Chapter 2

Boolean Algebra and Logic Gates

Outline

- Basic Definitions
- Basic Theorems and Properties
- Boolean Functions
- Canonical and Standard Forms
- Digital Logic Gates
- Integrated Circuits

Basic Definition

Boolean Algebra:

- A deductive mathematical system
- Defined with
  - A set of elements
    - ex: B = {0, 1}
  - A set of operators
    - ex: +, *, …
  - A number of unproved axioms or postulates

Most Common Postulates (1/2)

- Closure: (封閉空間)
  - The result of any operators is still in a set *S* if all operands are also in set *S*
  - ex: natural number N = {1, 2, 3, …}
    natural **+** natural = natural
- Associate law: (結合律)
  - ex: (X * Y) * Z = X * (Y * Z) for all X, Y, Z in set *S*
- Commutative law: (交換律)
  - ex: X * Y = Y * X for all X, Y in set *S*

Most Common Postulates (2/2)

- Identity element: (單位元素)
  - If e is an identity element w.r.t an operator *, then
    e * X = X * e = X for every X in set *S*
  - ex: x + 0 = 0 + x = x (x is an integer)
- Inverse: (反元素)
  - If I is an inverse of x w.r.t an operator *, then
    x * I = e
  - ex: a + (-a) = 0 (0 is the identity element of +)
- Distributive law: (分配律)
  - ex: X * (Y + Z) = X * Y + X * Z (* over +)

P.S: w.r.t = with respect to

History of Boolean Algebra

- In 1854, George Boole
  - Introduced a systematic treatment of logic
  - Developed an algebraic system called ***Boolean algebra***
- In 1938, C. E. Shannon
  - Introduced a two-valued Boolean algebra called ***switching algebra***
  - The properties of bistable electrical switching circuits (digital circuits) can be represented by it
- In 1904, E. V. Huntington
  - Formulate the postulates as the formal definitions

Postulates of Boolean Algebra

Boolean algebra is

- Defined by a set of elements ***B***
- Defined with two binary operators
  - **+** (***OR***), * (***AND***)
- Satisfies the following postulates:
  - (1) Closure w.r.t the operators **+** and *
  - (2) Identity w.r.t **+** is 0; identity w.r.t * is 1
  - (3) Commutative w.r.t **+** and *
  - (4) Distributive w.r.t **+** and *
  - (5) x **+** x' (complement) = 1; x * x' = 0
  - (6) There exists at least two different elements in ***B***

Boolean v.s. Ordinary

- Boolean Algebra
  - Associate law not included (but still valid)
  - Distributive law is valid
  - No additive or multiplicative inverses
  - Define ***complement*** in postulate 5
  - No. of elements is not clearly defined
    - 2 for two-valued Boolean algebra
- Ordinary Algebra
  - Associate law included
  - Distributive law may not valid
  - Have additive and multiplicative inverses
  - No complement operator
  - Deal with real numbers
    - Infinite set of elements

Two-Valued Boolean Algebra

A two-valued Boolean algebra is

- Defined on a set of two elements B = {0, 1}
- With rules for the binary operators + and *

x	y	x*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

- Satisfying the six Huntington postulates
  - Proofs are shown in the text book
- Called "switching algebra", "***binary logic***"

Outline

- Basic Definitions
- Basic Theorems and Properties
- Boolean Functions
- Canonical and Standard Forms
- Digital Logic Gates
- Integrated Circuits

Duality Principle

- Every Boolean algebraic expression remains valid if the ***operators*** and ***identity elements*** are interchanged
  - Part (a) and part (b) are dual in the Huntington postulates
- For two-valued Boolean algebra:
  - Interchange OR and AND operators and replace 1's by 0's and 0's by 1's
  - Ex: X + 1 = 1 → X * 0 = 0

Postulates of Binary Logic

- Postulate 2:
  (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:
  (a) X + X' = 1 (b) X * X' = 0

Theorems of Binary Logic (1/6)

- Theorem 1(a)

  X + X = X

<proof>

X + X = (X + X) * 1 ..... p2(b)
= (X + X) (X + X') ... p5(a)
= X + XX' ..... p4(b)
= X + 0 ..... P5(b)
= X ..... P2(a)

- Theorem 1(b)

  X * X = X

<proof>

X * X = XX + 0 ..... p2(a)
= XX + XX' ..... p5(b)
= X (X + X') ..... p4(a)
= X * 1 ..... p5(a)
= X ..... p2(a)

Theorems of Binary Logic (2/6)

