Chapter 2
Boolean Algebra and Logic Gates

- The most common postulates used to formulate various algebraic structures are:

1. Closure. N={1,2,3,4…}, for any a,b $\in$N we obtain a unique c $\in$N by the operation a+b=c. Ex:2−3= −1 and 2,3 $\in$N, while (−1) $\notin$ N.
2. Associative law. A binary operator $*$ on a set S is said to be associative whenever

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

3. Commutative law.

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

2-1. Basic Definitions

4. Identity element. e is identity element which belongs to S.

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

   Ex: $x + \boxed{0} = 0 + x = x$ for any $x \in I=\{\ldots,-2, -1, 0, 1, 2,\ldots\}$

   $$x * \boxed{1} = 1 * x = x$$

5. Inverse. In the set of integers, I, with $e = 0$

   $$x * \boxed{y} = e \; ; \quad a + (\boxed{-a}) = 0$$

   $-a$ and y are inverse elements

6. Distributive law. If $*$ and $\cdot$ are two binary operators on a set S, $*$ is said to be distributive over $\cdot$ Whenever

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

2-1. Basic Definitions

- The operators and postulates have the following meanings:

The binary operator + defines addition.

The additive identity is 0.

The additive inverse defines subtraction.

The binary operator · defines multiplication.

The multiplicative identity is 1.

The multiplicative inverse of a = 1/a defines division, i.e.,

$$a \cdot 1/a = 1$$

The only distributive law applicable is that of · over +:

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

2-2. Axiomatic Definition of Boolean Algebra

- Boolean algebra is defined by a set of elements, B, provided following postulates with two binary operators, + and ·, are satisfied:

1. Closure with respect to the operators + and ·.
2. An identity element with respect to + and · is 0 and 1,respectively.
3. Commutative with respect to + and ·. Ex: $x + y = y + x$
4. + is distributive over · : $x + (y \cdot z)=(x + y) \cdot (x + z)$

   . is distributive over + : $x \cdot (y + z)=(x \cdot y) + (x \cdot z)$
5. Complement elements: $x + x' = 1$ and $x \cdot x' = 0$.
6. There exists at least two elements $x,y \in B$ such that $x \neq y$.

Comparing Boolean algebra with arithmetic and ordinary algebra.

1. Huntington postulates don’t include the associative law, however, this holds for Boolean algebra.
2. The distributive law of + over · is valid for Boolean algebra, but not for ordinary algebra.
3. Boolean algebra doesn’t have additive and multiplicative inverses; therefore, no subtraction or division operations.
4. Postulate 5 defines an operator called complement that is not available in ordinary algebra.
5. Ordinary algebra deals with the real numbers. Boolean algebra deals with the as yet undefined set of elements, B, in two-valued Boolean algebra, the B have two elements, 0 and 1.

Two-Valued Boolean Algebra

- With rules for the two binary operators + and . as shown in the following table, exactly the same as AND, OR , and NOT operations, respectively.
- From the tables as defined by postulate 2.

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

Diagram of the Distributive law

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

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

- To emphasize the similarities between two-valued Boolean algebra and other binary systems, this algebra was called “binary logic”. We shall drop the adjective “two-valued” from Boolean algebra in subsequent discussions.

2-3. Basic theorems and properties of Boolean algebra

- If the binary operators and the identity elements are interchanged, it is called the duality principle. We simply interchange OR and AND operators and replace 1's by 0's and 0's by 1's.

- The theorem 1(b) is the dual of theorem 1(a) and that each step of the proof in part (b) is the dual of part (a). Show at the slice after next slice.

Postulates and 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)	$x(x + y) = x$

Basic Theorems

- Basic Theorems: proven by the postulates of table 2-1 as shown above.

Theorem 1(a): $x + x = x$

	$= (x \boxed{+} x) \boxed{\cdot} 1$		by postulate 2(b)
	$= (x + x) \cdot (x + x')$		5(a)
	$= x + xx'$		4(b)
Dual	$= x + 0$	Dual	5(b)
	$= x$		2(a)

back

Theorem 1(b): $x \cdot x = x$

$= x \boxed{\cdot} x \boxed{+} 0$	by postulate 2(a)
$= xx + xx'$	5(b)
$= x (x + x')$	4(a)
$= x \cdot 1$	5(a)
$= x$	2(b)

Basic Theorems

Theorem 6(a): $x + xy = x$

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

- The theorems of Boolean algebra can be shown to hold true by means of truth tables.

