Codificación de la información y álgebra de conmutación

ÍNDICE

- Analógico vs Digital
- Niveles lógicos
- Sistemas de numeración
- Álgebra booleana
- Implementación de funciones

- Bibliografía
  - Analógico vs Digital. Capítulo 1. Pags 1 a 9
  - Sistemas de numeración. Capítulo 2. Pags 25 a 46
  - Algebra booleana. Capítulo 4. Pags 193 a 236

Analógico vs. digital

Analógico:

– Las señales varían de forma continua en un rango dado de tensiones, corrientes, etc.

Digital:

– Las señales varían de forma continua en un rango dado de tensiones, corrientes, etc…

– ¡ Pero asumiremos que no lo hacen !

• **Abstracción digital**: La señal digital se modela como si tuviera, en cada instante, un valor de entre dos posibles (ignoramos comportamiento analógico)

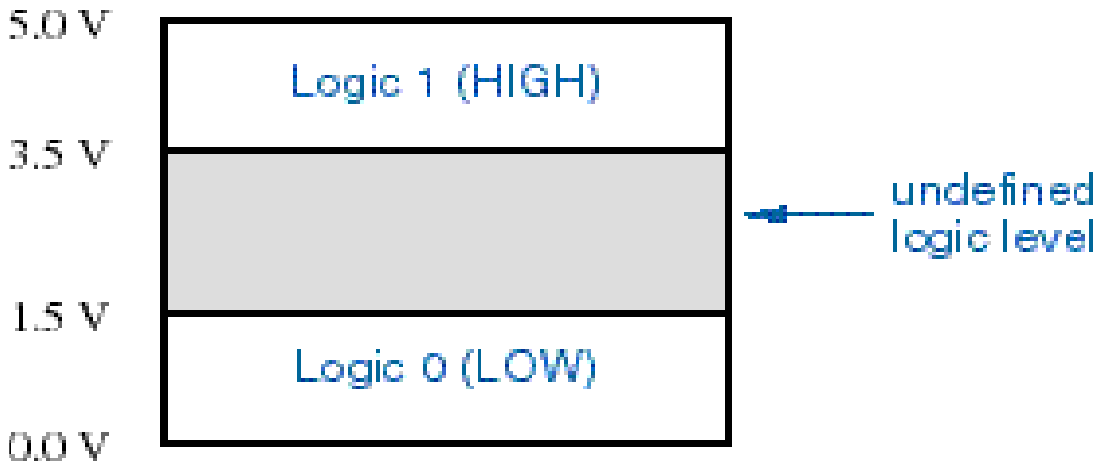
Ventajas de los circuitos digitales

• Reproducibilidad de los resultados (misma entrada misma salida), facilidad de diseño (“ diseño lógico”), flexibilidad y funcionalidad, programabilidad, velocidad, economía, fácil integración con nuevos avances tecnológicos

• **OJO** : No podemos prescindir del diseño analógico

Niveles lógicos

- Región indefinida debida a:
  - Sistema digital
  - Diferenciación débil => fuerte



- El umbral de conmutación varía con la tensión de alimentación, temperatura, proceso tecnológico, envejecimiento, etc.

Es necesario tener un margen de ruido "*noise margin*"

Sistemas de numeración

Un número “N” se representa en base “b” como “...$\mathbf{d_7\, d_6\, d_5\, d_4\, d_3\, d_2\, d_1\, d_0}$”, donde “$\mathbf{d_i}$” son los dígitos.

$$N = ...d_7 b^7 + d_6 b^6 + d_5 b^5 + d_4 b^4 + d_3 b^3 + d_2 b^2 + d_1 b^1 + d_0$$

$d_3\, d_2\, d_1\, d_0$

- d_0 LSB ó bit menos significativo
- d_3 MSB ó bit más significativo

Binario	Decimal	Octal	Hexadecimal
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	8	10	8
1001	9	11	9
1010	10	12	A
1011	11	13	B
1100	12	14	C
1101	13	15	D
1110	14	16	E
1111	15	17	F

Conversión entre las bases más comunes

Conversion	Method	Example
Binary to		
Octal	Substitution	$10111011001_2 = 10\ 111\ 011\ 001_2 = 2731_8$
Hexadecimal	Substitution	$10111011001_2 = 101\ 1101\ 1001_2 = 5D9_{16}$
Decimal	Summation	$10111011001_2 = 1 \cdot 1024 + 0 \cdot 512 + 1 \cdot 256 + 1 \cdot 128 + 1 \cdot 64 + 0 \cdot 32 + 1 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = 1497_{10}$
Octal to		
Binary	Substitution	$1234_8 = 001\ 010\ 011\ 100_2$
Hexadecimal	Substitution	$1234_8 = 001\ 010\ 011\ 100_2 = 0010\ 1001\ 1100_2 = 29C_{16}$
Decimal	Summation	$1234_8 = 1 \cdot 512 + 2 \cdot 64 + 3 \cdot 8 + 4 \cdot 1 = 668_{10}$

Conversión entre las bases más comunes

Conversion	Method	Example
Hexadecimal to		
Binary	Substitution	$C0DE_{16} = 1100\ 0000\ 1101\ 1110_2$
Octal	Substitution	$C0DE_{16} = 1100\ 0000\ 1101\ 1110_2 = 1\ 100\ 000\ 011\ 011\ 110_2 = 140336_8$
Decimal	Summation	$C0DE_{16} = 12 \cdot 4096 + 0 \cdot 256 + 13 \cdot 16 + 14 \cdot 1 = 49374_{10}$
Decimal to		
Binary	Division	$108_{10} \div 2 = 54$ remainder 0 (LSB) $\div 2 = 27$ remainder 0 $\div 2 = 13$ remainder 1 $\div 2 = 6$ remainder 1 $\div 2 = 3$ remainder 0 $\div 2 = 1$ remainder 1 $\div 2 = 0$ remainder 1 (MSB) $108_{10} = 1101100_2$
Octal	Division	$108_{10} \div 8 = 13$ remainder 4 (least significant digit) $\div 8 = 1$ remainder 5 $\div 8 = 0$ remainder 1 (most significant digit) $108_{10} = 154_8$
Hexadecimal	Division	$108_{10} \div 16 = 6$ remainder 12 (least significant digit) $\div 16 = 0$ remainder 6 (most significant digit) $108_{10} = 6C_{16}$

Representación de números negativos

- **Signo y magnitud**.

$$-(2^{n-1}-1) \leq \text{número} \leq (2^{n-1}-1)$$

  - El bit más significativo indica el signo y el resto el módulo
    - $01010101_2 = 85_{10}$
    - $11010101_2 = -85_{10}$

- **Complemento a uno**.

$$-(2^{n-1}-1) \leq \text{número} \leq (2^{n-1}-1)$$

  - El bit más significativo indica el signo y el resto el módulo si el número es positivo. En caso de ser negativo el módulo es el resultante de cambiar los “1” por “0” y viceversa
    - $01010101_2 = 85_{10}$
    - $10101010_2 = -85_{10}$

- **Complemento a dos**.

$$-2^{n-1} \leq \text{número} \leq (2^{n-1}-1)$$

  - Igual al complemento a uno, pero si el número es negativo se suma “1” al resultado. Es la representación más utilizada.
    - $01010101_2 = 85_{10}$
    - $10101011_2 = -85_{10}$

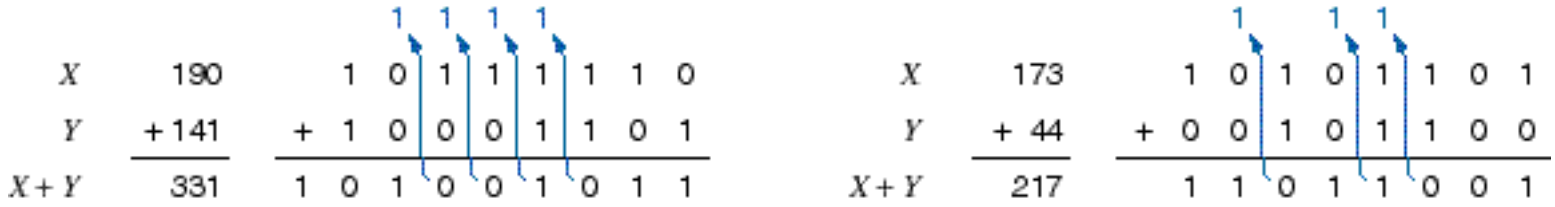
Ejemplo de representación con 4 bits

Decimal	Complemento a dos	Complemento a uno
-8	1000	---
-7	1001	1000
-6	1010	1001
-5	1011	1010
-4	1100	1011
-3	1101	1100
-2	1110	1101
-1	1111	1110
0	0000	1111 ó 0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111

Doble representación para el 0

Suma, resta y producto binarios

- **Suma**



- **Resta**: utilizando el complemento a dos
- **Producto**: El producto "**S**" tiene un número de bits igual a la suma de los del multiplicando "**A**" y el multiplicador "**B**".

