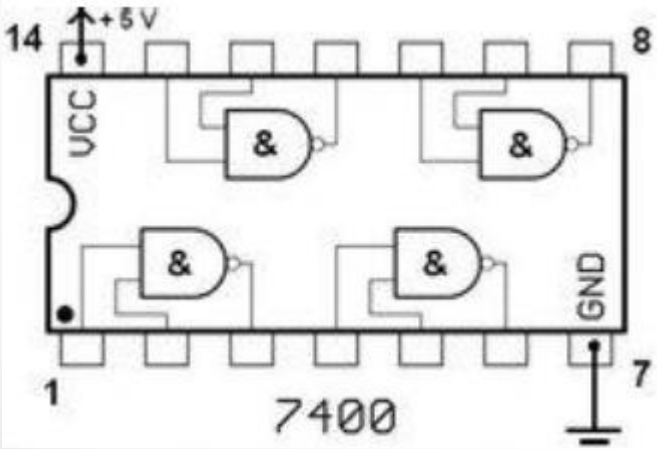
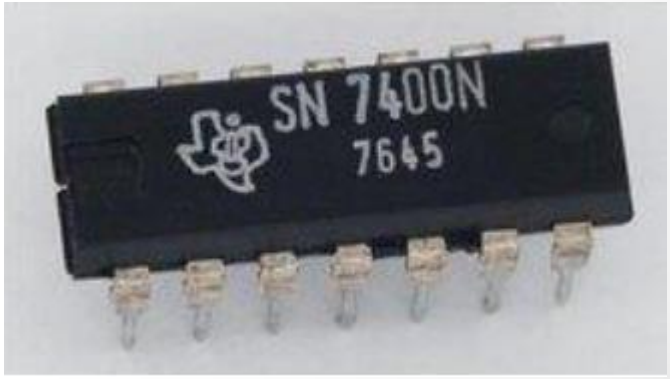
Unidad 3: Circuitos combinacionales

Repaso de conceptos de Algebra de Boole. Circuitos digitales básicos. Circuitos integrados. Circuitos lógicos combinacionales: sumadores, sustractores, conversores de código.

Relojes. El factor tiempo: circuitos lógicos secuenciales. Concepto de memoria y lógica programable. Registros y contadores.





Algebra de Boole

Es un sistema matemático usado para la representación de la lógica, desarrollada en 1854 por George Boole (Inglaterra). En 1938, Claude Shannon (EE.UU.) la propuso como la herramienta para resolver problemas de diseño de circuitos de conmutación.

Al ser una lógica binaria (verdadero/falso; todo/nada), resulta una herramienta adecuada para para el análisis y diseño de circuitos digitales, que utilizan dos niveles lógicos (“0” y “1”; “bajo” y “alto”).

Símbolos y notación

- ✓ En general, las variables booleanas se representan con letras: A, B, etc.; 0 y 1 son constantes.
- ✓ El operador de la suma lógica se representa con el de la suma algebraica (+), y el del producto lógico con el del producto aritmético (·) o bien se omite.
- ✓ Los complementos de una variable o de un grupo de variables relacionadas por un operador pueden representarse como:

$$\overline{A} \equiv A' \quad ; \quad \overline{A+B} \equiv (A+B)'$$

- ✓ En ocasiones, se utilizan los símbolos X o * para indicar que el valor de una variable puede ser 0 o 1, sin importar su valor.

Postulados y teoremas

	Relación	Dual	Propiedad
Postulados	$A\,B = B\,A$	$A + B = B + A$	Conmutativa
	$A\,(B + C) = A\,B + A\,C$	$A + B\,C = (A + B)\,(A + C)$	Distributiva
	$1\,A = A$	$0 + A = A$	Elemento neutro
	$A\,\overline{A} = 0$	$A + \overline{A} = 1$	Complemento
Teoremas	$0\,A = 0$	$1 + A = 1$	Identidad
	$A\,A = A$	$A + A = A$	Idempotencia
	$A\,(B\,C) = (A\,B)\,C$	$A + (B + C) = (A + B) + C$	Asociatividad
	$\overline{\overline{A}} = A$		Involución
	$\overline{A\,B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$	DeMorgan
	$AB + \overline{A}C + BC = AB + \overline{A}C$	$(A + B)(\overline{A} + C)(B + C) = (A + B)(\overline{A} + C)$	Consenso
	$A\,(A + B) = A$	$A + A\,B = A$	Absorción

Postulados

Teoremas

Principio de dualidad: El dual de una función booleana se obtiene intercambiando: 1) sumas lógicas (O) y productos (Y); 2) unos y ceros.

Funciones lógicas

Todas las posibles combinaciones de salida de dos variables definen 16 funciones:

A	B	*Falso*	*AND*	$A\overline{B}$	A	$\overline{A}B$	B	*XOR*	*OR*	*NOR*	*XNOR*	$\overline{B}$	$A+\overline{B}$	$\overline{A}$	$\overline{A}+B$	*NAND*	*Verdad*
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Siempre existen tres formas equivalentes de expresar una función:

Ecuación booleana →

Tabla de verdad →

Símbolo lógico →

NOT

$F = \overline{A}$

A	F
0	1
1	0

AND

$F = A\,B$

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

OR

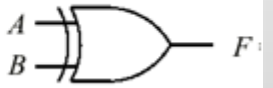
$F = A + B$

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

XOR

$F = A \oplus B$

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



Cosas útiles para recordar

- En un AND, basta que una de sus entradas sea 0 para que la función valga 0.

