

Ingeniería en Computación - Curso 2023

Clase 3.

Representaciones en Punto Fijo – Álgebra de Boole – Compuertas Lógicas – Circuitos Lógicos

## En esta clase:

- Concepto de Representación en Punto Fijo
- Precisión – Rango
- Operaciones en Punto Fijo

# Representacion de Punto Fijo

- En los sistemas de numeración posicionales de cualquier base, se pueden diferenciar los valores fraccionarios de los enteros mediante el uso convencional de un simbolo que los separa (punto o coma decimal)
- En el caso binario por ejemplo el número representado con 9 bits en la siguiente figura sería:

$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	●	$2^{-1}$	$2^{-2}$	$2^{-3}$	$2^{-4}$
1	0	0	1	0	●	1	1	1	1

$$2^4 + 2^1 + 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} = 16 + 2 + 0.5 + 0.25 + 0.125 + 0.0625 = 18.9375$$

- ¿Cuál sería el número más grande y el más chico (omitiendo el cero) que puede representarse en este caso?
  - Es fácil ver que el mayor sería 31.9375 y el menor 0.0625
  - tambien que la *resolución* o sea el incremento mínimo entre un valor y el siguiente también es  $0.0625 = 2^{-4}$

# Representacion de Punto Fijo

- Si ahora cambiamos la posición del punto, podemos obtener otro rango de representaciones y resolución.

•	$2^{-1}$	$2^{-2}$	$2^{-3}$	$2^{-4}$	$2^{-5}$	$2^{-6}$	$2^{-7}$	$2^{-8}$	$2^{-9}$
•	1	0	0	1	0	1	1	1	1

$$2^{-1}+2^{-4}+2^{-6}+2^{-7}+2^{-8}+2^{-9} = 1/2+1/16+1/64+1/128+1/256+1/512 = 0.591796875$$

- ¿Y ahora cuál sería el número más grande y el más chico (omitiendo el cero) que puede representarse?
  - Es fácil ver que el mayor sería 0.998046875 y el menor 0.001953125
  - tambien que la **resolución** o sea el incremento mínimo entre un valor y el siguiente también es  $0.001953125=2^{-9}$

# Representacion de Punto Fijo

- Por último situemos el punto en otra posición. Veamos que rango de valores podemos representar.

$2^8$	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	•
1	0	0	1	0	1	1	1	1	•

$$2^8 + 2^5 + 2^3 + 2^2 + 2^1 + 2^0 = 256 + 32 + 8 + 4 + 2 + 1 = 303$$

- ¿Y ahora cuál sería el número más grande y el más chico (omitiendo el cero) que puede representarse?
  - Es fácil ver que el mayor sería 511 y el menor 1.
  - También que la **resolución** o sea el incremento mínimo entre un valor y el siguiente también es  $1=2^0$ .

# Representacion de Punto Fijo

- Representación de valores grandes

$2^{12}$	$2^{11}$	$2^{10}$	$2^9$	$2^8$	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	•
1	0	0	1	0	1	1	1	1	0	0	0	0	•

Aqui estamos usando los mismos 9 bits para representar el número, pero asumimos que hay 4 digitos menos significativos “ficticios” que no utilizamos.

- En el ejemplo anterior teníamos:

$$2^8+2^5+2^3+2^2+2^1+2^0 = 256+32+8+4+2+1 = 303$$

y en este tenemos

$$2^{12}+2^9+2^7+2^6+2^5+2^4 = 4096+512+128+64+32+16 = 4848$$

- ¿Y ahora cuál sería el número más grande y el más chico (omitiendo el cero) que puede representarse?
  - Es fácil ver que el mayor sería 8176 y el menor 16.
  - Tambien que la **resolución** o sea el incremento mínimo entre un valor y el siguiente también es  $16=2^4$ .
- Se puede verificar que los valores representados de esta manera son los mismos que se obtenían en el ejemplo anterior, pero multiplicados por 16 o sea  $2^4$ .

# Representacion de Punto Fijo

- Si hacemos el cociente entre el valor mas alto y el mas pequeño que se pueden representar en todos los casos vistos podemos encontrar que:

$$31.9375 / 0.0625 = (2^5 - 2^{-4}) / 2^{-4} = 2^9 - 1 = 511$$

$$0.998046875 / 0.001953125 = (2^0 - 2^{-9}) / 2^{-9} = 2^9 - 1 = 511$$

$$511 / 1 = (2^9 - 2^0) / 2^0 = 2^9 - 1 = 511$$

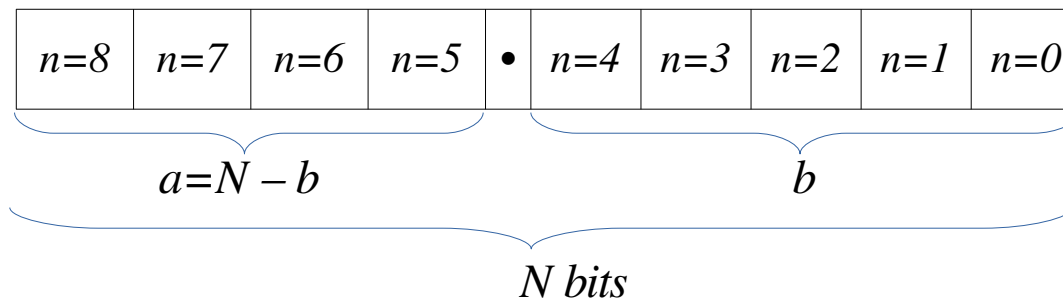
$$8176 / 16 = (2^{13} - 2^4) / 2^4 = 2^4 (2^9 - 1) / 2^4 = 511$$

- Podemos observar que:
  - Con la misma cantidad de bits, podemos representar distintos rangos de valores numéricos dependiendo de donde situemos el punto decimal que separa la parte fraccionaria de la parte entera.
  - La cantidad de valores que pueden representarse es la misma.
  - La resolución corresponde al minimo valor representable distinto que cero y esta dada por el valor asignado al LSB.
  - Esta idea se puede extender para representar valores muy pequeños o muy grandes
  - Puede interpretarse que la ubicación del punto con respecto al LSB puede modificarse multiplicando por una *escala*  $2^M$ , donde  $M$  es un entero.

# Representacion de Punto Fijo

Vamos a formalizar esto para representaciones de números positivos

- Sea una representación de punto fijo de  $N$  bits, que llamaremos  $U(a, b)$ .  
Con  $b \leq N$ ;  $a = N - b$



- Esto se puede interpretar como ubicar el punto fraccionario entre los bits  $n = b - 1$  y  $n = b$  representando al conjunto de números:

$$P = \{p / 2^b \mid 0 \leq p \leq 2^N - 1, p \in \mathbb{Z}\}.$$

- El bit  $n$  tiene un peso de  $2^n / 2^b = 2^{n-b}$  y la resolución es el menor valor representable  $2^{-b}$
- El valor de un numero  $x$  en particular está dado por  $x = (1/2^b) \sum_{n=0}^{N-1} 2^n x_n$  dónde  $x_n$  representa el enésimo bit de  $x$ .
- El rango de la representación está dado por :  $0 \leq x \leq (2^N - 1) / 2^b = 2^{N-b} - 2^{-b} = 2^a - 2^{-b}$

$$\text{Rango: } 0 \leq x \leq 2^a - 2^{-b}$$

$$\text{Resolución: } 2^{-b}$$



# Representacion de Punto Fijo: Ca1 y Ca2

- Si consideremos  $U(N,0)$ ,  $a=N$ ,  $b=0$ , nos queda que la resolución es 1 y el rango representable será  $0 \leq x \leq 2^N - 1$
- Esto coincide con la representación binaria natural en N bits, y sabemos que si queremos representar números con signo, debemos utilizar el MSB para esta finalidad.
- Se puede verificar que si definimos  $\tilde{x} = Ca1(x)$ , entonces para  $U(N,0)$  tendremos  $\tilde{x} = 2^N - 1 - x$ .
- Si definimos  $\hat{x} = Ca2(x)$ , entonces  $\hat{x} = \tilde{x} + 1 = 2^N - x$
- Ahora si queremos representar números con signo, y el MSB nos da el signo, entonces podemos decir que el rango de números positivos, lo vamos a representar con N-1 bits, y el MSB será cero.

