Boolean Algebra and Logic Gates

By:

Ratnakar Dash

August 17, 2020

Table of Contents

Boolean Algebra

- Boolean algebra, like any other deductive mathematical system, may be defined with a set of elements, a set of operators, and a number of unproved axioms or postulates.
- The most common postulates used to formulate various algebraic structures are:
  - Closure.
  - Associative law.
  - Commutative law.
  - Inverse.
  - Distributive law.

Boolean Algebra (contd.)

Closure

- For example, the set of natural numbers N = 1, 2, 3, 4, c is closed with respect to the binary operator + by the rules of arithmetic addition, since, for any a, b $\in$ N, there is a unique c $\in$ N such that a + b = c.
- The set of natural numbers is not closed with respect to the binary operator − by the rules of arithmetic subtraction, because 2 - 3 = -1 and 2, 3 $\in$ N, but (-1) $\notin$ N.

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

Boolean Algebra (contd.)

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

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

Boolean Algebra (contd.)

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
- $x * y = e$

Distributive law.

- If $*$ and . are two binary operators on a set S, $*$ is said to be distributive over . whenever
- $x * (y.z) = (x * y).(x * z)$

Boolean Algebra (contd.)

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:
  - Closure with respect to the operators + and ..
  - An identity element with respect to + and . is 0 and 1,respectively.
  - Commutative with respect to + and . Ex: $x + y = y + x$
  - + is distributive over . : $x + (y.z) = (x + y).(x + z)$
    . is distributive over + : $x.(y + z) = (x.y) + (x.z)$
  - Complement elements: $x + x' = 1 and x.x' = 0$.
  - There exists at least two elements x,y $\in$ B such that x$\neq$y.

Difference between Boolean Algebra and ordinary Algebra

- Huntington postulates do not include the associative law. However, this law holds for Boolean algebra and can be derived (for both operators) from the other postulates.
- The distributive law of + over . (i.e., x + (y . z) = (x + y) . (x + z)) is valid for Boolean algebra, but not for ordinary algebra.
- Boolean algebra does not have additive or multiplicative inverses; therefore, there are no subtraction or division operations.
- Postulate 5 defines an operator called the complement that is not available in ordinary algebra.
- Ordinary algebra deals with the real numbers, which constitute an infinite set of elements. Boolean algebra deals with the as yet undefined set of elements, B, but in the two-valued Boolean algebra defined next (and of interest in our subsequent use of that algebra), B is defined as a set with only two elements, 0 and 1.

Two-Valued Boolean Algebra

- A two-valued Boolean algebra is defined on a set of two elements, B = 0, 1, with rules for the two binary operators + and . as shown in the following operator tables

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

Two-Valued Boolean Algebra (contd.)

- These rules are exactly the same as the AND, OR, and NOT operations, respectively.We must now show that the Huntington postulates are valid for the set B = 0, 1 and the two binary operators + and . :
  - That the structure is closed with respect to the two operators is obvious from the tables, since the result of each operation is either 1 or 0 and 1, 0 $\in$ B.
  - From the tables, we see that
    (a) 0 + 0 =0 , 0+ 1 = 1 + 0 = 1
    (b) 1 . 1 = 1 , 1. 0 = 0 . 1 = 0.
  - The commutative laws are obvious from the symmetry of the binary operator tables.

Two-Valued Boolean Algebra (contd.)

Distributive Law

x	y	z	y + z	x · (y + z)	x · y	x · z	(x · y) + (x · 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

- From the complement table, it is easily shown that (a) x + x' = 1, since 0 + 0' = 0 + 1 = 1 and 1 + 1' = 1 + 0 = 1.
  (b) x . x' = 0, since 0 . 0' = 0 . 1 = 0 and 1 . 1' = 1 . 0 = 0.
- Postulate 6 is satisfied because the two-valued Boolean algebra has two elements, 1 and 0, with $1 \neq 0$.

Basic Theorems and Properties of Boolean Algebra

Duality

Huntington postulates are listed in pairs and one part can be obtained from other if binary operators and the identity operators are interchanged. This important property is called duality principle. It states that every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged. The dual of an expression can be obtained by interchanging OR and AND operators and replace 1's by 0's and 0's by 1's.

