TEMA - 2

ÁLGEBRA DE BOOLE. LÓGICA COMBINACIONAL.

El control digital, y en particular el binario, está presente en todos los campos de la vida, desde los sistemas de refrigeración hasta los complejos sistemas de control de vuelo. Aunque los circuitos electrónicos de estos sistemas pueden tener niveles de complejidad muy diferentes, todos se basan en combinaciones de elementos más pequeños llamados **puertas lógicas**, las cuales se construyen a partir de transistores y elementos pasivos.

En este tema se aborda el estudio de dichas puertas lógicas, el álgebra de conmutación que se utiliza para manipular las magnitudes binarias y algunas aplicaciones.

1. Estados lógicos y función lógica.

Los elementos que constituyen los circuitos digitales se caracterizan por admitir sólo dos estados. Es el caso por ejemplo de un conmutador que sólo puede estar ENCENDIDO o APAGADO, o una válvula hidráulica que sólo pueda estar ABIERTA o CERRADA.

Para representar estos dos estados se usan los símbolos '0' y '1'. Generalmente, el '1' se asociará al estado de conmutador CERRADO, ENCENDIDO, VERDADERO, y el '0' se asocia al estado de conmutador ABIERTO, APAGADO o FALSO.

En el circuito de la Figura 2-1 se representa el estado del conmutador con la variable S y el de la lámpara con la variable binaria L. En la tabla se observa la relación entre ambas.

S	L
ABIERTO	APAGADA
CERRADA	ENCENDIDA

Tabla de verdad

S	L
0	0
1	1

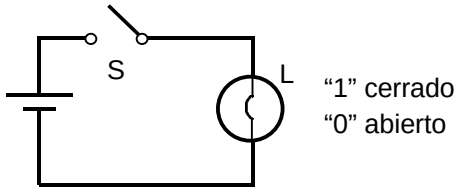


Figura 2-1. Circuito binario.

La **función lógica** es aquella que relaciona las entradas y salidas de un circuito lógico. Puede expresarse mediante:

1. **Tabla de verdad**: Es ella se representan a la izquierda todos los estados posibles de las entradas (en el ejemplo, el estado del conmutador) y a la derecha los estados correspondientes a la salida (en el ejemplo, la lámpara).

2. **Función booleana**: Es una expresión matemática que emplea los operadores booleanos (en el ejemplo, L = S).

2. Puertas lógicas elementales.

Una puerta lógica es un elemento que toma una o más señales binarias de entrada y produce una salida binaria función de estas entradas. Cada puerta lógica se representa mediante un símbolo lógico. Hay tres tipos elementales de puertas: AND, OR y NOT. A partir de ellas se pueden construir otras más complejas, como las puertas: NAND, NOR y XOR.

2.1. Puerta AND.

El funcionamiento de la puerta lógica AND es equivalente al de un circuito con dos conmutadores en serie como el de la Figura 2-2. En dicho circuito es necesario que los dos conmutadores estén cerrados para que la lámpara se encienda.

La relación entre las posiciones de los conmutadores y el estado de la lámpara se muestra en la tabla de verdad.

A	B	L
0	0	0
0	1	0
1	0	0
1	1	1

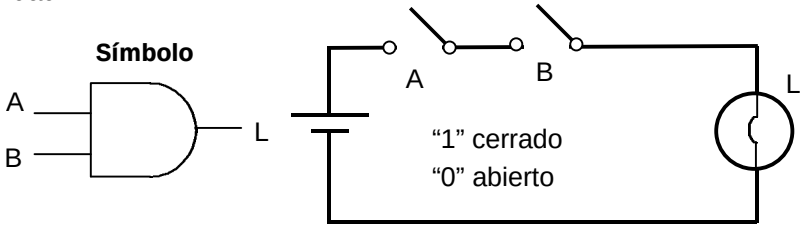


Figura 2-2. Circuito equivalente a una puerta AND de dos entradas.

La relación es la siguiente: la lámpara se enciende sólo si el conmutador A Y el conmutador B están a '1', es decir, L = A (AND) B. Esta relación se conoce como **AND**.

Las puertas AND pueden tener más de dos entradas. En la Figura 2-3 se representa una puerta AND de tres entradas.

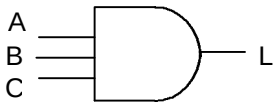


Figura 2-3. AND de tres entradas.

La salida de una puerta AND es verdadera ('1') si, y sólo si, todas las entradas son verdaderas. Esta operación corresponde a una multiplicación lógica binaria que para dos entradas sería: $L = A \cdot B$.

2.2. Puerta OR.

El funcionamiento de esta puerta es equivalente al de dos conmutadores en paralelo como en la Figura 2-4. En esta configuración la lámpara se encenderá si cualquiera de los dos conmutadores se cierra.

A	B	L
0	0	0
0	1	1
1	0	1
1	1	1

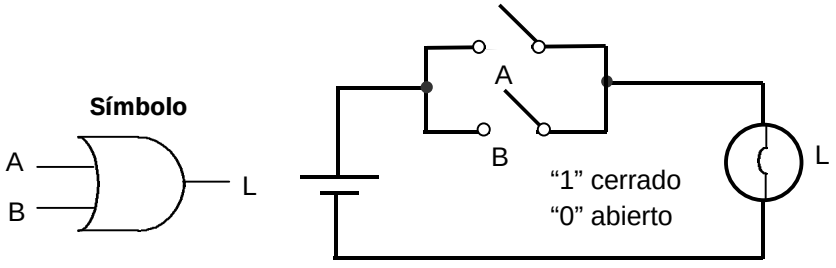


Figura 2-4. Circuito equivalente a una puerta OR de dos entradas.

En este caso la relación es la siguiente: la lámpara se encenderá si y sólo si, el conmutador A O (OR) el B están cerrados. Esta función se describe en la tabla de verdad.

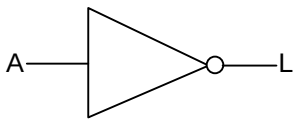
La salida de una puerta OR es verdadera ('1') si, y sólo si, al menos una de las entradas es verdadera. Esta relación corresponde a una suma lógica binaria**:** L= A + B.

2.3. Puerta NOT.

La salida de una puerta NOT es siempre el complementario de la entrada, de tal manera que si la entrada es '0' la salida es '1' y viceversa. Se conoce también como INVERSOR y posee una única entrada.

A	L
0	1
1	0

Símbolo

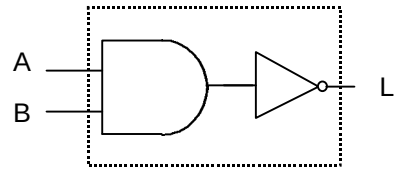


La operación lógica se conoce como negación y se escribe: $L = \overline{A}$ (negado de A).

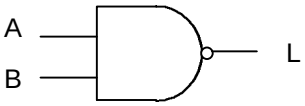
El indicador de negación es un círculo (◦) que indica inversión o complementación cuando aparece en la entrada o en la salida de un elemento lógico. El símbolo triangular sin el círculo representaría una función en la que el estado de la salida sería idéntico al de la entrada, esta función recibe el nombre de *buffer*. Los *buffers* se usan para cambiar las propiedades eléctricas de una señal sin afectar al estado lógico de la misma.

2.4. Puerta NAND.

Equivale a una puerta AND seguida de un INVERSOR. Su nombre viene de *Not-AND* . El símbolo lógico es una puerta AND con un círculo en la salida. La tabla de verdad es igual al de la puerta AND con el estado de salida negado. Una puerta NAND puede tener más de dos entradas.