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

First absorption theorem

Operator Precedence

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

x'	y'	$x'y'$
1	1	1
1	0	0
0	1	0
0	0	0

DeMorgan theorem

- The operator Precedence for evaluating Boolean expression is:

1. Parentheses
2. NOT
3. AND
4. OR

2-4. Boolean Functions

- Consider the following Boolean function:

  $F_1 = x + y'z$

- A Boolean function can be represented in a truth table.
- the binary combinations for the truth table obtained by counting from 0 through 2^n-1 see table 2-2[0~7(2^n-1)].

Table 2-2
Truth Tables for F_1 and F_2

x	y	z	F_1
0	0	0	0
0	0	1	$\boxed{1}$
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

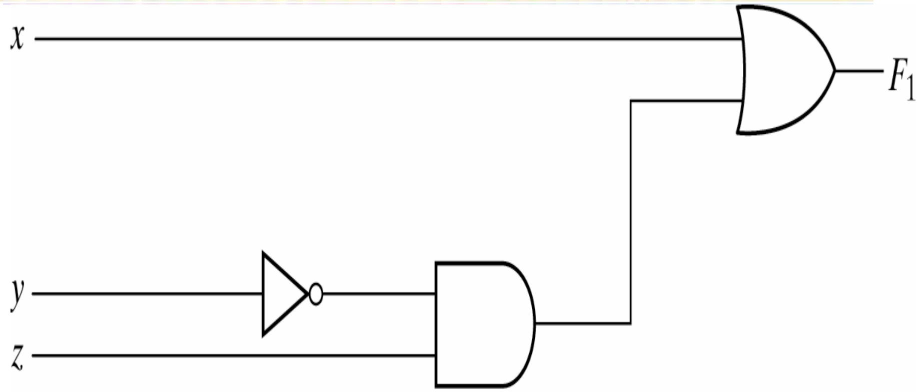


Fig. 2-1 Gate implementation of $F_1 = x + y'z$

Simplification of the algebraic

- There is only one way to represent Boolean function in a truth table.
- In algebraic form, it can be expressed in a variety of ways.
- By simplifying Boolean algebra, we can reduce the number of gates in the circuit and the number of inputs to the gate.

Before simplification of Boolean function

- Consider the following Boolean function:

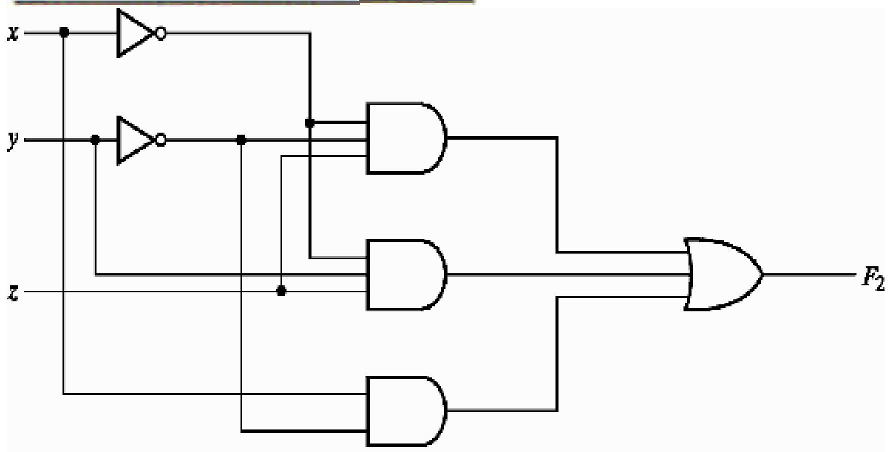
  $F_2 = x'y'z + x'yz + xy'$

This function with logic gates is shown in Fig. 2-2(a)

- The function is equal to 1 when xyz = 001 or 011 or when xyz = 10x .

Table 2-2
Truth Tables for F_1 and F_2

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



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

After simplification of Boolean function

- Simplify the following Boolean function:

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

- In 2-2 (b), would be preferable because it requires less wires and components.

