



# Digital and Logic Circuit

## Module 6

### BOOLEAN ALGEBRA






WEEK 11 TO 12

CILO 8.

ILLUSTRATE AND CONSTRUCT  
TRUTH TABLES USING BOOLEAN  
NOTATIONS



# Topics

- Concepts and history of Boolean Algebra
- Postulates
- Basic Theorems
- Operator Precedence
- Cananonical and Standard forms
- Maxterm
- Minterm
- Sum
- Product
- Logic Gates

# Algebras

- What is an algebra?
  - *Mathematical system consisting of*
    - Set of elements
    - Set of operators
    - Axioms or postulates
- Why is it important?
  - *Defines rules of “calculations”*
- Example: arithmetic on natural numbers
  - *Set of elements:  $N = \{1, 2, 3, 4, \dots\}$*
  - *Operator:  $+$ ,  $-$ ,  $*$*
  - *Axioms: associativity, distributivity, closure, identity elements, etc.*
- Note: operators with two inputs are called binary
  - *Does not mean they are restricted to binary numbers!*
  - *Operator(s) with one input are called unary*

# Basic Terminologies

- A set is collection of having the same property.
  - *S: set, x and y: element or event*
  - *For example:  $S = \{1, 2, 3, 4\}$* 
    - If  $x = 2$ , then  $x \in S$ .
    - If  $y = 5$ , then  $y \notin S$ .
- A *binary operator* defines on a set S of elements is a rule that assigns, to each pair of elements from S, a unique element from S.
  - *For example: given a set S, consider  $a * b = c$  and  $*$  is a binary operator.*
  - *If (a, b) through  $*$  get c and  $a, b, c \in S$ , then  $*$  is a binary operator of S.*
  - *On the other hand, if  $*$  is not a binary operator of S and  $a, b \in S$ , then  $c \notin S$ .*

# Basic Terminologies

- The most common postulates used to formulate various algebraic structures are as follows:
  1. **Closure**: a set  $S$  is closed with respect to a binary operator if, for every pair of elements of  $S$ , the binary operator specifies a rule for obtaining a unique element of  $S$ .
    - For example, natural numbers  $N=\{1,2,3,\dots\}$  is closed w.r.t. the binary operator  $+$  by the rule of arithmetic addition, since, for any  $a, b \in N$ , there is a unique  $c \in N$  such that
      - $a+b = c$
      - But operator  $-$  is not closed for  $N$ , because  $2-3 = -1$  and  $2, 3 \in N$ , but  $(-1) \notin N$ .
  2. **Associative law**: a binary operator  $*$  on a set  $S$  is said to be associative whenever
    - $(x * y) * z = x * (y * z)$  for all  $x, y, z \in S$ 
      - $(x+y)+z = x+(y+z)$
  3. **Commutative law**: a binary operator  $*$  on a set  $S$  is said to be commutative whenever
    - $x * y = y * x$  for all  $x, y \in S$ 
      - $x+y = y+x$

# BASIC DEFINITIONS

4. *Identity element*: a set  $S$  is said to have an identity element with respect to a binary operation  $*$  on  $S$  if there exists an element  $e \in S$  with the property that

- $e * x = x * e = x$  for every  $x \in S$ 
  - $0 + x = x + 0 = x$  for every  $x \in I$ .  $I = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$ .
  - $1 * x = x * 1 = x$  for every  $x \in I$ .  $I = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$ .

5. *Inverse*: a set having the identity element  $e$  with respect to the binary operator to have an inverse whenever, for every  $x \in S$ , there exists an element  $y \in S$  such that

- $x * y = e$ 
  - The operator  $+$  over  $I$ , with  $e = 0$ , the inverse of an element  $a$  is  $(-a)$ , since  $a + (-a) = 0$ .

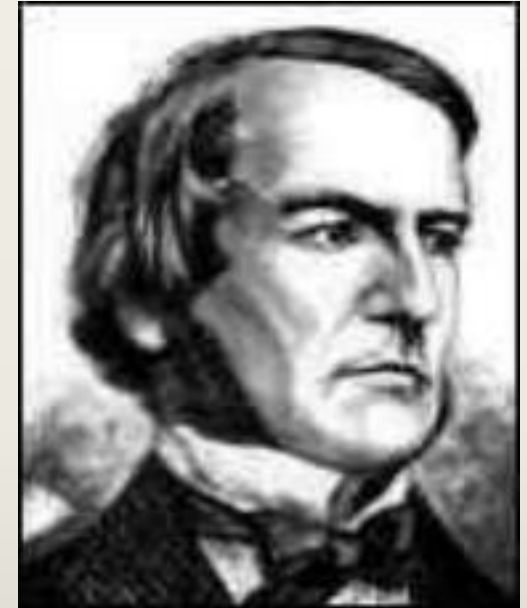
6. *Distributive law*: if  $*$  and  $.$  are two binary operators on a set  $S$ ,  $*$  is said to be distributive over  $.$  whenever

- $x * (y . z) = (x * y) . (x * z)$

# George Boole

## ■ Father of Boolean algebra

- He came up with a type of linguistic algebra, the three most basic operations of which were (and still are) AND, OR and NOT. It was these three functions that formed the basis of his premise, and were the only operations necessary to perform comparisons or basic mathematical functions.
- Boole's system (detailed in his 'An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities', 1854) was based on a binary approach, processing only two objects - the yes-no, true-false, on-off, zero-one approach.
- Surprisingly, given his standing in the academic community, Boole's idea was either criticized or completely ignored by the majority of his peers.
- Eventually, one bright student, Claude Shannon (1916-2001), picked up the idea and ran with it



George Boole (1815 - 1864)



# Axiomatic Definition of Boolean Algebra

- We need to define algebra for binary values
  - *Developed by George Boole in 1854*
- Huntington postulates for Boolean algebra (1904):
- $B = \{0, 1\}$  and two binary operations,  $+$  and  $\cdot$ .
  - *Closure with respect to operator  $+$  and operator  $\cdot$ .*
  - *Identity element 0 for operator  $+$  and 1 for operator  $\cdot$ .*
  - *Commutativity with respect to  $+$  and  $\cdot$ .*
$$x+y = y+x, \quad x \cdot y = y \cdot x$$
  - *Distributivity of  $\cdot$  over  $+$ , and  $+$  over  $\cdot$ .*
$$x \cdot (y+z) = (x \cdot y) + (x \cdot z) \quad \text{and} \quad x + (y \cdot z) = (x+y) \cdot (x+z)$$
    - Complement for every element  $x$  is  $x'$  with  $x+x'=1$ ,  $x \cdot x'=0$
  - *There are at least two elements  $x, y \in B$  such that  $x \neq y$*

# Boolean Algebra

## ■ Terminology:

- *Literal*: A variable or its complement
- *Product term*: literals connected by  $\cdot$
- *Sum term*: literals connected by  $+$

# Postulates of Two-Valued Boolean Algebra

- $B = \{0, 1\}$  and two binary operations,  $+$  and  $\cdot$ .
- The rules of operations: AND、OR and NOT.

