جبر بول

Boolean Algebra

جبر بول

- **عامل اصلي قوت سيستم‌هاي ديجيتال:**
  - جامعيت و قدرت فرمولاسيون رياضي جبر بول
- **مثال:**

IF the garage door is open
AND the car is running
THEN the car can be backed out of the garage

هر دو شرط:
The garage door is open و
The car is running
باید true باشند تا بتوان ماشین را از گاراژ بیرون آورد.

Digital Systems: Boolean Algebra and Logical Operations

- **In Boolean algebra:**
  - Values: 0,1
  - 0: if a logic statement is false,
  - 1: if a logic statement is true.
- **Operations: AND, OR, NOT**

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

بررسی مثال قبلی

IF the garage door is open
AND the car is running
THEN the car can be backed out of the garage

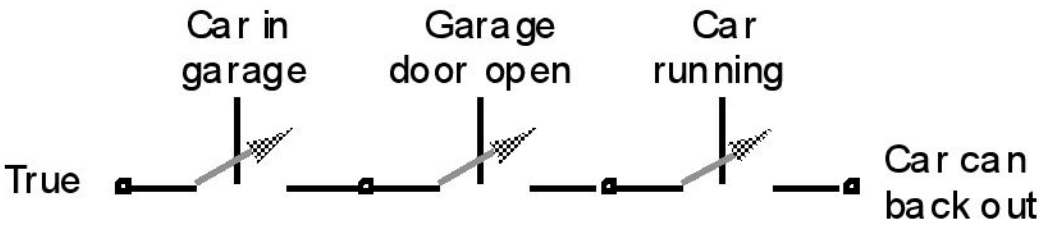
door open?	car running?	back out car?
false/0	false/0	false/0
false/0	true/1	false/0
true/1	false/0	false/0
true/1	true/1	TRUE/1

- 2 گزینه برای هر متغیر
- 2^n گزینه برای n متغیر
- هر متغیر می‌تواند توسط یک سویچ پیاده‌سازی شود
- چینش سویچ‌ها توسط نوع عملگرها تعیین می‌شود

سويچ‌ها: عملگرهاي منطقي

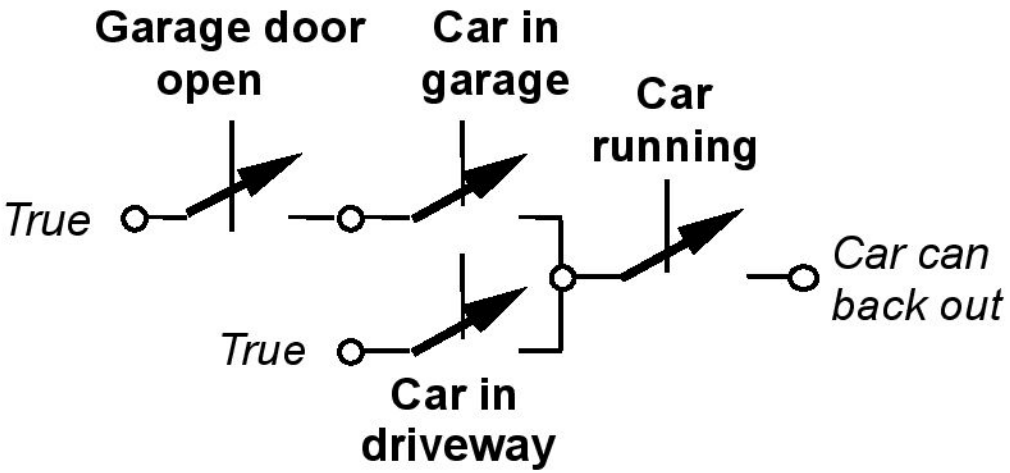
EXAMPLE:

IF car in garage
AND garage door open
AND car running
THEN back out car



EXAMPLE:

IF (car in driveway
OR (car in garage
AND garage door open))
AND car running
THEN can back out car



Binary Logic

- Deals with **binary variables** that take 2 discrete values (0 and 1), and **logic operations**
- Basic logic **operations**:
  - AND, OR, NOT
- Binary/logic **variables** are typically represented by letters: A,B,C,…,X,Y,Z

Binary Logic Function

F(vars) = *expression*

set of binary variables

- Operators (+, •, ')
- Variables
- Constants (0, 1)
- Groupings (parenthesis)

Example:

$F(a,b) = a' \cdot b + b'$

$G(x,y,z) = x \cdot (y+z')$

Boolean Logic Operators

- AND (also $\cdot, \wedge$)
- OR (also $+, \vee$)
- NOT (also $'$, $\bar{}$)

Binary

Unary

- $F(a,b) = a \cdot b$, reads F is 1 if and only if a=b=1
- $G(a,b) = a+b$, reads G is 1 if either a=1 or b=1 or both
- $H(a) = a'$, reads H is 1 if a=0

Boolean Operators (cont.)

- **1-bit logic AND resembles binary multiplication:**

  $0 \cdot 0 = 0, \quad 0 \cdot 1 = 0,$

  $1 \cdot 0 = 0, \quad 1 \cdot 1 = 1$

- **1-bit logic OR resembles binary addition, except for one operation:**

  $0 + 0 = 0, \quad 0 + 1 = 1,$

  $1 + 0 = 1, \quad 1 + 1 = 1 \ (\neq 10_2)$

Truth Tables for logic operators

Truth table:

A tabular form that uniquely represents the relationship between the input variables of a function and its output

2-Input AND

A	B	F=A·B
0	0	0
0	1	0
1	0	0
1	1	1

2-Input OR

A	B	F=A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT

A	F=A'
0	1
1	0

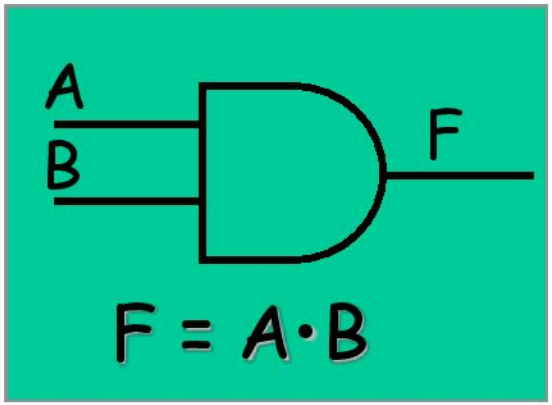
Truth Tables (cont.)

- **Q:**
  - Let a function F() depend on *n* variables. How many rows are there in the truth table of F() ?
- **A:**
  - 2^n rows, since there are 2^n possible binary patterns/combinations for the *n* variables

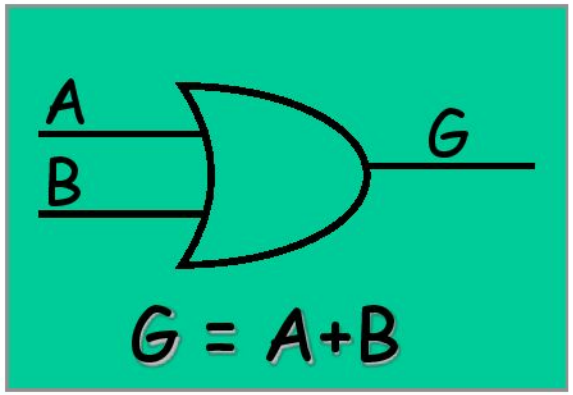
Logic Gates

- Logic gates are abstractions of electronic circuit components that operate on one or more input signals to produce an output signal.

