DIGITAL ELECTRONICS

Prof. Rasmita Lenka

Overview

- Binary logic and Gates
- Boolean Algebra
  - Basic Properties
  - Algebraic Manipulation
- Standard and Canonical Forms
  - Minterms and Maxterms (Canonical forms)
  - SOP and POS (Standard forms)
- Karnaugh Maps (K-Maps)
  - 2, 3, 4, and 5 variable maps
  - Simplification using K-Maps
- K-Map Manipulation
  - Implicants: Prime, Essential
  - Don’t Cares
- More Logic Gates

Binary Logic

- Deals with binary variables that take 2 discrete values (0 and 1), and with logic operations
- Three basic logic operations:
  - AND, OR, NOT
- Binary/logic variables are typically represented as 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’•b + b’

G(x,y,z) = x•(y+z’)

Basic Logic Operators

- AND
- OR
- NOT

Binary

Unary

- F(a,b) = a•b,   F is 1 <u>if and only if</u> a=b=1
- G(a,b) = a+b, G is 1 if either a=1 or b=1
- H(a) = a’, H is 1 if a=0

Basic Logic Operators (cont.)

- 1-bit logic AND resembles binary multiplication:

$$0 \bullet 0 = 0, \quad 0 \bullet 1 = 0,$$

$$1 \bullet 0 = 0, \quad 1 \bullet 1 = 1$$

- 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: tabular form that uniguely 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.)

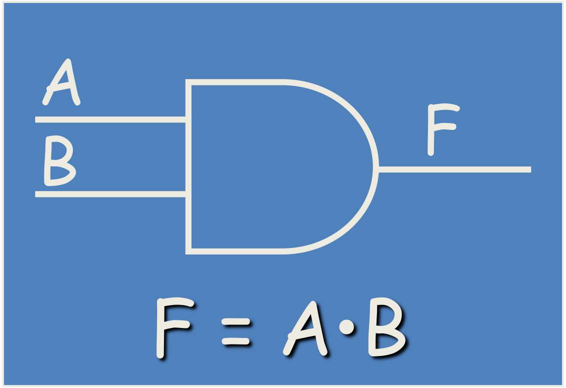
- Q: Let a function F() depend on *n* variables. How many rows are there in the truth table of F() ?

- **A:** 2^n rows, since there are 2^n possible binary patterns/combinations for the *n* variables

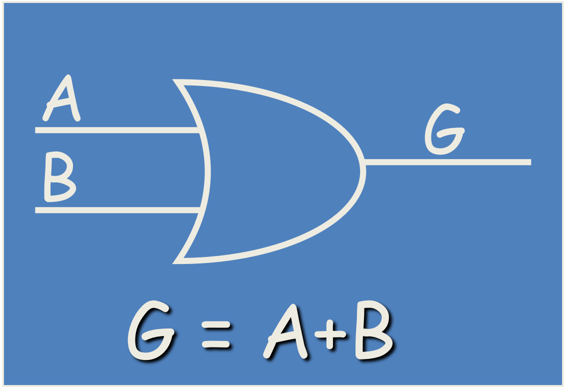
Logic Gates

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

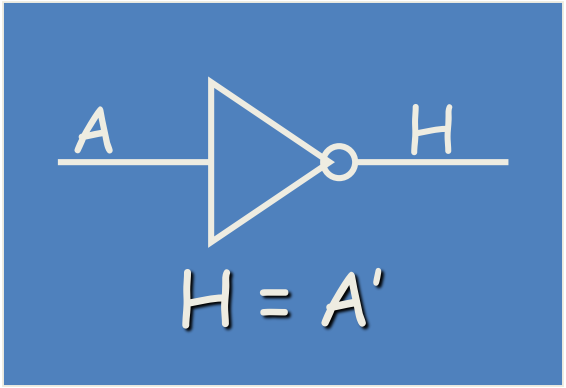
2-Input AND



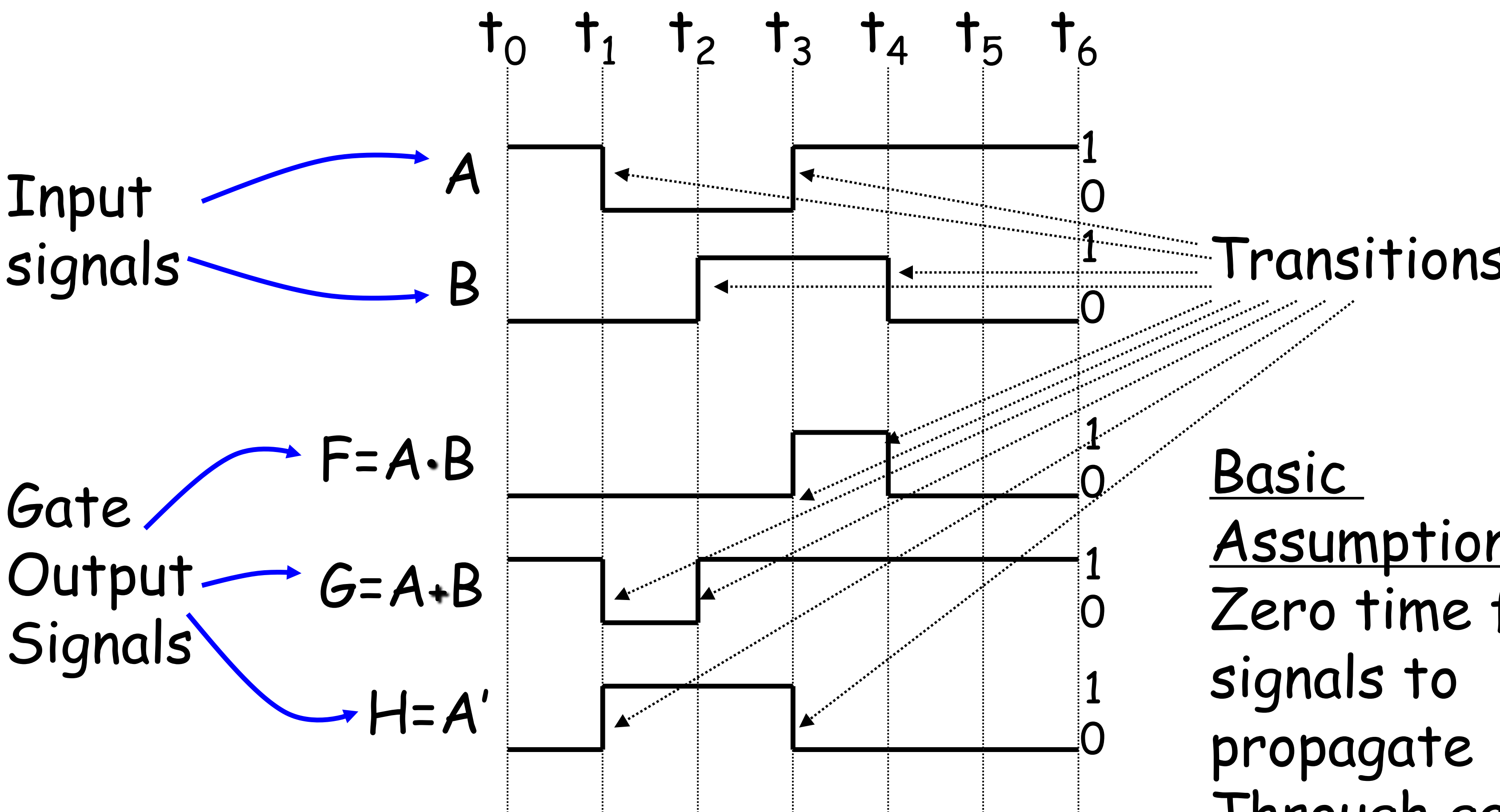
2-Input OR



NOT (Inverter)



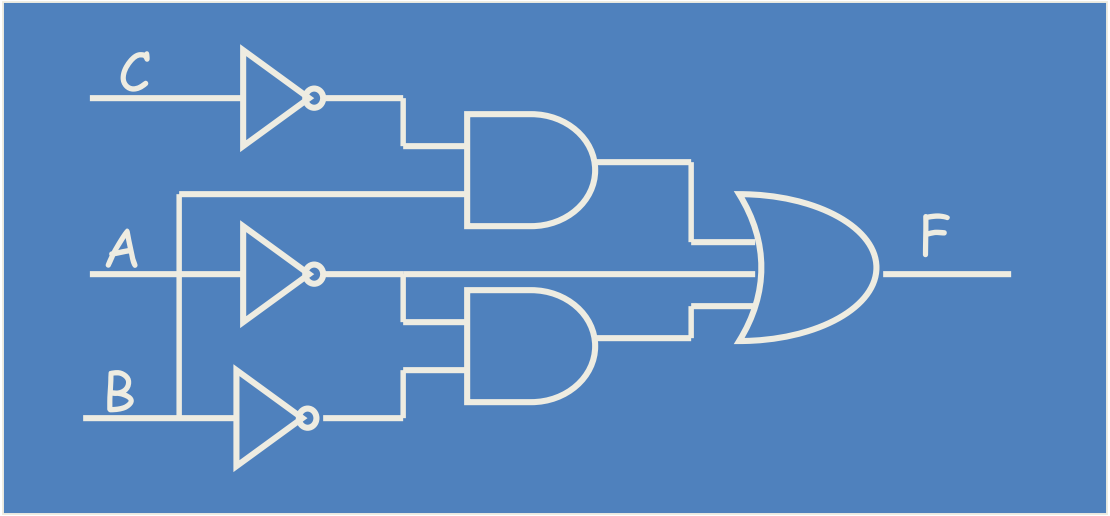
Timing Diagram



<u>Basic Assumption:</u>
Zero time for signals to propagate Through gates

Combinational Logic Circuit from Logic Function

- Consider function F = A’ + B•C’ + A’•B’
- A combinational logic circuit can be constructed to implement F, by appropriately connecting 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

