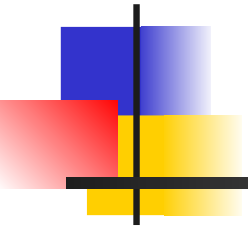
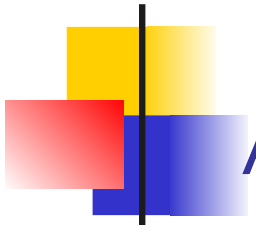


Arquitectura de Computadoras
Circuitos básicos

La mayor parte de los gráficos son de ***Principles of Computer Architecture*** *(1999) de Miles Murdocca y Vincent Heuring*

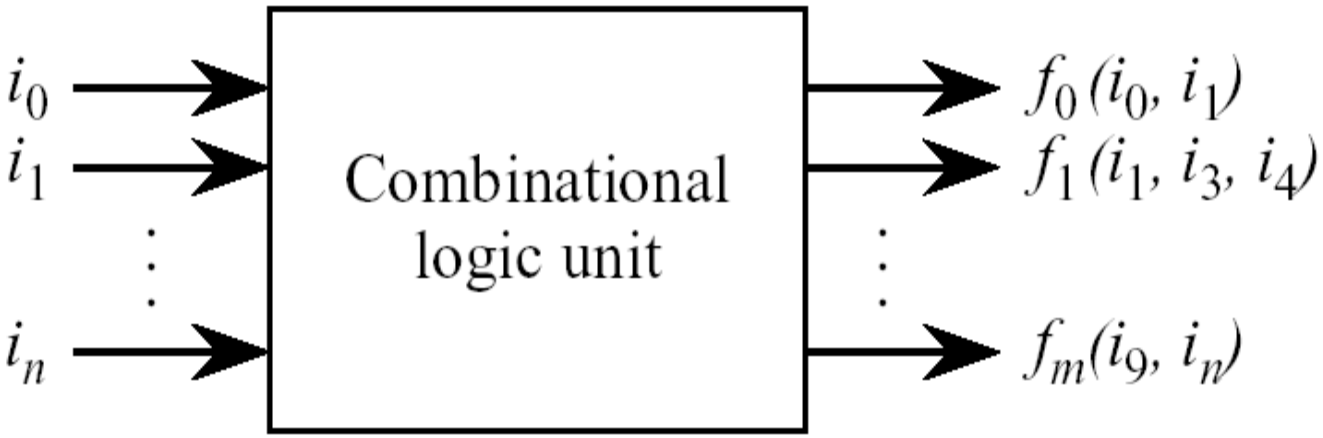


Analógico y digital

- Analógico
  - Valores continuos (un número infinito de estados distintos)
  - Se *mide*
- Digital
  - Valores discretos (un número finito de estados distintos)
  - Se *cuenta*
  - Habitualmente el número de valores distintos es 2 (binario)
  - Esos valores se identifican habitualmente con 0 y 1
  - Las computadoras son habitualmente binarias, aunque pueden darse circuitos de niveles múltiples (más de 2)
  - El problema es distinguir con seguridad entre más de dos niveles

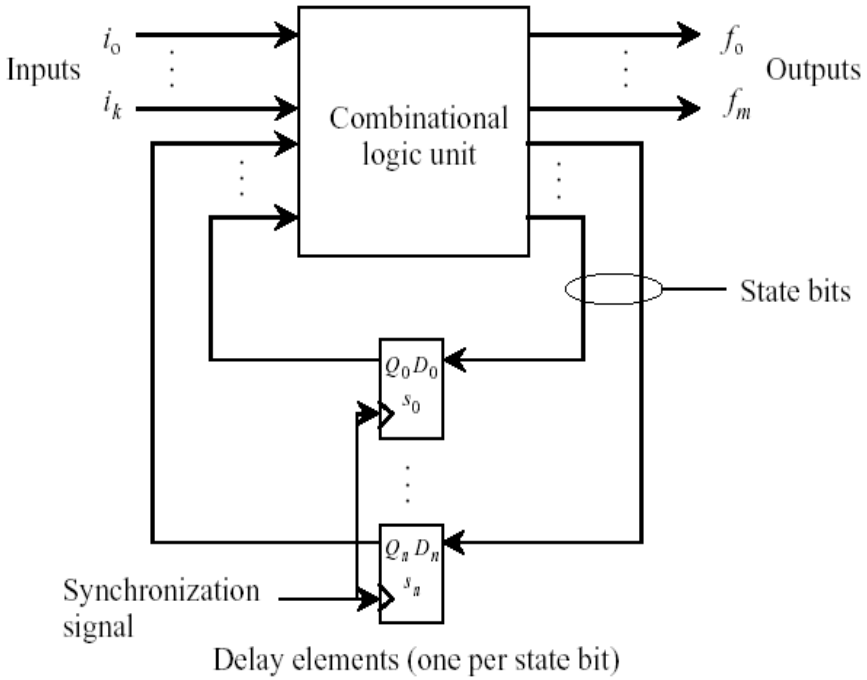
Circuitos digitales combinatorios

- Combinatorios
  - Los valores de las salidas dependen exclusivamente de los valores presentes en las entradas
  - Normalmente las entradas y salidas tienen dos estados binarios distintos: 1 y 0, alto y bajo, 5 voltios y 0 voltios (por ejemplo)
  - El comportamiento se puede expresar lógicamente por medio de un conjunto de expresiones booleanas



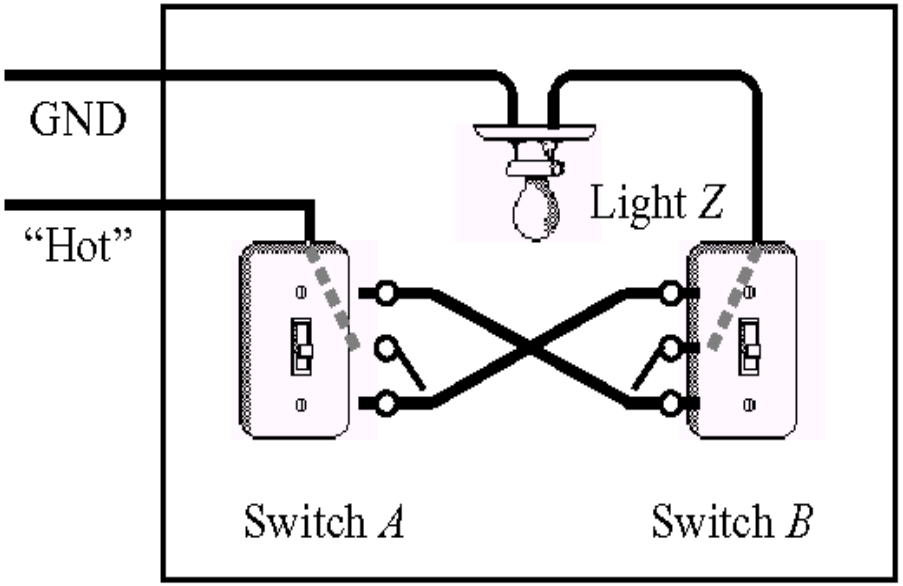
Circuitos digitales secuenciales

- Secuenciales
  - Los valores de las salidas dependen de los valores presentes y pasados de las entradas
  - El comportamiento debe expresarse por medio de una secuencia temporal de entradas y estados internos



Algebra de Boole: tablas de verdad

- Desarrolladas originalmente por George Boole (1854) como parte de su Algebra (una matemática del pensamiento)
- Perfeccionadas por Claude Shannon (Laboratorios Bell)
- Cada salida se calcula para todos los posibles valores de las entradas
- Ejemplo: una habitación con dos llaves de luz “combinadas”