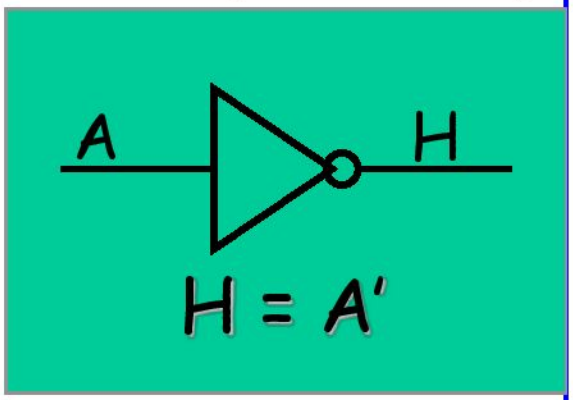
2-Input AND



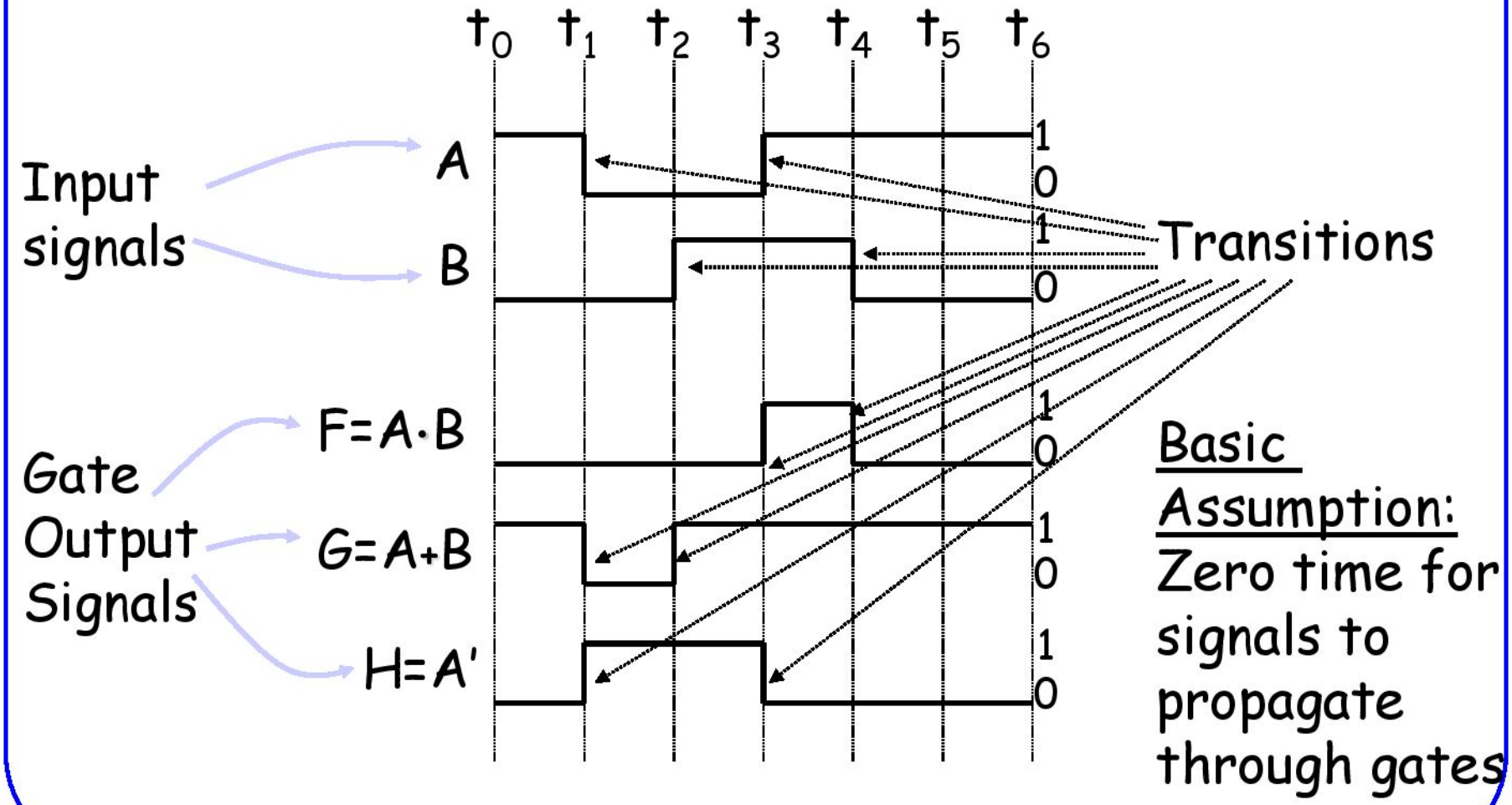
2-Input OR



NOT (Inverter)

