Digital Logic Design
Chapter 2

Boolean Algebra and Logic Gate

2.1 Algebras

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

2.2 BASIC DEFINITIONS

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

BASIC DEFINITIONS

- The common postulates used to formulate algebraic structures are:

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

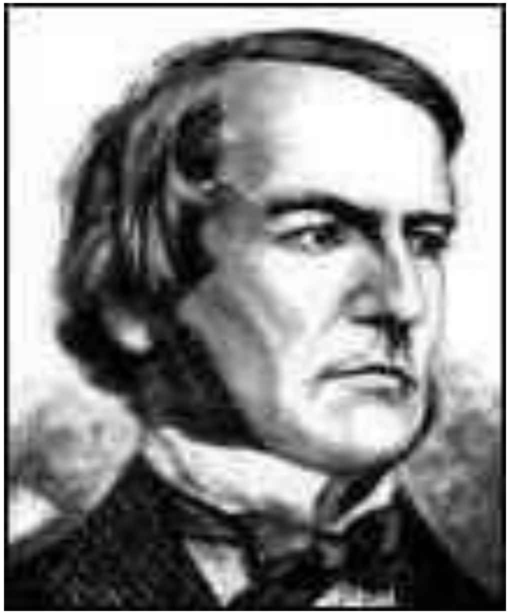
BASIC DEFINITIONS

4. ***Identity element***: a set S is said to have an identity element with respect to a binary operation * on S if there exists an element $e \in S$ with the property that
   - $e * x = x * e = x$ **for every** $x \in S$
     - $0+x = x+0 = x$ **for every** $x \in I$ $I = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$.
     - $1 \times x = x \times 1 = x$ **for every** $x \in I$ $I = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$.
5. ***Inverse***: a set having the identity element **e** with respect to the binary operator to have an inverse whenever, for every $x \in S$, there exists an element $y \in S$ such that
   - $x * y = e$
     - **The operator + over I, with $e = 0$, the inverse of an element a is $(-a)$, since $a+(-a) = 0$.**
6. ***Distributive law***: if (*) and (.) are two binary operators on a set S, (*) is said to be distributive over (.) whenever
   - $x * (y.z) = (x * y).(x * z)$

George Boole

Father of Boolean algebra

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



George Boole (1815 - 1864)

Boolean Algebra

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

Postulates of Two-Valued Boolean Algebra

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

AND

x	y	$X.y$
0	0	0
0	1	0
1	0	0
1	1	1

OR

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

NOT

x	X'
0	1
1	0

1. Closure (+ and‧)
2. The identity elements

   (1) + = 0

   (2) ‧ = 1

Postulates of Two-Valued Boolean Algebra

3. The commutative laws $x+y = y+x, \quad x.y = y.x$
4. The distributive laws

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

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

2.4 Basic Theorems And Properties Of Boolean Algebra
Duality

- The principle of *duality* is an important concept. This says that if an expression is valid in Boolean algebra, the dual of that expression is also valid.
- To form the dual of an expression, replace all (+) operators with (·) operators, all (·) operators with (+) operators, all ones with zeros, and all zeros with ones.
- Form the dual of the expression **a(b + c) = ab + ac**

  a(b + c) = a + (bc) = (a + b)(a + c)

- Take care not to alter the location of the parentheses if they are present.

Basic Theorems

Table 2.1
Postulates and Theorems of Boolean Algebra

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

DeMorgan's Theorem

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

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

Operator Precedence

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

2.5 Boolean Functions

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

Boolean Functions

- The truth table of 2^n entries (n=number of variables)

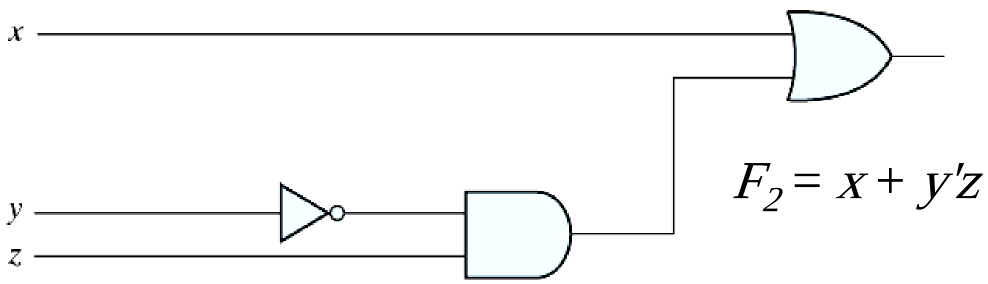
x	y	z	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

