ALGEBRA BOOLEANA

Historia de George Boole y el Origen del Álgebra Booleana

George Boole (1815–1864) fue un matemático y lógico británico autodidacta. Nació en Lincoln, Inglaterra, en una familia de recursos modestos.

A pesar de no haber recibido educación universitaria formal, se convirtió en uno de los matemáticos más influyentes del siglo XIX.

¿Qué propuso Boole?

Antes de Boole, la lógica se basaba en razonamientos verbales (la lógica aristotélica). Boole introdujo una forma matemática de describir enunciados lógicos, usando símbolos algebraicos para representar ideas como:

1: Verdadero (True)
0: Falso (False)
+: **OR** lógico (disyunción)
• o concatenación: **AND** lógico (conjunción)
¬ o –: **NOT** lógico (negación)

Esta abstracción permitió manipular proposiciones lógicas como si fueran ecuaciones algebraicas, lo cual fue una revolución.

Ejemplo simple de idea de Boole:

'El sistema puede estar encendido (A) o apagado (B)'

En lógica booleana:

A + B (se activa si A o B son verdaderos)
A • B (se activa solo si A y B son verdaderos).

Aplicaciones posteriores

Aunque en vida Boole no pudo preverlo, su trabajo fue la base matemática de los circuitos digitales.

Un siglo después, su álgebra fue utilizada por Claude Shannon (1938) en su tesis de maestría, donde mostró que los principios del álgebra booleana podían ser aplicados a interruptores eléctricos:

Conmutadores abiertos/cerrados = 0/1.
Combinaciones de interruptores = lógica booleana.
Base de diseño de computadoras y electrónica digital.

Legado de George Boole

Fundador del campo de la lógica matemática moderna.
Base de la computación, electrónica digital y circuitos lógicos.

Lenguajes de programación y bases de datos (SQL usa lógica booleana en consultas con AND, OR, NOT)

En su honor, se llaman 'operaciones booleanas' a aquellas que solo toman valores 0 o 1

Aplicaciones Actuales del Álgebra Booleana

1. Electrónica Digital

El álgebra booleana es la base matemática del diseño de todos los circuitos digitales. Se utiliza para controlar el flujo de corriente en circuitos que solo tienen dos estados posibles: encendido (1) y apagado (0).

Principales aplicaciones:

Compuertas lógicas:

AND
OR
NOT
NAND
NOR
XOR
XNOR

Diseño de circuitos combinacionales:

Sumadores
Restadores
Comparadores
Codificadores
Decodificadore
Multiplexores
Demultiplexores

Diseño de circuitos secuenciales:

Flip-Flops
Registros
Contadores
Memorias

Sistemas digitales:

Microcontroladores
Microprocesadores
PLCs

2. Computación y Arquitectura de Computadoras

En las computadoras, todos los datos y operaciones se representan con bits (0s y 1s). Las operaciones booleanas son fundamentales para procesar información en el nivel más bajo del hardware.

Usos en la computadora:

Unidad lógica y aritmética (ALU):

Realiza operaciones AND, OR, NOT, XOR entre bits.
Toma de decisiones del CPU: control de instrucciones y flujos de datos.

Direcciones de memoria y registros:

Manipulación de bits por medio de operaciones booleanas.
Control de dispositivos periféricos.

3. Lógica en Programación

En programación, el álgebra booleana se aplica en estructuras de control y expresiones lógicas para tomar decisiones automáticas.

Lenguajes como C, Python, Java, JavaScript, VB.NET, usan:

Operadores booleanos: AND, OR, NOT (también &&, ||, !).
Condiciones: usadas en if, while, for, switch.
Evaluación de expresiones: combinar varias condiciones con lógica booleana.

