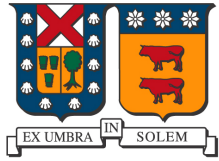




UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
CAMPUS SAN JOAQUÍN

Arquitectura y Organización de Computadores

2023



UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
CAMPUS SAN JOAQUÍN

Diseño Lógico Combinacional

Índice

- ✓ Introducción
- ✓ Compuertas Lógicas
- ✓ Axiomas Básicos
- ✓ Definiciones y Teoremas
- ✓ Funciones
- ✓ Minimización
- ✓ Combinaciones Superfluas

Introducción

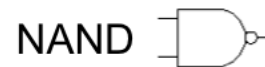
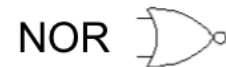
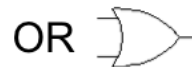
- ✓ La base conceptual del diseño de los sistemas digitales corresponde a la lógica del álgebra de Boole.
- ✓ Para poder construir los circuitos digitales será necesario conocer las compuertas lógicas.
- ✓ La minimización de funciones permitirá construir circuitos eficientes y a bajo costo.

Compuertas Lógicas

- ✓ Existen dispositivos electrónicos que son capaces de **representar funciones de conmutación**. Estos dispositivos se denominan compuertas lógicas. Estás Construidos físicamente con electrónica integrada en sustratos de silicio.
- ✓ Son altamente utilizadas en el campo de la electrónica digital, debido al bajo costo que se logra con la alta densidad de integración.
- ✓ Las compuertas corresponden a bloques fundamentales para la construcción de circuitos lógicos y sistemas digitales.

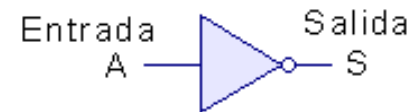
Compuertas Lógicas

- ✓ Una red de compuertas lógicas se denomina circuito combinacional.
- ✓ Los circuitos combinacionales constituyen una fracción importante de una CPU Moderna.



Compuertas Lógicas, NOT

- ✓ Realiza la operación denominada inversión o complementación.
- ✓ Cambia el nivel lógico al nivel opuesto. En términos de bits, cambia:
 - ✓ 1 por 0.
 - ✓ 0 por 1.
- ✓ El símbolo lógico estándar corresponde a:

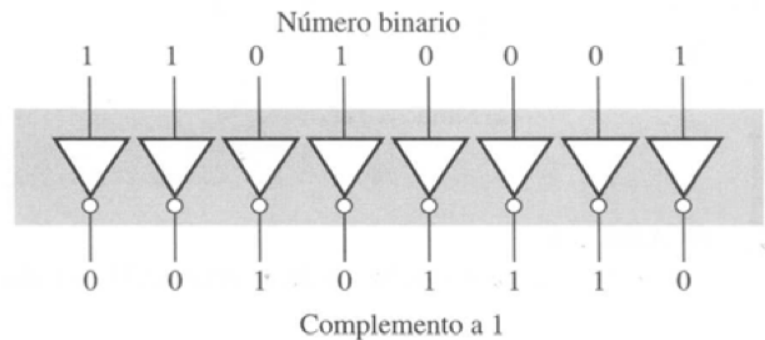


- ✓ La tabla de verdad:

A	S
0	1
1	0

Compuertas Lógicas, NOT

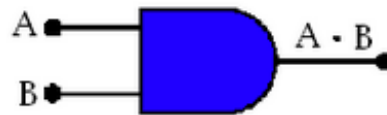
- ✓ En el álgebra de Boole, una variable se designa mediante una letra.
- ✓ Las variables pueden tomar dos valores: 0 y 1.
- ✓ El complemento de una variable se designa mediante una barra encima de la letra. Si una variable tiene un valor de verdad 1, su complemento corresponde a 0 y viceversa.
- ✓ Podemos construir un circuito que calcula el complemento 1 de un número binario de 8 bits solamente utilizando este tipo de compuertas.



Compuertas Lógicas, AND

- ✓ Corresponde a una de las compuertas básicas con la que se puede construir todas las funciones lógicas.
- ✓ Puede tener dos o más entradas, pero solo una única salida.
- ✓ Realiza la operación que se conoce como multiplicación lógica.

- ✓ El símbolo estándar:

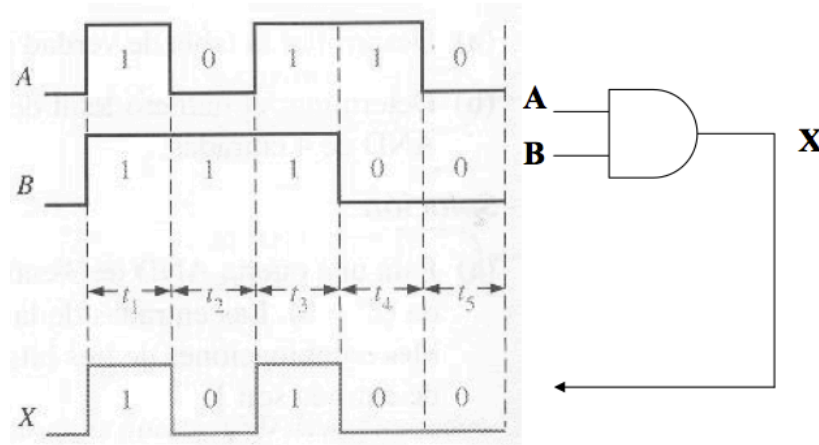


Compuertas Lógicas, AND

✓ La Tabla de Verdad:

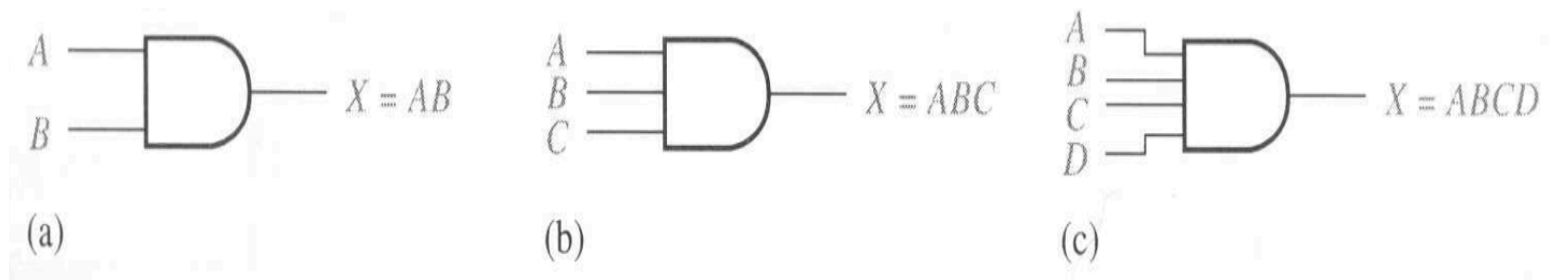
Entrada A	Entrada B	Salida $A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

✓ Diagrama de tiempos:




Compuertas Lógicas, AND

- ✓ La ecuación lógica AND de dos variables se representa con el operador “multiplicación” (\cdot) entre dos o más variables lógicas.
- ✓ La multiplicación booleana sigue las mismas reglas básicas que la multiplicación binaria.
- ✓ La implementación de la compuerta AND para más de dos entradas resulta trivial:



Compuertas Lógicas, OR

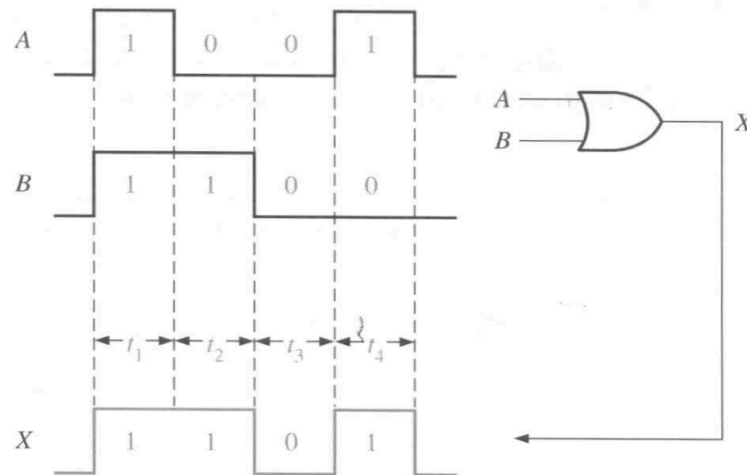
- ✓ Es otra de las puertas lógicas básicas con las que se construyen todas las funciones lógicas.
- ✓ Tiene dos o más entradas y una única salida. Realiza la operación que se conoce como suma lógica.
- ✓ Símbolo estándar: The diagram shows a standard OR gate symbol. It has two input lines on the left, labeled X_0 and X_1 , which merge into a single curved line. This line then leads to a single output line on the right, labeled Z .
- ✓ La ecuación lógica OR de dos variables se representa con el operador “suma” (+) entre dos o más variables lógicas. La suma booleana sigue las mismas reglas básicas que la suma binaria.

Compuertas Lógicas, OR

✓ La Tabla de Verdad:

Entrada A	Entrada B	Salida A+B
0	0	0
0	1	1
1	0	1
1	1	1

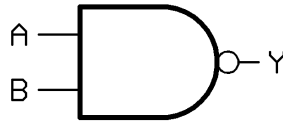
✓ Diagrama de tiempos:



Compuertas Lógicas, NAND

- ✓ Es un elemento lógico “popular” debido a que se puede utilizar como puerta universal:
 - ▶ Se pueden combinar para implementar las operaciones AND, OR y NOT.
- ✓ El término NAND se puede entender como un NOT-AND.

✓ Símbolo lógico estándar:

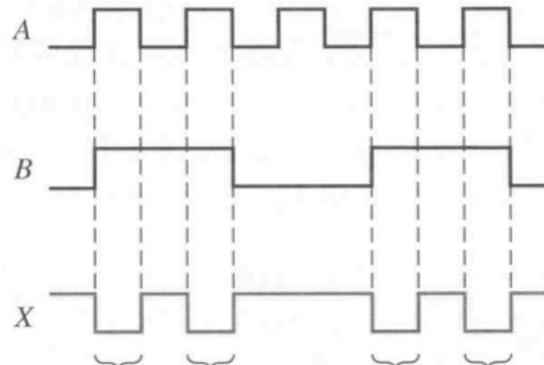


Compuertas Lógicas, NAND

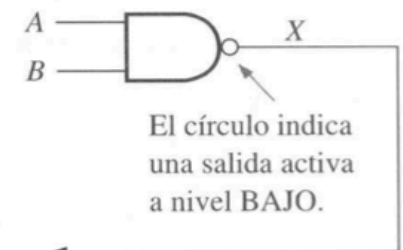
✓ La Tabla de Verdad:

Entrada A	Entrada B	X
0	0	1
0	1	1
1	0	1
1	1	0

✓ Diagrama de tiempos:



A y B están a nivel ALTO durante estos cuatro intervalos. Por tanto, X está a nivel BAJO.

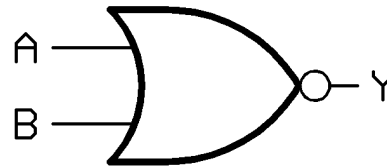


$$NAND(x, y) = \overline{x \cdot y}$$

Compuertas Lógicas, NOR

- ✓ Al igual que NAND, es un elemento lógico útil porque también se puede emplear como puerta universal.
- ✓ El término NOR puede entenderse como el NOT de OR.