## AND

$x$	$y$	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

## OR

$x$	$y$	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

## NOT

$x$	$x'$
0	1
1	0

1. Closure ( $+$  and  $\cdot$ )
2. The identity elements
  - (1)  $+: 0$
  - (2)  $\cdot: 1$

# Postulates of Two-Valued Boolean Algebra

- 3. The commutative laws
- 4. The distributive laws

$x$	$y$	$z$	$y+z$	$x \cdot (y+z)$	$x \cdot y$	$x \cdot z$	$(x \cdot y) + (x \cdot z)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

# Postulates of Two-Valued Boolean Algebra

## 5. Complement

- $x+x'=1 \rightarrow 0+0'=0+1=1; 1+1'=1+0=1$
- $x \cdot x'=0 \rightarrow 0 \cdot 0'=0 \cdot 1=0; 1 \cdot 1'=1 \cdot 0=0$

## 6. Has two distinct elements 1 and 0, with $0 \neq 1$

### ■ Note

- *A set of two elements*
- *$+$  : OR operation;  $\cdot$  : AND operation*
- *A complement operator: NOT operation*
- *Binary logic is a two-valued Boolean algebra*

# Duality

- The principle of *duality* is an important concept. This says that if an expression is valid in Boolean algebra, the dual of that expression is also valid.
- To form the dual of an expression, replace all + operators with . operators, all . operators with + operators, all ones with zeros, and all zeros with ones.
- Form the dual of the expression
$$a + (bc) = (a + b)(a + c)$$
- Following the replacement rules...
$$a(b + c) = ab + ac$$
- Take care not to alter the location of the parentheses if they are present.

# Basic Theorems

**Table 2.1**

*Postulates and Theorems of Boolean Algebra*

---

Postulate 2	(a) $x + 0 = x$	(b) $x \cdot 1 = x$
Postulate 5	(a) $x + x' = 1$	(b) $x \cdot x' = 0$
Theorem 1	(a) $x + x = x$	(b) $x \cdot x = x$
Theorem 2	(a) $x + 1 = 1$	(b) $x \cdot 0 = 0$
Theorem 3, involution	$(x')' = x$	
Postulate 3, commutative	(a) $x + y = y + x$	(b) $xy = yx$
Theorem 4, associative	(a) $x + (y + z) = (x + y) + z$	(b) $x(yz) = (xy)z$
Postulate 4, distributive	(a) $x(y + z) = xy + xz$	(b) $x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a) $(x + y)' = x'y'$	(b) $(xy)' = x' + y'$
Theorem 6, absorption	(a) $x + xy = x$	(b) $x(x + y) = x$

---

# Boolean Theorems

- Huntington's postulates define some rules

Post. 1: closure

Post. 2: (a)  $x+0=x$ , (b)  $x \cdot 1=x$

Post. 3: (a)  $x+y=y+x$ , (b)  $x \cdot y=y \cdot x$

Post. 4: (a)  $x(y+z) = xy+xz$ ,

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

Post. 5: (a)  $x+x'=1$ , (b)  $x \cdot x'=0$

- Need more rules to modify algebraic expressions

- *Theorems that are derived from postulates*

- What is a theorem?

- *A formula or statement that is derived from postulates (or other proven theorems)*

- Basic theorems of Boolean algebra

- *Theorem 1 (a):  $x + x = x$  (b):  $x \cdot x = x$*

- *Looks straightforward, but needs to be proven !*



# Proof of $x+x=x$

- We can only use Huntington postulates:

## Huntington postulates:

**Post. 2:** (a)  $x+0=x$ , (b)  $x \cdot 1=x$

**Post. 3:** (a)  $x+y=y+x$ , (b)  $x \cdot y=y \cdot x$

**Post. 4:** (a)  $x(y+z) = xy+xz$ ,  
(b)  $x+yz = (x+y)(x+z)$

**Post. 5:** (a)  $x+x'=1$ , (b)  $x \cdot x'=0$

- Show that  $x+x=x$ .

$$\begin{aligned}x+x &= (x+x) \cdot 1 && \text{by 2(b)} \\&= (x+x)(x+x') && \text{by 5(a)} \\&= x+xx' && \text{by 4(b)} \\&= x+0 && \text{by 5(b)} \\&= x && \text{by 2(a)}\end{aligned}$$

*Q.E.D.*

- We can now use Theorem 1(a) in future proofs

# Proof of $x \cdot x = x$

- Similar to previous proof

Huntington postulates:

**Post. 2:** (a)  $x+0=x$ , (b)  $x \cdot 1=x$

**Post. 3:** (a)  $x+y=y+x$ , (b)  $x \cdot y=y \cdot x$

**Post. 4:** (a)  $x(y+z) = xy+xz$ ,  
(b)  $x+yz = (x+y)(x+z)$

**Post. 5:** (a)  $x+x'=1$ , (b)  $x \cdot x'=0$

**Th. 1:** (a)  $x+x=x$

- Show that  $x \cdot x = x$ .

$$\begin{aligned} x \cdot x &= xx+0 && \text{by 2(a)} \\ &= xx+xx' && \text{by 5(b)} \\ &= x(x+x') && \text{by 4(a)} \\ &= x \cdot 1 && \text{by 5(a)} \\ &= x && \text{by 2(b)} \end{aligned}$$

*Q.E.D.*

# Proof of $x+1=1$

## ■ Theorem 2(a): $x + 1 = 1$

$$x + 1 = 1. \quad (x + 1) \quad \text{by 2(b)}$$

$$= (x + x')(x + 1) \quad 5(a)$$

$$= x + x' 1 \quad 4(b)$$

$$= x + x' \quad 2(b)$$

$$= 1 \quad 5(a)$$

## ■ Theorem 2(b): $x \cdot 0 = 0$ by duality

## ■ Theorem 3: $(x')' = x$

- Postulate 5 defines the complement of  $x$ ,  $x + x' = 1$  and  $x x' = 0$
- The complement of  $x'$  is  $x$  is also  $(x')'$