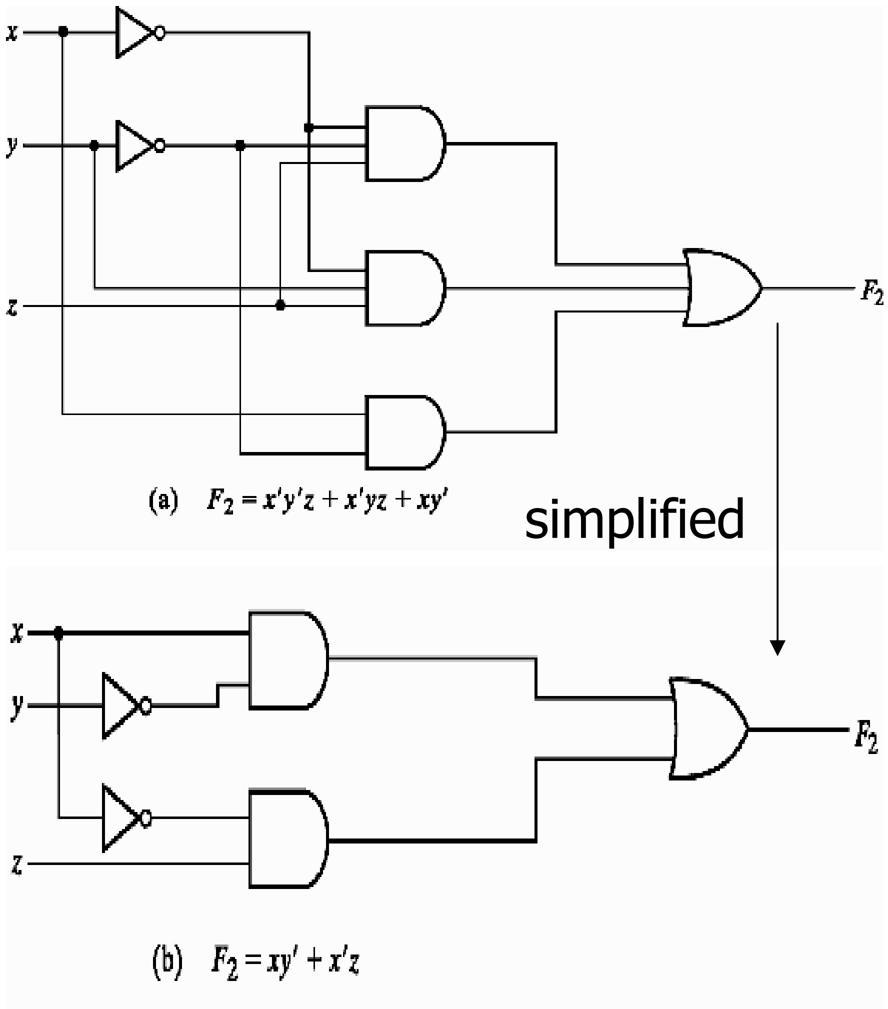


Fig. 2-2 Implementation of Boolean function F_2 with gates

Equivalent

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

$F_2=1$ when xyz=001 or 011 or

when xy=10x

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

F=1 when xz=01 or

when xy=10

- Since both expression produce the same truth table, they are said to be equivalent.

Table 2-2

Truth Tables for F_1 and F_2

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

Ex 2-1: 4. $\underbrace{\overbrace{x\,y}^{\text{counteracted}}}\; + \overbrace{x'\,z} + \boxed{y\,z}$ → Consensus term

Merged

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

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

$$= xy(1 + z) + x'z(1 + y)$$

$$= xy + x'z$$

Function 5 can be derived from the dual of the steps used to derive function 4.

- Functions 4 and 5 are known as the consensus theorem.

Complement of the function

Ex 2-2: Find the complement of the function $F_1 = x'yz' + x'y'z$ by applying DeMorgan's theorem.

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

Ex2-3: Find the complement of the function $F_1 = x'yz' + x'y'z$ by taking their dual and complementing each literal.

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

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

2-5. Canonical and Standard forms

- n variables can form 2^n($0 \sim 2^n-1$) Minterms, so does Maxterms (Table 2-3).
- Minterms and Maxterms

  Minterms: obtain from an AND term of the n variables, or called standard product.

  Maxterms: n variables form an OR term, or called standard sum.
- Each Maxterm is the complement of its corresponding Minterm, and vice versa.
- A sum of minterms or product of maxterms are said to be in canonical form.

Minterms & Maxterms

Table 2-3
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

Sum of Minterms

- From a truth table can express a minterm for each combination of the variables that produces a 1 in a Boolean function, and then taking the OR of all those terms.

<EX.> Upon the table 2-4 that produces 1 in $f_1=1$:

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

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

$$\Rightarrow f_1 = M_0 \bullet M_2 \bullet M_3 \bullet M_5 \bullet M_6$$

Table 2-4
Functions of Three Variables

x	*y*	*z*	Function 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	0
1	1	0	0
1	1	1	1

Example

Ex.2-4 Express the Boolean function $F = A + B'C$ in a sum of minterms.

A→ lost two variables

$A = A(B+B') = AB + AB'$ → still missing one variable

$A = AB(C + C') + AB'(C + C')$

$=ABC + ABC' + AB'C + AB'C'$

B'C→ lost one variable

$B'C = B'C(A + A') = AB'C + A'B'C$

Combining all terms

$$F = A'B'C + AB'C' + AB'C + ABC' + ABC = m_1 + m_4 + m_5 + m_6 + m_7$$

Convenient expression

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

Table 2-5 is a directly derivation by using truth table.