# Representacion de Punto Fijo: Ca1 y Ca2

- Para representar los números negativos, usaremos Ca2, y los valores que obtendremos es ese caso estarán dados por  $\hat{x} = \tilde{x} + 1 = 2^N - x$
- El rango de números que podemos representar estará dado por:
- $P = \{p \mid -2^{N-1} \leq p \leq (2^{N-1} - 1), p \in \mathbb{Z}\}$
- y el valor obtenido para una representación dada en N bits será:

$$x = \left[ -2^{N-1} x_{N-1} + \sum_{n=0}^{N-2} 2^n x_n \right]$$

- Puede verse que si el MSB (ubicado en la posición  $N-1$ ) es cero, la representación de  $x$  es la de un número positivo de  $N-1$  bits.
- Si el MSB es 1, podemos comprobar que se está realizando la operación de Ca2 sobre  $x$  y queda representado un número negativo.

# Representacion de Punto Fijo con signo

- Si combinamos esta representación con lo que vimos para punto fijo positivo , estaríamos pasando de  $U(N,0)$  a una representación con signo, que llamaremos  $Q(N-1,0)$ .
- Si ahora tomamos como antes  $a$  bits para la parte entera y  $b$  bits para la parte fraccionaria, generalizamos la representación a  $Q(a,b)$  en la cual  $a=N-b-1$  y el conjunto de números que pueden representarse será:

$$P = \{ p / 2^b \mid -2^{N-1} \leq p \leq (2^{N-1} - 1), p \in \mathbb{Z} \}$$

- El valor  $x$  representado por una  $Q(a,b)$  de  $N$  bits será entonces:

$$x = \left( \frac{1}{2^b} \right) \left[ -2^{N-1} x_{N-1} + \sum_{n=0}^{N-2} 2^n x_n \right]$$

- El factor  $\left( \frac{1}{2^b} \right)$  se llama **Escala** y el rango de representación será:  
$$-2^{N-1-b} \leq x \leq +2^{N-1-b} - 1/2^b$$

- Nótese que  $a+b = N-1$  porque se utiliza el MSB para el signo

# Representacion de Punto Fijo con signo

- *Ejemplo 1: ¿Qué número representa 01011001 en  $Q(3,4)$ ?*
  - Podemos ver que el número es positivo, entonces el valor corresponde a la representación entera del número 89 y debe multiplicarse por la **escala** dada por  $1/2^4 = 1/16$  se obtiene 5,5625
- *Ejemplo 2: ¿Qué número representa 10010110 en  $Q(2,5)$ ?*
  - El número es negativo (MSB=1) y la escala es  $1/2^5 = 1/32$ , entonces el número representado es  $-106 / 32 = -3,3125$

# Operaciones

- Largo sin signo
  - El número de bits para representar  $U(a,b)$  es  $a+b$
- Largo con signo
  - El número de bits para representar  $Q(a,b)$  es  $a+b+1$
- Rango sin signo.
  - El rango de  $U(a,b)$  es  $0 \leq x \leq 2^a - 2^{-b}$ .
- Rango con signo
  - el rango de  $Q(a,b)$  es  $-2^a \leq x \leq 2^a - 2^{-b}$ .
- Cambio de escala:
  - Si tengo una representación de un número  $x$  en  $Q(a,b)$  donde la escala es  $1/2^b$  y quiero pasarlo a otra representación  $Q_1(a_1, b_1)$  con la misma cantidad de bits  $N$ , se debe cambiar la escala desplazando los bits de la representación hacia la izquierda o hacia la derecha según  $b_1$  sea mayor o menor que  $b$  respectivamente. Esto equivale a multiplicar o dividir por 2 respectivamente.
  - Debe analizarse si no se pierden bits más significativos al realizar esta operación
- Operandos de adición
  - Dos números binarios deben tener igual escala para poder sumarlos.
  - Esto es, la operación  $X(c,d) + Y(e,f)$  es solamente válida si  $X = Y$  (ya sea  $Q$  o  $U$ ) y además se tiene que cumplir que  $c=e$  y  $d=f$ .

- Podemos representar números con parte entera y parte fraccionaria, con o sin signo usando una representación en punto fijo de  $N$  bits.
- De los  $N$  bits,  $b$  bits representan la parte fraccionaria y  $a=N-b$  se utilizan para representar la parte entera
- La representación del signo está a cargo de los bits de parte entera.
- la cantidad de valores que pueden representarse es igual a la cantidad de valores enteros que pueden representarse con  $N$  bits.
- La representación en Punto Fijo utiliza una constante llamada Escala que es  $1/2^b$  y que multiplica a todo el número.
- La representación en punto fijo, también puede utilizarse para representar números grandes, en ese caso la escala es  $2^b$

# Parte II Álgebra de Boole – Puertas Lógicas

- Variables booleanas o lógicas
- Tablas de verdad. (TdV)
- Ejemplos de TdV para puertas AND, NAND, OR, y NOR, y el circuito inversor NOT.
- Expresiones booleanas para puertas lógicas.
- Teoremas de Boole y DeMorgan para simplificar expresiones lógicas.

# Variables booleanas o lógicas

- **Constantes y Variables Booleanas:**
  - Solamente puede adoptar dos valores
  - Muchas veces se utilizan distintas palabras como sinónimos de estos valores. En la tabla se muestran los más usuales

0 lógico	1 lógico
Falso	Verdadero
Apagado	Encendido
Bajo	Alto
No	Si
Interruptor abierto	Interruptor cerrado



# Operaciones lógicas

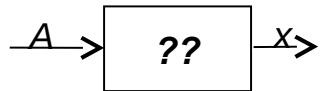
- Operaciones básicas
  - Conjunción: AND, Y, intersección.  $(A \bullet B)$
  - Disyunción inclusiva : OR, Ó, unión.  $(A+B)$
  - Negación: Not, No, complemento.  $(\neg A)$  o también  $\overline{A}$
- Minterm o minitérmino
  - Expresión con una o más variables no repetidas, directas o negadas, relacionadas entre sí mediante conjunción
  - Ejemplos:  $\overline{A}.B.C$        $x.\overline{y}.z$
- Maxterm o maxitérmino
  - Expresión con una o más variables no repetidas, directas o negadas, relacionadas entre sí mediante disyunción inclusiva
  - Ejemplos:  $(\overline{A}+B+C)$        $(x+\overline{y}+z)$

# Tablas de Verdad (TdV)

- La Tabla de Verdad describe la relación entre entradas y salidas.
- Los valores posibles de las entradas y salidas son valores binarios (0 o 1).
- El número de entradas define la cantidad de combinaciones posibles a considerar
- $N$  entradas  $\rightarrow 2^N$  combinaciones
- Deben listarse todas las combinaciones posibles de entradas y los valores que toma la salida para cada combinación.
- Esto puede interpretarse como que la TdV, define una *función lógica* cuyo dominio son las combinaciones posibles de entradas y su imagen son los valores que adopta la salida en cada caso

