1. FUNCIONES LOGICAS

1.1. Función Lógica:

Dadas **n** variables lógicas (X_1, X_2, …., X_n) cuyos valores pueden tomar **0** o **1**, es posible definir una función lógica **f(X_1, X_2, …., X_n)** que tomará un valor **0** o **1** según los valores que tomen cada una de las variables y las operaciones que se realicen.

1.2. Tablas de la Verdad:

Dado que una variable lógica solo puede tomar dos valores (**0** y **1**), las funciones lógicas también están acotadas a estos valores. Es posible representar a una función lógica por medio de lo que se denomina *tabla de verdad*. Una tabla de verdad contiene todas las combinaciones posibles de las variables lógicas, y el valor que la función toma para cada caso. Por lo tanto la tabla contiene toda la información relativa a la función lógica.
Por ejemplo, la tabla de verdad de una función lógica de dos variables **Y=f(A,B)** se representa de la siguiente manera:

ENTRADAS		SALIDA
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

2. COMPUERTAS LOGICAS

Las compuertas lógicas son circuitos electrónicos que operan con niveles definidos de tensión, materializando las funciones lógicas. Estos circuitos se construyen en diferentes tecnologías, utilizando resistencias eléctricas, diodos y transistores. También es posible materializar a las funciones lógicas mediante contactos de relés (lógica de contactos).

2.1. Compuerta lógica OR:

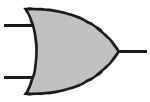
La compuerta **OR** define la operación *suma lógica*, la que se representa por el signo **+**. Por ejemplo, la función suma lógica de dos variables se escribe:

Y = A + B

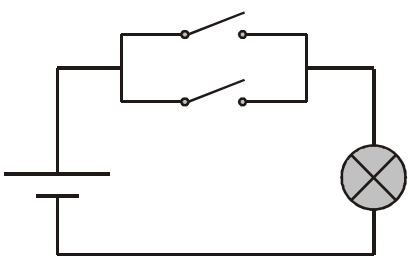
En esta compuerta, la salida siempre será un **1** si cualquier variable de entrada tiene el valor **1**. La tabla de verdad de la compuerta **OR** es entonces:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

El símbolo para representar a una compuerta **OR** de dos entradas es:



Y el símil de una compuerta **OR** representándola por lógica de contactos es;



La lámpara se encenderá si al menos un interruptor (o ambos a la vez) se encuentra cerrado (es decir, en **1**).

2.2. Compuerta AND:
Esta compuerta define la operación *producto lógico*, y se representa por un punto (a veces directamente se omite el punto). Por ejemplo, la función producto lógico entre dos variables se puede representar con cualquiera de las siguientes formas:

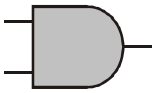
Y = A . B

Y = A B

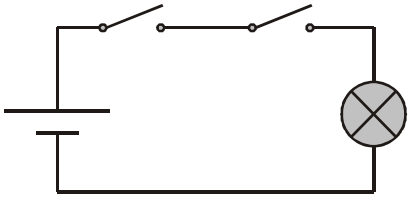
En la compuerta **AND**, la salida será **1** sólo si todas sus entradas se encuentran en **1**. La tabla de verdad de esta compuerta es:

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

El símbolo para representar a una compuerta **AND** de dos entradas es:



Y el símil de una compuerta **AND** representándola por lógica de contactos es:



La lámpara se encenderá sólo si los dos interruptores (ambos a la vez) se encuentren cerrados (es decir, en **1**).

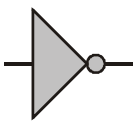
2.3. Compuerta lógica NOT:
La compuerta **NOT** define la *negación* o *inversión*. A diferencia de las compuertas anteriores que pueden tener dos o más entradas, la compuerta **NOT** sólo posee una entrada. Si la entrada se encuentra en **1** entonces la salida será **0**, y viceversa.
La operación **NOT** se representa mediante una raya horizontal sobre la variable, aunque también se puede representar mediante una comilla simple (este último muy poco utilizado). Es decir:

$Y = \overline{A}$
Y = A'

La tabla de verdad de esta compuerta es:

