CSE 4205
Digital Logic Design

Boolean Algebra & Logic Gates

Course Teacher: Md. Hamjajul Ashmafee

Lecturer, CSE, IUT

Email: ashmafee@iut-dhaka.edu

Basic Definitions - I

- **Boolean algebra** – similar with others deductive mathematical systems
  - Defined with a set of elements (S), a set of operators, and a number of unproved axioms or postulates
- **A set of elements, S** – any collections of objects having a common property – e.g. {a,b,c,…x,y,z}, {0,1,2,…}
- **A set of operators** – a set of rules that defined on the set **S** – e.g. {+,*}
- **Postulates** – form of the basic assumptions – other rules, theorems, properties of the system are deduced from them

Common Postulates of a Mathematical System

1. **Closure:** A set that is closed under an operation or collection of operations is said to satisfy a closure property.
   - A *set* has closure under an operation if performance of that operation on members of the set always produce a member of the same set – the set is closed under the operation.
   - ***Example:*** The set of natural number, N ={1,2,3,…} is closed with respect to the binary operator plus (+) by the rules of arithmetic addition but not closed with respect to binary operator minus (-).
   - $a,b \in \mathbf{N}$ and we obtain a unique $c \in \mathbf{N}$ by the operation $a+b=c$ but $2-3=-1$ where $2,3 \in \mathbf{N}$ but $-1 \notin \mathbf{N}$
2. **Associative Law:** A binary operator **#** on a set *S* is said to be associative whenever

   $(x \,\#\, y) \,\#\, z = x \,\#\, (y \,\#\, z)$, *for all* $x,y,z \in S$

Common Postulates…

3. **Commutative Law:** A binary operator **#** on a set S is said to be commutative whenever:

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

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:

   $$x \# e = e \# x = x, \text{ for all } x,e \in S$$

   - ***Example:*** The element 0 is an identity element with respect to the operation + on the set of integers **I** = {…-3,-2,-1,0,1,2,3…} but not on the set **N.**
   - $x+0 = 0+x = x$, for any $x \in$ **I**

Common Postulates…

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

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

   - ***Example:*** In the set ***I***, $e=0$ and the inverse of every 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:

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

Basic Definitions - II

- **Field**: A set of elements, together with two binary operators, each having properties 1 to 5 and both operators combined to give property 6.
  - Example of an algebraic structure

    Example: Set of real number, **R**, with two binary operators + and * form the **field of real numbers**. (*verification?*)

Boolean Algebra

- In 1854, **George Boole** introduced systematic treatment of logic and developed Boolean algebra
- In 1938, **C. E. Shannon** introduced a two level Boolean algebra called switching circuit/algebra
- For formal definition of Boolean algebra, we follow the postulates by **E. V. Huntington** (1904).
- *Boolean algebra* is a **field** with set of elements ***B*,** together with two binary operators + and **.** that follows **Huntington** postulates.

Huntington Postulates for Boolean Algebra

1. a. **Closure** with respect to the operator +

   b. **Closure** with respect to the operator **.**

2. a. **An identity element** with respect to +, is designated by 0 :

   $x+0 = 0+x = x$

   b. **An identity element** with respect to **.**, is designated by 1 :

   $x.1 = 1.x = x$

3. a. **Commutative** with respect to + : $x+y = y+x$

   b. **Commutative** with respect to **.** : $x.y = y.x$

4. a. . is **distributed** over + : $x.(y+z) = (x.y) + (x.z)$

   b. + is **distributed** over . : $x+(y.z) = (x+y) . (x+z)$

Huntington Postulates…

5. For every element $x \in B$ there exists an element $x' \in B$ (**complement of x**) such that
   a. $x + x' = 1$
   b. $x.x' = 0$
6. There exists at least **two elements** $x,y \in B$ such that $x \neq y$

Boolean Algebra VS Ordinary Algebra

- Huntington postulates do not mention **associative law** but it holds here. (derivable)
- The **distributive law of + over .** is only valid for Boolean algebra. *Example: x+(y.z) = (x+y).(x+z)*
- Boolean algebra does not have any **additive or multiplicative inverse**. So there is **no subtraction or division** operations in Boolean algebra
- **Complement** is only available in Boolean algebra
- Boolean algebra deals with set B having **two elements 0 and 1** but real numbers deals with infinite set of elements

Two-valued Boolean Algebra

- Verification to satisfy all Huntington postulates

Basic Definitions - III

- **Duality:** Every algebraic expression deducible from the postulates of the Boolean algebra remains valid if the operators and identity elements are interchanged.
  - For this reason, Huntington postulates are listed in pairs. For dual of any algebraic expression, we simply interchange OR and AND operators and replace 1s by 0s and 0s by 1s.
  - **Examlpe:** Postulate – 5 from Huntington postulates:
    a. $x + x' = 1$
    b. $x.x' = 0$