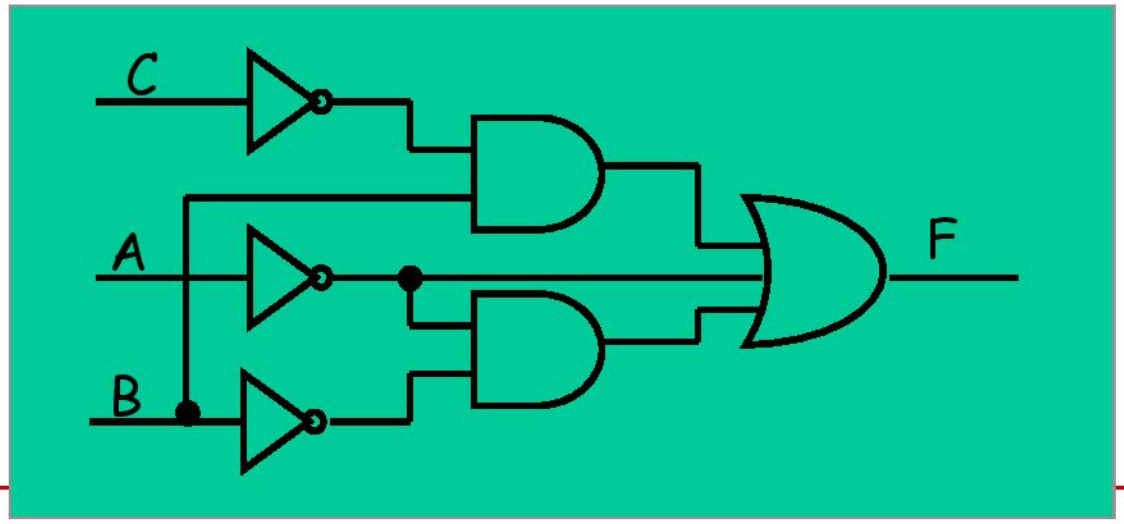


Timing Diagram



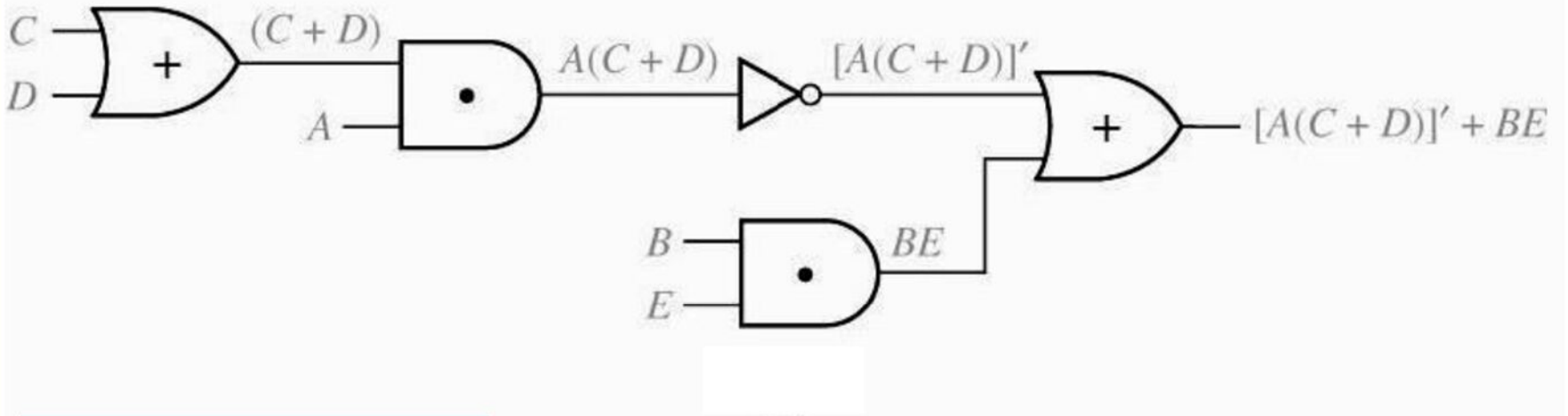
Combinational Logic Circuit from Logic Function

- **Combinational Circuit Design:**
- **F = A' + B·C' + A'·B'**
  - Connect input signals and logic gates:
    - Circuit input signals → from function variables (A, B, C)
    - Circuit output signal → function output (F)
    - Logic gates → from logic operations



Logic Evaluation

Circuit of logic gates :



Logic Expression : $[A(C+D)]'+BE$

Logic Evaluation : If A=B=C=1, D=E=0, then:

$$[A(C+D)]'+BE = [1(1+0)]'+1\cdot 0 = [1(1)]'+0 = 0+0 = 0$$

Combinational Logic Optimization

- Consider functions F and G:

  F=A' + B·C' + A'·B' and

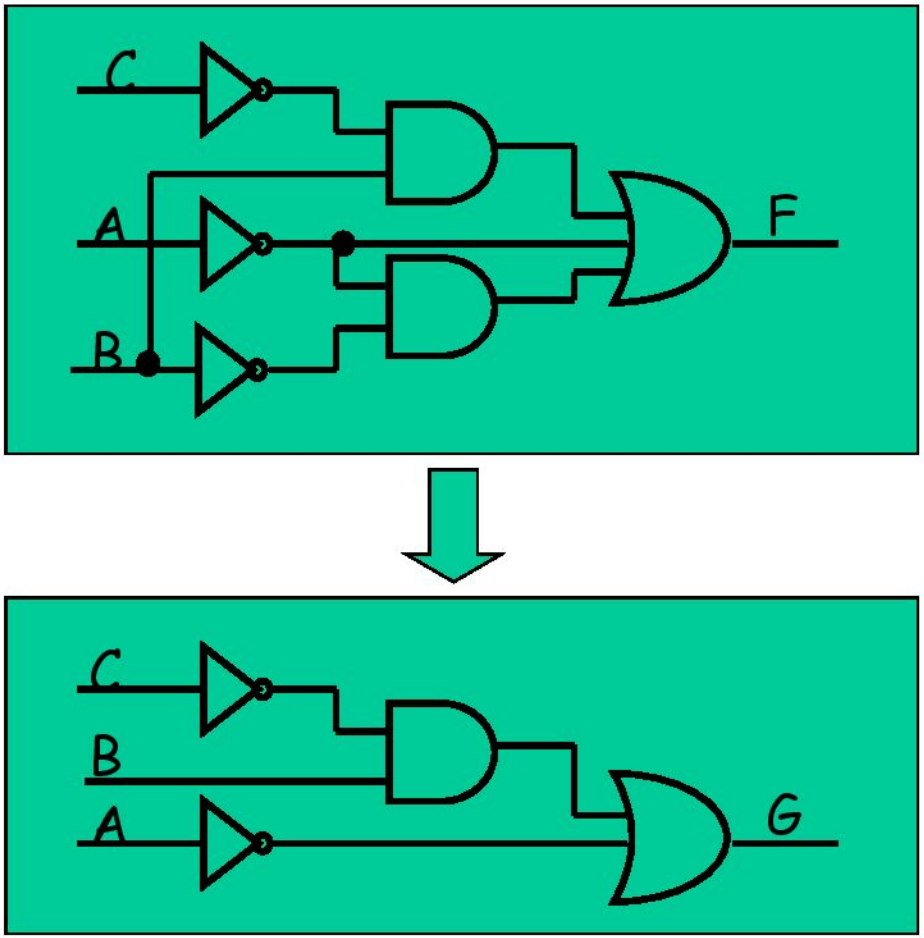
  G=A' + B·C'

- They seem different but let's observe their truth tables
- They are identical → same function
- In order to design a cost-effective and efficient circuit, we must minimize
  - the circuit's size
    - area
  - propagation delay
    - time required for an input signal change to be observed at the output line
- Use G to implement the logic circuit
  - Fewer components

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

Combinational Logic Optimization

F=A’ + B•C′ + A′•B′



G=A’ + B•C′

Equivalence Checking Using Truth Table

$$AB'+C \stackrel{?}{=} (A+C)(B'+C)$$

n variable needs $\underbrace{2\times 2\times 2\times \ldots}_{\text{n times}} = 2^n$ rows

A B C	B'	AB'	AB' + C	A + C	B' + C	(A + C) (B' + C)
0 0 0	1	0	0	0	1	0
0 0 1	1	0	1	1	1	1
0 1 0	0	0	0	0	0	0
0 1 1	0	0	1	1	1	1
1 0 0	1	1	1	1	1	1
1 0 1	1	1	1	1	1	1
1 1 0	0	0	0	1	0	0
1 1 1	0	0	1	1	1	1

Boolean Algebra

- VERY nice machinery used to manipulate (simplify) Boolean functions
- George Boole (1815-1864): "An investigation of the laws of thought"
- Terminology:
  - *Literal:* A variable or its complement

    A, B', x'
  - *Product term:* literals connected by •

    X.Y

    A.B'.C.D,
  - *Sum term:* literals connected by +

    A+B'

    A'+B'+C

Boolean Algebra Properties

Let X: Boolean variable,

0,1: constants

1. X + 0 = X -- Zero Axiom
2. X • 1 = X -- Unit Axiom
3. X + 1 = 1 -- Unit Property
4. X • 0 = 0 -- Zero Property

A	B	F=A+B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	F=A • B
0	0	0
0	1	0
1	0	0
1	1	1

Example: $(AB'+D)E+1=1$

Boolean Algebra Properties (cont.)