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

Boolean Functions

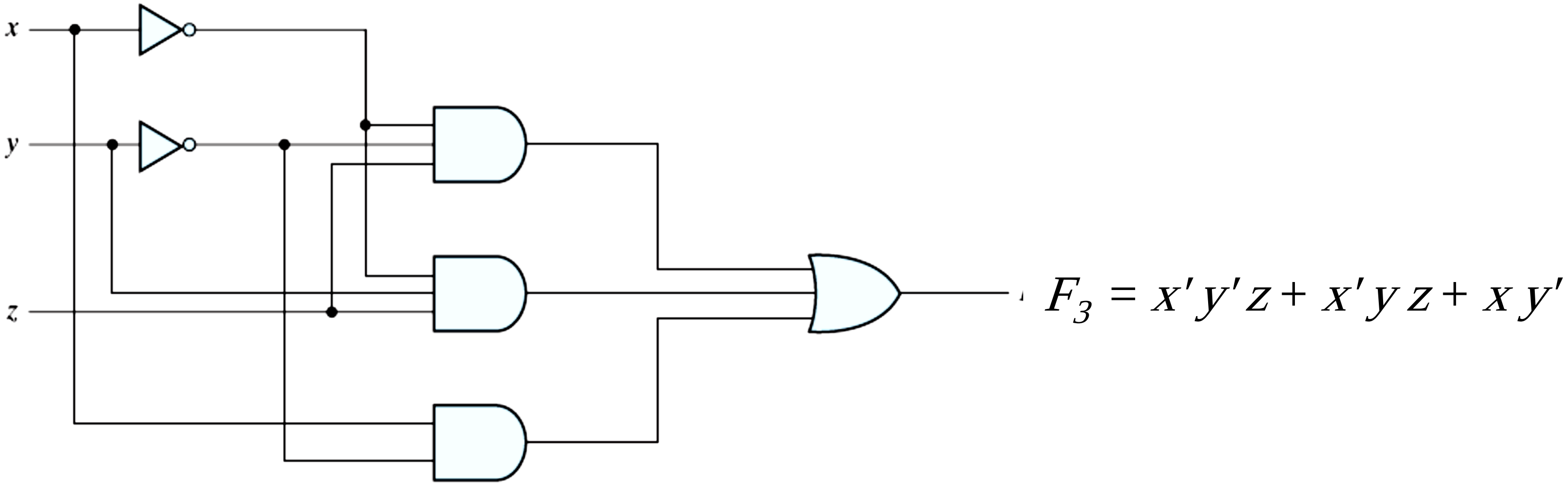
Implementation with logic gates

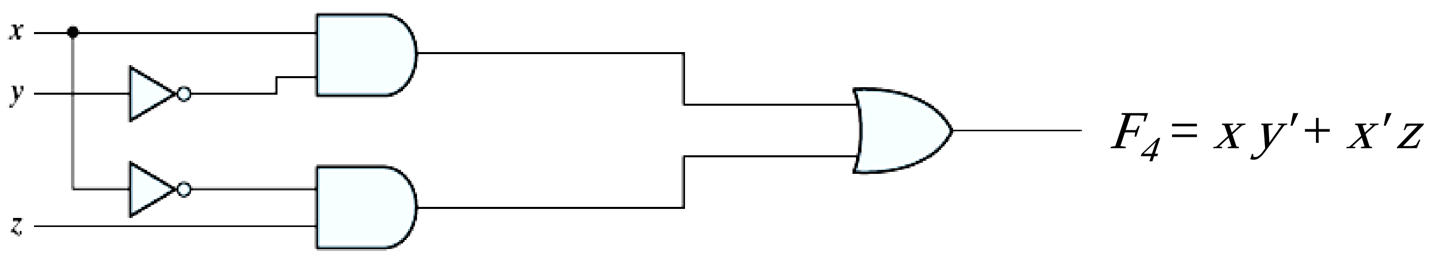
- When a Boolean expression is implemented with logic gates, each **term** requires a gate and each variable (**Literal**) within the term designates an input to the gate. (F3 has 3 terms and 8 literal)
- $F_2 = x + y'z$