A	Y
0	1
1	0

El símbolo que representa a una compuerta **NOT** es:



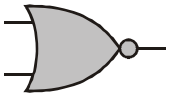
2.4. Compuerta lógica NOR:
La compuerta **NOR** es una combinación de una compuerta **OR** seguida por una **NOT**. Define la función:

$$Y = \overline{A + B}$$

Su tabla de verdad correspondiente es:

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

El símbolo que representa a una compuerta **NOR** es:



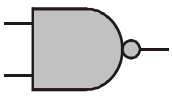
2.5. Compuerta lógica NAND:
Esta compuerta es una combinación de una compuerta **AND** seguida de una **NOT**. La función que define es:

$$Y = \overline{A.B}$$

La tabla de verdad de esta función es:

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

El símbolo que representa a una compuerta **NAND** es:



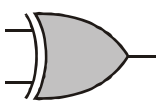
2.6. Compuerta lógica OR Exclusiva:
Esta compuerta define la siguiente función lógica:

$$Y = \overline{A}.B + A.\overline{B}$$

Su tabla de verdad es:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Es decir, para el caso de una **OR Exclusiva** de dos variables de entrada, la salida será **1** sólo si una sola de sus entradas está en **1**. En forma más general, si la compuerta tiene más de dos variables de entrada, la salida será **1** sólo si un número impar de sus entradas están en **1**.
El símbolo correspondiente a la compuerta **OR Exclusiva** es:



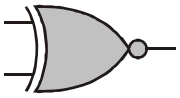
2.7. Compuerta lógica NOR Exclusiva:
Es una **OR Exclusiva** seguida de una **NOT**. La función lógica que define es la siguiente:

$$Y = \overline{\overline{A}.B + A.\overline{B}}$$

Y la tabla de verdad es:

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Es decir, la salida será **1** sólo si un número par de sus entradas están en **1**. El símbolo de la **NOR Exclusiva** es:



3. ALGEBRA DE BOOLE
El Algebra de Boole es una herramienta matemática compuesta por una serie de teoremas que son útiles para simplificar expresiones de variables lógicas.
Para el caso de variables lógicas binarias, el Algebra de Boole hace uso de tres operaciones fundamentales:

- La suma lógica (**OR**) **Y = A + B**
- El producto lógico (**AND**) **Y = A.B**
- La negación (**NOT**) **Y = $\overline{A}$**

3.1. Postulados y propiedades del Algebra de Boole:

A + 0 = A	**A.1 = A**
A + 1 = 1	**A.0 = 0**
A + A = A	**A.A = A**
A + $\overline{A}$ = 1	**A.$\overline{A}$ = 0**

$\overline{\overline{A}}$ = A

Estos nueve teoremas se refieren a una sola variable. Observar que, salvo en el último caso, hay una relación entre los teoremas de la columna de la izquierda con los de la columna de la derecha. Dada una ecuación en una columna, la ecuación correspondiente en la otra columna se puede escribir intercambiando **0** por **1**, e intercambiando (**+**) por (**.**). Los teoremas relacionados entre sí por este doble intercambio se denominan *duales*.

- **Leyes de conmutación**
  A + B = B + A
  A.B = B.A

- **Leyes de asociación**
  A + (B + C) = (A + B) + C = A + B + C
  A.(B.C) = (A.B).C = A.B.C

- **Leyes de distribución**
  A.(B + C) = A.B + A.C
  A + (B.C) = (A + B).(A + C)

- **Leyes de Morgan**
  $\overline{A + B}$ = $\overline{A}$.$\overline{B}$
  $\overline{A.B}$ = $\overline{A}$ + $\overline{B}$