Basic Theorems

Postulate 2 (identity)	a. x+0 = x	b. x.1 = x
Postulate 3 (commutative)	a. x+y = y+x	b. x.y = y.x
Postulate 4 (distributive)	a. x(y+z) = xy+xz	b. x+yz = (x+y)(x+z)
Postulate 5 (complement)	a. x+x' = 1	b. x.x' = 0
Theorem 1	a. x+x = x	b. x.x = x
Theorem 2	a. x+1 = 1	b. x.0 = 0
Theorem 3 (involution)	(x')' = x	
Theorem 4 (associative)	a. x+(y+z) = (x+y)+z	b. x(yz) = (xy)z
Theorem 5 (De Morgan)	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

THEOREM 1(a): $x + x = x.$

Statement	Justification
$x + x = (x + x) \cdot 1$	postulate 2(b)
$= (x + x)(x + x')$	5(a)
$= x + xx'$	4(b)
$= x + 0$	5(b)
$= x$	2(a)

Proof of Basic Theorems…

THEOREM 1(b): $x \cdot x = x$.

Statement	Justification
$x \cdot x = xx + 0$	postulate 2(a)
$= xx + xx'$	5(b)
$= x(x + x')$	4(a)
$= x \cdot 1$	5(a)
$= x$	2(b)

Proof of Basic Theorems…

THEOREM 2(a): $x + 1 = 1.$

Statement	Justification
$x + 1 = 1 \cdot (x + 1)$	postulate 2(b)
$= (x + x')(x + 1)$	5(a)
$= x + x' \cdot 1$	4(b)
$= x + x'$	2(b)
$= 1$	5(a)

THEOREM 2(b): $x \cdot 0 = 0$ by duality.

Proof of Basic Theorems...

THEOREM 6(a): $x + xy = x$.

Statement	Justification
$x + xy = x \cdot 1 + xy$	postulate 2(b)
$= x(1 + y)$	4(a)
$= x(y + 1)$	3(a)
$= x \cdot 1$	2(a)
$= x$	2(b)

THEOREM 6(b): $x(x + y) = x$ by duality.

Proof of Basic Theorems…

- **Theorem 3:** $(x')' = x$.
  - From postulate 5 (complement), we have $x + x' = 1$ and $x.x' = 0$ which define the complement of x. So the complement of x' is x and $(x')'$. As a result we will get, $x = (x')'$
- Theorems of Boolean algebra can be proven using truth table.
- The algebraic proofs of the **associative law** and **De Morgan's** theorem are long but also can be proved with truth tables.

Proof using Truth Table

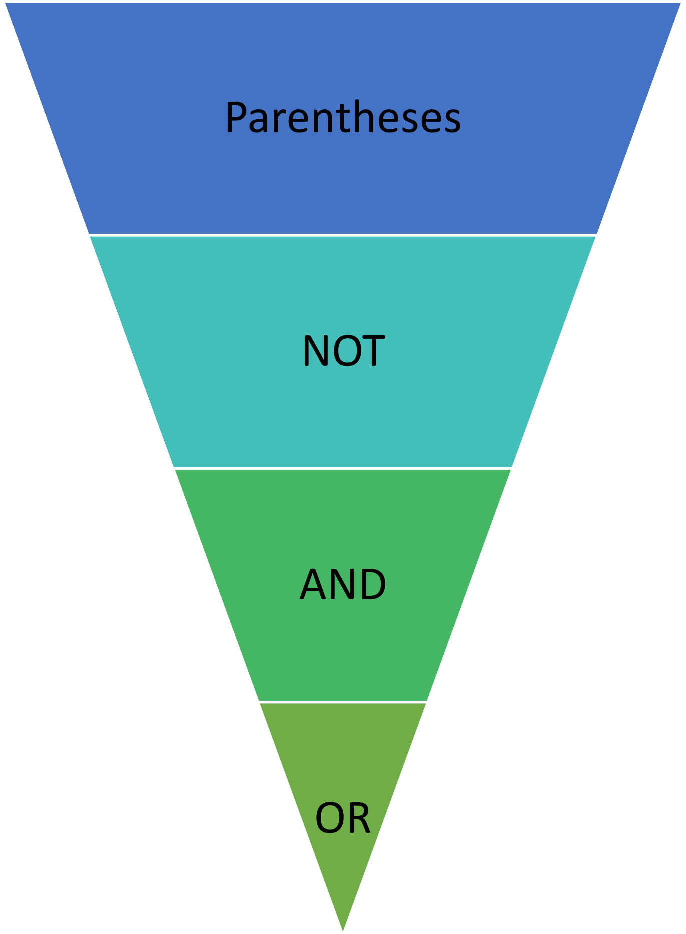
- Absorption Rule (x + xy = x):

X	Y	XY	X+XY
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

- De Morgan's Law [(x+y)’ = x’+y’]:

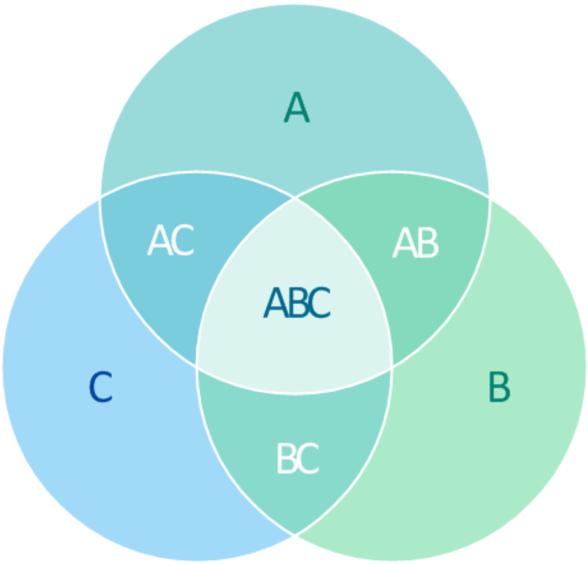
X	Y	X+Y	(X+Y)’	X’	Y’	X’Y’
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Operator Precedence



Venn Diagram

Similarity with Boolean expressions.



Boolean Function

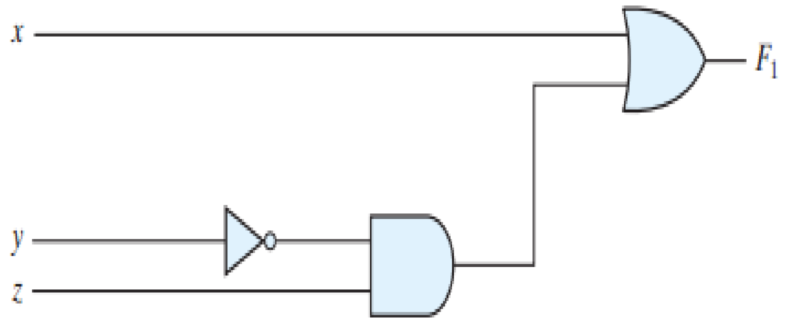
- Boolean function is an algebra with Boolean variables and logic operations – expressed as an expression with binary variables, Boolean constants, logic operation symbols
- **Expression** is a mathematical phrase whereas **function** is a relationship between a set of input and corresponding output
- For a given values for binary variables the **function** produces 0 or 1
- Example: $F_1 = x + y'z$
- Also can represented with **Truth Table –** number of the rows of the table is 2^n; $\boldsymbol{n}$ = number of variables in the function – count from 0 ~ 2^n-1

Boolean Function…

x	y	z	F_1
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

Boolean Function…

- Also can be expressed with **circuit diagram** composed of logic gates – **Schematic diagram**



- **Variables** of the function are taken as **input**, and binary variable ***F*** as **output** of the circuit
- Instead of listing all combinations of input and output like truth table, it rather indicates how to generate the output from given input values.

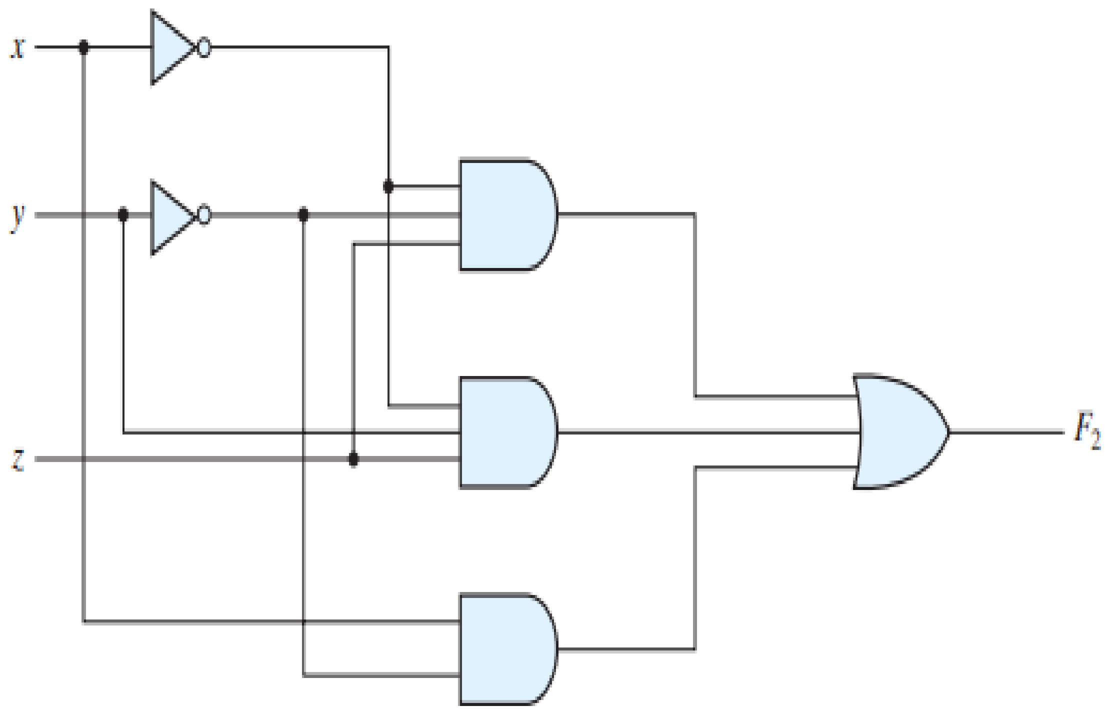
Boolean Function…

- There are only one to express a Boolean function through the truth table – but function in algebraic form or schematic form can be expressed in variety of ways (equivalent logic)
- **Truth table** is the only way to verify the expressions' equivalence
- The **motivation** of the Boolean algebra is the manipulating a Boolean expression using rules to reduce it into a simpler expression – thus reduce the number of gates, inputs, interconnections in the circuit – ultimately **reduce** the complexity, cost, and **increase** the reliability, efficiency
- **Example:**