Boolean Functions

- Implementation with logic gates
  - F_4 is more economical





Algebraic Manipulation

- To minimize Boolean expressions, minimize the number of literals and the number of terms → a circuit with less equipment
  - It is a hard problem (no specific rules to follow)

- Example 2.1

  1. $\boldsymbol{x(x'+y)} = xx' + xy = 0 + xy = xy$

  2. $\boldsymbol{x+x'y} = (x+x')(x+y)$ [Distributive] $= 1\,(x+y) = x+y$

  3. $\boldsymbol{(x+y)(x+y')} = x+xy+xy'+yy' = x+xy+xy'+0 = x(1+y+y') = x$

  4. $\boldsymbol{xy + x'z + yz} = xy + x'z + yz(x+x') = xy + x'z + yzx + yzx' =$

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

Complement of a Function

- An interchange of 0's for 1's and 1's for 0's in the value of F
  - By DeMorgan's theorem
  - $(A+B+C)' = (A+X)'$ let $B+C = X$

    $= A'X'$ by theorem 5(a) (DeMorgan's)

    $= A'(B+C)'$ substitute $B+C = X$

    $= A'(B'C')$ by theorem 5(a) (DeMorgan's)

    $= A'B'C'$ by theorem 4(b) (associative)
- Generalization: a function is obtained by interchanging AND and OR operators and complementing each literal.
  - $(A+B+C+D+ \dots +F)' = A'B'C'D' \dots F'$
  - $(ABCD \dots F)' = A'+ B'+C'+D' \dots +F'$

Examples

- Example 2.2: **(1)** $F_1 = x'yz' + x'y'z$ ***(2)*** $F_2 = x(y'z' + yz)$
  - $F_1' = (x'yz' + x'y'z)' = (x'yz')'(x'y'z)' = (x+y'+z)(x+y+z')$
  - $F_2' = [x(y'z'+yz)]' = x' + (y'z'+yz)' = x' + (y'z')'(yz)'$

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

    $= x' + yz'+y'z$
- Example 2.3: A simpler procedure
  - **Take the dual of the function and complement each literal**

  1. $F_1 = x'yz' + x'y'z.$
     a) The dual of F_1 is $(x'+y+z')(x'+y'+z)$.
     b) Complement each literal: $(x+y'+z)(x+y+z') = F_1'$
  2. $F_2 = x(y'z' + yz).$
     a) The dual of F_2 is $x+(y'+z')(y+z)$.
     b) Complement each literal: $x'+(y+z)(y'+z') = F_2'$

2.6 Canonical and Standard Forms

Minterms and Maxterms

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

Minterms and Maxterms

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

Table 2.3
Minterms and Maxterms for Three Binary Variables

			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

- An Boolean function can be expressed by
  - A truth table
  - Sum of minterms for each combination of variables that produces a (1) in the function.
  - $f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$ (Minterms)
  - $f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$ (Minterms)

Table 2.4
Functions of Three Variables

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

Minterms and Maxterms

- The complement of a Boolean function
  - The minterms that produce a (0)
  - $f_1' = m_0 + m_2 + m_3 + m_5 + m_6 = x'y'z' + x'yz' + x'yz + xy'z + xyz'$
  - $f_2' = m_0 + m_1 + m_2 + m_4$
- Any Boolean function can be expressed as
  - A sum of minterms ("sum" meaning the ORing of terms).
  - A product of maxterms ("product" meaning the ANDing of terms).
  - Both boolean functions are said to be in **Canonical** form.
  - $f_1 = (f_1')'$
  - $= (x+y+z)(x+y'+z)\,(x+y'+z')\,(x'+y+z')(x'+y'+z) = M_0\,M_2\,M_3\,M_5\,M_6$
  - $f_2 = (x+y+z)(x+y+z')(x+y'+z)(x'+y+z) = M_0 M_1 M_2 M_4$