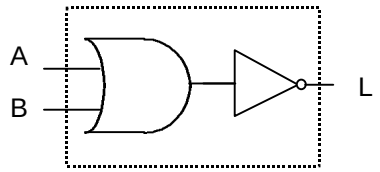
Símbolo



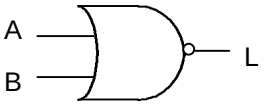
A	B	L
0	0	1
0	1	1
1	0	1
1	1	0

2.5 Puerta NOR.

Equivale a una puerta OR seguida de un INVERSOR. Su nombre viene de *Not-OR* . El símbolo lógico es una puerta OR con un círculo en la salida. La tabla de verdad es igual al de la puerta OR con el estado de salida negado. También puede tener más de dos entradas.



Símbolo



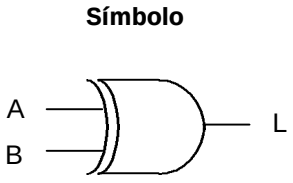
A	B	L
0	0	1
0	1	0
1	0	0
1	1	0

2.6. Puerta OR exclusiva (XOR).

La salida de una puerta OR exclusiva es verdadera ('1') si, y sólo si, una y sólo una de sus dos entradas es verdadera. Se asemeja a la OR (inclusiva), excepto que excluye el caso en que las dos entradas son verdaderas. La figura muestra un circuito equivalente. En una puerta OR exclusiva la salida será '1' cuando el número de entradas que son '1' sea impar.

El circuito equivalente de la Figura 2-6 se deriva de considerar el funcionamiento de al puerta XOR como combinación de dos condiciones X e Y. X representa la condición de que cualquiera de las entradas: A **o** (OR) B sea '1', e Y la condición de que A **y** (AND) B **no** (NOT) sean '1' (NAND).

A	B	L
0	0	0
0	1	1
1	0	1
1	1	0



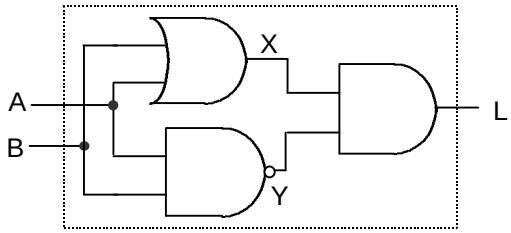


Figura 2-6. Circuito equivalente a una puerta XOR.

A	B	C	L
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

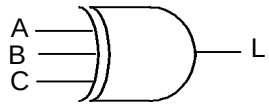
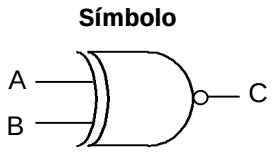


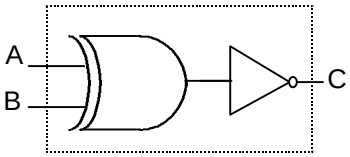
Figura 2-5. XOR de tres entradas.

2.7. Puerta NOR exclusiva.

Es la negación de la puerta OR exclusiva (puerta OR seguida de un INVERSOR).

A	B	C
0	0	1
0	1	0
1	0	0
1	1	1





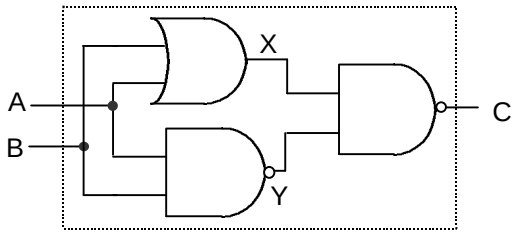


Figura 2-7. Circuito equivalente a una NOR exclusiva.

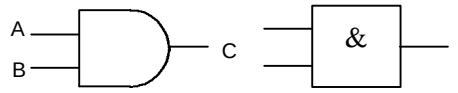
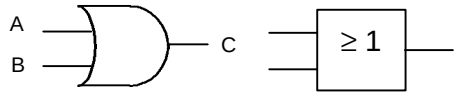
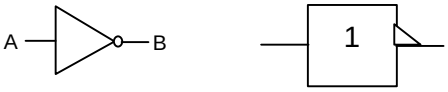
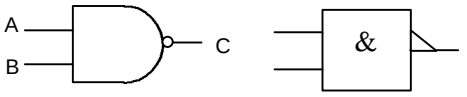
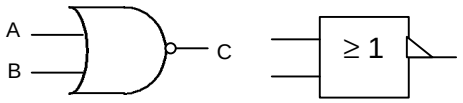
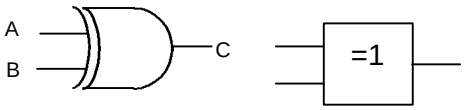
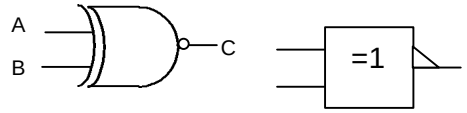
3. Algebra de Boole.

Proporciona una notación para describir funciones lógicas y define un número de operaciones que se pueden realizar con el fin de simplificarlas.

El álgebra de Boole define *variables, constantes y funciones* para describir sistemas binarios, y una serie de *teoremas* que permiten manipular expresiones lógicas.

- **Constantes booleanas**: Se definen dos: '0' (estado FALSO) y '1' (VERDADERO).
- **Variables booleanas**: Son magnitudes que pueden tomar diferentes valores en diferentes momentos. Pueden representar señales de entrada o de salida y reciben nombres de caracteres alfabéticos como: A, B, X, Y. Sólo pueden tomar los valores '0' o '1'.
- **Funciones booleanas**: Describen el comportamiento del sistema. Cada operación lógica (suma, multiplicación, negación, ...) posee una notación en el álgebra booleana, como se muestra en la Tabla 2-1.

Tabla 2-1. Funciones lógicas elementales.

Función	Símbolo	Notación	Tabla de verdad
AND		$C = A \cdot B$	A B C 0 0 0 0 1 0 1 0 0 1 1 1
OR		$C=A+B$	A B C 0 0 0 0 1 1 1 0 1 1 1 1
NOT		$B = \overline{A}$	A B 0 1 1 0
NAND		$C = \overline{A \cdot B}$	A B C 0 0 1 0 1 1 1 0 1 1 1 0
NOR		$C = \overline{A + B}$	A B C 0 0 1 0 1 0 1 0 0 1 1 0
EXOR		$C = \overline{A}\,B + A\overline{B}$ $C = A \oplus B$	A B C 0 0 0 0 1 1 1 0 1 1 1 0
NOR exclusiva		$C = \overline{A} \cdot \overline{B} + A \cdot B$ $C = \overline{A \oplus B}$	A B C 0 0 1 0 1 0 1 0 0 1 1 1

En la Tabla 2-1 además de los símbolos distintivos vistos con anterioridad se muestran los símbolos rectangulares que con frecuencia se emplea en la documentación industrial. En estos símbolos el indicador de negación en lugar de un círculo (◦) es un triángulo (◺) que indica inversión cuando se coloca a la entrada o en la salida de un elemento lógico.

Ejemplo 2-1. Extracción de la expresión booleana de un circuito a partir de su tabla de verdad.