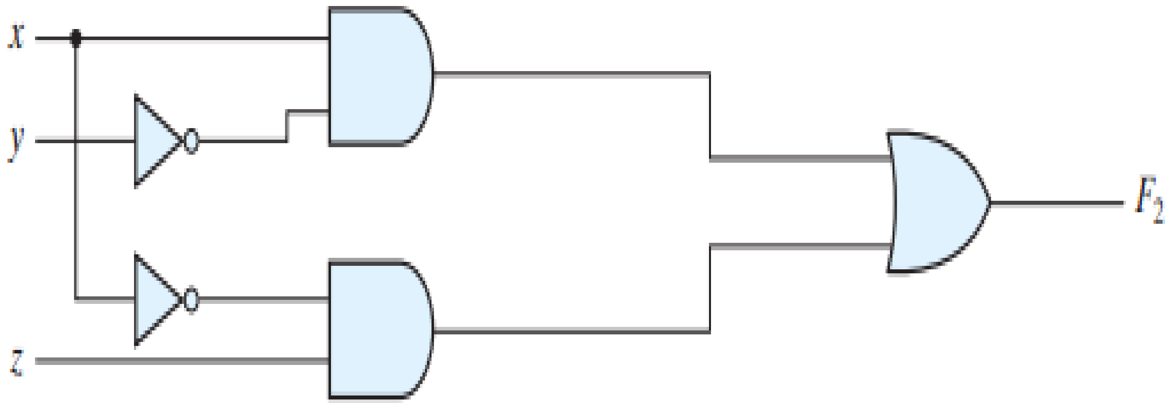
$$F_2 = x'y'z + x'yz + xy'$$

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

Boolean Function…



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



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

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

Algebraic Manipulation

- In Boolean expression –
  - **Term** designates a gate with inputs
  - **Literals/ variables** designate each input in primed or unprimed form
- Example:

$$F_2 = x'y'z + x'yz + xy'$$ **(3 terms and 8 literals)**

$$F_2 = x'y'z + x'yz + xy' = x'z(y' + y) + xy' = x'z + xy'$$ **(2 terms and 4 literals)**

- <u>To reduce the expression</u>, we have to **reduce** the number of terms or number of literals or both - to make simpler circuit
- **Methods** – cut-and-try procedure (manual – error-prone), map, computer minimization programs

Simplification of Boolean Function

- Cut and Try procedure:

$$x(x' + y) = xx' + xy = 0 + xy = xy.$$

$$x + x'y = (x + x')(x + y) = 1(x + y) = x + y.$$

$$(x + y)(x + y') = x + xy + xy' + yy' = x(1 + y + y') = x.$$

Complement of a function

- Complement of a function *F* is *F’*
- Algebraically derived through **De Morgen's law –** interchanging **AND** and **OR** operators and complementing each literal

$$(A + B + C + D + \cdots + F)' = A'B'C'D' \dots F'$$

$$(ABCD \dots F)' = A' + B' + C' + D' + \cdots + F'$$

- **Example:**

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

$$F_1' = (x'yz' + x'y'z)' = (x'yz')'(x'y'z)' = (x + y' + z)(x + y + z')$$

- **Alternative Method:** take the dual of the function and complement each literal.

Minterms and Maxterms

- Binary Variable – appeared as x or x’
- **Minterm:** A product that contains all variables of a particular function in either complemented (0/absence) or non-complemented (1/present) form. **(Standard Product)**
  - n variables can be combined with **AND** to form 2^n minterms. (counting $0 \sim 2^n$)
  - Symbol of minterm – $\mathbf{m_j}$ (Where j = decimal equivalent)
- **Maxterm:** A sum that contains all variables of a particular function in either non-complemented (0/absence) or complemented (1/present) form. **(Standard Sum)**
  - n variables can be combined with **OR** to form 2^n maxterms. (counting $0 \sim 2^n$)
  - Symbol of maxterm – $\mathbf{M_j}$ (Where j = decimal equivalent)
- *Maxterm and minterm are complements each other ($m_j = M_j'$).*

Minterms & Maxterms

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 Functions Using Minterms & Maxterms

- **Boolean expression** – from a given *truth table*, form a minterm for each combination of the variables which produces a **1** in the function and then the **OR** of all of those terms.

$$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

$$f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$$

Boolean Functions Using Minterms & Maxterms…

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

Boolean Functions Complement - Minterms

- **Complement:** From the truth table, taking each combinations that produces 0in the function and then OR all of those terms.

$$f_1' = x'y'z' + x'yz' + x'yz + xy'z + xyz'$$

$f_2' =$?