Ejemplo: $S = A \bullet B$

$\mathbf{S}(S_7..S_0) = \mathbf{A}(a_3..a_0) \bullet \mathbf{B}(b_3..b_0)$

				a_3	a_2	a_1	a_0
			*	b_3	b_2	b_1	b_0
				b_0a_3	b_0a_2	b_0a_1	b_0a_0
+			b_1a_3	b_1a_2	b_1a_1	b_1a_0	
+		b_2a_3	b_2a_2	b_2a_1	b_2a_0		
+	b_3a_3	b_3a_2	b_3a_1	b_3a_0			
S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

Reglas de suma y resta

- Suma:
  - Se ignora el acarreo de salida del bit más significativo

+ 3	0 0 1 1	− 2	1 1 1 0	+ 6	0 1 1 0	+ 4	0 1 0 0
+ + 4	+ 0 1 0 0	+ − 6	+ 1 0 1 0	+ − 3	+ 1 1 0 1	+ − 7	+ 1 0 0 1
+ 7	0 1 1 1	− 8	1 1 0 0 0	+ 3	1 0 0 1 1	− 3	1 1 0 1

  - Se produce desbordamiento si los signos de los sumandos son iguales y el del resultado es diferente

    ó si los acarreos de entrada y salida del bit más significativo son diferentes

− 3	1 1 0 1	+ 5	0 1 0 1	− 8	1 0 0 0	+ 7	0 1 1 1
+ − 6	+ 1 0 1 0	+ + 6	+ 0 1 1 0	+ − 8	+ 1 0 0 0	+ + 7	+ 0 1 1 1
− 9	1 0 1 1 1	+ 11	1 0 1 1	− 16	1 0 0 0 0	+ 14	1 1 1 0

- Resta:
  - Complementar a dos el sustraendo y realizar la suma, o bien invertir el sustraendo y sumar “1” al resultado

Álgebra Booleana

- También se conoce como álgebra de conmutación
  - Trabaja con valores “0” y “1”
- Existen dos convenciones
  - Lógica positiva
    - “L” o nivel bajo es equivalente a “0”
    - “H” o nivel alto es equivalente a “1”
  - Lógica negativa (poco utilizada)
    - “L” o nivel bajo es equivalente a “1”
    - “H” o nivel alto es equivalente a “0”
- Las señales se nombran con variables (por ejemplo **X**, **Y**, **RST**, **CLK**, etc.)
- Las señales pueden valer solamente “0” o “1”

Operadores booleanos

- NOT: X´(complementario de X; otras veces X)
- AND: X • Y
- OR: X + Y

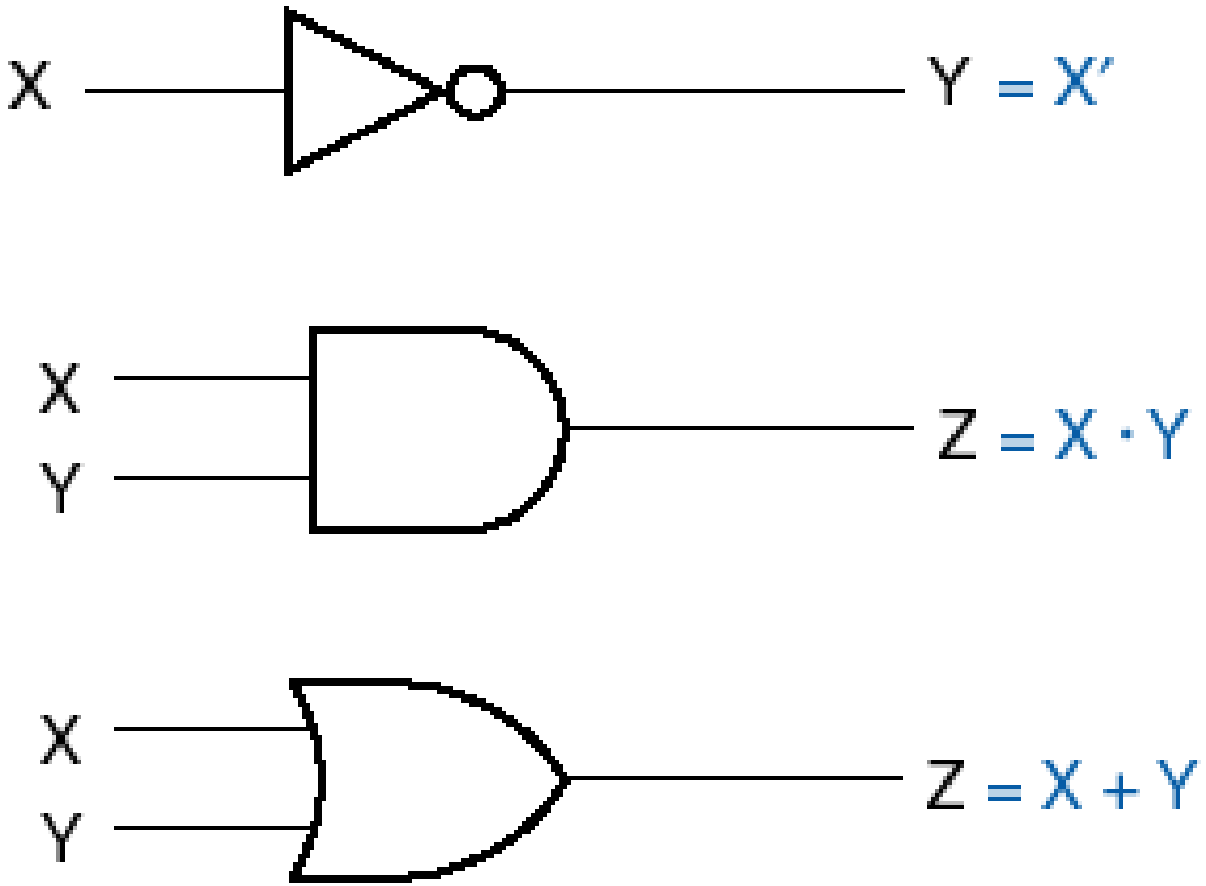
X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