A	B	C
0	0	0
0	1	1
1	0	0
1	1	1

$$C = (\overline{A} \cdot B) + (A \cdot B) = \overline{A}B + AB$$

Esta expresión se ha extraído de la tabla tan sólo mediante la descripción de los estados de A y B para cada línea en la que C es '1' y uniéndolos mediante la función OR. Las funciones booleanas que describen el comportamiento de un sistema binario las podemos expresar de dos formas: en **minterms** o en **maxterms**.

a) Se genera un **minterm** por cada fila de la tabla de verdad donde la salida es '1'.

   1. El minterm contiene el producto de cada variable de entrada en orden. La entrada está no negada si para esa combinación es un '1' y negada si es un '0'.

2. La expresión global para la función lógica es suma de los minterms.

b) Se genera un **maxterm** por cada fila de la tabla de verdad en la que la salida es '0'.

1. El maxterm contiene la suma de cada variable de entrada en orden. La entrada está no negada si es un '0' y negada si es un '1' (al contrario que en minterms).

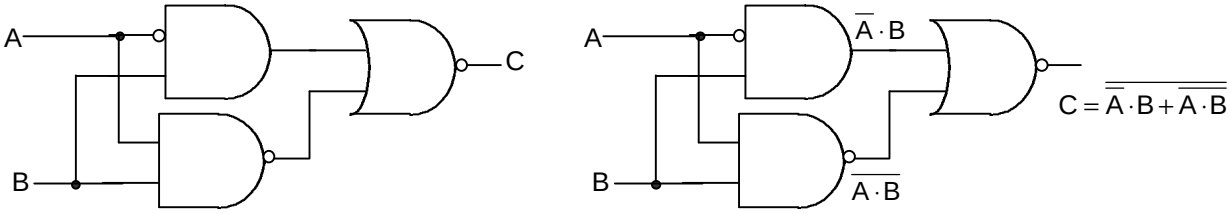
2. La expresión global para la función lógica es producto de los maxterms.

Para el ejemplo anterior sería: $C = (A + B) \cdot (\overline{A} + B)$

La **función canónica** es aquella en la que están presentes en cada minterm o en cada maxterm todas las variables de entrada, es decir, está sin simplificar.

Ejemplo 2-2. Obtención de la expresión booleana de un circuito a partir del diagrama lógico.

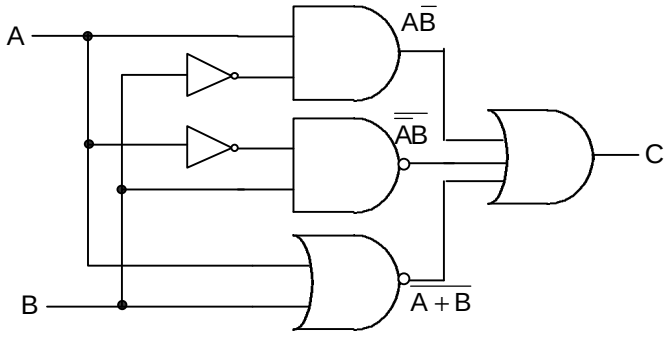
El método más sencillo es escribir sobre el diagrama la salida de cada puerta lógica.



Ejemplo 2-3. Generación de un diagrama lógico de un sistema a partir de su expresión booleana.

Considerar la expresión: $C = A\overline{B} + \overline{\overline{A}B} + (\overline{A + B})$

La función tiene tres componentes unidos por la función OR, por tanto, la salida vendrá de un puerta OR de tres entradas. Las entradas de esta puerta serán los tres componentes de la expresión: la 1ª, $A\overline{B}$ proviene de una puerta AND de dos entradas A y $\overline{B}$; la 2ª de una NAND de entradas $\overline{A}$ y B, y la 3ª de una puerta NOR de dos entradas.



3.1. Teoremas booleanos.

Hasta ahora se ha visto como generar expresiones booleanas para describir una función especificada en una tabla de verdad o un diagrama lógico, pero estas expresiones no son siempre las más sencillas. El álgebra de Boole define varios teoremas para simplificar dichas expresiones.

Ley conmutativa: AB = BA
A + B = B+A

Ley distributiva: A(B+C) = AB + AC
A + BC = (A+B) (A + C)

Ley asociativa: A(BC) = (AB)C
A+(B+C)=(A+B)+C

Ley de la absorción A + AB= A
A(A+B)=A

Ley de DeMorgan $\overline{A+B}=\overline{A}\cdot\overline{B}$
$\overline{A\cdot B}=\overline{A}+\overline{B}$

$A+\overline{A}B=A+B$
$A(\overline{A}+B)=AB$

Operación suma lógica (OR) el resultado es "1" si alguno de los sumandos es "1"
1+A=1
0+A=A
A+A=A
$A+\overline{A}=1$

Operación producto lógico (AND) el resultado es "0" si alguno de los elementos es "0"
1·A=A
0·A=0
A·A=A
$A\cdot\overline{A}=0$

Operación negación (NOT) $\overline{0}=1$
$\overline{1}=0$
$\overline{\overline{A}}=A$

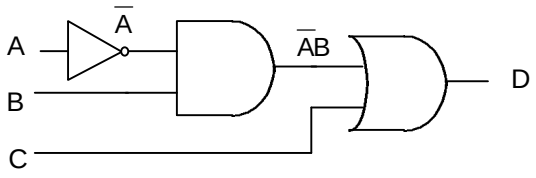
4. Simplificación de funciones.

4.1. Mediante la aplicación de los teoremas.

Para simplificar una expresión algebraica se pueden aplicar los teoremas booleanos vistos con anterioridad.

Ejemplo 2-4. $D = \overline{B}C + \overline{A}BC + ABC + \overline{A}B\overline{C}$, se puede reducir:

$$D = \overline{B}\,C + \overline{A}\,B\,C + A\,B\,C + \overline{A}\,B\,\overline{C}$$
$$D = \overline{B}\,C + \overline{A}\,B\,\overline{C} + B\,C\,(\overline{A} + A)$$
$$D = \overline{B}\,C + \overline{A}\,B\,\overline{C} + B\,C$$
$$D = \overline{A}\,B\,\overline{C} + C\,(\overline{B} + B)$$
$$D = \overline{A}\,B + C$$

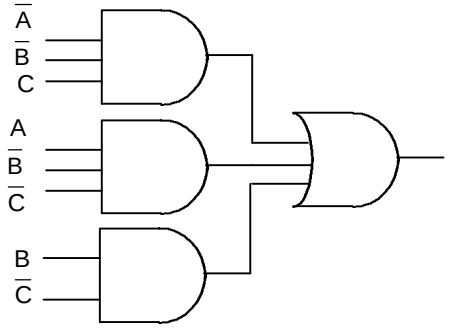


4.2. Homogeneización de una función con puertas NAND.

A menudo es más sencillo y económico a la hora de realizar un circuito emplear sólo un tipo de puerta lógica. En varias familias lógicas las puertas NAND son las más simples, por lo que resulta útil poder construir circuitos usando sólo éstas.