- **Otras leyes**
  $A + (A.B) = A$
  $A.(A + B) = A$

Ejemplo: reducir la siguiente función $Y = A.C + B.C + \overline{A}.C + \overline{A}.\overline{B}$
Primero saco factor común **C** en el primer y tercer término:
$Y = C.(A + \overline{A}) + B.C + \overline{A}.\overline{B} = C.1 + B.C + \overline{A}.\overline{B} = C + B.C + \overline{A}.\overline{B}$
Haciendo lo mismo con el primer y segundo término:
$Y = C(1 + B) + \overline{A}.\overline{B} = C.1 + \overline{A}.\overline{B} = C + \overline{A}.\overline{B}$
Negando dos veces el último término y distribuyendo la primer negación (ley de Morgan):
$Y = C + \overline{\overline{A}.\overline{B}} = C + \overline{A + B}$
Entonces mientras que para sintetizar (construir) la función original se requerían cuatro compuertas **AND** de dos entradas y una **OR** de cuatro entradas, la implementación de la función una vez reducida requiere sólo de dos compuertas: una **NOR** de dos entradas y una **OR** de dos entradas (suponiendo en ambos casos que se dispone de las variables y sus complementos, ya que de lo contrario se deberían agregar dos compuertas **NOT** en cada caso).

4. Universalidad de las compuertas NOR y NAND
Las funciones lógicas en general se expresan combinando las operaciones básicas **OR**, **AND**, y **NOT**, aplicadas a las variables lógicas. Sin embargo es posible prescindir de la operación **OR** o de la operación **AND** y expresar una función lógica en términos de:

- La negación y la suma lógica (**NOT** y **OR**).
- La negación y el producto lógico (**NOT** y **AND**).

Esto es posible aprovechando las transformaciones producidas por la aplicación de las Leyes de Morgan que, recordando, para el caso de dos variables toman la forma:
$\overline{A + B} = \overline{A}.\overline{B}$
$\overline{A.B} = \overline{A} + \overline{B}$

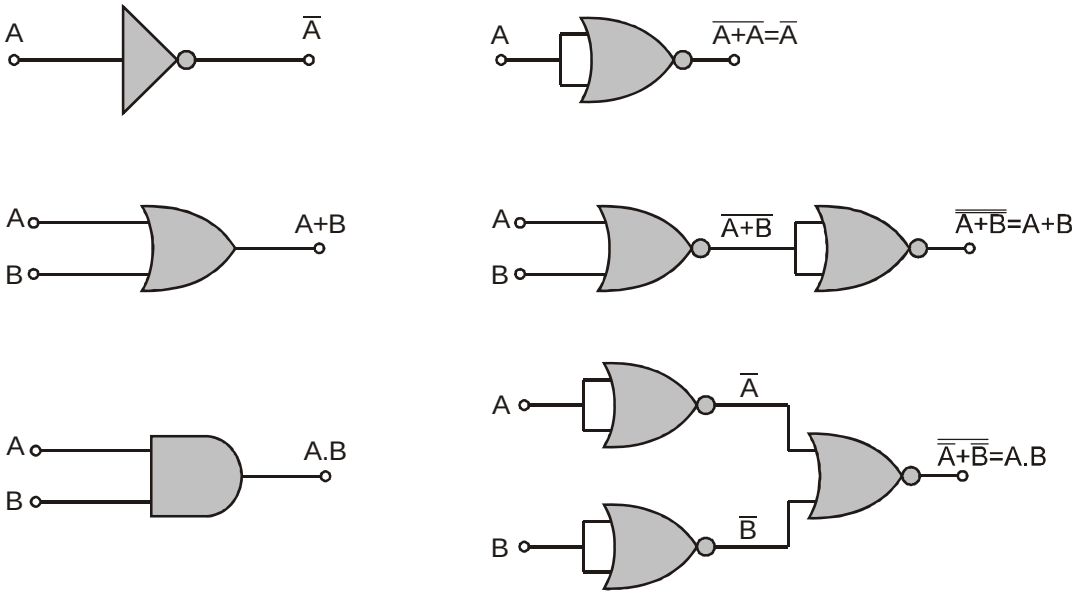
De esta manera es posible reemplazar las sumas lógicas por productos lógicos, y análogamente productos lógicos por sumas lógicas.

