Boolean Algebra and Logic Gates

Chun-Jen Tsai
National Chiao Tung University
10/01/2012

Algebraic Systems (1/2)

- An algebraic structure has following properties:
  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.
  2. **Associative law**. A binary operator $*$ on a set S is said to be associative whenever

     $$(x * y) * z = x * (y * z) \text{ for all } x, y, z \in S$$

  3. **Commutative law**. A binary operator $*$ on a set S is said to be commutative whenever

     $$x * y = y * x \text{ for all } x, y \in S$$

Algebraic Systems (2/2)

4. **Identity**. 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

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

5. **Inverse**. A set S having the identity element e with respect 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 two binary operators on a set S, $*$ is said to be distributive over • whenever

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

Example of Identity and Inverse

- The element 0 is an identity element with respect to the binary operator + on the set of integers $I = \{ \ldots, -3, -2, -1, 0, 1, 2, 3, \ldots \}$, since

$$x + 0 = 0 + x = x \text{ for any } x \in I$$

  The set of natural numbers, N, has no identity, since 0 is excluded from the set.
- In the set of integers I and the operator +, with $e = 0$, the inverse of an element a is $(-a)$, since $a + (-a) = 0$.
- Question: Is 1 an identity?

Fields

- A field is an example of an algebraic structure. The field of real numbers is the basis for arithmetic and algebra:
  - The binary operator $+$ defines addition.
  - The additive identity is 0.
  - The additive inverse defines subtraction.
  - The binary operator $\cdot$ defines multiplication.
  - The multiplicative identity is 1.
  - For $a \neq 0$, the multiplicative inverse of $a = 1/a$ defines division (i.e., $a \cdot 1/a = 1$).
  - The only distributive law applicable is that of $\cdot$ over $+$:

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

Boolean Algebra (1/2)

- In 1854, George Boole developed an algebraic system now called Boolean algebra.
- Boolean algebra defines two binary operators + and $\cdot$ and assumes the Huntington postulates:

1. (a) The structure is closed with respect to the operator +
   (b) The structure is closed with respect to the operator $\cdot$
2. (a) The element 0 is an identity element with respect to +; that is, $x + 0 = 0 + x = x$.
   (b) The element 1 is an identity element with respect to $\cdot$; that is, $x \cdot 1 = 1 \cdot x = x$.

Boolean Algebra (2/2)

3. (a) The structure is commutative with respect to +;
   that is, $x + y = y + x$.
   (b) The structure is commutative with respect to • ;
   that is, $x \cdot y = y \cdot x$.
4. (a) The operator $\cdot$ is distributive over +;
   that is, $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.
   (b) The operator + is distributive over $\cdot$;
   that is, $x + (y \cdot z) = (x + y) \cdot (x + z)$.
5. For every element $x \in B$, there exists an element $x' \in B$ (called the complement of x) such that
   (a) $x + x' = 1$ and (b) $x \cdot x' = 0$.
6. There exist at least two elements $x, y \in B$ such that $x \neq y$.

Boolean vs. Ordinary Algebra (1/2)

- Huntington postulates proposed in 1904 do not include the associative law for + and ·, which can be derived from the other postulates.
  - In 1933, Huntington realized that for Boolean algebra, we only need + and ′ operators and three postulates (+ being associative and commutative, and $(x' + y')' + (x'+y)' = x$).
- The distributive law of + over · is valid for Boolean algebra, but not for ordinary algebra.
- Boolean algebra does not have additive or multiplicative inverses; therefore, there are no subtraction or division operations.

Boolean vs. Ordinary Algebra (2/2)

- Postulate 5 defines an operator called the complement that is not available in ordinary algebra.
- Ordinary algebra deals with the real numbers, which constitute an infinite set of elements. Boolean algebra deals with a two-valued set B. For example, B can be defined as a set with only two elements, 0 and 1.