### Huntington postulates:

**Post. 2:** (a)  $x+0=x$ , (b)  $x \cdot 1=x$

**Post. 3:** (a)  $x+y=y+x$ , (b)

$$x \cdot y = y \cdot x$$

**Post. 4:** (a)  $x(y+z) = xy+xz$ ,

$$(b) \quad x+yz = (x+y)(x+z)$$

**Post. 5:** (a)  $x+x'=1$ , (b)  $x \cdot x'=0$

**Th. 1:** (a)  $x+x=x$

# Absorption Property (Covering)

- Theorem 6(a):  $x + xy = x$ 
  - $x + xy = x \cdot 1 + xy$  by 2(b)
  - $= x(1 + y)$  4(a)
  - $= x(y + 1)$  3(a)
  - $= x \cdot 1$  Th 2(a)
  - $= x$  2(b)

Huntington postulates:

**Post. 2:** (a)  $x+0=x$ , (b)  $x \cdot 1=x$

**Post. 3:** (a)  $x+y=y+x$ , (b)  $x \cdot y=y \cdot x$

**Post. 4:** (a)  $x(y+z) = xy+xz$ ,  
(b)  $x+yz = (x+y)(x+z)$

**Post. 5:** (a)  $x+x'=1$ , (b)  $x \cdot x'=0$

**Th. 1:** (a)  $x+x=x$

- Theorem 6(b):  $x(x + y) = x$  by duality
- By means of truth table (another way to proof )

$x$	$y$	$xy$	$x+xy$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

# DeMorgan's Theorem

- Theorem 5(a):  $(x + y)' = x'y'$
- Theorem 5(b):  $(xy)' = x' + y'$
- By means of truth table

$x$	$y$	$x'$	$y'$	$x+y$	$(x+y)'$	$x'y'$	$xy$	$x'+y'$	$(xy)'$
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

# Consensus Theorem

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

■ **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).

# Operator Precedence

- The operator precedence for evaluating Boolean Expression is
  - *Parentheses*
  - *NOT*
  - *AND*
  - *OR*
- Examples
  - $x y' + z$
  - $(x y + z)'$

# Boolean Functions

- A Boolean function
  - *Binary variables*
  - *Binary operators OR and AND*
  - *Unary operator NOT*
  - *Parentheses*
- Examples
  - $F_1 = x y z'$
  - $F_2 = x + y'z$
  - $F_3 = x' y' z + x' y z + x y'$
  - $F_4 = x y' + x' z$



# Boolean Functions

■ *The truth table of  $2^n$  entries*

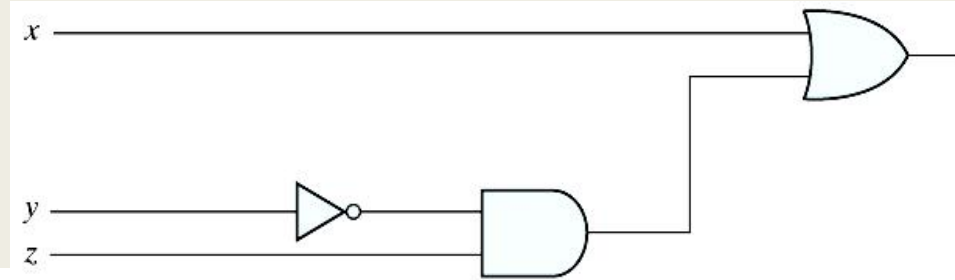
$x$	$y$	$z$	$F_1$	$F_2$	$F_3$	$F_4$
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

- Two Boolean expressions may specify the same function
  - $F_3 = F_4$

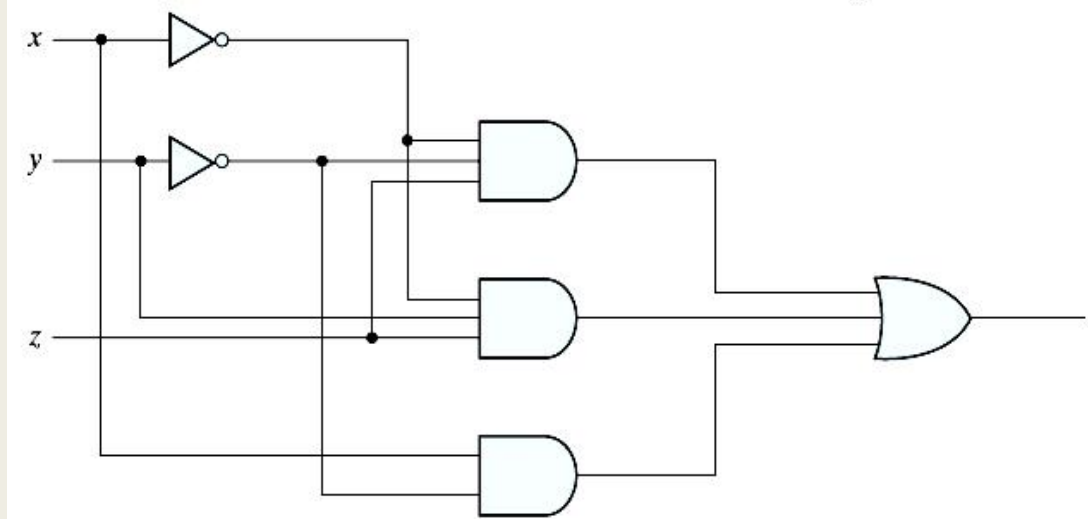
# Boolean Functions

## ■ Implementation with logic gates

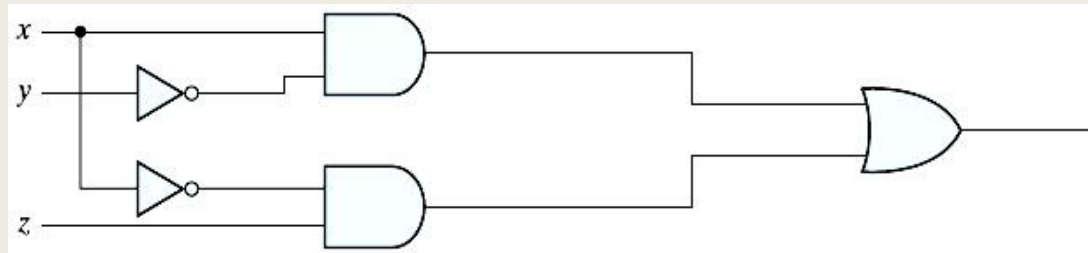
- $F_4$  is more economical



$$F_2 = x + y'z$$



$$F_3 = x'y'z + x'yz + xy'$$



$$F_4 = xy' + x'z$$

# Algebraic Manipulation

## ■ To minimize Boolean expressions

- *Literal*: a primed or unprimed variable (an input to a gate)
- *Term*: an implementation with a gate
- The minimization of the number of literals and the number of terms  $\rightarrow$  a circuit with less equipment
- It is a hard problem (no specific rules to follow)