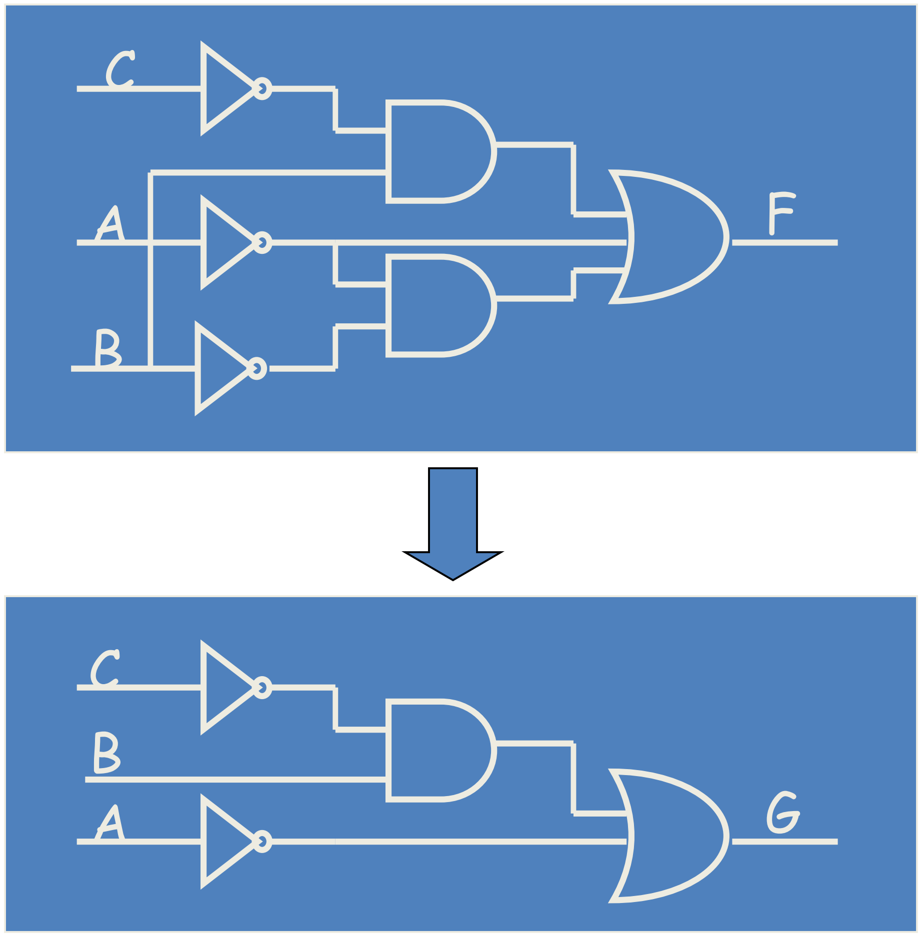


Combinational Logic Circuit from Logic Function (cont.)

- In order to design a cost-effective and efficient circuit, we must minimize the circuit's size (area) and propagation delay (time required for an input signal change to be observed at the output line)
- Observe the truth table of F=A' + B•C' + A'•B' and G=A' + B•C'
- Truth tables for F and G are identical → same function
- Use G to implement the logic circuit (less 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 Circuit from Logic Function (cont.)



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
  - *Product term:* literals connected by •
  - *Sum term:* literals connected by +

Boolean Algebra Properties

Let X: boolean variable, 0,1: constants

1. $X + 0 = X$ -- Zero Axiom
2. $X \bullet 1 = X$ -- Unit Axiom
3. $X + 1 = 1$ -- Unit Property
4. $X \bullet 0 = 0$ -- Zero Property

Boolean Algebra Properties (cont.)

Let X: boolean variable, 0,1: constants

5. $X + X = X$ -- Idepotence
6. $X \bullet X = X$ -- Idepotence
7. $X + X' = 1$ -- Complement
8. $X \bullet X' = 0$ -- Complement
9. $(X')' = X$ -- Involution

Duality

- 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.
- Cannot exchange x with x'
- Example:
  - Find $H(x,y,z)$, the dual of $F(x,y,z) = x'yz' + x'y'z$
  - $H = (x'+y+z')(x'+y'+z)$

Duality (cont'd)

With respect to duality, Identities 1 – 8 have the following relationship:

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 8)

More Boolean Algebra Properties

Let X,Y, and Z: boolean variables

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

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

14. $X\bullet(Y+Z) = X\bullet Y + X\bullet Z$ 15. $X+(Y\bullet Z) = (X+Y) \bullet (X+Z)$

-- Distributive

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

In general,

$(X_1 + X_2 + \ldots + X_n)' = X_1'\bullet X_2' \bullet \ldots \bullet X_n'$, and

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

Absorption Property

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

- **Proof:**

$$
\begin{aligned}
x + x \bullet y &= x \bullet 1 + x \bullet y \\
&= x \bullet (1+y) \\
&= x \bullet 1 \\
&= x
\end{aligned}
$$

QED (2 true by duality, why?)

Power of Duality

1. $x + x \bullet y = x$ is true, so $(x + x \bullet y)' = x'$
2. $(x + x \bullet y)' = x' \bullet (x' + y')$
3. $x' \bullet (x' + y') = x'$
4. Let $X = x'$, $Y = y'$
5. $X \bullet (X + Y) = X$, which is the dual of $x + x \bullet y = x$.
6. The above process can be applied to any formula. So if a formula is valid, then its dual must also be valid.
7. Proving one formula also proves its dual.

Consensus Theorem

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

- **Proof:**

$$
\begin{aligned}
xy + x'z + yz &= xy + x'z + (x+x')yz \\
&= xy + x'z + xyz + x'yz \\
&= (xy + xyz) + (x'z + x'zy) \\
&= xy + x'z
\end{aligned}
$$

QED (2 true by duality).

Truth Tables (revisited)

- Enumerates all possible combinations of variable values and the corresponding function value
- Truth tables for some arbitrary functions $F_1(x,y,z)$, $F_2(x,y,z)$, and $F_3(x,y,z)$ are shown to the right.

x	y	z		F_1	F_2	F_3
0	0	0		0	1	1
0	0	1		0	0	1
0	1	0		0	0	1
0	1	1		0	1	1
1	0	0		0	1	0
1	0	1		0	1	0
1	1	0		0	0	0
1	1	1		1	0	1

Truth Tables (cont.)

- Truth table: a <u>unique</u> representation of a Boolean function
- If two functions have identical truth tables, the functions are equivalent (and vice-versa).
- Truth tables can be used to prove equality theorems.
- However, the size of a truth table grows <u>exponentially</u> with the number of variables involved, hence unwieldy. This motivates the use of Boolean Algebra.

Boolean expressions-NOT unique

- Unlike truth tables, expressions representing a Boolean function are NOT unique.
- Example:
  - F(x,y,z) = x'•y'•z' + x'•y•z' + x•y•z'
  - G(x,y,z) = x'•y'•z' + y•z'
- The corresponding truth tables for F() and G() are to the right. They are identical.
- Thus, F() = G()

x	y	z		F	G
0	0	0		1	1
0	0	1		0	0
0	1	0		1	1
0	1	1		0	0
1	0	0		0	0
1	0	1		0	0
1	1	0		1	1
1	1	1		0	0

Algebraic Manipulation

- Boolean algebra is a useful tool for simplifying digital circuits.
- Why do it? Simpler can mean cheaper, smaller, faster.
- Example: Simplify F = x'yz + x'yz' + xz.

$$
\begin{aligned}
F &= x'yz + x'yz' + xz \\
&= x'y(z+z') + xz \\
&= x'y \bullet 1 + xz \\
&= x'y + xz
\end{aligned}
$$

Algebraic Manipulation (cont.)

- Example: Prove

  $x'y'z' + x'yz' + xyz' = x'z' + yz'$

- **Proof:**

  $x'y'z' + x'yz' + xyz'$

  $= x'y'z' + x'yz' + x'yz' + xyz'$

  $= x'z'(y'+y) + yz'(x'+x)$

  $= x'z' \bullet 1 + yz' \bullet 1$

  $= x'z' + yz'$

  QED.

Complement of a Function

- The complement of a function is derived by interchanging (• and +), and (1 and 0), and complementing each variable.
- Otherwise, interchange 1s to 0s in the truth table column showing F.
- The *complement* of a function IS NOT THE SAME as the *dual* of a function.

Complementation: Example

- Find $G(x,y,z)$, the complement of $F(x,y,z) = xy'z' + x'yz$
- $G = F' = (xy'z' + x'yz)'$
  $= (xy'z')' \bullet (x'yz)'$ *DeMorgan*
  $= (x'+y+z) \bullet (x+y'+z')$ *DeMorgan* again
- Note: The complement of a function can also be derived by finding the function's *dual,* and then complementing all of the literals

Canonical and Standard Forms

- We need to consider formal techniques for the simplification of Boolean functions.
  - Identical functions will have exactly the same canonical form.
  - Minterms and Maxterms
  - Sum-of-Minterms and Product-of- Maxterms
  - Product and Sum terms
  - Sum-of-Products (SOP) and Product-of-Sums (POS)

Definitions

- *Literal:* A variable or its complement
- *Product term:* literals connected by •
- *Sum term:* literals connected by +
- *Minterm:* a product term in which all the variables appear exactly once, either complemented or uncomplemented
- *Maxterm:* a sum term in which all the variables appear exactly once, either complemented or uncomplemented

Minterm

- Represents exactly one combination in the truth table.
- Denoted by m_j, where *j* is the decimal equivalent of the minterm's corresponding binary combination *(b_j).*
- A variable in m_j is complemented if its value in b_j is 0, otherwise is uncomplemented.
- Example: Assume 3 variables (A,B,C), and *j*=3. Then, b_j = 011 and its corresponding minterm is denoted by m_j = A'BC