X	NOT X
0	1
1	0

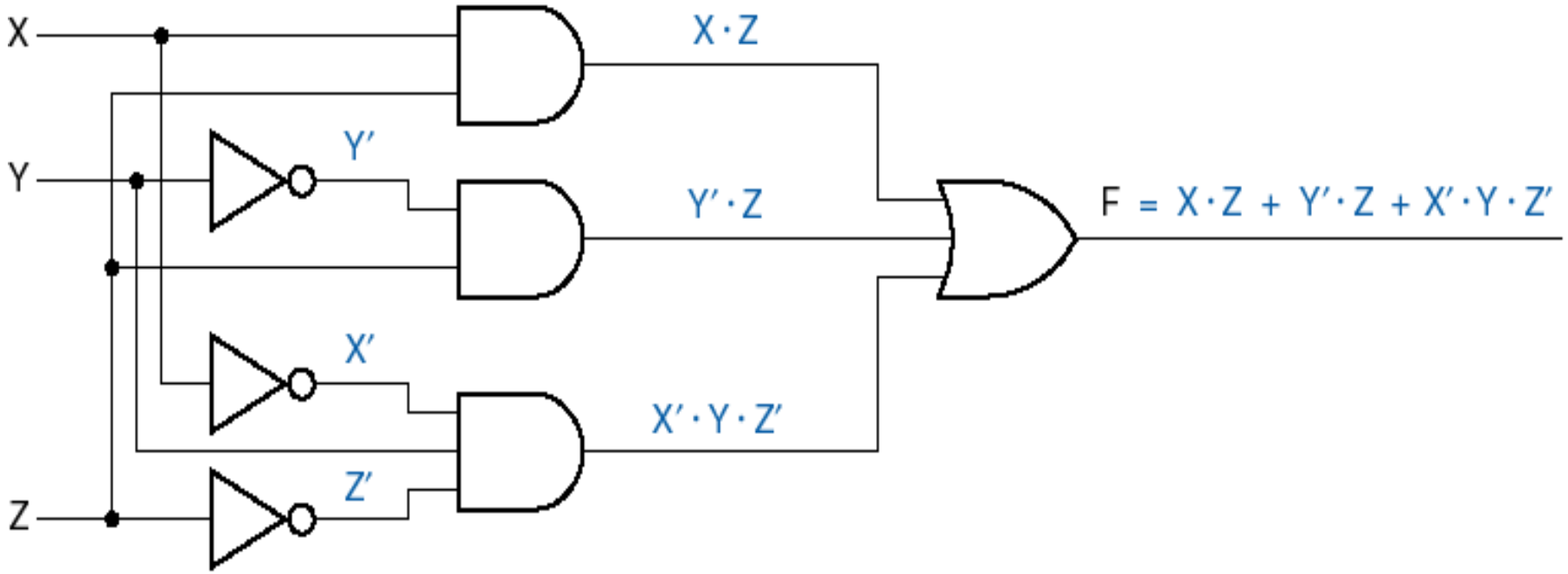
- Los operadores se definen por su tabla de verdad

Símbolos Lógicos



Ejemplo

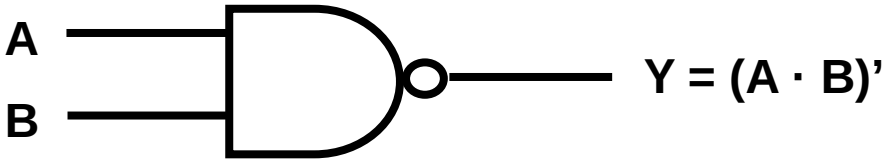
implementar F = (X • Z) + (Y´ • Z) + (X´ • Y • Z´)



Puertas NAND y NOR

NAND

Combinación de una puerta
AND seguida de un inversor (NOT)



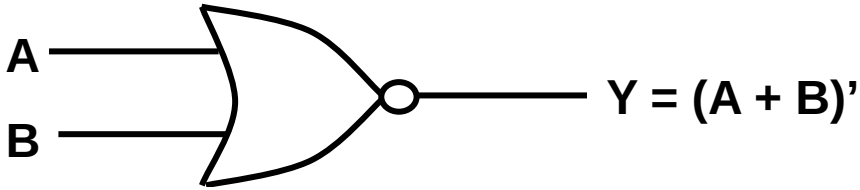
Y = (A · B)’

Inputs		Output
A	B	Y
L	L	H
L	H	H
H	L	H
H	H	L

H = High Logic Level
L = Low Logic Level

NOR

Combinación de una puerta
OR seguida de un inversor (NOT)

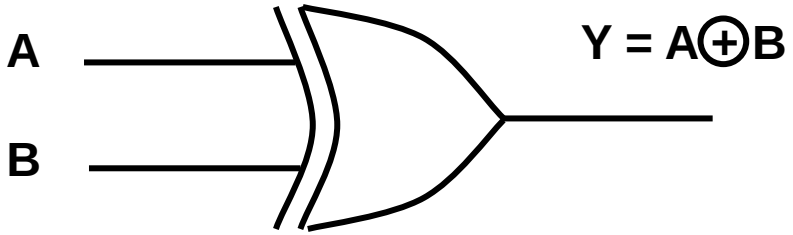


Y = (A + B)’

Inputs		Output
A	B	Y
L	L	H
L	H	L
H	L	L
H	H	L

H = High Logic Level
L = Low Logic Level

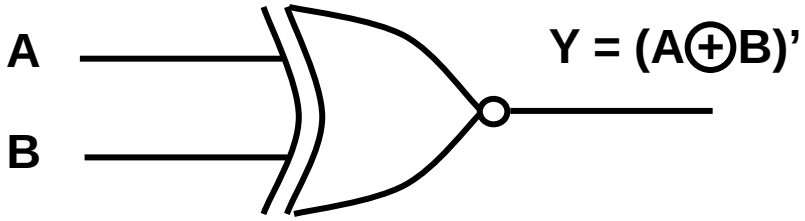
Puertas EXOR y EXNOR



$Y = A \oplus B$

Inputs		Output
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	L

H = High Logic Level
L = Low Logic Level



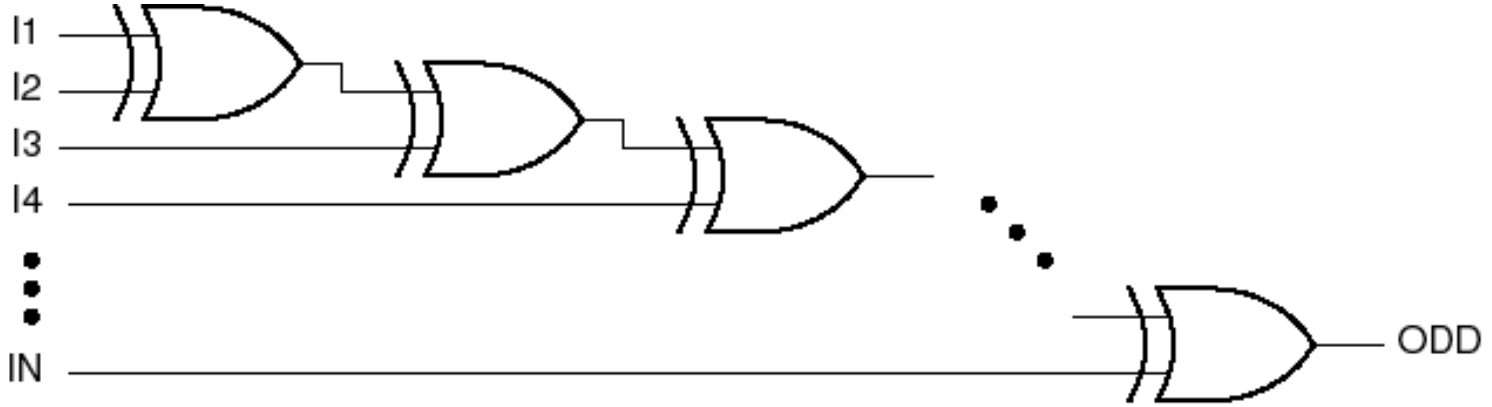
$Y = (A \oplus B)'$

Inputs		Output
A	B	Y
L	L	H
L	H	L
H	L	L
H	H	H

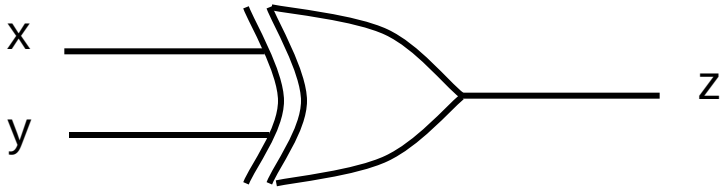
H = High Logic Level
L = Low Logic Level