## ■ Example 2.1

1.  $x(x'+y) = xx' + xy = 0 + xy = xy$
2.  $x+x'y = (x+x')(x+y) = 1(x+y) = x+y$
3.  $(x+y)(x+y') = x+xy+xy'+yy' = x(1+y+y') = x$
4.  $xy + x'z + yz = xy + x'z + yz(x+x') = xy + x'z + yzx + yzx' = xy(1+z) + x'z(1+y) = xy + x'z$
5.  $(x+y)(x'+z)(y+z) = (x+y)(x'+z)$ , by duality from function 4.  
(*consensus theorem* with duality)

# Complement of a Function

- An interchange of 0's for 1's and 1's for 0's in the value of  $F$ 
  - *By DeMorgan's theorem*
  - $(A+B+C)' = (A+X)'$       *let  $B+C = X$*   
     $= A'X'$       *by theorem 5(a) (DeMorgan's)*  
     $= A'(B+C)'$       *substitute  $B+C = X$*   
     $= A'(B'C')$       *by theorem 5(a) (DeMorgan's)*  
     $= A'B'C'$       *by theorem 4(b) (associative)*
- *Generalizations:* a function is obtained by interchanging AND and OR operators and complementing each literal.
  - $(A+B+C+D+ \dots +F)' = A'B'C'D' \dots F'$
  - $(ABCD \dots F)' = A'+ B'+C'+D' \dots +F'$

# Examples

## ■ Example 2.2

- $F_1' = (x'yz' + x'y'z)' = (x'yz')' (x'y'z)' = (x+y'+z) (x+y+z')$
- $F_2' = [x(y'z'+yz)]' = x' + (y'z'+yz)' = x' + (y'z')' (yz)'$   
 $= x' + (y+z) (y'+z')$   
 $= x' + yz' + y'z$

## ■ Example 2.3: a simpler procedure

- *Take the dual of the function and complement each literal*

1.  $F_1 = x'yz' + x'y'z.$

*The dual of  $F_1$  is  $(x'+y+z') (x'+y'+z).$*

*Complement each literal:  $(x+y'+z)(x+y+z') = F_1'$*

2.  $F_2 = x(y'z' + yz).$

*The dual of  $F_2$  is  $x+(y'+z') (y+z).$*

*Complement each literal:  $x'+(y+z)(y' +z') = F_2'$*

# Canonical and Standard Forms

## Minterms and Maxterms

- A minterm (standard product): an AND term consists of all literals in their normal form or in their complement form.
  - *For example, two binary variables  $x$  and  $y$ ,*
    - $xy, xy', x'y, x'y'$
  - *It is also called a standard product.*
  - *$n$  variables can be combined to form  $2^n$  minterms.*
- A maxterm (standard sums): an OR term
  - *It is also call a standard sum.*
  - *$2^n$  maxterms.*

# Minterms and Maxterms

- Each maxterm is the complement of its corresponding minterm, and vice versa.

**Table 2.3**  
*Minterms and Maxterms for Three Binary Variables*

<i>x</i>	<i>y</i>	<i>z</i>	Minterms		Maxterms	
			Term	Designation	Term	Designation
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	$xyz$	$m_7$	$x' + y' + z'$	$M_7$

# Minterms and Maxterms

- An Boolean function can be expressed by
  - *A truth table*
  - *Sum of minterms*
  - $f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$  (Minterms)
  - $f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$  (Minterms)

**Table 2.4**  
*Functions of Three Variables*

<b>x</b>	<b>y</b>	<b>z</b>	<b>Function <math>f_1</math></b>	<b>Function <math>f_2</math></b>
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



# Minterms and Maxterms

- The complement of a Boolean function
  - *The minterms that produce a 0*
  - $f_1' = m_0 + m_2 + m_3 + m_5 + m_6 = x'y'z' + x'yz' + x'yz + xy'z + xyz'$
  - $f_1 = (f_1')' =$   
 $(x+y+z)(x+y'+z)(x+y'+z')(x'+y+z')(x'+y'+z) = M_0 M_2 M_3 M_5 M_6$
  - $f_2 = (x+y+z)(x+y+z')(x+y'+z)(x'+y+z) = M_0 M_1 M_2 M_4$
- Any Boolean function can be expressed as
  - *A sum of minterms (“sum” meaning the ORing of terms).*
  - *A product of maxterms (“product” meaning the ANDing of terms).*
  - *Both boolean functions are said to be in Canonical form.*

# Sum of Minterms

- Sum of minterms: there are  $2^n$  minterms and  $2^{2n}$  combinations of function with  $n$  Boolean variables.
- Example 2.4: express  $F = A+BC'$  as a sum of minterms.
  - $F = A+B'C = A(B+B') + B'C = AB + AB' + B'C = AB(C+C') + AB'(C+C') + (A+A')B'C = ABC+ABC'+AB'C+AB'C'+A'B'C$
  - $F = A'B'C + AB'C' + AB'C + ABC' + ABC = m_1 + m_4 + m_5 + m_6 + m_7$
  - $F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$
  - or, built the truth table first

**Table 2.5**

*Truth Table for  $F = A + B'C$*

<b>A</b>	<b>B</b>	<b>C</b>	<b>F</b>
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

# Product of Maxterms

- Product of maxterms: using distributive law to expand.
  - $x + yz = (x + y)(x + z) = (x+y+zz')(x+z+yy') = (x+y+z)(x+y+z')(x+y'+z)$
- Example 2.5: express  $F = xy + x'z$  as a product of maxterms.
  - $F = xy + x'z = (xy + x')(xy + z) = (x+x')(y+x')(x+z)(y+z) = (x'+y)(x+z)(y+z)$
  - $x'+y = x' + y + zz' = (x'+y+z)(x'+y+z')$
  - $F = (x+y+z)(x+y'+z)(x'+y+z)(x'+y+z') = M_0M_2M_4M_5$
  - $F(x, y, z) = \prod(0, 2, 4, 5)$

# Conversion between Canonical Forms

- The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function.
  - $F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$
  - Thus,  $F'(A, B, C) = \Sigma(0, 2, 3)$
  - By DeMorgan's theorem
$$F(A, B, C) = \Pi(0, 2, 3)$$
$$F'(A, B, C) = \Pi(1, 4, 5, 6, 7)$$
  - $m_j' = M_j$
  - Sum of minterms = product of maxterms
  - Interchange the symbols  $\Sigma$  and  $\Pi$  and list those numbers missing from the original form
    - $\Sigma$  of 1's
    - $\Pi$  of 0's

■ Example

- $F = xy + x'z$
- $F(x, y, z) = \Sigma(1, 3, 6, 7)$
- $F(x, y, z) = \Pi(0, 2, 4, 6)$