- ✓ Símbolo lógico estándar:

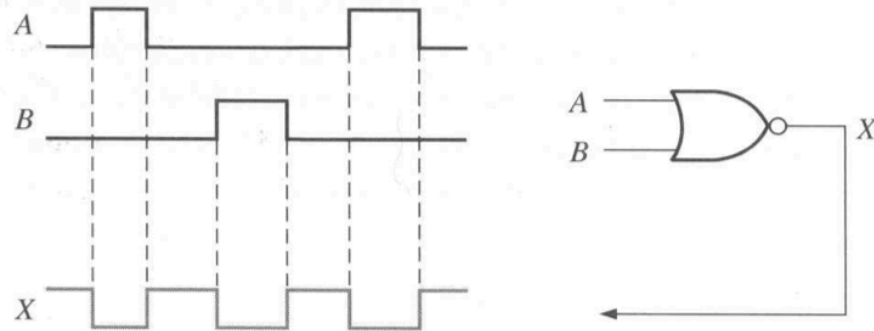


Compuertas Lógicas, NOR

✓ La Tabla de Verdad:

Entrada A	Entrada B	X
0	0	1
0	1	0
1	0	0
1	1	0

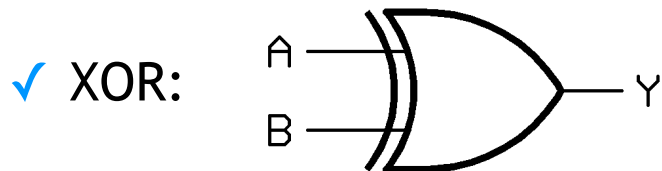
✓ Diagrama de tiempos:



$$NOR(x, y) = \overline{x + y}$$

Compuertas Lógicas, XOR y XNOR

- ✓ Las compuertas OR-Exclusiva (XOR) y NOR-Exclusiva (XNOR) se forman mediante la combinación de otras compuertas ya vistas.
- ✓ Debido a su importancia fundamental en muchas aplicaciones, estas compuertas se tratan como elementos lógicos básicos con su propio símbolo único.

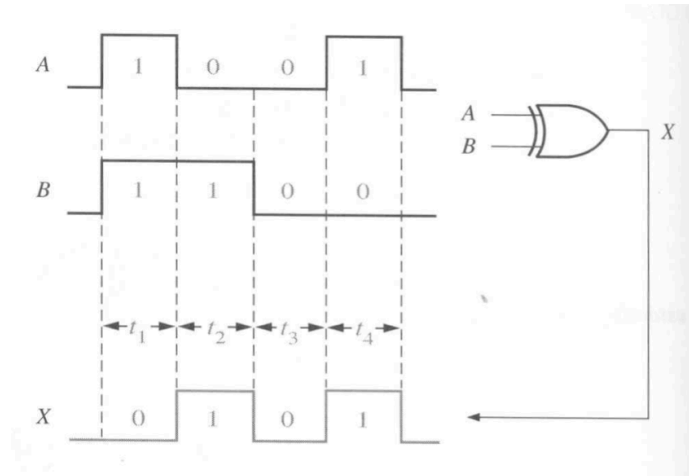


Compuertas Lógicas, XOR

✓ La Tabla de Verdad:

Entrada A	Entrada B	X
0	0	0
0	1	1
1	0	1
1	1	0

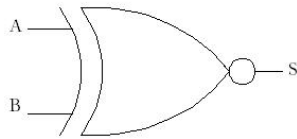
✓ Diagrama de tiempos:



$$XOR(x, y) = x \oplus y = \bar{x} \cdot y + x \cdot \bar{y}$$

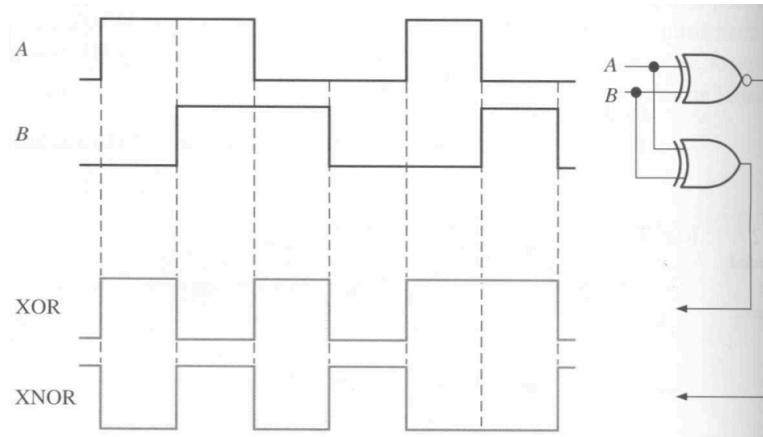
Compuertas Lógicas, XNOR

✓ La Tabla de Verdad:



Entrada A	Entrada B	X
0	0	1
0	1	0
1	0	0
1	1	1

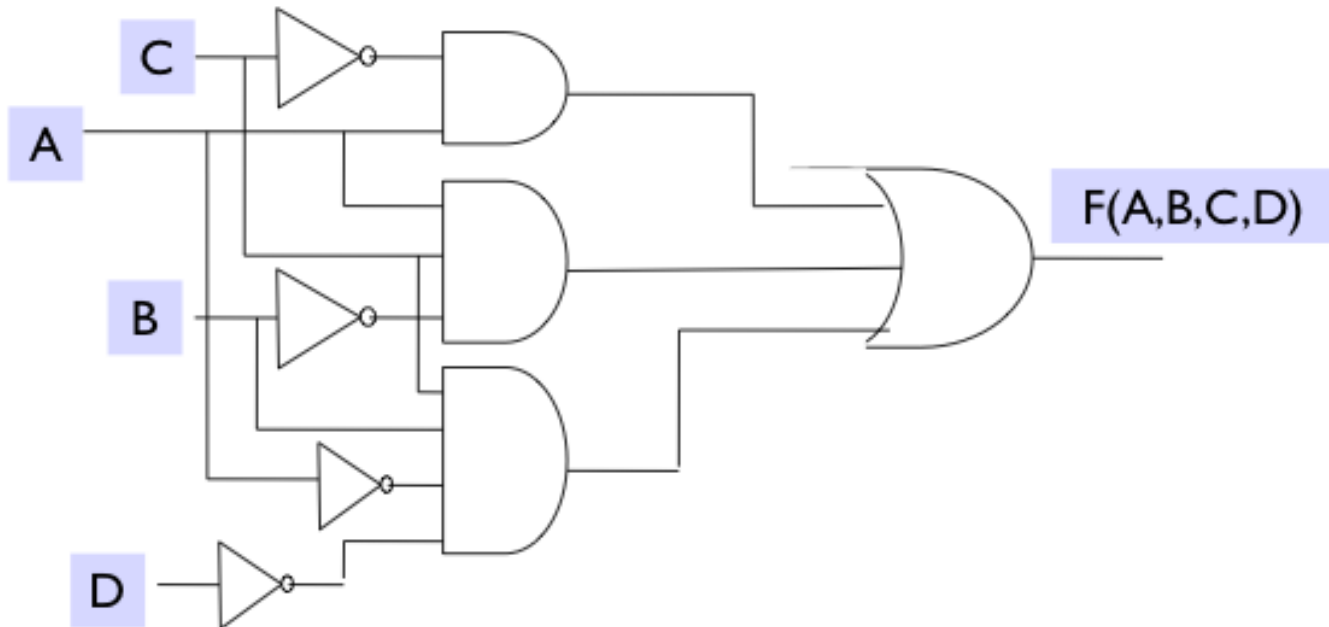
✓ Diagrama de tiempos:



Compuertas Lógicas

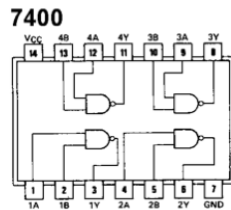
✓ Ejemplo:

$$F(A,B,C,D) = A \cdot \bar{C} + A \cdot \bar{B} \cdot C + \bar{A} \cdot B \cdot C \cdot \bar{D}$$

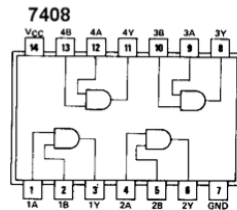


Compuertas Lógicas

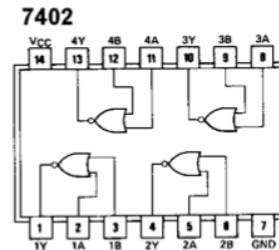
- ✓ Las compuertas lógicas se pueden encontrar en dispositivos pequeños de uso general, llamadas pastillas lógicas TTL.
- ✓ Ejemplo:



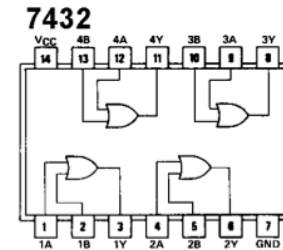
NAND



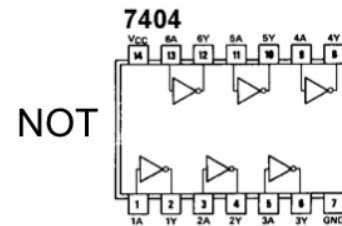
AND



NOR



OR



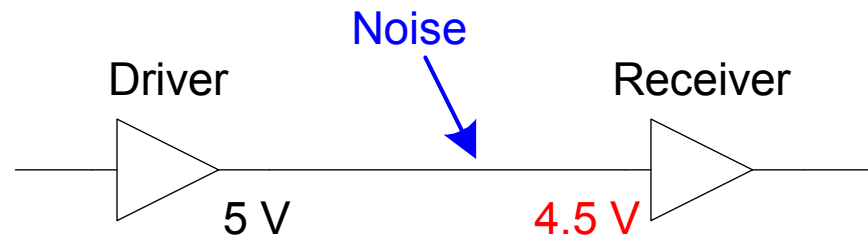
NOT

Niveles Lógicos

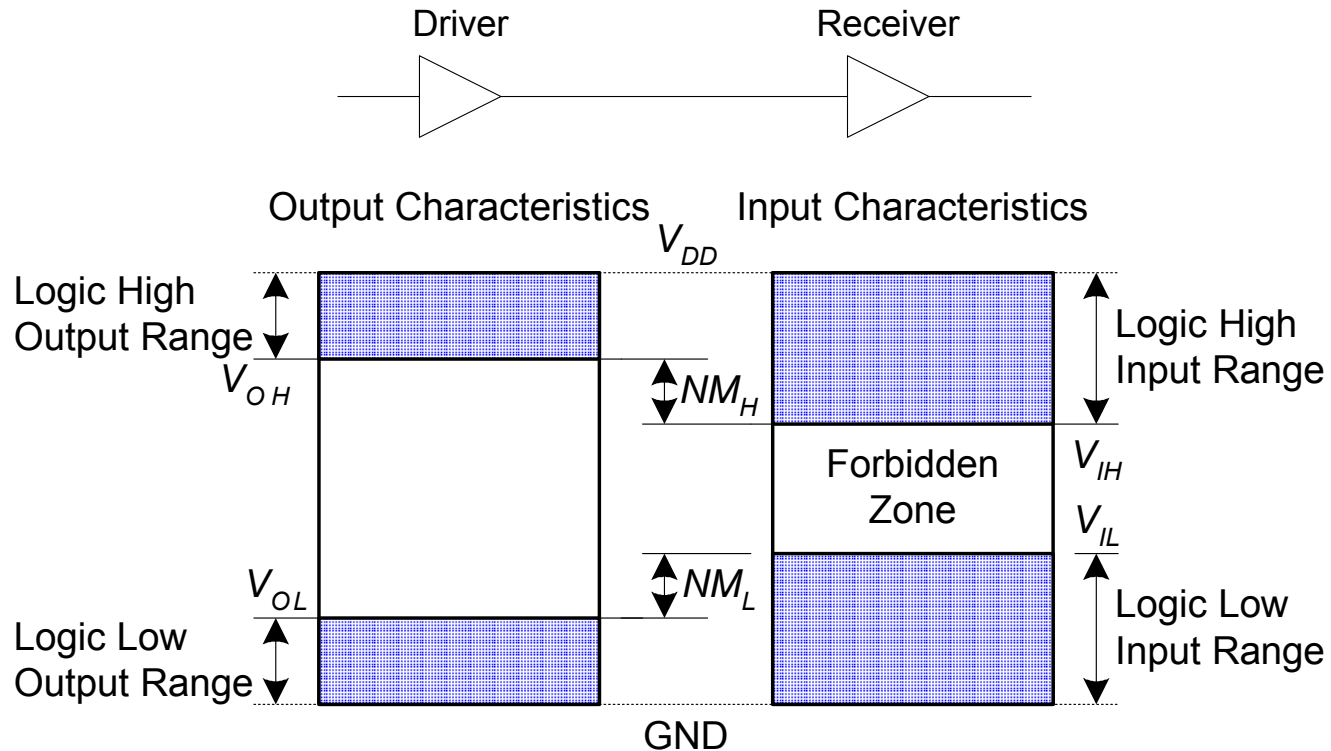
- ✓ Los sistemas digitales utilizan valores discretos para las variables. Sin embargo, estas variables son representadas por cantidades físicas continuas, tal como el Voltaje.
- ✓ Ejemplo: Una señal A. $A = 0 \rightarrow 0V$ y $A = 1$ con 5V. La mayoría de los sistemas deben tolerar un poco de “ruido”. Por lo tanto 4,97V debiesen representar $A = 1$. Pero que pasa con 4.3V, 2.8V, etc?
- ✓ El mapeo que se realiza de una variable continua a una variable discreta permite definir los niveles lógicos.