# Tablas de Verdad 1, 2, 3 y 4 variables

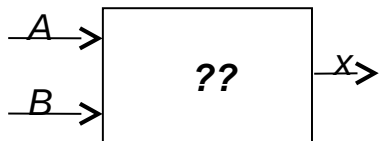
A	x
0	0
1	1



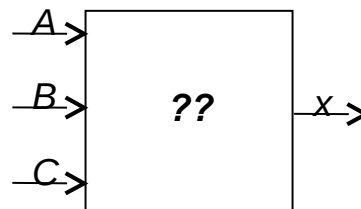
1 entrada → 2 combinaciones

A	B	C	x
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	0
1	1	1	0

A	B	x
0	0	0
0	1	0
1	0	1
1	1	0

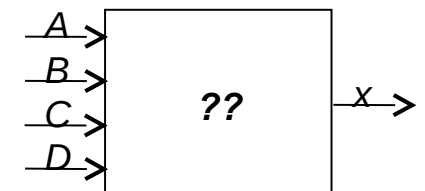


2 entradas → 4 combinaciones



3 entradas → 8 combinaciones

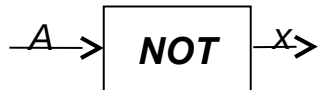
A	B	C	D	x
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



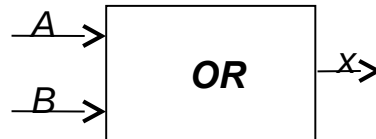
4 entradas → 16 combinaciones

# Tablas de verdad de funciones lógicas

A	x
0	1
1	0



A	B	x
0	0	0
0	1	1
1	0	1
1	1	1



A	B	x
0	0	0
0	1	0
1	0	0
1	1	1



A	B	x
0	0	0
0	1	1
1	0	1
1	1	0



- De una variable: NOT
  - Invierte el valor lógico de la entrada
- De dos variables OR, AND, XOR
  - OR da una salida Verdadera, cuando cualquiera de sus entradas es verdadera
  - AND da una salida Verdadera cuando todas sus entradas son Verdaderas
  - XOR da una salida Verdadera si sus entradas son distintas.

# Funciones y compuertas

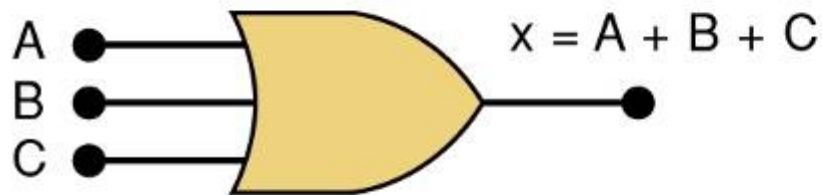
- Las primeras implementaciones de funciones lógicas digitales recibieron el nombre de compuertas lógicas (*logical gates*).
- La Compuerta NOT (También llamada INVERSOR) tiene una sola entrada.
  - NOT: la salida es opuesta a la entrada.
- Los tipos más comunes que poseen más de una entrada son OR, NOR, AND, NAND, XOR y XNOR.
  - OR: la salida es verdadera si cualquier entrada es verdadera.
  - NOR: la salida es la opuesta a la de la OR.
  - AND: la salida es verdadera si todas las entradas son verdaderas.
  - NAND: la salida es la opuesta a la AND.
  - XOR: (Or Exclusiva o disyunción exclusiva) la salida es verdadera si un número impar de entradas es verdadera.
  - XNOR: XOR Negada, la salida es verdadera si un número par de entradas es verdadera.

# Funciones y compuertas

- Las funciones lógicas pueden denotarse de distintas maneras, hemos visto la TdV y los nombres de algunas de ellas. También pueden representarse de manera gráfica con los símbolos de compuerta que veremos más adelante o con una notación de tipo algebraico que permite operar matemáticamente (álgebra de boole)
- Función NOT: si la entrada es  $x$  la salida es  $\bar{x}$ . Como es difícil escribir la barra sobre la variable, se suele usar otra notación  $\bar{x} = \backslash x$ .
- Función AND: si las entradas son  $A$  y  $B$ , y la salida es  $x$ , se escribe como si fuera un producto  $x = A \cdot B = AB$
- Función OR: si las entradas son  $A$  y  $B$ , y la salida es  $x$ , se escribe como si fuera una suma:  $x = A + B$
- XOR: si las entradas son  $A$  y  $B$ , y la salida es  $x$ , se escribe como se muestra a continuación:  $x = A \oplus B$
- NAND: si las entradas son  $A$  y  $B$ , y la salida es  $x$ , se escribe como se muestra a continuación:  $x = \overline{AB} = \backslash (AB)$
- NOR: si las entradas son  $A$  y  $B$ , y la salida es  $x$ , se escribe como se muestra a continuación:  $x = \overline{A + B} = \backslash (A + B)$
- XNOR: si las entradas son  $A$  y  $B$ , y la salida es  $x$ , se escribe como se muestra a continuación:  $x = \overline{A \oplus B} = \backslash (A \oplus B)$

# Compuerta OR

## Compuerta OR de tres entradas, Función y Tabla de Verdad

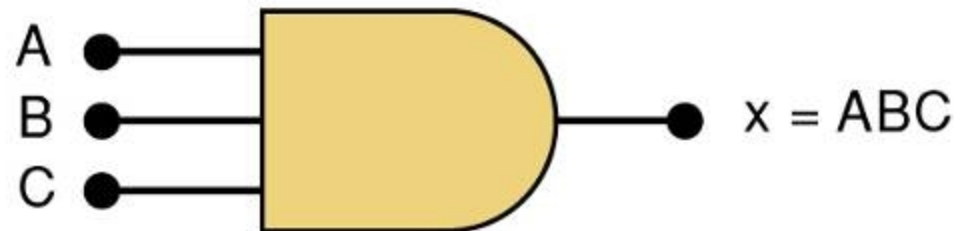


A	B	C	$x = A + B + C$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

# Compuerta AND

## Compuerta AND de tres entradas, Función y Tabla de Verdad

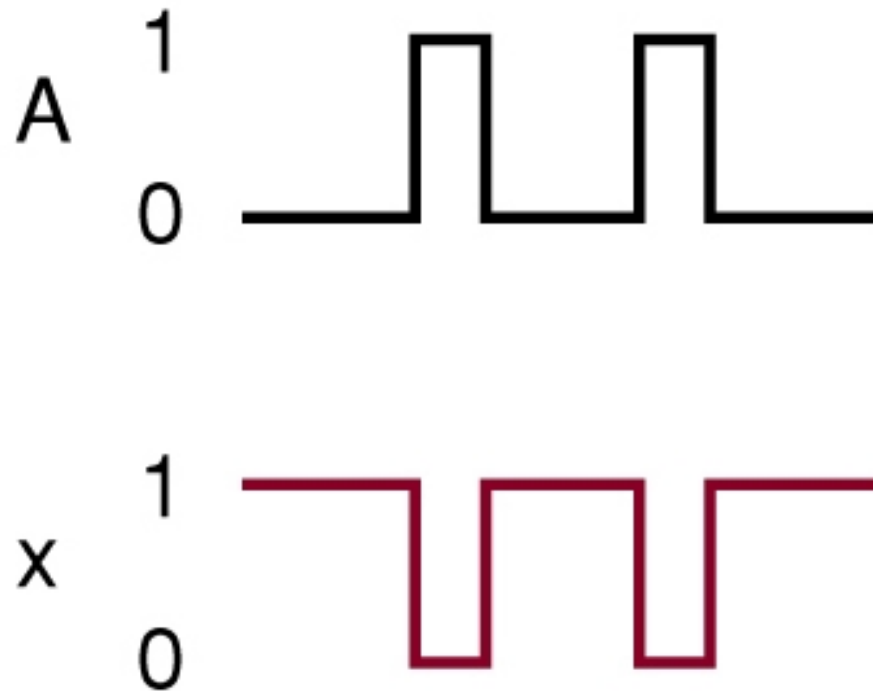
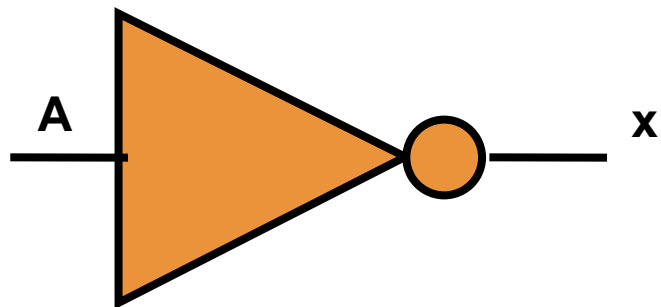
A	B	C	$x = ABC$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1