- Theorem 2(a): X + 1 = 1

<proof>

X + 1 = 1 * (X + 1) ..... p2(b)
= (X + X') (X + 1) ..... p5(a)
= X + X' * 1 ..... p4(b)
= X + X' ..... p2(b)
= 1 ..... p5(a)

- Theorem 2(b): X * 0 = 0

<proof>

by duality of theorem 2(a)

Theorems of Binary Logic (3/6)

- Theorem 3: (X')' = X (involution, 乘方)

<proof>

X + X' = 1 and X * X' = 0 (from p5)

$\Rightarrow$ X and X' are complement to each other

$\therefore$ the complement of X' = (X')' = X

Theorems of Binary Logic (4/6)

- Theorem 4: (associative)

  (a) X + (Y + Z) = (X + Y) + Z

  (b) X (YZ) = (XY) Z

<proof for (a)>

X + (Y + Z) = X*1 + (Y + Z)*1 ..... p2(b)

= X*1 + Y*1 + Z*1 ..... p4(a)

= (X + Y)*1 + Z*1 ..... p4(a)

= (X + Y) + Z ..... p2(b)

<proof for (b)>

can be obtained by the duality of theorem 4(a)

Theorems of Binary Logic (5/6)

- Theorem 5: (DeMorgan)

  (a) (X + Y)‘ = X‘Y‘ (b) (XY)‘ = X‘ + Y‘

<proof>

(a) by truth table

x	y	x+y	(x+y)′	x′	y′	x′y′
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

(b) can be proofed by similar way

Theorems of Binary Logic (6/6)

- Theorem 6: (absorption, 合併)

  (a) X + XY = X (b) X (X + Y) = X

<proof for (a)>

X + XY = X * 1 + XY …… p2(b)
= X (1 + Y) …….. p4(a)
= X (Y + 1) …….. p3(a)
= X * 1 …….. p2(a)
= X …….. p2(b)

<proof for (b)>

can be obtained by the duality of theorem 6(a)

Operator Precedence

(1) Parentheses (括號)

(2) NOT …… similar to the sign of numbers

(3) AND …… similar to multiplication

(4) OR ……. similar to addition

Ex: (X + Y)‘ + Z

step 1: X + Y …… inside the parentheses

step 2: (X + Y)‘

step 3: (X + Y)‘ + Z

Outline

- Basic Definitions
- Basic Theorems and Properties
- Boolean Functions
- Canonical and Standard Forms
- Digital Logic Gates
- Integrated Circuits

Boolean Functions

- A Boolean function is described by an algebraic expression that consists of
  - Binary variables
  - The constant 0 and 1
  - The logic operation symbols
- For a given value of the binary variables, the function can be equal to either 1 or 0
- Ex: F1 = X +Y' Z

  X = 1 → F1 = 1

  Y = 0 and Z = 1 → F1 = 1

  otherwise → F1 = 0

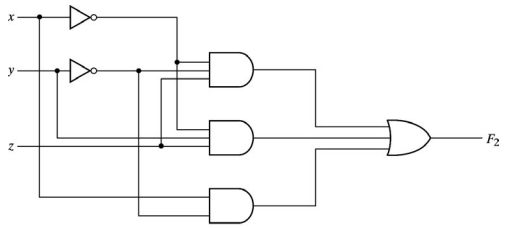
Truth Tables

- A Boolean function can be represented in a ***truth table***
  - An unique representation
- A truth table includes
  - A list of combinations of 1's and 0's assigned to the binary variables
  - A column that shows the value of the function for each binary combination
- Ex: F1 = X +Y' Z
  - No. of binary variables = 3
  - No. of rows = $2^3 = 8$

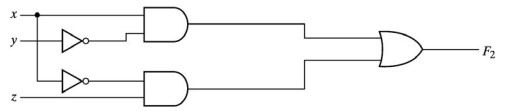
x	y	z	F1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Gate Implementation

- A Boolean function can be transformed from an algebraic expression into a circuit diagram composed of logic gates
- The forms of the algebraic expressions have large impact on the logic circuit diagram
  - Not an unique representation



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



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

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

Algebraic Manipulation

- Literal: a single variable within a term that may be complement or not
  - F2 = x'y'z + x'yz + xy' (3 terms, 8 literals)
  - F2 = xy' + x'z (2 terms, 4 literals)
- Reducing the **number of terms** and the **number of literals** can often lead to a simpler circuit
- Simplify methods:
  - By map methods described in Chapter 3 (up to 5 variables)
  - By computer minimization programs
  - By a cut-and-try procedure employing the algebraic manipulation techniques (the only manual method)