Two-valued Boolean Algebra

- $B = \{\ 0, 1\ \}$
- Rules of operations:

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

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

x	x'
0	1
1	0

- It can be shown easily that the Huntington postulates are valid for the set B and operators + and ·
- The equivalent operations are the same as AND (·), OR (+), and NOT (′)

Properties of Boolean Algebra

- Duality Principle: In an expression, if the binary operators AND $\leftrightarrow$ OR and the identities $1 \leftrightarrow 0$ are interchanged, the expression still holds
- Basic theorems:

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)	$x(x + y) = x$

Proof of Basic Theorems (1/4)

- Theorem 1(a): $x + x = x$
  - $$\begin{aligned} x + x &= (x + x) \cdot 1 \\ &= (x + x)\,(x + x') \\ &= x + xx' \\ &= x + 0 \\ &= x \end{aligned}$$
- Theorem 1(b): $x \cdot x = x$
  - $$\begin{aligned} x \cdot x &= x \cdot x + 0 \\ &= xx + xx' \\ &= x\,(x + x') \\ &= x \cdot 1 \\ &= x \end{aligned}$$

Proof of Basic Theorems (2/4)

- Theorem 2 (a)
  - $x + 1 = 1 \cdot (x + 1)$
    $= (x + x')(x + 1)$
    $= x + x' \cdot 1$
    $= x + x'$
    $= 1$
- Theorem 2 (b)
  - $x \cdot 0 = 0$ by duality.
- Theorem 3: $(x')' = x$
  - Postulate 5 defines the complement of x.
    Since $x + x' = 1$ and $x \cdot x' = 0$, the complement of x' is x.

Proof of Basic Theorems (3/4)

- Theorem 6(a)
  - $x + xy = x \cdot 1 + xy$
    $= x\,(1 + y)$
    $= x \cdot 1$
    $= x$
  - Proof by the truth table:

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

- Theorem 6(b)
  - $x\,(x + y) = x$ by duality.

Proof of Basic Theorems (4/4)

- DeMorgan's Theorems
  - $(x+y)' = x' y'$
  - $(x y)' = x' + y'$

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

- Operator Precedence
  - Parentheses
  - NOT
  - AND
  - OR
- Examples
  - $x\,y' + z$
  - $(x\,y + z)'$

Boolean Functions

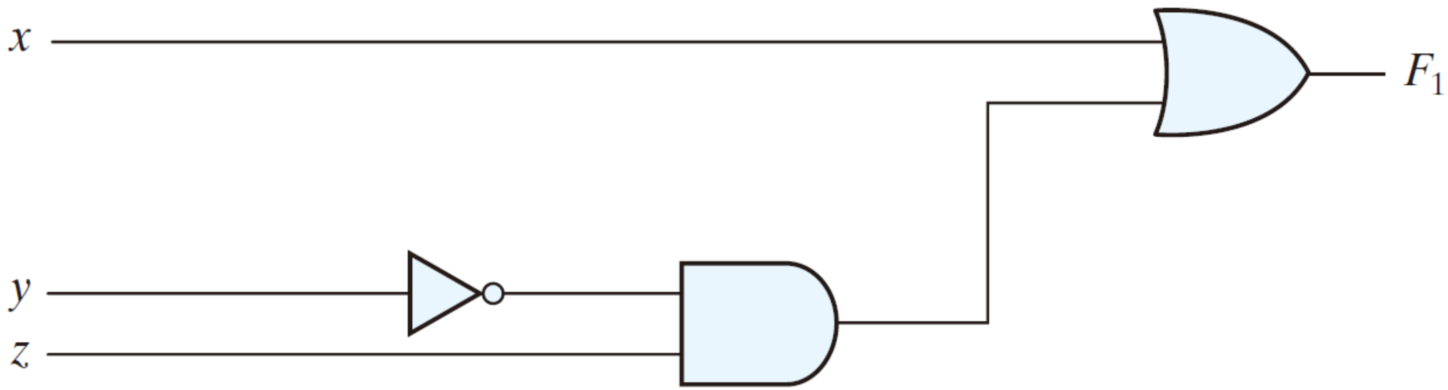
- A Boolean function is composed of:
  - binary variables
  - binary operators OR and AND
  - unary operator NOT
  - parentheses
- Examples
  - $F_1 = x + y'z$
  - $F_2 = x'y'z + x'yz + xy'$