Sum of Minterms

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

Table 2.5
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

Conversion between Canonical Forms

- The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function.
  - $F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$
  - Thus, $F'(A, B, C) = \Sigma(0, 2, 3)$
  - By DeMorgan's theorem

    $F(A, B, C) = \Pi(0, 2, 3)$

    $F'(A, B, C) = \Pi(1, 4, 5, 6, 7)$
  - $m_j' = M_j$
- To convert from one canonical form to another: **interchange** the symbols Σ and Π and list those numbers **missing** from the original form
  - » Σ of 1's
  - » Π of 0's

Example

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

Table 2.6
Truth Table for $F = xy + x'z$

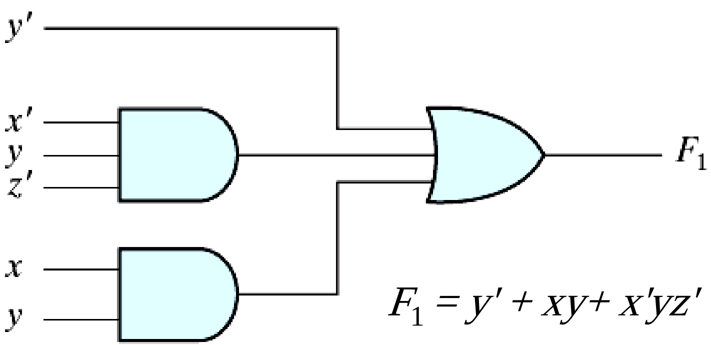
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Standard Forms

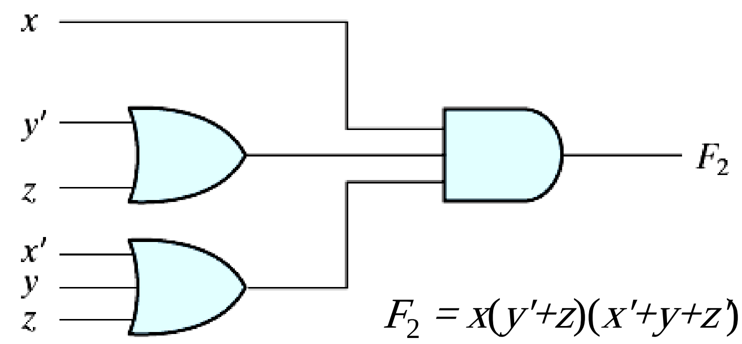
- In canonical forms each minterm or maxterm must contain **all the variables** either complemented or uncomplemented, thus these forms are very seldom the ones with the least number of literals.
- Standard forms: the terms that form the function may obtain **one, two, or any number** of literals, .There are two types of standard forms:
  - Sum of products: $F_1 = y' + xy + x'yz'$
  - Product of sums: $F_2 = x(y'+z)(x'+y+z')$
- A Boolean function may be expressed in a nonstandard form
  - $F_3 = AB + C(D + E)$
- But it can be changed to a standard form by using The distributive law
  - $F3 = AB + C(D + E) = AB + CD + CE$