**Table 2.6**

*Truth Table for  $F = xy + x'z$*

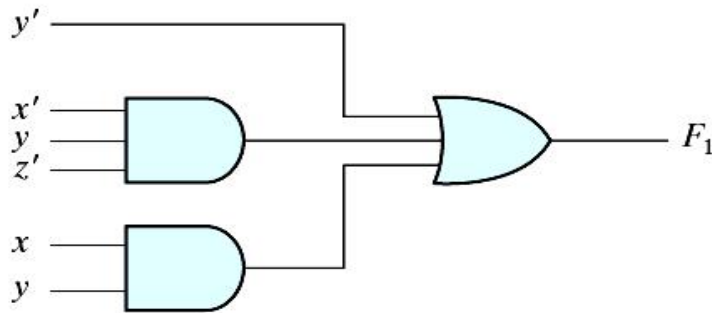
<b>x</b>	<b>y</b>	<b>z</b>	<b>F</b>
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

# Standard Forms

- Canonical forms are very seldom the ones with the least number of literals.
- Standard forms: the terms that form the function may obtain one, two, or any number of literals.
  - *Sum of products:*  $F_1 = y' + xy + x'yz'$
  - *Product of sums:*  $F_2 = x(y' + z)(x' + y + z')$
  - $F_3 = A'B'CD + ABC'D'$

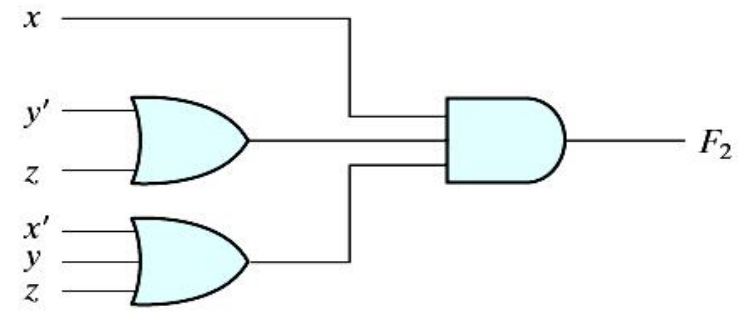
# Implementation

## ■ Two-level implementation



(a) Sum of Products

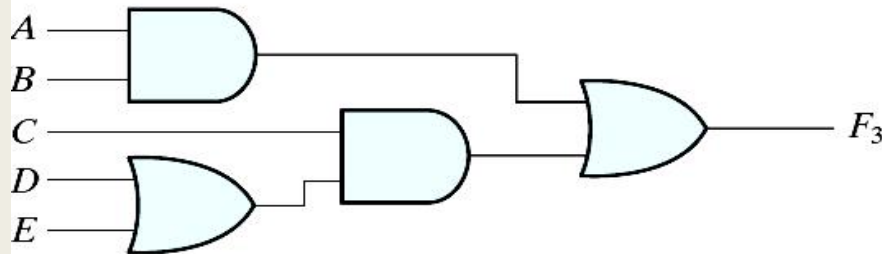
$$F_1 = y' + xy + x'yz'$$



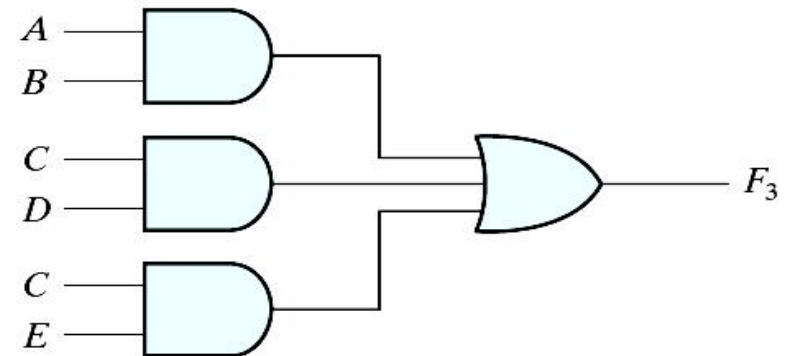
(b) Product of Sums

$$F_2 = x(y' + z)(x' + y + z')$$

## ■ Multi-level implementation



(a)  $AB + C(D + E)$



(b)  $AB + CD + CE$

## 2.7 Other Logic Operations (

- $2^n$  rows in the truth table of  $n$  binary variables.
- $2^{2^n}$  functions for  $n$  binary variables.
- 16 functions of two binary variables.

**Table 2.7**

*Truth Tables for the 16 Functions of Two Binary Variables*

<i>x</i>	<i>y</i>	<i>F</i> <sub>0</sub>	<i>F</i> <sub>1</sub>	<i>F</i> <sub>2</sub>	<i>F</i> <sub>3</sub>	<i>F</i> <sub>4</sub>	<i>F</i> <sub>5</sub>	<i>F</i> <sub>6</sub>	<i>F</i> <sub>7</sub>	<i>F</i> <sub>8</sub>	<i>F</i> <sub>9</sub>	<i>F</i> <sub>10</sub>	<i>F</i> <sub>11</sub>	<i>F</i> <sub>12</sub>	<i>F</i> <sub>13</sub>	<i>F</i> <sub>14</sub>	<i>F</i> <sub>15</sub>
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

- All the new symbols except for the exclusive-OR symbol are not in common use by digital designers.



# Boolean Expressions

**Table 2.8**

*Boolean Expressions for the 16 Functions of Two Variables*

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	$x$ and $y$
$F_2 = xy'$	$x/y$	Inhibition	$x$ , but not $y$
$F_3 = x$		Transfer	$x$
$F_4 = x'y$	$y/x$	Inhibition	$y$ , but not $x$
$F_5 = y$		Transfer	$y$
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	$x$ or $y$ , but not both
$F_7 = x + y$	$x + y$	OR	$x$ or $y$
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	$x$ equals $y$
$F_{10} = y'$	$y'$	Complement	Not $y$
$F_{11} = x + y'$	$x \subset y$	Implication	If $y$ , then $x$
$F_{12} = x'$	$x'$	Complement	Not $x$
$F_{13} = x' + y$	$x \supset y$	Implication	If $x$ , then $y$
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

# Digital Logic Gates

- Boolean expression: AND, OR and NOT operations
- Constructing gates of other logic operations
  - *The feasibility and economy;*
  - *The possibility of extending gate's inputs;*
  - *The basic properties of the binary operations (commutative and associative);*
  - *The ability of the gate to implement Boolean functions.*

# Standard Gates

- Consider the 16 functions in Table 2.8 (slide 33)
  - Two are equal to a constant ( $F_0$  and  $F_{15}$ ).
  - Four are repeated twice ( $F_4$ ,  $F_5$ ,  $F_{10}$  and  $F_{11}$ ).
  - Inhibition ( $F_2$ ) and implication ( $F_{13}$ ) are not commutative or associative.
  - The other eight: complement ( $F_{12}$ ), transfer ( $F_3$ ), AND ( $F_1$ ), OR ( $F_7$ ), NAND ( $F_{14}$ ), NOR ( $F_8$ ), XOR ( $F_6$ ), and equivalence (XNOR) ( $F_9$ ) are used as standard gates.
  - Complement: inverter.
  - Transfer: buffer (increasing drive strength).
  - Equivalence: XNOR.