Truth Tables for F_1 and F_2

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

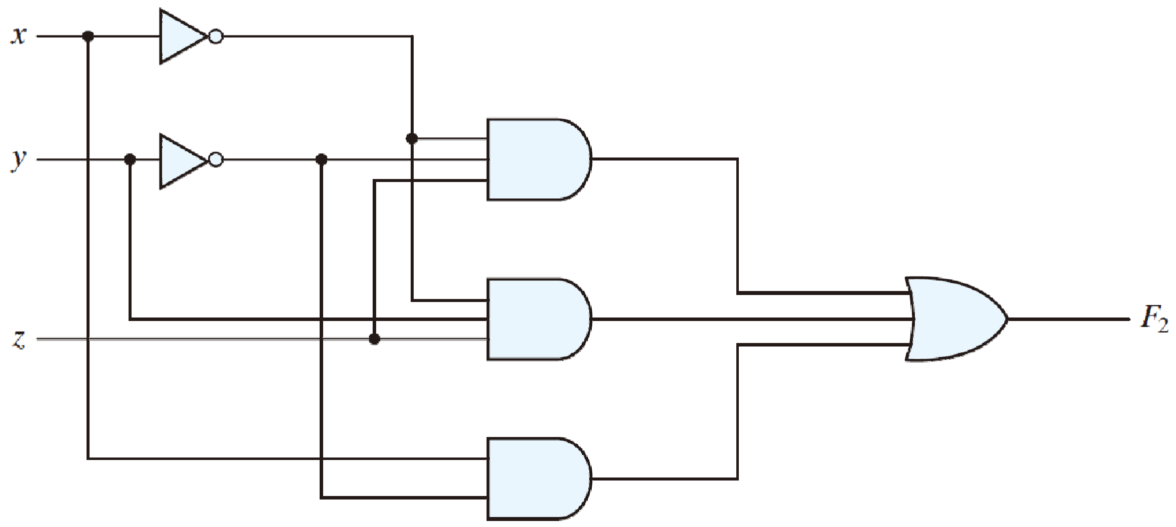
Logic Implementation (1/2)

- We can use logic gates to implement $F_1 = x + y'z$

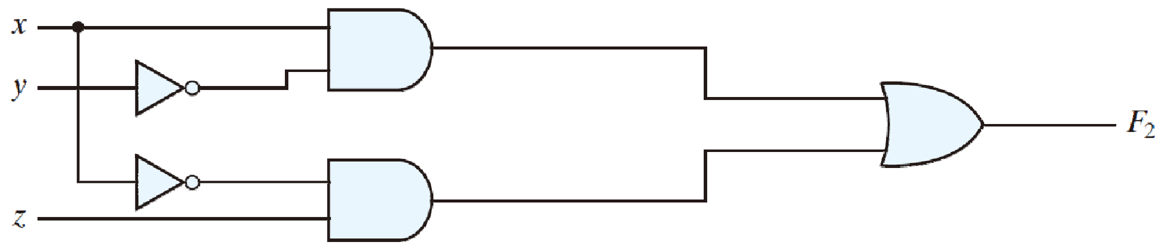


Logic Implementation (2/2)

- Gate-level Implementations may not be unique



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



(b) $F_2 = xy' + x'z$

Algebraic Manipulation

- In a Boolean expression, we define
  - 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 can help us implement a circuit with less components

Examples: Algebraic Minimization

- Simplify the following Boolean functions:

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.
$$\begin{aligned} xy + x'z + yz &= xy + x'z + yz(x + x') \\ &= xy + x'z + xyz + x'yz \\ &= xy(1 + z) + x'z(1 + y) \\ &= xy + x'z. \end{aligned}$$

5. $(x + y)(x' + z)(y + z) = (x + y)(x' + z)$, by duality from function 4.

Complement of a Function

- The complement of a function F is F'. It can be computed by DeMorgan's theorem.