Implementation

- Two-level implementation

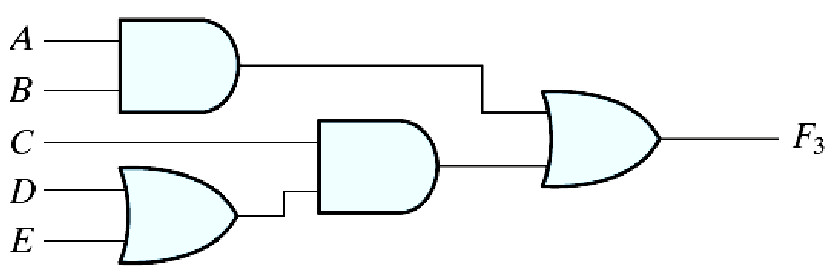


(a) Sum of Products

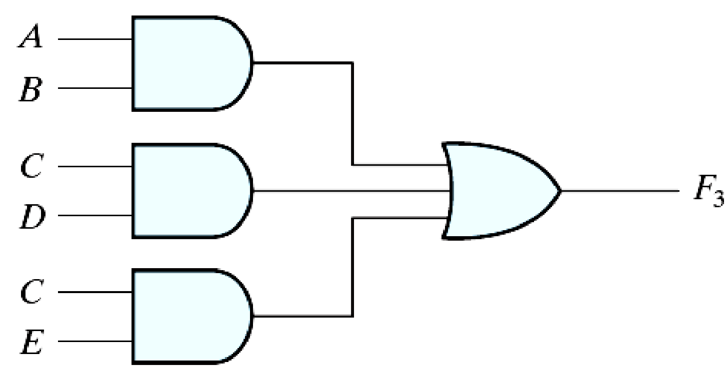


(b) Product of Sums

- Multi-level implementation



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



(b) $AB + CD + CE$

2.7 Other Logic Operations

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

Table 2.7
Truth Tables for the 16 Functions of Two Binary Variables

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

Boolean Expressions

Table 2.8
Boolean Expressions for the 16 Functions of Two Variables

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

Standard Gates

- Consider the 16 functions in Table 2.8
  - **Two** functions produce a constant : (F_0 and F_{15}).
  - **Four** functions with unary operations: complement and transfer: (F_3, F_5, F_{10} and F_{12}).
  - The other **ten** functions with binary operators
- **Eight** function are used as standard gates : complement (F_{12}), transfer (F_3), AND (F_1), OR (F_7), NAND (F_{14}), NOR (F_8), XOR (F_6), and equivalence (XNOR) (F_9).
  - Complement: inverter.
  - Transfer: buffer (increasing drive strength).
  - Equivalence: XNOR.

Summary of Logic Gates

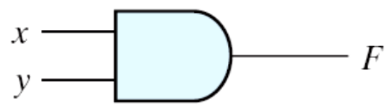
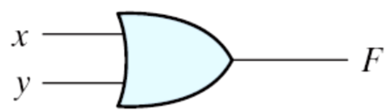
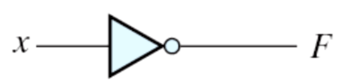
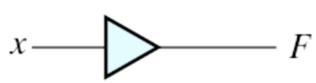
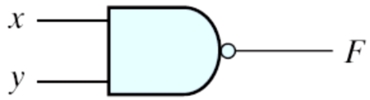
<table>
<tr><th>Name</th><th>Graphic symbol</th><th>Algebraic function</th><th>Truth table</th></tr>
<tr><td>AND</td><td></td><td>$F = xy$</td><td><table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</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>

Figure 2.5 Digital logic gates

Summary of Logic Gates

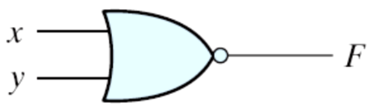
NAND



$F = (xy)'$

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

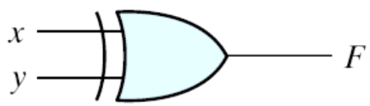
NOR



$F = (x + y)'$

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

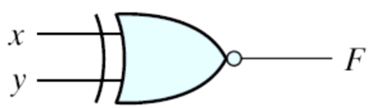
Exclusive-OR (XOR)