Example 2-1

- Simplify the following Boolean functions to a minimum number of literals
  1. $x(x' + y) = xx' + xy = 0 + xy = xy$
  2. $x + x'y = (x + x')(x + y) = 1(x + y) = x + y$
  3. $(x + y)(x + y') = x + xy + xy' + yy'$
     $= x(1 + y + y') + 0 = x$
  4. $xy + x'z + yz = xy + x'z + yz(x + x')$
     $= xy + x'z + xyz + x'yz$
     $= xy(1 + z) + x'z(1 + y) = xy + x'z$
  5. $(x + y)(x' + z)(y + z) = (x + y)(x' + z)$
     by duality from function 4

Complement of a Function

- Can be derived algebraically through DeMorgan's theorem
  - The basic theorem 5 for two variables
- DeMorgan's theorem can be extended to three or more variables
  - $(A + B + C + \ldots + F)' = A'B'C' \ldots F'$
  - $(ABC \ldots F)' = A' + B' + C' + \ldots + F'$
  - Proof for three variables is shown in the text book
- The complement of a function is

Example 2-2 & 2-3

- 2-2: complement by DeMorgan's theorem

  F1' = (x'yz' + x'y'z)' = (x'yz')'(x'y'z)'

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

  F2' = [x(y'z' + yz)]' = x' + (y'z' + yz)'

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

- 2-3: complement by taking duals and complementing each literal

  F1 = x'yz' + x'y'z *(dual)* → (x' + y + z')(x' + y' + z)

  (complement) → (x + y' + z)(x + y + z') = F1'

  F2 = x(y'z' + yz) *(dual)* → x + (y' + z')(y + z)

  (complement) → x' + (y + z)(y' + z') = F2'

Outline

- Basic Definitions
- Basic Theorems and Properties
- Boolean Functions
- Canonical and Standard Forms
- Digital Logic Gates
- Integrated Circuits

Minterms and Maxterms

			Minterms		Maxterms	
x	y	z	Term	Name	Term	Name
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

Minterm = standard product Maxterm = standard sum

Canonical Form

- F1 = x'y'z+xy'z'+xyz
  = $m_1+m_4+m_7$
- F1' = x'y'z'+x'yz'+x'yz+xy'z+xyz'
  → F1 = (x+y+z)(x+y'+z)(x+y'+z')
  (x'+y+z')(x'+y'+z)
  = $M_0M_2M_3M_5M_6$
- Similarly:
  F2 = x'yz+xy'z+xyz'+xyz = $m_3+m_5+m_6+m_7$
  = (x+y+z)(x+y+z')(x+y'+z)(x'+y+z) = $M_0M_1M_2M_4$
- Boolean functions expressed as a sum of minterms or product of maxterms are said to be in ***canonical form***

x	y	z	F1	F2
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

Sum of Minterms

- Each term must contain all the variables
  - If x is missing, ANDed with (x + x')
- Example 2-4:
  - Express F = A + B'C in a sum of minterms

    First term is missing two variables (B, C):

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

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

    Second term is missing one variable (A):

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

  $\Rightarrow$ F = A+B'C = ABC+ABC'+AB'C(twice)+AB'C'+A'B'C

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

Notation for Sum of Minterms

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

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

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

  - $\sum$: ORing of terms
- Can be derived directly from the truth table

A	B	C	F1	
0	0	0	0	
0	0	1	1	$\rightarrow m_1$
0	1	0	0	
0	1	1	0	
1	0	0	1	$\rightarrow m_4$
1	0	1	1	$\rightarrow m_5$
1	1	0	1	$\rightarrow m_6$
1	1	1	1	$\rightarrow m_7$

$\rightarrow \sum(1,4,5,6,7)$

Product of Maxterms

- Each term must contain all the variables
  - If x is missing, ORed with xx' and apply distributive law
- Example 2-5:
  - Express F = xy + x'z in 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+z = x+z+yy' = (x+y+z)(x+y'+z)

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

$\Rightarrow F = (x+y+z)^2(x+y'+z)(x'+y+z)^2(x'+y+z') = M_0M_2M_4M_5$

$\Rightarrow F(x, y, z) = \prod(0,2,4,5)$ $\quad \prod$: ANDing of terms

Conversion between Canonical Forms

- The complement of a function = the sum of minterms missing from the original function
  - $F(A,B,C) = \sum(1,4,5,6,7)$
  - $F'(A,B,C) = \sum(0,2,3) = m_0+m_2+m_3$
- From DeMorgan's theorem:
  - $F = (m_0+m_2+m_3)' = m_0'm_2'm_3' = M_0M_2M_3 = \prod(0,2,3)$
    - $m_j' = M_j$ are shown in Table 2-3
- To convert from one canonical from to another:
  - Interchange the symbol $\sum$ and $\prod$
  - List those numbers missing from the original form