Product of Maxterms

Ex.2-5 Express the Boolean function $F = xy + x'z$ in a product of maxterm form.

using distributive law→ $F = xy + x'z = (xy+x')(xy+z)$

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

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

Each OR term missing one variable

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

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

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

Combining all the terms

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

$= M_0M_2M_4M_5$

A convenient way to express this function

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

Conversion between canonical forms

Ex. Boolean expression: F = xy + x'z

xy = 11 or xz = 01

sum of minterms is

F(x, y, z) = $\sum$(1, 3, 6, 7)

Since have a total of eight minterms or maxterms in a function of three variable.

product of maxterms is

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

Take complement of F' by DeMorgan's theorem

- To convert from one canonical form to another, interchange the symbols $\sum$ and $\prod$ and list those numbers missing from the original form.

Table 2-6
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

Standard forms

- Another way to express Boolean functions is in standard form.

1. Sum of products(SOP): $F_1 = y' + xy + x'yz'$
2. Product of sums(POS): $F_2 = x(y' + z)(x' + y + z')$

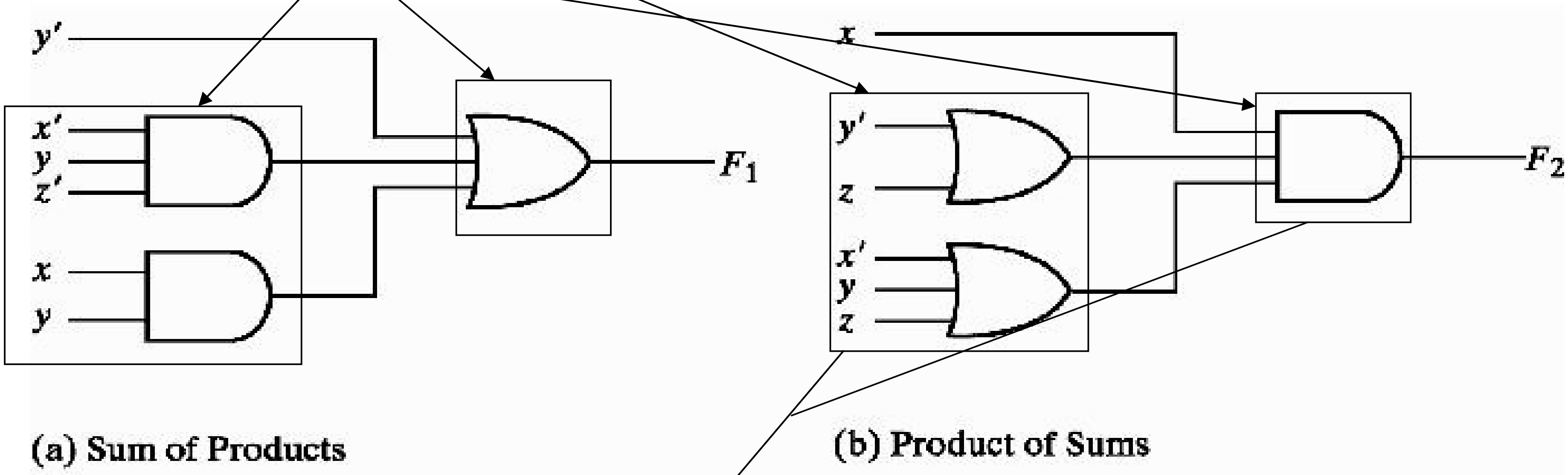


Fig. 2-3 Two-level implementation

Standard forms

- F_3 is a non-standard form, neither in SOP nor in POS.
- F_3 can change to a standard form by using distributive law and implement in a SOP type.