Inputs		Output
A	*B*	*Z*
0	0	0
0	1	1
1	0	1
1	1	0

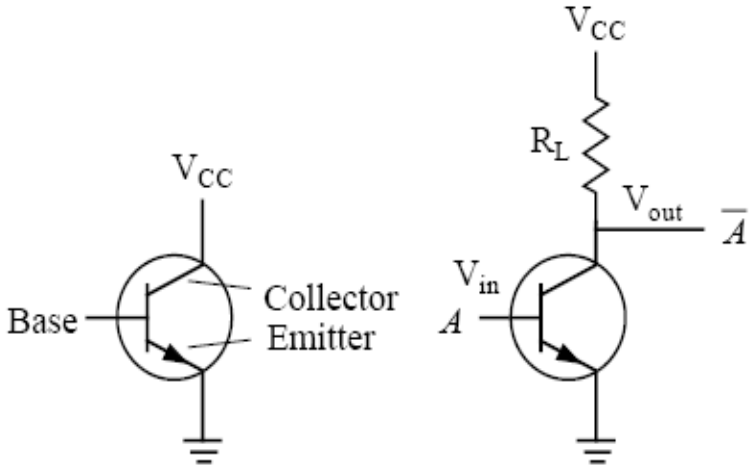
Tablas de verdad para funciones booleanas

Inputs		Outputs							
A	B	*False*	*AND*	$A\overline{B}$	A	$\overline{A}B$	B	*XOR*	*OR*
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

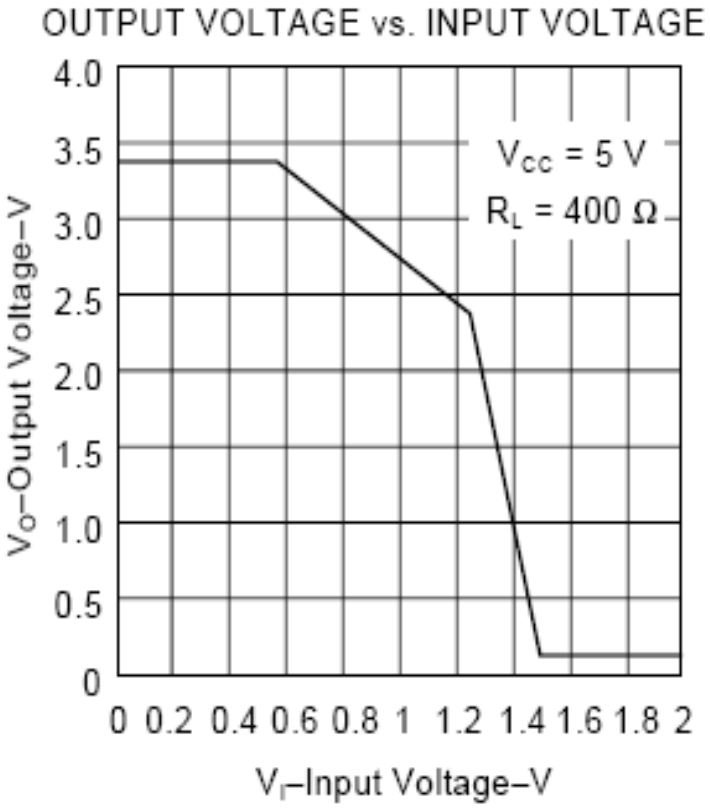
Inputs		Outputs							
A	B	*NOR*	*XNOR*	$\overline{B}$	$A+\overline{B}$	$\overline{A}$	$\overline{A}+B$	*NAND*	*True*
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

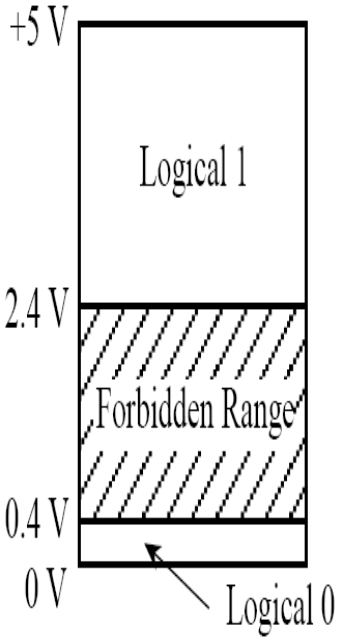
Compuertas lógicas: implementación (1/2)

- Un dispositivo físico que implementa una función del Algebra de Boole
- Se utilizan transistores (dispositivos analógicos)
- V_{CC}: tensión de colector
- GND: tierra
- R_L: Resistencia
- Las conexiones V_{CC} y GND a los bornes de una fuente de alimentación no se indican en las compuertas

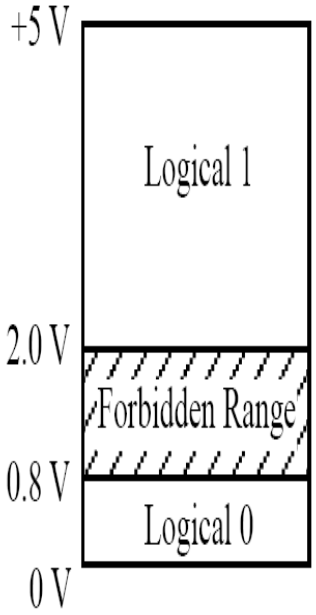


Compuertas lógicas: implementación (2/2)





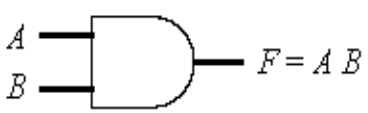
Entradas



Salidas

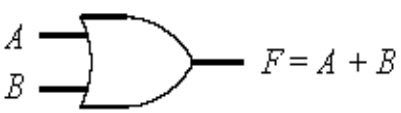
Representación de compuertas lógicas (1/2)

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1



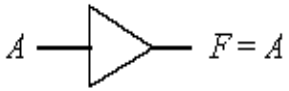
AND

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1



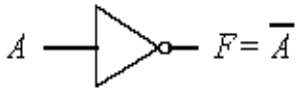
OR

A	F
0	0
1	1



Buffer

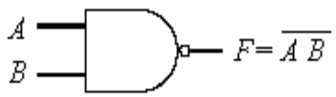
A	F
0	1
1	0



NOT (Inverter)

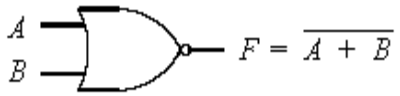
Representación de compuertas lógicas (2/2)

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0



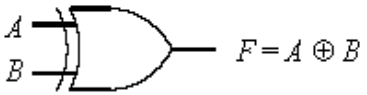
NAND

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0



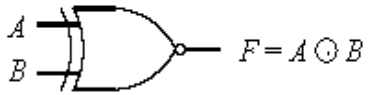
NOR

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0



Exclusive-OR (XOR)