Niveles Lógicos

- ✓ ¿Qué es el ruido? Cualquier cosa que degrade la señal
 - ✓ Por ejemplo: Resistencias, ruido de la fuente de poder, problemas en circuitos, etc.
 - ✓ Considere una compuerta con salida de 5V pero una resistencia en un circuito largo entrega 4,5V



Niveles Lógicos



$$NM_H = V_{OH} - V_{IH}$$

$$NM_L = V_{IL} - V_{OL}$$

Niveles Lógicos

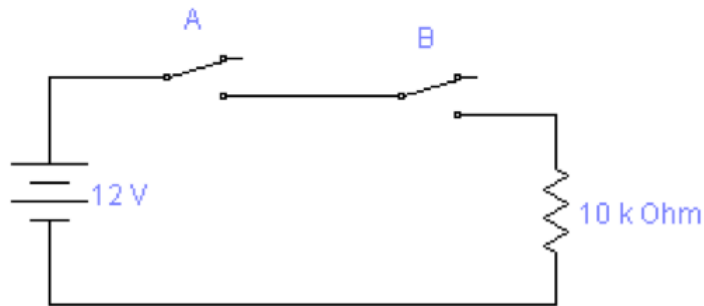
- ✓ Ejemplo: Considere un circuito inversor con los siguientes valores.
 $V_{dd} = 5V$, $V_{il} = 1.35V$, $V_{ih} = 3.15V$, $V_{ol} = 0.33V$ y $V_{oh} = 3.84V$.
¿Cuáles son los márgenes de ruido?
- ✓ Los márgenes son: $N_{Ml} = 1.35 - 0.33 = 1.02 V$
 $N_{Mh} = 3.84 - 3.15 = 0.69 V$
- ✓ Considere que la salida del Driver es igual a V_{oh} . Si el ruido reduce 1 V antes de entrar al receptor, la entrada a éste tendrá un valor de 2.84 V. Este valor es menor a lo aceptado para el valor más alto, lo que provocaría un compartimiento indefinido para la entrada.

Álgebra de Boole

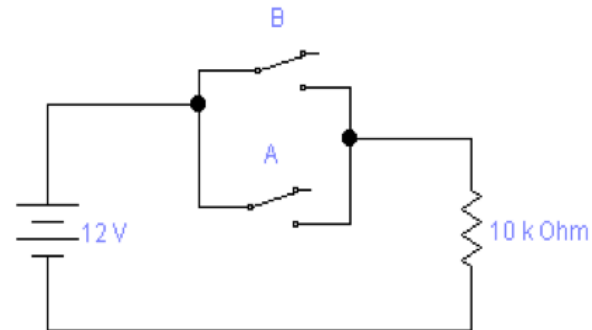
- ✓ El Álgebra de Boole es una herramienta matemática que permite modelar los llamados Sistemas Digitales.
- ✓ Fue desarrollada por el matemático inglés George Boole (1815) y propuesta en un libro llamado “Una investigación sobre las leyes del pensamiento”.
- ✓ El Álgebra de Boole es un sistema **cerrado** que utiliza variables y operadores lógicos. Las **variables** pueden tomar valores del conjunto $\{0,1\}$. Las operaciones básicas son OR y AND. Se definen las expresiones de conmutación como un número finito de variables y constantes, relacionadas mediante los operadores (OR y AND)

Álgebra de Boole

- ✓ En la ausencia de paréntesis, se utilizan las mismas reglas de precedencia, que tienen los operadores suma (OR) y multiplicación (AND) en el álgebra normal.
- ✓ Ejemplo: Circuito Conmutador:



Lógica AND



Lógica OR

Axiomas Básicos

✓ El Álgebra de Boole se rige por los siguientes axiomas o postulados:

✓ Conmutatividad:

$$A + B = B + A$$

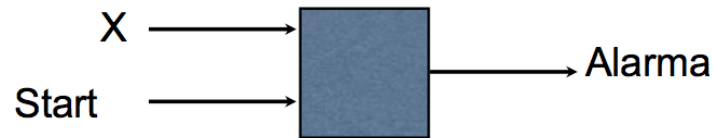
$$A \cdot B = B \cdot A$$

✓ Asociatividad: $A + (B + C) = (A + B) + C$

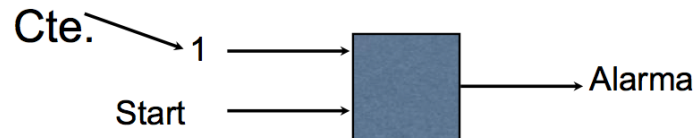
$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

Definiciones Básicas

- ✓ **Variable:** Son símbolos que pueden representar cualquier valor entre $\{0,1\}$. El valor de las variables puede cambiar en el transcurso del tiempo. El concepto es igual a variables en los lenguajes de programación.



- ✓ **Constante:** Son símbolos que representan sólo un valor del conjunto para cualquier instante de tiempo. Su concepto es igual al de constantes en lenguajes de programación.



Definiciones Básicas

- ✓ **Expresión de Conmutación:** Es una combinación de un número finito de **variables** y **constantes** relacionadas mediante operaciones **OR** y **AND**.
- ✓ Para simplificar el uso de paréntesis, se aplican reglas de precedencia de operadores, al igual que las expresiones del álgebra normal, considerando el OR como suma y AND como producto.
- ✓ Ejemplo: $A \cdot B + C + B \cdot D + 0 + D$

Definiciones Básicas

- ✓ **Literal:** Es toda ocurrencia de una variable, ya sea complementada o sin complementar, en una expresión de conmutación.
 - ✓ Ejemplo: $\bar{A} \cdot B + \bar{C} + \bar{B} \cdot D + 0 + D$
 - ✓ Existen 4 variables: A, B, C y D
 - ✓ Existen 5 literales: $\bar{A}, B, \bar{C}, \bar{B}, D$
- ✓ **Expresión Dual:** Es la expresión que se obtiene intercambiando las operaciones AND por OR, OR por AND y las constantes 0 por 1 y 1 por 0 en una expresión de conmutación.

$$\bar{A} + (B \cdot \bar{C}) + 0 \rightarrow \bar{A} \cdot (B + \bar{C}) \cdot 1$$

Teoremas

- ✓ De los axiomas anteriores se desprenden un conjunto de teoremas.
La forma de demostración es:
 - ✓ Aplicando axiomas
 - ✓ Por inducción
- ✓ Se verán un conjunto de teoremas que sirven para trabajar con las expresiones de conmutación dándoles un sentido práctico.

Teoremas

- ✓ Teorema 1. Operaciones Básicas:
 - ▶ $1 + 0 = 1$
 - ▶ $1 \cdot 1 = 1$
 - ▶ $0 + 0 = 0$
 - ▶ $0 \cdot 1 = 0$
- ✓ Se desprende del axioma: $A+0=A$, $A \cdot 1=A$.
- ✓ Corolario: $\overline{0} = 1$ $\overline{1} = 0$

Teoremas

✓ Teorema 2. Operaciones Básicas:

- $A + 1 = 1$
- $A \cdot 0 = 0$

✓ Demostración:

$$A + 1 = (A + 1) \cdot 1$$

$$\dots\dots\dots = (A + 1) \cdot (A + \bar{A})$$

$$\dots\dots\dots = (A + 1 \cdot \bar{A})$$

$$\dots\dots\dots = A + \bar{A}$$

$$\dots\dots\dots = 1$$

Operaciones AND, OR y NOT

AND	OR	NOT
$0 \cdot 0 = 0$	$0 + 0 = 0$	$\text{Not}(1) = 0$
$0 \cdot 1 = 0$	$0 + 1 = 1$	$\text{Not}(0) = 1$
$1 \cdot 0 = 0$	$1 + 0 = 1$	
$1 \cdot 1 = 1$	$1 + 1 = 1$	

Teoremas

✓ Teorema 3. Los operadores son idempotentes:

$$\forall A \in P:$$

$$A + A = A$$

$$A \cdot A = A$$

✓ Demostración:

$$A + A = (A + A) \cdot 1$$

$$\dots\dots\dots = (A + A) \cdot (A + \bar{A})$$

$$\dots\dots\dots = A + (A \cdot \bar{A})$$

$$\dots\dots\dots = A + 0$$

$$\dots\dots\dots = A$$

Teoremas

✓ Teorema 4. Absorción:

$$A + A \cdot B = A$$

$$A \cdot (A + B) = A$$

✓ Demostración:

$$A + A \cdot B = A \cdot 1 + A \cdot B$$

$$\dots\dots\dots = A \cdot (1 + B)$$

$$\dots\dots\dots = A \cdot (B + 1)$$

$$\dots\dots\dots = A \cdot 1$$

$$\dots\dots\dots = A$$

Teoremas

✓ Teorema 5. Simplificación:

$$A + \bar{A} \cdot B = A + B$$

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

✓ Demostración:

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

$$\dots\dots\dots = 1 \cdot (A + B)$$

$$\dots\dots\dots = A + B$$

Teoremas

- ✓ Teorema 6. El complemento de A es único. Supongamos que existen dos complementos de A: A_1 y A_2 .

$$A + A_1 = 1 \qquad A + A_2 = 1$$

$$A \cdot A_1 = 0 \qquad A \cdot A_2 = 0$$

$$A_1 = A_1 \cdot 1 = A_1 (A + A_2) = A_1 \cdot A + A_1 \cdot A_2$$

$$A_1 = 0 + A_2 \cdot A_1$$

$$A_1 = A \cdot A_2 + A_1 \cdot A_2 = (A + A_1) \cdot A_2$$

$$A_1 = A_2$$

Teoremas

- ✓ Teorema 7. Involución: $\overline{\overline{A}} = A$
- ✓ ¿Cuál es el complemento de \overline{A} ?
 - ▶ A es un complemento de \overline{A}
 - ▶ Este complemento es único.

Teoremas

- ✓ Teorema 8. Leyes de Morgan:

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

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

- ✓ Morgan establece que para obtener el complemento de una expresión se debe complementar cada variable e intercambiar las operaciones OR y AND y las constantes 1 y 0.
- ✓ Por ejemplo:

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

Funciones de Conmutación

- ✓ Una función de conmutación se define como una relación de P^n en P . Formalmente es:

$$P = \{0, 1\}$$

$$f : P^n \rightarrow P$$

- ✓ Se pueden expresar en:
 - Forma algebraica.
 - Por una tabla de verdad.
 - Forma canónica.

Tablas de Verdad

- ✓ La forma más intuitiva de **representar una función** de conmutación (FC) es por medio de una tabla de verdad.
- ✓ La tabla de verdad expresa el valor de salida de una función para cada combinación de entrada.
- ✓ La tabla de verdad permite modelar un tipo especial de sistema digital llamado **Sistema Combinacional**.

Tablas de Verdad

✓ Ejemplo: $f(x_1, x_2, x_3) = x_1 \cdot \bar{x}_2 + x_2 \cdot \bar{x}_3$

✓ Tabla de verdad:

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Tablas de Verdad

✓ Ejemplo: Dada una tabla de verdad, obtener la forma algebraica.