# Summary of Logic Gates

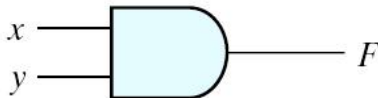
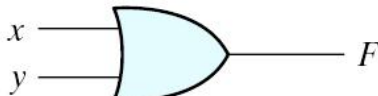
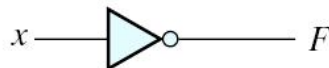
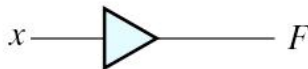
Name	Graphic symbol	Algebraic function	Truth table															
AND		$F = xy$	<table><tr><th><math>x</math></th><th><math>y</math></th><th><math>F</math></th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	$x$	$y$	$F$	0	0	0	0	1	0	1	0	0	1	1	1
$x$	$y$	$F$																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = x + y$	<table><tr><th><math>x</math></th><th><math>y</math></th><th><math>F</math></th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	$x$	$y$	$F$	0	0	0	0	1	1	1	0	1	1	1	1
$x$	$y$	$F$																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Inverter		$F = x'$	<table><tr><th><math>x</math></th><th><math>F</math></th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	$x$	$F$	0	1	1	0									
$x$	$F$																	
0	1																	
1	0																	
Buffer		$F = x$	<table><tr><th><math>x</math></th><th><math>F</math></th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	$x$	$F$	0	0	1	1									
$x$	$F$																	
0	0																	
1	1																	

Figure 2.5 Digital logic gates

# Summary of Logic Gates

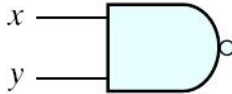
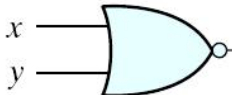
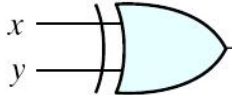
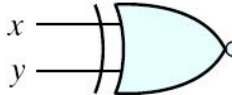
NAND	 $F = (xy)'$	<table> <tr> <th><math>x</math></th><th><math>y</math></th><th><math>F</math></th></tr> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </table>	$x$	$y$	$F$	0	0	1	0	1	1	1	0	1	1	1	0
$x$	$y$	$F$															
0	0	1															
0	1	1															
1	0	1															
1	1	0															
NOR	 $F = (x + y)'$	<table> <tr> <th><math>x</math></th><th><math>y</math></th><th><math>F</math></th></tr> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </table>	$x$	$y$	$F$	0	0	1	0	1	0	1	0	0	1	1	0
$x$	$y$	$F$															
0	0	1															
0	1	0															
1	0	0															
1	1	0															
Exclusive-OR (XOR)	 $F = xy' + x'y$ $= x \oplus y$	<table> <tr> <th><math>x</math></th><th><math>y</math></th><th><math>F</math></th></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </table>	$x$	$y$	$F$	0	0	0	0	1	1	1	0	1	1	1	0
$x$	$y$	$F$															
0	0	0															
0	1	1															
1	0	1															
1	1	0															
Exclusive-NOR or equivalence	 $F = xy + x'y'$ $= (x \oplus y)'$	<table> <tr> <th><math>x</math></th><th><math>y</math></th><th><math>F</math></th></tr> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	$x$	$y$	$F$	0	0	1	0	1	0	1	0	0	1	1	1
$x$	$y$	$F$															
0	0	1															
0	1	0															
1	0	0															
1	1	1															

Figure 2.5 Digital logic gates

# Multiple Inputs

- Extension to multiple inputs
  - *A gate can be extended to multiple inputs.*
    - If its binary operation is commutative and associative.
  - *AND and OR are commutative and associative.*
    - OR
      - $x+y = y+x$
      - $(x+y)+z = x+(y+z) = x+y+z$
    - AND
      - $xy = yx$
      - $(x\ y)z = x(y\ z) = x\ y\ z$

# Multiple Inputs

- *NAND and NOR are commutative but not associative  $\rightarrow$  they are not extendable.*

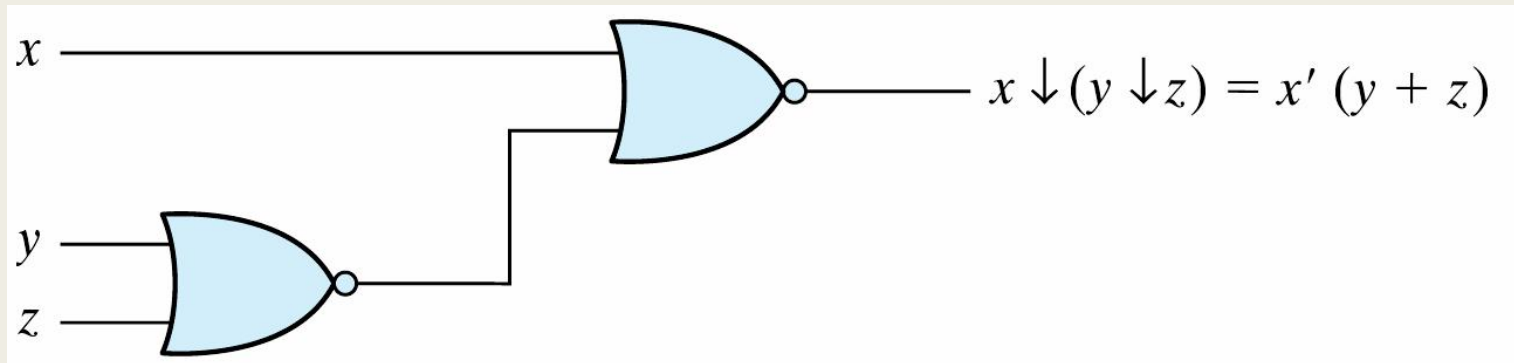
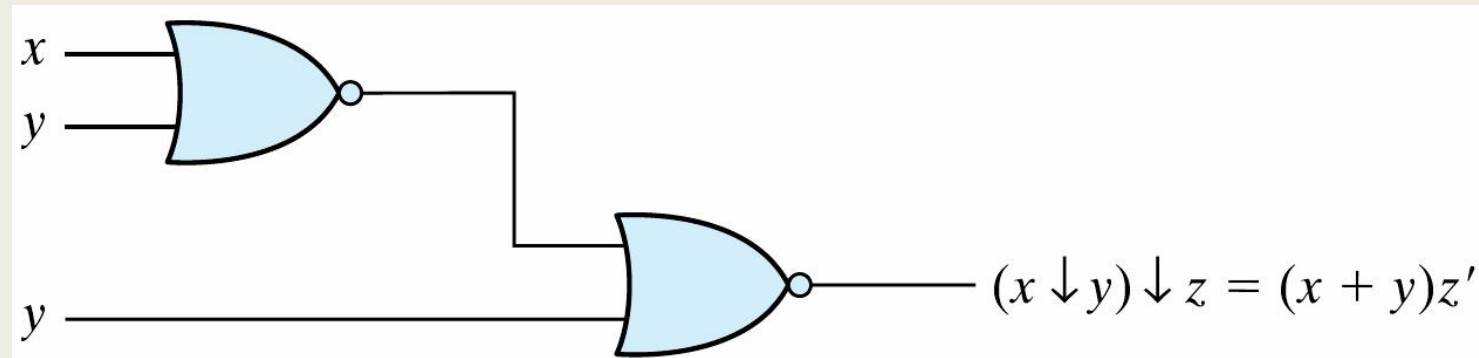
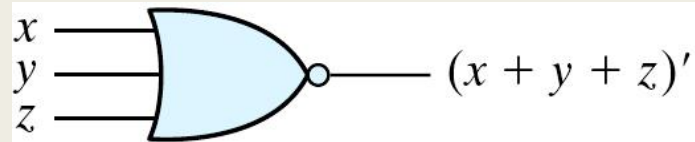