Maxterm

- Represents exactly one combination in the truth table.
- Denoted by M_j, where j is the decimal equivalent of the maxterm's corresponding binary combination (b_j).
- A variable in M_j is complemented if its value in b_j is 1, otherwise is uncomplemented.
- Example: Assume 3 variables (A,B,C), and j=3. Then, b_j = 011 and its corresponding maxterm is denoted by M_j = A+B'+C'

Truth Table notation for Minterms and Maxterms

- Minterms and Maxterms are easy to denote using a truth table.
- Example:
  Assume 3 variables x,y,z
  (order is fixed)

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 (under a fixed variable ordering).
- 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

- Truth table for $f_1(a,b,c)$ at right
- The canonical sum-of-products form for f_1 is
  $f_1(a,b,c) = m_1 + m_2 + m_4 + m_6$
  $= a'b'c + a'bc' + ab'c' + abc'$
- The canonical product-of-sums form for f_1 is
  $f_1(a,b,c) = M_0 \bullet M_3 \bullet M_5 \bullet M_7$
  $= (a+b+c)\bullet(a+b'+c')\bullet$
  $(a'+b+c')\bullet(a'+b'+c')$.
- Observe that: $m_j = M_j'$

a	b	c		f_1
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		1
1	1	1		0

Shorthand: $\sum$ and $\prod$

- $f_1(a,b,c) = \sum m(1,2,4,6)$, where $\sum$ indicates that this is a sum-of-products form, and $m(1,2,4,6)$ indicates that the minterms to be included are m_1, m_2, m_4, and m_6.
- $f_1(a,b,c) = \prod M(0,3,5,7)$, where $\prod$ indicates that this is a product-of-sums form, and $M(0,3,5,7)$ indicates that the maxterms to be included are M_0, M_3, M_5, and M_7.
- Since $m_j = M_j'$ for any *j*,
  $\sum m(1,2,4,6) = \prod M(0,3,5,7) = f_1(a,b,c)$

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 do not.
- Example:

$$
\begin{aligned}
f_1(a,b,c) &= a'b'c + a'bc' + ab'c' + abc' \\
&= m_1 + m_2 + m_4 + m_6 \\
&= \sum(1,2,4,6) \\
&= \prod(0,3,5,7) \\
&= (a+b+c)\bullet(a+b'+c')\bullet(a'+b+c')\bullet(a'+b'+c')
\end{aligned}
$$

Standard Forms (NOT Unique)

- Standard forms are "*like*" canonical forms, except that not all variables need 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
- $f_1(a,b,c) = (a+b+c) \bullet (b'+c') \bullet (a'+c')$
  is a *standard* product-of-sums form.

Conversion of SOP from standard to canonical form

- Expand *non-canonical* terms by inserting equivalent of 1 in 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 form

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

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

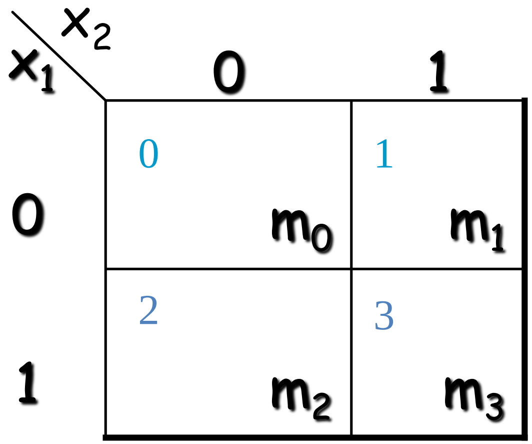
$$= (a+b+c)\bullet(a+b'+c')\bullet(a'+b'+c')\bullet(a'+b+c')\bullet(a'+b'+c')$$

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

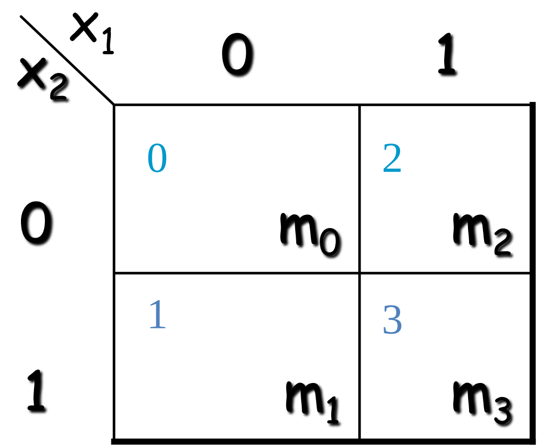
Karnaugh Maps

- Karnaugh maps (K-maps) are *graphical* representations of boolean functions.
- One ***map cell*** corresponds to a row in the truth table.
- Also, one map cell corresponds to a minterm or a maxterm in the boolean expression
- Multiple-cell areas of the map correspond to standard terms.

Two-Variable Map



OR



NOTE: ordering of variables is IMPORTANT for $f(x_1,x_2)$, x_1 is the row, x_2 is the column.

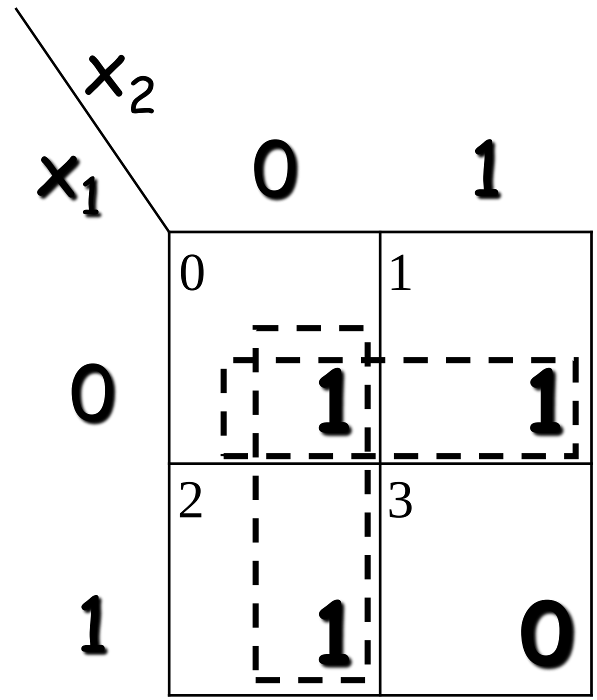
Cell 0 represents $x_1'x_2'$; Cell 1 represents $x_1'x_2$; etc. If a minterm is present in the function, then a 1 is placed in the corresponding cell.

Two-Variable Map (cont.)

- Any two adjacent cells in the map differ by ONLY one variable, which appears complemented in one cell and uncomplemented in the other.
- Example:
  m_0 ($=x_1'x_2'$) is adjacent to m_1 ($=x_1'x_2$) and m_2 ($=x_1x_2'$) but NOT m_3 ($=x_1x_2$)

2-Variable Map -- Example

- $f(x_1,x_2) = x_1'x_2' + x_1'x_2 + x_1x_2'$
  $= m_0 + m_1 + m_2$
  $= x_1' + x_2'$
- 1s placed in K-map for specified minterms m_0, m_1, m_2
- Grouping (ORing) of 1s allows simplification
- What (simpler) function is represented by each dashed rectangle?
  - $x_1' = m_0 + m_1$
  - $x_2' = m_0 + m_2$
- Note m_0 covered twice

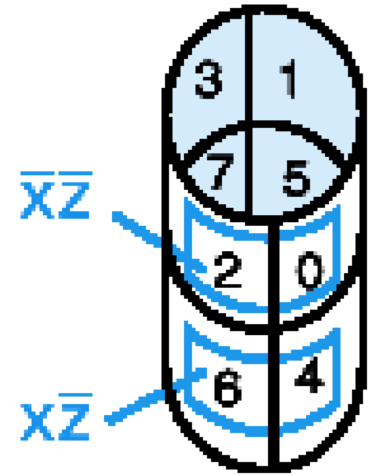


Minimization as SOP using K-map

- Enter 1s in the K-map for each product term in the function
- Group *adjacent* K-map cells containing 1s to obtain a product with fewer variables. Group size must be in power of 2 (2, 4, 8, …)
- Handle "boundary wrap" for K-maps of 3 or more variables.
- Realize that answer may not be unique