Boolean Functions Using Maxterms

- **Boolean function using Maxterms**: Taking the complement of complemented function (f_1').

$$f_1 = (x + y + z)(x + y' + z)(x' + y + z')(x' + y' + z)(x + y' + z')$$
$$= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6$$

$$f_2 = (x + y + z)(x + y + z')(x + y' + z)(x' + y + z)$$
$$= M_0 M_1 M_2 M_4$$

- Any Boolean function can be written as a product (**AND**) of maxterms.
- *Boolean function written as a sum of minterms or product of maxterms* – **Canonical form**

Boolean Function in Canonical Form (Minterms)

- It is convenient to express the Boolean function in **canonical form** using minterms. (**Sum of minterms - SOP**)
  - Each term is inspected if it contains all the variables.
  - If any variable (x) is missed, **AND**ed that term with (x+x').

$$F = A + B'C$$

$$F = A'B'C + AB'C + AB'C + ABC' + ABC$$

$$= m_1 + m_4 + m_5 + m_6 + m_7$$

$$F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$$

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

Alternative method

Boolean Function in Canonical Form (Maxterms)

- Boolean function is also expressed in **canonical form** using maxterms (**Product of maxterms – POS**)
  - Each term is inspected if it contains all the variables.
  - If any variable (x) is missed, **OR**ed that term with (x.x').
  - Use **distributive law** to express. [**x+yz=(x+y)(x+z)**]

$$F = xy + x'z$$

$$F = (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z')$$
$$= M_0 M_2 M_4 M_5$$

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

Conversions between Canonical Forms

- $m_j = M_j'$
- Example:

$$F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$$

$$F'(A, B, C) = \Sigma(0, 2, 3) = m_0 + m_2 + m_3$$

$$F = (m_0 + m_2 + m_3)' = m_0' \cdot m_2' \cdot m_3' = M_0 M_2 M_3 = \Pi(0, 2, 3)$$

Conversions between Canonical Forms…

- Another 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 $F = xy + x'z$

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 form:**
  - Easily formed from truth table
  - Rarely having least number of literals (!!!)
- **Standard form:**
  - Having one, two, three or any number of literals.
  - Two types: SOP and POS

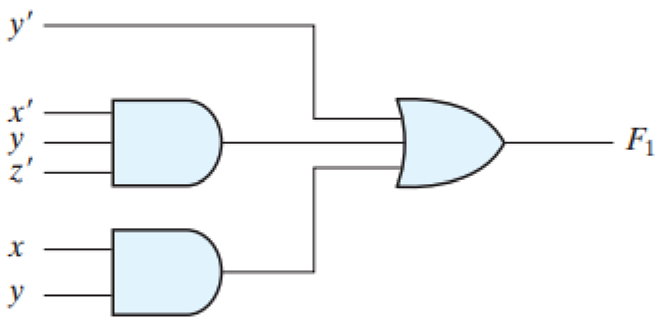
SOP: $F_1 = y' + xy + x'yz'$

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

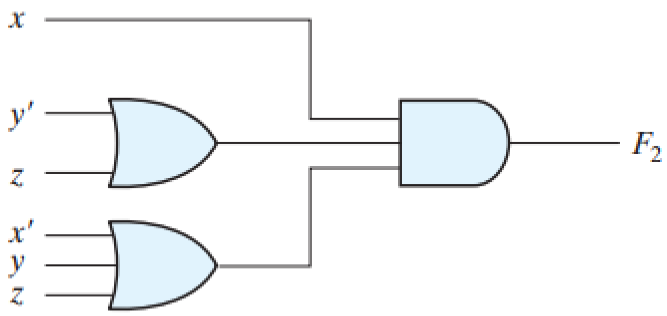
Non standard Form: $F_3 = AB + C(D + E)$

Standard Form: $F_3 = AB + C(D + E) = AB + CD + CE$

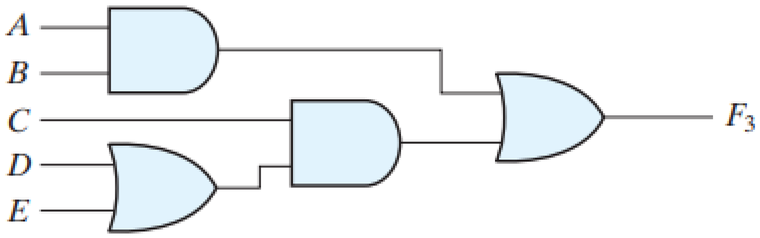
Standard Forms – 2 level Implementation



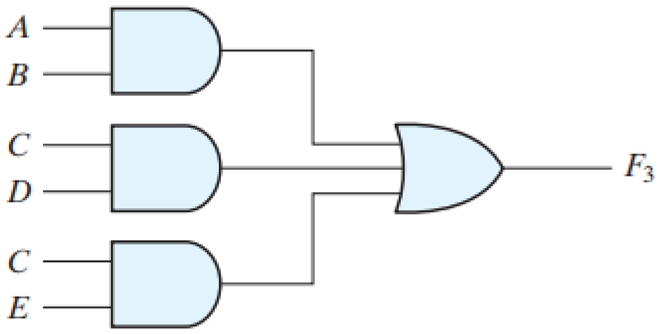
(a) Sum of Products



(b) Product of Sums



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



(b) $AB + CD + CE$

Note: It is assumed that Input variables are directly available in their complements.