Ejemplo: realizar la suma lógica de dos variables utilizando las operaciones de producto lógico y de negación.
$Y = A + B$
Realizo una doble negación, con lo cual la función no cambia $Y = \overline{\overline{A + B}}$
Ahora distribuyo la primer negación (Ley de Morgan) $Y = \overline{\overline{A}.\overline{B}}$

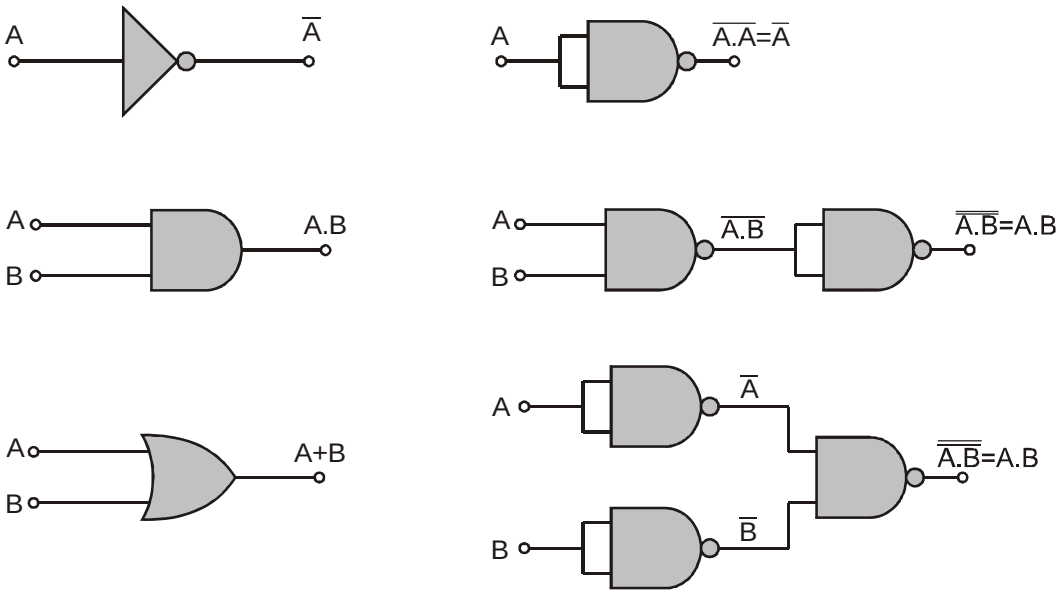
Ejemplo: realizar el producto lógico de dos variables utilizando las operaciones de suma lógica y negación.
$Y = A.B$
Realizo una doble negación, la función no se altera $Y = \overline{\overline{A.B}}$
Por último distribuyo la primer negación (Ley de Morgan) $Y = \overline{\overline{A} + \overline{B}}$

Dado que las compuertas **NOR** y **NAND** ya involucran los operadores **OR** y **NOT** (para la **NOR**) y **AND** y **NOT** (para la **NAND**), es posible entonces realizar las operaciones básicas **OR**, **AND** y **NOT** utilizando solamente compuertas **NAND** o compuertas **NOR**. Más aún, cualquier circuito combinacional se puede implementar utilizando solamente compuertas **NAND** o **NOR**.

4.1. Operaciones básicas con la compuerta NOR:
A la izquierda se representan las compuertas estándar, y a la derecha su equivalente utilizando compuertas **NOR**.



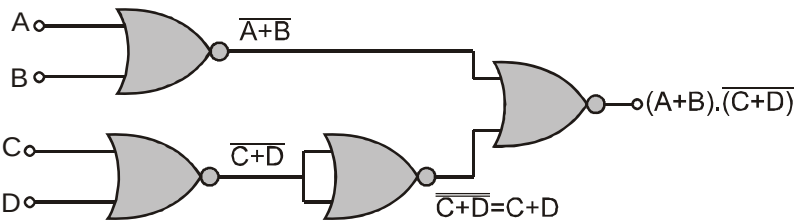
4.2. Operaciones básicas con la compuerta NAND:
A la izquierda se representan las compuertas estándar, y a la derecha su equivalente utilizando compuertas **NAND**.



Ejemplo: Implementar la función lógica $\mathbf{Y = (A + B).\overline{(C + D)}}$ con compuertas **NOR** de dos entradas.