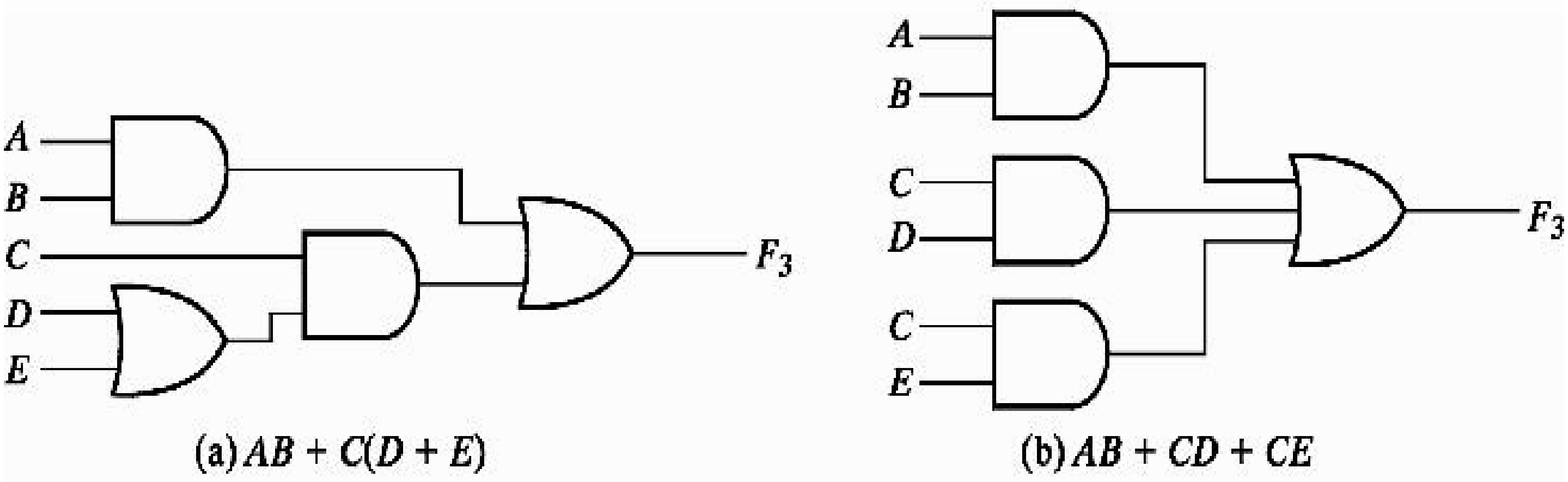


Fig. 2-4 Three- and Two-Level implementation

2-6. Other logic operations

- There are 2^{2^n} functions for n binary variables, for two variables, n=2, and the possible Boolean functions is 16. see tables 2-7 and 2-8.

inhibition XOR equivalence implication

Table 2-7

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

Other logic operations

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

Function categories

- The 16 functions listed in table 2-8 can be subdivided into three categories:

1. Two functions that produce a constant 0 or 1.
2. Four functions with unary operations: complement and transfer.
3. Ten functions with binary operators that define eight different operations: AND, OR, NAND, NOR, exclusive-OR, equivalence, inhibition, and implication.

Impracticality in standard logic gates

2-7. Digital logic gates

- The graphic symbols and truth tables of the gates of the eight different operations are shown in Fig.2-5

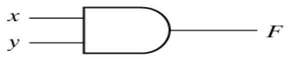
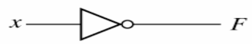
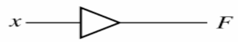
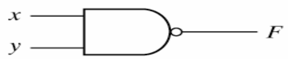
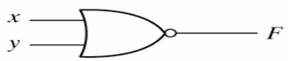
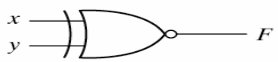
Name	Graphic symbol	Algebraic function	Truth table
AND		$F = xy$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$F = x'$	x | F 0 | 1 1 | 0
Buffer		$F = x$	x | F 0 | 0 1 | 1
NAND		$F = (xy)'$	x y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = (x + y)'$	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

Fig. 2-5 Digital logic gates

Extension to multiple inputs

- In Fig.2-5, except for the inverter and buffer, can be extended to have more than two inputs.
- The AND and OR operations possess two properties: commutative and associative.

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

Non-associativity of the NOR operator

- The NAND and NOR functions are commutative, not associative .