- Example: 3-variable DeMorgan's theorem
  $(A + B + C)' = A'B'C'$

- General DeMorgan's law:

$$(A + B + C + \ldots + F)' = A'B'C' \ldots F'$$
$$(ABC \ldots F)' = A' + B' + C' + \ldots + F'$$

Examples: Finding Complement

- Find the complement of the functions $F_1 = x'yz' + x'y'z$ and $F_2 = x(y'z' + yz)$:

$$
\begin{aligned}
F_1' &= (x'yz' + x'y'z)' = (x'yz')'(x'y'z)' = (x + y' + z)(x + y + z') \\
F_2' &= [x(y'z' + yz)]' = x' + (y'z' + yz)' = x' + (y'z')'(yz)' \\
&= x' + (y + z)(y' + z') \\
&= x' + yz' + y'z
\end{aligned}
$$

Complement by Duality

- Complement of a Boolean function can be obtained by taking its duals and complementing each literal
- Example: finding the complement of
  $F_1 = x' y z' + x' y' z$

  Dual of F_1 is $(x' + y + z')(x' + y' + z)$.
  Complementing each literal, we have

$$(x + y' + z)(x + y + z').$$

Canonical and Standard Forms

- Minterm
  - An AND term consists of all literals in their normal or complement forms
  - For example, with two binary variables x and y, there are four minterms: xy, xy', $x'y$, $x'y'$
  - It is also called a standard product
  - n variables can be combined to form 2^n minterms
- Maxterm:
  - An OR term consists of all literals in their normal or complement forms
  - It is also call a standard sum
  - n variables can be combined to form 2^n maxterms

Three-variable Min- and Max-terms

- In the table, each maxterm is the complement of its corresponding minterm, and vice versa

Minterms and Maxterms 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

Boolean Function Representation

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

Functions of Three Variables

x	y	z	Function f_1	Function 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

- How do we calculate the complement of f_1 and f_2?

Canonical Form

- Boolean functions expressed as a sum of minterms or product of maxterms are said to be in canonical form
- Express the Boolean function $F=A+B'C$ as a sum of minterms:
  - $F = A\,(B+B') + B'C$
    $= AB + AB' + B'C$
    $= AB(C+C')+AB'(C+C')+(A+A')B'C$
    $= ABC+ABC'+AB'C+AB'C'+A'B'C$
  - $F = A'B'C+AB'C'+AB'C+ABC'+ABC$
    $= m_1 + m_4 + m_5 + m_6 + m_7$
  - $F(A,B,C) = \Sigma(1, 4, 5, 6, 7)$

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

Example: Product of Maxterms

- Express the Boolean function $F = xy + x'z$ as a product of maxterms:
  - $F = xy + x'z$
    $= (xy + x')(xy + z)$
    $= (x + x')(y + x')(x + z)(y + z)$
    $= (x' + y)(x + z)(y + z)$

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

    $= (x'+y+z)(x'+y+z')(x+y+z)(x+y'+z)$
    $= M_4M_5M_2M_0$
    $= \Pi(0, 2, 4, 5)$

Conversion btw. Canonical Forms

- Example:
  - $F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$
  - $F'(A, B, C) = \Sigma(0, 2, 3) = m_0 + m_2 + m_3$
  - Since $m_j' = M_j$, by DeMorgan's theorem, we have

$$\begin{aligned} F &= (m_0 + m_2 + m_3)' = m_0' \cdot m_2' \cdot m_3' \\ &= M_0 M_2 M_3 = \Pi(0, 2, 3) \end{aligned}$$

- Conversion rule: interchange the symbols Σ and Π and list those numbers missing from the original form
  - Σ of 1's, Π of 0's

Example: Conversion

- $F = xy + x'z$
  - Sum of Minterms: $F(x, y, z) = \Sigma(1, 3, 6, 7)$
  - Product of Maxterms: $F(x, y, z) = \Pi(0, 2, 4, 6)$

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

x	y	z	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

Minterms

Maxterms

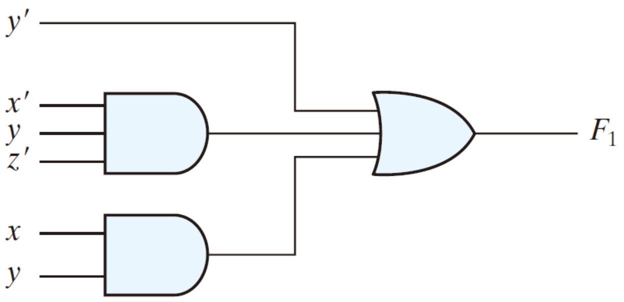
Standard Forms

- Canonical forms are verbose. In practice, we prefer the standard forms of Boolean functions
  - In standard form, we don't require all variables in each term
  - Two forms: sum-of-products and product-of-sums