Ejemplo 2-5. Homogeneización con puertas NAND de una expresión dada en forma de minterms:

$$D = \overline{A}\,\overline{B}\,C + A\,\overline{B}\,\overline{C} + B\,\overline{C}$$

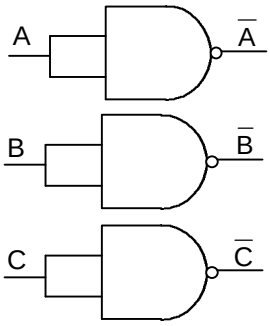


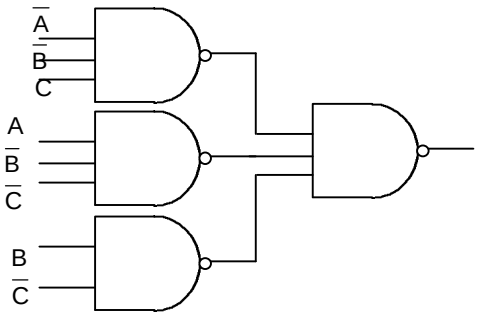
- En primer lugar hay que negar dos veces toda la expresión:

$$D = \overline{\overline{\overline{A}\,\overline{B}\,C + A\,\overline{B}\,\overline{C} + B\,\overline{C}}}$$

- Y aplicar el 1º teorema de DeMorgan:

$$D = \overline{\overline{(\overline{A}\,\overline{B}\,C)} \cdot \overline{(A\,\overline{B}\,\overline{C})} \cdot \overline{(B\,\overline{C})}}$$



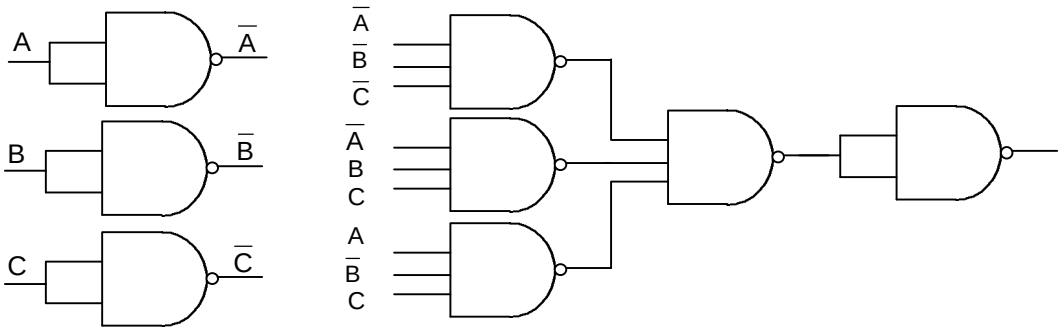


Ejemplo 2-6. Homogeneización con puertas NAND de una expresión dada en forma de maxterms:
$D = (A + B + C) \cdot (A + \overline{B} + \overline{C}) \cdot (\overline{A} + B + \overline{C})$

- Se niega dos veces cada elemento del producto y dos veces toda la expresión:

$$D = \overline{\overline{(A + B + C)} \cdot \overline{(A + \overline{B} + \overline{C})} \cdot \overline{(\overline{A} + B + \overline{C})}}$$

- Se aplica el 1º teorema de DeMorgan: $D = \overline{(\overline{A} \cdot \overline{B} \cdot \overline{C}) \cdot (\overline{A} \cdot B \cdot C) \cdot (A \cdot \overline{B} \cdot C)}$



4.3. Homogeneización de una función con puertas NOR.

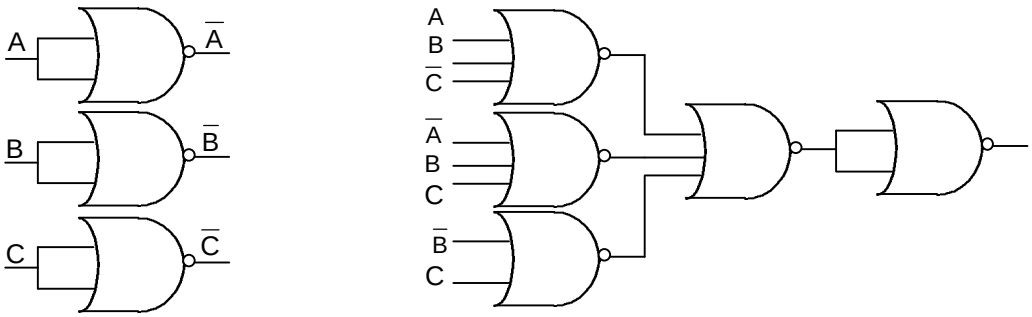
En algunas familias lógicas las puertas NOR son las más simples.

Ejemplo 2-7. Homogeneización con puertas NOR de una expresión dada en forma de minterms:
$D = \overline{A}\,\overline{B}\,C + A\,\overline{B}\,\overline{C} + B\,\overline{C}$

- Se niega dos veces cada sumando y dos veces toda la función:

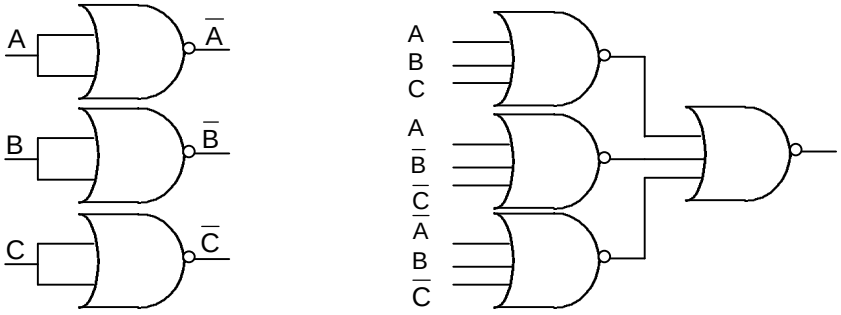
$$D = \overline{(\overline{A}\,\overline{B}\,C) + (A\,\overline{B}\,\overline{C}) + (B\,\overline{C})}$$

- Se aplica el 2º teorema de DeMorgan: $D = \overline{(A + B + \overline{C}) + (\overline{A} + B + C) + (\overline{B} + C)}$



Ejemplo 2-8. Homogeneización con puertas NOR de una expresión dada en forma de maxterms:
$D = (A+B+C)\cdot(A+\overline{B}+\overline{C})\cdot(\overline{A}+B+\overline{C})$

- Se niega dos veces toda la función: $D = \overline{\overline{(A+B+C)\cdot(A+\overline{B}+\overline{C})\cdot(\overline{A}+B+\overline{C})}}$
- Se aplica el 2º teorema de DeMorgan: $D = \overline{\overline{(A+B+C)}+\overline{(A+\overline{B}+\overline{C})}+\overline{(\overline{A}+B+\overline{C})}}$



4.4 Mapas de Karnaugh.

Es un método gráfico de representación de la información que se encuentra en la tabla de verdad. Permite simplificar una función booleana de manera sencilla. En un mapa de Karnaugh cada combinación posible de entradas está representada por una caja dentro de una rejilla, y el valor correspondiente de la salida se escribe dentro de la caja. Las cajas están escritas de forma que al cambiar de una a otra sólo varía una de las entradas. La secuencia corresponde al código Gray.

Mapa de Karnaugh de dos entradas

A	B	C
0	0	0
0	1	0
1	0	1
1	1	0

C \ A	0	1
B 0	0	1
1	0	0

Mapa de Karnaugh de tres entradas

D \ AB	00	01	11	10
C 0				
1				

Mapa de Karnaugh de cuatro entradas

E \ AB	00	01	11	10
CD 00				
01				
11				
10				

Simplificación del mapa de Karnaugh.

Se pueden agrupar **dos términos** adyacentes porque por características del mapa de Karnaugh sabemos que sólo difieren en el estado de una entrada. Por tanto, cualquier par de elementos adyacentes que contenga un '1' se pueden representar mediante una expresión simplificada.

