Chapter 2
Boolean Algebra and Logic Gates

Outline

- Basic Definitions of Boolean Algebra
- Axiomatic Definitions
- Basic Theorems and Properties of Boolean Algebra
- Boolean Functions
- Canonical and Standard Forms
- Other Logic Operations
- Digital Logic Gates
- Integrated Circuits

History of Boolean Algebra

- In 1854, George Boole introduced a systematic treatment algebra for logic now called Boolean algebra.
- In 1904, Edward V. Huntington proposed a formal definition of Boolean Algebra.
- In 1938, Claude E. Shannon introduced two-value Boolean Algebra called switching algebra for bistable electrical switching circuits.

The postulates of a mathematical system

1. **Closure**: A set S is closed with respect to (w.r.t.) 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 any a, $b \in S$, a unique $c \in S$ such that $a * b = c$.
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, $z \in S$.
3. **Commutative law**: A binary operator $*$ on a set is said to be commutative whenever $x * y = y * x$ for all x, $y \in S$.

The postulates of a mathematical system (Cont'd)

4. **Identity element**: A set S is said to have an identity element w.r.t. a binary operator $*$ on S there exist an element $e \in S$ with the property:

$$e * x = x * e = x \text{ for every } x \in S$$

5. **Inverse**: A set S having the identity element e w.r.t. to a binary operator $*$ is said to have an inverse whenever, for every $x \in S$, there exists an element $y \in S$ such that $x * y = e$.
6. **Distributive Law**: If $*$ and • are binary operators on S, $*$ is said to be distributive over $\cdot$ whenever

$$x * (y \cdot z) = (x * y) \cdot (x * z)$$

Example

- For real number:
  - The operator "+" defines as addition.
  - The additive identity is 0.
  - Additive inverse is "subtraction."
  - The operator "•" defines multiplication.
  - The multiplicative identity is 1.
  - For $a \neq 0$, the multiplicative inverse of a is $1/a$ defines division.
  - The distributive law is "•" over "+":

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

Boolean Algebra (Huntington Postulates)

- A set of elements B and two binary operators "+" and "•" are defined by the following postulates.
  - 1. Closure with respect to "+" and "•".
  - 2. An identity element with respect to "+" and "•".

    $x + 0 = 0 + x = x$ and $x \bullet 1 = 1 \bullet x = x$
  - 3. Commutative with respect to "+" and "•" .

    $x + y = y + x$ and $x \cdot y = y \cdot x$
  - 4. Distributive over "+" and "•".

    $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$ and

    $x + (y \cdot z) = (x + y) \cdot (x + z)$
  - 5. For $x \in B$, there exists $x' \in B$ (complement of x) such that $x + x' = 1$ and $x \bullet x' = 0$.
  - 6. There exist at least two elements $x, y \in B$, such that $x \neq y$.

Differences b/w Boolean and Ordinary Algebra

1. Huntington's postulates do not include *associative law*, but it holds for Boolean algebra.
2. The distributive law of "+" over "•" ($x + (y \cdot z) = (x + y) \cdot (x + z)$) is valid only for Boolean algebra, but not for ordinary algebra.
3. Boolean algebra has no *additive* and *multiplicative* inverses. Therefore, there are no *subtraction* and *division* operations.
4. The complement element is not available in ordinary algebra.
5. The two-value algebra (special case of Boolean algebra) is defined as a set of limited two elements, 0 and 1.

Two-Valued Boolean Algebra

- B = {0, 1} is the set of two-valued Boolean Algebra
- The binary operators "+" and "•" have the following characteristics:

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

AND Logic

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

OR Logic

x	x'
0	1
1	0

NOT Logic

Two-Valued Boolean Algebra (Cont'd)

- Verifying Huntington postulates:
  - – 1. Closure: The result of each operator belongs to B.
  - – 2. Identity elements:
    - (a) 0 + 0 = 0   0 + 1 = 1 + 0 = 1   (0: identity element for +)
    - (b) 1 • 1 = 1   1 • 0 = 0 • 1 = 0   (1: identity element for •)
  - – 3. Commutative: The commutative is obvious from the symmetry of the operator table.
  - – 5. Complement:
    - (a) $x + x'$ = 1: 0 + 0' = 0 + 1 = 1; 1 + 1' = 1 + 0 = 1
    - (b) $x \cdot x'$ = 0: 0 • 0' = 0 • 1 = 0; 1 • 1' = 1 • 0 = 0
  - – 6. The two-valued Boolean algebra has two distinct elements, 1 and 0.