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

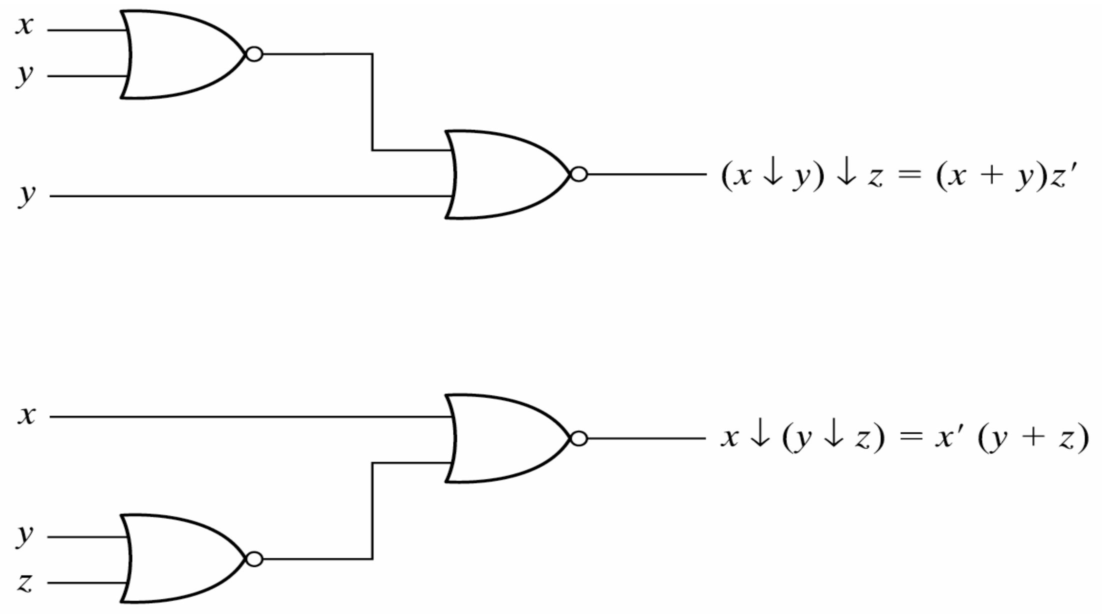


Fig. 2-6 Demonstrating the nonassociativity of the NOR operator; $(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$

Cascade of NAND gates

- In writing cascaded NOR and NAND operations, one must use the correct parentheses to signify the proper sequence of the gates.

Fig.2-7

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

obtain from DeMorgan's theorem.

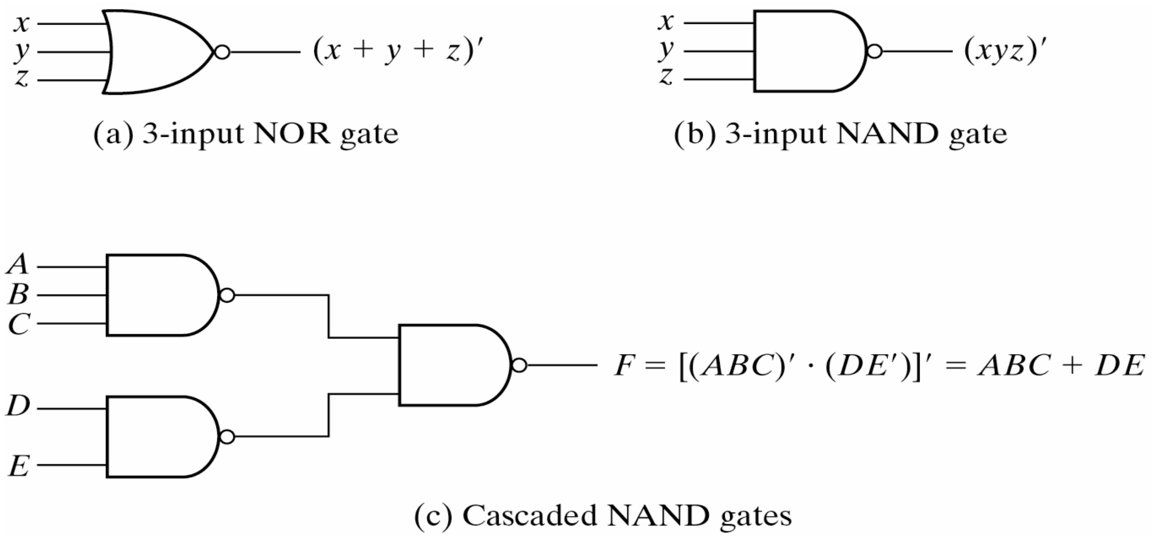
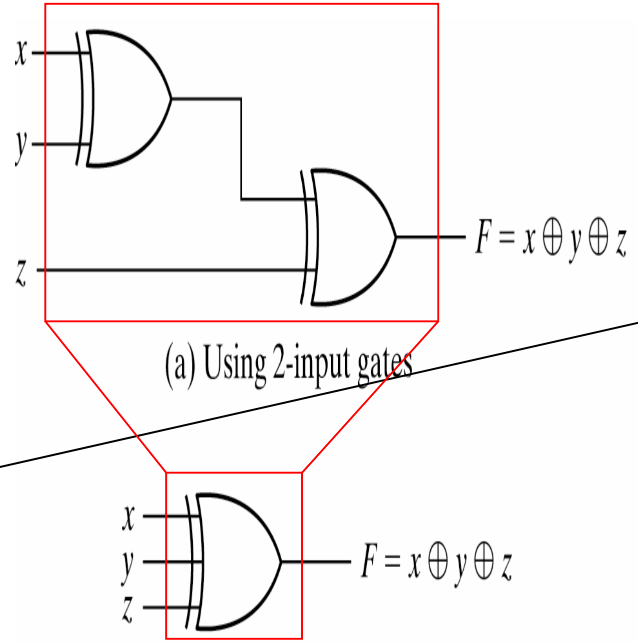


Fig. 2-7 Multiple-input and cascated NOR and NAND gates

XOR gate property

- The XOR and equivalence gates are both commutative and associative and can be extended to more than two inputs.
- The XOR is an odd function.
- The three inputs XOR is normally implemented by cascading 2-input gates.