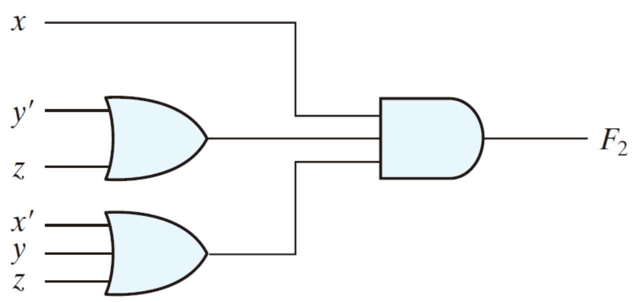
- Examples:

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

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



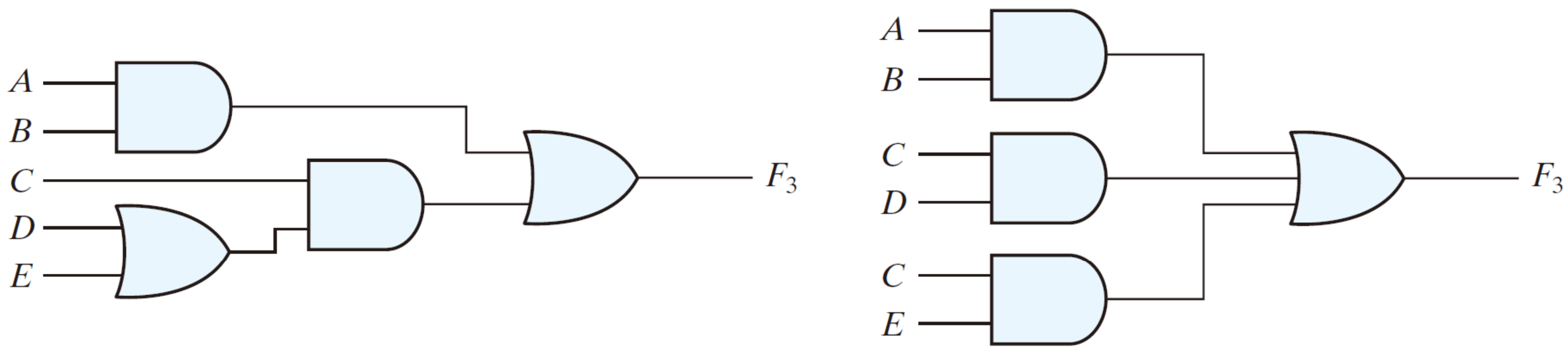
(a) Sum of Products



(b) Product of Sums

Equivalent Implementations

- A Boolean function can be implemented in different ways. For example, $F_3 = AB + C(D + E)$:



(a) $AB + C(D + E)$ (b) $AB + CD + CE$

Three- and two-level implementation

Logic Operations (1/2)

- For n-variable Boolean functions
  - 2^{2^n} functions for n binary variables
  - 2^n rows in the truth table of n binary variables
- Example: 16 functions of two binary variables

Truth Tables for the 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

Logic Operations (2/2)

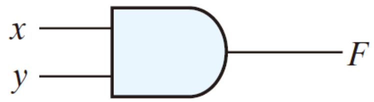
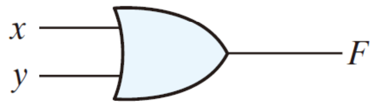
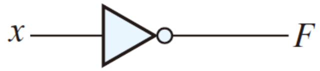
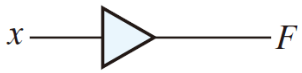
- A list of all two-variable functions

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

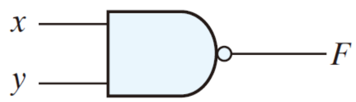
- Boolean expression: AND, OR and NOT operations
- Factors for constructing gate of a logic operation
  - The feasibility and economy of producing the gate.
  - The possibility of extending gate's inputs above two.
  - The properties (e.g. commutative) of the operations.
  - The ability of the gate to implement Boolean functions.
- Consider the 16 two-variable functions
  - Two are equal to a constant; four are repeated twice.
  - Inhibition and implication are not commutative or associative.
  - The other eight: Complement (inverter), Transfer (buffer), AND, OR, NAND, NOR, XOR, and XNOR (equivalence) are standard gates.