$$X \cdot Y \cdot Z \cdot \ldots \cdot 0 = 0$$

- En un OR, basta que una de sus entradas sea 1 para que la función valga 1.

$$X + Y + Z + \ldots + 1 = 1$$

- El XOR de una variable con 1 invierte el valor de la variable.

$$A \oplus 1 = A'$$

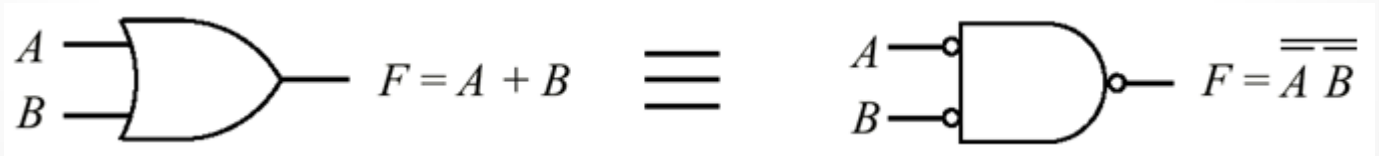
- El XOR de una variable con 0 deja el valor de la variable como estaba.

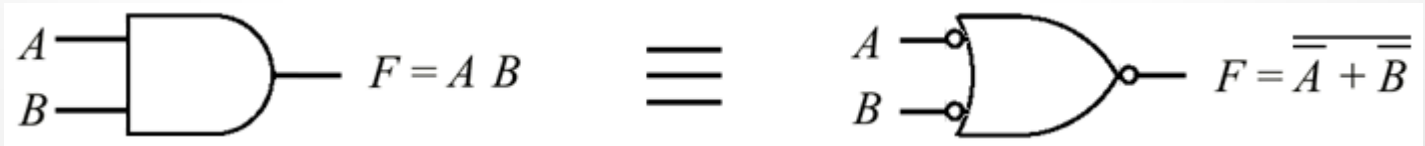
$$A \oplus 0 = A$$

Teorema de DeMorgan

Es particularmente útil, porque permite transformar una compuerta en otra

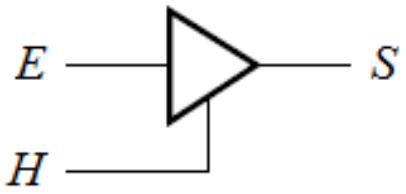
A	B	$\overline{A\,B} = \overline{A} + \overline{B}$		$\overline{A+B} = \overline{A}\,\overline{B}$	
0	0	1	1	1	1
0	1	1	1	0	0
1	0	1	1	0	0
1	1	0	0	0	0





Separadores (buffers) tri-estado

Si bien no forman parte del álgebra de boole, en los circuitos lógicos que utilizan buses (líneas a las que se conectan varias entradas y salidas), es común utilizar separadores "*tri-state*", que se representan y funcionan según la siguiente tabla de verdad:



H	E	S
0	X	*Hi-Z*
1	0	0
1	1	1

La condición "*Hi-Z*" deriva del funcionamiento eléctrico y significa "alta impedancia", un estado en el cual la salida se comporta como si estuviera desconectada del resto del circuito.

Circuitos combinacionales y secuenciales

- **Circuito Combinacional**: circuito lógico en el cual el valor de las salidas depende *exclusivamente* de los valores lógicos presentes en las entradas.

  → Un cambio en una entrada *siempre* produce el mismo cambio en la salida.

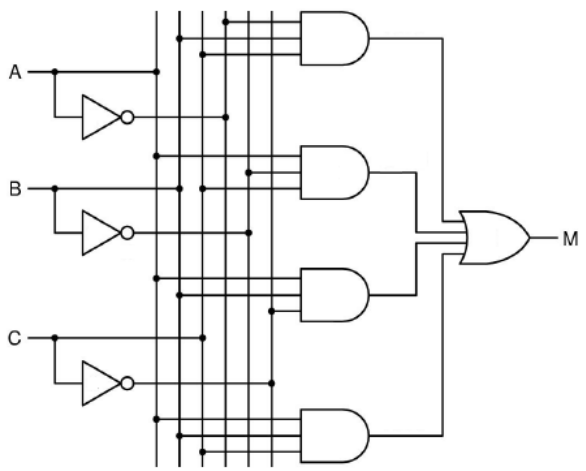
- **Circuito Secuencial**: circuito lógico en el cual el valor de las salidas depende *no solo* de los valores lógicos presentes en las entradas, sino también de la secuencia temporal previa de esos valores en esas entradas.

  → Un cambio en una entrada *no siempre* produce el mismo cambio en la salida.

Circuitos combinacionales

Como en el caso de las funciones lógicas básicas, las salidas de los circuitos combinacionales pueden expresarse por tres formas equivalentes: una *expresión booleana*, una *tabla de verdad*, y un *esquema circuital*.

A	B	C	M
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



$$M(A,B,C) = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

→ *Es posible encontrar expresiones booleanas y diagramas circuitales diferentes, que sin embargo representan la misma función lógica. Esto se verifica* ***si las tablas de verdad son las mismas****.*

Obtención de circuitos a partir de la tabla de verdad

En general, la forma más sencilla de resolver cualquier problema de naturaleza combinacional es partir de su tabla de verdad. Ésta esta se obtiene naturalmente listando el valor lógico de la salida para cada combinación de las variables de entrada, a partir de las especificaciones o la descripción verbal del problema.