# Compuerta Not o Inversor

**El inversor genera una salida que es la opuesta de la entrada para todos los puntos de la señal de entrada. La operación se llama también NOT o complementación.**



**Siempre que la entrada sea = 0, salida = 1,  
y viceversa.**

# Equivalencia entre compuertas y operaciones booleanas

## Sumario de reglas para OR, AND y NOT

***OR***

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 1$$

***AND***

$$0 \cdot 0 = 0$$

$$0 \cdot 1 = 0$$

$$1 \cdot 0 = 0$$

$$1 \cdot 1 = 1$$

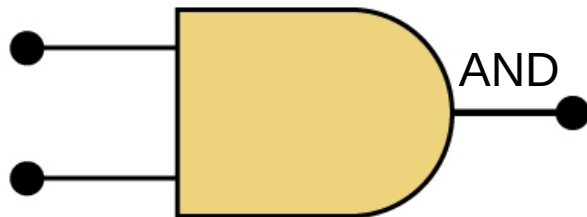
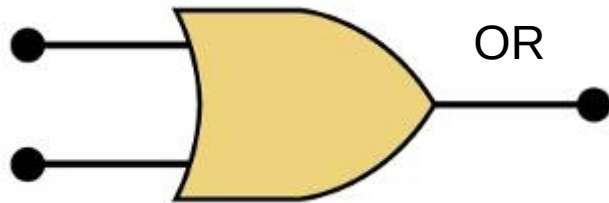
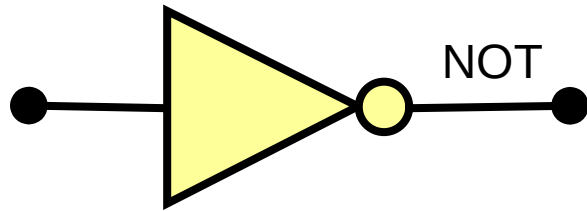
***NOT***

$$\overline{0} = 1$$

$$\overline{1} = 0$$

**Cualquier operación booleana puede escribirse mediante estas tres operaciones elementales**

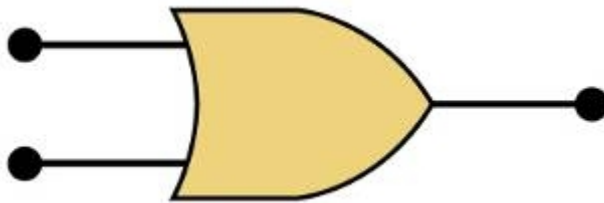
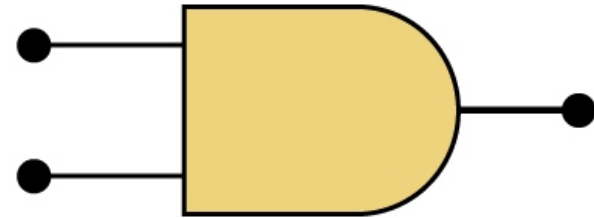
# Equivalencia entre compuertas y operaciones booleanas



**Como estas compuertas implementan las operaciones elementales, con ellas se puede implementar cualquier operación booleana.**

# Equivalencia entre compuertas y operaciones booleanas

El símbolo AND en un circuito lógico dice que la salida va a ir a HIGH **sólo** si **todas** las entradas son HIGH



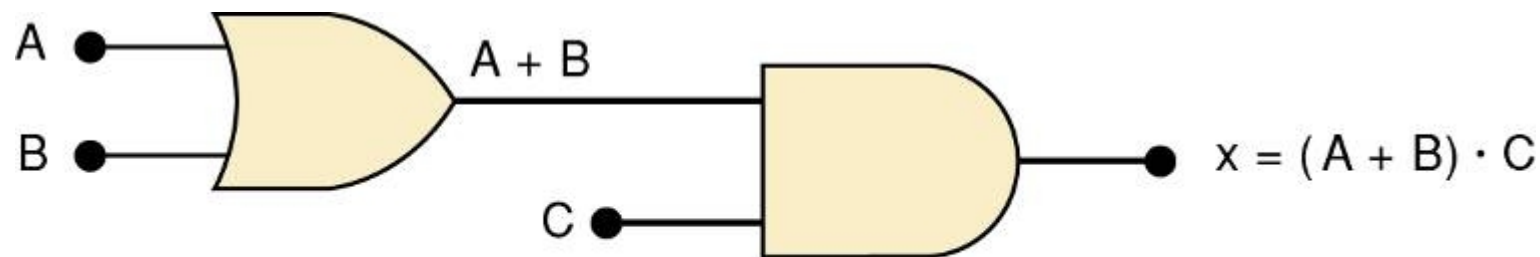
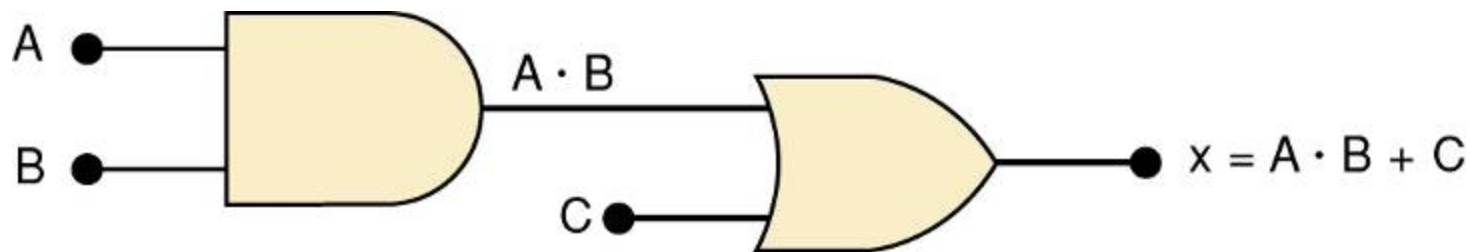
El símbolo OR dice que la salida va a ir a HIGH cuando **cualquier** entrada sea HIGH

# Operaciones y Precedencia

- En una expresión booleana, se analiza de izquierda a derecha.
- La operación AND ( $\bullet$ ) tiene precedencia sobre la operación OR ( $+$ ).
- Como en la multiplicación, la operación AND puede tomarse como implícita  $ABC=A\bullet B\bullet C$
- Si se quiere explicitar o modificar la precedencia puede utilizarse paréntesis:
  - $A+BC=A+(BC)$
  - $x=(A+B)C$  es distinto que  $x=A+BC$
- La operación de negación tiene la mayor precedencia  
 $\neg AB = (\neg A)\bullet B$

# Ejemplos: Análisis

- Determinar cual es la función lógica que implementa un circuito dado



- También se procede de izquierda a derecha y se va operando con los resultados parciales como se muestra en las figuras.