Other Logic Operations

- For ***n*** number of variables-
  - 2^n number of rows of combinations (minterms or maxterms)
  - 2^{2^n} number of functions are possible (**How?**)

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

Other Logic Operations…

- Each of the functions from the previous slide with a name:

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

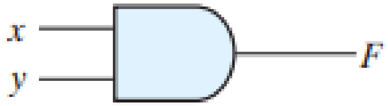
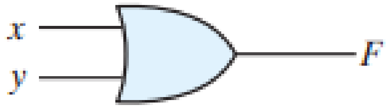
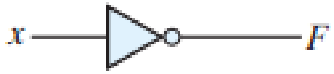
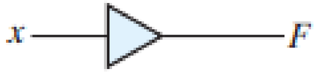
Other Logic Operations…

- 16 functions are subdivided into 3 categories:
  - 2 functions that produce **a constant** 0 or 1
  - 4 functions with **unary operations**: complement and transfer.
  - 10 functions with **binary operators** that define eight different operations: AND, OR, NAND, NOR, exclusive-OR, equivalence, inhibition, and implication.

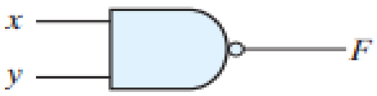
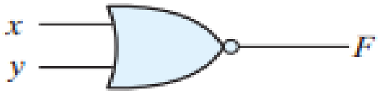
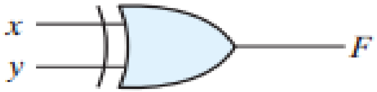
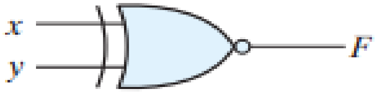
Digital Logic Gates

- Boolean functions are implemented with AND, OR, NOT gates
- Consideration to construct logic gates:
  - Feasibility and economy of producing gates with physical components
  - The possibility of extending the number of inputs more than two
  - Basic properties of binary operator like commutativity and associativity
  - The ability of the gate to implement Boolean function alone or in conjunction with others
- Among 16 functions from previous slide – 2 are constants and 4 are repeated – only 10 functions left to be considered as logic gates
- Implication and Inhibition don't follow the associative and commutative laws.

Digital Logic Gates…

<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</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>

Digital Logic Gates…

Name	Graphic symbol	Algebraic function	Truth table
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 Logic Gates…

- Small circle (Bubble) in the graphic symbol designates logical complement
- Triangle symbol designates a buffer circuit – doesn’t produce any logical operation – used only power amplification of signal
- NAND and NOR are used extensively as they are standard logic gates – constructed easily with transistors and any digital circuits can be implemented with them.

Extension Multiple Inputs

- A gate can be extended to have multiple inputs if it follows commutative and associative principles (e.g. AND, OR)

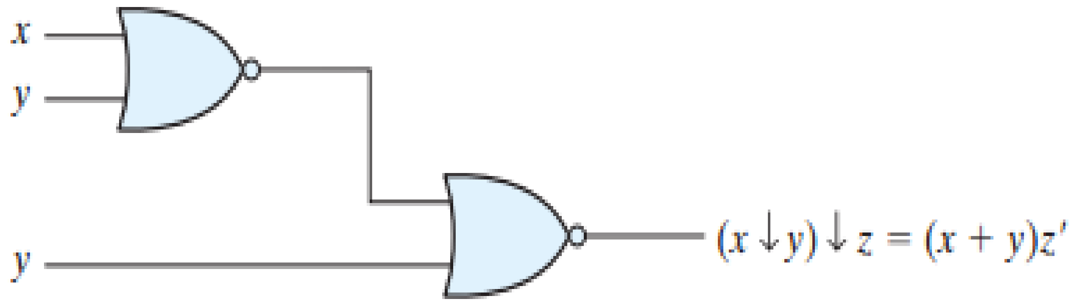
$$x + y = y + x \quad \text{(commutative)}$$

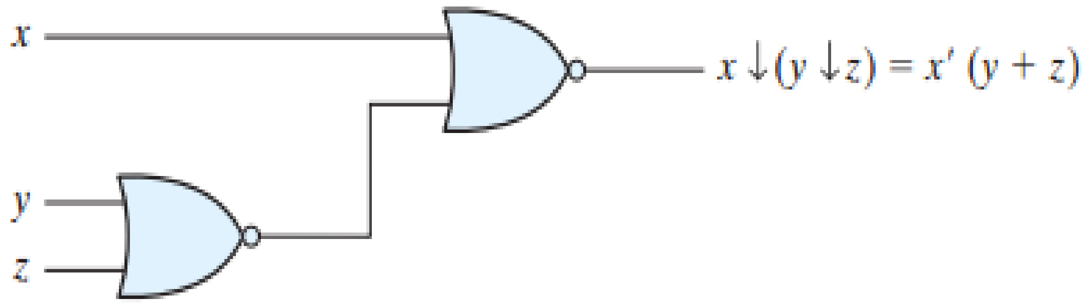
$$(x + y) + z = x + (y + z) = x + y + z \quad \text{(associative)}$$