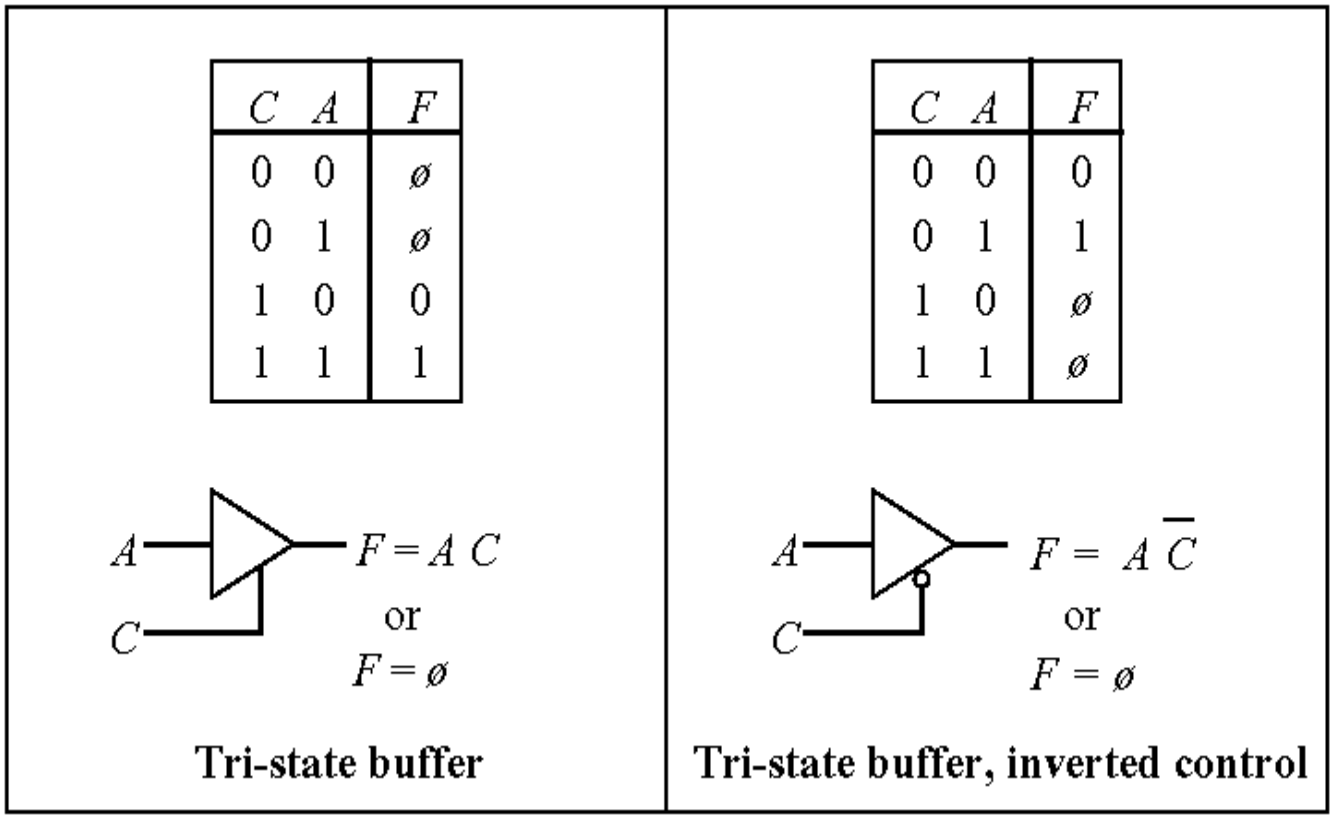
A	B	F
0	0	1
0	1	0
1	0	0
1	1	1



Exclusive-NOR (XNOR)

Buffer de 3 estados (three state buffer)

- La salida puede ser 0, 1 o estar "eléctricamente desconectada" (ni 0, ni 1: ausencia de señal)



Propiedades del Algebra de Boole

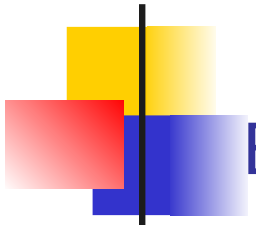
- Principio de dualidad: El dual de una función booleana se obtiene reemplazando ANDs por Ors, Ors por ANDs, 0s por 1s y 1s por 0s.

	Relationship	Dual	Property
Postulates	$A\,B = B\,A$	$A + B = B + A$	Commutative
	$A\,(B + C) = A\,B + A\,C$	$A + B\,C = (A + B)\,(A + C)$	Distributive
	$1\,A = A$	$0 + A = A$	Identity
	$A\,\overline{A} = 0$	$A + \overline{A} = 1$	Complement
Theorems	$0\,A = 0$	$1 + A = 1$	Zero and one theorems
	$A\,A = A$	$A + A = A$	Idempotence
	$A\,(B\,C) = (A\,B)\,C$	$A + (B + C) = (A + B) + C$	Associative
	$\overline{\overline{A}} = A$		Involution
	$\overline{A\,B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$	DeMorgan's Theorem
	$AB + \overline{A}C + BC = AB + \overline{A}C$	$(A+B)(\overline{A}+C)(B+C) = (A+B)(\overline{A}+C)$	Consensus Theorem
	$A\,(A + B) = A$	$A + A\,B = A$	Absorption Theorem

Aplicaciones del Teorema de DeMorgan

A B	$\overline{A\,B} = \overline{A} + \overline{B}$		$\overline{A + B} = \overline{A}\,\overline{B}$	
0 0	1	1	1	1
0 1	1	1	0	0
1 0	1	1	0	0
1 1	0	0	0	0

DeMorgan's theorem: $A + B = \overline{\overline{A + B}} = \overline{\overline{A}\,\overline{B}}$



Ejemplos de circuitos combinatorios

- Circuitos aritméticos
  - Semisumadores
  - Sumadores
  - Sumadores / restadores
  - Multiplicadores
  - Etc.
- Otros
  - Codificadores / decodificadores
  - Multiplexores / Demultiplexores
  - Desplazadores
  - Etc.

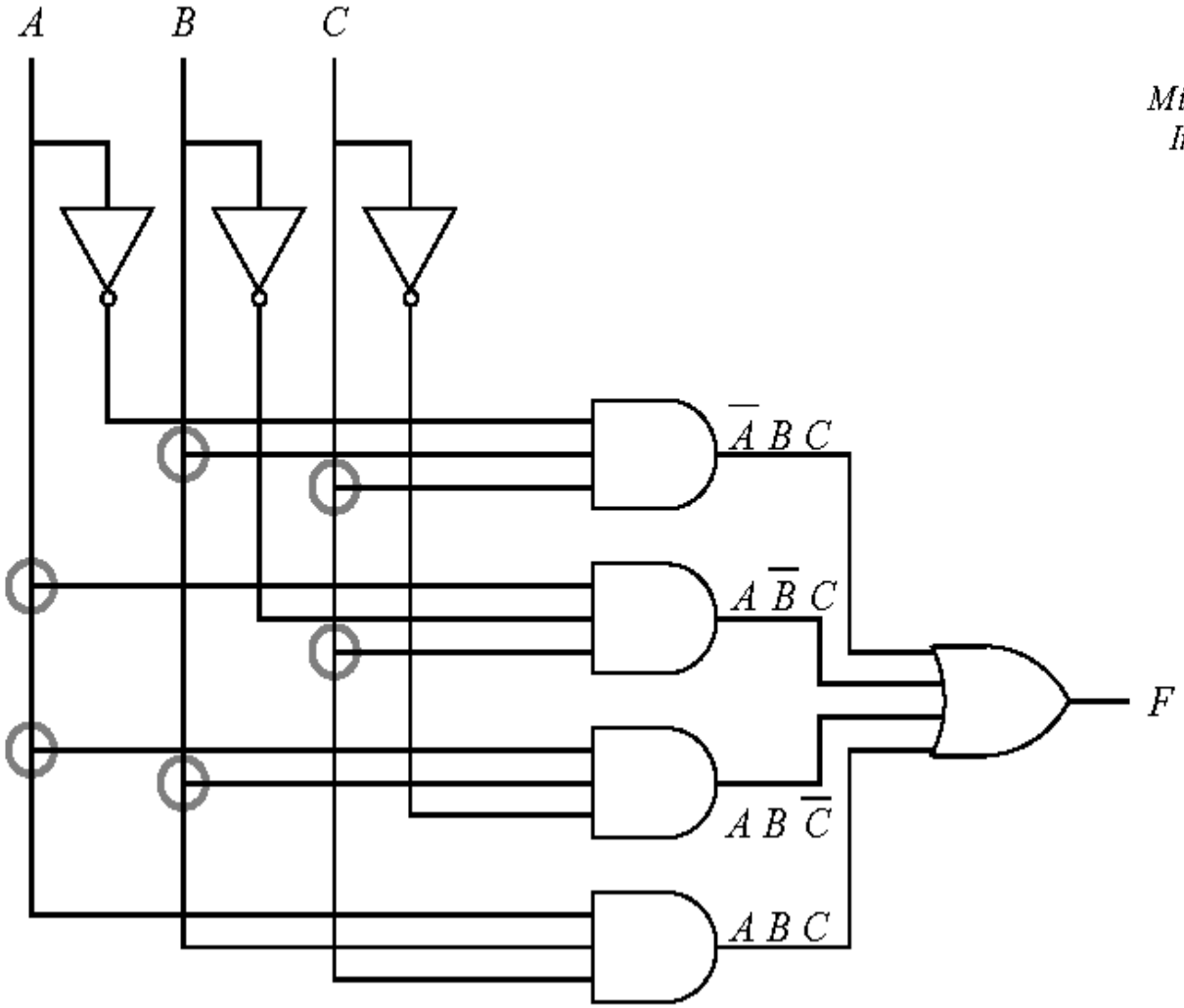
Representación de una función como suma de productos