4. Inteligencia Artificial y Bases de Datos

En bases de datos, las búsquedas usan operadores booleanos para filtrar resultados:

Ejemplo en SQL:

SELECT * FROM empleados WHERE edad > 30 AND departamento = 'Ventas'

Diferencias entre Álgebra Clásica y Álgebra Booleana

Característica	Álgebra Clásica (Numérica)	Álgebra Booleana
Valores posibles	Números reales o enteros (∞ valores)	Solo 2 valores: 0 (falso) y 1 (verdadero)
Operaciones principales	Suma, resta, multiplicación, división	AND, OR, NOT (Y, O, NO)
Símbolos de operaciones	+(OR), · (AND), ¬ o ' (NOT)	+, −, ×, ÷
Propósito	Resolver problemas aritméticos o algebraicos	Representar lógica o decisiones
Sistema de valores	Continuo o discreto, numérico	Binario y lógico
Resultados esperados	Cualquier número real	Solo 0 o 1
Aplicaciones típicas	Finanzas, física, ingeniería, ecuaciones	Computación, lógica digital, electrónica
Identidad aditiva	0 (porque a + 0 = a)	0 (a + 0 = a, mismo efecto)
Identidad multiplicativa	1 (porque a × 1 = a)	1 (a · 1 = a, igual en lógica)
Complemento	No siempre aplicable (no existe 'no 5')	Siempre: ¬A (complemento lógico de A)

Ejemplos comparativos

Álgebra clásica: Si A = 2 y B = 3 A + B = 5 A × B = 6

Álgebra booleana: Si A = 1 y B = 0 A + B = 1 (1 OR 0) A · B = 0 (1 AND 0) ¬A = 0 (NOT 1)"

Observaciones

El álgebra clásica trabaja con cantidades.
El álgebra booleana trabaja con condiciones o estados lógicos
En álgebra booleana, no se usan fracciones, raíces ni potencias, todo se reduce a lógica binaria.
Álgebra booleana es la base para decidir, no calcular.

Variables booleanas y su interpretación lógica (1 = verdadero, 0 = falso)

¿Qué es una variable booleana?

Una variable booleana es una letra o símbolo que puede tomar solo dos valores posibles:
1, que representa verdadero (true)
0, que representa falso (false)

Estas variables se utilizan para modelar condiciones o decisiones que solo pueden ser verdaderas o falsas. Son la base del razonamiento lógico y del diseño de circuitos digitales.

Ejemplos de variables booleanas en la vida diaria

Condición	Variable	Valor lógico
¿Está encendida la luz?	L	1 (Sí) o 0 (No)
¿La puerta está cerrada?	D	1 (Sí) o 0 (No)
¿El número es mayor que 10?	A	1 (Sí) o 0 (No)
¿Hay conexión a internet?	C	1 (Conectado) o 0 (Desconectado)

¿Cómo se usan estas variables?

Las variables booleanas permiten representar expresiones como:

A AND B: Ambas condiciones deben ser verdaderas.
A OR B : Al menos una condición debe ser verdadera.
NOT A : El valor contrario de A."

Ejemplo sencillo :

Supón : A = 1 (La alarma está activada) B = 0 (La ventana está cerrada)

Entonces :
A AND B = 1 AND 0 = 0 → No hay peligro, porque aunque la alarma está activa, no hay intrusión.
A OR B = 1 OR 0 = 1 → Al menos una condición de seguridad está cumplida.
NOT A = 0 → La alarma no está activada.

Sistemas numéricos: conversión entre bases (binario, decimal, hexadecimal)

¿Qué es un sistema numérico?"

Un sistema numérico es una forma de representar cantidades usando una base. La base indica el número de símbolos diferentes que se usan.

Sistema	Base	Símbolos usados	Uso principal
Decimal	10	0–9	Sistema cotidiano
Binario	2	0,1	Electrónica digital, computadoras
Hexadecimal	16	0–9 y A–F	Abreviación de binario, programación

Conversión: Decimal → Binario

Método: División sucesiva entre 2

Ejemplo:
Convertir 13 (decimal) a binario
13 ÷ 2 = 6, residuo '1',
6 ÷ 2 = 3, residuo '0',
3 ÷ 2 = 1, residuo '1'
1 ÷ 2 = 0, residuo '1'
→ Binario = 1101

Conversión: Binario → Decimal

Método: Suma de potencias de 2

Ejemplo:
Convertir 1011 a decimal
$1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$
= 8 + 0 + 2 + 1 = 11

Conversión: Decimal → Hexadecimal

Método: división sucesiva entre 16

Ejemplo:
Convertir 254 a hexadecimal
254 ÷ 16 = 15, residuo 14 → 14 = E
15 ÷ 16 = 0, residuo 15 → 15 = F
→ Hexadecimal = FE

Conversión: Binario → Hexadecimal

Rápido y directo, agrupa bits en bloques de 4 (de derecha a izquierda)

Ejemplo:
Convertir 10110111 a hexadecimal
Agrupar: 1011 0111
1011 = B
0111 = 7
→ Hex = B7

Hexadecimal → Binario

Ejemplo:

Convertir A3 a binario
A = 1010
3 = 0011
→ Binario = 10100011

Operaciones Básicas Booleana

Definición y tabla de verdad de:

Operación OR (Suma lógica)

La operación **OR** (también llamada disyunción) devuelve 1 si al menos una de las entradas es 1.