Primero realizo una doble negación, la función no cambia $\mathbf{Y = \overline{\overline{(A + B).\overline{(C + D)}}}}$

Ahora distribuyo la primer negación $\mathbf{Y = \overline{\overline{(A + B)} + \overline{\overline{(C + D)}}}}$

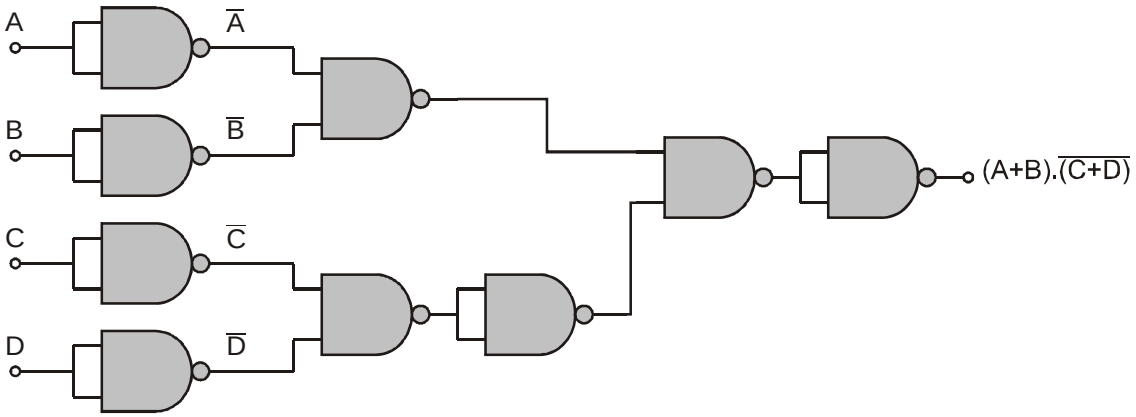


Ejemplo: Implementar la misma función lógica anterior, pero con compuertas **NAND** de dos entradas.

Primero realizo una doble negación, la función no cambia $\mathbf{Y = \overline{\overline{(A+B).\overline{(C+D)}}}}$

Ahora distribuyo la primer negación $\mathbf{Y = \overline{\overline{(A+B)} + \overline{\overline{(C+D)}}} = \overline{\overline{A}.\overline{B}} + \overline{\overline{C}.\overline{D}}}$

Distribuyo ahora la segunda negación $\mathbf{Y = \overline{\overline{A}.\overline{B}}.\overline{\overline{C}.\overline{D}} = \overline{\overline{A}.\overline{B}}.\overline{\overline{C}.\overline{D}}}$



En los ejemplos anteriores se puede observar que una de las alternativas (**NAND** o **NOR**) requiere menor cantidad de compuertas. En algunos casos el circuito mas reducido resultará con compuertas **NAND**, y en otros casos con compuertas **NOR**. En la práctica se trata de implementar un circuito con la menor cantidad de compuertas posibles, por dos razones importantes:

1. Económica, por requerir la menor cantidad de circuitos integrados, lo que redunda en una placa para contenerlos mas sencilla.
2. Menor tiempo de propagación de las señales eléctricas lógicas sobre el circuito.

5. Síntesis de Funciones Lógicas

La escritura explícita o algebraica de una función lógica resulta, en general, de mucha utilidad para realizar la síntesis de un sistema. Pero como generalmente los datos del problema se obtienen en forma de tabla de verdad, se hace necesario transformar la información contenida en la tabla de verdad por una expresión algebraica. Para la búsqueda de esta función lógica se recurre normalmente al método Shanon. Este método permite encontrar la función lógica mediante dos variantes: método de la suma de productos, y método del producto de sumas.

Consideremos el siguiente ejemplo: se tienen tres sensores que detectan la presencia de llama en el interior de una caldera. Asociemos a cada sensor una variable lógica $\mathbf{S_n}$ que valdrá **1** si detecta llama, y **0** si no la detecta. Definamos también una variable de salida **Y** que actuará sobre la electroválvula de inyección de los quemadores de la caldera, cerrándola (**0**) si por lo menos dos de los sensores detectan falta de llama.