- Cada combinación posible de las entradas se denomina ***minitérmino***
- Si hay ***n*** entradas habrá 2^n minitérminos (desde 0 hasta $2^n - 1$)
- Una salida cualquiera puede representarse como la ***SUMA DE PRODUCTOS*** de todos aquellos minitérminos para los cuales la función vale 1.
- Para la función MAYORIA DE 1s la SUMA DE PRODUCTOS es

$$M = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC = m3 + m5 + m6 + m7 = \sum(3, 5, 6, 7)$$

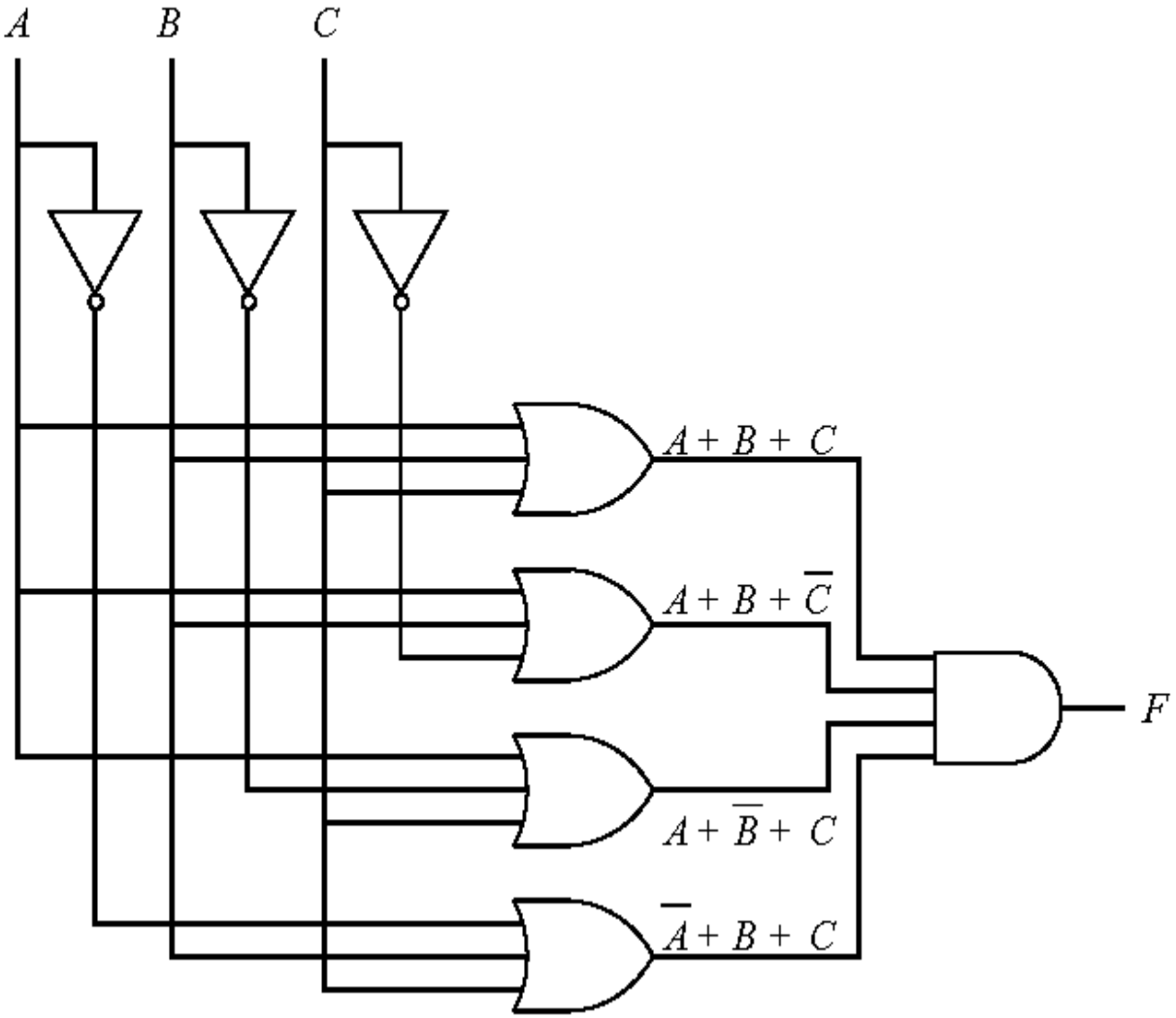
Minterm Index	*A*	*B*	*C*	*F*
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

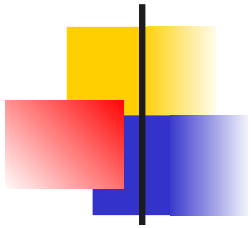
La función mayoría como suma de productos



Minterm Index	A	B	C	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

La función mayoría como producto de sumas



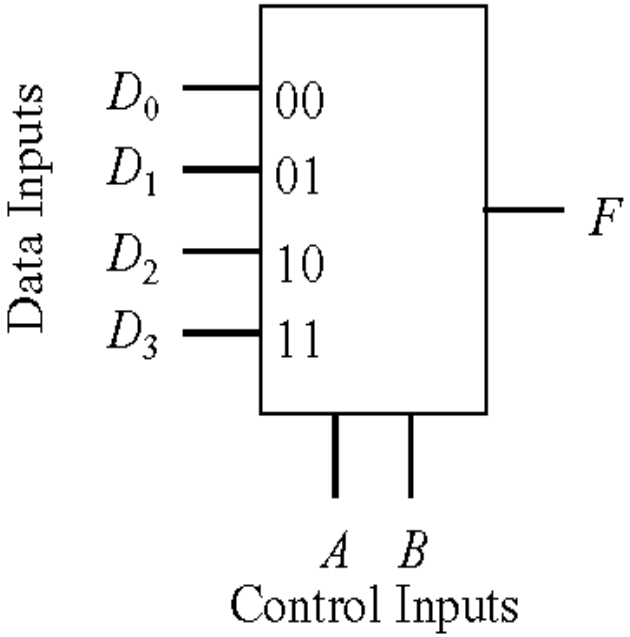


Componentes digitales

- Los circuitos digitales de alto nivel son normalmente implementados utilizando colecciones de compuertas lógicas conocidas como ***componentes*** en vez de utilizar compuertas individuales
- Los niveles de integración (número de compuertas lógicas / mm^2) en un circuito integrado son **aproximadamente** los siguientes:
  - SSI (Small Scale Integration): <100 (compuertas y biestables)
  - MSI (Medium Scale Integration): 100 – 1.000 (codificadores, sumadores, registros)
  - LSI (large Scale Integration): 1.000 – 100.000 (circuitos aritméticos complejos, memorias)
  - VLSI (Very Large Scale Integration): 100.000 – 1.000.000 (microprocesadores, memorias, microcontroladores)
  - ULSI (Ultra Large Scale Integration): + de 1.000.000 (microprocesadores avanzados)