Example 2-5 by Conversion

- F = xy + x'z = x'y'z + x'yz + xyz' + xyz

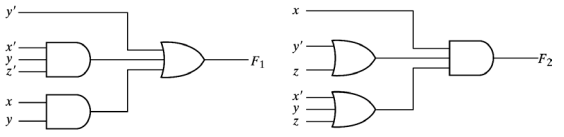
x	y	z	F	
0	0	0	0	
0	0	1	1	→ m_1
0	1	0	0	
0	1	1	1	→ m_3
1	0	0	0	
1	0	1	0	
1	1	0	1	→ m_6
1	1	1	1	→ m_7

$\Sigma(1,3,6,7)$

- The missing numbers are 0, 2, 4, 5
  - F = $\prod(0,2,4,5)$

Standard Forms

- The canonical forms are basic forms obtained from the truth table
  - Very seldom to have the least number of literals
- Standard forms : not required to have all variables in each term
  - ***Sum of products*** [ex: F1 = y' + xy + x'yz']
  - ***Product of sums*** [ex: F2 = x(y' + z)(x' + y + z')]
- Results in a two-level gating structure

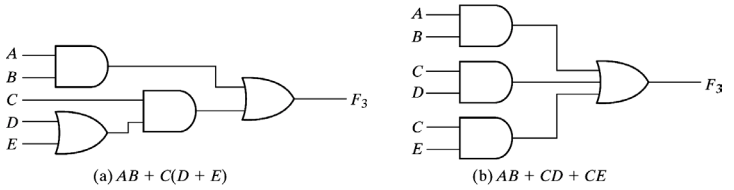


(a) Sum of Products (b) Product of Sums

Fig. 2-3 Two-level implementation

Non-Standard Forms

- Neither in sum of products nor in product of sums
  - F3 = AB + C(D+E) …… non-standard form
  - F3 = AB + CD + CE …….. standard form
- Results in a multi-level gating structure
- In general, two-level implementations are preferred
  - Produce the least amount of delay from inputs to outputs



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

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

Outline

- Basic Definitions
- Basic Theorems and Properties
- Boolean Functions
- Canonical and Standard Forms
- Digital Logic Gates
- Integrated Circuits

Functions of Two Variables

- There are 2^{2^n} functions for n binary variables
  - For 2 variables, there are 16 functions as shown below
- We can assign special operator symbols for each function
  - They can still be expressed by AND, OR, NOT
  - Except the exclusive-OR ($\oplus$), those new symbol are not commonly used by digital designers

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

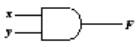
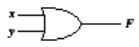
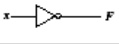
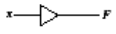
Table 2-8

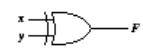
Boolean functions	Operator symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	x * y	AND	x and y
$F_2 = xy'$	x / y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y / x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not both
$F_7 = x + y$	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

- Considerations for constructing other logic gates:
  - The feasibility and economy of producing the gate with physical components
  - The possibility of extending to more than two inputs
  - The basic properties (commutativity, associativity, …)
  - The ability to implement Boolean functions
- In the 16 functions defined in Table 2-8:
  - Two are equal to a constant and four are repeated twice
  - Two (inhibition and implication) are not commutative or associative
  - The other eight are used as standard gates

Standard Gates

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

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

*Bubble: inversion Triangle: transfer

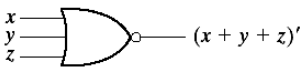
- NAND and NOR gates are more popular than AND and OR gates because they are more easily constructed with transistor circuits

Extension to Multiple Inputs (1/3)

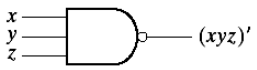
- A gate can be extended to have multiple inputs if the binary operation it represents is commutative and associative
  - The standard gates, except the inverter and buffer, can be extended to have more than two inputs
- The AND and OR operations posses these two properties
  - X+Y = Y+X (commutative)
  - (X+Y)+Z = X+(Y+Z) = X+Y+Z (associative)
  - Can be extended to more than two inputs

Extension to Multiple Inputs (2/3)

- The NAND and NOR functions are commutative but not associative
  - $(x \downarrow y) \downarrow z = [(x + y)' + z]' = (x + y)z' = xz' + yz'$
  - $x \downarrow (y \downarrow z) = [x + (y + z)']' = x'(y + z) = x'y + x'z$
- Therefore, the multiple-input NOR (or NAND) gate as a complemented OR (or AND) gate
  - $x \downarrow y \downarrow z = (x+y+z)'$
  - $x \uparrow y \uparrow z = (xyz)'$