Let X: Boolean variable,
0,1: constants

1. X + X = X -- Idempotent
2. X • X = X -- Idempotent
3. X + X' = 1 -- Complement
4. X • X' = 0 -- Complement
5. (X')' = X -- Involution

A	B	F=A+B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	F=A • B
0	0	0
0	1	0
1	0	0
1	1	1

Example:

$$(AB'+D)(AB'+D)'=0$$

The Duality Principle

- The dual of an expression is obtained by exchanging (• and +), and (1 and 0) in it,
  - provided that the precedence of operations is not changed.
- Do not exchange x with x′
- Example:
  - Find H(x,y,z), the dual of F(x,y,z) = x′yz′ + x′y′z
  - H = (x′+y+z′).(x′+y′+z)
- Dual is not always equal to the original expression

- If a Boolean equation/equality is valid, its dual is also valid

The Duality Principle (cont.)

With respect to duality, identities 1 – 8 have the following relationships:

1. $X + 0 = X$	2. $X \cdot 1 = X$	(dual of 1)
3. $X + 1 = 1$	4. $X \cdot 0 = 0$	(dual of 3)
5. $X + X = X$	6. $X \cdot X = X$	(dual of 5)
7. $X + X' = 1$	8. $X \cdot X' = 0$	(dual of 7)

More Boolean Algebra Properties

Let X,Y, and Z: Boolean variables

10. $X + Y = Y + X$ 11. $X \cdot Y = Y \cdot X$ -- Commutative

12. $X + (Y+Z) = (X+Y) + Z$ 13. $X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$ -- Associative

14. $X \cdot (Y+Z) = X \cdot Y + X \cdot Z$ 15. $X+(Y \cdot Z) = (X+Y) \cdot (X+Z)$ -- Distributive

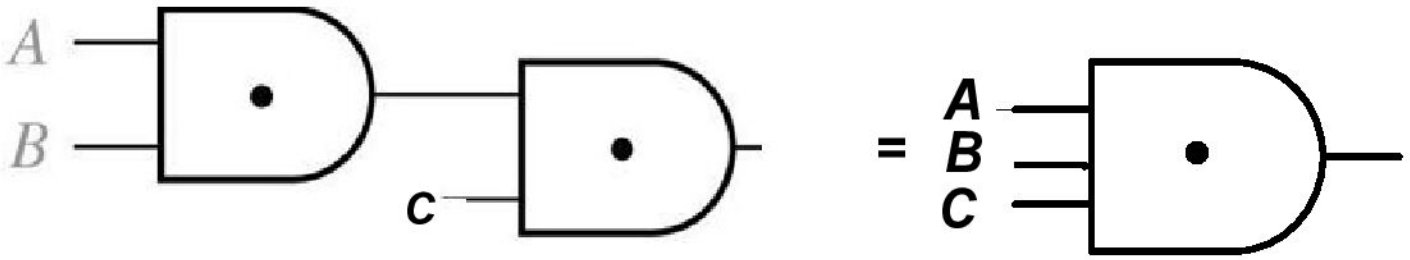
16. $(X + Y)' = X' \cdot Y'$ 17. $(X \cdot Y)' = X' + Y'$ -- DeMorgan's Laws

In general:

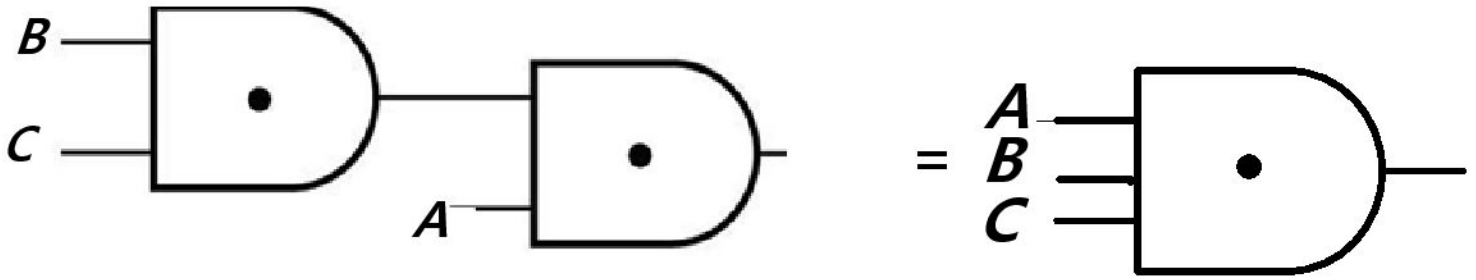
$$(X_1 + X_2 + \ldots + X_n)' = X_1' \cdot X_2' \cdot \ldots \cdot X_n'$$

$$(X_1 \cdot X_2 \cdot \ldots \cdot X_n)' = X_1' + X_2' + \ldots + X_n'$$

Associative Laws for AND

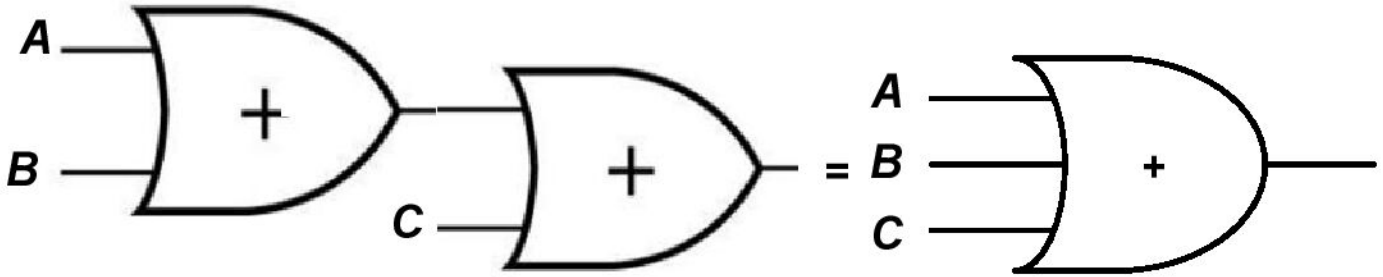


$(AB)C=ABC$



$A(BC)=ABC$

Associative Laws for OR



$(A+B)+C=A+B+C$

Proof of Associative Law

Associative Laws:

$$(XY)Z = X(YZ) = XYZ$$

$$(X+Y)+Z = X+(Y+Z) = X+Y+Z$$

Proof of Associative Law for AND:

X	Y	Z	XY	YZ	(XY)Z	X(YZ)
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	1	0	0	0
1	1	1	1	1	1	1

Distributive Law

Distributive Laws:

$$X(Y+Z) = XY + XZ$$

$$X + YZ = (X+Y)(X+Z)$$

Valid only for Boolean algebra not for ordinary algebra

Proof of the second law:

$$(X+Y)(X+Z) = X(X+Z) + Y(X+Z) = XX + XZ + YX + YZ$$

$$= X + XZ + XY + YZ = X \cdot 1 + XZ + XY + YZ$$

$$= X(1 + Z + Y) + YZ = X \cdot 1 + YZ = X + YZ$$

Note that the first law is used to prove the second one. But considering duality, there was no need to prove the second one!

