a) (d+c+a) (e+d+a)

Conviene aquí la segunda simplificación

Método de McCluskey Es un método tabular numérico de simplificación apropiado para funciones con muchas variables, ya que, aunque es más laborioso y complicado que el método Karnaugh, este es difícil de aplicar a funciones con más de 6 variables.

Opera con el valor numérico decimal correspondiente a los términos canónicos de la función (minterm o maxterm), y es un método programable mediante ordenador.

Funciones incompletas

Son funciones cuyo valor puede ser indistintamente 0 o 1 para algunas de las combinaciones de las variables de entrada, bien porque dichas combinaciones no vayan a darse nunca en la práctica, o porque sea indiferente para el diseño el valor de la función para dichas combinaciones. Estos valores indeterminados de la función representan un aspa (X) en la tabla de verdad de la función

Las funciones incompletas se simplifican considerando dichos valores indeterminados de la función como 0 o 1, según nos convenga para obtener la mayor simplificación posible.

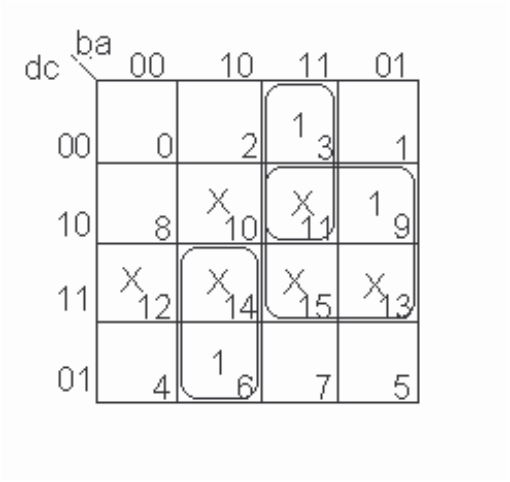
Simplificando por Karnaugh, se colocan aspas en los cuadros correspondientes a las configuraciones de entrada indeterminadas, y consideraremos dichas aspas como valor 1 cuando puedan formar grupos de unos mayores que si fueran 0, a fin de simplificar más.

Resolución de ejemplos

Diseñar un circuito simplificado que tenga por entrada una cifra decimal codificada en binario (de 0 a 9), y detecte a su salida múltiplos de tres.

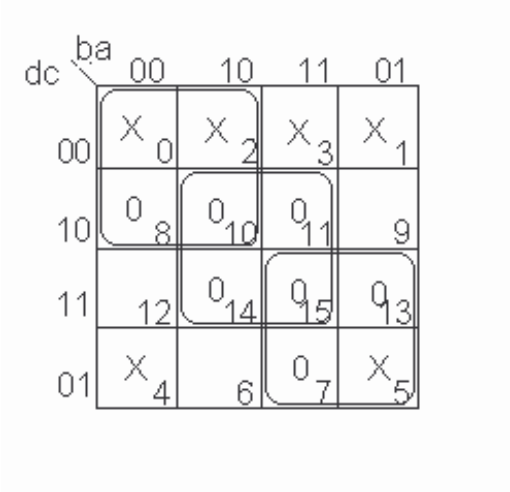
d c b a	F	d c b a	F
0 0 0 0	0	1 0 0 0	0
0 0 0 1	0	1 0 0 1	1
0 0 1 0	0	1 0 1 0	X
0 0 1 1	1	1 0 1 1	X
0 1 0 0	0	1 1 0 0	X
0 1 0 1	0	1 1 0 1	X
0 1 1 0	1	1 1 1 0	X
0 1 1 1	0	1 1 1 1	X

La terminología utilizada para describir abreviadamente estas funciones es...
f(d,c,b,a) = 3_4 (3,6,9) + 3_x (10,11,12,13,14,15)



f(d,c,b,a) = da + cba + cba

Pasando esta función a maxterms, queda:
f(d,c,b,a) = ϑ_4 (15,14,13,11,10,8,7) + ϑ_x (5,4,3,2,1,0)



Simplificando ...
f(d,c,b,a) = (c+a)(d+b)(c+a)
La simplificación por maxterm resulta aquí más reducida.

Funciones múltiples

Son grupos de dos o más funciones que dependen de las mismas variables de entrada, y que han de ser obtenidas simultaneamente a partir de estas. En la práctica, casi todos los circuitos combinacionales constituyen una multifunción, ya que tienen más de una linea de salida.