Three-Variable Map

x \ yz	00	01	11	10
0	0 m_0	1 m_1	3 m_3	2 m_2
1	4 m_4	5 m_5	7 m_7	6 m_6

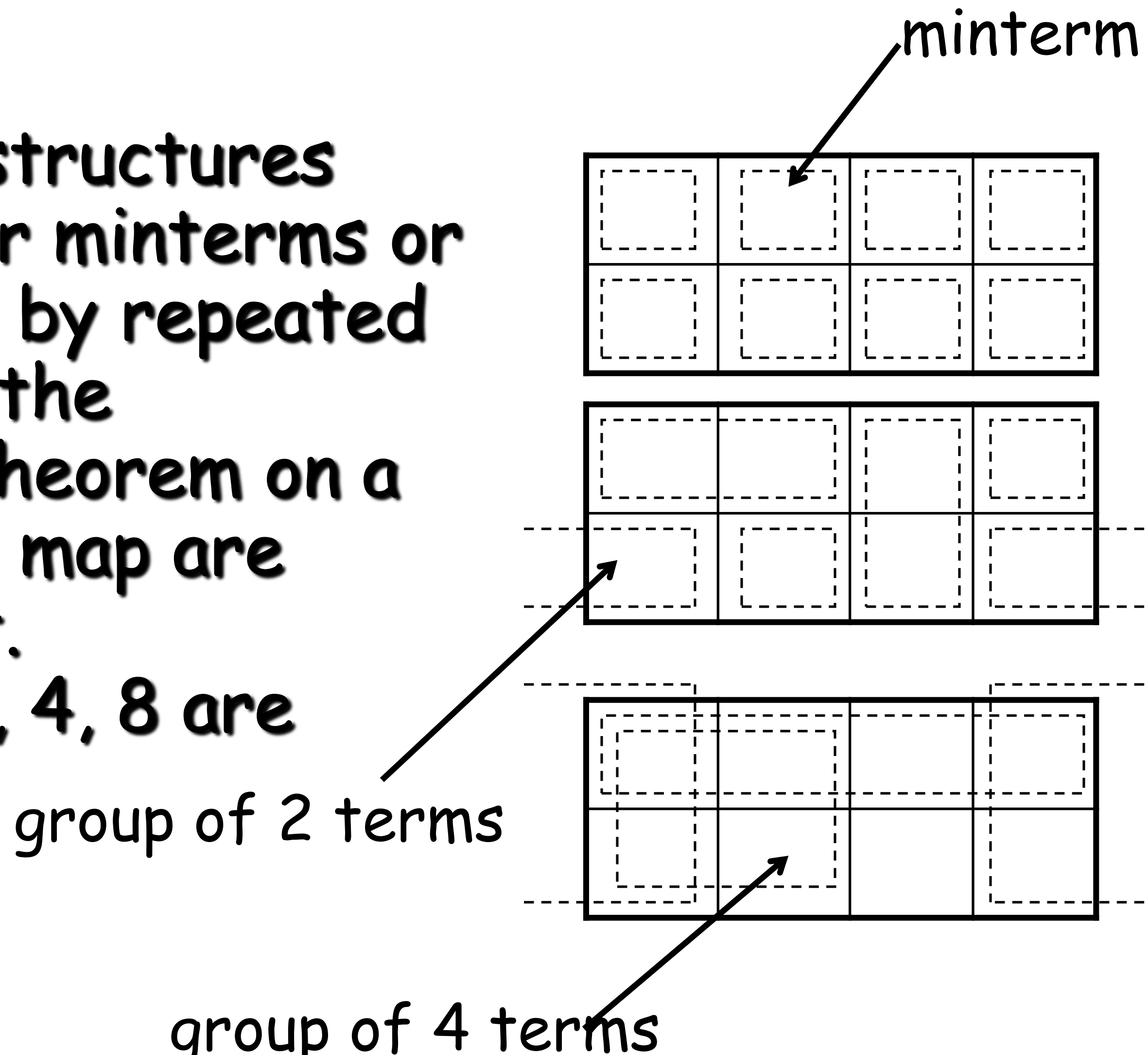


-Note: variable ordering is (x,y,z); yz specifies column, x specifies row.

-Each cell is adjacent to ***three*** other cells (left or right or top or bottom or edge wrap)

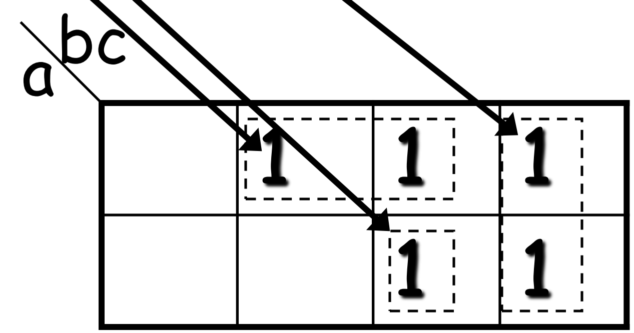
Three-Variable Map (cont.)

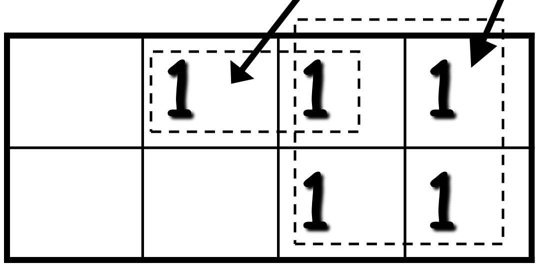
The types of structures that are either minterms or are generated by repeated application of the minimization theorem on a three variable map are shown at right.
Groups of 1, 2, 4, 8 are possible.



Simplification

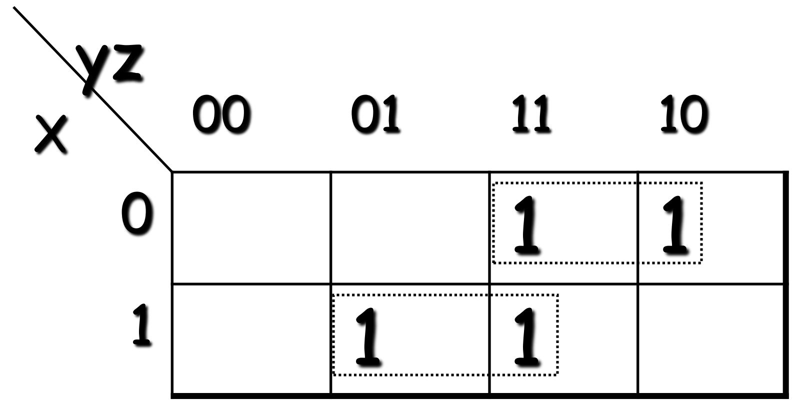
- Enter minterms of the Boolean function into the map, then group terms
- Example: f(a,b,c) = a’c + abc + bc’
- Result: f(a,b,c) = a’c+ b



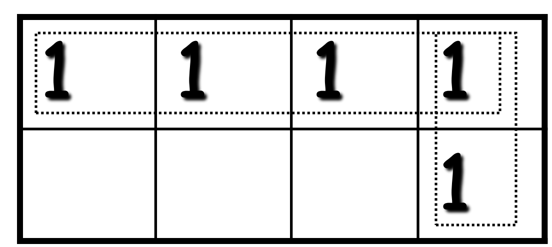


More Examples

- $f_1(x, y, z) = \sum m(2,3,5,7)$
  - $f_1(x, y, z) = x'y + xz$

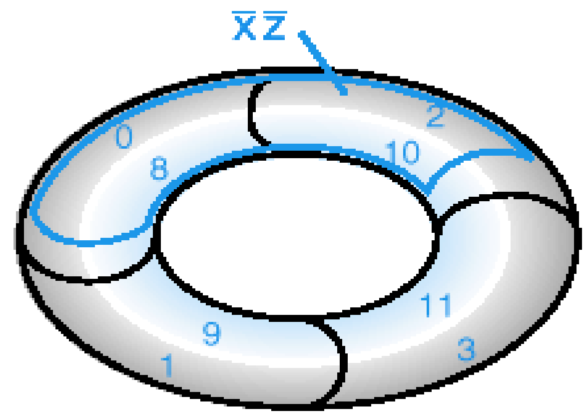


- $f_2(x, y, z) = \sum m(0,1,2,3,6)$
  - $f_2(x, y, z) = x' + yz'$



Four-Variable Maps

WX \ YZ	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}



- Top cells are adjacent to bottom cells. Left-edge cells are adjacent to right-edge cells.
- Note variable ordering (WXYZ).

Four-variable Map Simplification

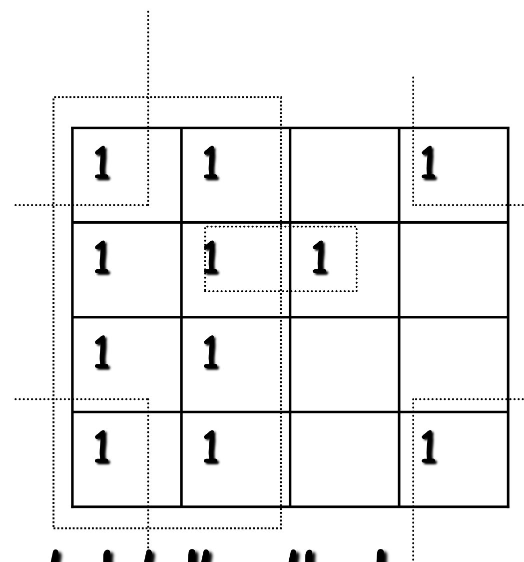
- One square represents a minterm of 4 literals.
- A rectangle of 2 adjacent squares represents a product term of 3 literals.
- A rectangle of 4 squares represents a product term of 2 literals.
- A rectangle of 8 squares represents a product term of 1 literal.
- A rectangle of 16 squares produces a function that is equal to logic 1.

Example

- Simplify the following Boolean function (A,B,C,D) = $\sum m(0,1,2,4,5,7,8,9,10,12,13)$.
- First put the function g( ) into the map, and then group as many 1s as possible.

ab \ cd

1	1		1
1	1	1	
1	1		
1	1		1



$g(A,B,C,D) = c' + b'd' + a'bd$

Don't Care Conditions

- There may be a combination of input values which
  - will **never** occur
  - if they do occur, the output is of no concern.
- The function value for such combinations is called a *don't care*.
- They are denoted with **x** or **–**. Each **x** may be arbitrarily assigned the value 0 or 1 in an implementation.
- Don’t cares can be used to ***further*** simplify a function

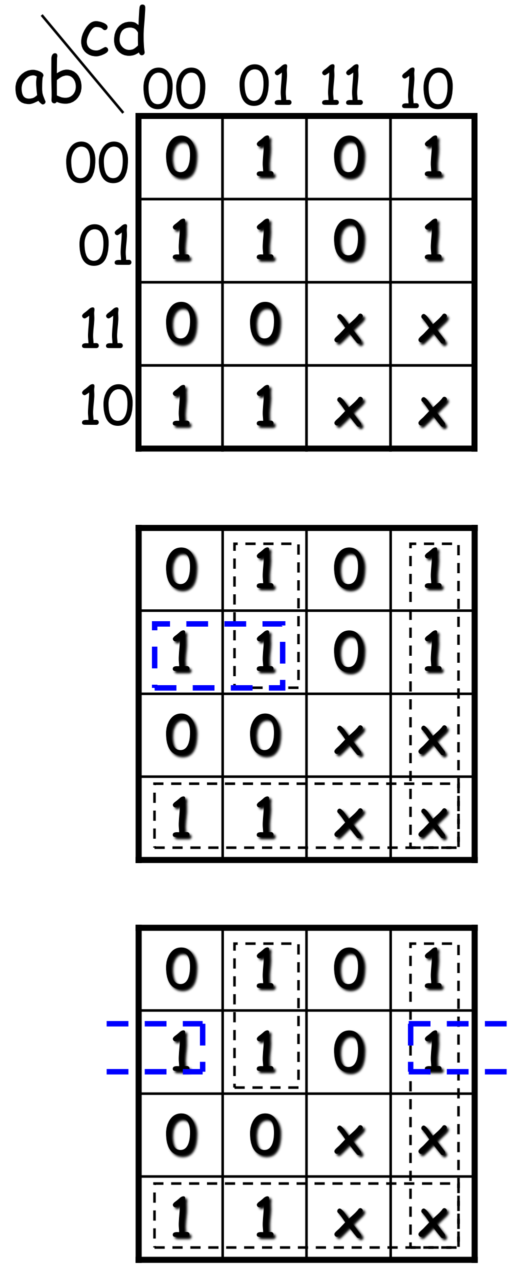
Minimization using Don’t Cares

- Treat don't cares as if they are 1s to generate PIs.
- Delete PI's that cover only don't care minterms.
- Treat the covering of remaining don't care minterms as optional in the selection process (*i.e.* they may be, but need not be, covered).