✓ Tabla de verdad:

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

$$\bar{x}_1 \cdot x_2 \cdot \bar{x}_3$$

$$x_1 \cdot \bar{x}_2 \cdot \bar{x}_3$$

$$x_1 \cdot \bar{x}_2 \cdot x_3$$

$$x_1 \cdot x_2 \cdot \bar{x}_3$$

Formas Canónicas

- ✓ Se denomina mintérmino a un término de una expresión booleana que está formado por el AND de todas las variables.
- ✓ Una función de conmutación corresponde al OR de mintérminos. La función generada de esta manera se denomina OR canónico de AND.
- ✓ Para la representación de la forma canónica, se utilizan las posiciones de los mintérminos en la tabla de verdad.

$$F(x_1, x_2, x_3) = OR(m_1, m_2, \dots, m_n)$$

$$F(x_1, x_2, x_3) = \sum(m_1, m_2, \dots, m_n)$$

Formas Canónicas

✓ Para la función:

$$F(x_1, x_2, x_3) = \bar{x}_1 \cdot x_2 \cdot \bar{x}_3 + x_1 \cdot \bar{x}_2 \cdot \bar{x}_3 + x_1 \cdot \bar{x}_2 \cdot x_3 + x_1 \cdot x_2 \cdot \bar{x}_3$$

✓ La forma canónica:

$$F(x_1, x_2, x_3) = \sum (2, 4, 5, 6)$$

Formas Canónicas

- ✓ Una forma alternativa de expresar la función es examinando las combinaciones en las cuales la función vale 0. Esto corresponde a los maxtérminos.

x_1	x_2	x_3	f
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	0

$$x_1 + x_2 + x_3$$

$$x_1 + \bar{x}_2 + x_3$$

$$\bar{x}_1 + x_2 + x_3$$

$$\bar{x}_1 + \bar{x}_2 + \bar{x}_3$$

$$F(x_1, x_2, x_3) = \prod (0, 2, 4, 7)$$

Formas Canónicas

- ✓ Se denomina maxtérmino a la expresión booleana que está formada por el OR de todas las variables.
- ✓ Una función conmutación corresponde al AND de maxtérminos. La función generada de esta manera se denomina AND de OR.

$$F(x_1, x_2, x_3) = \text{AND}(M_1, M_2, \dots, M_n)$$

$$F(x_1, x_2, x_3) = \prod (M_1, M_2, \dots, M_n)$$

Formas Canónicas

- ✓ ¿Es posible obtener la forma canónica de cualquier función booleana?
- ✓ Por ejemplo:

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

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

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

$$F(A, B, C) = \sum (4, 5, 7)$$

Formas Canónicas

- ✓ Dada una función en OR canónico de AND. ¿Es posible obtener la forma canónica AND canónico de OR? Y ¿viceversa?
- ✓ Por ejemplo:

$$F(A, B, C) = \sum (0, 1, 2, 7)$$

$$\overline{F(A, B, C)} = \sum (3, 4, 5, 6)$$

$$\overline{F(A, B, C)} = \bar{A} \cdot B \cdot C + A \cdot \bar{B} \cdot \bar{C} + A \cdot \bar{B} \cdot C + A \cdot B \cdot \bar{C}$$

$$\overline{F(A, B, C)} = (A + \bar{B} + \bar{C}) \cdot (\bar{A} + B + C) \cdot (\bar{A} + B + \bar{C}) \cdot (\bar{A} + \bar{B} + C)$$

$$F(A, B, C) = \prod (3, 4, 5, 6)$$

Funciones Equivalentes

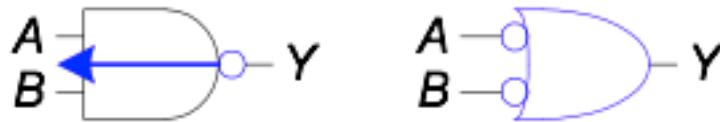
- ✓ Dos funciones de conmutación son equivalentes cuando sus **expansiones en formas canónicas son idénticas**, es decir, tienen el mismo valor de salida para las mismas combinaciones de entradas.
- ✓ Una forma similar de expresar lo mismo es que dos funciones de conmutación son equivalentes cuando **tienen la misma tabla de verdad**.
- ✓ ¿Cuántas funciones distintas (no equivalentes) existen para un número n de variables? La respuesta radica en la pregunta ¿Cuántas tablas de verdad existen con n variables?

Algunos Operadores

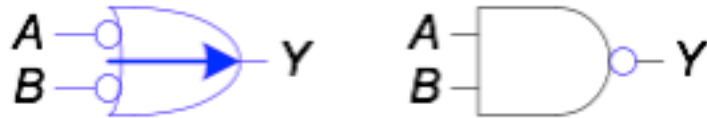
- NOT: $F(x) = \bar{x}$
- AND: $F(x, y) = x \cdot y$
- OR: $F(x, y) = x + y$
- NAND: $F(x, y) = \bar{x} + \bar{y}$
- NOR: $F(x, y) = \bar{x} \cdot \bar{y}$
- XAND: $F(x, y) = x \cdot y + \bar{x} \cdot \bar{y}$
- XOR: $F(x, y) = \bar{x} \cdot y + x \cdot \bar{y}$

Burbujas

✓ Hacia atrás:

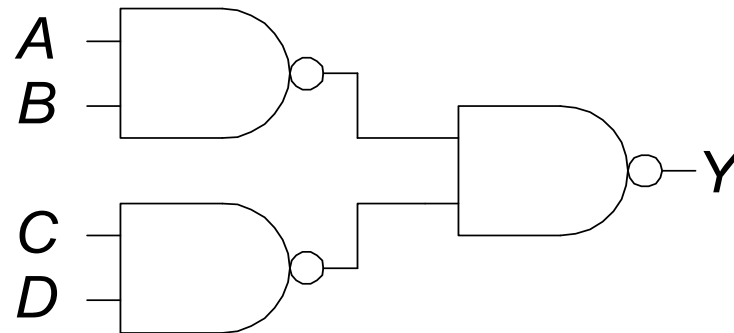


✓ Hacia adelante:



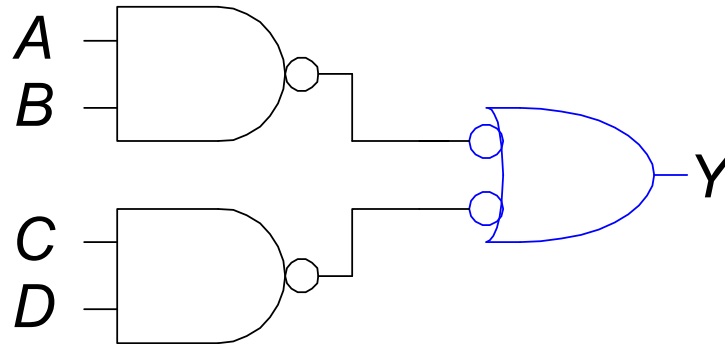
Burbujas

✓ Función booleana que represente el siguiente circuito?



Burbujas

✓ Función booleana que represente el siguiente circuito?



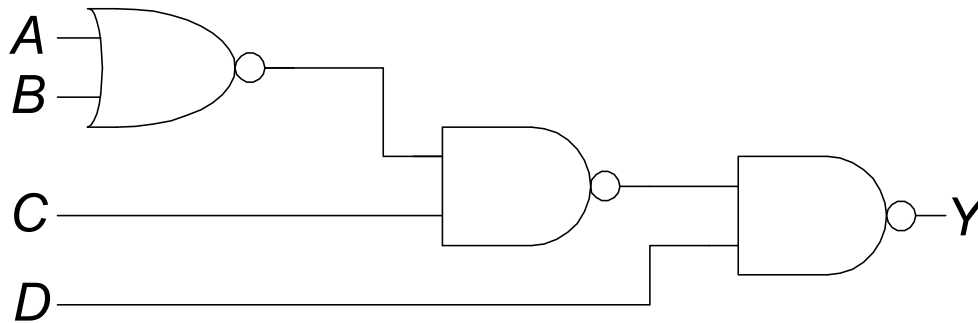
$$Y = AB + CD$$

Burbujas

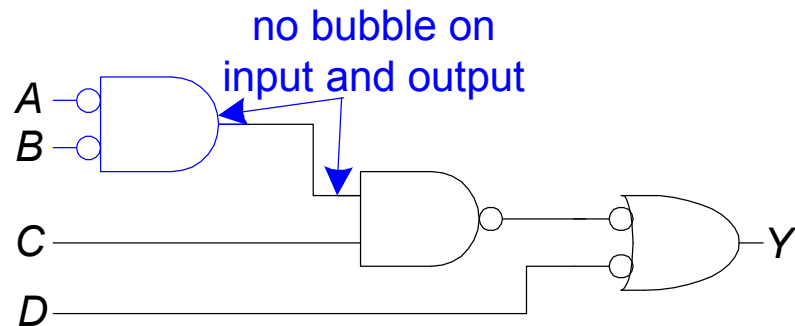
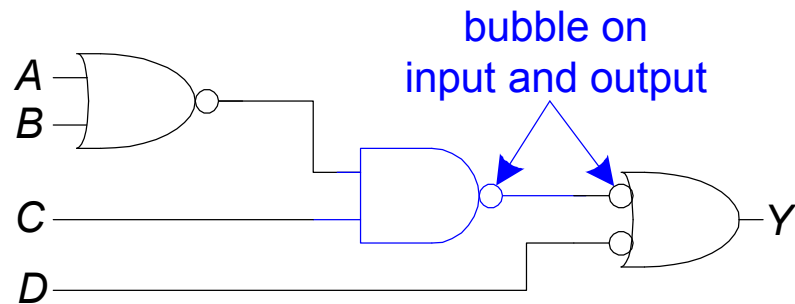
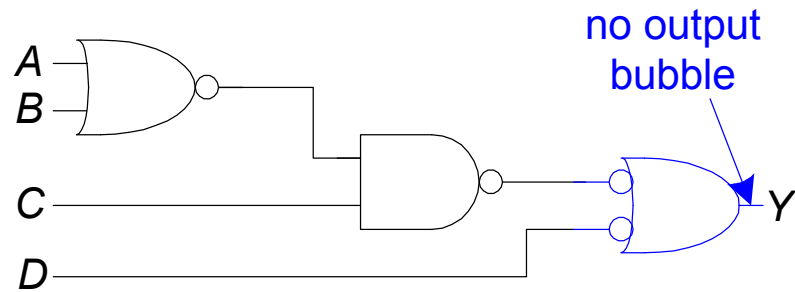
✓ Algunas reglas:

- ▶ Comenzar por las salidas, luego revisar las entradas.
- ▶ Eliminar las burbujas de la salidas.
- ▶ Intentar eliminar las burbujas.