A partir de ella, hay dos opciones:

- Usar componentes MSI (como se verá más adelante) y obtener el circuito en forma directa a partir de la misma tabla; o bien,
- Obtener una expresión booleana, y con ésta, un circuito. Sin embargo, rara vez se obtiene la forma más simple, y es necesario *simplificarlos* aplicando las propiedades y teoremas del álgebra de Boole, o técnicas como el *mapa de Karnaugh.*

Un ejemplo: la función mayoría

→ *Es más simple entender los conceptos anteriores a partir de un ejemplo concreto*

Se define una función M de tres variables A, B y C, tal que M vale uno si en las entradas hay más unos que ceros (una mayoría de votos afirmativos).

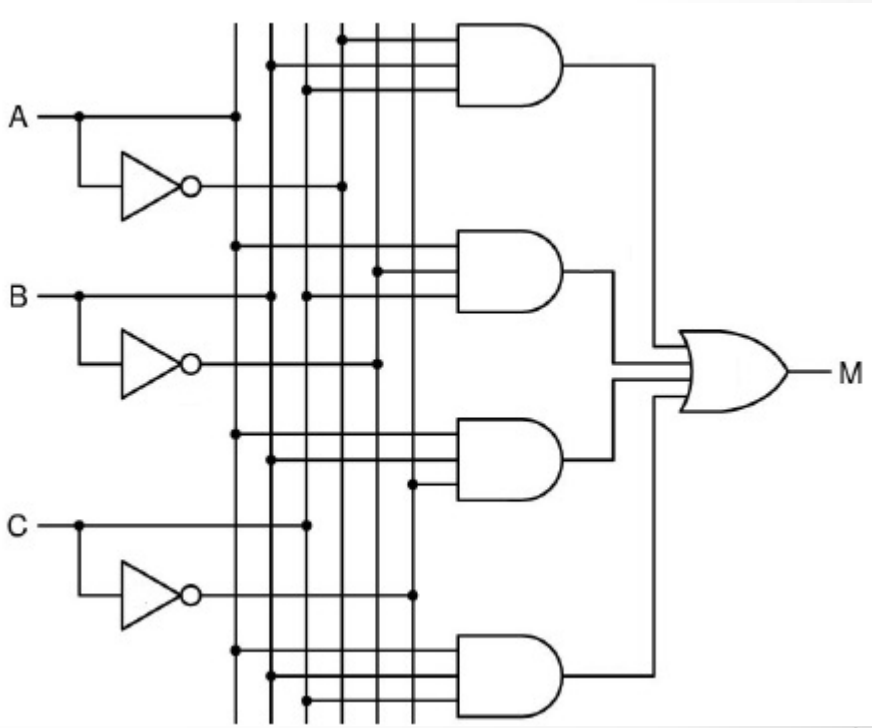
Su tabla de verdad resulta:

A	B	C	M
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

En forma directa, se obtiene que:

$$M = A'BC + AB'C + ABC' + ABC$$

Es circuito resultante es:



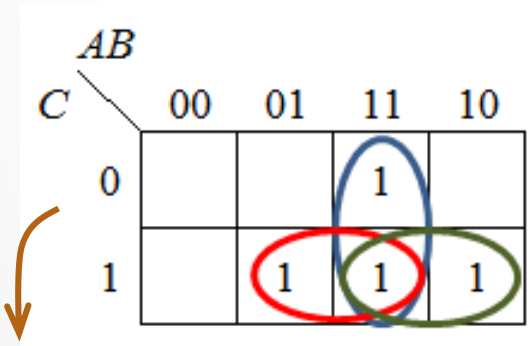
Un ejemplo: la función mayoría

El circuito obtenido puede hacerse más pequeño

Es deseable minimizar los circuitos porque, en general:

- ✓ *Se consume menos energía*
- ✓ *El circuito es más rápido (se reducen los retardos)*
- ✓ *Al haber menos conexiones, hay menor probabilidad de fallas*
- ✓ *El diseño es más barato!!!*

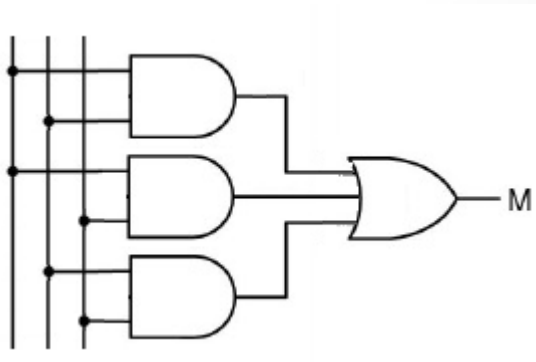
Simplificando por Karnaugh:



Se obtiene:

$$M = AB + AC + BC$$

El circuito resultante es:



Componentes combinacionales MSI

Existen circuitos combinacionales constituidos por un conjunto de compuertas lógicas que actúan como un bloque funcional. Como integran un alto número de compuertas, reciben el nombre de MSI (*Middle Scale of Integration*).

Además de su función específica –muchas de las cuales se utilizan en los diseños de los circuitos típicos de una computadora-, varios de ellos permiten una implementación más rápida de las funciones combinacionales.