Símbolos comunes:
A + B
A OR B
A v B (símbolo lógico)

Ejemplo práctico:
Se enciende la luz si el interruptor A o el B está activado.

Operación AND (Multiplicación lógica)

La operación **AND** (también llamada conjunción) devuelve 1 solo si ambas entradas son 1.

Símbolos comunes:
A · B
A AND B
A ^ B (símbolo lógico)

Ejemplo práctico:
Se abre la puerta si hay electricidad y se activa el botón.

Operación NOT (Negación lógica)

La operación NOT invierte el valor lógico de la entrada. Si A = 1, entonces ¬A = 0. Si A = 0, entonces ¬A = 1.

Símbolos comunes: ¬A A'
NOT A $\bar{A}$ (barra sobre la variable)
Ejemplo práctico: La alarma suena si no está cerrada la puerta.

Operaciones Compuestas

Las operaciones compuestas combinan varias operaciones básicas (AND, OR, NOT) para formar expresiones booleanas más complejas.

Ejemplo 1:

Expresión:
(A AND B) OR C → (A · B) + C

Ejemplo 2:

Expresión:
NOT (A OR B) → ¬(A + B) → Aplica Ley de DeMorgan

Leyes del Álgebra Booleana

Estas leyes permiten simplificar expresiones booleanas sin usar tablas de verdad.

Leyes Fundamentales

Ley de identidad
A + 0 = A ; A · 1 = A
0 y 1 no cambian el valor de A.

Ley del complemento
A + ¬A = 1 ; A · ¬A = 0
Una variable y su negación

Ley idempotente
A + A = A ; A · A = A
Repetir A no cambia el resultado

Ley nula (dominación)
A + 1 = 1 ; A · 0 = 0
Valores dominantes

Ley conmutativa
A + B = B + A ; A · B = B · A
El orden no altera el resultado

Ley asociativa
A + (B + C) = (A + B) + C
Agrupación no cambia resultado
A · (B · C) = (A · B) · C

Ley distributiva
$A \cdot (B + C) = A \cdot B + A \cdot C$
Distribuye como en álgebra común
$A + (B \cdot C) = (A + B) \cdot (A + C)$

Ley de involución
$\neg(\neg A) = A$
Negar dos veces = original

¿Qué es una Expresión Booleana?

Una expresión booleana es una combinación de variables (como A, B, C) y operaciones lógicas (AND, OR, NOT) que representan condiciones lógicas en circuitos o programas.

Ejemplos:

$A + B \rightarrow$ A OR B
$A \cdot \neg B \rightarrow$ A AND (NOT B)
$(A + B) \cdot C \rightarrow$ C AND (A OR B)

¿Por qué simplificar expresiones booleanas?

Objetivo:
Reducir el número de compuertas en un circuito, lo que mejora el diseño, ahorra componentes y reduce el costo.

Beneficios:
Menor complejidad en circuitos electrónicos.
Mejores tiempos de respuesta.
Reducción de errores en diseño lógico.

Técnicas de simplificación

1. Uso de Leyes Booleanas

Aplicar leyes como:

Identidad $A + 0 = A$
Dominación $A \cdot 0 = 0$
Idempotente $A + A = A$
Involución $\neg(\neg A) = A$
Ley de absorción $A + A \cdot B = A$
Ley distributiva $A \cdot (B + C) = A \cdot B + A \cdot C$
Ley de DeMorgan $\neg(A + B) = \neg A \cdot \neg B$

Mapas de Karnaugh (K-Maps)

¿Qué es un Mapa de Karnaugh?

Un Mapa de Karnaugh (K-Map) es una tabla organizada que representa todas las posibles combinaciones de una función booleana para visualizar patrones y simplificar expresiones de manera lógica. Sirve para identificar términos comunes y agruparlos visualmente, evitando errores al aplicar leyes booleanas manualmente.