Aplicaciones de las puertas EXOR

- Suma módulo 2. Tabla de verdad. (se verá en sumadores)
- Cálculo de paridades (par e impar)



- Comparadores (se verá en comparadores magnitud)



La salida **z** se activa si los bits **x** e **y** son diferentes

Teoremas del Algebra de Boole

(T1)	$X + 0 = X$	(T1′)	$X \cdot 1 = X$	(Identities)
(T2)	$X + 1 = 1$	(T2′)	$X \cdot 0 = 0$	(Null elements)
(T3)	$X + X = X$	(T3′)	$X \cdot X = X$	(Idempotency)
(T4)	$(X')' = X$			(Involution)
(T5)	$X + X' = 1$	(T5′)	$X \cdot X' = 0$	(Complements)

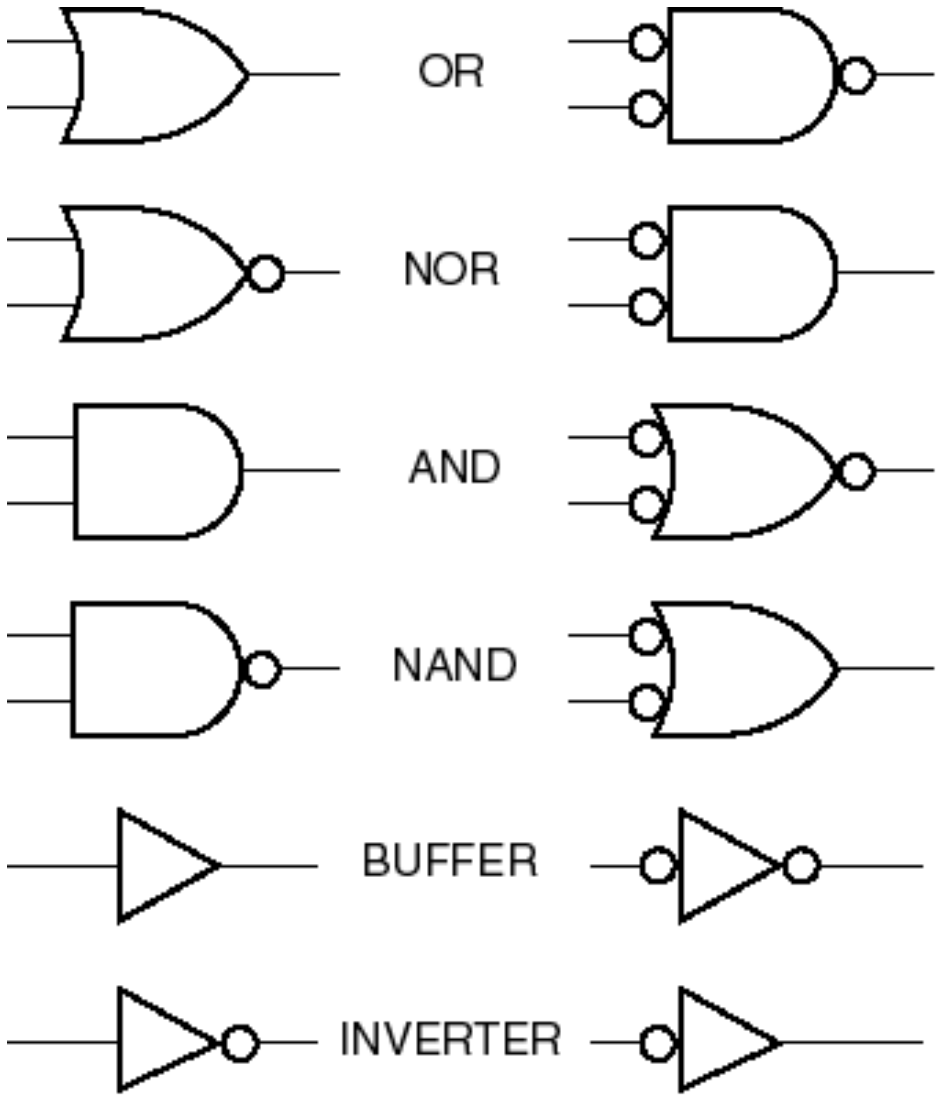
(T6)	$X + Y = Y + X$	(T6′)	$X \cdot Y = Y \cdot X$	(Commutativity)
(T7)	$(X + Y) + Z = X + (Y + Z)$	(T7′)	$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$	(Associativity)
(T8)	$X \cdot Y + X \cdot Z = X \cdot (Y + Z)$	(T8′)	$(X + Y) \cdot (X + Z) = X + Y \cdot Z$	(Distributivity)
(T9)	$X + X \cdot Y = X$	(T9′)	$X \cdot (X + Y) = X$	(Covering)
(T10)	$X \cdot Y + X \cdot Y' = X$	(T10′)	$(X + Y) \cdot (X + Y') = X$	(Combining)
(T11)	$X \cdot Y + X' \cdot Z + Y \cdot Z = X \cdot Y + X' \cdot Z$			(Consensus)
(T11′)	$(X + Y) \cdot (X' + Z) \cdot (Y + Z) = (X + Y) \cdot (X' + Z)$			

Teoremas de n variables

(T12)	$X + X + \cdots + X = X$	(Generalized idempotency)
(T12′)	$X \cdot X \cdot \ \cdots \ \cdot X = X$	
(T13)	$(X_1 \cdot X_2 \cdot \ \cdots \ \cdot X_n)' = X_1' + X_2' + \cdots + X_n'$	(DeMorgan's theorems)
(T13′)	$(X_1 + X_2 + \cdots + X_n)' = X_1' \cdot X_2' \cdot \ \cdots \ \cdot X_n'$	
(T14)	$[F(X_1, X_2, \ldots, X_n, +, \cdot\ )]' = F(X_1', X_2', \ldots, X_n', \cdot, +)$	(Generalized DeMorgan's theorem)
(T15)	$F(X_1, X_2, \ldots, X_n) = X_1 \cdot F(1, X_2, \ldots, X_n) + X_1' \cdot F(0, X_2, \ldots, X_n)$	(Shannon's expansion theorems)
(T15′)	$F(X_1, X_2, \ldots, X_n) = [X_1 + F(0, X_2, \ldots, X_n)] \cdot [X_1' + F(1, X_2, \ldots, X_n)]$	

- Prueba utilizando inducción finita
- Los más importantes son los de teoremas de DeMorgan

Símbolos utilizando DeMorgan



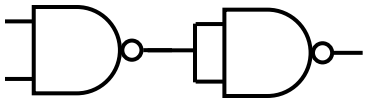
Propiedades interesantes de la puerta NAND

Se puede utilizar la puerta NAND para obtener cualquiera de los tres operadores lógicos básicos (NOT, AND y OR)

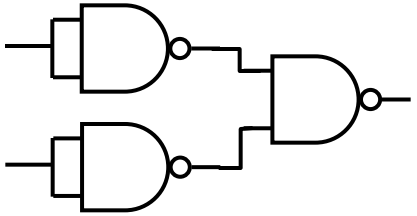
Aplicando el Álgebra de Boole

NAND(a,a)=(aa)’ = a’ = NOT(a)

NAND’(a,b)=(ab)’’ = ab = AND(a,b)



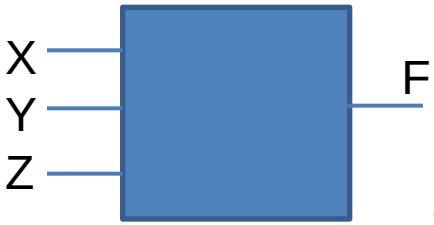
NAND(a’,b’)=(a’b’)’ = a+b = OR(a,b)



También se puede hacer lo mismo con **puertas NOR**. Comprobar (ejercicio).