Multiplexor

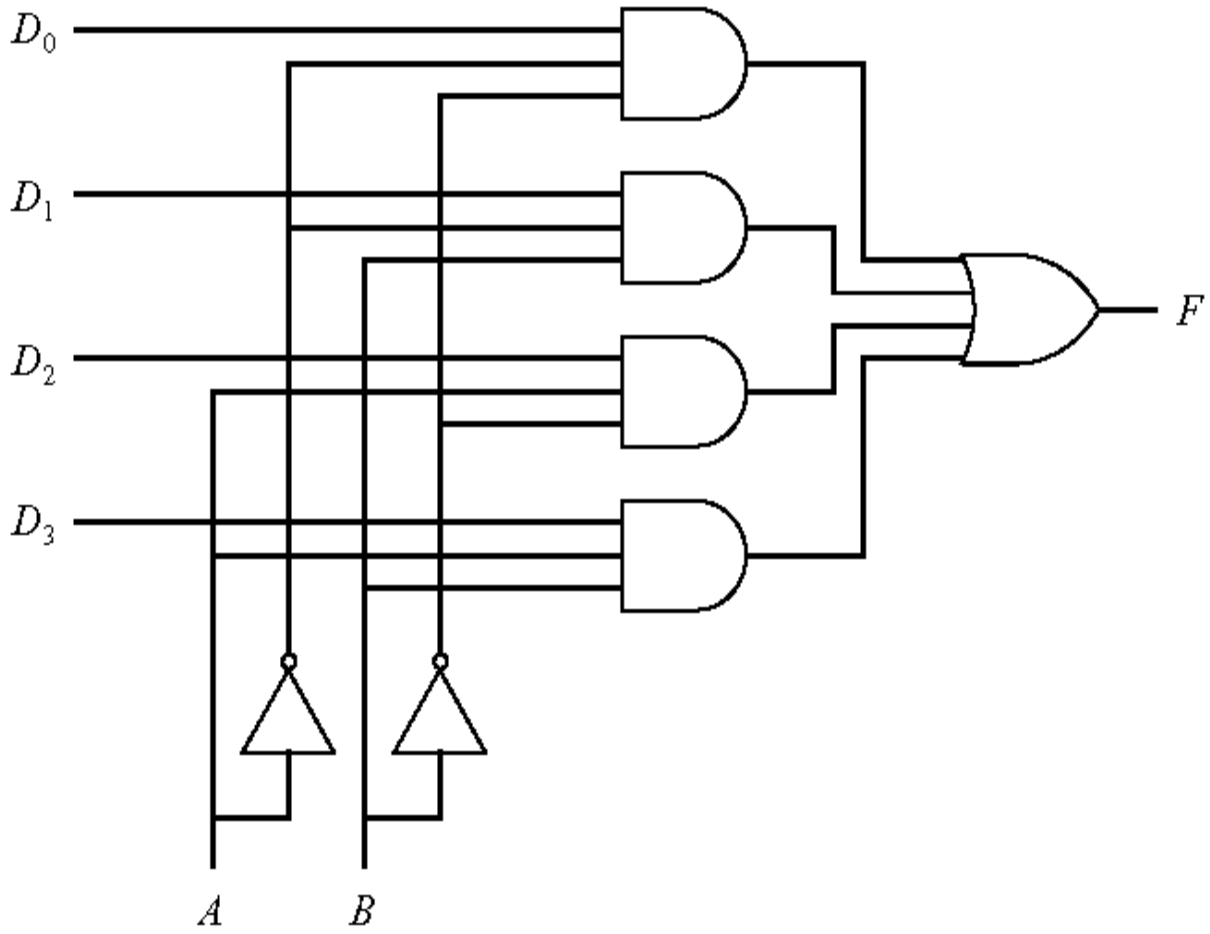
- Mediante ***n*** líneas de control selecciona la información presente en una de las 2^n líneas de entrada y la pone en la única salida



A	B	F
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

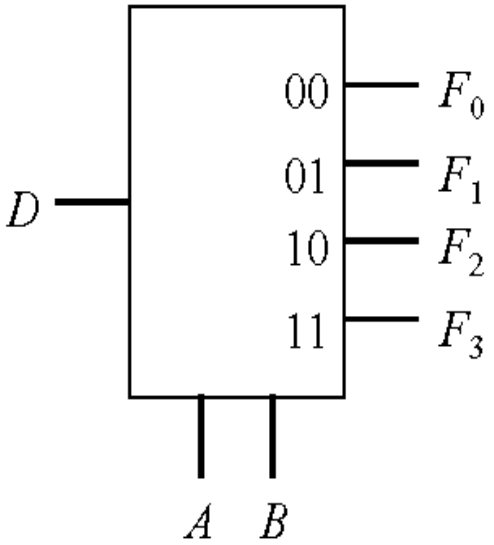
$$F = \overline{A}\,\overline{B}\,D_0 + \overline{A}\,B\,D_1 + A\,\overline{B}\,D_2 + A\,B\,D_3$$

Una implementación de un multiplexor



Demultiplexor

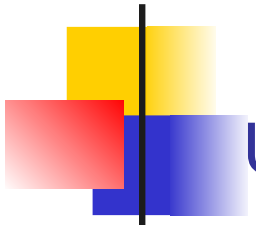
- Mediante ***n*** líneas de control selecciona una de las 2^n líneas de salida y pone en ella la información presente en la única entrada



D	A	B	F_0	F_1	F_2	F_3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

$$F_0 = D\overline{A}\,\overline{B} \qquad F_2 = DA\overline{B}$$

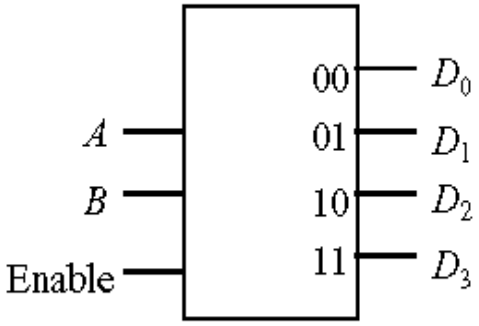
$$F_1 = D\overline{A}B \qquad F_3 = DAB$$



Una implementación de un demultiplexor

Decodificador

- Las ***n*** líneas de entrada se interpretan como un código que identifica una de las 2^n líneas de salida. La salida identificada se pone en 1. Suele usarse una señal de habilitación.