En función de su utilización en los circuitos presentes en una computadora, se estudiarán:

- Sumadores
- Decodificadores
- Multiplexores y demultiplexores

Sumadores

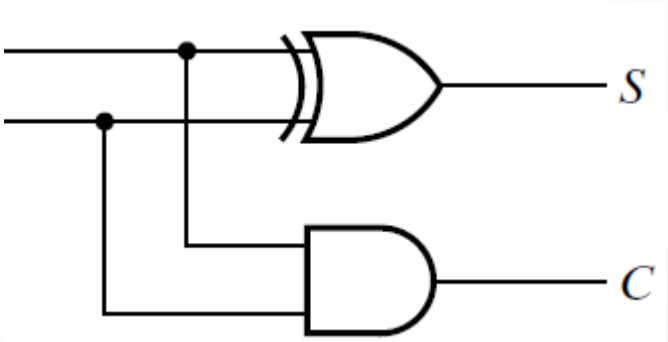
Los circuitos sumadores implementan la función de la suma *aritmética*.

→ Para la suma de un dígito, intervienen dos operandos de entrada y uno de salida (el resultado). Además, en cualquier base es necesario establecer una salida adicional, para contemplar el acarreo que se produce cuando la suma de dos dígitos es mayor que el símbolo de mayor valor de la base.

En base 2 será, siendo S el resultado y C el acarreo:

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

→Se obtiene un *semi sumador:*

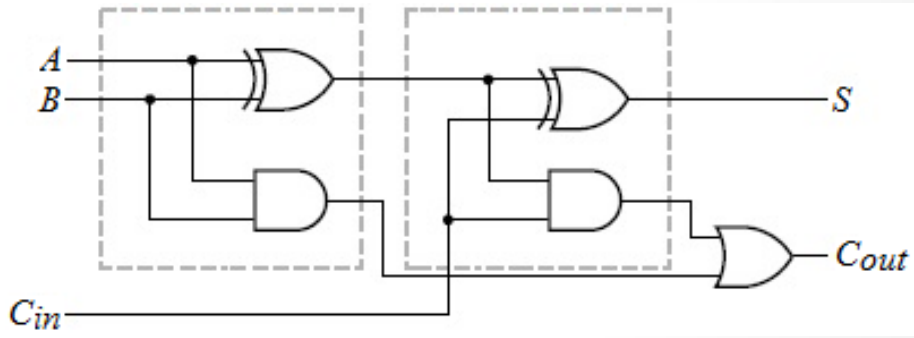


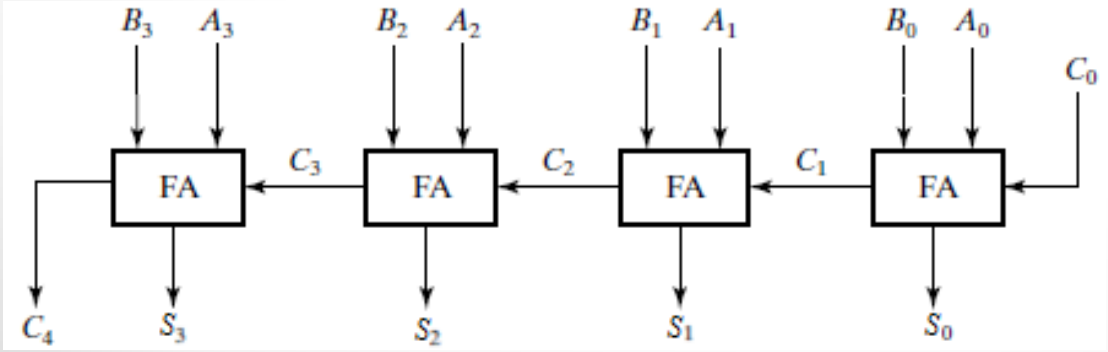
Sumadores

El circuito anterior representa la suma del dígito menos significativo; para los restantes, es necesario considerar también el efecto del acarreo de la etapa anterior.

C_{in}	A	B	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

→Un sumador completo puede ser implementado con dos semi sumadores





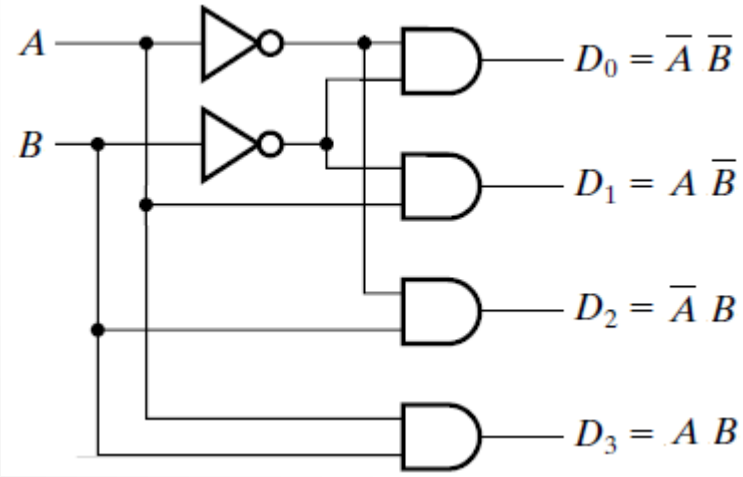
→Varios sumadores completos de un bit se pueden conectar en cascada para formar uno de *n* bits

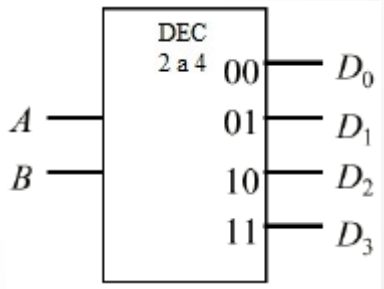
Decodificadores

Para las aplicaciones en que se utilizarán, un decodificador puede considerarse como un circuito que recibe como entradas *n* líneas conteniendo un código binario, y activa en sus *m* salidas solo la correspondiente al valor presente en la entrada. En general, es $m = 2^n$.