Objetivo del K-Map:

Simplificar funciones lógicas al : Eliminar redundancias. Reducir el número de términos. Minimizar el número de compuertas en un circuito digital"

Tabla de verdad:

Antes de crear un K-Map, necesitas la tabla de verdad de una función, es decir, todas las combinaciones de entrada y su resultado lógico (0 o 1).

Número de variables y estructura del mapa:

Existen varias posibilidades dependiendo del número de variables a analizar. Para 2 variables se requiere 4 celdas (tabla de 2x2), para 3 variables se reuiere de 8 celdas (tabla de 2 x 4) y para 4 variables se requiere de 16 celdas (tabla 4 x 4).

Cada celda representa una combinación única de entradas, y se rellena con el valor de salida (1 o 0).

Reglas importantes:

1. El orden de las combinaciones sigue el Código Gray (solo cambia un bit a la vez).
2. Puedes agrupar celdas adyacentes de 1, 2, 4, 8… valores de 1, que formen potencias de 2. 3. Los grupos deben ser rectangulares (1×2, 2×2, etc.) y lo más grandes posible.
4. El mapa es circular: los extremos se tocan (puedes agrupar esquinas opuestas).
5. Cada celda con valor 1 debe estar en al menos un grupo.

Ejemplo 1:

Mapa de Karnaugh de 2 variables:

Paso 1:
Observe la tabla de verdad F(A,B) la cual contiene los diferentes estados de acuerdo a sus entradas 'A' y 'B', y su salida 'F'.

Paso 2:
Observe la estructura del mapa (2×2) que se forma a partir de la tabla de verdad F(A,B).

Paso 3:
Agrupamos las tres celdas con valor 1:

Un grupo de dos: (1,1) y (1,0) → A = 1.
Otro grupo de dos: (0,1) y (1,1) → B = 1.

Resultado
F(A, B) = A + B.

Mapa de Karnaugh de 3 variables:

Ejemplo 2:

Paso 1:
Estructura del mapa (2 filas por A, 4 columnas por B y C).

Paso 2:
Agrupar los unos.

Grupo 1:
Fila de abajo (4 unos) → A = 1.

Grupo 2:
Columna del medio (2 unos) → C = 1.

Resultado
F(A,B,C) = A + C.

Método visual para simplificar expresiones hasta 4 o 5 variables.

Ejemplo:

Una tabla de verdad con 2 variables A y B da 4 combinaciones posibles.
Se colocan en un mapa 2x2 para encontrar agrupaciones de unos (1s) y minimizar la función.

Ejemplo paso a paso

Expresión original:
F = A·B + A·¬B

Aplicamos ley distributiva inversa:

F = A · (B + ¬B)
Sabemos que B + ¬B = 1 (ley del complemento), por lo tanto:
F = A · 1 = A
por lo tanto:
Expresión simplificada: F = A

Leyes de DeMorgan

Muy importantes para simplificar expresiones negadas.

1ª Ley de DeMorgan
¬(A + B) = ¬A · ¬B

2ª Ley de DeMorgan
¬(A · B) = ¬A + ¬B

Ejemplo:

¬(A + B) se convierte en ¬A · ¬B
Útil para pasar de una compuerta NOR a una combinación AND–NOT

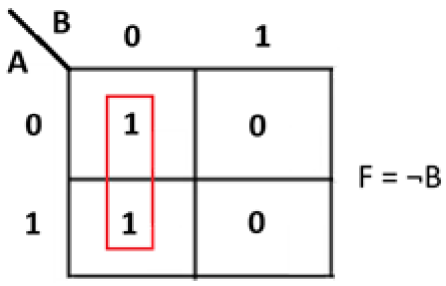
Ejemplo de simplificación:

Simplificar:
A · (A + B)
Aplica la ley de absorción:
A · (A + B) = A

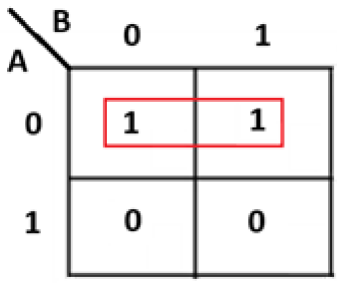
Ejemplos de Mapas de Karnaugh de 2 y 3 variables

A continuación se presentan varios ejemplos de la interpretación de simplificación de ecuaciones booleanas a partir del mapa de Karnaug.