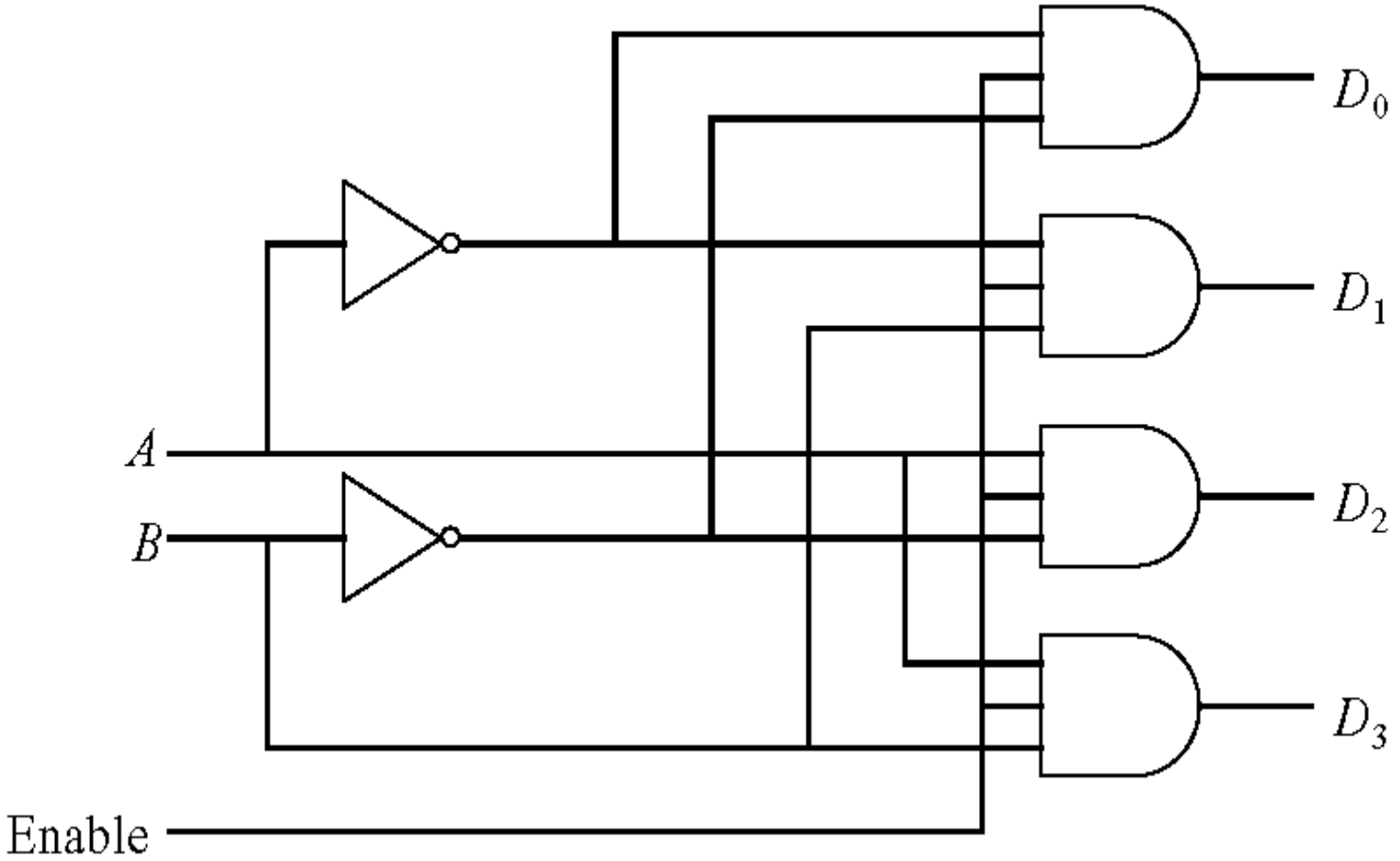


Enable = 1						
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

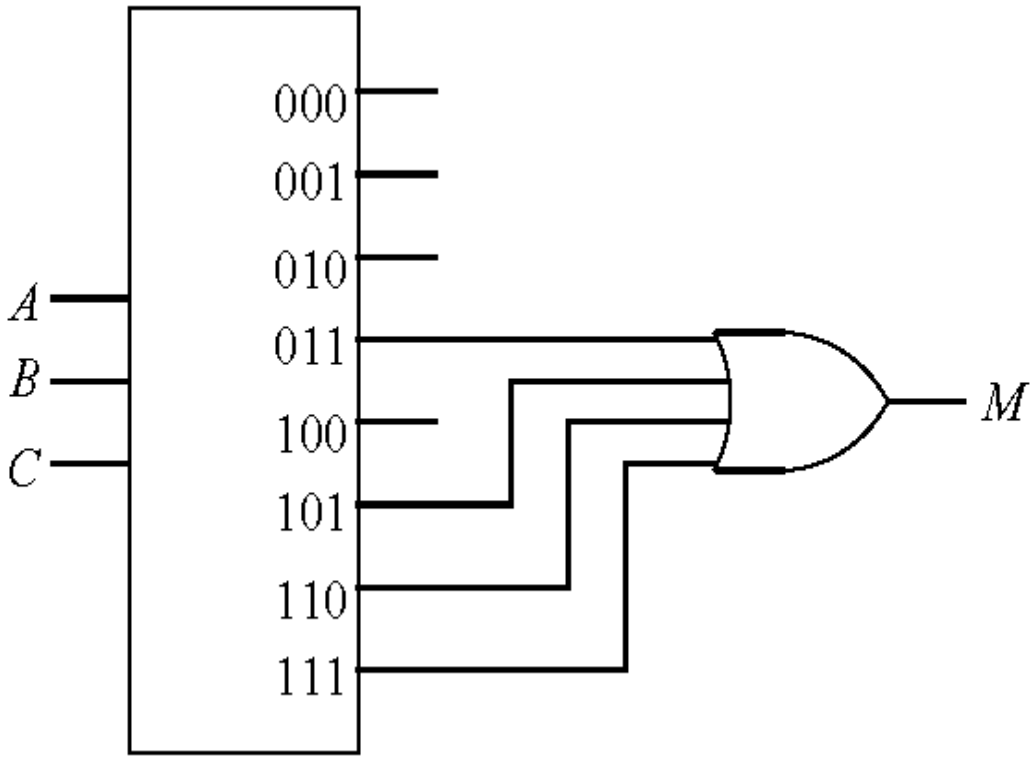
Enable = 0						
A	B	D_0	D_1	D_2	D_3
0	0	0	0	0	0
0	1	0	0	0	0
1	0	0	0	0	0
1	1	0	0	0	0

$$D_0 = \overline{A}\,\overline{B} \qquad D_1 = \overline{A}\,B \qquad D_2 = A\,\overline{B} \qquad D_3 = A\,B$$

Una implementación de un decodificador

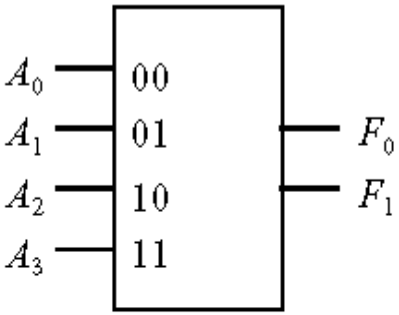


Uso de un decodificador para implementar la función mayoría



Codificador con prioridad

- Codifica, en ***n*** líneas de salida, el número de una de las 2^n líneas de entrada que está en 1
- Puede pensarse como el inverso de un decodificador
- Si no puede asegurarse que una y sólo una de las entradas estará en 1 se utilizan codificadores con prioridad
- En un codificador con prioridad, si varias entradas están en 1 el codificador codifica la línea de orden más alto (o más bajo)