Representación de funciones lógicas

El objetivo final es implementar un circuito digital que genere una señal de salida, F, en función de una serie de señales de entrada (X, Y y Z, por ejemplo).



La señal de salida F se puede definir de varias maneras

Con su **EXPRESIÓN LÓGICA**: $F = (X \bullet Z) + (Y' \bullet Z) + (X' \bullet Y \bullet Z')$

TABLA de VERDAD

Fila	X	Y	Z	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

SUMA CANÓNICA

$$F = \sum_{X,Y,Z}(1,2,3,5,7)$$

PRODUCTO CANÓNICO

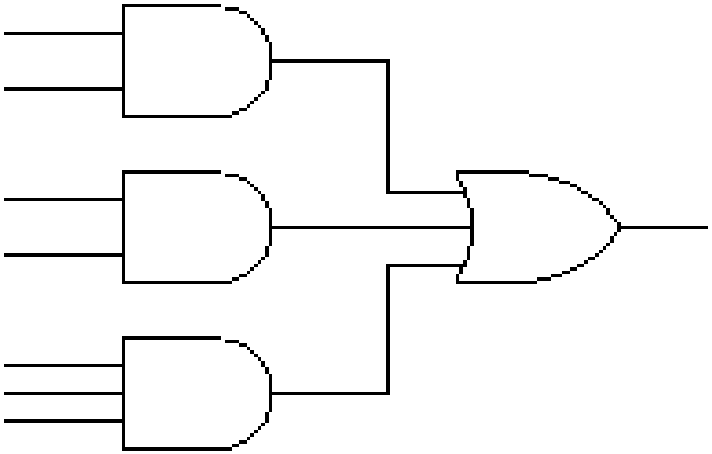
$$F = \prod_{X,Y,Z}(0,4,6)$$

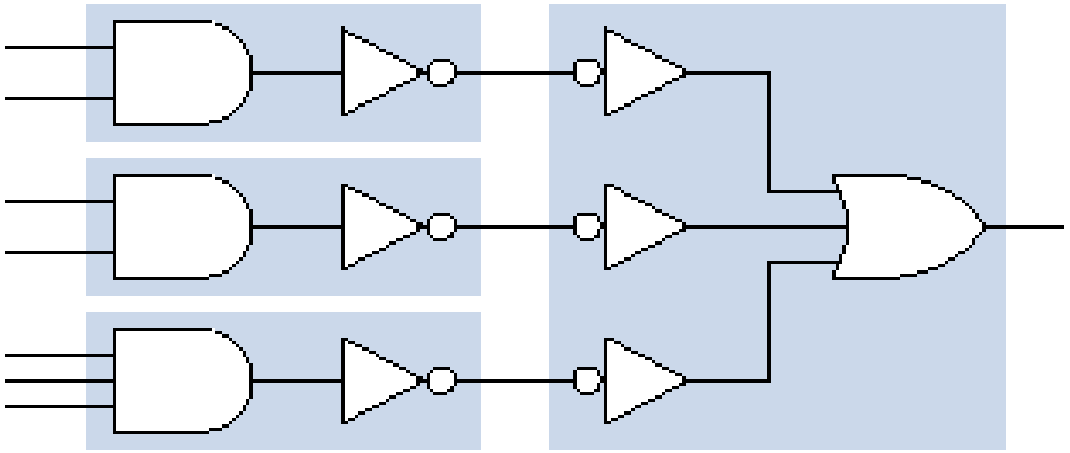
Tabla de verdad: minterms y maxterms

Row	X	Y	Z	F	*Minterm*	*Maxterm*
0	0	0	0	F(0,0,0)	$X' \cdot Y' \cdot Z'$	$X + Y + Z$
1	0	0	1	F(0,0,1)	$X' \cdot Y' \cdot Z$	$X + Y + Z'$
2	0	1	0	F(0,1,0)	$X' \cdot Y \cdot Z'$	$X + Y' + Z$
3	0	1	1	F(0,1,1)	$X' \cdot Y \cdot Z$	$X + Y' + Z'$
4	1	0	0	F(1,0,0)	$X \cdot Y' \cdot Z'$	$X' + Y + Z$
5	1	0	1	F(1,0,1)	$X \cdot Y' \cdot Z$	$X' + Y + Z'$
6	1	1	0	F(1,1,0)	$X \cdot Y \cdot Z'$	$X' + Y' + Z$
7	1	1	1	F(1,1,1)	$X \cdot Y \cdot Z$	$X' + Y' + Z'$

Implementación formal de funciones

- Implementación como suma de productos



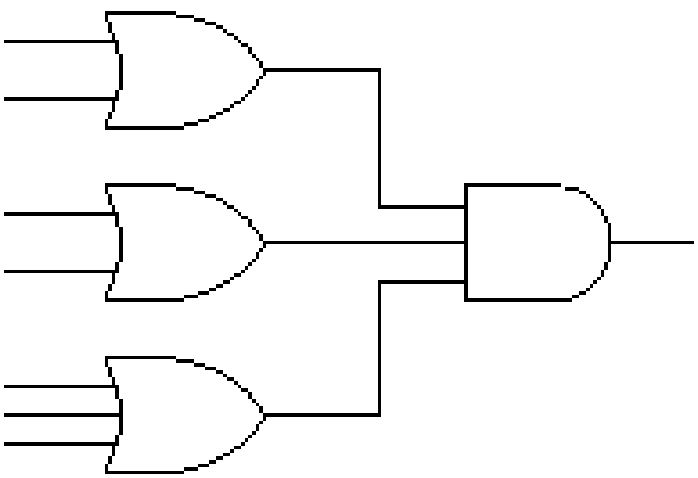


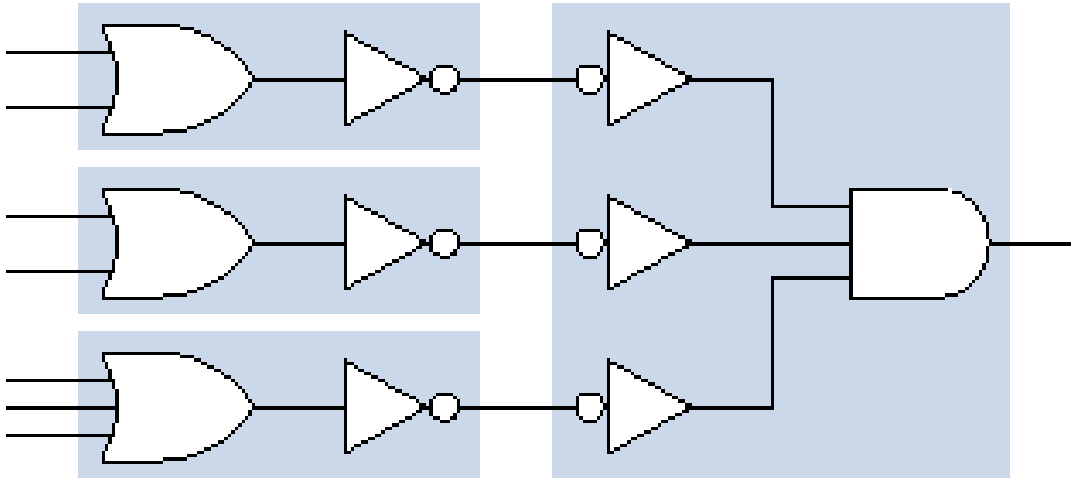
AND-OR

NAND-NAND
(mejor para CMOS)

Implementación formal de funciones

- Implementación como producto de sumas





OR-AND

NOR-NOR
(mejor para CMOS)