Two-Valued Boolean Algebra (Cont’d)

- 4. The distributive law of “•” over “+”:

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

- The distributive law of “+” over “•” ?

Basic Theorems and Properties of Boolean Algebra

- **Duality:** every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged.
- Example:
  - – Postulate 2, Identity elements:
    - (a) x + 0 = x (change 0 to 1 and “+” to “•”, we get (b))
    - (b) x • 1 = x (change 1 to 0 and “•” to “+”, we get (a))

Basic Theorems and Properties of Boolean Algebra (Cont'd)

- Six theorems and four postulates of Boolean algebra:

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

Basic Theorems

- The basic theorems can be derived from basic postulates.
- Thm. 1.(a): $x + x = x$

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

- Thm. 1(b): $x \cdot x = x$

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

Basic Theorems

- Thm. 2: $x + 1 = 1$

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

  – $x \cdot 0 = 0$ is valid by duality.

- Thm. 3: $(x')' = x$
  – From Pos. 5: $x + x' = 1$ and $x \cdot x' = 0$, defines the complement of x'. $\Rightarrow$ x is the complement of x'.
  – The complement of x' is x and is also $(x')'$. Since the complement is unique, $(x')' = x$.

Basic Theorems

- Thm. 6: $x + xy = x$

$$\begin{aligned} x+xy &= x\cdot 1 + x\cdot y && \text{Pos. 2(b)} \\ &= x\cdot(1+y) && \text{Pos. 4(a)} \\ &= x\cdot(y+1) && \text{Pos. 3(a)} \\ &= x\cdot 1 && \text{Pos. 2(a)} \\ &= x && \text{Pos. 2(b)} \end{aligned}$$

  – $x \cdot (x + y) = x$ by duality.

- By means of truth table.

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

Basic Theorems

- DeMorgan’s Theorem:
  - $(x + y)' = x' \cdot y'$
  - $(x \cdot y)' = x' + y'$

  Verified by truth table:

x	y	$x + y$	$(x + y)'$	x'	y'	$x'y'$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Operator Precedence

- The operator precedence for evaluating Boolean expressions:
  - 1. parentheses
  - 2. NOT
  - 3. AND
  - 4. OR
- Examples:
  - $xy' + z$
  - $(xy + z)'$

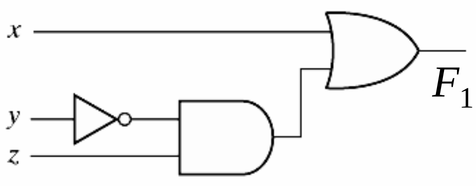
Boolean Functions

- Boolean algebra deals with binary variables and logic operations.
- A Boolean function consists of
  - binary variables (1 or 0)
  - logic operators OR and AND
  - unary operator NOT
  - parentheses
- Examples: (by Truth Table)
  - $F_1 = x + y'z$
  - $F_2 = x'y'z + x'yz + xy'$

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

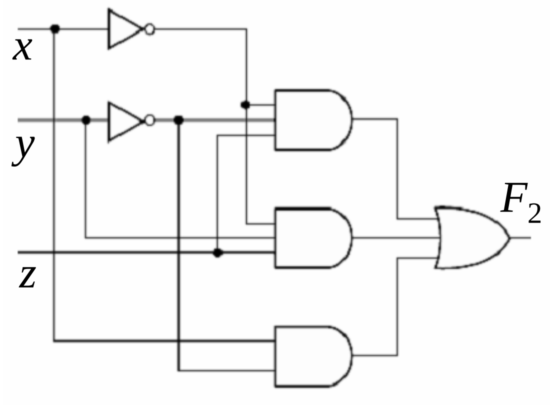
Implementations of Boolean Functions with Logic Gates

- Example: $F_1 = x + y'z$

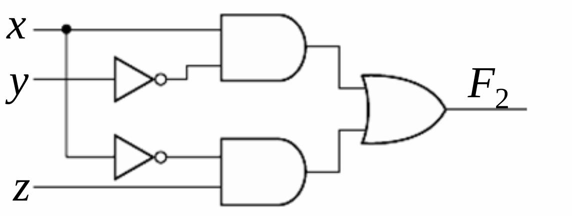


- Example:

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



=



Algebraic Manipulation

- To minimize Boolean expressions
  - literal: a primed or unprimed variable (an input to a gate)
  - term: an implementation with a gate (F_2: 3 terms, 8 literals)
  - The minimization of the number of literals and the number of terms $\Rightarrow$ a circuit with less equipments
  - It is a hard problem (no specific rules to follow)
- Examples:
  - $x(x' + y) = xx' + xy = 0 + xy = xy$
  - $x + x'y = (x + x')(x + y) = 1\ (x + y) = x + y$ (by Pos.4(b))
  - $(x + y)(x + y') = x + xy + xy' + yy' = x(1 + y + y') = x$
  - $$\begin{aligned} xy + x'z + yz &= xy + x'z + yz(x + x') \\ &= xy + x'z + yzx + yzx' \\ &= xy(1 + z) + x'z(1 + y) = xy + x'z \end{aligned}$$

Complement of a Function

- The complement of a function F is F' and is an interchange of 0's for 1's and 1's for 0's in the value of F
  - by DeMorgan's theorem
  - DeMorgan’s theorem can be extended to n variables:
  - $$\begin{aligned} (A + B + C)' &= (A + x)' && \text{let } B + C = x \\ &= A'x' && \text{by DeMorgan's} \\ &= A'(B + C)' && \text{substitute } B + C = x \\ &= A'(B'C') && \text{by DeMorgan's} \\ &= A'B'C' && \text{associative} \end{aligned}$$
- generalizations
  - $(A + B + C + \dots + F)' = A'B'C' \dots F'$
  - $(ABC \dots F)' = A' + B' + C' + \dots + F'$

Examples

- $(x'yz' + x'y'z)' = (x'yz')' (x'y'z)'$
  $= (x + y' + z)(x + y + z')$
- $[x(y'z' + yz)]' = x' + (y'z' + yz)'$
  $= x' + (y'z')'(yz)'$
  $= x' + (y + z)(y' + z')$
- A simpler procedure
  - take the dual of the function and complement each literal
  - $x'yz' + x'y'z \Rightarrow (x' + y + z')(x' + y' + z)$ (the dual)
  - $(x'yz' + x'y'z)' \Rightarrow (x + y' + z)(x + y + z')$

Canonical and Standard Forms

- Minterms and Maxterms
  - A minterm: 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: an OR term
  - It is also called a standard sum
  - n variables can be combined to form 2^n maxterms

Minterms and Maxterms

- Minterms (standard products) and Maxterms (standard sums) for three binary variables

			Minterms		Maxterms	
x	y	z	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

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

x	y	z	f_1	f_2
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
  - $f_1' = m_0 + m_2 + m_3 + m_5 + m_6$
    $= x'y'z' + x'yz' + x'yz + xy'z + xyz'$
  - $f_1 = (f_1')'$
    $= (x+y+z)(x+y'+z)\,(x+y'+z')\,(x'+y+z')(x'+y'+z)$
    $= M_0\, M_2\, M_3\, M_5\, M_6$
  - Any Boolean function can be expressed as
    - a sum of minterms
    - a product of maxterms

    } canonical form

Sum of Minterms

- Express the general logic function as a sum of minterms
  - $F = A+B'C$
    $= A\,(B+B') + B'C$
    $= AB + AB' + B'C$
    $= AB(C+C') + AB'(C+C') + (A+A')B'C$
    $=ABC + ABC' + AB'C + AB'C' + A'B'C$
  - $F = A'B'C + AB'C' + AB'C + ABC' + ABC$
    $= m_1 + m_4 + m_5 + m_6 + m_7$
  - $F(A,B,C) = \Sigma(1, 4, 5, 6, 7) = \Pi(0, 2, 3)$
  - or, built the truth table first

Truth table for $F = A+B'C$

A	B	C	F
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
  - $x + yz = (x + y)(x + z)$
    $= (x + y + zz')(x + z + yy')$
    $=(x + y + z)(x + y + z')(x + y' + z) = M_0M_1M_2$
  - $F = xy + x'z$
    $= (xy + x') (xy + z)$
    $= (x + x')(y + x')(x + z)(y + z)$
    $= (x' + y)(x + z)(y + z)$
  - $x'+ y = x' + y + zz'$
    $= (x‘ + y + z)(x‘ + y + z')$
  - $F = (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z')$
    $= M_0M_2M_4M_5$
  - $F(x, y, z) = \Pi(0, 2, 4, 5)$