Por ejemplo, para dos bits de entrada:

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



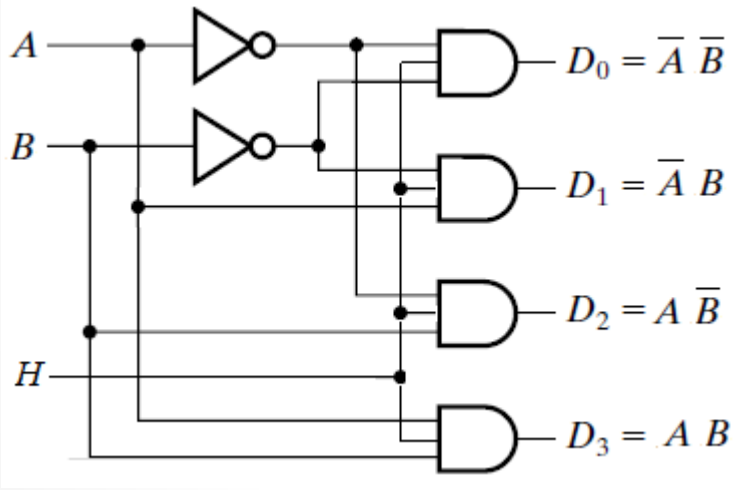


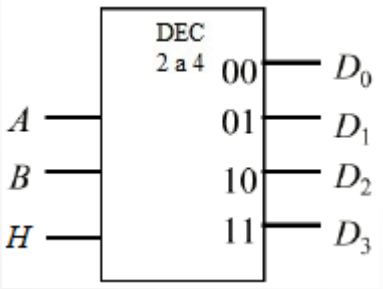
Decodificadores

En muchas aplicaciones, los decodificadores poseen una entrada adicional de habilitación *H*, que al estar inactiva provoca que ninguna salida esté activa, independientemente del código de entrada.

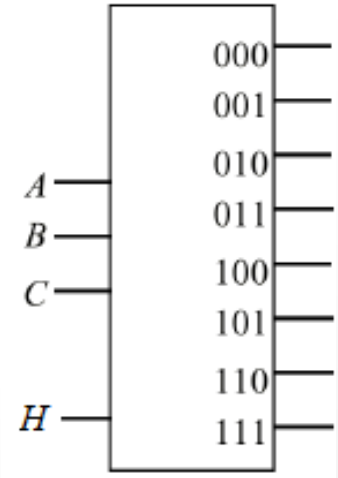
H	A	B	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

X = no importa





Esquemáticamente, un decodificador de 3 a 8 será:



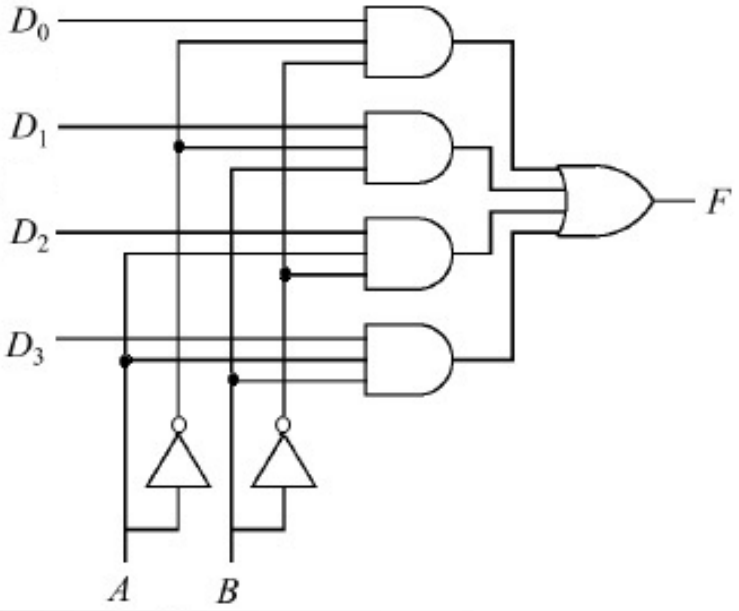
Multiplexores

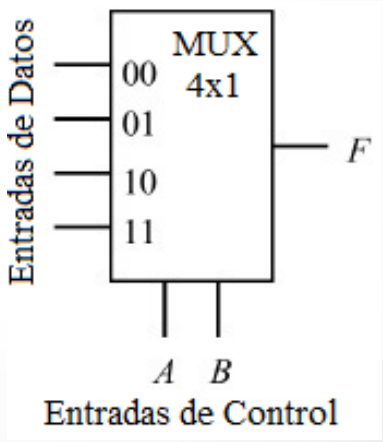
El multiplexor es un circuito que selecciona el valor lógico de una de sus *n* entradas y lo presenta su única salida, de acuerdo al valor presente en sus *entradas de selección*.