(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

Fig. 2-8 3-input exclusive-OR gate

Positive and negative logic

- We assign to the relative amplitudes of the two signal levels as the high-level and low-level (Fig.2-9).

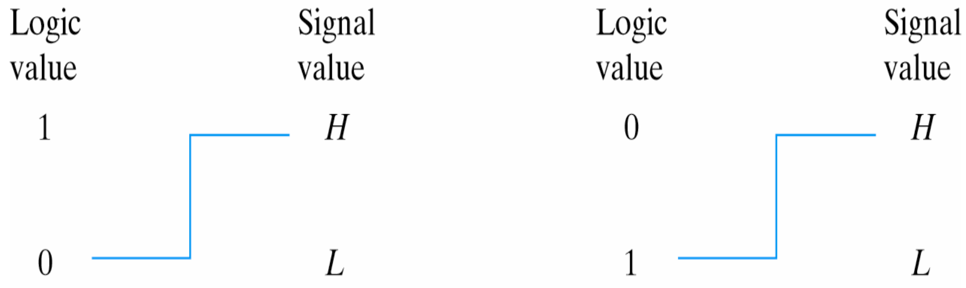
1. High-level (H): represent logic-1 as a positive logic system.
2. Low-level (L): represent logic-0 as a negative logic system.

- It is up to the user to decide on a positive or negative logic polarity between some certain potential.

Positive and negative logic

Ex. The electronic shown in Fig.2-10(b), truth table listed in (a).

It specifies the physical behavior of the gate when H is 3 volts and L is 0 volts.

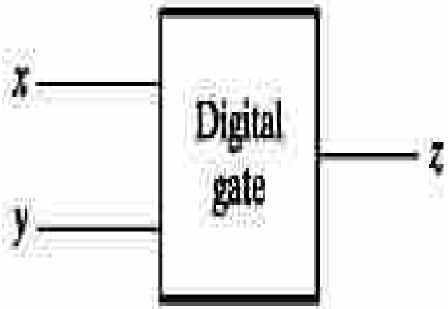


(a) Positive logic (b) Negative logic

Fig. 2-9 signal assignment and logic polarity

x	y	F
L	L	L
L	H	L
H	L	L
H	H	H

(a) Truth table with H and L



(b) Gate block diagram

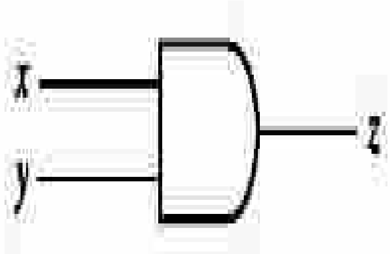
Fig. 2-10 Demonstration of positive and negative logic

Positive logic

- The truth table of Fig.2-10(c) assumes positive logic assignment , with H=1 and L=0.
- It is the same as the one for the AND operation.

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

Fig.2-10 Demonstration of positive and negative logic

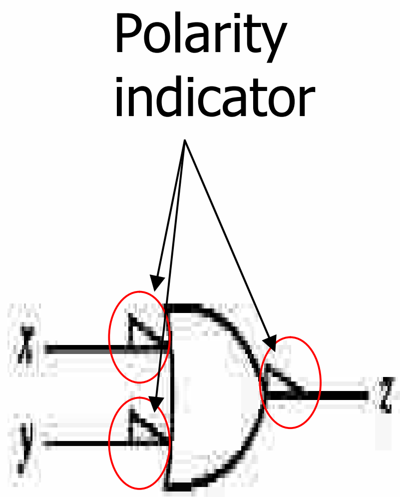
Negative logic

- The table represents the OR operation even though the entries are reversed.

- The conversion from positive logic to negative logic, and vice versa, is essentially an operation that changes 1's to 0's and 0's to 1's(dual) in both the inputs and the outputs of a gate.