Example

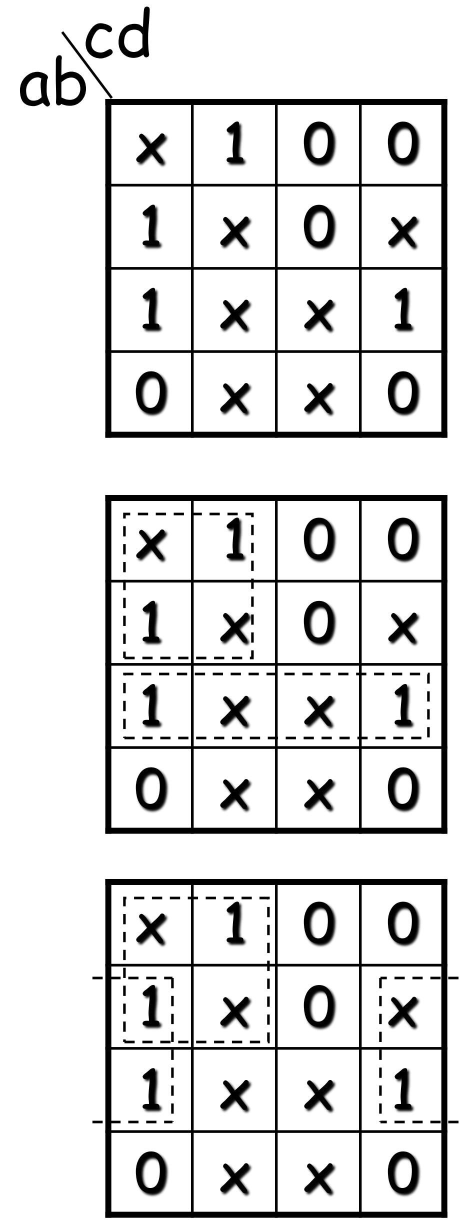
- Simplify the function f(a,b,c,d) whose K-map is shown at the right.
- f = a'c'd+ab'+cd'+a'bc'

  or
- f = a'c'd+ab'+cd'+a'bd'



Another Example

- Simplify the function g(a,b,c,d) whose K-map is shown at right.
- g = a’c’+ ab
  or
- g = a’c’+b’d



Algorithmic minimization

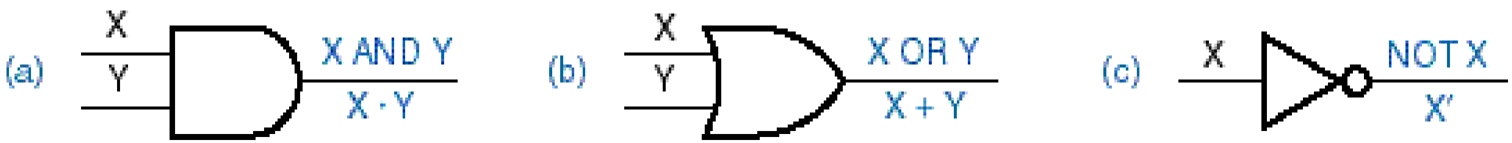
- What do we do for functions with more variables?
- You can “code up” a minimizer (Computer-Aided Design, CAD)
  - Quine-McCluskey algorithm
  - Iterated consensus
- We won’t discuss these techniques here

More Logic Gates

- NAND and NOR Gates
  - NAND and NOR circuits
  - Two-level Implementations
  - Multilevel Implementations
- Exclusive-OR (XOR) Gates
  - Odd Function
  - Parity Generation and Checking

More Logic Gates

- We can construct any combinational circuit with AND, OR, and NOT gates



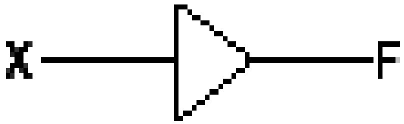
X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

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

X	NOT X
0	1
1	0

- Additional logic gates are used for practical reasons

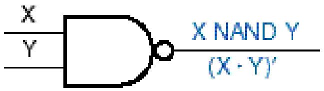
BUFFER, NAND and NOR



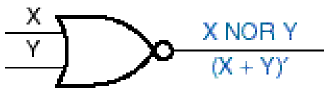
F = X

X	F
0	0
1	1

(a)



(b)



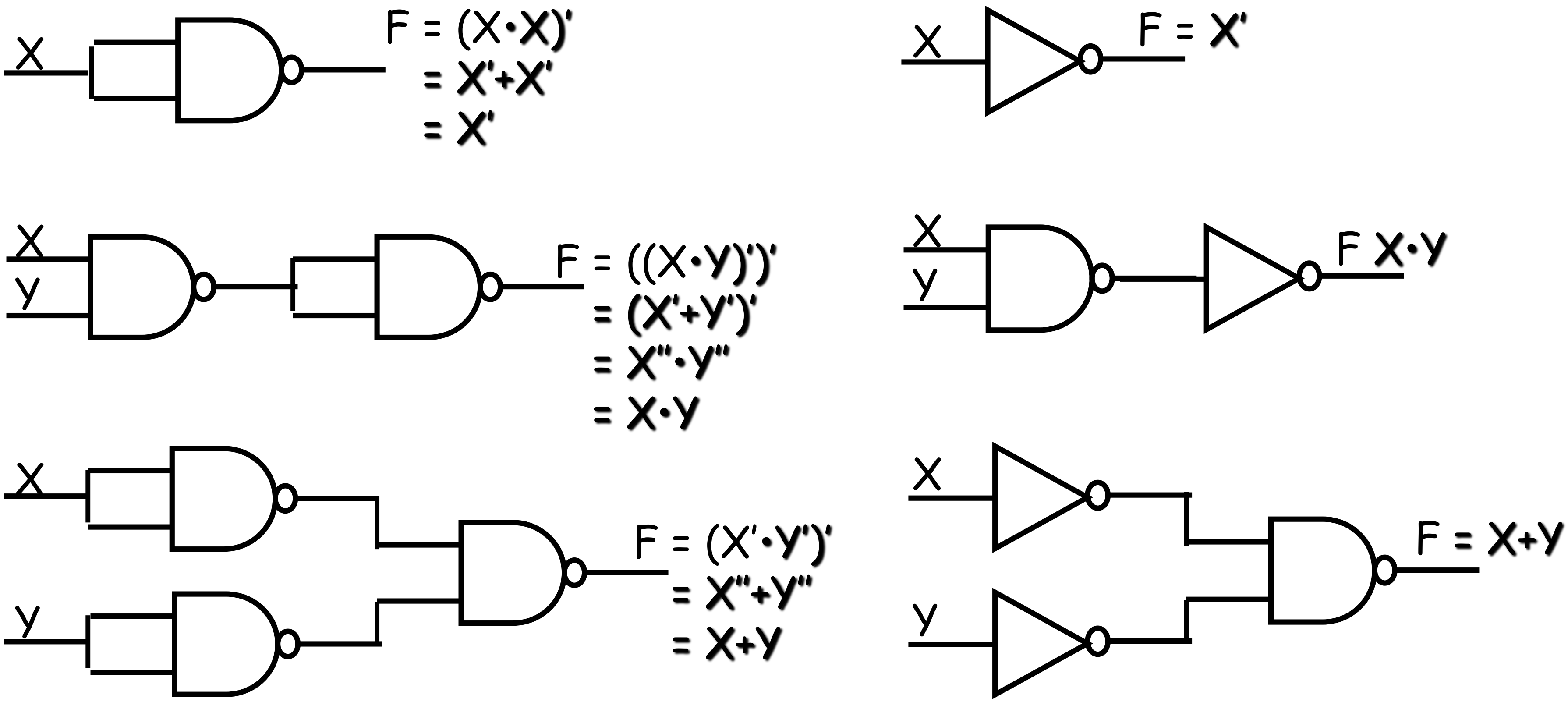
X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0

X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0

NAND Gate

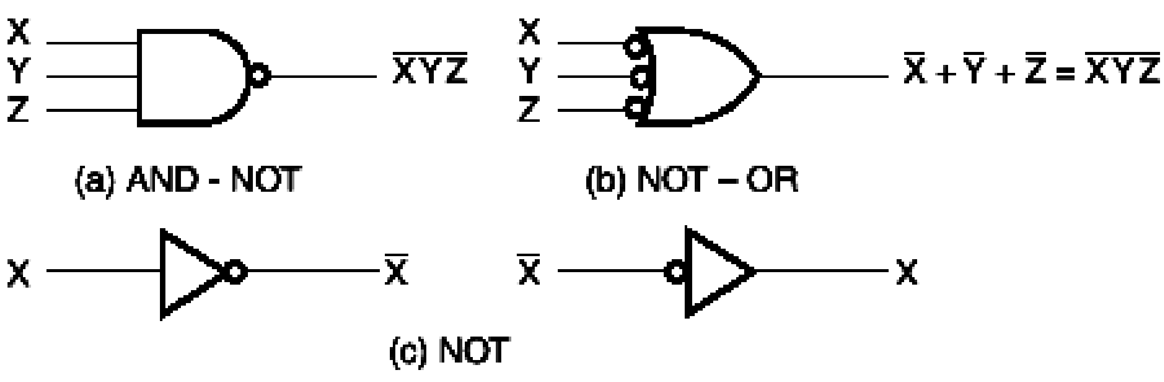
- Known as a “universal” gate because ANY digital circuit can be implemented with NAND gates alone.
- To prove the above, it suffices to show that AND, OR, and NOT can be implemented using NAND gates only.