Diseño

- Ejemplo: Detector de números primos
  - 4-bits de entrada, $\mathbf{N_3N_2N_1N_0}$,
    que codifican un número en binario
  - 1 bit de salida, **F**,
    que se activa si la entrada corresponde
    a un número primo
- Tabla de Verdad
- Suma Canónica
  (suma de minterms)

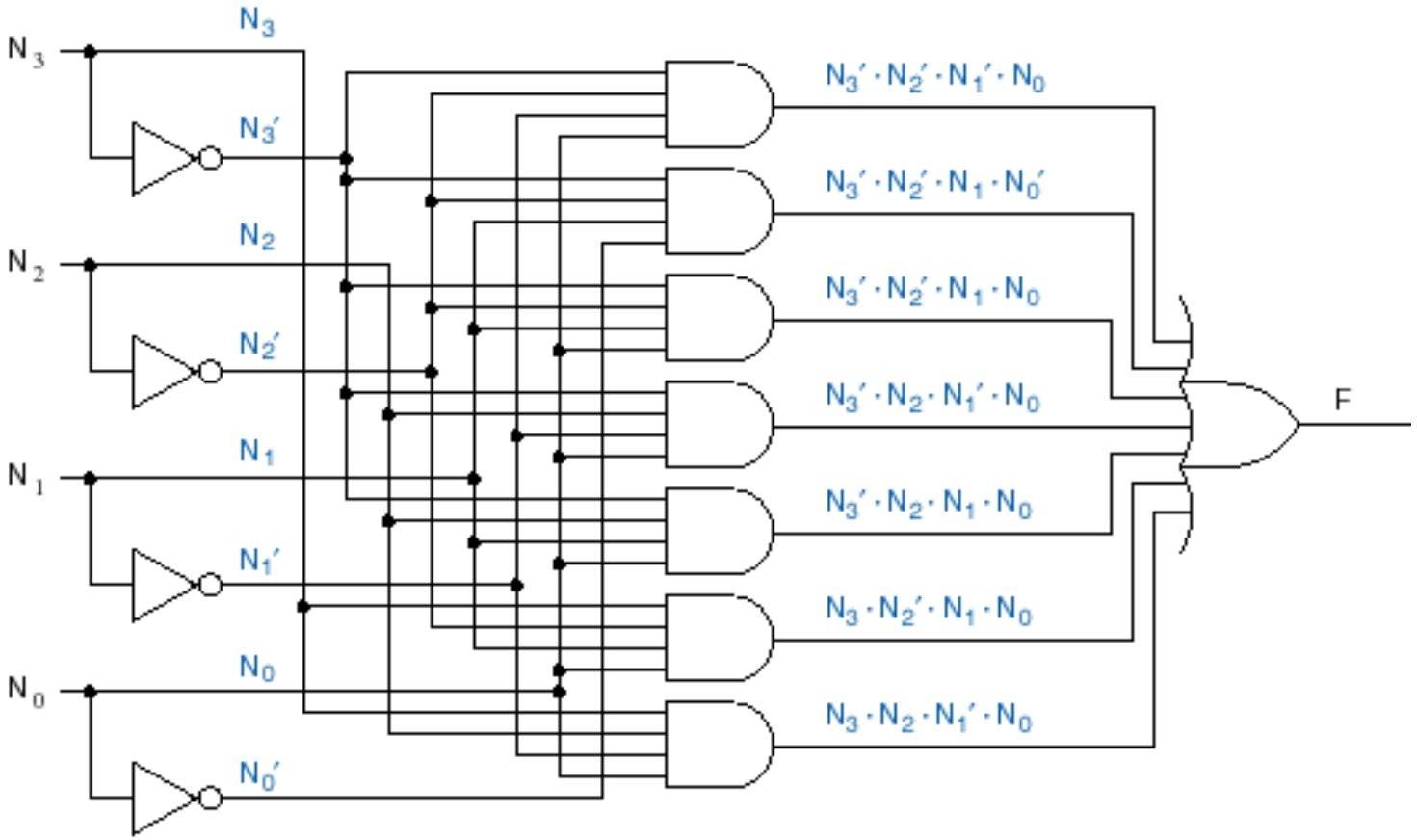
$$F = \Sigma_{N_3,N_2,N_1,N_0}(1,2,3,5,7,11,13)$$

Fila	N_3	N_2	N_1	N_0	F
0	0	0	0	0	0
1	**0**	**0**	**0**	**1**	**1**
2	**0**	**0**	**1**	**0**	**1**
3	**0**	**0**	**1**	**1**	**1**
4	0	1	0	0	0
5	**0**	**1**	**0**	**1**	**1**
6	0	1	1	0	0
7	**0**	**1**	**1**	**1**	**1**
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	**1**	**0**	**1**	**1**	**1**
12	1	1	0	0	0
13	**1**	**1**	**0**	**1**	**1**
14	1	1	1	0	0
15	1	1	1	1	0

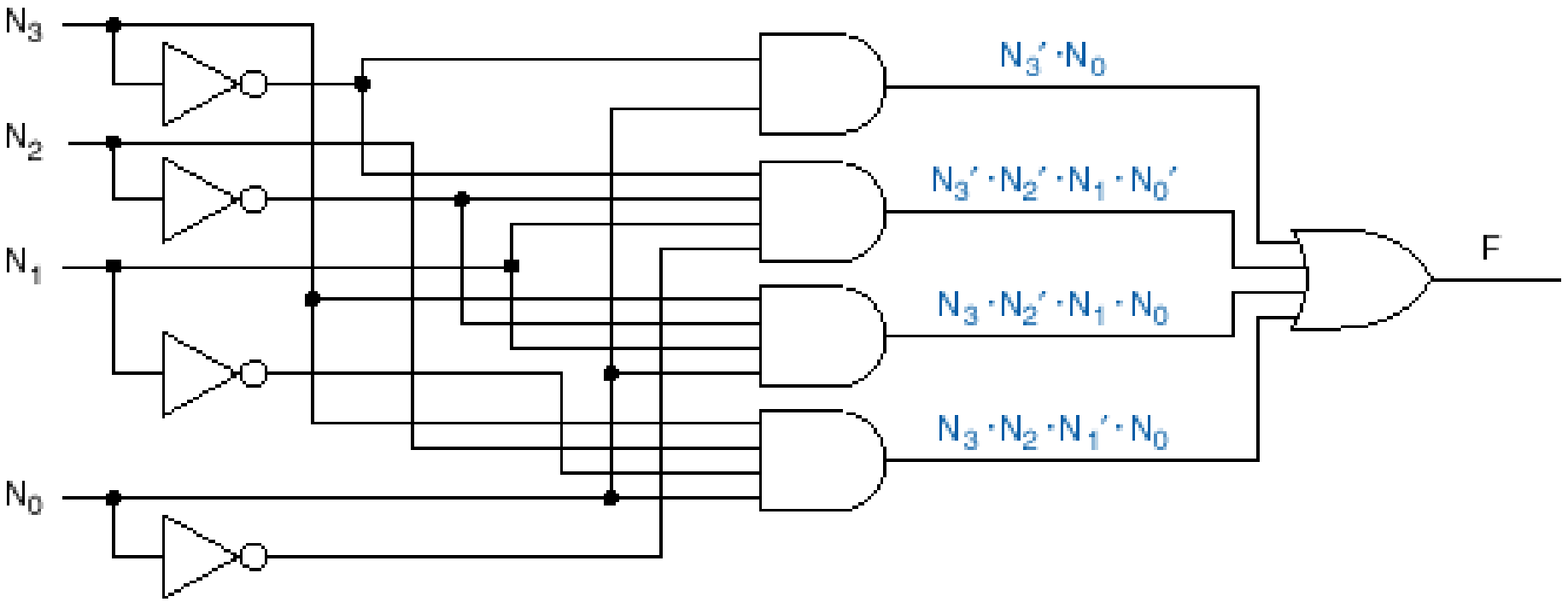
Lista de minterms → suma canónica

$$F = \Sigma_{N_3,N_2,N_1,N_0}(1, 2, 3, 5, 7, 11, 13)$$

$$= N_3' \cdot N_2' \cdot N_1' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 \cdot N_0' + N_3' \cdot N_2' \cdot N_1 \cdot N_0 + N_3' \cdot N_2 \cdot N_1' \cdot N_0 + N_3' \cdot N_2 \cdot N_1 \cdot N_0 + N_3 \cdot N_2' \cdot N_1 \cdot N_0 + N_3 \cdot N_2 \cdot N_1' \cdot N_0$$