Resolución de ejemplos

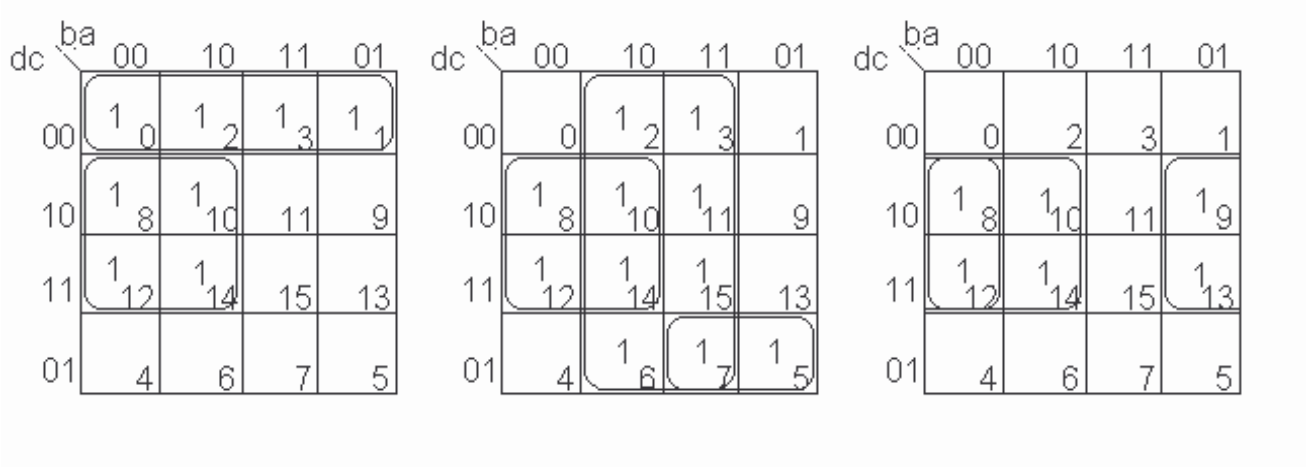
$f_1 = \Sigma_4 (0,1,2,3,8,10,12,14)$

$f_2 = \Sigma_4 (2,3,5,6,7,8,10,11,12,14,15)$

$f_3 = \Sigma_4 (8,9,10,12,13,14)$

d	c	b	a	f_1	f_2	f_3
0	0	0	0	1	0	0
0	0	0	1	1	0	0
0	0	1	0	1	1	0
0	0	1	1	1	1	0
0	1	0	0	0	0	0
0	1	0	1	0	1	0
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	1	1	1
1	0	0	1	0	0	1
1	0	1	0	1	1	1
1	0	1	1	0	1	0
1	1	0	0	1	1	1
1	1	0	1	0	0	1
1	1	1	0	1	1	1
1	1	1	1	0	1	0

Dado que suelen existir configuraciones de entrada para las cuales el valor de las distintas funciones es el mismo, conviene efectuar la simplificación de dichas funciones de forma conjunta. Suele realizarse por karnaugh esta simplificación intentando encontrar agrupaciones de términos que sean comunes a todas o a algunas de las funciones, agrupaciones que darán lugar cada una a un término simplificado común, es decir, a una puerta lógica compartida por varias funciones:



$f_1 =$ + dc $\qquad$ $f_2 = b + da + dca$ $\qquad$ $f_3 = da + db$

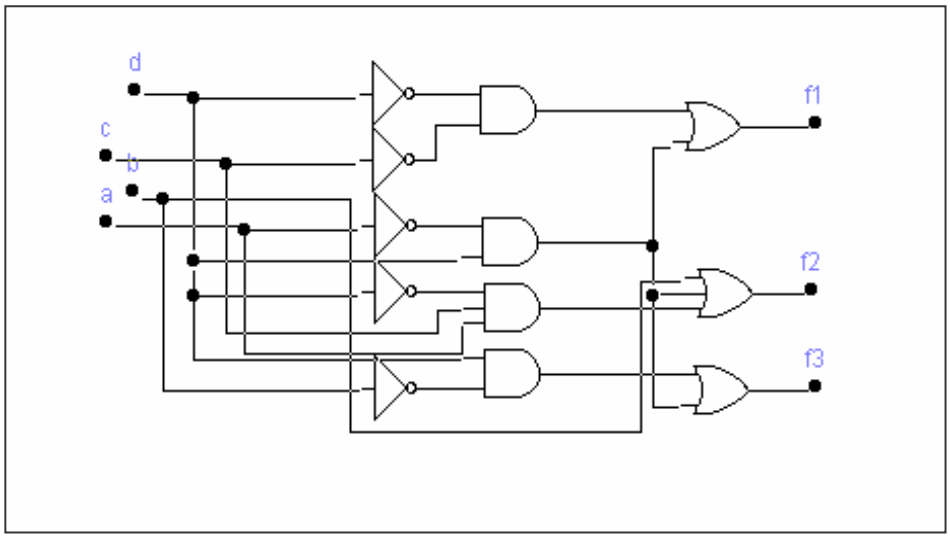


Figura 4.1. Implementación