Conversion between Canonical Forms

- Conversion between Canonical Forms
  - $F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$
  - $F'(A, B, C) = \Sigma (0, 2, 3)$
  - By DeMorgan's theorem
    $F(A, B, C) = \Pi(0, 2, 3)$
  - $m_j{}' = M_j$
  - Sum of minterms = product of maxterms
  - Interchange the symbols Σ and Π and list those numbers missing from the original form
  - Σ of 1's = Π of 0's

Example

- $F = xy + x'z$
- $F(x, y, z) = \Sigma(1, 3, 6, 7)$
- $F(x, y, z) = \Pi(0, 2, 4, 5)$

Truth table for $F = xy + x'z$

A	B	C	F
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

- Standard Forms
  - Canonical forms are seldom used
  - sum of products

    $F_1 = y' + xy + x'yz'$ (figure (a))

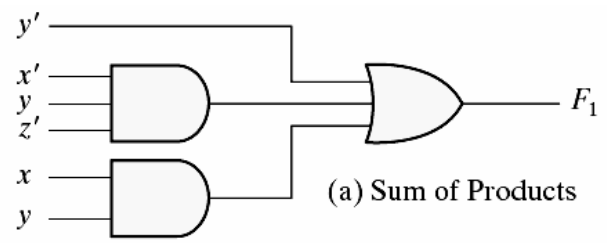
    $F_3 = AB + C(D + E)$ nonstandard form (figure (c))

    $F_3 = AB + CD + CE$ standard form (figure (d))
  - product of sums

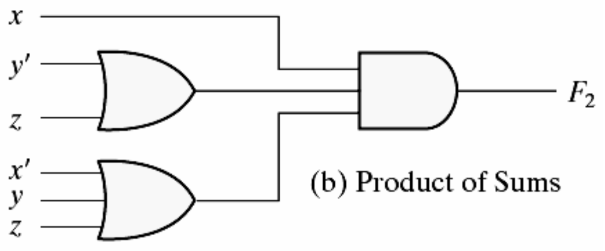
    $F_2 = x(y' + z)(x' + y + z)$ (figure (b))
  - Standard form is preferred because the gate delay is minimized (figure (c) vs. figure (d))

Standard Form Logic

- Two-level implementation

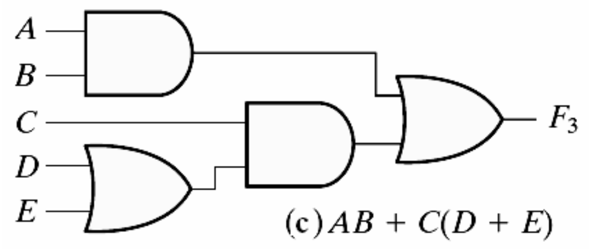


(a) Sum of Products

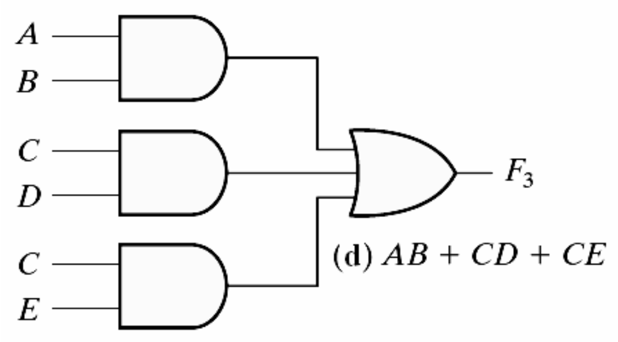


(b) Product of Sums

- Multi-level implementation



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



(d) $AB + CD + CE$

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

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
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
Operator Symbol			$\cdot$	/		/		$\oplus$	+	$\downarrow$	$\odot$	′	$\subset$	′	$\supset$	$\uparrow$	