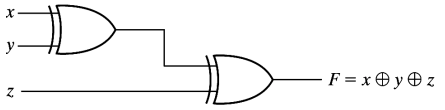
(a) 3-input NOR gate



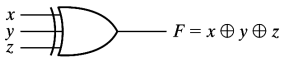
(b) 3-input NAND gate

Extension to Multiple Inputs (3/3)

- Exclusive-OR (XOR) and equivalence (XNOR) gates are both commutative and associative
  - Can be extended to more than two inputs
- Multiple-input XOR gates are very uncommon
  - Usually constructed with other gates for easier implementation



(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

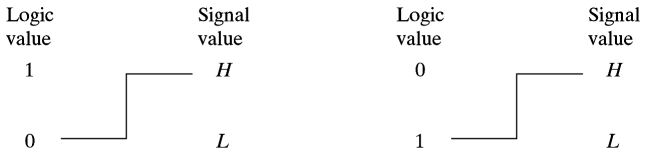
(c) Truth table

"Odd" function

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

Logic Polarity

- Positive logic system:
  Choose the high-level H as logic 1
  Choose the low-level L as logic 0
- Negative logic system:
  Choose the high-level H as logic 0
  Choose the low-level L as logic 1



(a) Positive logic (b) Negative logic

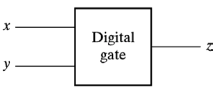
Fig. 2-9 signal assignment and logic polarity

Positive and Negative Logic

- The small triangles in the inputs and output designate a *polarity indicator*
- The same physical gate can operate either as a positive logic AND gate or as a negative logic OR gate
- To convert between them:
  - Interchange 1's and 0's
  - Take the dual function

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

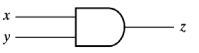
(a) Truth table with H and L



(b) Gate block diagram

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

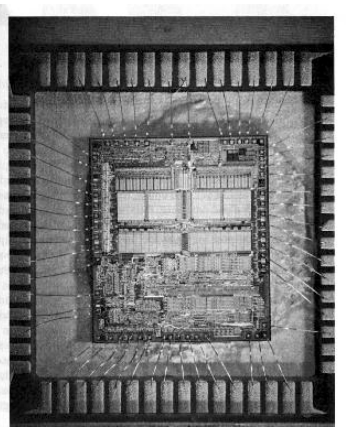
Fig. 2-10 Demonstration of positive and negative logic

Outline

- Basic Definitions
- Basic Theorems and Properties
- Boolean Functions
- Canonical and Standard Forms
- Digital Logic Gates
- Integrated Circuits

Integrated Circuits

- An integrated circuit (**IC**) is a silicon semiconductor crystal, called a ***chip***, containing the electronic components for constructing digital gates
- The chip is mounted in a ceramic or plastic container
- Connections are welded to external pins from the chip
- No. of pins may range from 14 to several thousands



Level of Integration

- Small-scale integration (SSI):
  - Contain several independent gates in a package
  - The I/O of the gates are connected directly to external pins
- Medium-scale integration (MSI):
  - Approximate 10 to 1000 gates in a package
  - Usually perform specific elementary operations (adder, …)
- Large-scale integration (LSI):
  - Contain thousands of gates within a package
  - Include digital systems (processors, memory, …)
- Very large-scale integration (VLSI):
  - Contain hundred of thousands of gates within a package
- Ultra large-scale integration (ULSI), …

Digital Logic Families

- Logic families: classified by the specific circuit technology for implementing the logic
- Many logic families have been used commercially
  - TTL (transistor-transistor logic):
    - Standard logic family
  - ECL (emitter-coupled logic):
    - For high-speed operations
  - MOS (metal-oxide semiconductor):
    - High component density
  - CMOS (complementary MOS):
    - Low power consumption
    - The dominant logic family in VLSI design

Important Parameters for ICs

- Fan-out
  - No. of standard loads that the output can drive without impairing its normal operation
- Fan-in
  - No. of inputs available in a gate
- Power dissipation
  - The power consumed by the gate
- Propagation delay
  - The average transition delay time for the signal to propagate from input to output
- Noise margin
  - The maximum external noise voltage that does not cause an undesirable change in the circuit output

Computer-Aided Design (CAD)

- VLSI circuits contain million of transistors
  - Impossible to develop and verify without the assistance of **CAD** tools
- Electronic design automation (**EDA**) covers all phase of the design of integrated circuits
- An important development is the use of a ***hardware description language*** (HDL)
  - Describe the circuits by a formal language
  - Can be used to simulate the system before its construction to check the functionality
  - Translated to real circuits automatically by ***logic synthesis*** tools