✓ Ejemplo:



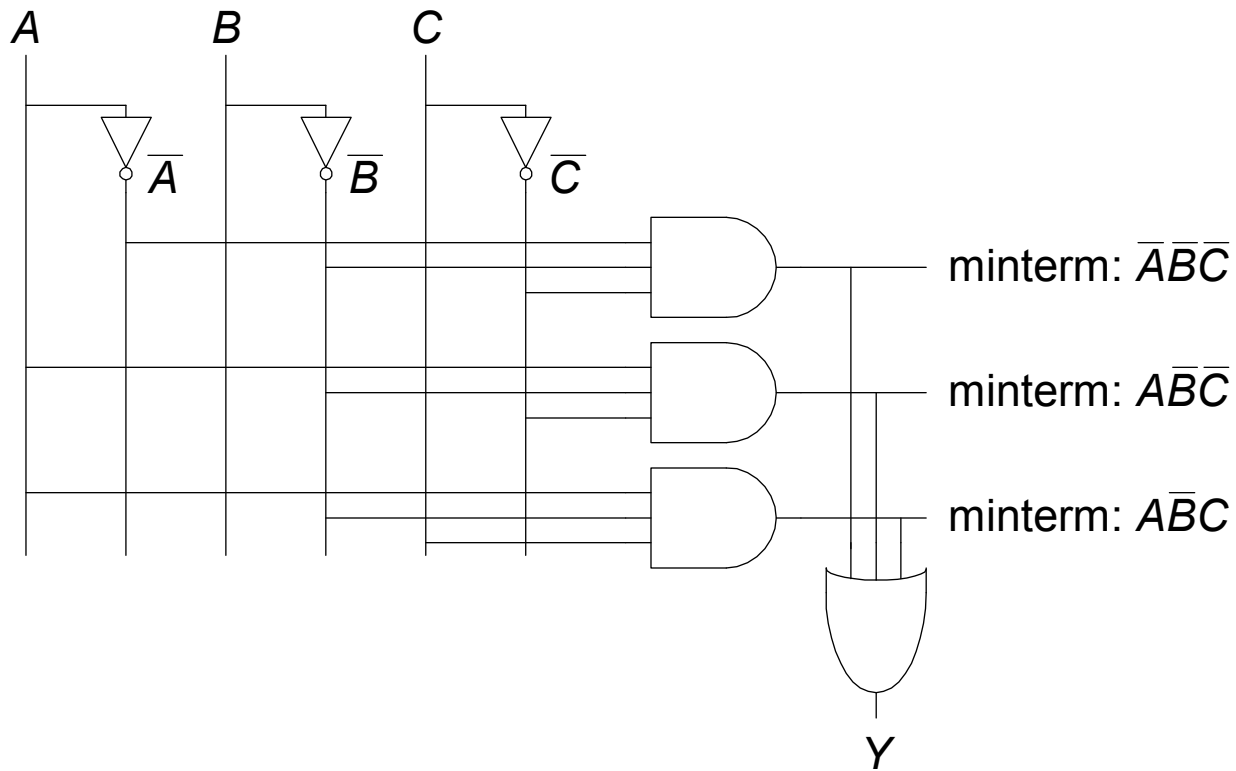
Burbujas



$$Y = \overline{A}\overline{B}C + \overline{D}$$

De la lógica a las compuertas

- Dos niveles: ANDs seguidos de ORs
- Ejemplo: $Y = \neg A \neg B \neg C + A \neg B \neg C + A \neg B C$



Operadores Funcionalmente Completos

- ✓ Las funciones, en general, se pueden expresar por medio de los operadores: AND, OR, NAND, NOR, NOT, XOR y XAND.
- ✓ Se dice que un conjunto de operaciones es **funcionalmente completo** si cualquier función se puede expresar sólo con los operadores del conjunto.
- ✓ El conjunto {AND, OR, NOT} es funcionalmente completo por definición del álgebra. El conjunto {AND, NOT} también lo es.
- ✓ Demostrar que NAND y NOR son funcionalmente completos.

Minimización de Funciones

- ✓ Minimizar una función de conmutación $F(x_1, \dots, x_n)$ es encontrar una función $G(x_1, \dots, x_n)$ equivalente a F y que contenga el mínimo número de términos y literales en una expresión OR de AND.

- ✓ Ejemplo:

$$F(x, y, z) = \bar{x} \cdot \bar{y} \cdot z + \bar{x} \cdot \bar{y} \cdot \bar{z}$$

$$F(x, y, z) = \bar{x} \cdot \bar{y} \cdot (z + \bar{z})$$

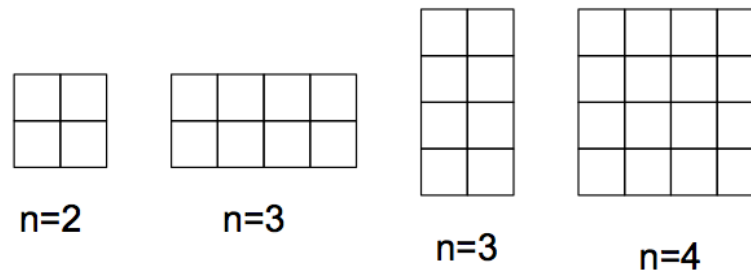
$$F(x, y, z) = \bar{x} \cdot \bar{y}$$

Mapas de Karnaugh

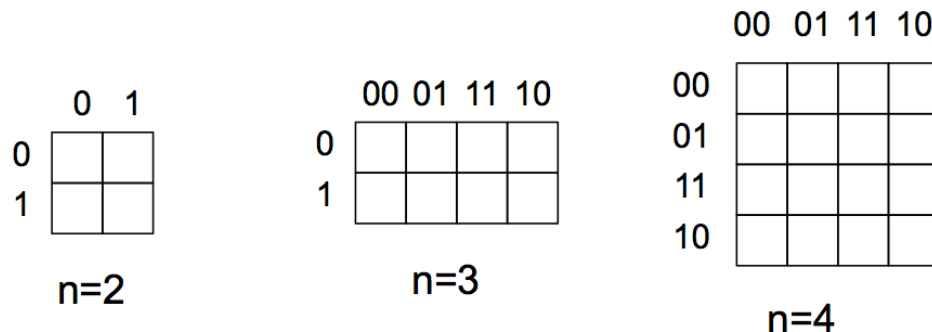
- ✓ Los mapas de Karnaugh son una herramienta gráfica utilizada para simplificar las ecuaciones lógicas o bien, minimizar funciones de conmutación.
- ✓ Estos mapas son una versión modificada de las tablas de verdad, permitiendo mostrar la relación entre las entradas lógicas y la salida deseada.
- ✓ Los mapas de Karnaugh permiten el diseño de circuitos con el mínimo de compuertas, por lo que tienen un gran impacto en la reducción de costos.

Mapas de Karnaugh

- ✓ Al igual que en las tablas de verdad, una función de n variables tiene 2^n combinaciones de posibles valores de entrada. En el caso de los mapas de Karnaugh, estas combinaciones se representan por celdas.



- ✓ Luego, en las coordenadas se anotan las combinaciones de variables según el código Gray.



Mapas de Karnaugh

- ✓ Si se tiene una tabla de verdad, basta con escribir en cada celda la salida correspondiente de la tabla de verdad. Por ejemplo:

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

		AB			
		00	01	11	10
C	0	1	1	1	0
	1	0	0	1	1

Mapas de Karnaugh

- ✓ De igual forma se puede representar una función de la forma canónica, como un mapa de Karnaugh. Para ello se debe asignar un 0 a una variable complementada y 1 a una variable sin complementar.

		AB			
		00	01	11	10
C	0	$\overline{A}\overline{B}\overline{C}$	$\overline{A}B\overline{C}$	$A\overline{B}\overline{C}$	$AB\overline{C}$
	1	$\overline{A}\overline{B}C$	$\overline{A}BC$	$A\overline{B}C$	ABC

→

		AB			
		00	01	11	10
C	0	000	010	110	100
	1	001	011	111	101

Mapas de Karnaugh

- ✓ Con esto se consigue la siguiente numeración para las celdas:

		AB			
		00	01	11	10
C	0	0	2	6	4
	1	1	3	7	5

- ✓ Luego, si se requiere representar la función $F(A,B,C) = (0,2,3,7)$

		AB			
		00	01	11	10
C	0	1	1	0	0
	1	0	1	1	0

Mapas de Karnaugh

✓ Para 4 variables:

		AB			
		00	01	11	10
CD	00	0	4	12	8
	01	1	5	13	9
	11	3	7	15	11
	10	2	6	14	10

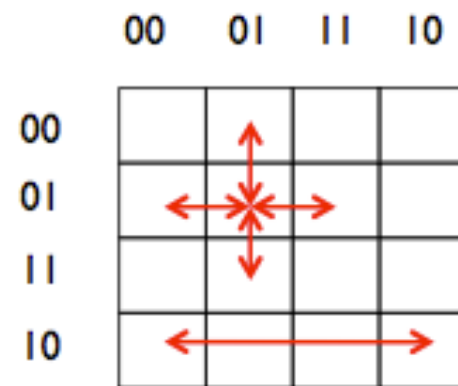
✓ Ejemplo:

$$F(A, B, C, D) = \Sigma(0,1,5,6,9,13,15)$$

		AB			
		00	01	11	10
CD	00	1	0	0	0
	01	1	1	1	1
	11	0	0	1	0
	10	0	1	0	0

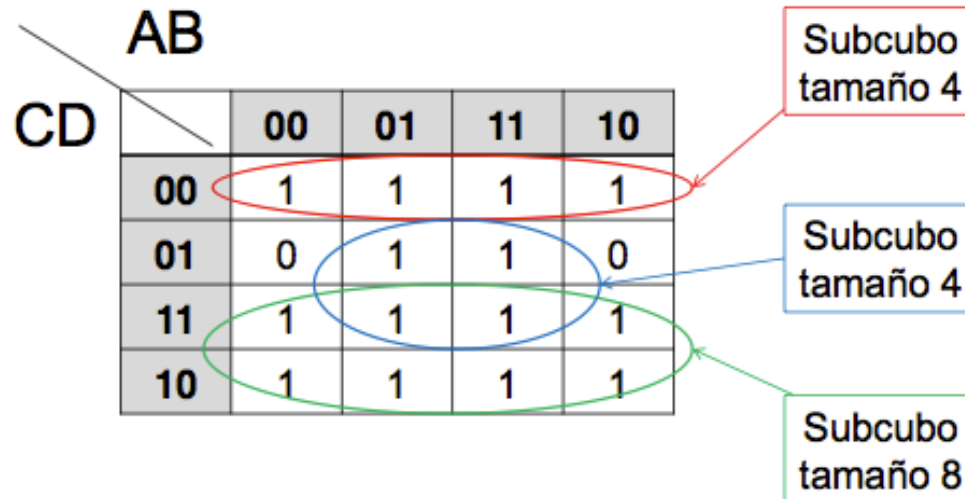
Mapas de Karnaugh, Minimización

- ✓ La minimización de funciones sobre el MK aprovecha que las celdas del mapa están arregladas de tal forma que entre una celda y otra, en forma horizontal o vertical existe **adyacencia lógica**. Esto quiere decir que entre una celda y otra sólo cambia una variable.
- ✓ Agrupando celdas adyacentes se obtienen términos que eliminan las variables que se complementan, resultado esto en una versión simplificada de la expresión.
- ✓ Dos celdas son adyacentes sólo si difieren en una de las variables.



Mapas de Karnaugh, Minimización

- ✓ Un subcubo es un conjunto de 2^m celdas con valor 1, las cuales tienen la propiedad que cada celda es adyacente a m celdas del conjunto.



Mapas de Karnaugh, Minimización

- ✓ Un subcubo puede ser expresado por un término algebraico que contiene $n-m$ literales, donde n es el número de variables y 2^m es el tamaño del subcubo.

AB \ CD	00	01	11	10
00	1	1	1	1
01	0	1	1	0
11	1	1	1	1
10	1	1	1	1

Red oval (top row): \overline{CD}

Blue oval (middle two rows): BD

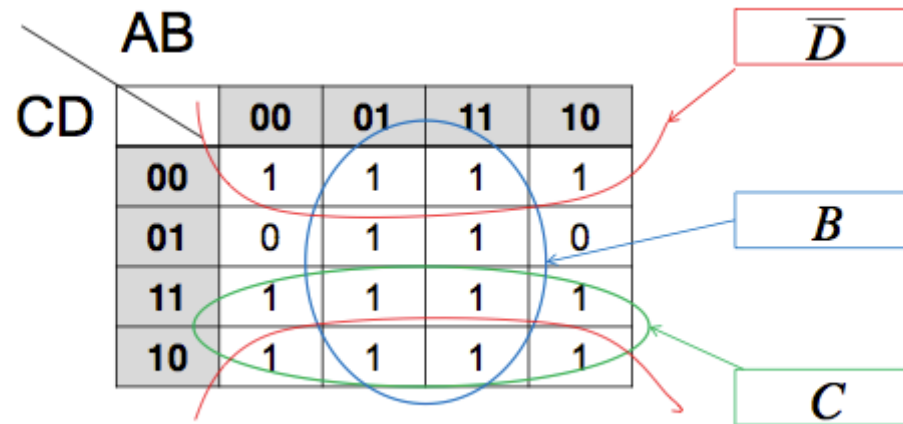
Green oval (bottom two rows): C

Mapas de Karnaugh, Minimización

- ✓ Un subcubo puede ser expresado por un término algebraico que contiene $n-m$ literales, donde n es el número de variables y 2^m es el tamaño del subcubo.
- ✓ Si se suman los términos dados por los subcubos que abarcan todos los unos del mapa, se obtiene la función algebraica.
- ✓ Para que la función sea mínima, se debe buscar el mínimo número de subcubos que cubren todos los unos. Esto se logra buscando los subcubos de mayor tamaño posible, sin importar que se traslapen.