Para este caso **Y** es una función lógica de tres variables $\mathbf{S_1}$, $\mathbf{S_2}$ y $\mathbf{S_3}$ que valdrá **0** o **1** de acuerdo a la siguiente tabla de verdad:

S_1	S_2	S_3	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

5.1. Desarrollo por suma de productos

Observando la tabla de verdad, la función **Y** tomará valor **1** en los casos en que:

$S_1 = 0 \quad S_2 = 1 \quad S_3 = 1$
$S_1 = 1 \quad S_2 = 0 \quad S_3 = 1$
$S_1 = 1 \quad S_2 = 1 \quad S_3 = 0$
$S_1 = 1 \quad S_2 = 1 \quad S_3 = 1$

Por lo tanto, la función lógica la podemos escribir de la siguiente manera:

$$\mathbf{Y = \overline{S_1}.S_2.S_3 + S_1.\overline{S_2}.S_3 + S_1.S_2.\overline{S_3} + S_1.S_2.S_3}$$

Es decir, la función lógica se obtiene observando en la tabla de verdad las filas que hacen **1** la salida. En cada una de estas filas se hace el producto lógico de las variables, y luego se hace la suma lógica de estos productos. Si la variable de entrada en cuestión toma un valor **0**, se la debe negar; si por el contrario toma el valor **1** se la deja sin negar.

5.2. Desarrollo por producto de sumas

La función lógica también puede obtenerse por producto de sumas, que es el caso dual al anterior. Primero se identifica en la tabla de verdad las filas que hacen **0** a la salida, y se hace la suma lógica de cada fila, teniendo en cuenta que si la variable vale **0** dicha variable no se niega, y si vale **1** se niega. Por último se hace el producto de cada término anterior.

Es decir, para el ejemplo, la salida **Y** valdrá **0** en los siguientes casos:

$S_1 = 0 \quad S_2 = 0 \quad S_3 = 0$
$S_1 = 0 \quad S_2 = 0 \quad S_3 = 1$
$S_1 = 0 \quad S_2 = 1 \quad S_3 = 0$
$S_1 = 1 \quad S_2 = 0 \quad S_3 = 0$

Por lo que la función lógica toma la forma:

$$\mathbf{\overline{Y} = (S_1 + S_2 + S_3).(S_1 + S_2 + \overline{S_3}).(S_1 + \overline{S_2} + S_3).(\overline{S_1} + S_2 + S_3)}$$

Ambas funciones anteriores, aunque distintas, son equivalentes y representan la misma función lógica.

5.3. Simplificación de funciones

Las funciones lógicas anteriores poseen términos redundantes, por lo que es posible simplificarlas, con lo que el circuito final resultante tendrá la menor cantidad posible de compuertas.

Para la simplificación de funciones disponemos en general de dos métodos: las propiedades del Algebra de Boole, y las tablas de Karnaught-Veitch.

Por ejemplo, por Algebra de Boole, consideremos las siguientes propiedades:

$$\mathbf{A + A + A = A}$$

$$\mathbf{A.B + A.\overline{B} = A.(B + \overline{B}) = A}$$

Con estas propiedades, simplifiquemos la función obtenida por suma de productos:

$$\mathbf{Y = \overline{S_1}.S_2.S_3 + S_1.\overline{S_2}.S_3 + S_1.S_2.\overline{S_3} + S_1.S_2.S_3}$$

Si agrego términos redundantes, la función no se altera:

$$\mathbf{Y = \overline{S_1}.S_2.S_3 + S_1.\overline{S_2}.S_3 + S_1.S_2.\overline{S_3} + S_1.S_2.S_3 + S_1.S_2.S_3 + S_1.S_2.S_3}$$