NAND Gate Emulation

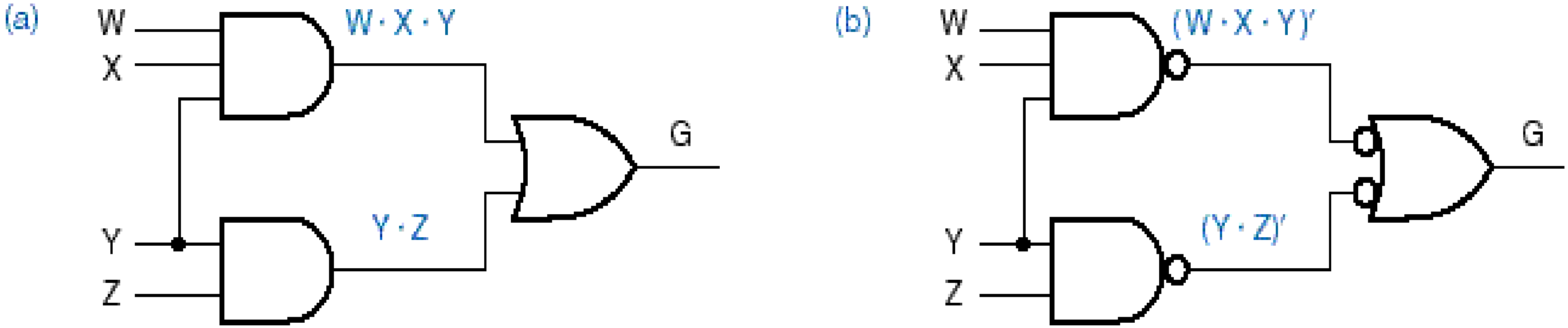


NAND Circuits

- To easily derive a NAND implementation of a boolean function:
  - Find a simplified SOP
  - SOP is an AND-OR circuit
  - Change AND-OR circuit to a NAND circuit
  - Use the alternative symbols below



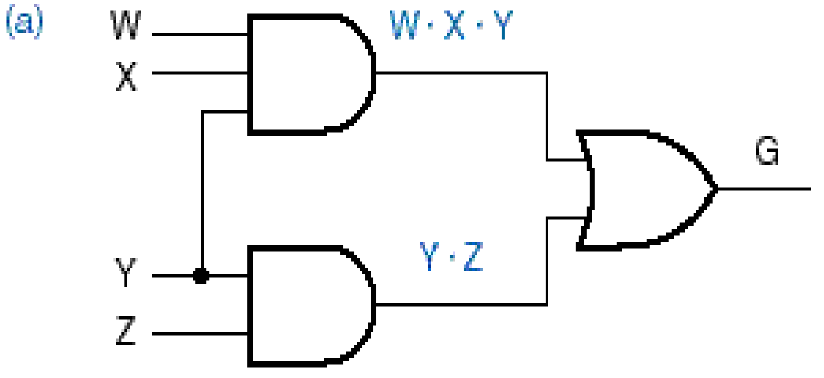
AND-OR (SOP) Emulation Using NANDs

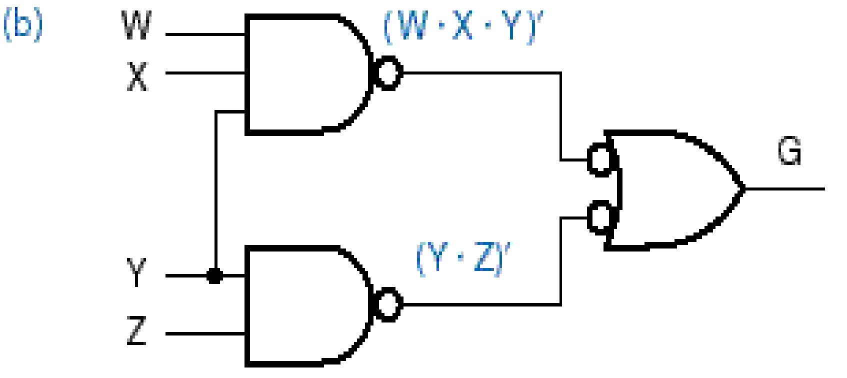


Two-level implementations

a) **Original SOP**
b) **Implementation with NANDs**

AND-OR (SOP) Emulation Using NANDs (cont.)





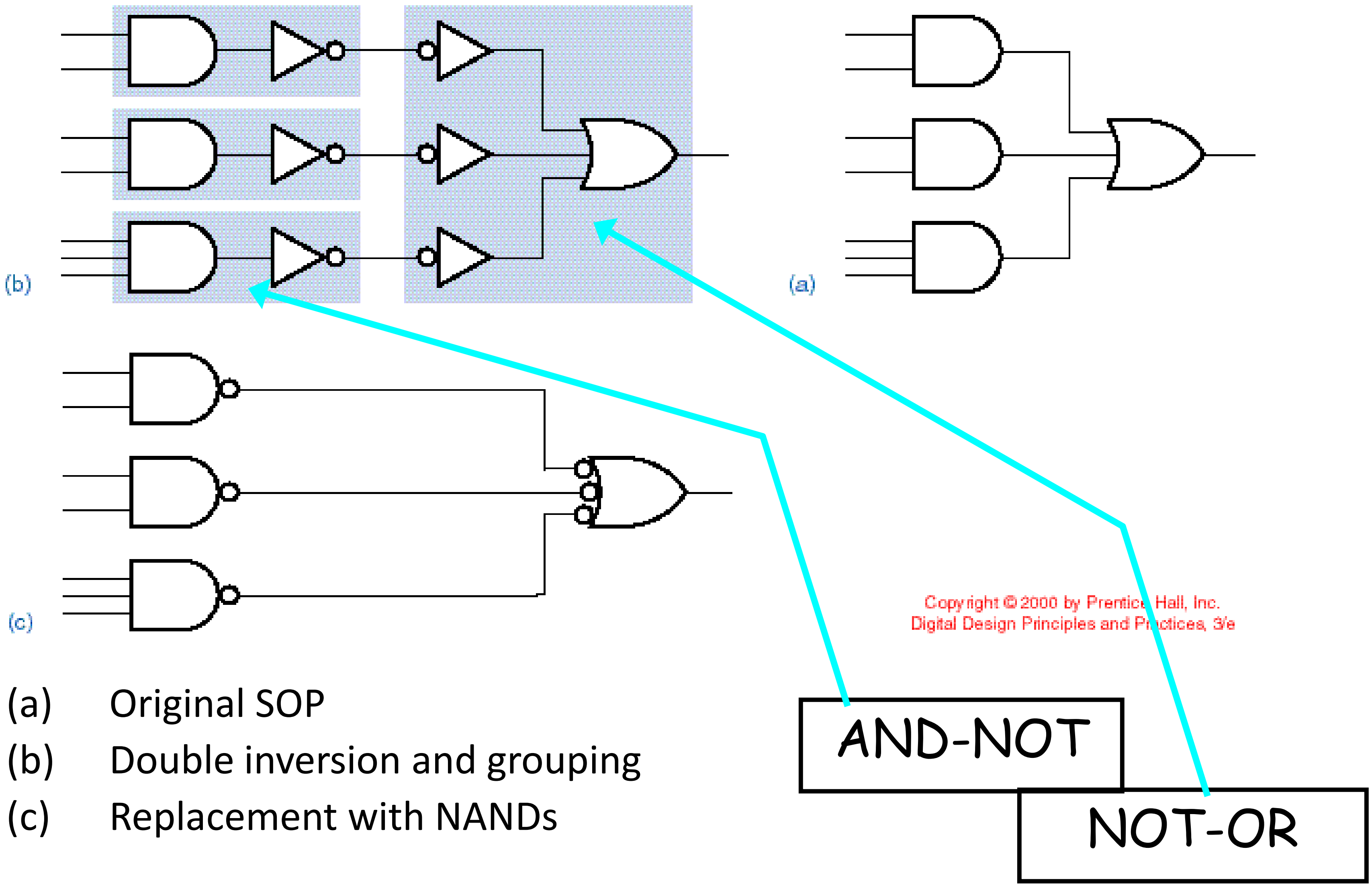
Verify:

(a) $G = WXY + YZ$

(b) $G = ((WXY)' \cdot (YZ)')'$

$= (WXY)'' + (YZ)'' = WXY + YZ$

SOP with NAND



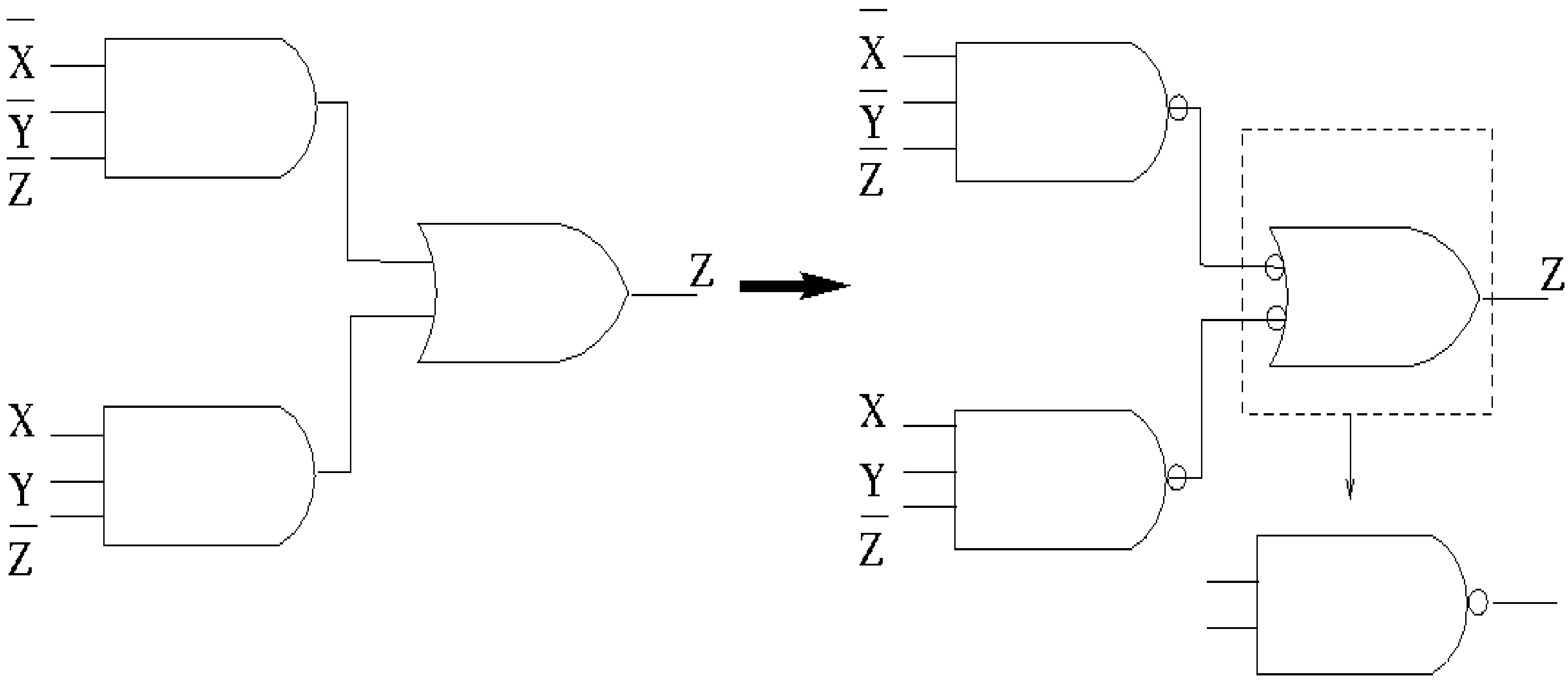
(a) Original SOP
(b) Double inversion and grouping
(c) Replacement with NANDs

Two-Level NAND Gate Implementation - Example

$F(X,Y,Z) = \Sigma m(0,6)$

1. Express F in SOP form:
   $F = X'Y'Z' + XYZ'$
2. Obtain the AND-OR implementation for F.
3. Add bubbles and inverters to transform AND-OR to NAND-NAND gates.

Example (cont.)



Two-level implementation with NANDs

F = X'Y'Z' + XYZ'

Multilevel NAND Circuits

Starting from a multilevel circuit:

1. Convert all AND gates to NAND gates with AND-NOT graphic symbols.
2. Convert all OR gates to NAND gates with NOT-OR graphic symbols.
3. Check all the bubbles in the diagram. For every bubble that is not counteracted by another bubble along the same line, insert a NOT gate or complement the input literal from its original appearance.

Example

Use NAND gates and NOT gates to implement

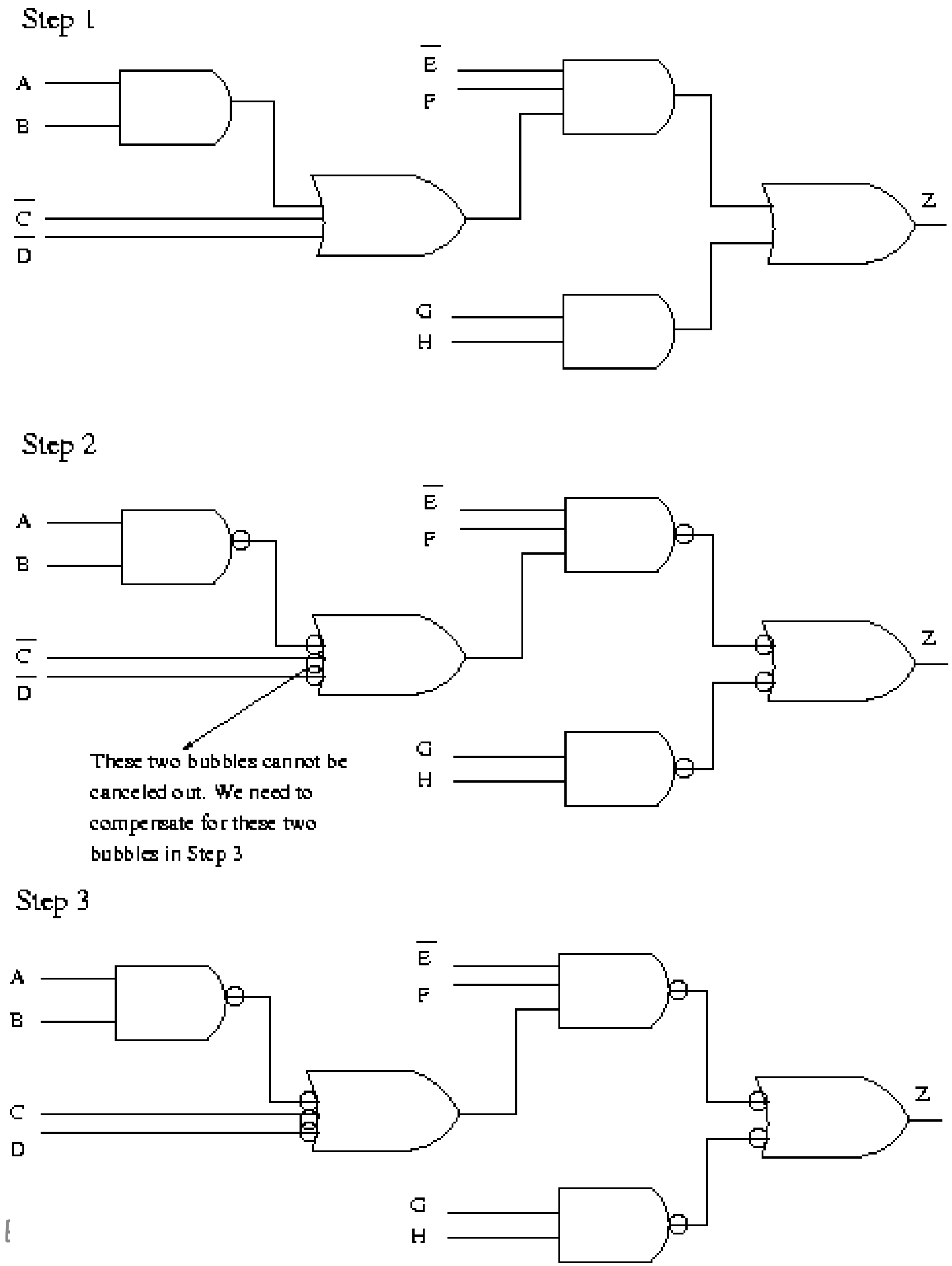
Z=E'F(AB+C'+D')+GH

AB

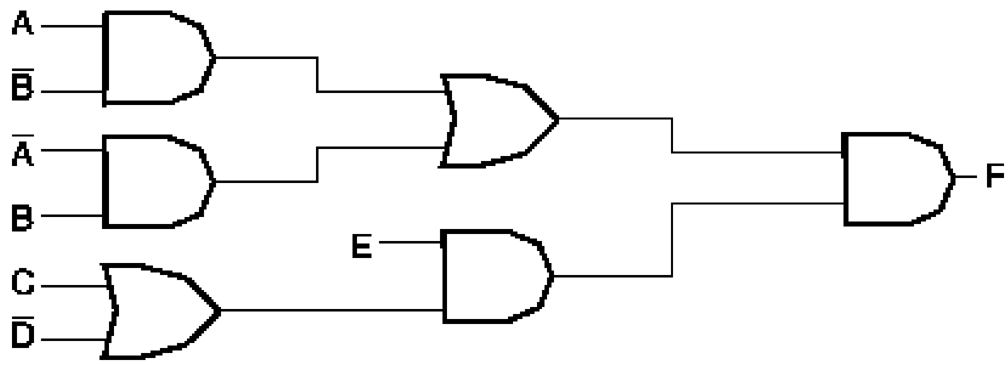
AB+C'+D'

E'F(AB+C'+D')

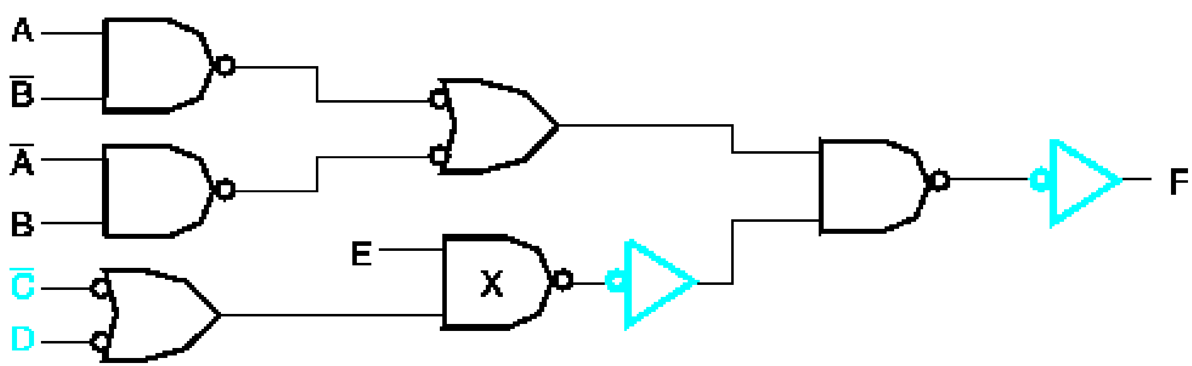
E'F(AB+C'+D')+GH



Yet Another Example!