# Ejemplos: Análisis

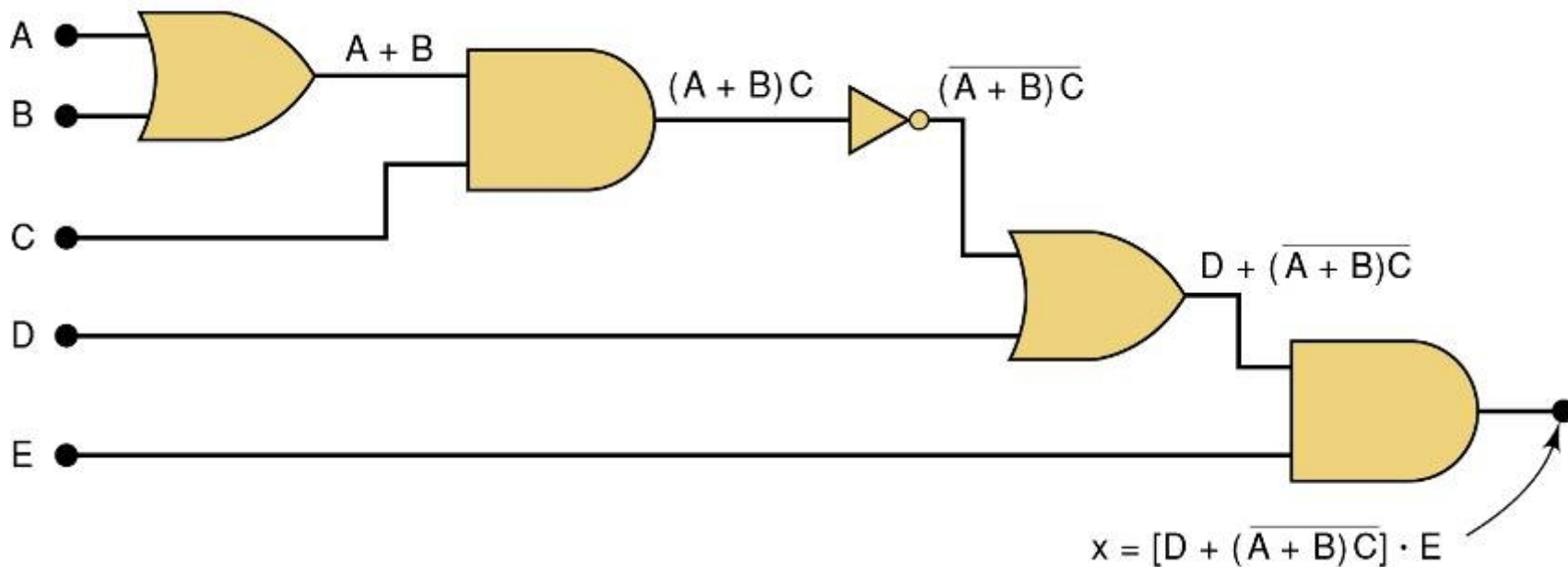
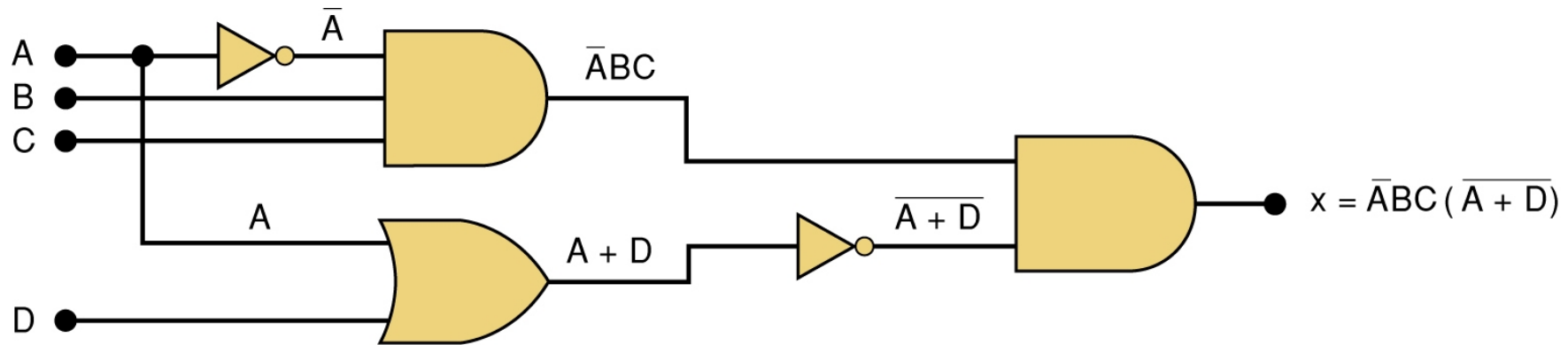
- Determinaremos las tablas de verdad de ambos circuitos.

$x=A \cdot B + C$			
A	B	C	x
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$x=(A+B) \cdot C$			
A	B	C	x
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	0
1	1	1	1

Se ve claramente que son distintas

# Ejemplos. Se animan a hacer la TdV?





# Evaluar expresiones booleanas

- **Reglas para la evaluación de expresiones booleanas:**
  - Realizar todas las inversiones de términos simples.
  - Realizar todas las operaciones dentro de paréntesis.
  - Realizar las operaciones AND antes que las OR a menos que los paréntesis indiquen otra cosa.
  - Si una expresión tiene una barra sobre ella, realizar todas las operaciones dentro de la expresión y luego invertir el resultado.

# Teoremas de Boole (Propiedades)

## Una Variable

1.  $x \cdot 0 = 0$

2.  $x \cdot 1 = x$

3.  $x \cdot x = x$

4.  $x \cdot \overline{x} = 0$

5.  $x + 0 = x$

6.  $x + 1 = 1$

7.  $x + x = x$

8.  $x + \overline{x} = 1$

## Dos Variables

9.  $x + y = y + x$  (conmutativa)

10.  $x \cdot y = y \cdot x$  (conmutativa)

11.  $x + (y + z) = (x + y) + z = x + y + z$  (asociativa)

12.  $x \cdot (y \cdot z) = (x \cdot y) \cdot z = x \cdot y \cdot z = xyz$  (asociativa)

13. a)  $x(y + z) = xy + xz$  (distributiva)

b)  $(w + x)(y + z) = wy + wz + xy + xz$  (distributiva)

14.  $x + xy = x$

15. a)  $x + \overline{x}y = x + y$

b)  $\overline{x} + xy = \overline{x} + y$

# Teoremas de De Morgan

$$\overline{x + y} = \overline{x} \cdot \overline{y}$$
$$\overline{x \cdot y} = \overline{x} + \overline{y}$$

Son muy útiles para simplificar expresiones booleanas.  
Puede demostrarse fácilmente que además:

$$\overline{x + y + z} = \overline{x} \cdot \overline{y} \cdot \overline{z}$$

$$\overline{\overline{x} \cdot \overline{y} \cdot \overline{z}} = \overline{\overline{x}} + \overline{\overline{y}} + \overline{\overline{z}}$$

Ejemplos:

$$z = \overline{(\overline{A} + C)(B + \overline{D})} = \overline{(\overline{A} + C)} + \overline{(B + \overline{D})} = \overline{\overline{A}} \overline{C} + \overline{B} \overline{\overline{D}} = A \overline{C} + \overline{B} D$$

$$z = \overline{A + \overline{B} + C} = \overline{A} \overline{\overline{B}} \overline{C} = \overline{A} B \overline{C}$$

$$z = \overline{(A + BC)(D + EF)} = \overline{(A + BC)} + \overline{(D + EF)} = \overline{A} \overline{(BC)} + \overline{D} \overline{(EF)}$$
$$= \overline{A} (\overline{B} + \overline{C}) + \overline{D} (\overline{E} + \overline{F}) = \overline{A} \overline{B} + \overline{A} \overline{C} + \overline{D} \overline{E} + \overline{D} \overline{F}$$

# Formas canónicas

- Se definen dos formas canónicas de expresión booleana
  - Conjunción de maxitérminos o “Producto de Sumas”
    - Ejemplo  $(A + \neg B)(\neg A + B)$
  - Disyunción de minitérminos o “Suma de Productos”
    - Ejemplo  $A \neg B + \neg A B$
- Podemos llevar cualquier expresión booleana a una forma canónica, operando sobre ella mediante los teoremas de Boole y De Morgan.
- En los ejemplos de la transparencia anterior se han llevado las expresiones a la forma de una “suma de productos”.