Mapas de Karnaugh, Minimización

✓ Ejemplo:



✓ Representa: $F(A, B, C, D) = B + C + \overline{D}$

Mapas de Karnaugh, AND de OR

- ✓ Una función se puede expresar también como el producto (AND) de los subcubos necesarios para cubrir todos los ceros del MK.
- ✓ Ejemplo: Minimizar: $F(A, B, C, D) = \prod(0, 2, 5, 8, 10, 13, 14)$
- ✓ Para minimizar se agrupan ceros del mapa:

		AB			
		00	01	11	10
CD	00	0	1	1	0
	01	1	0	0	1
	11	1	1	1	1
	10	0	1	0	0

- ✓ Resultado:
$$F(A, B, C, D) = (B + D) \cdot (\bar{B} + C + \bar{D}) \cdot (\bar{A} + \bar{C} + D)$$

Ejemplo

- ✓ Otro Ejemplo: La universidad quiere poner un letrero luminoso en el techo del edificio más alto del campus. El letrero se compone de las letras U, T, F, S y M. Cada letra se ilumina cuando su entrada toma el valor 1 y se apaga cuando su entrada toma el valor 0.
- ✓ El letrero tiene una entrada de control que permite controlar con un número de secuencia si cada una de las letras está iluminada o apagada. El letrero tiene la siguiente secuencia de iluminación:

UTFSM
Etapa 0
Todas las letras están apagadas.

UTFSM
Etapa 2
T y S están prendidas

UTFSM
Etapa 4
Todas las letras están prendidas

UTFSM
Etapa 1
U y M están prendidas

UTFSM
Etapa 3
F está prendida

Ejemplo

✓ Preguntas:

1. ¿Cuántos bits son necesarios para controlar el número de etapa de la secuencia de iluminación del letrero UTFSM?
2. ¿Cuáles son las combinaciones superfluas?
3. Determinar la tabla de iluminación de cada letrero en función de los bits de control del número de etapa.
4. Deducir de las tablas las ecuaciones booleanas que corresponden a cada letra.
5. Determinar la tabla de estado Q_n/Q_{n+1} correspondiente a la secuencia de iluminación del letrero.
6. Usando FF del tipo J-K construir el circuito digital de control de la secuencia de animación del letrero.

Quine y McCluskey

- ✓ El diseño de VLSI requiere de apoyo computacional.
- ✓ La minimización tiene un impacto directo en el costo de un procesador.
- ✓ La lógica de cualquier procesador moderno tiene cientos de variables, por lo tanto no sirve el método de los MK.
- ✓ ¿Solución?

Quine y McCluskey

- ✓ El método de Quine y McCluskey es una técnica tabular. Resulta fácil de programar, con lo que se logra una herramienta automática para la obtención de expresiones de conmutación mínimas.
- ✓ Una función de conmutación $F(A,B,C\dots)$ se puede expresar como la suma de términos donde cada término está compuesto de factores.
- ✓ Ejemplo: $F(A, B, C) = A \cdot B \cdot C + \bar{B} \cdot C$
- ✓ Se dice que la función contiene un término si cuando éste vale 1, la función vale 1. Por ejemplo, la función $F(A,B,C) = A \cdot B + C$ contiene dos términos, $A \cdot B$ y C .

Quine y McCluskey

- ✓ Se define como implicante primo a un término que está contenido en la función y que la eliminación de cualquier de sus literales genera un nuevo término que no está contenido en la función dada.
- ✓ En la función $F(A,B,C) = A \cdot B + C$, hay dos términos implicantes primos: $A \cdot B$ y C .
- ✓ En cambio la función: $F(A,B,C) = A \cdot B \cdot C + A + B \cdot C$. Tiene 3 términos, pero solo 2 de ellos son implicantes primos, esto porque el término $A \cdot B \cdot C$ no es implicante primo, ya que si se elimina el literal A , queda el término $B \cdot C$ que ya existe en la función.

Quine y McCluskey

- ✓ Se puede observar que los implicants primos corresponden a los subcubos en un mapa de Karnaugh. Por lo tanto, la ecuación minimizada tendrá tantos términos como **implicants primos** tenga la función.
- ✓ Los algoritmos computacionales para la minimización de funciones, se basan en la búsqueda automatizada de implicants primos.
- ✓ El método de Quine y McCluskey genera el conjunto de implicants primos de una función dada.

Quine y McCluskey

✓ Pasos:

1. Transformar la función a su forma canónica OR de AND.
2. Luego se representa cada término de la forma binaria.
3. Agrupar términos en función de la cantidad de unos que tengan.
4. Cada grupo (que representa la cantidad de 1's del término), se vuelve a agrupar con algún grupo adyacente buscando diferencias en un solo bit. El bit en que difieren es reemplazado por “-”.
5. Se vuelve a aplicar el paso anterior. Para la adyacencia se debe considerar que el símbolo “-” se encuentra en la misma posición.
6. Finalmente se deben cubrir todos los términos de la función original, utilizando el mínimo número de implicantes primos.

Quine y McCluskey

✓ Ejemplo: $F(x, y, z, v) = \sum (0, 2, 3, 5, 6, 7, 8, 9)$

✓ Describimos en forma binaria:

0	0000
2	0010
3	0011
5	0101
6	0110
7	0111
8	1000
9	1001

Quine y McCluskey

- ✓ Se agrupan los términos en función a la cantidad de 1's que tienen:

0	0000
2	0010
3	0011
5	0101
6	0110
7	0111
8	1000
9	1001



0 unos

0	0000
---	------

1 unos

2	0010
8	1000

2 unos

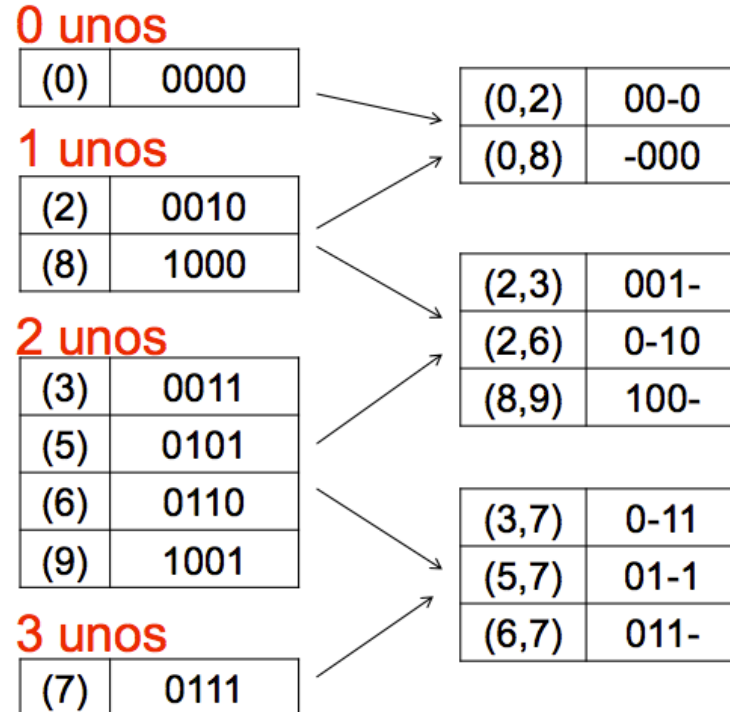
3	0011
5	0101
6	0110
9	1001

3 unos

7	0111
---	------

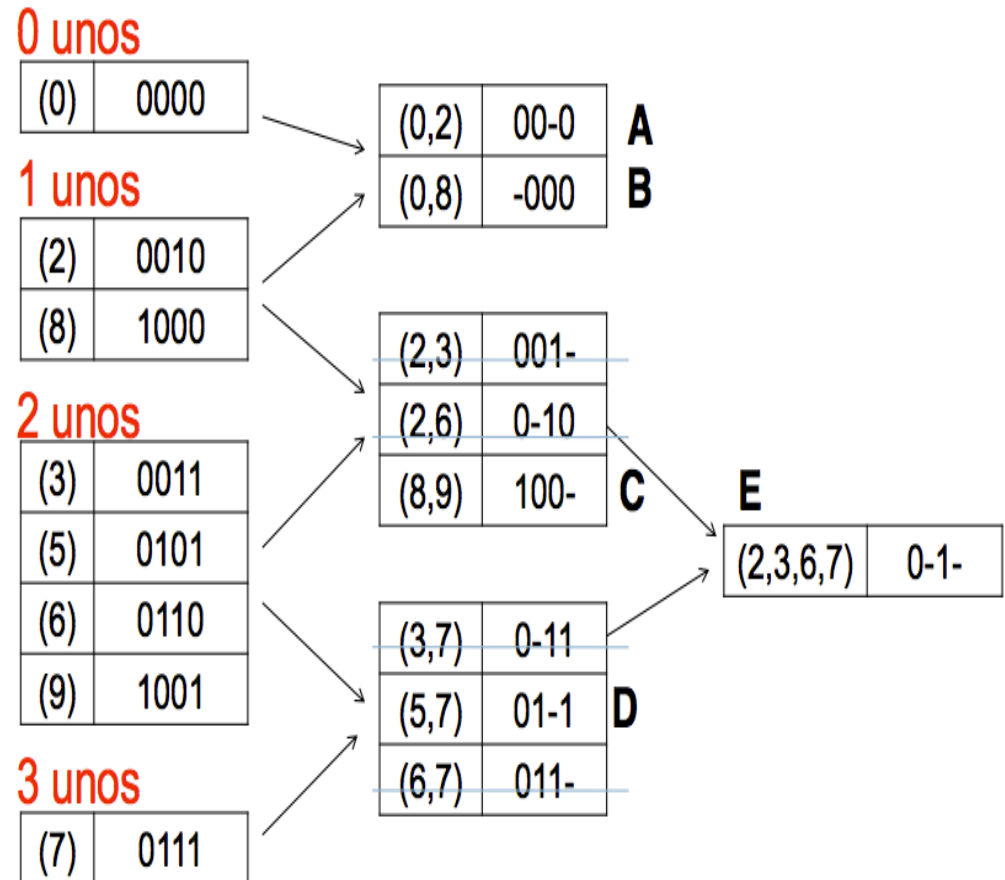
Quine y McCluskey

- ✓ Se reagrupan los grupos adyacentes buscando diferencias en un solo bit y reemplazando el bit en que difieren con un “-”



Quine y McCluskey

- ✓ Se repite y se vuelve a reagrupar considerando que el símbolo “-” se encuentre en la misma posición.



Quine y McCluskey

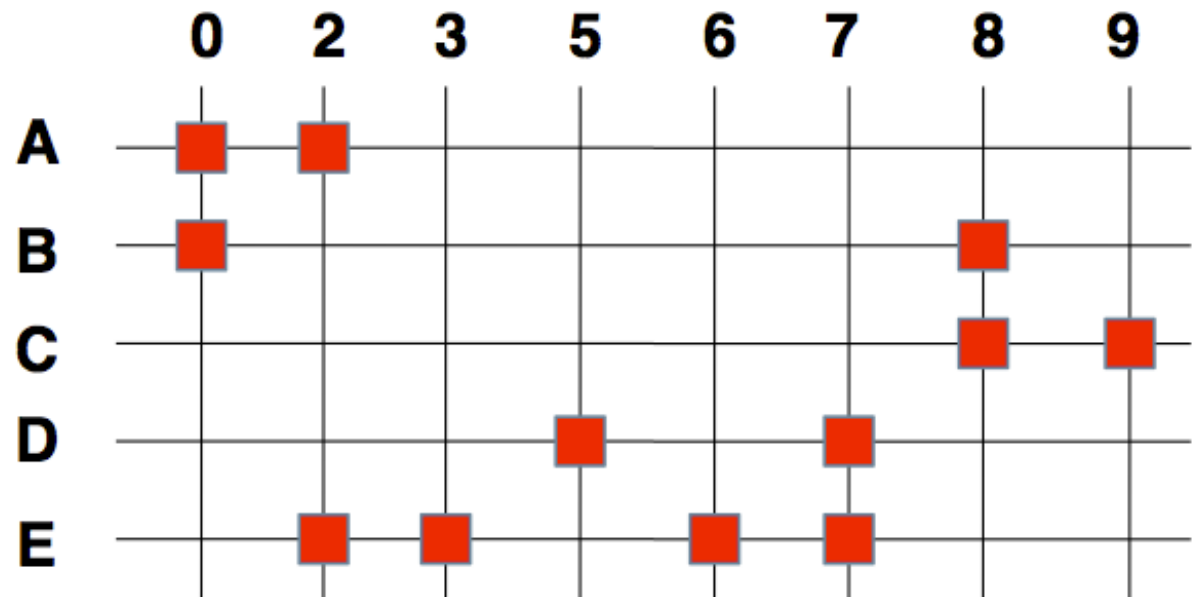
- ✓ Los siguientes términos corresponden a los implicantes primos:

(0,2)	00-0	A
(0,8)	-000	B
(8,9)	100-	C
(5,7)	01-1	D
(2,3,6,7)	0-1-	E

Quine y McCluskey

- ✓ Se debe cubrir todos los términos de la función original utilizando el número de implicants primos.