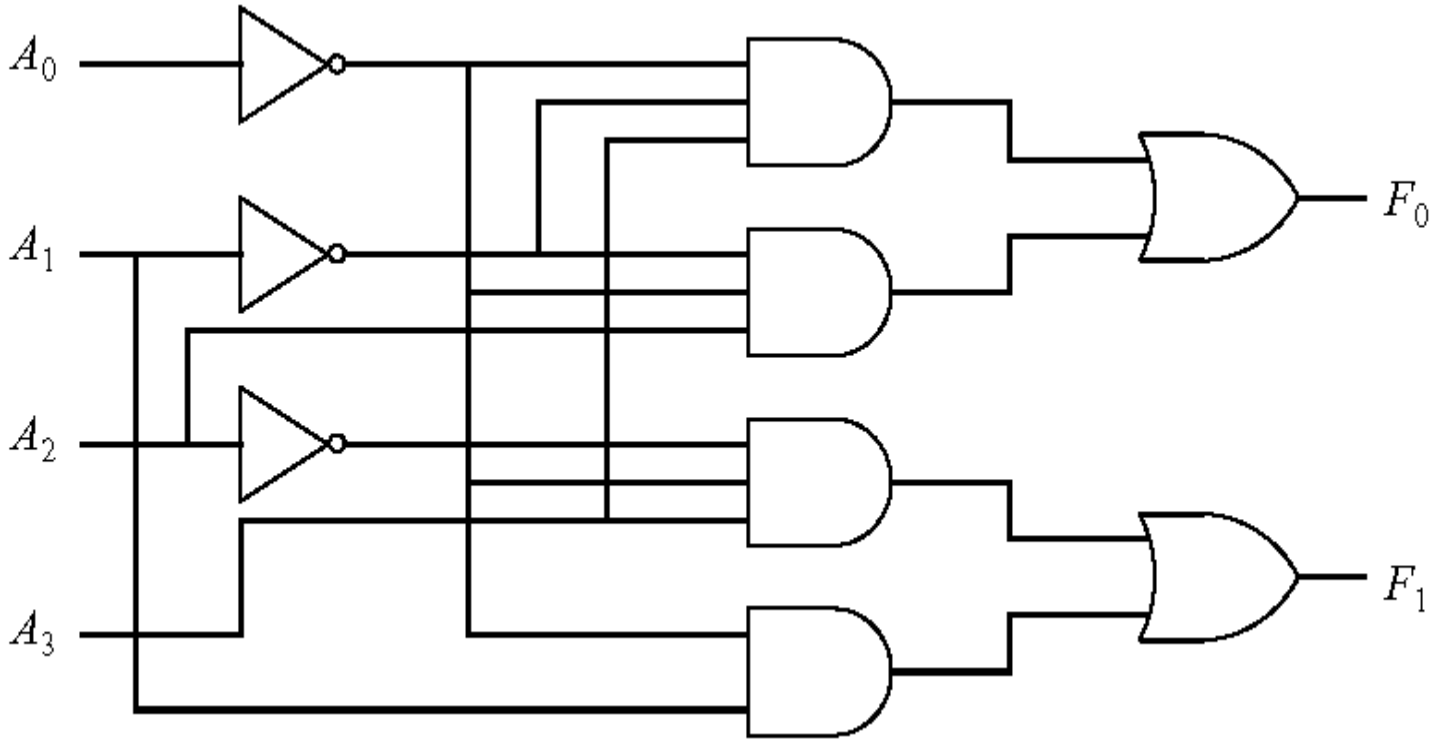


$$F_0 = \overline{A_0}\,\overline{A_1}\,A_3 + \overline{A_0}\,A_1\,A_2$$

$$F_1 = \overline{A_0}\,\overline{A_2}\,A_3 + \overline{A_0}\,A_1$$

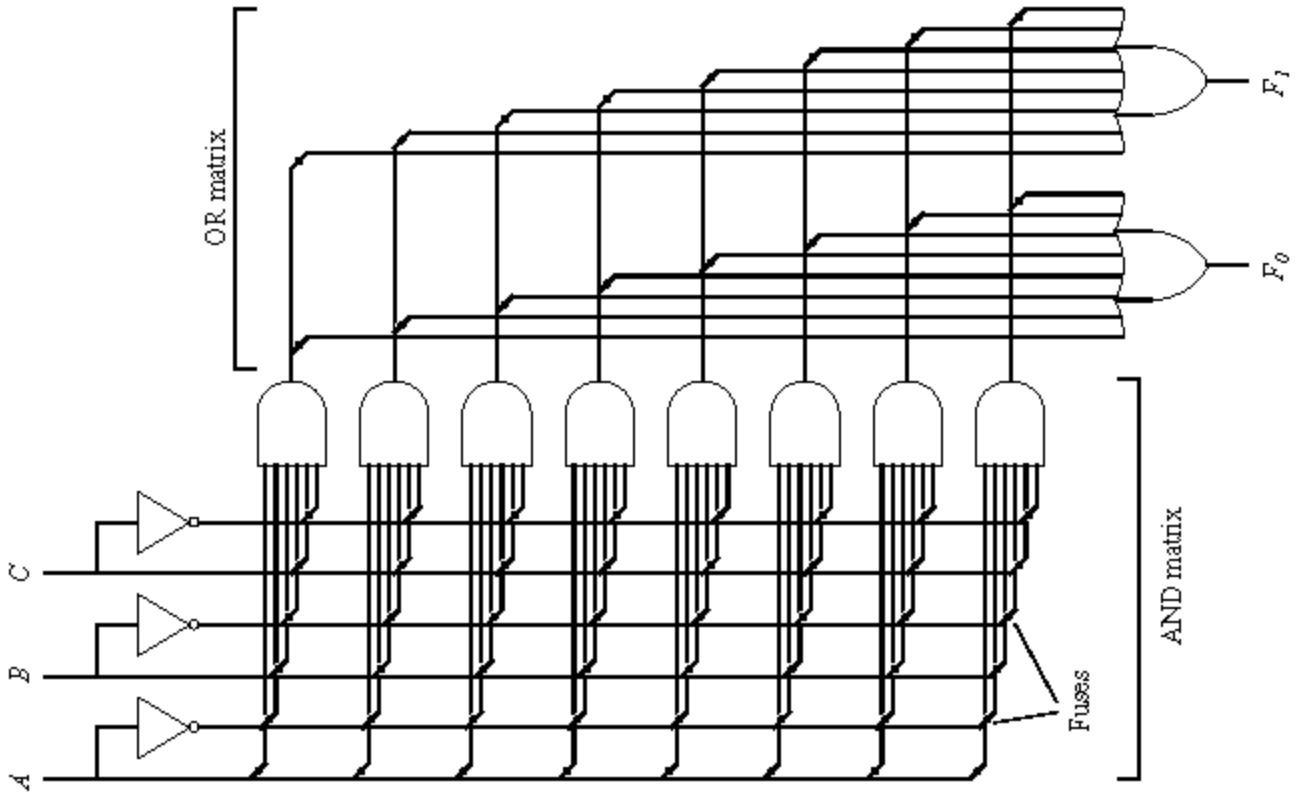
A_0	A_1	A_2	A_3	F_0	F_1
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	0	0

Implementación de un codificador con prioridad

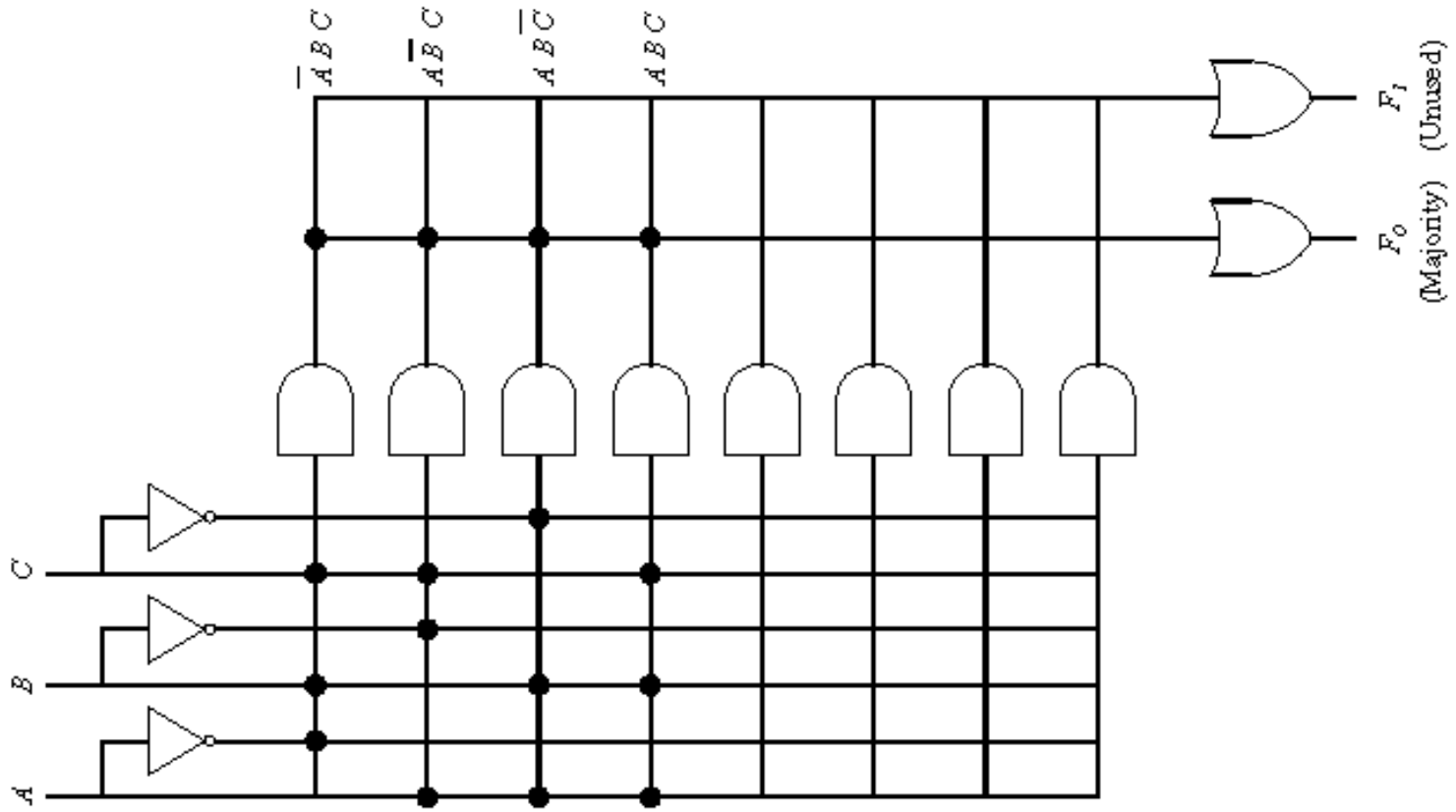


Una PLA (Programmable Array Logic)