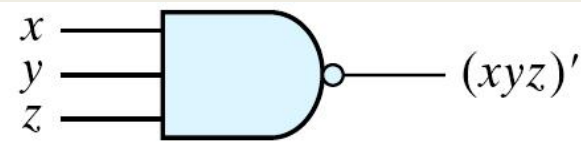
Figure 2.6 Demonstrating the nonassociativity of the NOR operator;  
 $(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$

# Multiple Inputs

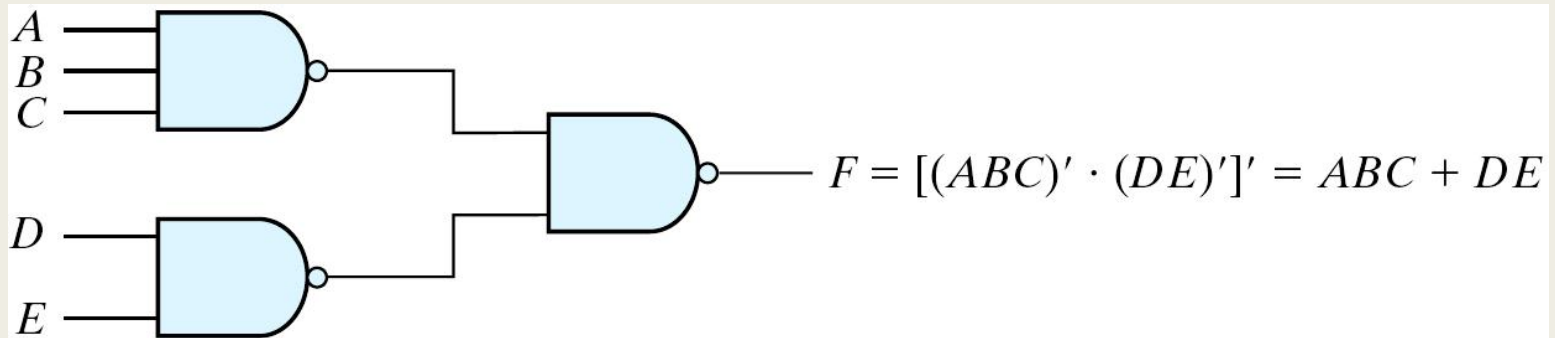
- Multiple NOR = a complement of OR gate, Multiple NAND = a complement of AND.
- The cascaded NAND operations = sum of products.
- The cascaded NOR operations = product of sums.



(a) 3-input NOR gate



(b) 3-input NAND gate



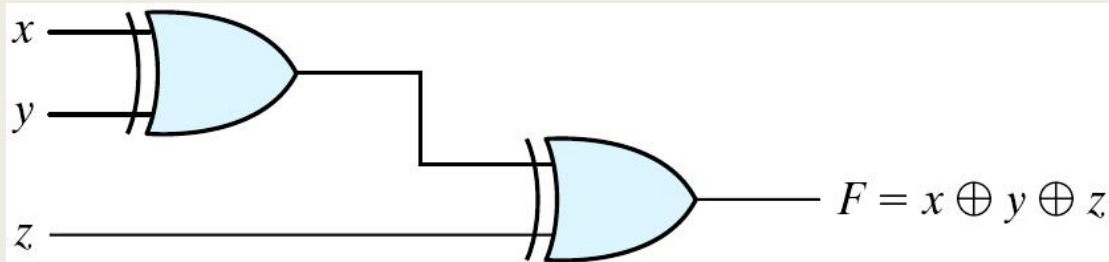
(c) Cascaded NAND gates

Figure 2.7 Multiple-input and cascaded NOR and NAND gates

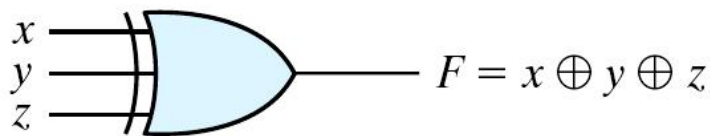


# Multiple Inputs

- The XOR and XNOR gates are commutative and associative.
- Multiple-input XOR gates are uncommon?
- XOR is an odd function: it is equal to 1 if the inputs variables have an odd number of 1's.



(a) Using 2-input gates



(b) 3-input gate

$x$	$y$	$z$	$F$
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

(c) Truth table

# Positive and Negative Logic

## Positive and Negative Logic

- *Two signal values  $\Leftrightarrow$  two logic values*
- *Positive logic:  $H=1$ ;  $L=0$*
- *Negative logic:  $H=0$ ;  $L=1$*

## Consider a TTL gate

- *A positive logic AND gate*
- *A negative logic OR gate*
- *The positive logic is used in this book*