Luego:

$$\mathbf{\overline{S_1}.S_2.S_3 + S_1.S_2.S_3 = S_2.S_3}$$

$$\mathbf{S_1.\overline{S_2}.S_3 + S_1.S_2.S_3 = S_1.S_3}$$

$$\mathbf{S_1.S_2.\overline{S_3} + S_1.S_2.S_3 = S_1.S_2}$$

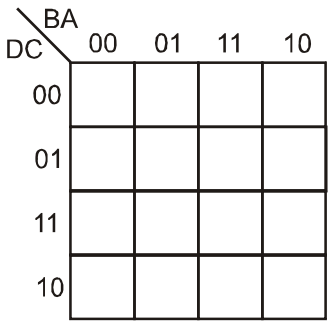
Con lo que la función simplificada queda:

$$\mathbf{Y = S_2.S_3 + S_1.S_3 + S_1.S_2}$$

5.4. Simplificación por mapas de Karnaught-Veitch

El mapa de Karnaught es un método gráfico que permite simplificar funciones de hasta seis variables. Para mayor cantidad de variables existen otros métodos (Quine-Mc Cluskey, Mc Boole) que sólo resultan útiles si se resuelven por procesos computacionales dado la complejidad de estos métodos.

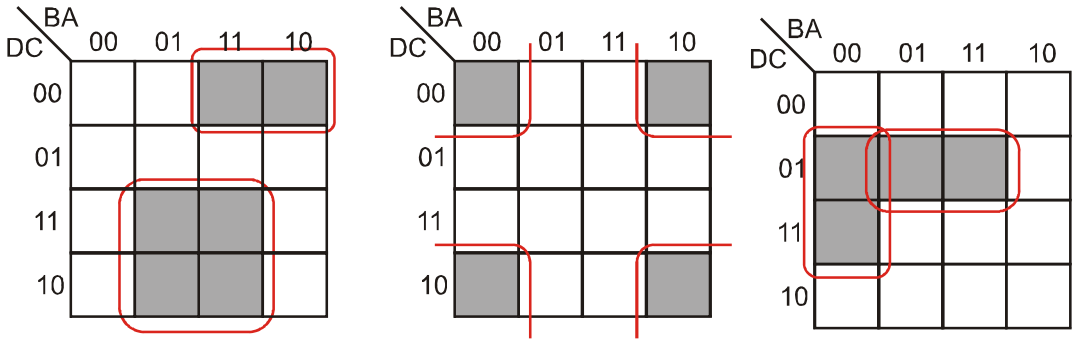
Un mapa de Karnaght es un cuadrilátero que posee tantas celdas como filas tenga la tabla de verdad. Por ejemplo, para 4 variables (digamos **DCBA**), una tabla de verdad posee 16 filas, por lo que el mapa de Karnaught toma la siguiente forma:



Cada una de las celdas se corresponde con una fila de la tabla de verdad, y contiene el valor de la función (**0** o **1**) para esa combinación de las variables. Es importante notar que las celdas son **adyacentes**, esto significa que al pasar de una celda a otra en sentido horizontal o vertical (no diagonal), sólo puede cambiar de **0** a **1** (o viceversa) una sola de las variables de entrada a la vez. También es importante notar que, aunque el mapa se dibuja como un rectángulo, en realidad se la debe imaginar como una esfera; esto significa que el borde izquierdo de la tabla está unido al borde derecho (son adyacentes), así como el borde superior está unido al borde inferior (adyacentes también).

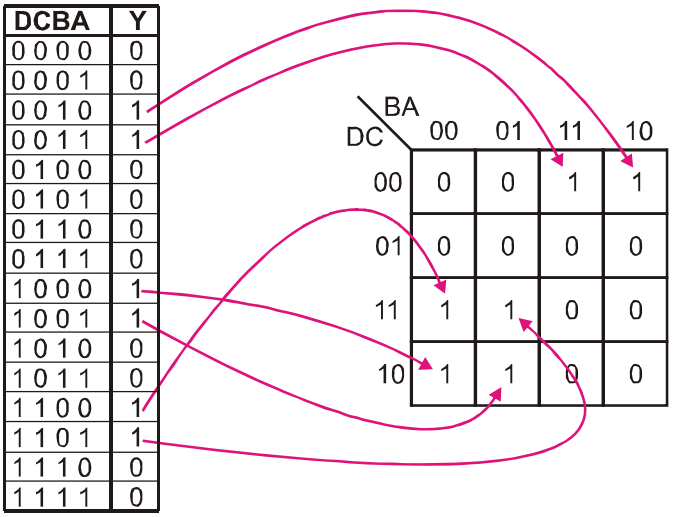
Para simplificar una función lógica, se debe comenzar por agrupar con lazos en el mapa de Karnaught las celdas cuyos valores hacen **1** a la salida. Esta agrupación no puede realizarse de cualquier forma, sino que hay reglas para ello:

- Cada lazo debe contener la mayor cantidad de **1** posibles, para lograr la mayor simplificación de la función lógica.
- Puede haber lazos superpuestos, es decir una celda puede pertenecer a dos o más lazos.
- No se pueden formar lazos entre parejas de **1** situados en diagonal.
- Deben agruparse con lazos todos los **1**, tratando de hacerlo con la menor cantidad de lazos posibles.
- Cada lazo debe agrupar 2^n celdas, con n=0, 1, 2, etc. Por ejemplo, no se pueden agrupar 7 celdas. A modo de ejemplo, algunos agrupamientos pueden ser:



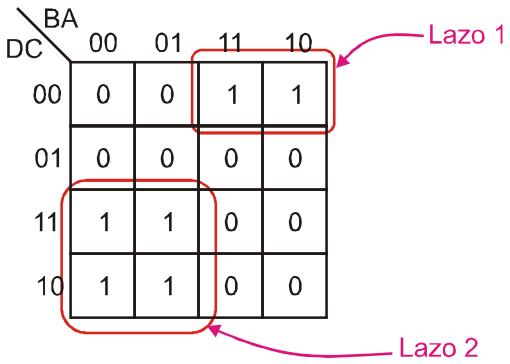
- Finalmente, se obtiene la función lógica por inspección de los lazos involucrados. La función tomará la forma de **suma de productos**:
  - La función tendrá tantos términos de sumas como lazos haya en el mapa.
  - Los productos de cada término quedan definidos por las variables involucradas en cada lazo. Sólo se deben tener en cuenta las variables que no cambien su valor (de **0** a **1** o viceversa) dentro del lazo, y en función de si permanecen en **0** o en **1** se las debe negar o no.

Veamos un ejemplo para mayor claridad. Dada la tabla de verdad, obtenemos el correspondiente mapa de Karnaught:



DCBA	Y
0 0 0 0	0
0 0 0 1	0
0 0 1 0	1
0 0 1 1	1
0 1 0 0	0
0 1 0 1	0
0 1 1 0	0
0 1 1 1	0
1 0 0 0	1
1 0 0 1	1
1 0 1 0	0
1 0 1 1	0
1 1 0 0	1
1 1 0 1	1
1 1 1 0	0
1 1 1 1	0

Paso siguiente, tratamos de enlazar la mayor cantidad de **1** con la menor cantidad de lazos, siempre respetando las reglas anteriores. Vemos que en este caso podemos hacer dos lazos:



Por lo tanto, la función lógica resultante estará compuesta por dos términos de suma lógica:

Y = Lazo 1 + Lazo 2

Analicemos el lazo 1, veamos que ocurre con las variables **DCBA**. Las variables **DC** permanecen en **0** en todo el lazo (no cambian de valor), por lo tanto estas variables quedan en la función lógica. Además, como toman el valor **0**, deben ir negadas. En cuanto a las variables **BA**, vemos que la única que mantiene su valor es **B** (ya que **A** vale **1** en la celda de la izquierda, y **0** en la celda de la derecha). Por lo tanto, la única variable que no cambia y que por lo tanto tengo que considerar es **B**, y además, como vale **1**, no va negada. Entonces el término correspondiente al lazo 1 es:

Lazo 1 = $\overline{D}.\overline{C}.B$

Veamos ahora el lazo 2. En cuanto a las variables **DC**, sólo **D** permanece constante y en valor **1**, por lo que se la debe tener en cuenta y además no va negada. En cuanto a las variables **BA**, la única que permanece constante en el lazo es **B** y con valor **0**, por lo que esta es la variable que debe tenerse en cuenta y además debe ir negada. Según lo anterior, el lazo 2 quedaría:

Lazo 2 = $D.\overline{B}$

Por último, la función lógica correspondiente a la tabla de verdad resulta:

Y = Lazo 1 + Lazo 2

$Y = D.\overline{B} + \overline{D}.\overline{C}.B$