- Vimos la noción de operaciones lógicas y tablas de verdad.
- Vimos que las operaciones lógicas pueden implementarse mediante compuertas (gates) lógicas
- Describimos propiedades y teoremas útiles para operar con expresiones lógicas.
- vimos formas canónicas para escribir funciones lógicas , suma de mintérminos y producto de maxtérminos.

- *Temas que se tratan*
  - Uso de compuertas para implementar circuitos lógicos
  - Compuertas universales (NAND / NOR)
  - Ejemplos

- Se definen dos formas canónicas de expresión booleana
  - Conjunción de maxitérminos o “Producto de Sumas”
    - Ejemplo  $(A+B)(\neg A+B)$
  - Disyunción de minitérminos o “Suma de Productos”
    - Ejemplo  $A\neg B + \neg AB$
- Podemos llevar cualquier expresión booleana a una forma canónica, operando sobre ella mediante los teoremas de Boole y De Morgan.

## Implementación de funciones lógicas

- Vimos que las expresiones lógicas se pueden expresar en las formas canónicas que llamamos “suma de productos” o “producto de sumas”.
- En el caso de “suma de productos” esto significa que tenemos una serie de “productos” (minitérminos) que se agrupan mediante la operación OR (“suma”) de todos ellos.

Un minitérmino está formado por la operación AND sobre distintas variables, que pueden aparecer en forma directa o negada.

- **Ejemplos:**  $f = A.B + \bar{A}.\bar{B}$      $z = \bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C}$



## Implementación de funciones lógicas

- En el caso de “producto de sumas” esto significa que tenemos una serie de “sumas” (maxitérminos) que se agrupan mediante la operación AND (“producto”) de todos ellos.

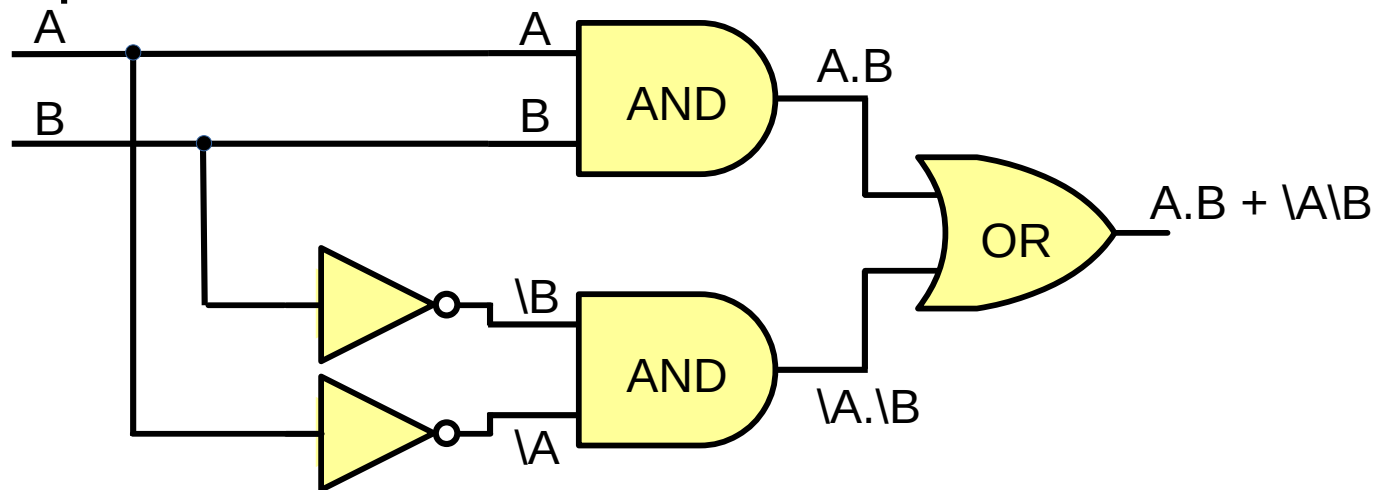
Un maxitérmino esta formado por la operación OR sobre distintas variables, que pueden aparecer en forma directa o negada.

- Ejemplos:  $y = (A + B) \cdot (\bar{A} + \bar{B})$

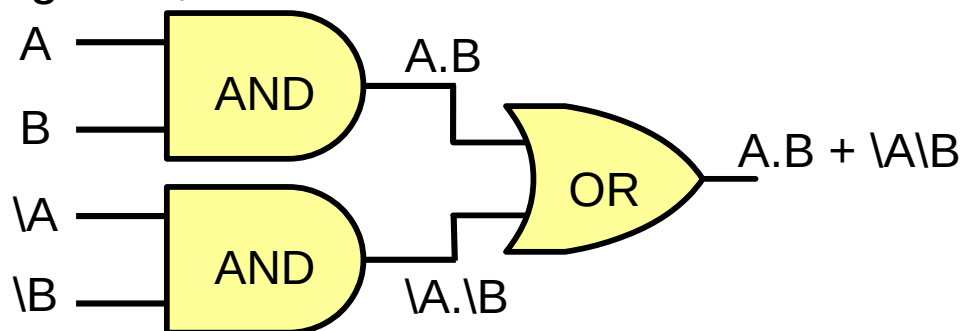
$$w = (\bar{A} + B + C) \cdot (A + \bar{B} + C) \cdot (A + B + \bar{C})$$