(0,2)	00-0
(0,8)	-000
(8,9)	100-
(5,7)	01-1
(2,3,6,7)	0-1-



Quine y McCluskey

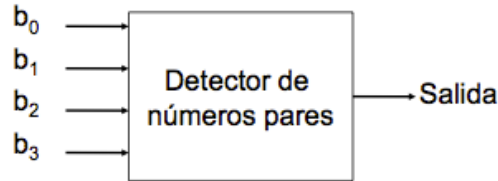
- ✓ Finalmente los implicants primos que representan a todos los términos pueden ser:

$$F(x, y, z, v) = C + D + E + A = x \cdot \bar{y} \cdot \bar{z} + \bar{x} \cdot y \cdot v + \bar{x} \cdot z + \bar{x} \cdot \bar{y} \cdot \bar{v}$$

$$F(x, y, z, v) = C + D + E + B = x \cdot \bar{y} \cdot \bar{z} + \bar{x} \cdot y \cdot v + \bar{x} \cdot z + \bar{y} \cdot \bar{z} \cdot \bar{v}$$

Combinaciones Superfluas

- ✓ Se denominan así a aquellas combinaciones de variables de entrada que por lógica no pueden ocurrir.
- ✓ Consideremos el siguiente Ejemplo: Sistema combinacional que detecta los números pares de un código BCD.



b_3	b_2	b_1	b_0	S	b_3	b_2	b_1	b_0	S
0	0	0	0	1	1	0	0	0	1
0	0	0	1	0	1	0	0	1	0
0	0	1	0	1	1	0	1	0	x
0	0	1	1	0	1	0	1	1	x
0	1	0	0	1	1	1	0	0	x
0	1	0	1	0	1	1	0	1	x
0	1	1	0	1	1	1	1	0	x
0	1	1	1	0	1	1	1	1	x

Nunca se pueden presentar estas combinaciones

Combinaciones Superfluas

- ✓ Las celdas superfluas pueden ser consideradas como ceros o bien como unos.
- ✓ De esta manera se agrupa según conveniencia, para obtener la menor cantidad de subcubos y que estos sean de mayor tamaño posible.

		b_3b_2			
		00	01	11	10
b_1b_0	00	1	1	-	1
	01	1	1	-	1
	11	1	1	-	-
	10	0	1	-	-

Combinaciones Superfluas

b_3b_2 b_1b_0					
		00	01	11	10
00	1	1	-	1	
01	1	1	-	1	
11	1	1	-	-	
10	0	1	-	-	

✓ Resultado

$$F(b_3, b_2, b_1, b_0) = \bar{b}_1 + b_0 + b_2$$

Ejercicio

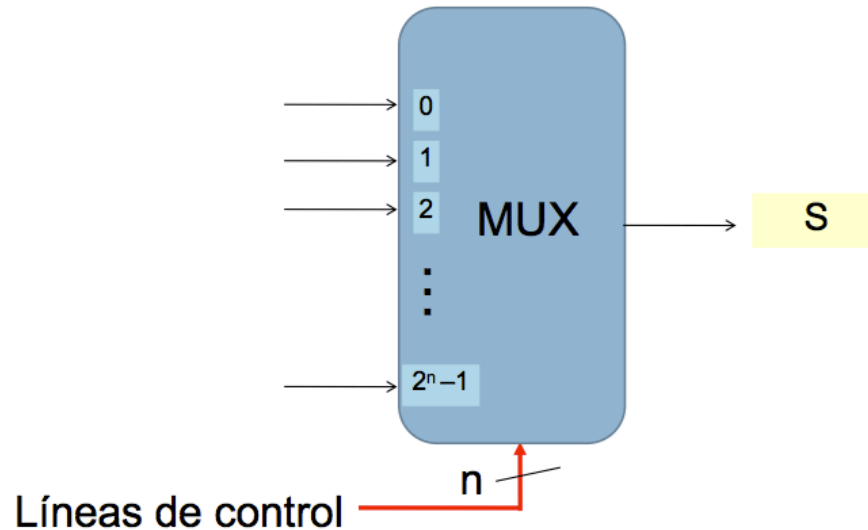
- ✓ Los circuitos combinacionales puede ser utilizados para construir bloques más complejos. Entre ellos están los Multiplexores y Decodificadores.
- ✓ Un Multiplexor o MUX es un circuito combinacional que permite seleccionar alguna de sus entradas.
- ✓ Los MUX son intensamente utilizados en arquitectura debido a que seleccionan flujos de información entre distintos registros.
- ✓ Un MUX de n entradas, tiene una salida y $\log_2(n)$ líneas de control.

Componentes

✓ Minimizar:

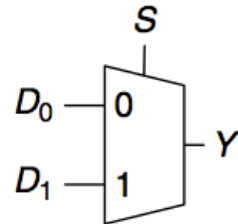
$$F(a,b,c,d) = \sum (0,2,6,8,9,10,11,13,15)$$

Componentes



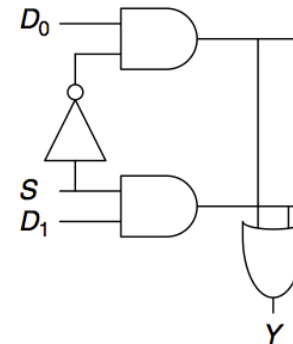
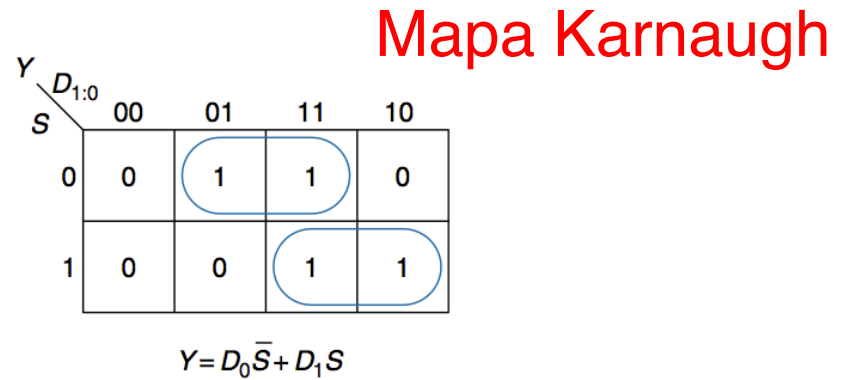
- ✓ Considere un MUX 2:1, el cual tiene dos bits de entrada D0 y D1, una línea de control S y una salida Y. Si $S = 0$, $Y = D0$. Si $S = 1$, $Y = D1$.

Componentes



S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Tabla de Verdad



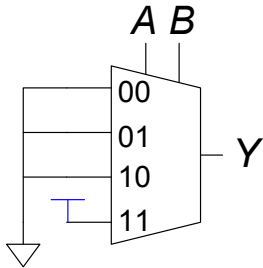
Circuito Combinacional

Componentes

✓ Otro Ejemplo:

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

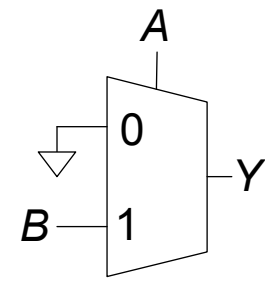
$$Y = AB$$



$$Y = AB$$

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

A	Y
0	0
1	B



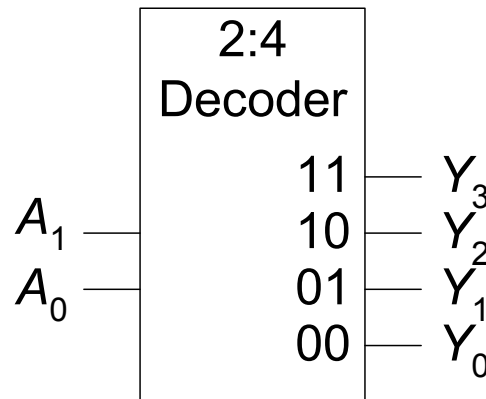
Componentes

✓ Otro Ejemplo: Construir

$$F(a,b) = (\overline{a}\overline{b}) + (\overline{a}bc) + (a\overline{b}\overline{c})$$

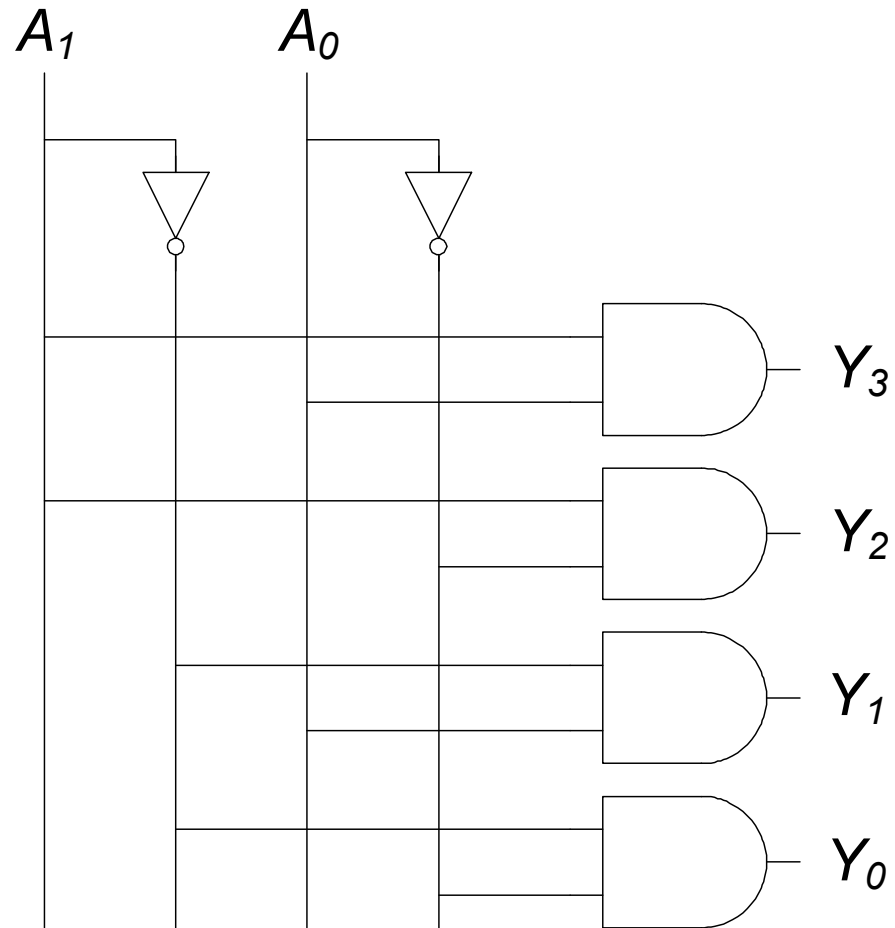
Componentes

- ✓ Decodificadores: N entradas, 2^N salidas.
- ✓ Solo 1 salida actividad por vez.