- NAND and NOR are commutative and not associative so we can't extend them into multiple inputs in traditional way.

$$(x \downarrow y) \downarrow z = [(x + y)' + z]' = (x + y)z' = xz' + yz'$$
$$x \downarrow (y \downarrow z) = [x + (y + z)']' = x'(y + z) = x'y + x'z$$

Extension Multiple Inputs…





Extension Multiple Inputs…

- To overcome this problem, we define the multiple NOR (or NAND) gate as a complemented OR (or AND) gate:

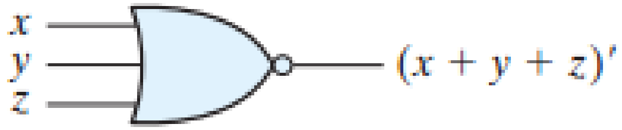
$$x \downarrow y \downarrow z = (x + y + z)'$$
$$x \uparrow y \uparrow z = (xyz)'$$

- In cascaded NOR and NAND operations, one use the parenthesis properly to make the sequence correct:

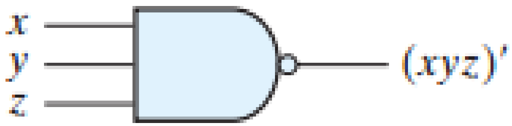
$$F = [(ABC)'(DE)']' = ABC + DE$$

- XOR and XNOR – both of them are associative and commutative – but definition will be modified (XOR – Odd function and XNOR – Even function)