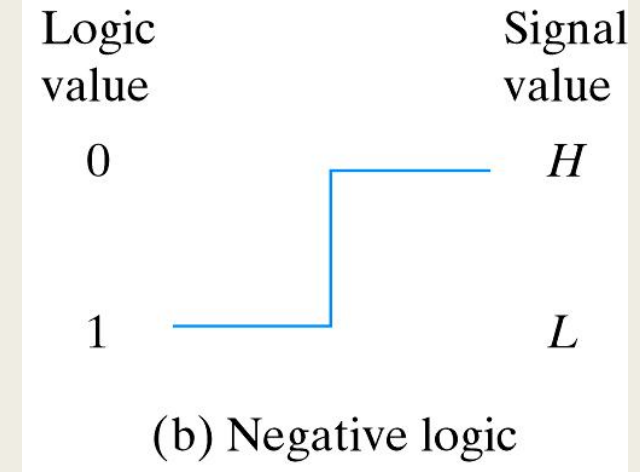
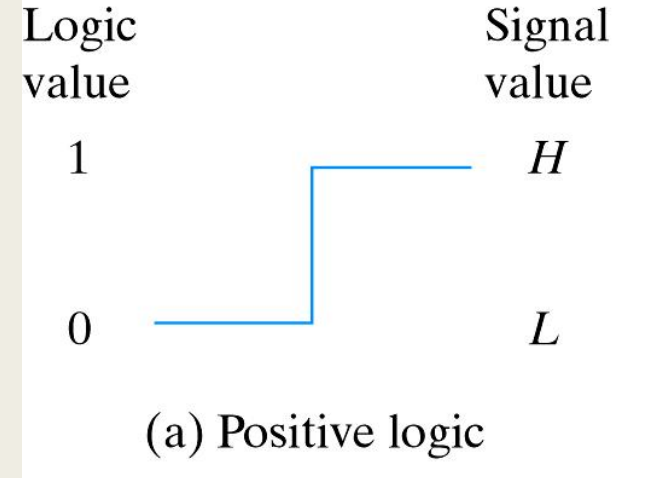


Figure 2.9 Signal assignment and logic polarity

# Positive and Negative Logic

$x$	$y$	$z$
$L$	$L$	$L$
$L$	$H$	$L$
$H$	$L$	$L$
$H$	$H$	$H$

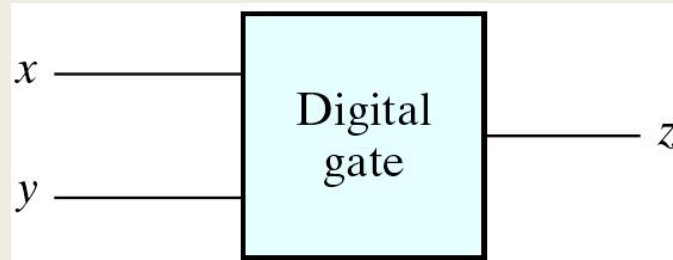
(a) Truth table with  $H$  and  $L$

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

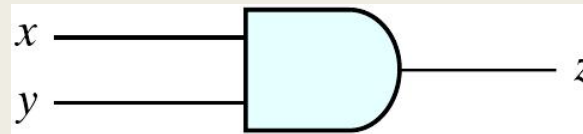
(c) Truth table for positive logic

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

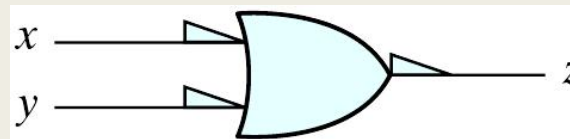
(e) Truth table for negative logic



(b) Gate block diagram



(d) Positive logic AND gate



(f) Negative logic OR gate

# 2.9 Integrated Circuits

## Level of Integration

- An IC (a chip)
- Examples:
  - *Small-scale Integration (SSI): < 10 gates*
  - *Medium-scale Integration (MSI): 10 ~ 100 gates*
  - *Large-scale Integration (LSI): 100 ~ xk gates*
  - *Very Large-scale Integration (VLSI): > xk gates*
- VLSI
  - *Small size (compact size)*
  - *Low cost*
  - *Low power consumption*
  - *High reliability*
  - *High speed*

# Digital Logic Families

- Digital logic families: circuit technology
  - *TTL: transistor-transistor logic (dying?)*
  - *ECL: emitter-coupled logic (high speed, high power consumption)*
  - *MOS: metal-oxide semiconductor (NMOS, high density)*
  - *CMOS: complementary MOS (low power)*
  - *BiCMOS: high speed, high density*

# Digital Logic Families

- The characteristics of digital logic families
  - *Fan-out: the number of standard loads that the output of a typical gate can drive.*
  - *Power dissipation.*
  - *Propagation delay: the average transition delay time for the signal to propagate from input to output.*
  - *Noise margin: the minimum of external noise voltage that caused an undesirable change in the circuit output.*

# CAD

- CAD – Computer-Aided Design
  - *Millions of transistors*
  - *Computer-based representation and aid*
  - *Automatic the design process*
  - *Design entry*
    - Schematic capture
    - HDL – Hardware Description Language
      - *Verilog, VHDL*
  - *Simulation*
  - *Physical realization*
    - ASIC, FPGA, PLD

# Chip Design

- Why is it better to have more gates on a single chip?
  - *Easier to build systems*
  - *Lower power consumption*
  - *Higher clock frequencies*
- What are the drawbacks of large circuits?
  - *Complex to design*
  - *Chips have design constraints*
  - *Hard to test*
- Need tools to help develop integrated circuits
  - *Computer Aided Design (CAD) tools*
  - *Automate tedious steps of design process*
  - *Hardware description language (HDL) describe circuits*
  - *VHDL (see the lab) is one such system*



# Summary

BOOLEAN ALGEBRA		
OR		AND
$x + 0 = x$ $x + 1 = 1$ $x + x = x$ $x + \bar{x} = 1$	$\left\{ \begin{array}{c} \text{DUALITY} \\ \text{THEOREMS} \end{array} \right\}$	$x \cdot 1 = x$ $x \cdot 0 = 0$ $x \cdot x = x$ $x \cdot \bar{x} = 0$
DOUBLE INVERSION		
$x = \bar{\bar{x}}$ $x + y = y + x$ $x + (y + z) = (x + y) + z$ $x (y + z) = \underline{xy} + \underline{xz}$	<b>COMMUTATIVE LAW</b> <b>ASSOCIATIVE LAW</b> <b>DISTRIBUTIVE LAW</b> <b>DE MORGAN'S THEOREM</b>	$\underline{xy} = \underline{yx}$ $x (\underline{yz}) = (\underline{xy}) z$ $x + \underline{yz} = (x + y)(x + z)$
$\overline{x + y} = \bar{x} \cdot \bar{y}$ $x + \underline{xy} = x$ $x + \bar{\underline{xy}} = x + y$	$\left\{ \begin{array}{c} \text{ABSORPTION} \\ \text{LAWS} \end{array} \right\}$	$\overline{\underline{xy}} = \bar{x} + \bar{y}$ $x (x + y) = x$ $x (\bar{x} + y) = \underline{xy}$

$$a + b + cd + d + e' + fg + 1 = 1$$

**BULLICER'S THEOREM**

Remember:  $\bar{x}$  is the same as  $x'$