Common Logic Gates (1/2)

<table>
<tr><th>Name</th><th>Graphic symbol</th><th>Algebraic function</th><th>Truth table</th></tr>
<tr><td>AND</td><td></td><td>$F = x \cdot y$</td><td><table><tr><th>x</th><th>y</th><th>F</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></td></tr>
<tr><td>OR</td><td></td><td>$F = x + y$</td><td><table><tr><th>x</th><th>y</th><th>F</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></td></tr>
<tr><td>Inverter</td><td></td><td>$F = x'$</td><td><table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table></td></tr>
<tr><td>Buffer
(used for signal relay)</td><td></td><td>$F = x$</td><td><table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table></td></tr>
</table>

Common Logic Gates (2/2)

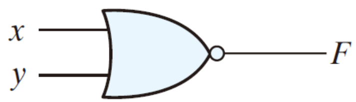
NAND



$F = (xy)'$

x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

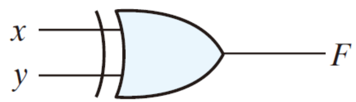
NOR



$F = (x + y)'$

x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

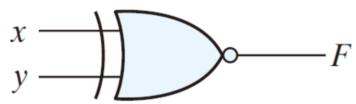
Exclusive-OR (XOR)



$$F = xy' + x'y = x \oplus y$$

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-NOR or equivalence

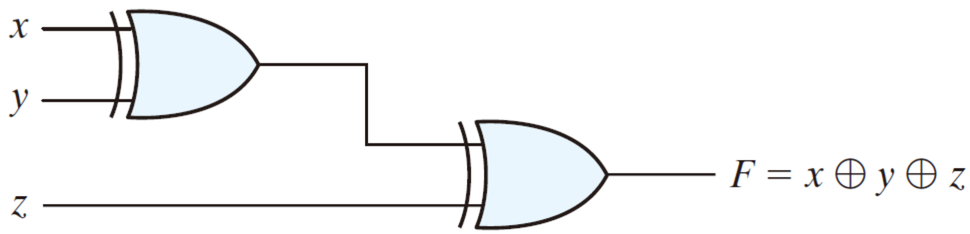


$$F = xy + x'y' = (x \oplus y)'$$

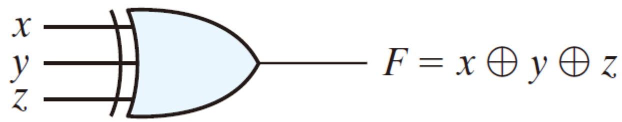
x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

Extension to Multiple Inputs

- A gate can be extended to multiple inputs easily if the binary operation is commutative and associative
  - AND and OR are extensible
  - Exclusive-OR (XOR) are extensible, but cascading implementation is often used