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

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

Figure 2.5 Digital logic gates

Multiple Inputs

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

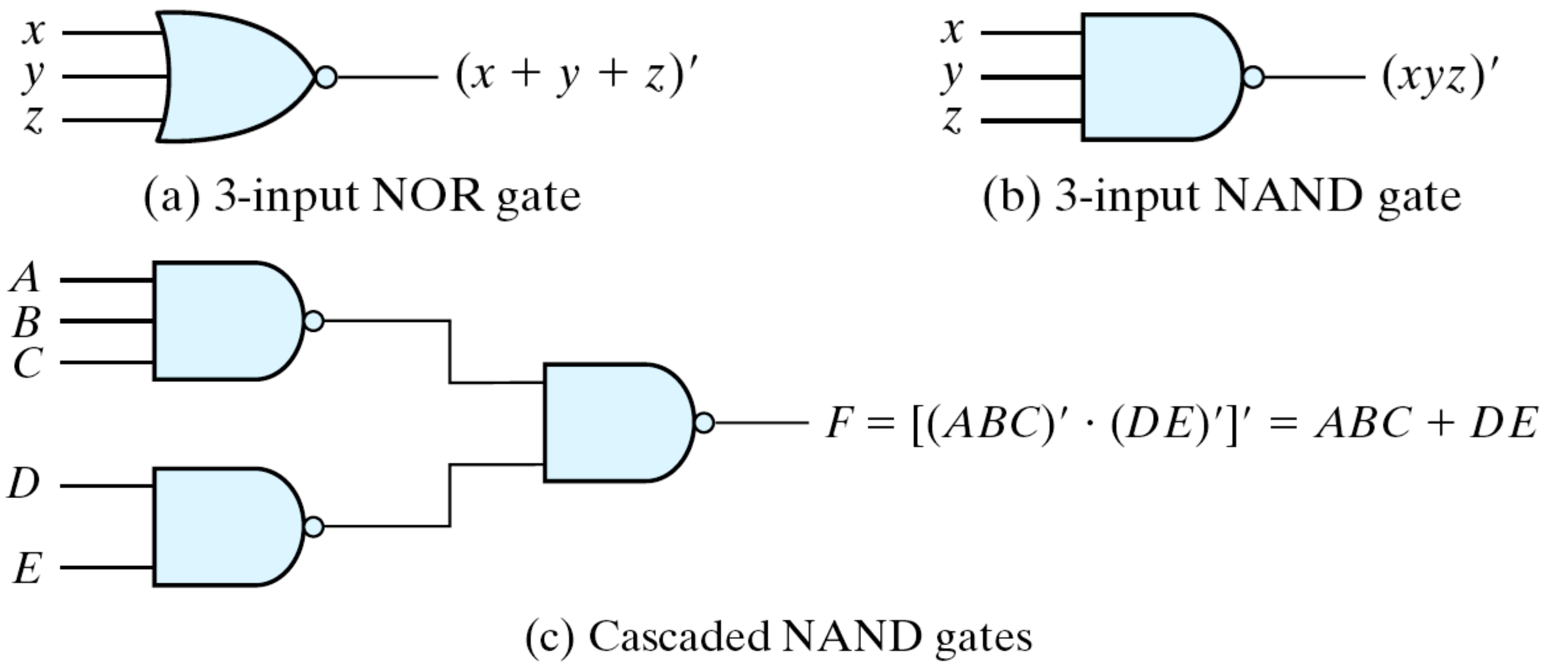
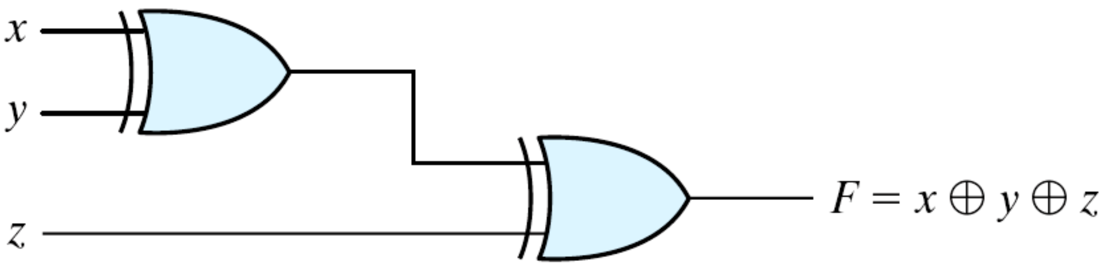