Por ejemplo, para un multiplexor de 4x1:

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

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





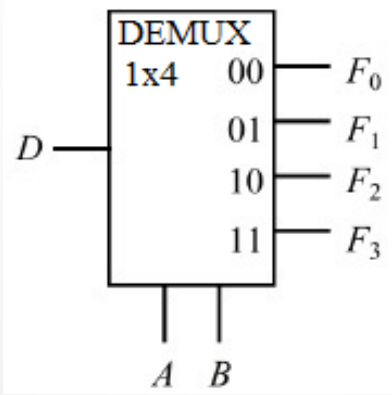
Demultiplexores

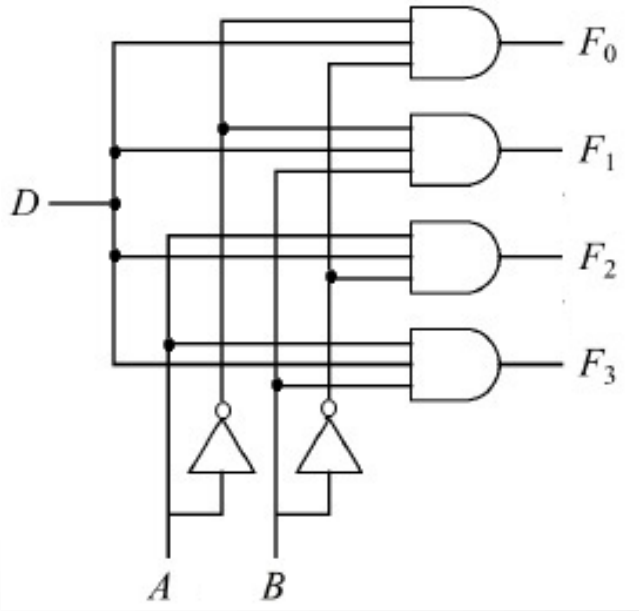
El demultiplexor realiza la función inversa del multiplexor: presenta el valor lógico presente en su única entrada en una de sus *n* salidas, de acuerdo al valor presente en sus entradas de selección.

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

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

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





Funciones combinacionales con MSI

Además de su función propia como bloque, los decodificadores y multiplexores permiten implementar muy fácilmente circuitos combinacionales a partir de su tabla de verdad.

Mintérminos: para la función mayoría, se dedujo que la expresión booleana resultante de su tabla de verdad es:

	A	B	C	M
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

$$M = A'BC + AB'C + ABC' + ABC$$

→ Cada expresión en la que intervienen *todas* las variables de la función, negadas o no, e interrelacionadas por el producto lógico, recibe el nombre de *minitérmino*, y se denota como m_i , donde i es el equivalente decimal del valor binario ABC

Funciones combinacionales con MSI

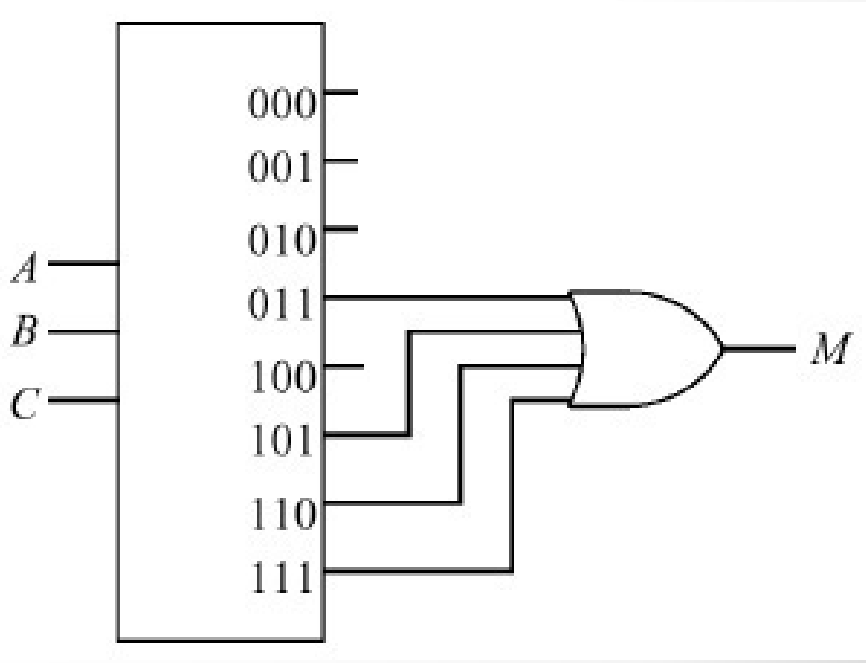
	A	B	C	M
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

$$M = A'BC + AB'C + ABC' + ABC$$

La función Mayoría puede expresarse entonces como:

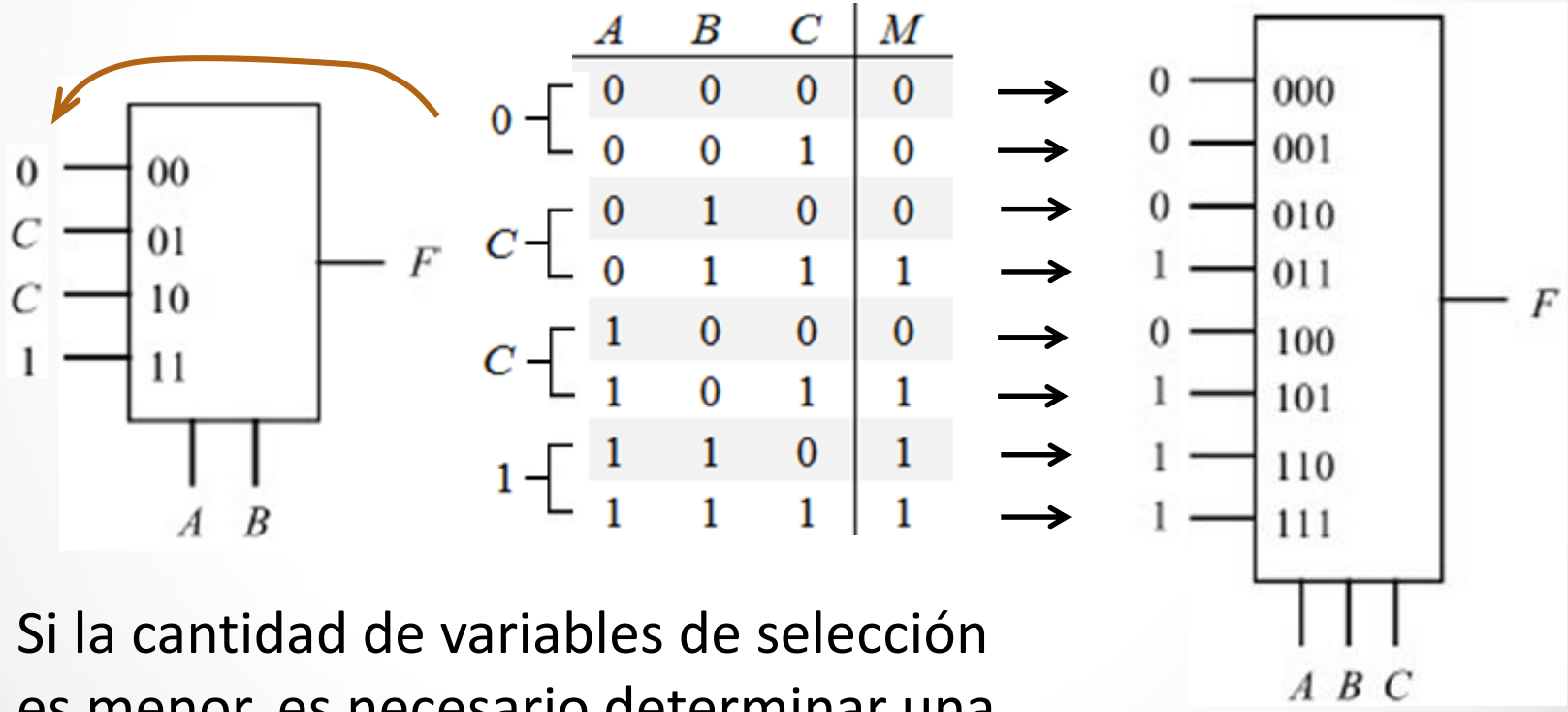
$$M = m_3 + m_5 + m_6 + m_7 = \Sigma\, m_i\,(3; 5; 6; 7)$$

Un decodificador puede verse como un “generador de minitérminos”, por lo que resulta muy simple la implementación de una función



Funciones combinacionales con MSI

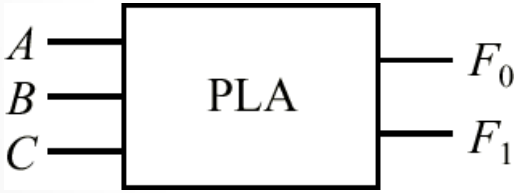
Recordando el funcionamiento de un multiplexor, se ve que, si se utiliza uno que tenga la misma cantidad de variables de selección que la de variables de la función, la implementación es directa:



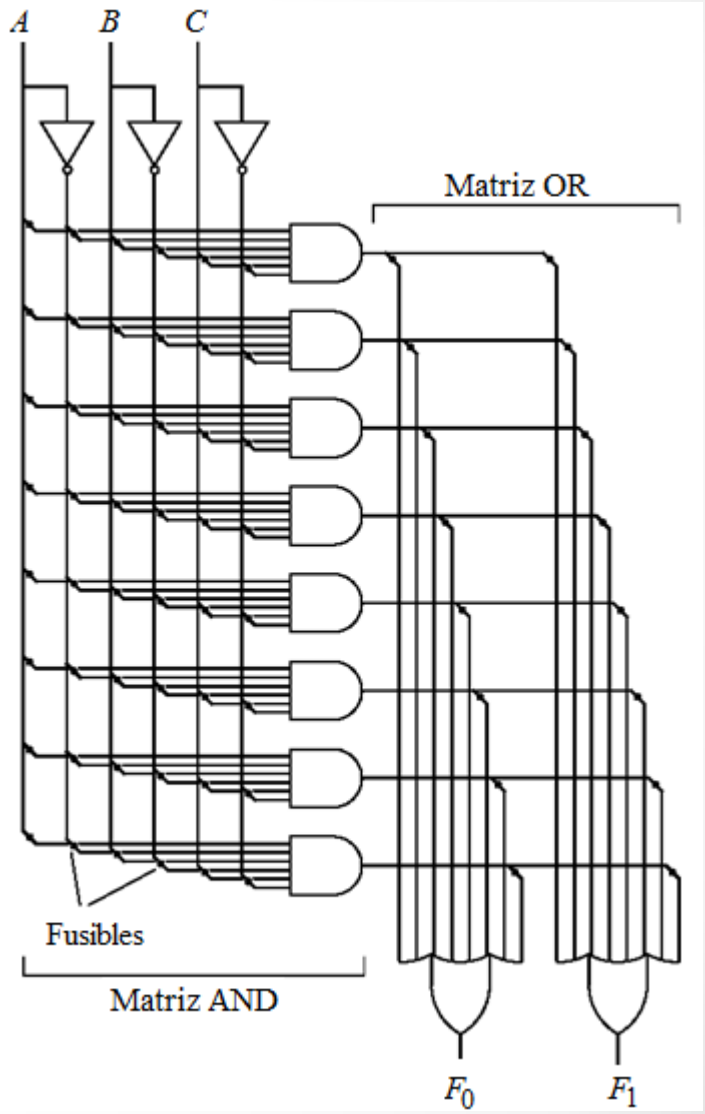
Si la cantidad de variables de selección es menor, es necesario determinar una función lógica para cada entrada