# Implementación con compuertas

- Implementemos  $x = A.B + \bar{A}.\bar{B}$

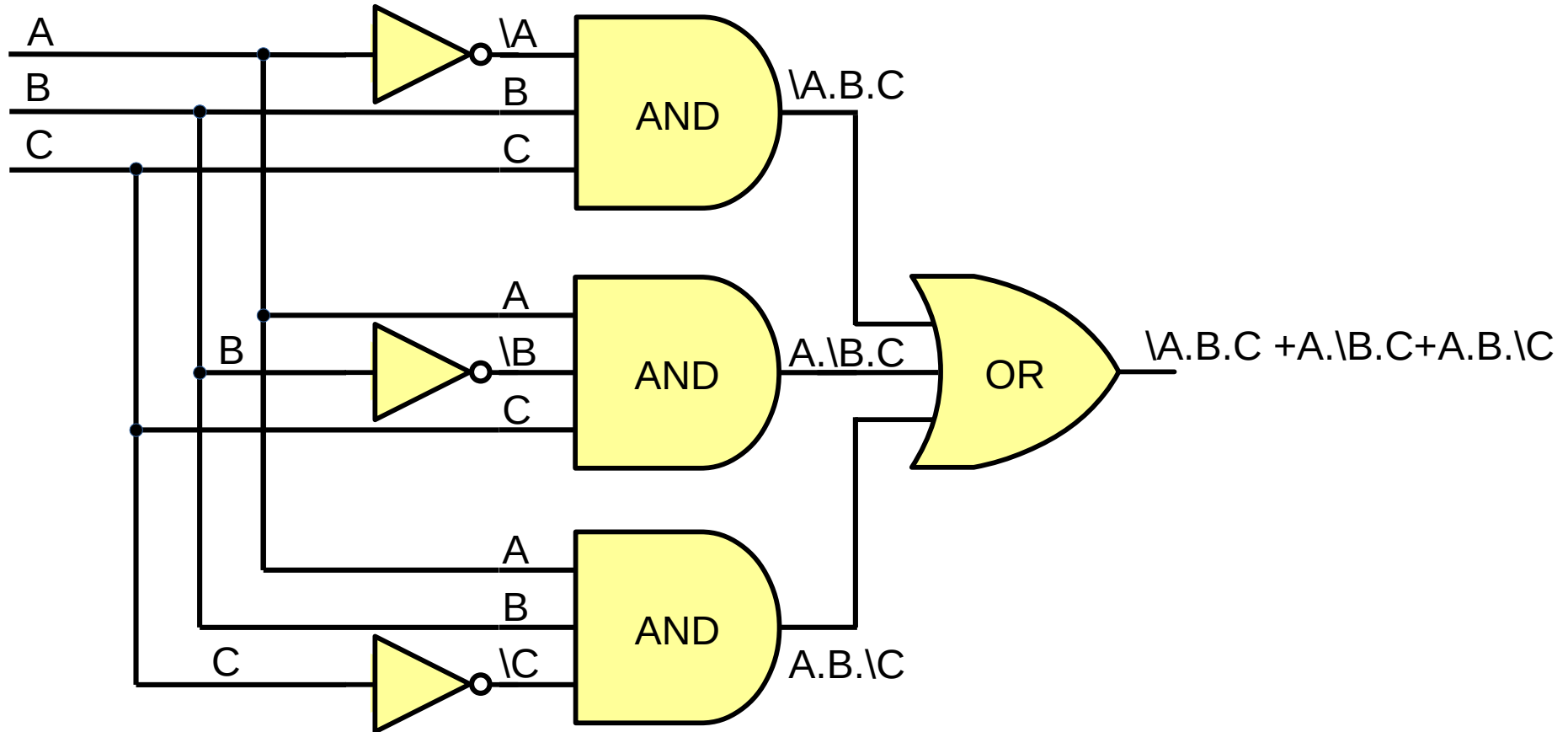


- Podemos ver que se utilizan los tres niveles mencionados.
- A veces se consideran disponibles tanto las variables como sus versiones negadas, en este caso se utilizan sólo dos niveles



# Implementación con compuertas

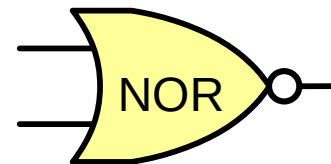
- Ahora, implementemos  $z = \bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C}$



- Podemos ver que también se utilizan los tres niveles mencionados.

# Compuertas universales

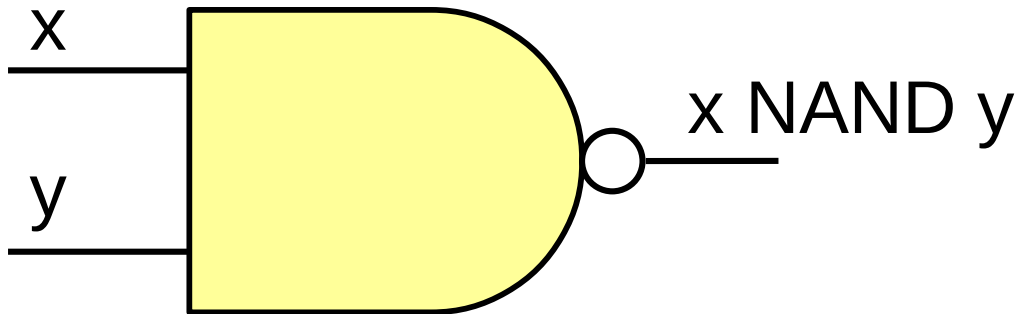
- Las compuertas NAND y NOR permiten implementar cualquier expresión lógica al conectarlas adecuadamente. Por eso se llaman Compuertas Universales.
- Durante la década del 70 y principios de los 80 se utilizaron masivamente para la implementación de sistemas digitales.
- Se distinguen facilmente de las compuertas OR y AND porque a la salida tienen un círculo que simboliza la inversión de la salida



# NAND

- La compuerta NAND es una compuerta AND con la salida invertida.

$$x \text{ NAND } y = \overline{x \cdot y}$$

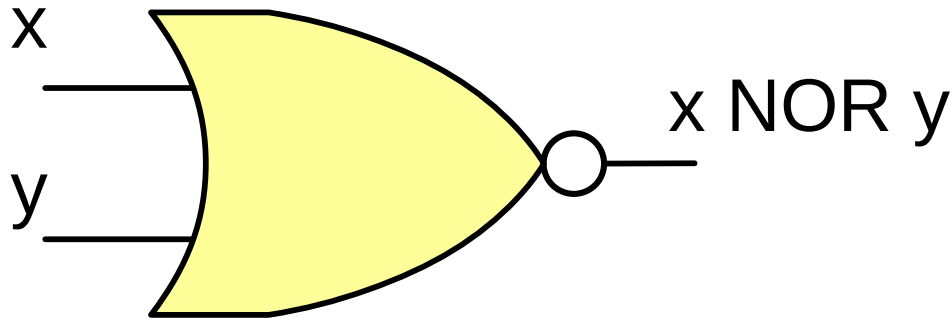


$x$	$y$	NAND
0	0	1
0	1	1
1	0	1
1	1	0

# NOR

- La compuerta NOR es una compuerta OR con la salida invertida.

$$x \text{ NOR } y = \overline{x + y}$$



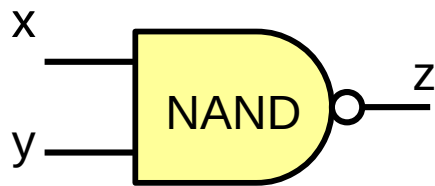
$x$	$y$	NOR
0	0	1
0	1	0
1	0	0
1	1	0

# Universalidad de NAND y NOR

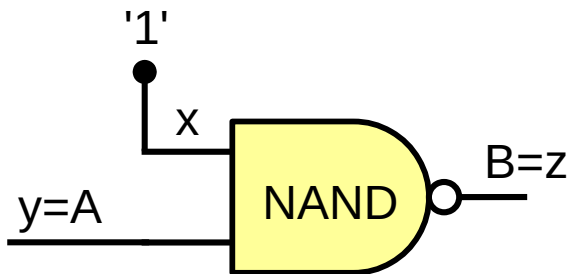
- ¿Por qué decimos que NAND y NOR son compuertas universales?
- Porque se puede implementar cualquier expresión lógica utilizando exclusivamente ya sea compuertas NOR, o bien compuertas NAND.
- Para ello debo verificar que puedo construir las tres expresiones lógicas básicas (NOT, OR, AND) a partir de ellas

# Universalidad de NAND

- NOT a partir de NAND: Dos maneras distintas

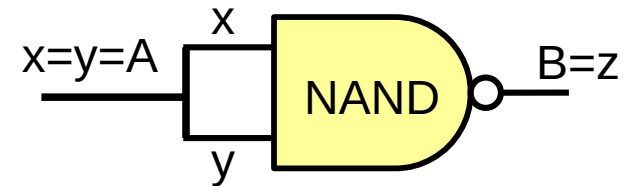


x	y	z
0	0	1
0	1	1
1	0	1
1	1	0



A	x	y	z	B
	0	0	1	
	0	1	1	
0	1	0	1	1
1	1	1	0	0

Como  $x$  no puede ser '0' las filas están en rojo



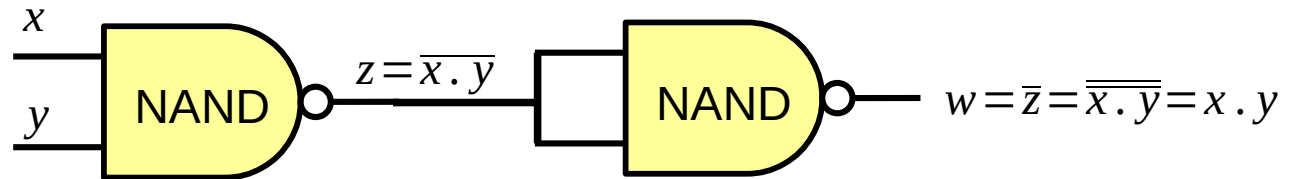
A	x	y	z	B
0	0	0	1	1
	0	1	1	
	1	0	1	
1	1	1	0	0