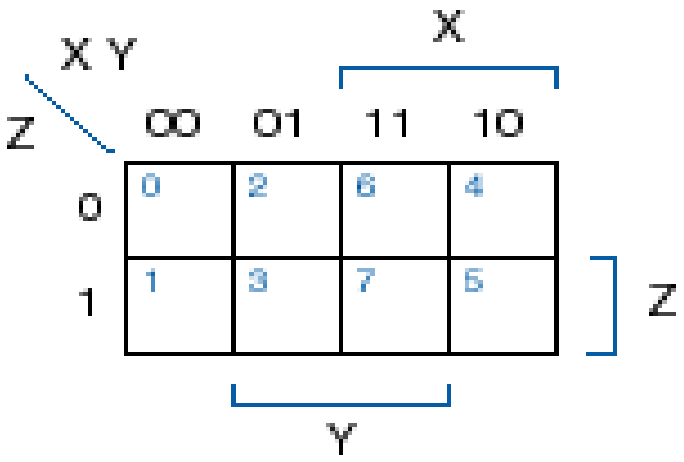


Simplificación con el T10: Ejercicio

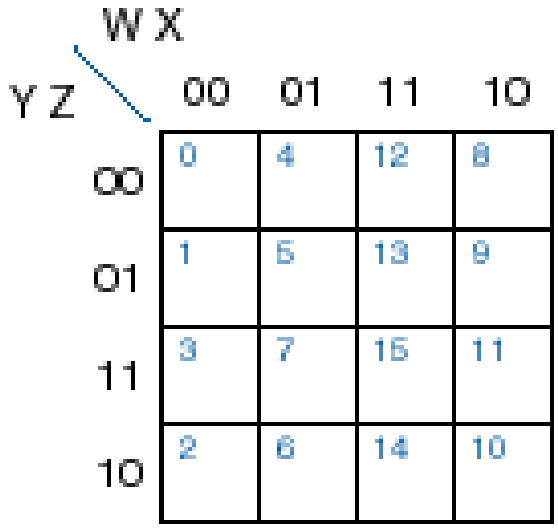


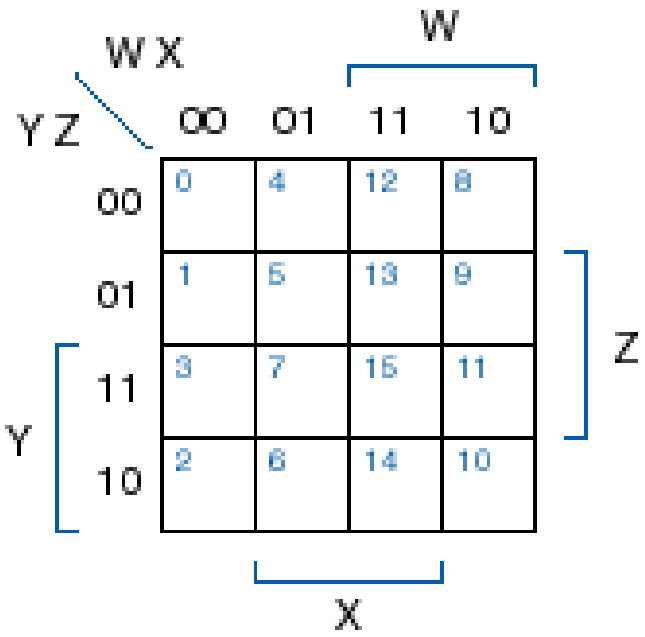
Mapa de Karnaugh: T10

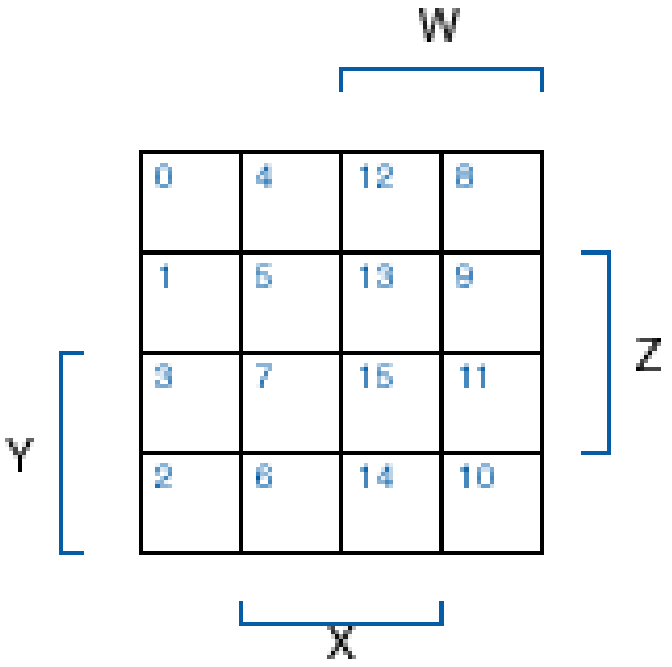
- Permiten realizar una minimización sistemática de la función
- En cualquier mapa de Karnaugh, los valores solamente pueden ser “1” y “0”.
- En caso de que existan “X”, habrá que asignarles un valor “1” ó “0” según el criterio de máxima minimización (rectángulos más grandes)