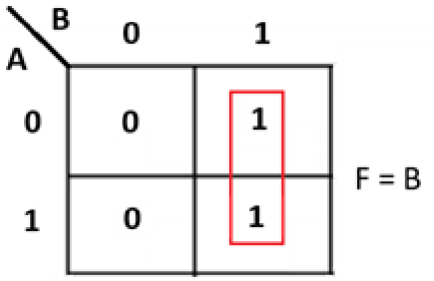
Ejemplos de 2 variables

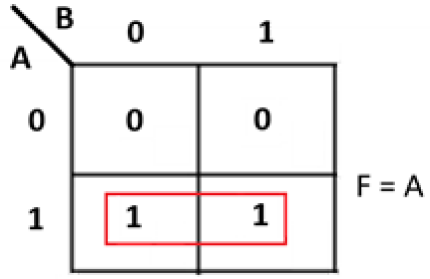


F = ¬B

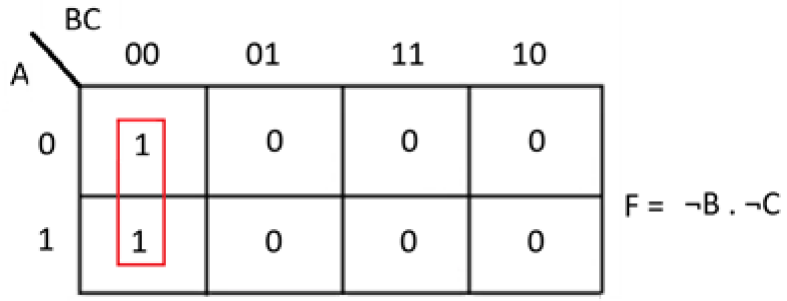


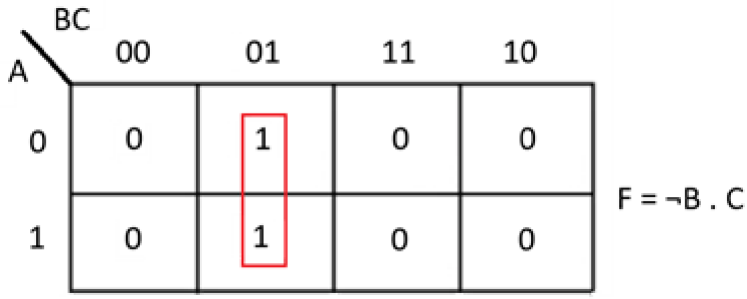
F = ¬A

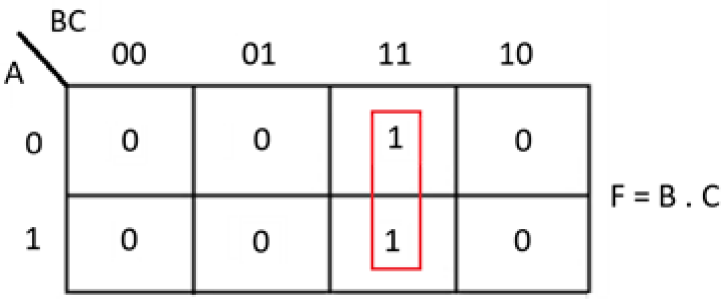


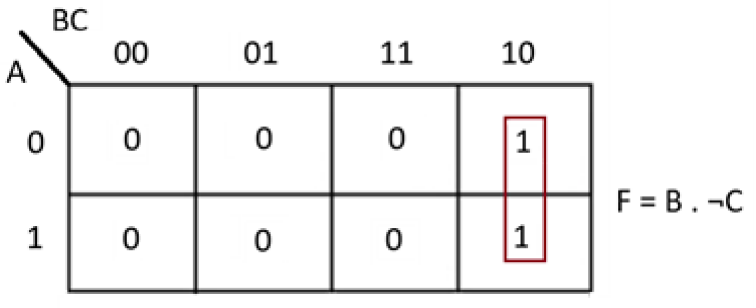


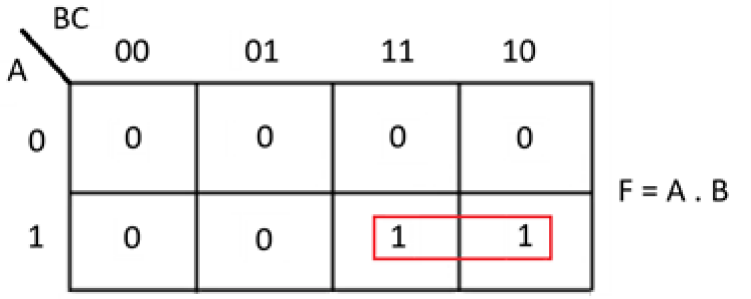
Ejemplos de 3 variables

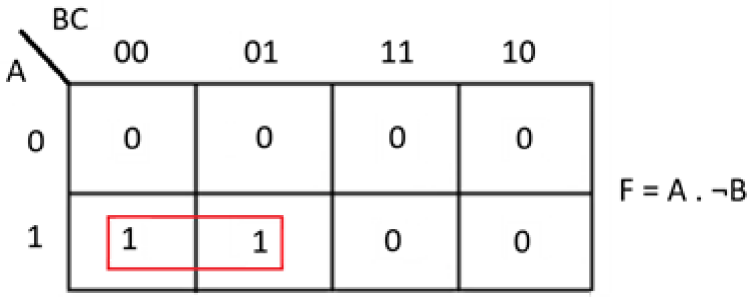


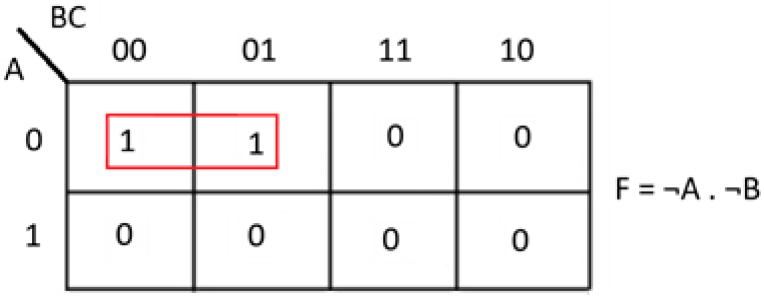


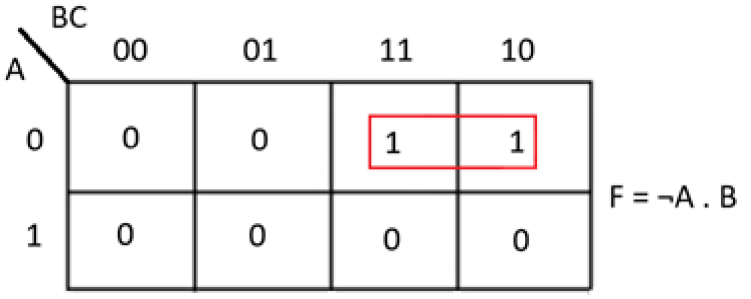


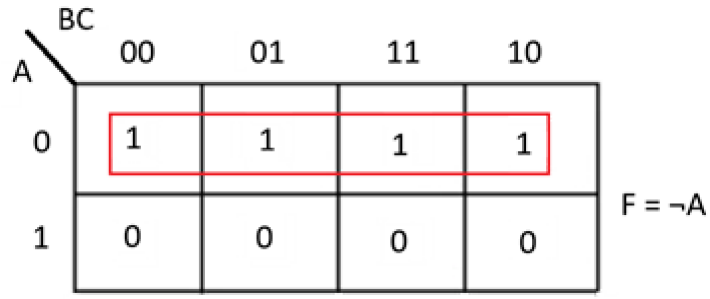