- Una PLA es una matriz personalizable de compuertas AND seguida de una matriz personalizable de compuertas OR



La función mayoría implementada con una PLA

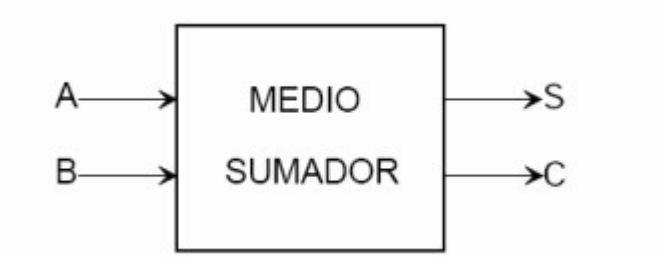


Semi-Sumador

- Este circuito es capaz de sumar dos datos de un bit y producir un bit de acarreo a la salida además del resultado

Tabla de verdad del semi-sumador

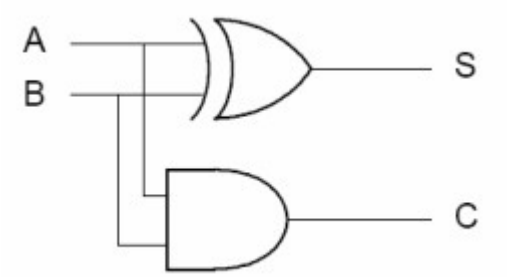
A B	C S
0 0	0 0
0 1	0 1
1 0	0 1
1 1	1 0



Se obtienen dos funciones

$S = A \oplus B$

$C = A \cdot B$

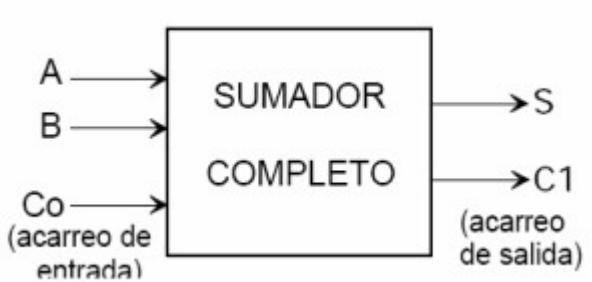


Sumador completo

- El semi-sumador no es capaz de interconectarse con otros sumadores para formar uno más grande.

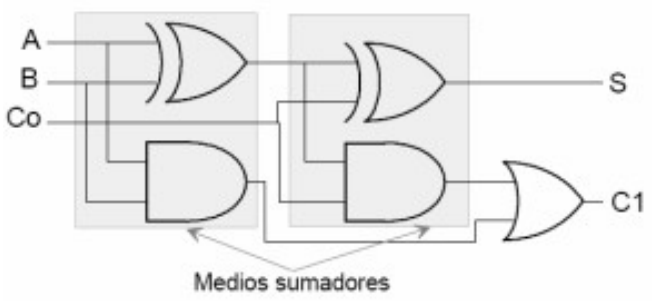
Tabla de verdad del sumador completo

A B Co	C_1 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

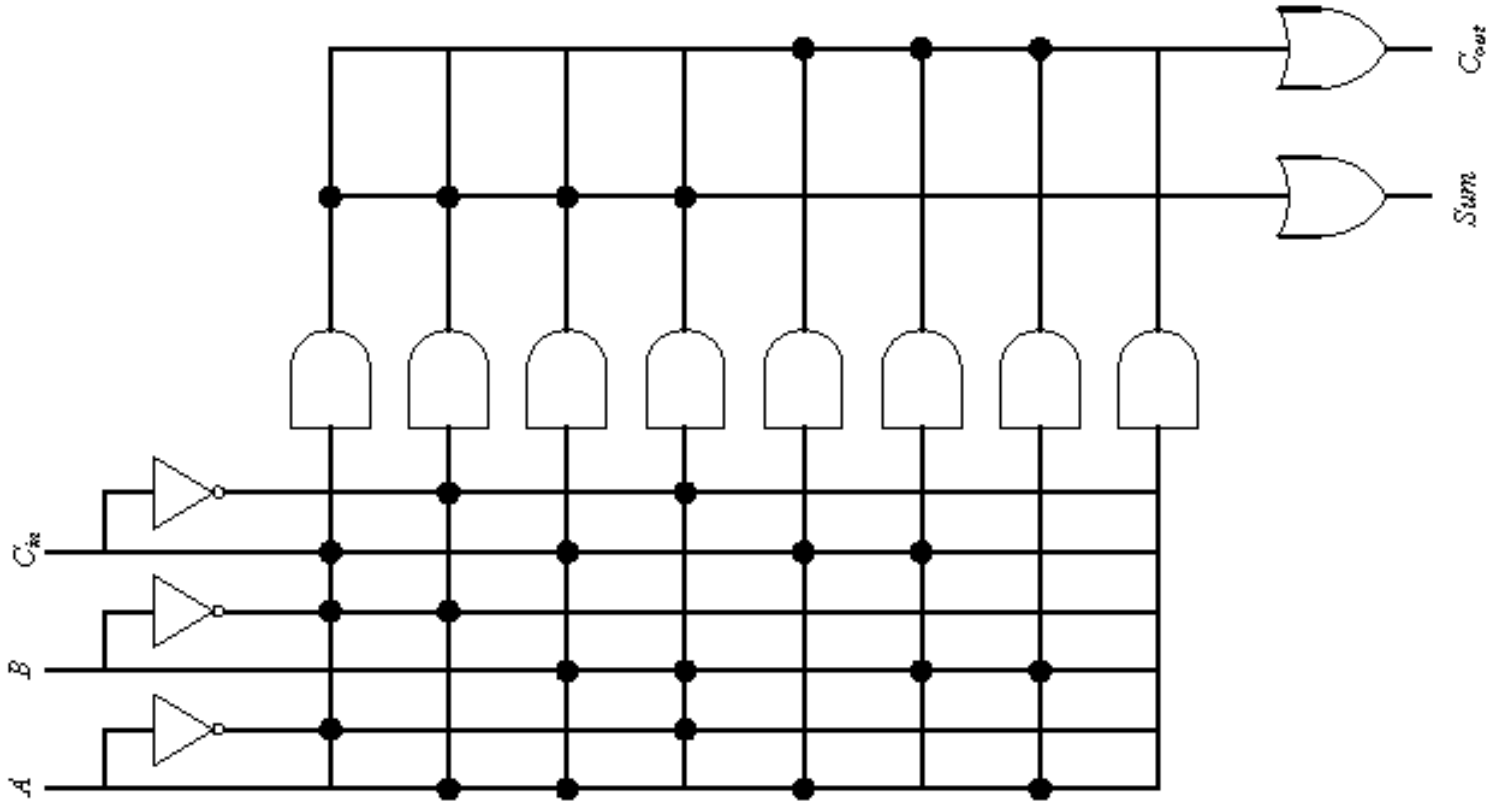


Se obtienen dos funciones

$$S = A \oplus B \oplus C_0, \quad C_1 = AB + (A \oplus B)C_0$$



Un sumador completo implementado con una PLA



Un sumador de cuatro bits en cascada