Mapas de Karnaugh: 4 variables





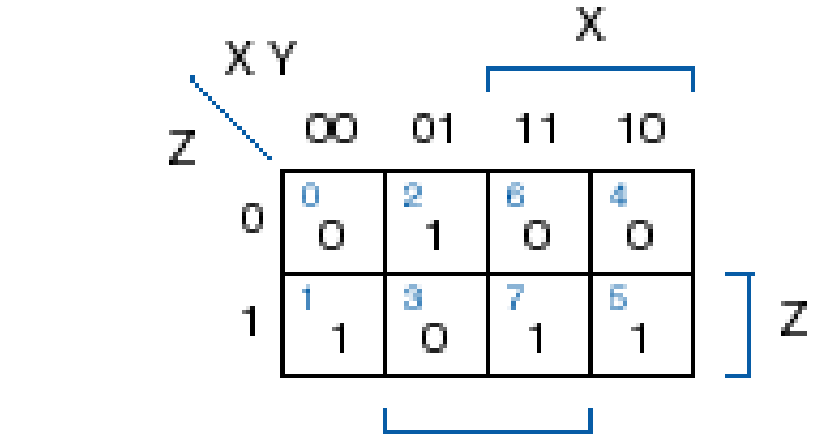


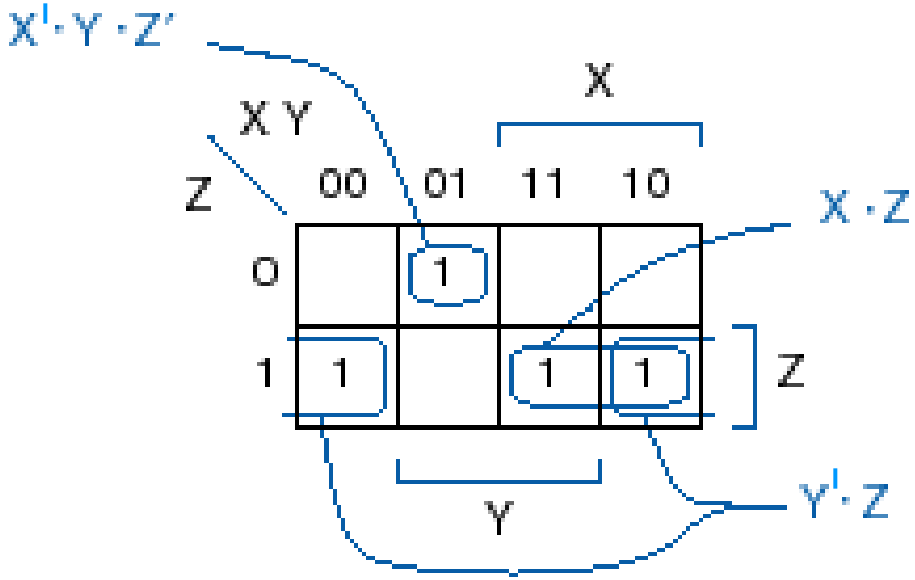
Uso de los mapas de Karnaugh

- Dibujar los “1” correspondientes al los minterms de la función
- Encuadrar el mayor número posible de grupos rectangulares de “1” hasta que no quede ninguno libre
  - El número de “1” en cada rectángulo tiene que ser potencia de 2
  - Está permitido pasar los bordes del mapa (en vertical y horizontal)
- Para cada uno de los rectángulos, obtener el término producto
  - Si la variable es “1” -> incluir la variable
  - Si la variable es “0” -> incluir la variable complementada
  - Si la variable vale a la vez “0” y “1” -> no incluir la variable
- Los rectángulos marcados y sus correspondientes términos producto se llaman “principales implicados”
- Esta minimización permite mínimo número de puertas y de entradas por puerta

Ejemplo: $F = \Sigma_{X,Y,Z}(1,2,5,7)$

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

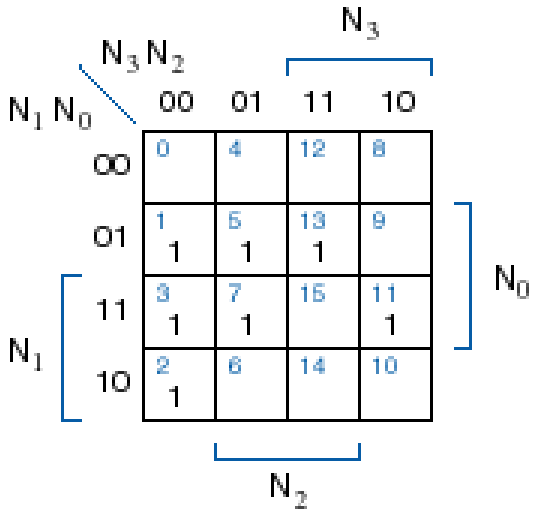




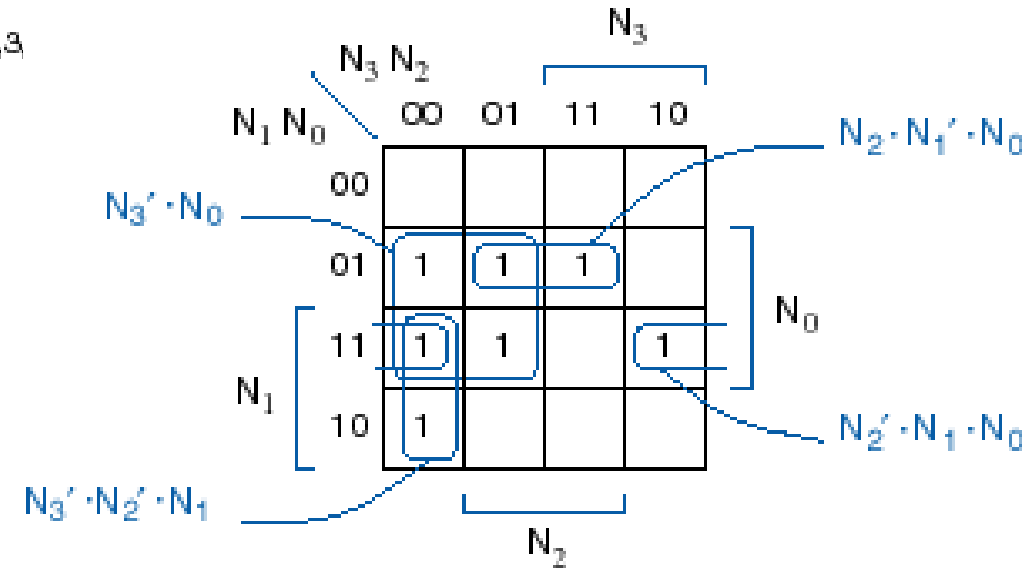
F = (X • Z) + (Y´ • Z) + (X´ • Y • Z´)

Ejemplo: Detector de números primos I

N3	N2	N1	N0	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0



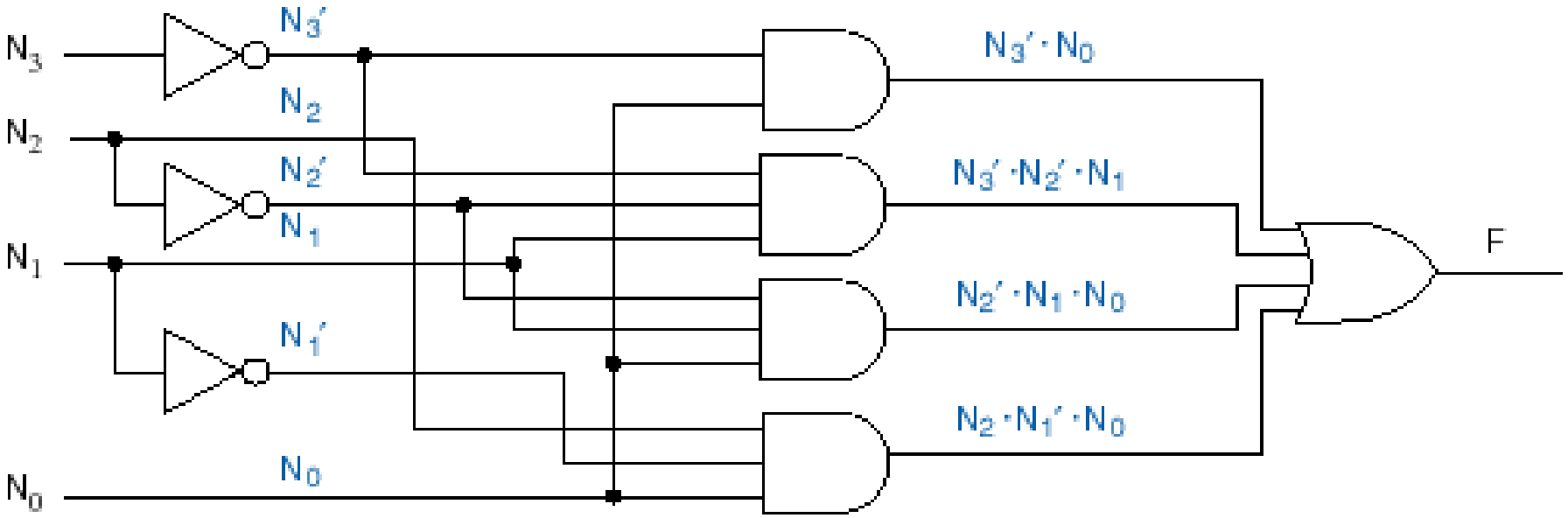
$F = \Sigma_{N3,N2,N1,N0}(1,2,3$



$$F = N_3' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 + N_2' \cdot N_1 \cdot N_0 + N_2 \cdot N_1' \cdot N_0$$

Ejemplo: Detector de números primos II

- Este circuito tiene tres entradas de puerta menos que en la solución algebraica anterior



Preguntas