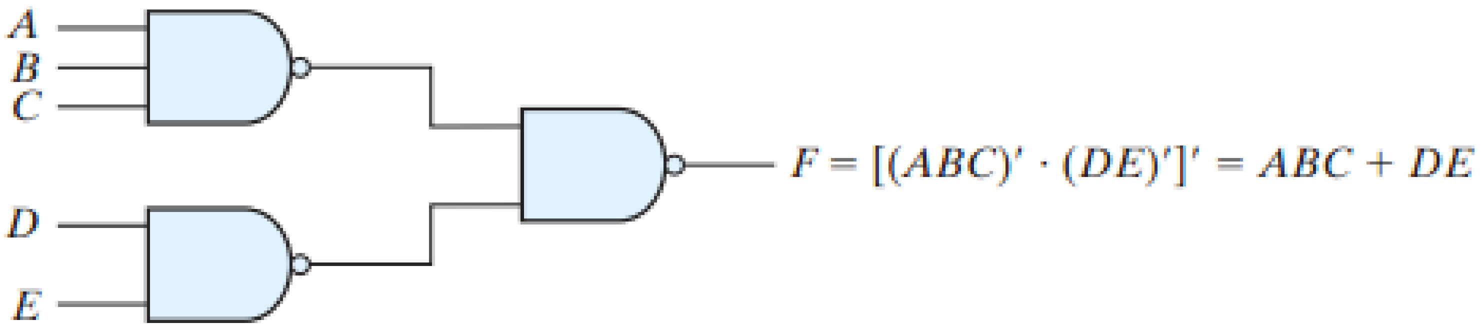
Extension Multiple Inputs…



(a) 3-input NOR gate

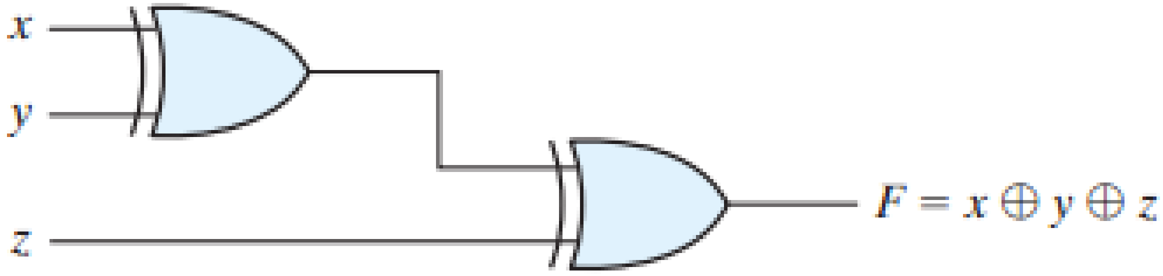


(b) 3-input NAND gate

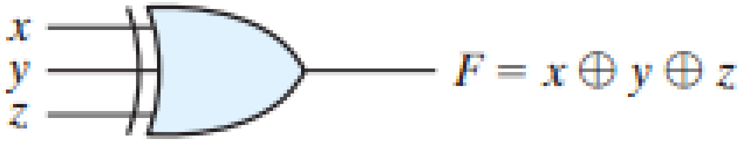


(c) Cascaded NAND gates

Extension Multiple Inputs…



(a) Using 2-input gates



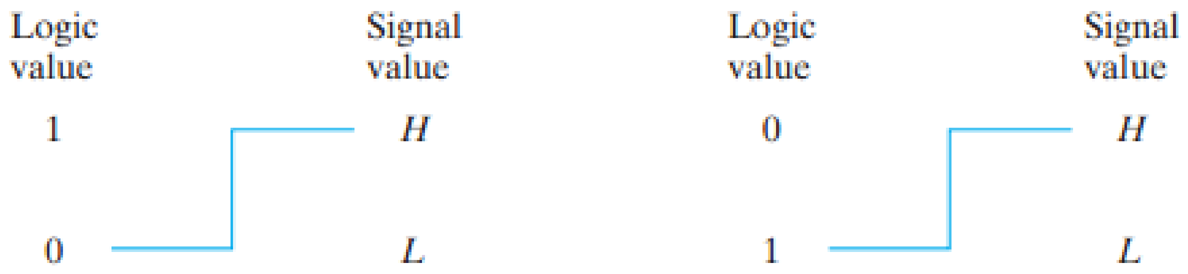
(b) 3-input gate

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

- Binary signal at inputs and outputs has one of two values – logic 1 and logic 0
- There are two different assignments of signal levels (**H**igh and **L**ow) to these logic values – assigning logic values to relative amplitudes of signal levels
- If **H** represent logic 1 – positive logic system
- If **L** represent logic 1 – negative logic system

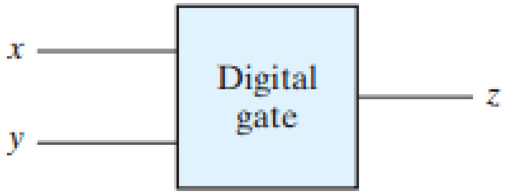


(a) Positive logic (b) Negative logic

Positive and Negative Logic…

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

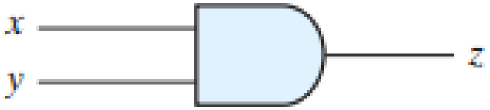
(a) Truth table with H and L



(b) Gate block diagram

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

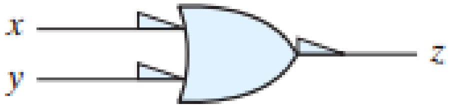
(c) Truth table for positive logic



(d) Positive logic AND gate

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

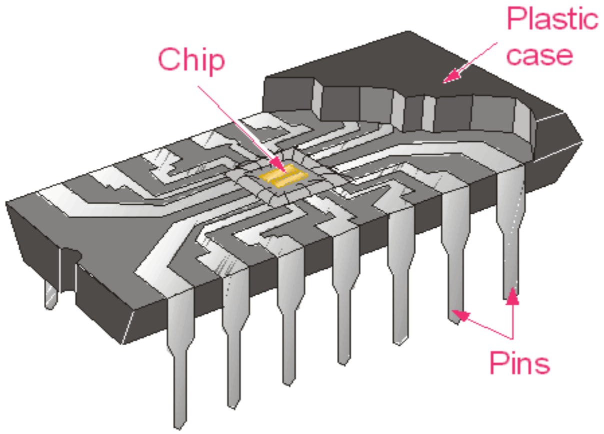
(e) Truth table for negative logic



(f) Negative logic OR gate

Integrated Circuits

- Digital circuit constructed with **integrated circuit**
- **IC** – small die (dice) of semiconductor crystal named as **chip**
  - **Components** of *Chip* – transistors, diodes, resistors, capacitors and so on
  - These components are interconnected inside
  - This chip is mounted (organized) on a **ceramic/plastic package** and connections are made of **external pin**
  - Differ from other detachable electronic circuit (with registers, capacitors)
  - Designation of IC printed on the surface
- **Two types** of packages
  - Flat
  - Dual in Line



Integrated Circuits…

- **Advantages:**
  - Small in size
  - Cost effective
  - Reduced power consumption
  - High reliability against failure
- **Linear and Digital IC** – for continuous signal and discrete signal
- Based on the **number of gates inside (complexity)** –
  - SSI – with several independent gates with their inputs and outputs (<10)
  - MSI – for specific elementary operations – decoder, encoder, adder, etc. (10-1000)
  - LSI – include digital system like processor, memory chip, etc. (>10,000)
  - VLSI – include complex microcomputer chip due to low cost and smaller in size (>1M)

Digital logic family

- TTL - transistor–transistor logic (standard)
- ECL - emitter-coupled logic (high speed)
- MOS - metal-oxide semiconductor (high component density)
- CMOS - complementary metal-oxide semiconductor (lower energy)

Digital logic family…

- **Parameters** in digital logic families:
  - Fan out
  - Fan in
  - Power dissipation
  - Propagation delay
  - Noise margin

E