(a) Using 2-input gates



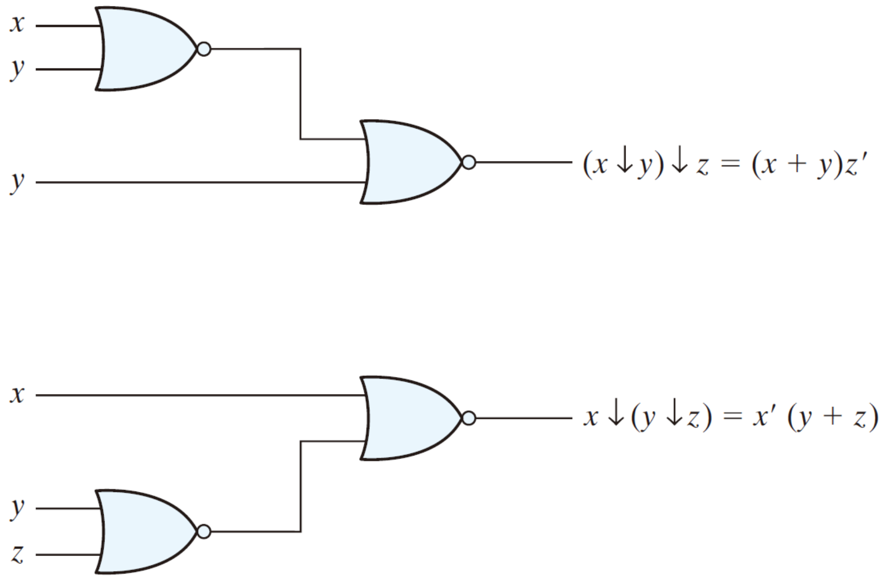
(b) 3-input gate

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

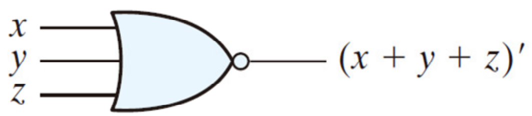
(c) Truth table

Multiple-Input NAND, NOR

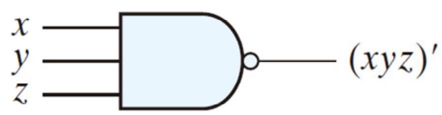
- NAND and NOR are not associative, what can we do?



- Define multiple-input NAND and OR gates as follows:



(a) 3-input NOR gate



(b) 3-input NAND gate

Signal Assignment & Polarity

❑ Positive and Negative Logic

- two signal values $\rightarrow$ two logic values
- positive logic: $H = 1$; $L = 0$
- negative logic: $H = 0$; $L = 1$
- the positive logic is used in this book

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

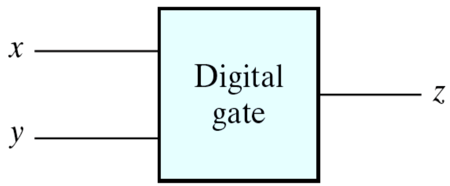
(a) Truth table with H and L

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

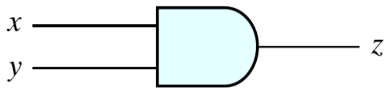
(c) Truth table for positive logic

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

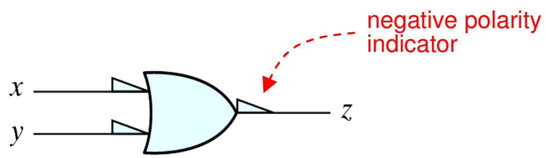
(e) Truth table for negative logic



(b) Gate block diagram



(d) Positive logic AND gate



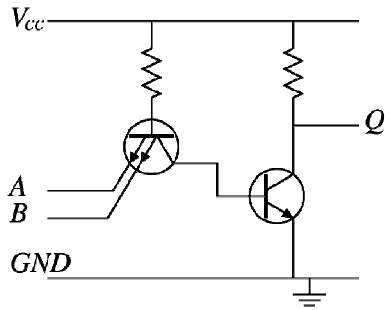
(f) Negative logic OR gate

Integrated Circuits (IC)

- An digital IC can be categorized according to its complexity as follows:
  - SSI: < 10 gates, less than 10 pins
  - MSI: 10 ~ 1000 gates
  - LSI: thousands of gates
  - VLSI: millions of gates
- A digital system can be built using multiple SSI, MSI, LSI ICs; or by a single VLSI IC.
  - The single-chip approach has many advantages: small size, low power consumption, low manufacturing cost, high reliability, and high performance.

Digital Circuit Technologies

- Different technologies for gate logic implementations
  - TTL: transistor-transistor logic (used for 50 years now)



(the implementation of a NAND gate)

  - ECL: emitter-coupled logic (high speed, high power consumption)
  - MOS: metal-oxide semiconductor (NMOS, high density)
  - CMOS: complementary MOS (low power)

Gate Implementation Properties

- Each digital logic family has different characteristics
  - 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

Computer-Aided Design

- Today, a complex digital circuit has millions of transistors, we need computers to assist the design process
- Digital circuit design flow involves several steps:
  - Design entry
    - Schematic capture
    - HDL – Hardware Description Languages: Verilog, VHDL
  - Simulation
    - *Waveform simulator* shows you the *cycle-accurate* output signals given an input signals (test patterns generated by a *testbench*)
  - Physical realization (logic synthesis, mapping, and layout)
    - ASIC, FPGA, PLD