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

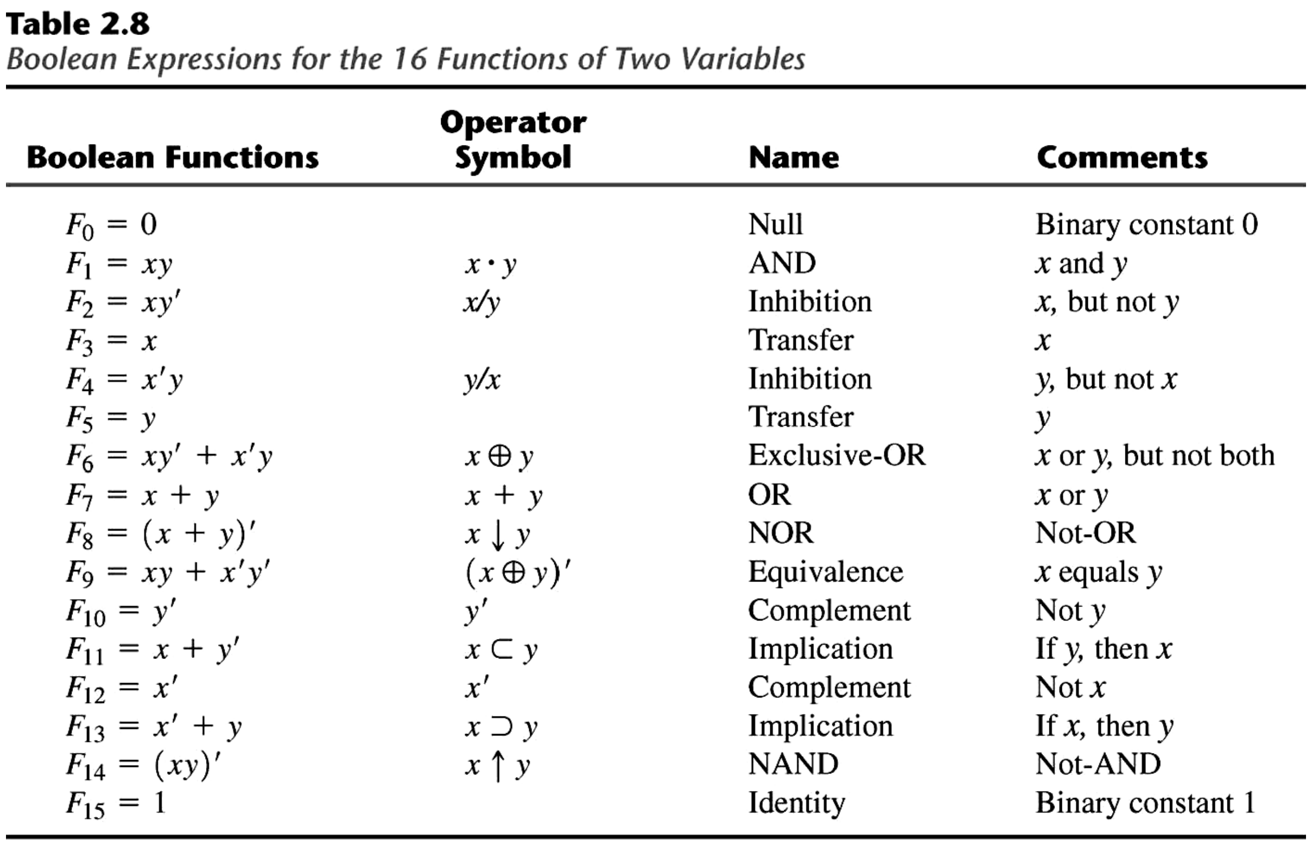


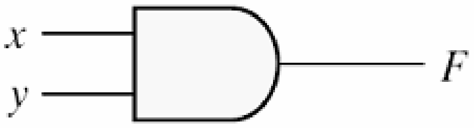
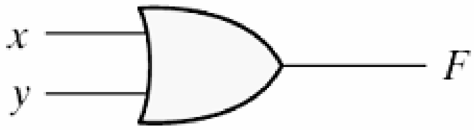
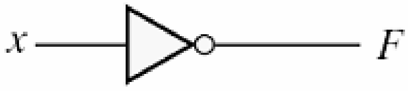
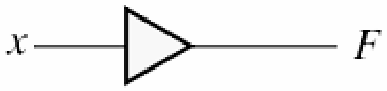
Table 2.8
Boolean Expressions for the 16 Functions of Two Variables