Los '1' adyacentes se suelen marcar con una línea que los bordea.

Ejemplo 2-9. Simplificación de una función a partir del mapa de Karnaugh.

F	AB 00	01	11	10
CD 00	0	0	0	0
01	0	1	1	0
11	0	0	0	0
10	0	0	0	0

A partir del mapa de Karnaugh se puede extraer la expresión algebraica de forma sencilla: $F = \overline{A}B\overline{C}D + AB\overline{C}D$

Se aprecia fácilmente que la función F se puede simplificar: $F = B\overline{C}D(\overline{A} + A) = B\overline{C}D$

Al simplificar se pierde el efecto de la variable que está presente tanto en su forma negada ( ) como en su forma normal (A). Es decir, cuando B='1', C='0' y D='1', la salida será verdadera independientemente del valor de la variable A (A='1' o A='0').

Combinación de pares adyacentes en el mapa de Karnaugh.

E	AB 00	01	11	10	
CD 00	0	1	1	0	$B\overline{C}\overline{D}$
01	0	0	0	0	$\overline{A}CD$
11	1	1	0	1	
10	0	0	0	1	$A\overline{B}C$

E	AB 00	01	11	10	
CD 00	0	0	1	0	$AB\overline{D}$
01	1	0	0	1	$\overline{B}\overline{C}D$
11	0	0	0	0	
10	1	0	1	1	$\overline{B}C\overline{D}$

La fila superior e inferior se consideran adyacentes, al igual que las columnas derecha e izquierda.

Se puede simplificar también agrupando **cuatro términos** adyacentes. Se pueden combinar cuatro '1' siempre que representen todas las combinaciones de dos variables.

Ejemplo 2-10. Simplificación de una función a partir del mapa de Karnaugh.

E	AB 00	01	11	10
CD 00	0	0	0	0
01	0	1	1	0
11	0	1	1	0
10	0	0	0	0

Si se agrupan de dos en dos los '1' se tiene:

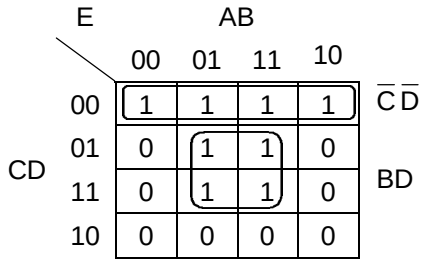
$$E = B\overline{C}D + BCD$$

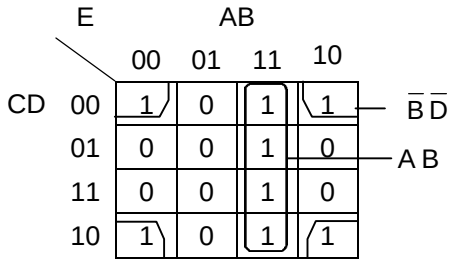
Que se puede simplificar aún más:

$$E = BD(C + \overline{C}) = BD$$

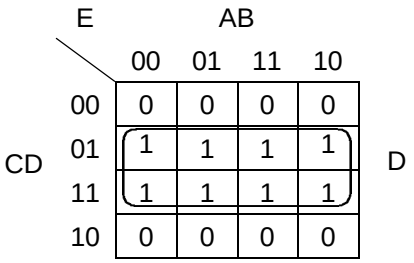
Como la salida es verdadera si B y D son verdaderas sin importar el estado de A y de C, estas dos últimas entradas se pueden eliminar de la expresión.

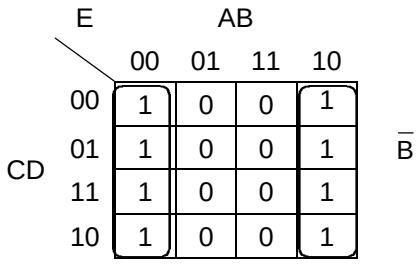
Combinaciones de cuatro elementos en el mapa de Karnaugh.





La simplificación también se puede realizar agrupando **ocho términos** adyacentes. En general los grupos pueden ser de 2^m elementos, donde m =1,2, … n (n = número de variables de entrada).





Para realizar las agrupaciones se siguen las siguientes reglas:

1. Primero se construirán los grupos de celdas más grandes posibles.
2. Agregar grupos más pequeños, hasta que cada celda que contenga un '1' se haya incluido al menos una vez.
3. Eliminar los grupos redundantes, aún cuando se trate de grupos grandes.

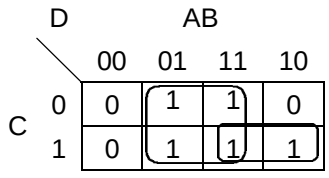
Los mapas de Karnaugh también se pueden emplear para simplificar expresiones con más de cuatro variables de entrada, pero el método se complica. Por lo general para muchas entradas se emplean técnicas de ordenador automatizadas, como el método desarrollado por McCluskey.

Condiciones irrelevantes.

Cuando el estado de una variable de salida no está definido, es decir, puede ser '0' o '1', se representará con una X y podremos elegir su valor para simplificar al máximo la función de salida.

Ejemplo 2-11. Consideremos la función: $D = A\,\overline{B}\,C + A\,B\,\overline{C} + \overline{A}\,B\,\overline{C} + A\,B\,C + \overline{A}\,B\,C$

Se puede representar por: $D = B + A\,C$



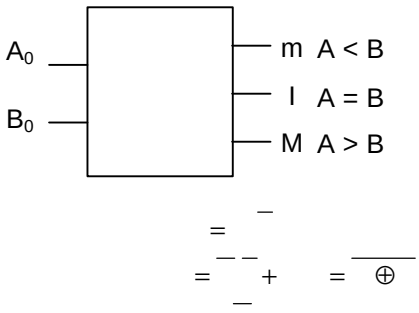
5.- Sistemas combinacionales. Funciones lógicas básicas.

Las puertas básicas pueden combinarse para formar circuitos lógicos más complejos que realicen muchas operaciones útiles. Algunas de las funciones lógicas combinacionales más comunes son: comparación, aritmética, conversión de códigos, codificación, decodificación y selección de datos.

5.1. Comparador binario.

La comparación de magnitudes se realiza mediante un circuito lógico denominado **comparador**. Un número en formato binario se introduce en la entrada *A* y otro en la entrada *B*. Las salidas *M, I, m,* indican la relación entre los dos números, produciendo un nivel alto en la línea de salida correspondiente, es decir, *M* ='1' si *A*>*B*, *I* ='1' si *A*=*B* y *m* ='1' si *A*<*B* .

Comparador de números de un bit.