BC
A
00
01
11
10
0
0
0
0
1
1
0
0
0
1
F = B . ¬C

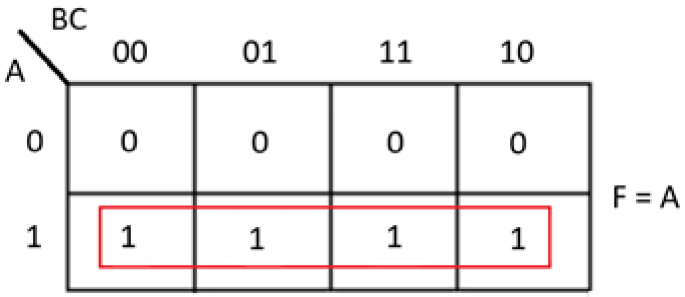

BC
A
00
01
11
10
0
0
0
0
0
1
0
0
1
1
F = A . B

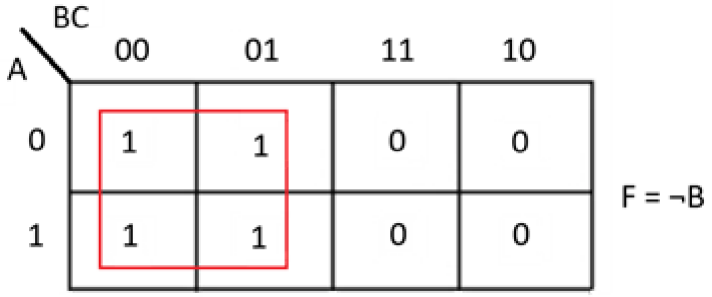

BC
A
00
01
11
10
0
0
0
0
0
1
1
1
0
0
F = A . ¬B

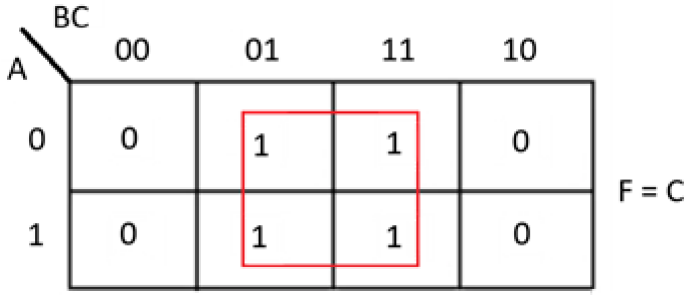

BC
A
00
01
11
10
0
1
1
0
0
1
0
0
0
0
F = ¬A . ¬B

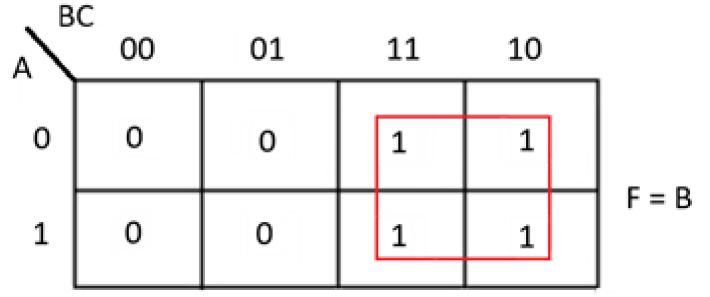

BC
A
00
01
11
10
0
0
0
1
1
1
0
0
0
0
F = ¬A . B

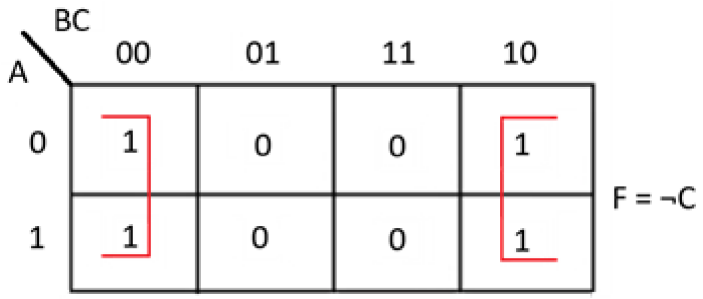

BC
A
00
01
11
10
0
1
1
1
1
1
1
0
0
0
0
F = ¬A


BC
A
00
01
11
10
0
1
0
0
0
0
1
1
1
1
F = A


BC
A
00
01
11
10
0
1
1
1
0
0
1
1
0
0
F = ¬B


BC
A
00
01
11
10
0
1
0
0
1
1
1
1
0
0
F = C


BC
A
00
01
11
10
0
1
0
0
0
0
1
1
1
1
F = B



Resumen de especificaciones del Mapa de Karnaugh

I. Estructura del mapa

Número de variables = tamaño del mapa

2 variables → 2×2
3 variables → 2×4
4 variables → 4×4

Cada eje usa **código Gray**, no binario simple.

Orden de las combinaciones (Gray code)

1 bit cambia entre casillas adyacentes.
Ejemplo 3 variables (A | BC): columnas BC = 00, 01, 11, 10.