Absorption Property (Covering)

1. $x + x \cdot y = x$
2. $x \cdot (x+y) = x$ (dual)

- Proof:

$$x + x \cdot y = x \cdot 1 + x \cdot y$$
$$= x \cdot (1+y)$$
$$= x \cdot 1$$
$$= x$$

QED (2 true by duality)

Absorption Property (Covering)

1. $x + x' \cdot y = x + y$
2. $x \cdot (x' + y) = x\ y$ (dual)

- Proof of 2:

$$x \cdot (x' + y) = x \cdot x' + x \cdot y$$
$$= 0 + (x \cdot y)$$
$$= x \cdot y$$

QED (1 true by duality)

DeMorgan's Laws

$$(X+Y)' = X'Y'$$

$$(XY)' = X'+Y'$$

Proof

X	Y	X'	Y'	X + Y	(X + Y)'	X' Y'	XY	(XY)'	X' + Y'
0	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	0	0	1	1
1	0	0	1	1	0	0	0	1	1
1	1	0	0	1	0	0	1	0	0

DeMorgan's Laws for *n* variables

$$(X_1 + X_2 + X_3 + \ldots + X_n)' = X_1' X_2' X_3' \ldots X_n'$$

$$(X_1 X_2 X_3 \ldots X_n)' = X_1' + X_2' + X_3' + \ldots + X_n'$$

Example

$$(X_1 + X_2 + X_3)' = (X_1 + X_2)' X_3' = X_1' X_2' X_3'$$

Consensus Theorem

1. $xy + x'z + yz = xy + x'z$
2. $(x+y)\cdot(x'+z)\cdot(y+z) = (x+y)\cdot(x'+z)$ (dual)

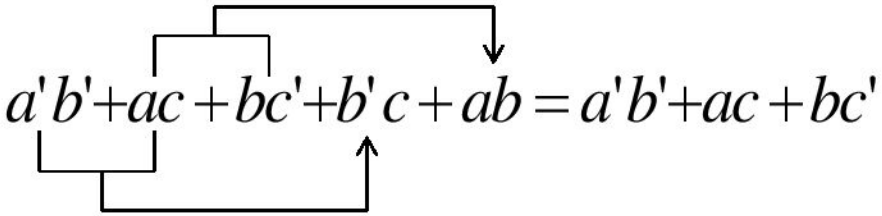
Proof:

$$xy + x'z + yz = xy + x'z + (x+x')yz$$
$$= xy + x'z + xyz + x'yz$$
$$= (xy + xyz) + (x'z + x'zy)$$
$$= xy + x'z$$

QED (2 true by duality).

Consensus Theorem

Example:



Consensus Theorem

Example:

$$A'C'D + \cancel{A'BD} + BCD + \cancel{ABC} + ACD'$$

Consensus Theorem

Example:

Reducing an expression:
Add one term and eliminate two!

$$F = ABCD + B'CDE + A'B' + BCE'$$

$$F = ABCD + B'CDE + A'B' + BCE' + ACDE$$

Consensus term added

Final expression $F = A'B' + BCE' + ACDE$

Algebraic Manipulation

- Boolean algebra is a useful tool for simplifying digital circuits.
- Why do simplification?
  1. Fan-in (number of gate inputs) is limited in some technologies:
     - Reduce the number of literals (gate inputs)
  2. Number of gates influences manufacturing costs
     - Reduce the number of gates
  3. Fewer levels of gates implies reduced signal propagation delays and faster design
     - Reduce number of levels of gates (logical depth)

Algebraic Manipulation

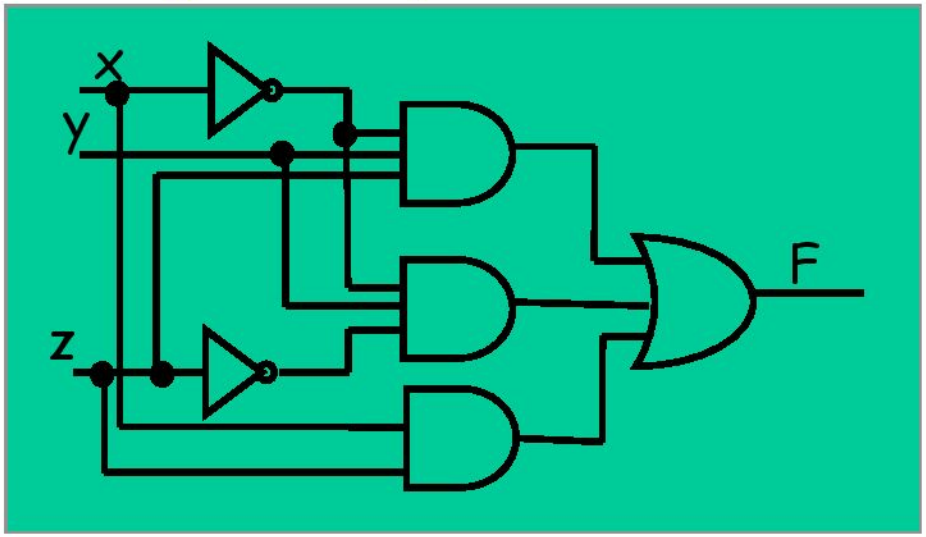
Example: Simplify F = x′yz + x′yz′ + xz.