(a) AND – OR gates



(b) NAND gates

Fig. 2-32 Implementing $F = (A\overline{B} + \overline{A}B)E(C + \overline{D})$

NOR Gate

- Also a “universal” gate because ANY digital circuit can be implemented with NOR gates alone.
- This can be similarly proven as with the NAND gate.

NOR Circuits

- To easily derive a NOR implementation of a boolean function:
  - Find a simplified POS
  - POS is an OR-AND circuit
  - Change OR-AND circuit to a NOR circuit
  - Use the alternative symbols below

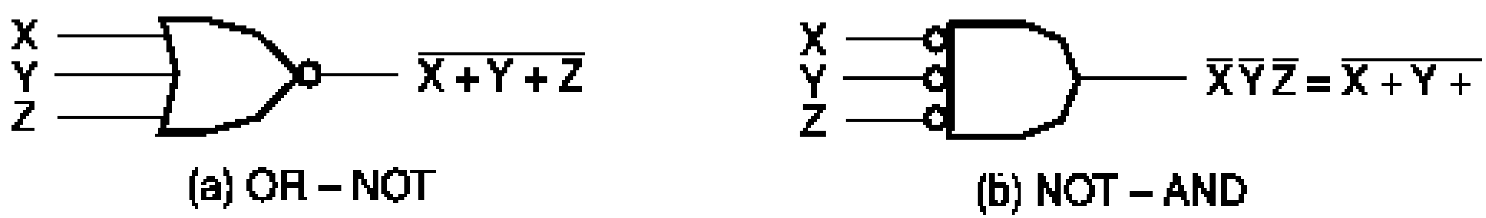


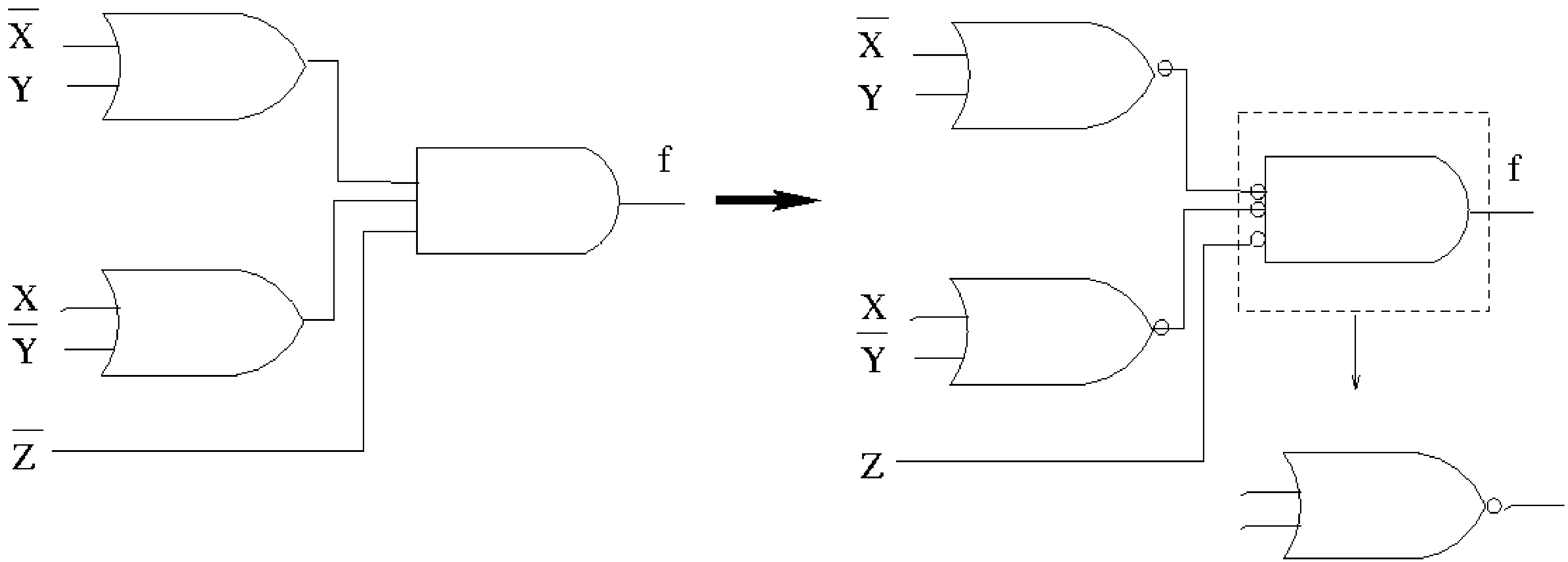
Fig. 2-34 Two Graphic Symbols for NOR Gate

Two-Level NOR Gate Implementation - Example

$F(X,Y,Z) = \Sigma m(0,6)$

1. Express F’ in SOP form:
   1. $F' = \Sigma m(1,2,3,4,5,7)$
      $= X'Y'Z + X'YZ' + X'YZ + XY'Z' + XY'Z + XYZ$
   2. $F' = XY' + X'Y + Z$
2. Take the complement of F’ to get F in the POS form:
   $F = (F')' = (X'+Y)(X+Y')Z'$
3. Obtain the OR-AND implementation for F.
4. Add bubbles and inverters to transform OR-AND implementation to NOR-NOR implementation.

Example (cont.)

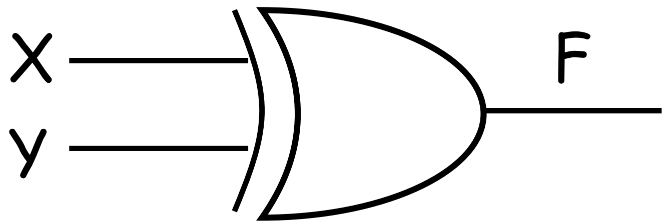


Two-level implementation with NORs

F = (F')' = (X'+Y)(X+Y')Z'

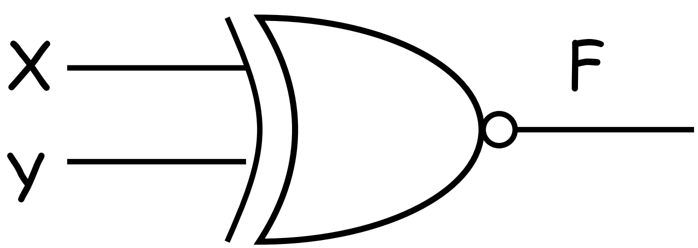
XOR and XNOR

XOR: "**not-equal**" **gate**



X	Y	$F = X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

XNOR: "**equal**" **gate**



X	Y	$F = \overline{X \oplus Y}$
0	0	1
0	1	0
1	0	0
1	1	1

Exclusive-OR (XOR) Function

- XOR (also $\oplus$) : the "not-equal" function
- $XOR(X,Y) = X \oplus Y = X'Y + XY'$
- Identities:
  - $X \oplus 0 = X$
  - $X \oplus 1 = X'$
  - $X \oplus X = 0$
  - $X \oplus X' = 1$
- Properties:
  - $X \oplus Y = Y \oplus X$
  - $(X \oplus Y) \oplus W = X \oplus (Y \oplus W)$

XOR function implementation

- XOR(a,b) = ab’ + a’b
- Straightforward: 5 gates
  - 2 inverters, two 2-input ANDs, one 2-input OR
  - 2 inverters & 3 2-input NANDs
- Nonstraightforward:
  - 4 NAND gates

XOR circuit with 4 NANDs

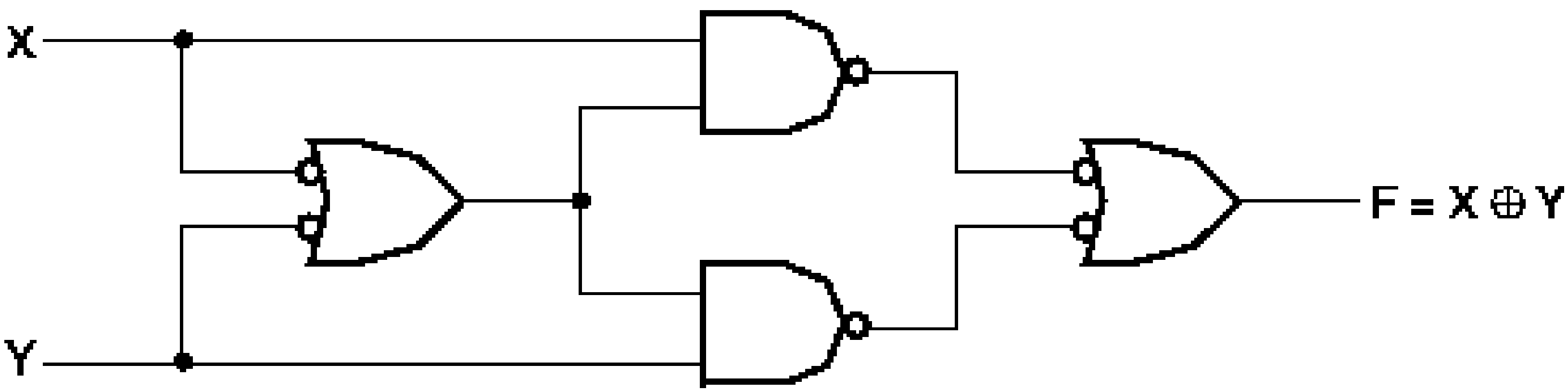


Fig. 2-37 Exclusive-OR Constructed with NAND Gates