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

(e) Truth table for negative logic

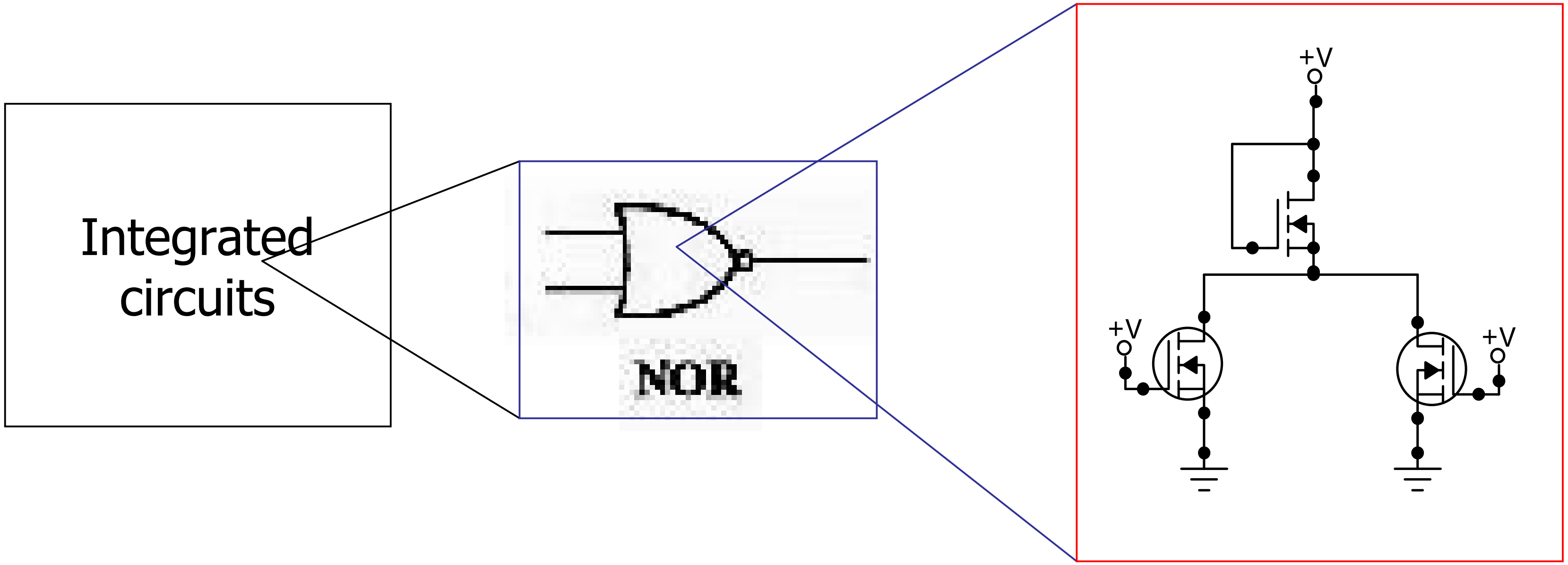


(f) Negative logic OR gate

Fig 2-10 Demonstration of positive and negative logic

2-8. Integrated circuits

- An integrated circuit(IC) is a silicon semiconductor crystal, called chip, containing the electronic components for constructing digital gates.



Levels of integration

1. Small-scale integration(SSI): the number of gates is usually fewer than 10 and is limited by the number of pins available in the IC.

2. Medium-scale integration(MSI): have a complexity of approximately 10 to 1000 gates in a single package, and usually perform specific elementary digital operations.

   Ex. Adders, multiplexers…(chapter 4), registers, counters(chapter6).

Levels of integration

3. Large-scale integration(LSI): contain thousands of gates in a single package.

   Ex. Memory chips, processors.

4. Very large-scale integration(VLSI): contain hundred of thousands of gates within a single package.

   Ex. Large memory arrays and complex microcomputer chips.

Digital logic families

- The circuit technology is referred to as a digital logic family.
- The most popular circuit technology:

1. TTL: transister-transister logic:

   has been used for a long time and is considered as standard; but is declining in use.

2. ECL: emitter-coupled logic:

   has high-speed operation in system; but is declining in use.

3. MOS: metal-oxide semiconductor:

   has high component density.

Fan out & fan in

- CMOS: has low power consumption, essential for VLSI design, and has become the dominant logic family.

- Fan out specifies the number of standard loads that the output of a typical gate can drive without impairing its normal operation.

- Fan in is the number of inputs available in a gate.

Power dissipation & Propagation delay & Noise margin

- Power dissipation is the power consumed by the gate that must be available from the power supply.
- Propagation delay is the average transition delay time for the signal to propagate from input to output.
- Noise margin is the maximum external noise voltage added to an input signal that does not cause an undesirable change in the circuit output.

Computer-Aided design (CAD)

- The design of digital systems with VLSI circuits are very complexity to develop and verify with using CAD tools.
- We can choose between an application specific integrated circuit (ASIC), a field-programmable gate array (FPGA), a programmable logic device (PLD), or a full-custom IC.
- HDL is an important development tool in the design of digital systems.