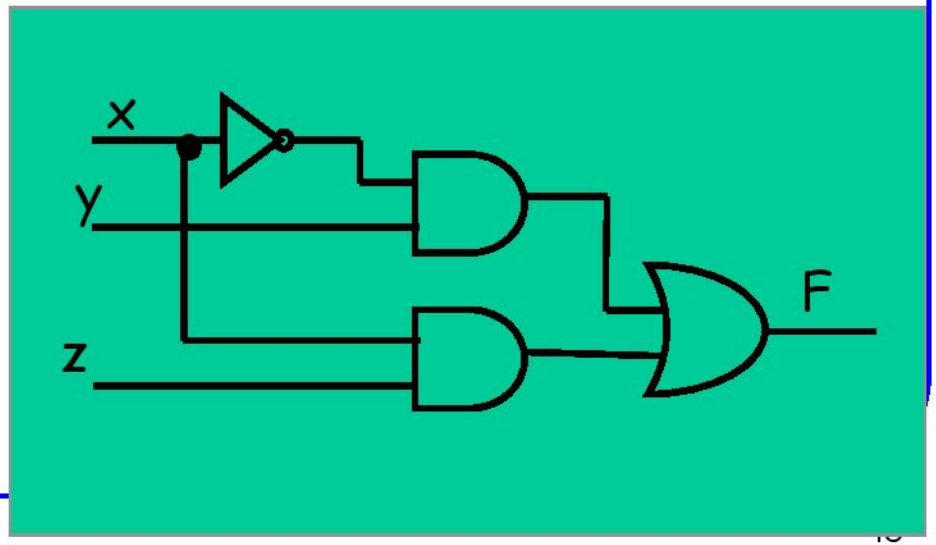
F= x′yz + x′yz′ + xz

= x′y(z+z′) + xz

= x′y•1 + xz

= x′y + xz





Sum of Products (SoP)

Sum of products form: $AB'+CD'E+AC'E$

Still considered to be in sum of products form:

$ABC'+DEFG+H$

$A+B'+C+D'E$

Not in sum of products form: $(A+B)CD+EF$

Multiplying out and eliminating redundant terms

$$(A+BC)(A+D+E) = A+AD+AE+ABC+BCD+BCE$$
$$= A(1+D+E+BC)+BCD+BCE$$
$$= A+BCD+BCE$$

Product of Sums (PoS)

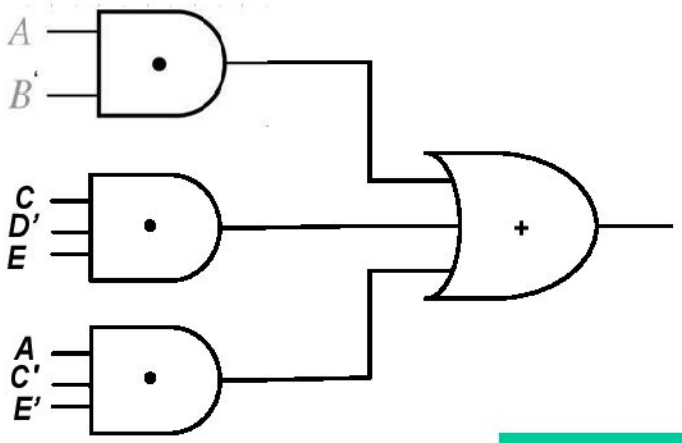
Product of sums form: $(A+B')(C+D'+E)(A+C'+E')$

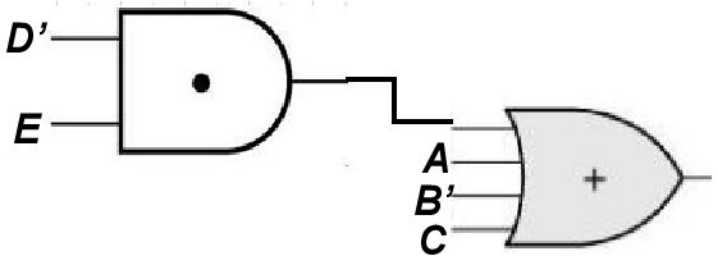
Still considered to be in product of sums form:

$(A+B)(C+D+E)F$

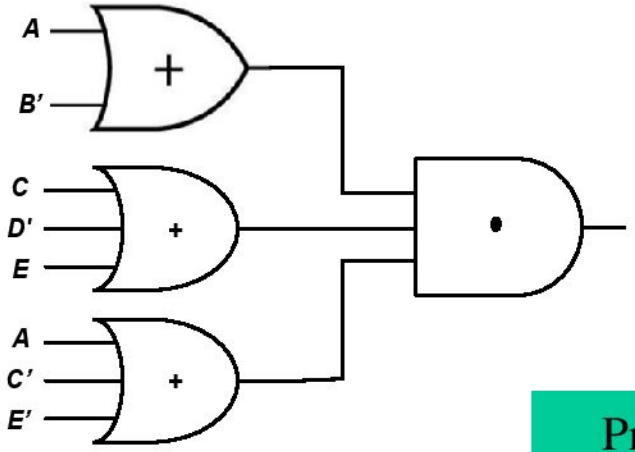
$AB'C(D'+E)$

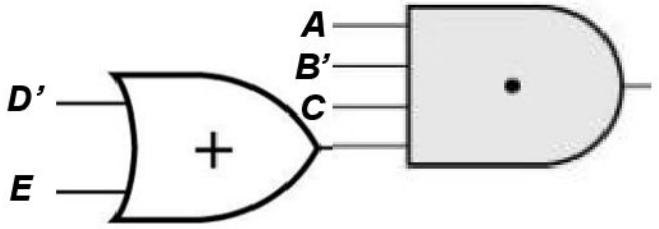
Circuits for SoP and PoS Forms





Sum of products form





Product of sums form

Multiplying Out and Factoring

To obtain a sum of products form ➔ Multiply out using distributive laws

$$X(Y+Z) = XY + XZ$$

$$(X+Y)(X+Z) = X + YZ$$

A useful theorem for multiplying out:

$$(X + Y)(X' + Z) = XZ + X'Y \qquad (3\text{-}3)$$

If X = 0, (3-3) reduces to Y(1+Z) = 0+1*Y or Y = Y.

If X = 1, (3-3) reduces to (1+Y)Z = Z+0*Y or Z = Z.

Example:

The use of Theorem 3-3 for factoring:

$$AB + A'C = (A+C)(A'+B)$$

Factoring Expressions

To obtain a product of sums form ➔ Factoring using distributive laws

Theorem for factoring:

$$\overbrace{AB + A'C} = (A + C)(A' + B)$$

Example of factoring:

$$AC + A'BD' + A'BE + A'C'DE$$

$$= \underbrace{AC}_{XZ} + A'(\underbrace{BD' + BE + C'DE}_{Y})$$

$$= (A + BD' + BE + C'DE)(A' + C)$$

$$= [\underbrace{A + C'DE}_{X} + B(\underbrace{D' + E}_{Z})](A' + C)$$

$$= (A + B + C'DE)(A + \cancel{C'DE} + D' + E)(A' + C)$$

$$= (A + B + C')(A + B + D)(A + B + E)(A + D' + E)(A' + C)$$

Minterms and Maxterms

- ***Minterm:***
  - A product term (with value 1) in which all the variables appear exactly once, either complemented or uncomplemented
- ***Maxterm:***
  - A sum term (with value 0) in which…
  - Minterms and Maxterms are easy to denote using a truth table.
  - Note: $m_i = M_i'$

x	y	z		Minterm	Maxterm
0	0	0		$x'y'z' = m_0$	$x+y+z = M_0$
0	0	1		$x'y'z = m_1$	$x+y+z' = M_1$
0	1	0		$x'yz' = m_2$	$x+y'+z = M_2$
0	1	1		$x'yz = m_3$	$x+y'+z' = M_3$
1	0	0		$xy'z' = m_4$	$x'+y+z = M_4$
1	0	1		$xy'z = m_5$	$x'+y+z' = M_5$
1	1	0		$xyz' = m_6$	$x'+y'+z = M_6$
1	1	1		$xyz = m_7$	$x'+y'+z' = M_7$

Canonical Forms: Unique

- Any Boolean function F( ) can be expressed as a *unique* **sum** of **min**terms (and a unique **product** of **max**terms)
- In other words, every function F() has two canonical forms:
  - Canonical Sum-Of-Products (sum of minterms)
  - Canonical Product-Of-Sums (product of maxterms)

Canonical Forms (cont.)