Las celdas representan minitérminos

Cada casilla corresponde a una combinación única de A,B,C,(D).
Ejemplo:
m0 = 000, m1 = 001, m2 = 011, m3 = 010, etc. (en orden Gray).

El mapa es toroidal (se envuelve)

Los bordes opuestos son adyacentes.
La primera y última columna son vecinas.
La primera y última fila también lo son (en 4 variables).

II. Reglas para colocar los “1”

Solo se marcan **los minitérminos que hacen la función = 1**.
Si la función es una suma de productos, cada producto activa una o más celdas.
No se colocan 1s fuera del rango de variables.
Las casillas vacías son “0”, salvo que se use don’t care (X).

III. Reglas para agrupar

Lo que SÍ se debe hacer

Los grupos deben tener tamaño potencia de 2: 1, 2, 4, 8, 16.

Todas las celdas del grupo deben ser 1.
Cada grupo debe ser rectangular (horizontal, vertical o cuadrado).
Se permiten superposiciones si ayudan a cubrir todos los 1s.
Se permite que un grupo cruce los bordes del mapa (efecto envolvente).
Cada 1 debe estar cubierto por al menos un grupo.
Se priorizan los grupos más grandes posibles (primero 8, luego 4, luego 2, luego 1).

Lo que NO se debe hacer

No agrupar celdas diagonales (no son adyacentes en Gray code).
Ejemplo: m3 y m6 no son adyacentes (difieren en dos bits).

No formar grupos de tamaño no permitido (3, 5, etc.).
No usar celdas con “0” para completar un grupo.
No mezclar don’t care “X” con 0. Solo se pueden usar “X” como si fueran 1 **cuando ayudan a simplificar**, nunca para forzar una agrupación imposible.
No crear dos grupos idénticos (redundancia pura).
No crear grupos más pequeños si ya existe uno mayor que cubre las mismas celdas.
No perder cobertura: todos los 1s deben estar incluidos al menos una vez.
No asumir adyacencia entre celdas con diferencia de más de un bit.

IV. Prioridad en la formación de grupos (para implementación)

Primero los grupos más grandes posibles.
En 4 variables: 16 → 8 → 4 → 2 → 1
Si hay empate, preferir el grupo que cubra un 1 no cubierto aún.
Evitar redundancia: si un 1 ya está en un grupo mayor, no repetirlo salvo que sea necesario.
Horizontal antes que vertical puede ser una convención didáctica, pero no obligatoria — en simplificación pura ambos tienen igual valor.
Registrar qué celdas están cubiertas (matriz Usado) para evitar duplicados.

V. Reglas para traducir los grupos a expresión booleana

Cada grupo elimina las variables que **cambian dentro del grupo**.
Si A cambia (de 0 a 1), A desaparece del término.
Las variables **que permanecen constantes** determinan el término resultante.
Si una variable está siempre en 0 → se usa **¬variable**.
Si está siempre en 1 → se usa **variable**.
Cada grupo produce un término de la forma:

A.B+¬A.C+...A·B + ¬A·C + ...

La función simplificada final es la **suma (OR)** de todos los términos.

VI. Ejemplo de aplicación

Para la función F=(A.B)+(¬A.C)
m1, m3, m6, m7 →

Mapa de 3 variables:

A\BC	00	01	11	10
0	0	1	1	0
1	0	0	1	1

Grupos válidos:
m1–m3 → ¬A·C
m6–m7 → A·B
(m3–m7) vertical también posible, si no redundante.
Resultado mínimo:
F= A·B + ¬A·C + B·C

Teoría de la conversión con compuertas universales

Existe un programa complementario a este tema denominado 'Compuertas Universales' que nos permite dibujar el circuito lógico utilizando compuertas del tipo NOT, AND y OR, así como el circuito equivalente utilizando compuertas universales NAND o NOR. Dicho programa, ademas de dibujar los circuitos, nos permite también simular los estados lógicos en las diferentes entradas y salidas de las compuertas involucradas en los circuitos ya que con este simulador podemos seleccionar manualmente las entradas de las diferentas variables (hasta 4 variables). Este programa tiene varias facilidades para el usuario y una de ellas es la de poder seleccionar uno de varios ejemplos para experimentar los diversos aspectos operativos. Se puede seleccionar una expresión booleana de los ejemplos y ésta misma editarla para crear una nueva ecuación o bien introducir mediante el teclado una nueva expresión bulleana.