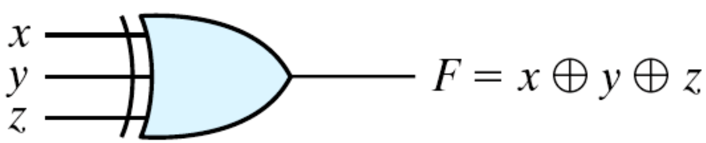
Figure 2.7 Multiple-input and cascated NOR and NAND gates

Multiple Inputs

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



(a) Using 2-input gates



(b) 3-input gate

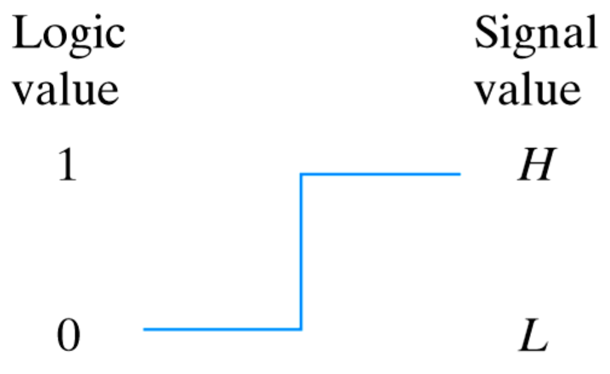
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(c) Truth table

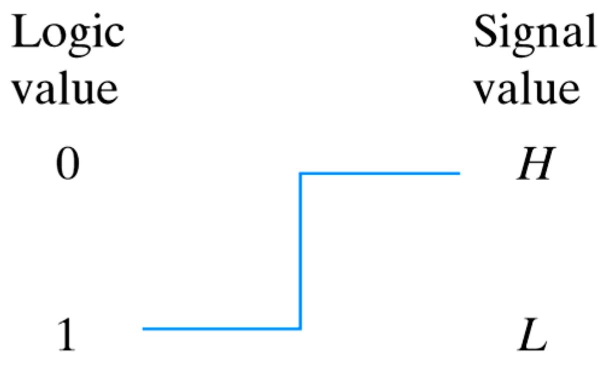
Figure 2.8 3-input XOR gate

Positive and Negative Logic

- 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 gates
  - A positive logic AND gate
  - A negative logic OR gate



(a) Positive logic



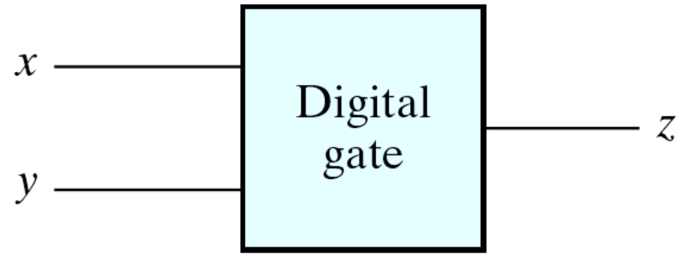
(b) Negative logic

Figure 2.9 Signal assignment and logic polarity

Positive and Negative Logic

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

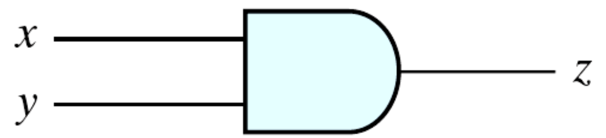
(a) Truth table with H and L



(b) Gate block diagram

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

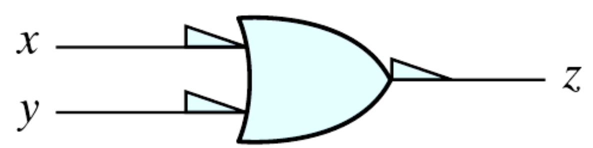
(c) Truth table for positive logic



(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

Figure 2.10 Demonstration of positive and negative logic

2.9 Integrated Circuits

Level of Integration

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

Digital Logic Families

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

Home Work (3)

Digital Design (**4th**)- Morris Mano-Page 66- Problems:

2.3 d,f
2.4 d,e
2.6 Only for (2.3 d,f)
2.7 Only for (2.4 d,e)
2.9
2.20
2.22

Home Work (4)

Digital Design (**4th**)- Morris Mano-Page 66-Problems:

2.13
2.14
2.15
2.27
2.28