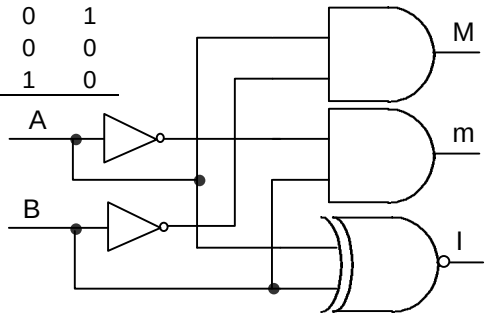


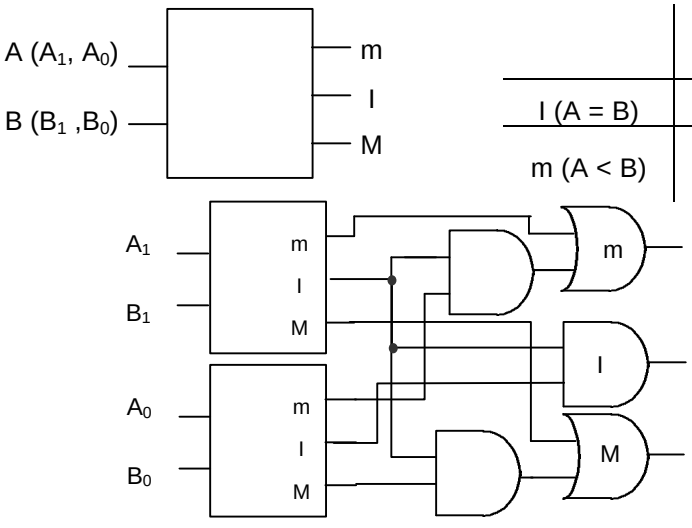
A_0	B_0	M	I	m
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

$= \quad \bar{} \quad$

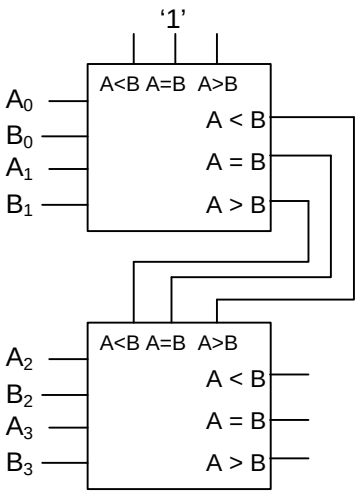
$= \quad \bar{} \bar{} \quad + \quad = \quad \overline{\quad\quad} \oplus$

$\bar{}$





I (A = B)	$A_1 = B_1 \Rightarrow A_0 > B_0$ $A_1 = B_1$ y $A_0 = B_0$
m (A < B)	$A_1 < B_1$ $A_1 = B_1 \Rightarrow A_0 < B_0$



Comparador de números de cuatro bits.

A partir de comparadores de números de dos bits se pueden construir comparadores mayores uniéndolos en cascada.

5.2. Funciones aritméticas. Suma.

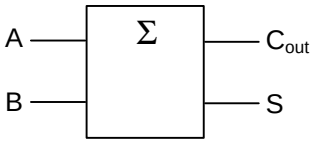
Los sumadores son muy importantes no sólo en los ordenadores, sino en muchos tipos de sistemas digitales.

Semi-sumador binario.

Recordemos las reglas básicas de la adición binaria:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10

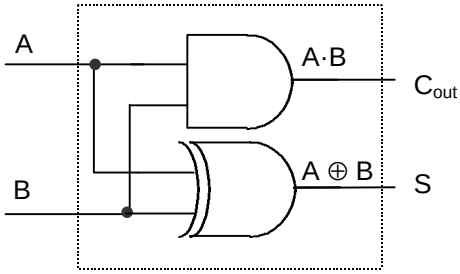
La función del semi-sumador es sumar dos números binarios que se aplican a las entradas A y B y generar la suma Σ y un acarreo de salida C_{out}.



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

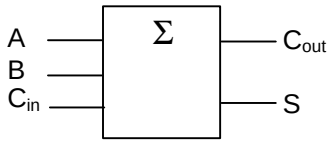
$C_{out} = AB$

$S = \overline{A}B + A\overline{B} = A \oplus B$



Sumador completo.

A diferencia del anterior, un sumador completo tiene tres entradas porque incluye una entrada de acarreo C_{in}.

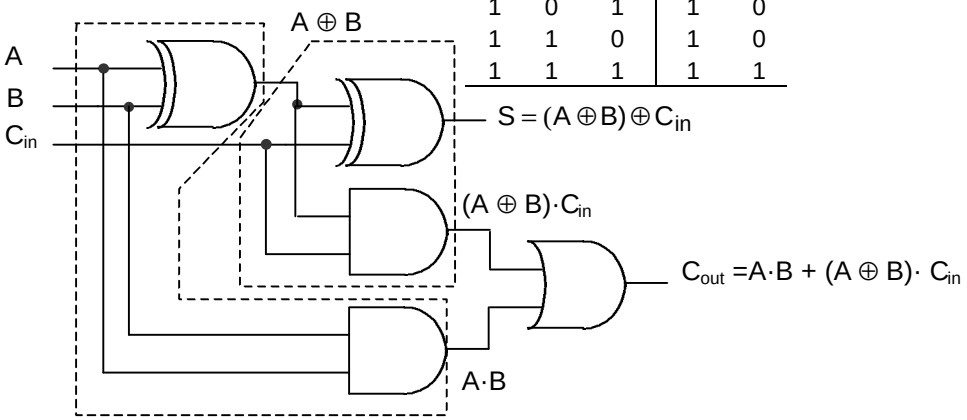


A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

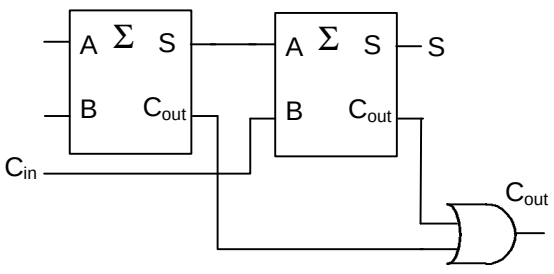
$C_{out} = A\,B + A\,C_{in} + B\,C_{in}$

$S = \overline{A}\,\overline{B}\,C_{in} + \overline{A}\,B\,\overline{C_{in}} + A\overline{B}\,\overline{C_{in}} + A\,B\,C_{in}$

$S = (A \oplus B) \oplus C_{in}$

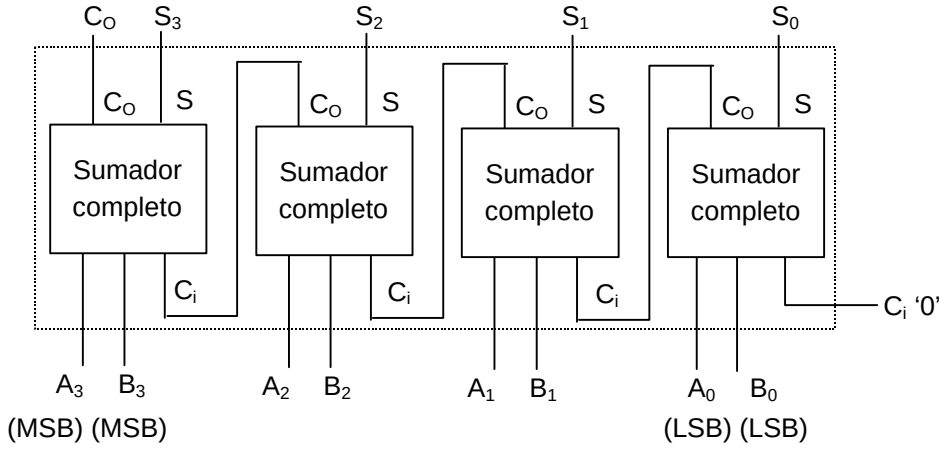


Sumador completo a partir de semi-sumadores.



Sumador de números de más de un 1 bit. Sumadores binarios en paralelo.