Como  $x$  e  $y$  no pueden ser distintos las filas están en rojo

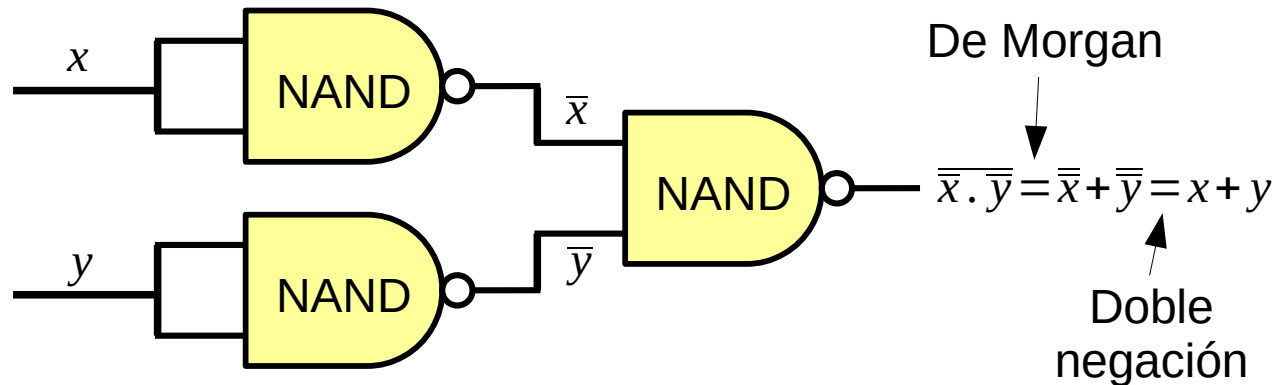


# Universalidad de NAND

- AND a partir de NAND



- OR a partir de NAND

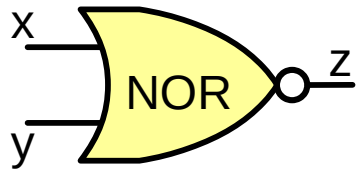


- Podemos implementar las operaciones básicas NOT, AND y OR con la compuerta NAND.

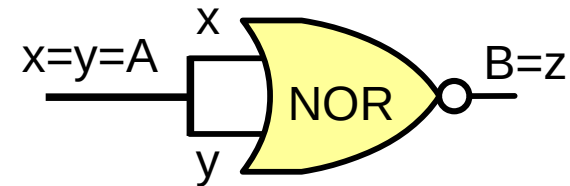
**NAND es una compuerta universal**

# Universalidad de NOR

- NOT a partir de NOR: Dos maneras distintas

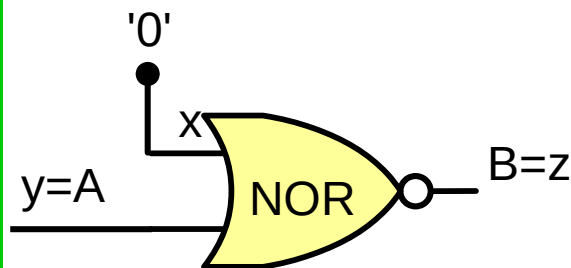


x	y	z
0	0	1
0	1	0
1	0	0
1	1	0



A	x	y	z	B
0	0	0	1	1
	0	1	0	
	1	0	0	
1	1	1	0	0

Como x e y no pueden ser distintos las filas están en rojo

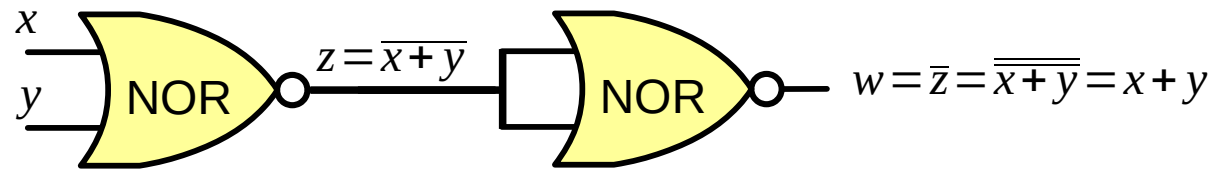


A	x	y	z	B
0	0	0	1	1
0	0	1	0	1
	1	0	0	
	1	1	0	

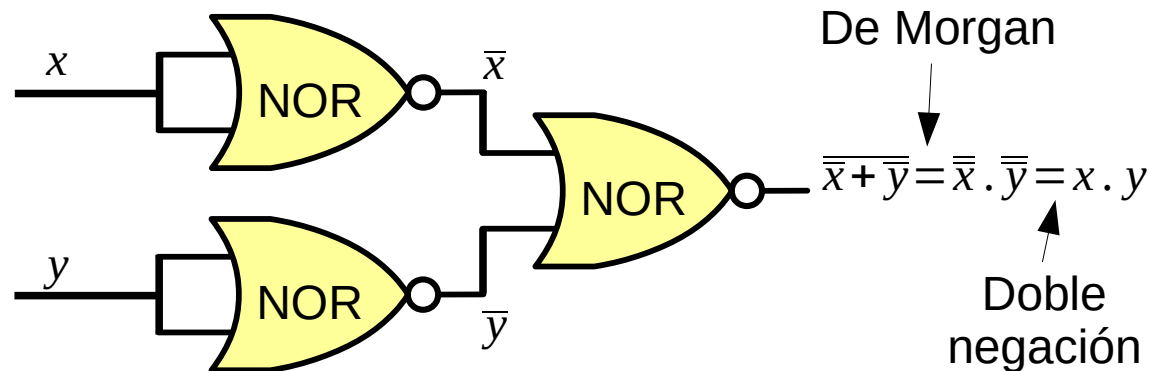
Como x no puede ser '1' las filas están en rojo

# Universalidad de NOR

- OR a partir de NOR

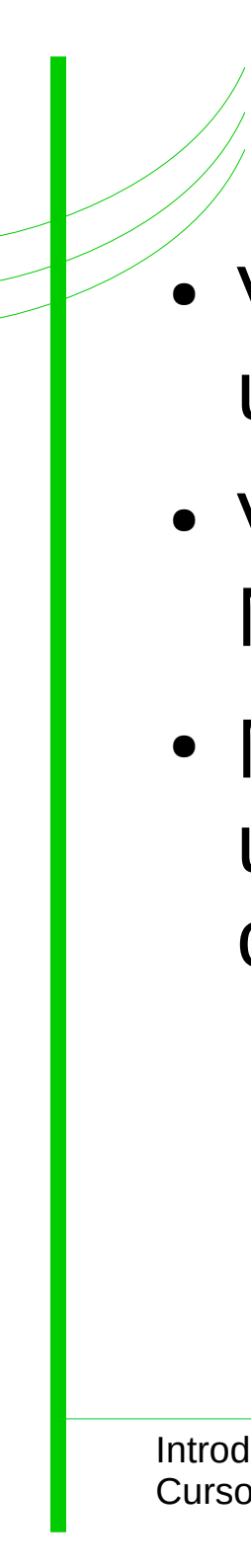


- AND a partir de NOR



- Podemos implementar las operaciones básicas NOT, AND y OR con la compuerta NOR.

**NOR es una compuerta universal**

- 
- Vimos como implementar funciones lógicas utilizando compuertas.
  - Vimos otros tipos de compuertas: NAND y NOR
  - Mostramos que NAND y NOR son compuertas universales porque permiten implementar cualquier circuito lógico