- **Canonical Sum-Of-Products:**
  - The minterms included are those m_j such that F( ) = 1 in row *j* of the truth table for F( ).
- **Canonical Product-Of-Sums:**
  - The maxterms included are those M_j such that F( ) = 0 in row *j* of the truth table for F( ).

Example

$f_1(a,b,c) = m_1 + m_2 + m_4 + m_6$

$= a'b'c + a'bc' + ab'c' + abc'$

$f_1(a,b,c) = M_0 \cdot M_3 \cdot M_5 \cdot M_7$

$= (a+b+c)\cdot(a+b'+c')\cdot$
$(a'+b+c')\cdot(a'+b'+c').$

a	b	c		f_1
0	0	0	0	0
0	0	1	1	1
0	1	0	2	1
0	1	1	3	0
1	0	0	4	1
1	0	1	5	0
1	1	0	6	1
1	1	1	7	0

- **Again observe that $m_j = M_j'$**

Shorthand: ∑ and ∏

- **$f_1(a,b,c) = \sum m(1,2,4,6)$,**
  - $m_1 + m_2 + m_4 + m_6$.
- **$f_1(a,b,c) = \prod M(0,3,5,7)$,**
  - $M_0 \cdot M_3 \cdot M_5 \cdot M_7$.

$$\sum m(1,2,4,6) = \prod M(0,3,5,7) = f_1(a,b,c)$$

Conversion Between Canonical Forms

- Replace $\sum$ with $\prod$ (or *vice versa*) and replace those *j*'s that appeared in the original form with those that did not.

- **Example:**
  - $f_1(a,b,c) = a'b'c + a'bc' + ab'c' + abc'$

    $= m_1 + m_2 + m_4 + m_6$

    $= \sum(1,2,4,6)$

    $= \prod(0,3,5,7)$

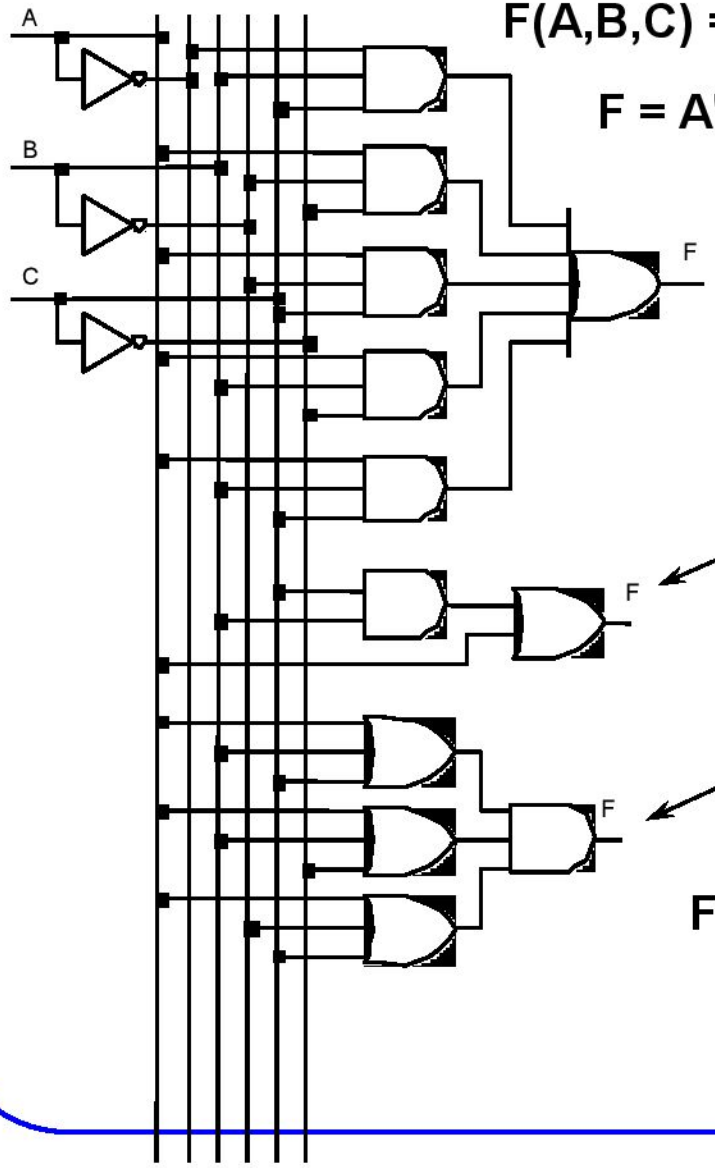
    $= (a+b+c)\cdot(a+b'+c')\cdot(a'+b+c')\cdot(a'+b'+c')$

Standard Forms (NOT Unique)

- Standard forms are "*like*" canonical forms,
  - not all variables need to appear in the individual product (SOP) or sum (POS) terms.

- **Example:**
  - $f_1(a,b,c) = a'b'c + bc' + ac'$ is a *standard* sum-of-products form
  - $f_1(a,b,c) = (a+b+c)\cdot(b'+c')\cdot(a'+c')$ is a *standard* product-of-sums form.

Alternative Implementations



$F(A,B,C) = \Sigma m(3,4,5,6,7)$

$F = A'\,B\,C + A\,B'\,C' + A\,B'\,C + A\,B\,C' + A\,B\,C$

Canonical Sum of Products

Standard Sum of Products

$A + B\,C$

Canonical Product of Sums

$F(A,B,C) = \Pi M(0,1,2)$

$= (A + B + C)\,(A + B + C')\,(A + B' + C)$

Conversion of SOP from standard to canonical

- Expand *non-canonical* terms by inserting equivalent of 1 for each missing variable
  - x: $(x + x') = 1$
- Remove duplicate minterms
- $f_1(a,b,c) = a'b'c + bc' + ac'$

$$= a'b'c + (a+a')bc' + a(b+b')c'$$
$$= a'b'c + abc' + a'bc' + abc' + ab'c'$$
$$= a'b'c + abc' + a'bc + ab'c'$$

Conversion of POS from standard to canonical

- Expand noncanonical terms by adding 0 in terms of missing variables (*e.g.*, $xx' = 0$) and using the distributive law
- Remove duplicate maxterms
- $f_1(a,b,c) = (a+b+c)\cdot(b'+c')\cdot(a'+c')$