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

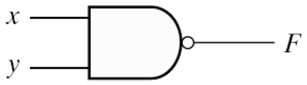
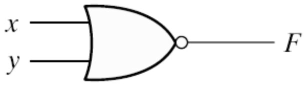
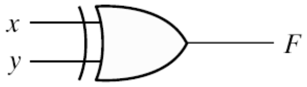
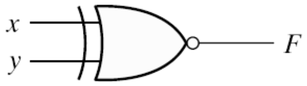
Digital Logic Gates

- Boolean expression: AND, OR and NOT operations
- Considerations of constructing other logic gates:
  - the feasibility and economy
  - the possibility of extending gate's inputs
  - the basic properties of the binary operations
  - the ability of the gate to implement Boolean functions alone
- Consider the 16 functions
  - two are equal to a constant
  - four are repeated twice
  - inhibition and implication are not commutative or associative
  - the other eight: complement, transfer, AND, OR, NAND, NOR, XOR, and equivalence are used as standard gates
  - complement: inverter
  - transfer: buffer (increasing drive strength)
  - equivalence: XNOR

Basic Digital Circuit Gates

Name	Graphic symbol	Algebraic function	Truth table
AND		$F = xy$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$F = x'$	x | F 0 | 1 1 | 0
Buffer		$F = x$	x | F 0 | 0 1 | 1

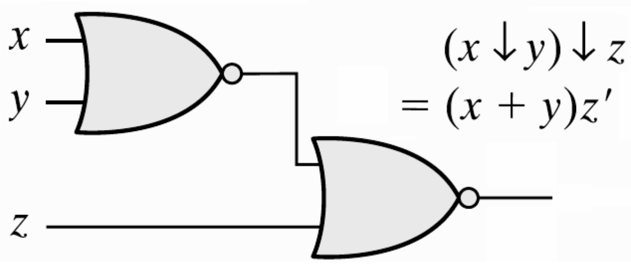
Basic Digital Circuit Gates (Cont'd)

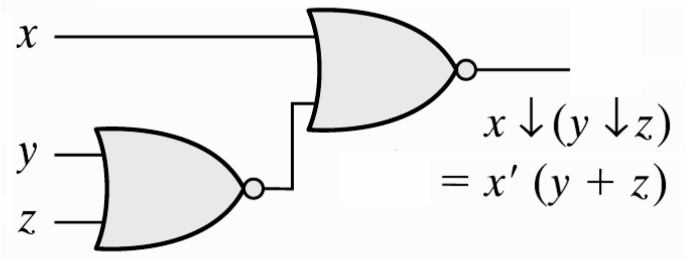
NAND		$F = (xy)'$	x y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = (x + y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Digital Circuit Gates – AND / OR, NAND / NOR

- 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
    - $(x + y) + z = x + (y + z) = x + y + z$
    - $(x \cdot y) \cdot z = x \cdot (y \cdot z) = x \cdot y \cdot z$

- NAND and NOR are commutative but not associative $\Rightarrow$ They are not extendable.



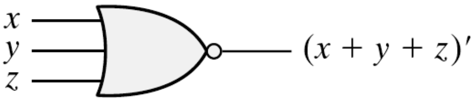


Digital Circuit Gates – NAND / NOR

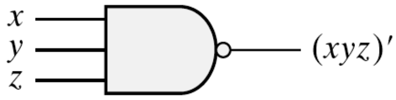
- Multiple NOR = a complement of OR gate
- Multiple NAND = a complement of AND

} Definition Modified

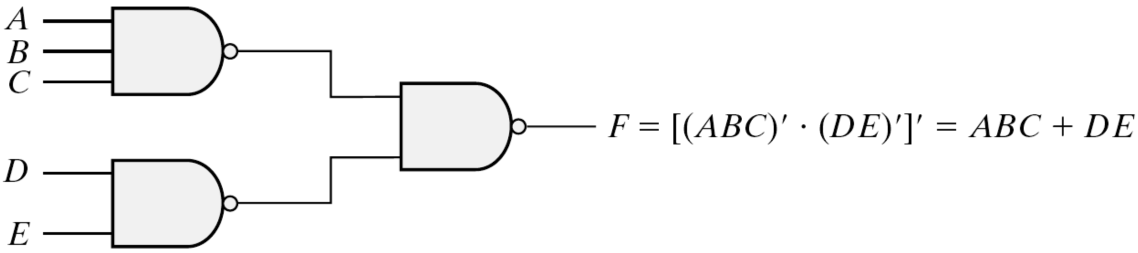
- The cascaded NAND operations = sum of products
- The cascaded NOR operations = product of sums