Matrices lógicas programables (PLAs)

- Una PLA es una matriz configurable de compuertas AND seguida de otra matriz configurable de compuertas OR.
- Se personalizan anulando fusibles para una función específica.

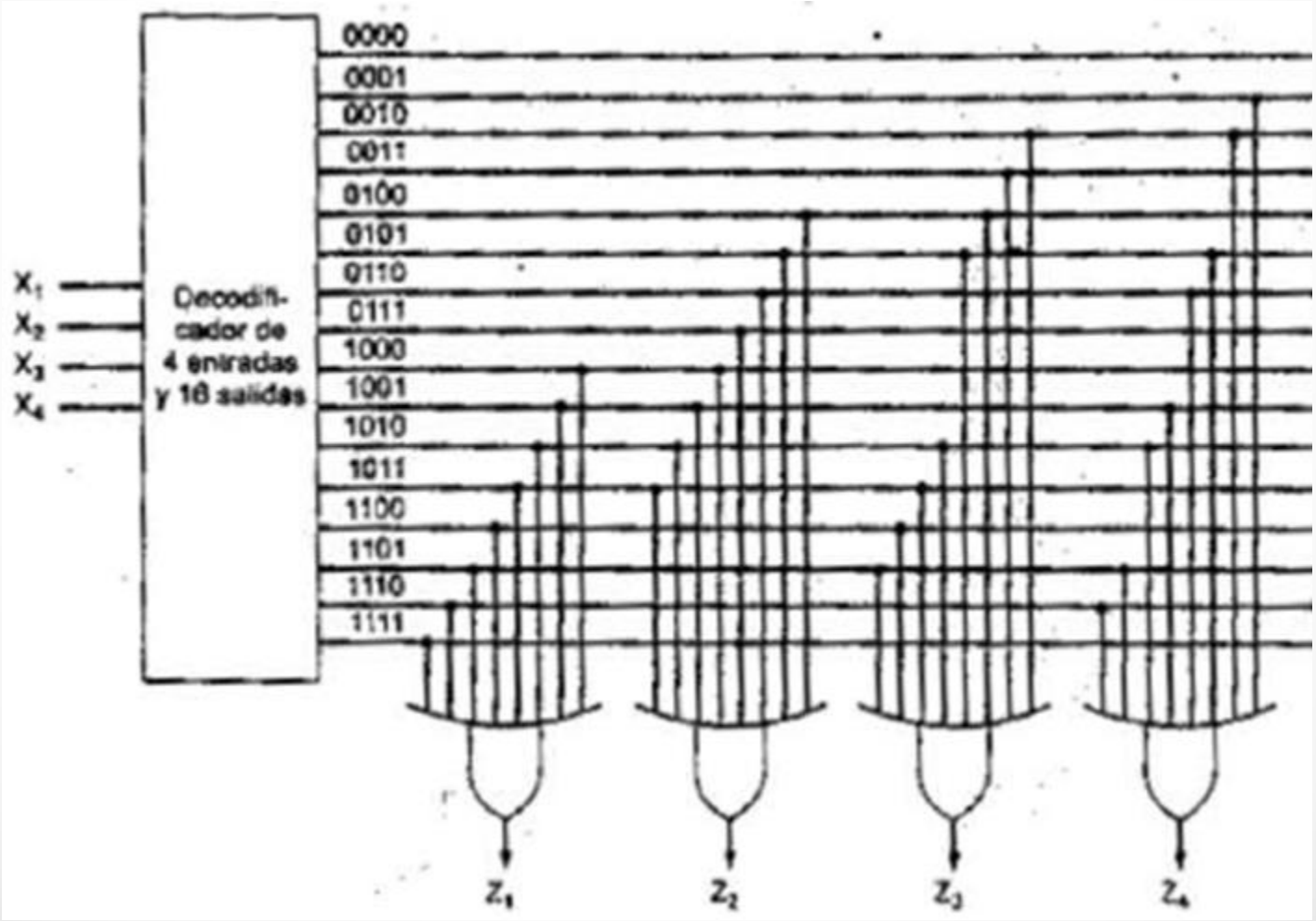


- La configuración y ejecución de las conexiones lo realiza un software específico.



Memorias de solo lectura (ROMs)

- Una ROM (*Read Only Memory*) puede considerarse un decodificador con varias compuertas OR conectadas. Cada conexión es programable.



Lecturas recomendadas

- Stallings, Williams - Organización y Arquitectura de Computadoras - 5º Ed. - Prentice Hall. Año 2000.

  → *Apéndice A*

- Murdocca, Miles J. - Principios de arquitectura de computadoras - 1º Ed. - Prentice Hall - Año 2002.

  → *Apéndices A y B*