  $= (a+b+c)\cdot(aa'+b'+c')\cdot(a'+bb'+c')$

  $= (a+b+c)\cdot(a+b'+c')\cdot(a'+b'+c')\cdot$

  $(a'+b+c')\cdot(a'+b'+c')$

  $= (a+b+c)\cdot(a+b'+c')\cdot(a'+b'+c')\cdot(a'+b+c')$

Conversion of Natural Language Sentences to Boolean Equations

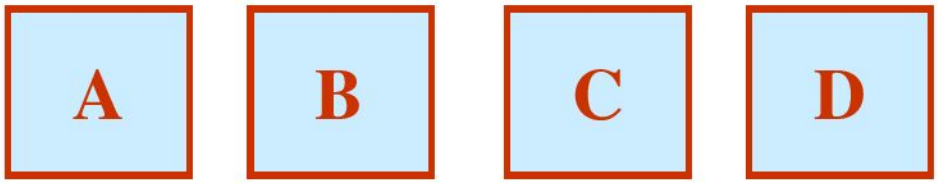
- The first step in designing a logic network:
  - To translate natural language sentences to Boolean equations:
    - We must break down each sentence into phrases
    - And associate a Boolean variable with each phrase (possible if a phrase can have a “true”/”false” value)
- Example:
  - Ali watches TV if it is Monday night and he has finished his homework.
  - → F = A . B

Main Steps

- Three main steps in designing a single-output combinational switching network:
  - Find a switching function which specifies the desired behavior of the network.
  - Find a simplified algebraic expression for the function.
  - Realize the simplified function using available logic elements.

Example

- Four chairs in a row: design a circuit that shows a 1 if no adjacent chairs are empty
  - Occupied: '1'
  - Empty: '0'
  - F = '1' iff there are no adjacent empty chairs.



F = (A'B' + B'C' + C'D')'

Example (Continued)

$$F = (A'B' + B'C' + C'D')'$$

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

$$= (A+B).(B+C).(C+D)$$

$$= (B+AC).(C+D)$$

$$= BC + BD + AC + ACD$$

$$= BC + BD + AC$$

Problem Solving

- Sometimes we have to work with imprecise word descriptions of logic functions.

- **Example 3:**
  - "The ERROR output should be 1 if the GEAR, CLUTCH, and BRAKE inputs are inconsistent."
    - e.g., the gear cannot be shifted unless the clutch is applied.
  - In this situation:
    - Truth-table approach is the best.
    - It allows us to determine the output required for every input combination, based on our understanding of the problem and the knowledge of environment.

Example

- **A 4-bit Prime Number Detector**

$F = \Sigma_{N3,N2,N1,N0}(1, 2, 3, 5, 7, 11, 13)$

$=N3'.N2'.N1'.N0 + N3'.N2'.N1.N0' + N3'.N2'.N1.N0 + N3'.N2.N1'.N0 + N3'.N2.N1.N0 + N3.N2'.N1.N0 + N3.N2.N1'.N0$

$=N3'.N1'.N0 + N3'.N2'.N1 + N3'.N2.N1.N0 + N3.N2'.N1.N0 + N3.N2.N1'.N0$

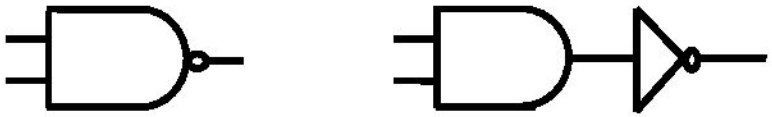
$=\ldots$

- برای عبارات پیچیده، کار بهینه‌سازی سخت‌تر می‌شود
- نیاز به روشی سیستماتیک و آسان‌تر: جدول کارنو
  - شما را از جبر بول بی‌نیاز نمی‌کند.

More Logic Gates

- **NAND:**
  - *NOT-AND*
    - Its output = 1 if at least one input is 0.
    - **(A • B)'**

A	B	(A.B)'
0	0	1
0	1	1
1	0	1
1	1	0

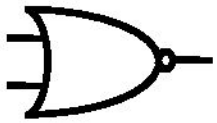


  - has traditionally been the *universal gate* in digital circuits.
    - It is simple to implement in hardware and can be used to construct all other gates.

More Logic Gates

- **NOR:**
  - *NOT-OR*
    - Its output = 1 if all inputs are 0.
    - **(A + B)'**

A	B	(A+B)'
0	0	1
0	1	0
1	0	0
1	1	0

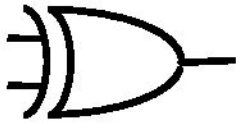


  - Can be a *universal gate.*

More Logic Gates

- **XOR:**
  - *Inequality*
    - Its output = 1 if inputs are <u>not</u> equal.
    - **A ⊕ B**

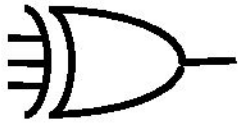
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0



More Logic Gates

- **3-input XOR:**
  - Its output = 1 only if one input is 1 or all three inputs are 1.
  - **(A ⊕ B ⊕ C) = (A ⊕ B) ⊕ C = A ⊕ (B ⊕ C)**

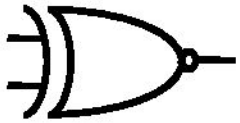
A	B	C	A XOR B XOR C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



More Logic Gates

- **XNOR:**
  - *Equality*
    - Its output = 1 if both inputs are equal.
    - **(A ⊕ B)'**

A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1



Theorems for XOR

$$X \oplus 0 = X$$

$$X \oplus 1 = X'$$

$$X \oplus X = 0$$

$$X \oplus X' = 1$$

$$X \oplus Y = Y \oplus X \text{ (commutativ e law)}$$

$$(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z) = X \oplus Y \oplus Z \text{ (associativ e law)}$$

$$X(Y \oplus Z) = XY \oplus XZ \text{ (distributi ve law)}$$

$$X \oplus Y = XY' + X'Y$$

$$(X \oplus Y)' = X \oplus Y' = X' \oplus Y = XY + X'Y'$$

$$(X \oplus Y)' = (XY' + X'Y)' = XY + X'Y'$$

Incompletely Specified Functions

- **Don't Care:**
  - *We don't care about the output values for some input patterns.*
  - *This fact can be exploited during circuit minimization.*
- ***Example:***
  - *BCD incrementer:*

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Off set of W

On set of W

Don't Care (DC) set of W

These input patterns should never be encountered so in practice associated output values are "Don't Cares"

Don't Cares and Canonical Forms

Canonical Representations of the BCD Incrementer:

$$Z = m0 + m2 + m4 + m6 + m8 + d10 + d11 + d12 + d13 + d14 + d15$$

$$Z = \Sigma m(0, 2, 4, 6, 8) + d(10, 11, 12 ,13, 14, 15)$$

$$Z = M1 \bullet M3 \bullet M5 \bullet M7 \bullet M9 \bullet D10 \bullet D11 \bullet D12 \bullet D13 \bullet D14 \bullet D15$$

$$Z = \Pi M(1, 3, 5, 7, 9) \bullet D(10, 11, 12, 13, 14 ,15)$$