Para implementar la suma de números binarios se requieren tantos sumadores completos como bits tengan los números que se quieren sumar. La salida de acarreo de cada sumador se coloca a la entrada de acarreo del sumador de orden inmediatamente superior



5.3. Función de conversión de código.

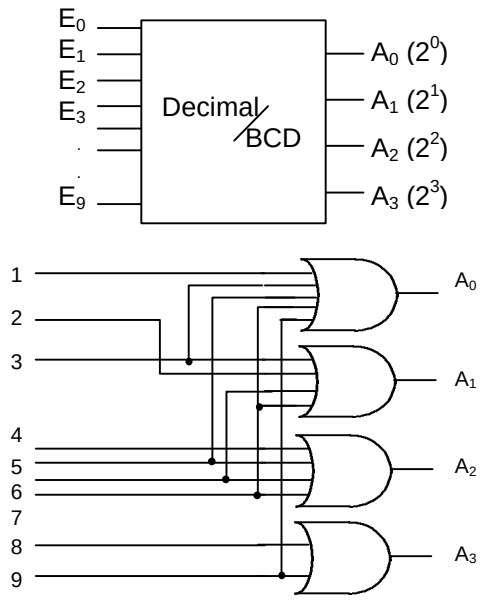
Un código es un conjunto de bits ordenados de acuerdo a un modelo que se emplean para representar información. Un convertidor de código cambia el formato de una información codificada a otro formato de código.

5.3.1. Función de codificación.

Se implementa mediante un circuito denominado **codificador**, que convierte la información, como por ejemplo un número decimal, en algún tipo de código, como el código binario o BCD.

Codificador decimal –BCD.

Este tipo de codificador posee diez entradas, una para cada dígito decimal, y cuatro salidas que corresponden al código BCD de la entrada activa. Este es un codificador básico de 10 líneas a 4 líneas.



Entrada decimal	Código BCD			
	A_3	A_2	A_1	A_0
0 (E_0)	0	0	0	0
1 (E_1)	0	0	0	1
2 (E_2)	0	0	1	0
3 (E_3)	0	0	1	1
4 (E_4)	0	1	0	0
5 (E_5)	0	1	0	1
6 (E_6)	0	1	1	0
7 (E_7)	0	1	1	1
8 (E_8)	1	0	0	0
9 (E_9)	1	0	0	1

$$A_0 = E_1 + E_3 + E_5 + E_7 + E_9$$
$$A_1 = E_2 + E_3 + E_6 + E_7$$
$$A_2 = E_4 + E_5 + E_6 + E_7$$
$$A_3 = E_8 + E_9$$

El funcionamiento básico del circuito es el siguiente: cuando aparece un nivel alto '1' en una de las líneas de entrada correspondientes a los dígitos decimales, se generan los niveles apropiados en las cuatro líneas BCD de salida. Por ejemplo, si la línea de entrada 9 está a nivel alto (suponiendo que todas las demás estén a nivel bajo), esta condición producirá el código BCD 1001, es decir, A_0 y A_3 a nivel alto y A_1 y A_2 a nivel bajo.

Codificador con prioridad decimal – BCD.

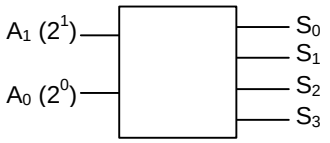
Realiza la misma función codificadora y además puede emplearse para detectar prioridad. La función de prioridad significa que cuando hay varias entradas decimales activas el codificador producirá la salida BCD correspondiente al dígito decimal de entrada de más alto orden que se encuentre activo, e ignorará cualquier otra entrada activa. Por ejemplo, si se encuentran activas las entradas 6 y 3, la salida BCD será 0110 (que representa al número decimal 6).

5.3.2. Función de decodificación.

Se implementa mediante un circuito denominado **decodificador** que convierte la información codificada, como puede ser un número binario, en otra información no codificada, como lo es un número decimal.

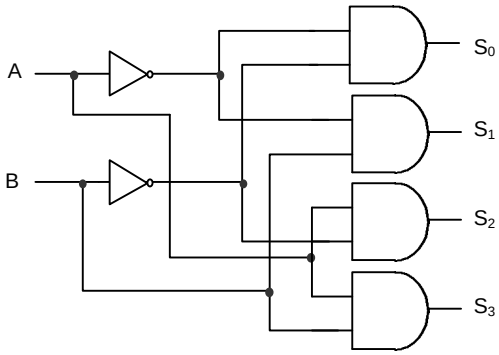
Decodificador binario- decimal.

Genera una salida para cada combinación de entradas. Para poder decodificar todas las posibles combinaciones de las entradas son necesarias 2^n salidas, siendo *n* el número de entradas. Por ejemplo un decodificador de 2 bits, denominado comúnmente *decodificador de 2 líneas a 4 líneas*, tendrá 4 salidas.



$$S_0 = \overline{A_1} \cdot \overline{A_0}$$
$$S_1 = \overline{A_1} \cdot A_0$$
$$S_2 = A_1 \cdot \overline{A_0}$$
$$S_3 = A_1 \cdot A_0$$

Dígito binario		Salida decimal			
A_1	A_0	S_0	S_1	S_2	S_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



Decodificador BCD-decimal.

Convierte código BCD en uno de los diez posibles dígitos decimales. Frecuentemente se le denomina *decodificador de 4 líneas a 10 líneas*.

Código BCD				Salida decimal
A_3	A_2	A_1	A_0	
0	0	0	0	0 (S_0)
0	0	0	1	1 (S_1)
0	0	1	0	2 (S_2)
0	0	1	1	3 (S_3)
0	1	0	0	4 (S_4)
0	1	0	1	5 (S_5)
0	1	1	0	6 (S_6)
0	1	1	1	7 (S_7)
1	0	0	0	8 (S_8)
1	0	0	1	9 (S_9)

Decodificador BCD-7 segmentos.

Este tipo de decodificador acepta código BCD en sus entradas y proporciona salidas capaces de excitar un display de 7 segmentos para indicar un dígito decimal. Por ejemplo para generar un 1, se excitan los segmentos *b* y *c*.

Nº	A_3	A_2	A_1	A_0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
10	1	0	1	0	X	X	X	X	X	X	X
11	1	0	1	1	X	X	X	X	X	X	X
12	1	1	0	0	X	X	X	X	X	X	X
13	1	1	0	1	X	X	X	X	X	X	X
14	1	1	1	0	X	X	X	X	X	X	X
15	1	1	1	1	X	X	X	X	X	X	X

A_1A_0 \ A_3A_2	00	01	11	10
00	1	0	X	1
01	0	1	X	1
11	1	1	X	X
10	1	1	X	X

$$a = A_1 + A_3 + A_0A_2 + \overline{A_0} \cdot \overline{A_2}$$

A_1A_0 \ A_3A_2	00	01	11	10
00	1	1	X	1
01	1	0	X	1
11	1	1	X	X
10	1	0	X	X

$$b = \overline{A_2} + A_1A_0 + \overline{A_1} \cdot \overline{A_0}$$

A_1A_0 \ A_3A_2	00	01	11	10
00	1	1	X	1
01	1	1	X	1
11	1	1	X	X
10	0	1	X	X

$$c = A_2 + A_0 + \overline{A_1}$$

A_1A_0 \ A_3A_2	00	01	11	10
00	1	0	X	1
01	0	1	X	0
11	1	0	X	X
10	1	1	X	X

$$d = A_1\overline{A_0} + A_1\overline{A_2} + \overline{A_0} \cdot \overline{A_2} + A_0\overline{A_1}A_2$$