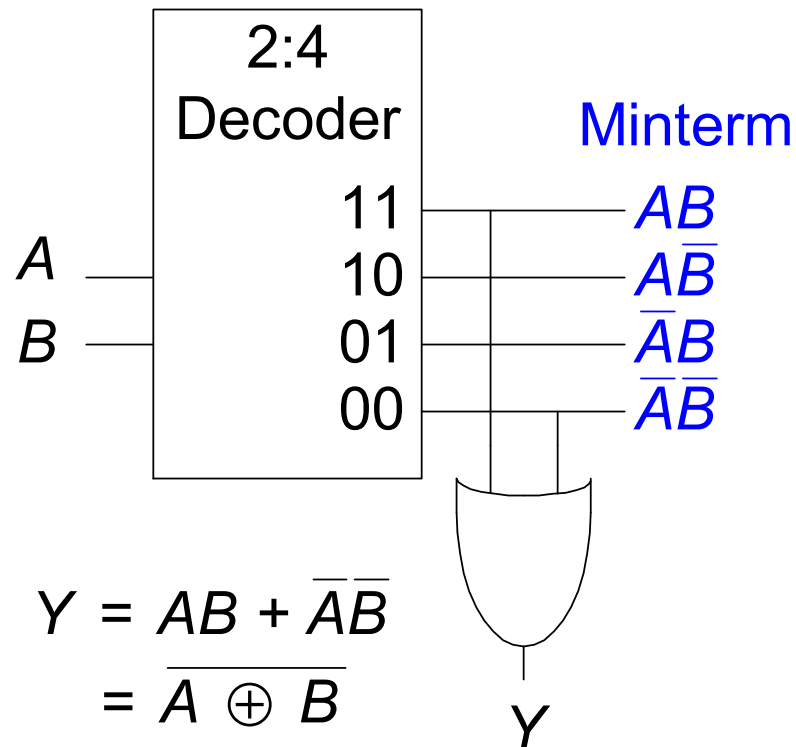
A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Componentes



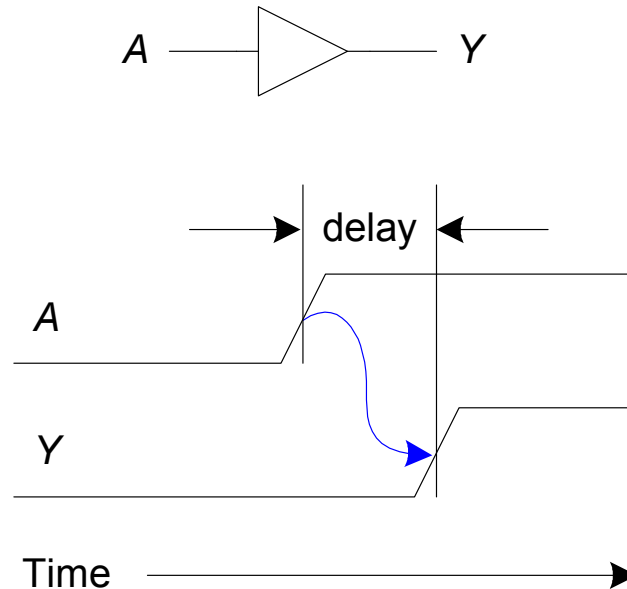
Componentes

- ✓ Algunas funciones lógicas utilizando decodificadores:



Tiempos

- ✓ Idealmente los circuitos deben construirse con pocas compuertas.
- ✓ Uno de los mayores desafíos es construir circuitos rápidos.

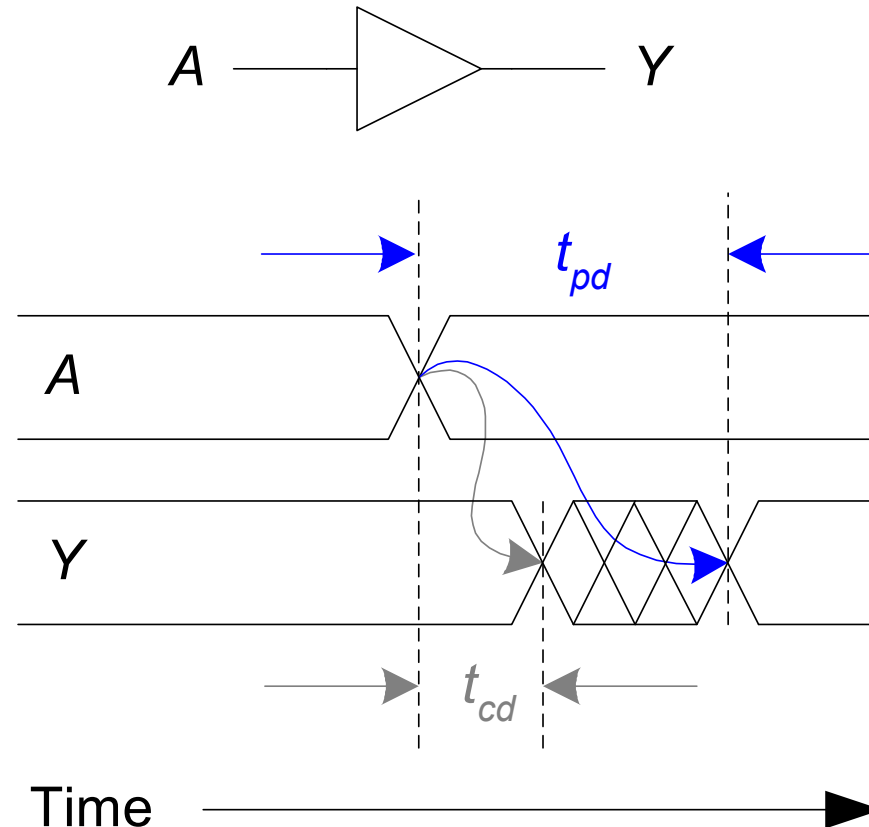


Tiempos

- ✓ Los circuitos combinacional presentan un delay de propagación. Corresponde al tiempo máximo desde que una entrada cambia hasta que las salidas toman su valor final.
- ✓ También existe el delay de contaminación, el cual corresponde al mínimo tiempo desde que una entrada cambia hasta que cualquier salida comienza a cambiar de valor.

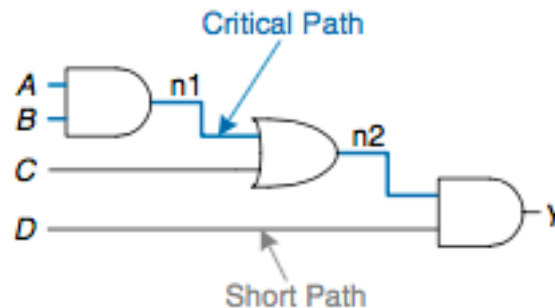
Tiempos

- **Propagation delay:** t_{pd} = max delay from input to output
- **Contamination delay:** t_{cd} = min delay from input to output

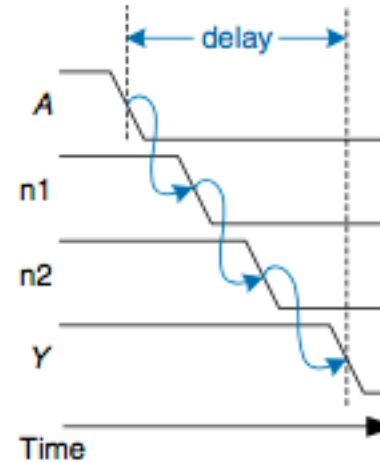
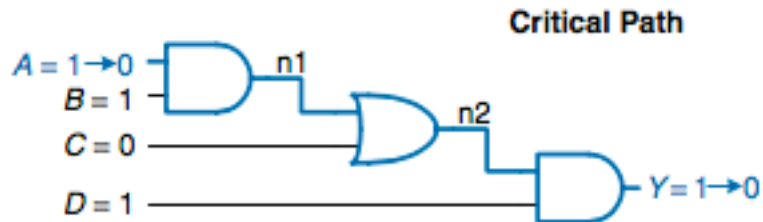


Tiempos

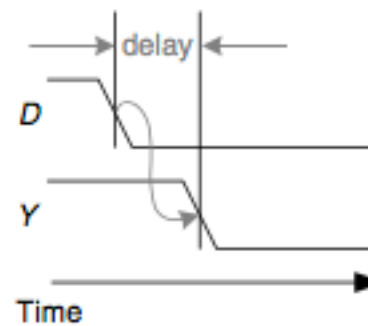
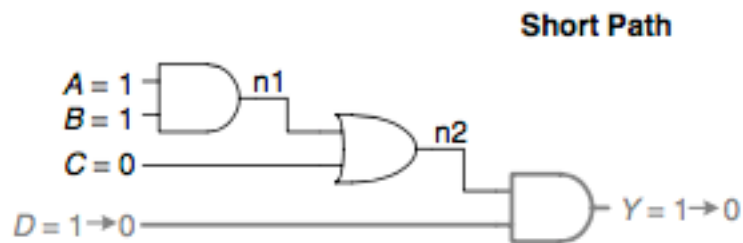
- ✓ Delay es causado por:
 - ✓ Capacitores y resistencias en los circuitos.
 - ✓ Limitaciones en la velocidad de la luz.
- ✓ Para calcular tpd y tcd debemos analizar el camino más largo y el más corto que sigue un circuito:



Tiempos



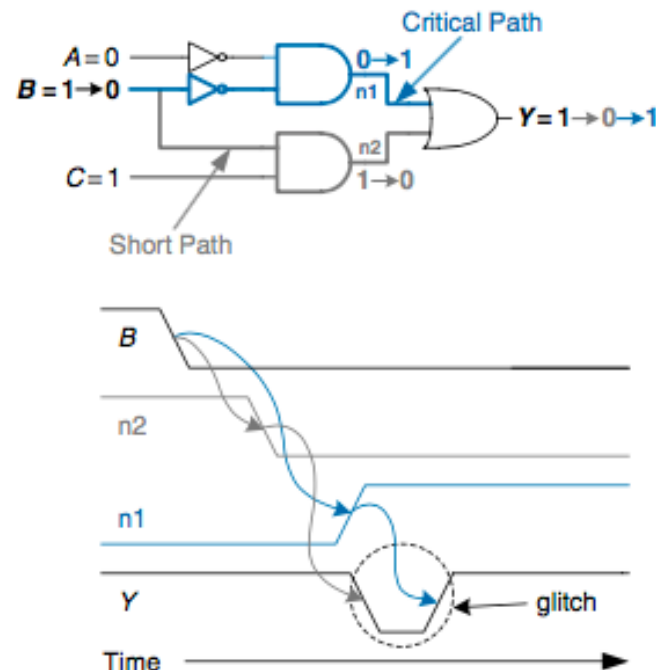
✓ $tpd = 2tpd_{and} + tpd_{or}$



✓ $tcd = tcd_{and}$

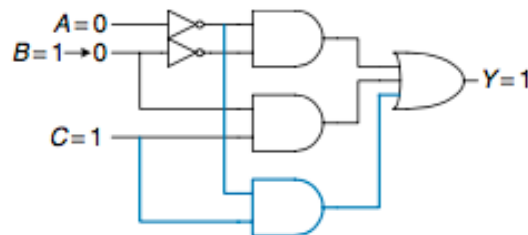
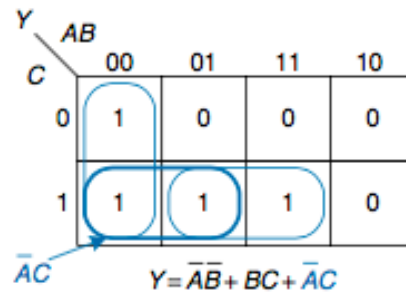
Glitches

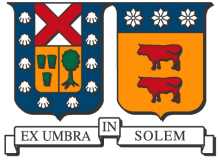
- ✓ Cuando el cambio de una entrada genera múltiples cambios en las salidas.
- ✓ Revisemos el siguiente ejemplo: $A = 0$, $C = 1$ y se genera la transición de 1 a 0 de B .



Glitches

- ✓ ¿Se puede evitar? Si, para eso revisamos el M.K y el posible comportamiento de los implicantes primos.





UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
CAMPUS SAN JOAQUÍN

Arquitectura y Organización de Computadores

2023