(a) 3-input NOR gate



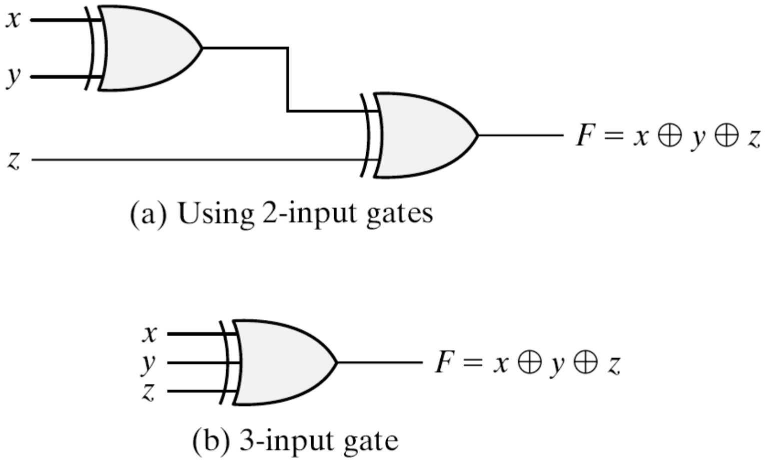
(b) 3-input NAND gate



(c) Cascaded NAND gates

Digital Circuit Gates - XOR

- 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

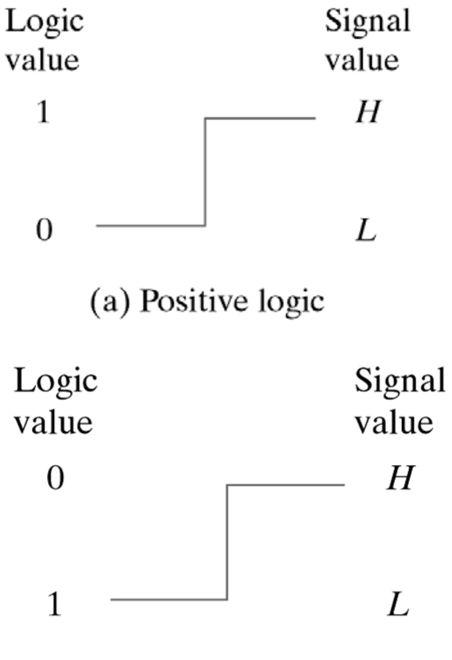
(b) 3-input gate

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

Digital Circuit Gates

- Positive and Negative Logic
  - two signal values ⇔ two logic values
  - positive logic: H=1; L=0
  - negative logic: H=0; L=1
- Consider a TTL gate
  - a positive logic NAND gate
  - a negative logic OR gate
  - the positive logic is used in this book

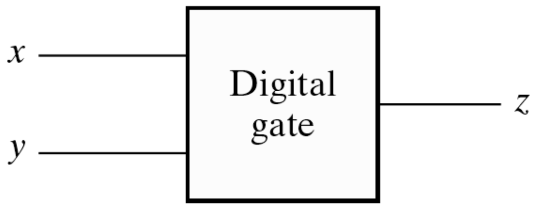


(a) Positive logic

(b) Negative logic

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

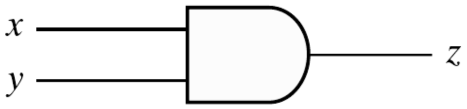
(a) Truth table with H and L



(b) Gate block diagram

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

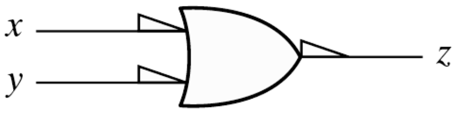
(c) Truth table for positive logic



(d) Positive logic AND gate

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

(e) Truth table for negative logic



(f) Negative logic OR gate

Integrated Circuits

- An integrated circuit (IC) is a silicon semiconductor crystal, called a chip, containing electronic digital gates.
- Examples:
  - – SSI: < 10 gates
  - – MSI: 10 ~ 100 gates
  - – LSI: 100 ~ xk gates
  - – VLSI: > xk gates
    - small size (compact size)
    - low cost
    - low power consumption
    - high reliability
    - high speed

Integrated Circuits

- 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
- 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

Integrated Circuits

- 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