A_1A_0 \ A_3A_2	00	01	11	10
00	1	0	X	1
01	0	0	X	0
11	0	0	X	X
10	1	1	X	X

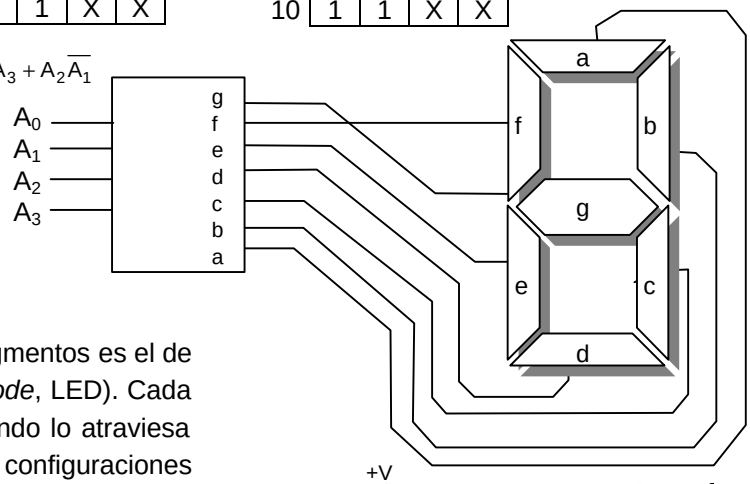
$$e = \overline{A_2} \cdot \overline{A_0} + A_1\overline{A_0}$$

A_1A_0 \ A_3A_2	00	01	11	10
00	1	1	X	1
01	0	1	X	1
11	0	0	X	X
10	0	1	X	X

$$f = A_2\overline{A_0} + \overline{A_1} \cdot \overline{A_0} + A_3 + A_2\overline{A_1}$$

A_1A_0 \ A_3A_2	00	01	11	10
00	0	1	X	1
01	0	1	X	1
11	1	0	X	X
10	1	1	X	X

$$g = A_3 + A_1\overline{A_0} + A_1\overline{A_2} + \overline{A_1}A_2$$



Display de LEDs

Un tipo común de display de 7 segmentos es el de diodos emisores de luz (*light-emitting diode*, LED). Cada segmento es un LED que emite luz cuando lo atraviesa una corriente eléctrica. Hay dos configuraciones posibles:

- Ánodo común. El segmento se encenderá cuando se le aplique un nivel bajo '0'.
- Cátodo común. El segmento se encenderá cuando se le aplique un nivel alto '1'.

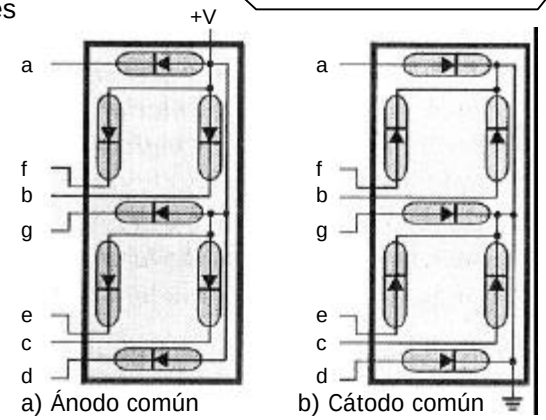
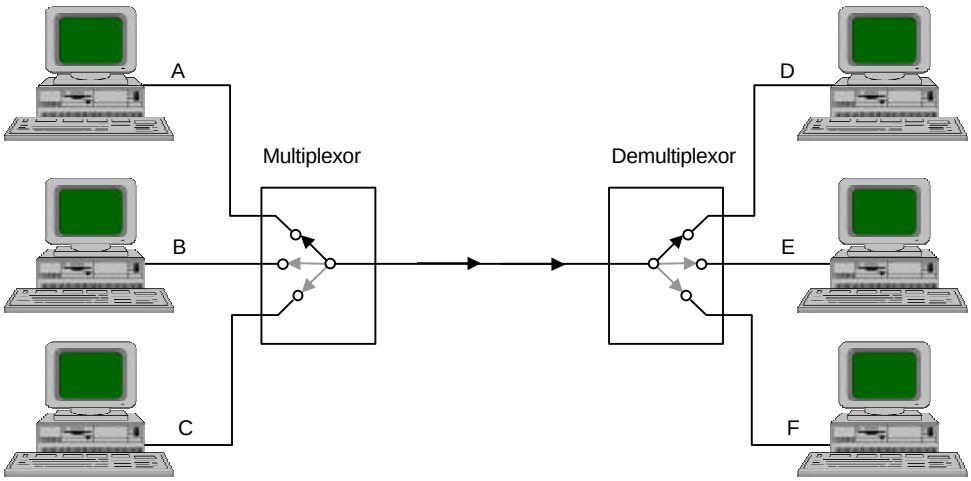


Figura 2-7 . Display de 7 segmentos.

5.4.- Función de selección de datos.

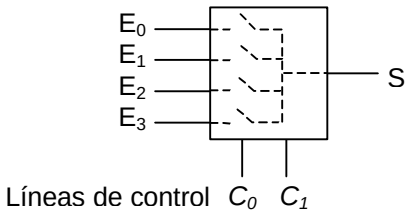
Existen dos tipos de circuitos dedicados a la selección de datos: el multiplexor y el demultiplexor. Se emplean cuando se tiene que transmitir datos de distintas fuentes a través de una línea hasta una localización distante, y deben redistribuirse en destino.



5.4.1.- Multiplexor.

Un **multiplexor** es un circuito que transmite los datos digitales procedentes de varias líneas de entrada a una única línea de salida según una secuencia específica. Funcionalmente, se puede representar mediante una operación de conmutación electrónica, que secuencialmente conecta cada una de las líneas de entrada a la línea de salida.

Son sistemas digitales de varias entradas y una salida, en los que la salida es igual a una de las entradas dependiendo de la combinación de las líneas de control. Para un multiplexor de *n* líneas de control C_i , el número de entradas será 2^n.

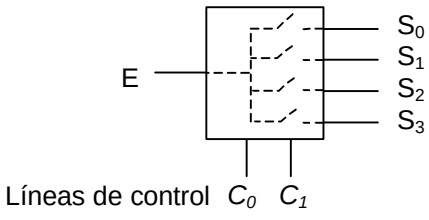


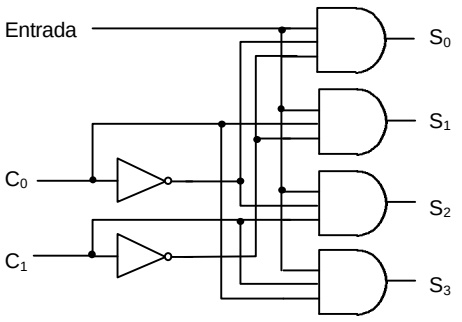
C_1	C_0	S
0	0	E_0
0	1	E_1
1	0	E_2
1	1	E_3

$$S = \overline{C_0}\,\overline{C_1}E_0 + C_0\overline{C_1}E_1 + \overline{C_0}\,C_1E_2 + C_0\,C_1E_3$$

5.4.2.- Demultiplexor.

Un **demultiplexor** es un circuito que transmite los datos digitales procedentes de una línea de entrada a varias líneas de salida según una determinada secuencia. Esencialmente, es un multiplexor invertido.





$$S_0 = E\,\overline{C_0}\,\overline{C_1}$$
$$S_1 = E\,C_0\,\overline{C_1}$$
$$S_2 = E\,\overline{C_0}\,C_1$$
$$S_3 = E\,C_0\,C_1$$