Basic Theorems and Properties of Boolean Algebra (contd.)

Postulates and Theorems

Postulates and Theorems of Boolean Algebra

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

Two-Valued Boolean Algebra (contd.)

Postulates and 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)

Two-Valued Boolean Algebra (contd.)

Postulates and 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)

Two-Valued Boolean Algebra (contd.)

Postulates and 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.

Two-Valued Boolean Algebra (contd.)

Postulates and Theorems

THEOREM 3: $(x')' = x$. From postulate 5, we have $x + x' = 1$ and $x \cdot x' = 0$, which together define the complement of x. The complement of x' is x and is also $(x')'$.

Since, The complement is unique, we have (x')' = x

Two-Valued Boolean Algebra (contd.)

Postulates and 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.

Two-Valued Boolean Algebra (contd.)

Postulates and Theorems

The following truth table verifies the first absorption theorem:

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

Two-Valued Boolean Algebra (contd.)

Postulates and Theorems

The truth table for the first DeMorgan's theorem, $(x + y)' = x'.y'$, is as follows:

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

For Boolean Expression Evaluation, the operator precedence are

1. Complement
2. AND
3. OR

Expressions inside parentheses must be evaluated first before other operations.

Boolean functions

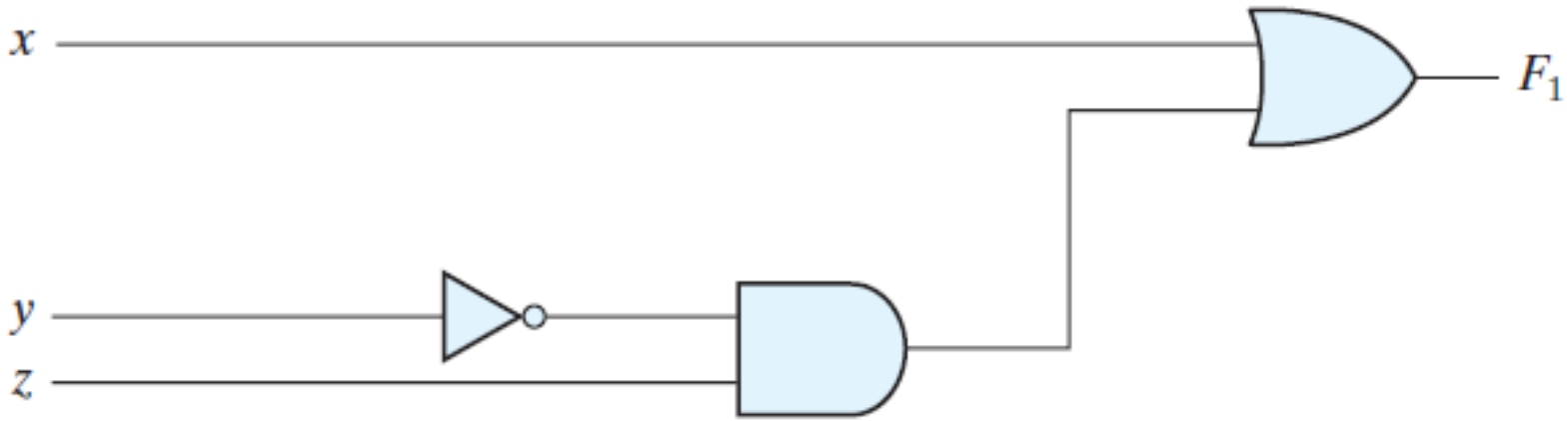
- A Boolean function described by an algebraic expression consists of binary variables, the constants 0 and 1, and the logic operation symbols. For a given value of the binary variables, the function can be equal to either 1 or 0.
- As an example, consider the Boolean function.
  $F_1 = x + y'z$
  $F_2 = x'y'z + x'yz + xy'$
- The function F1 is equal to 1 if x is equal to 1 or if both y' and z are equal to 1. F1 is equal to 0 otherwise.

Truth Tables for F_1 and F_2

x	y	z	F_1	F_2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0

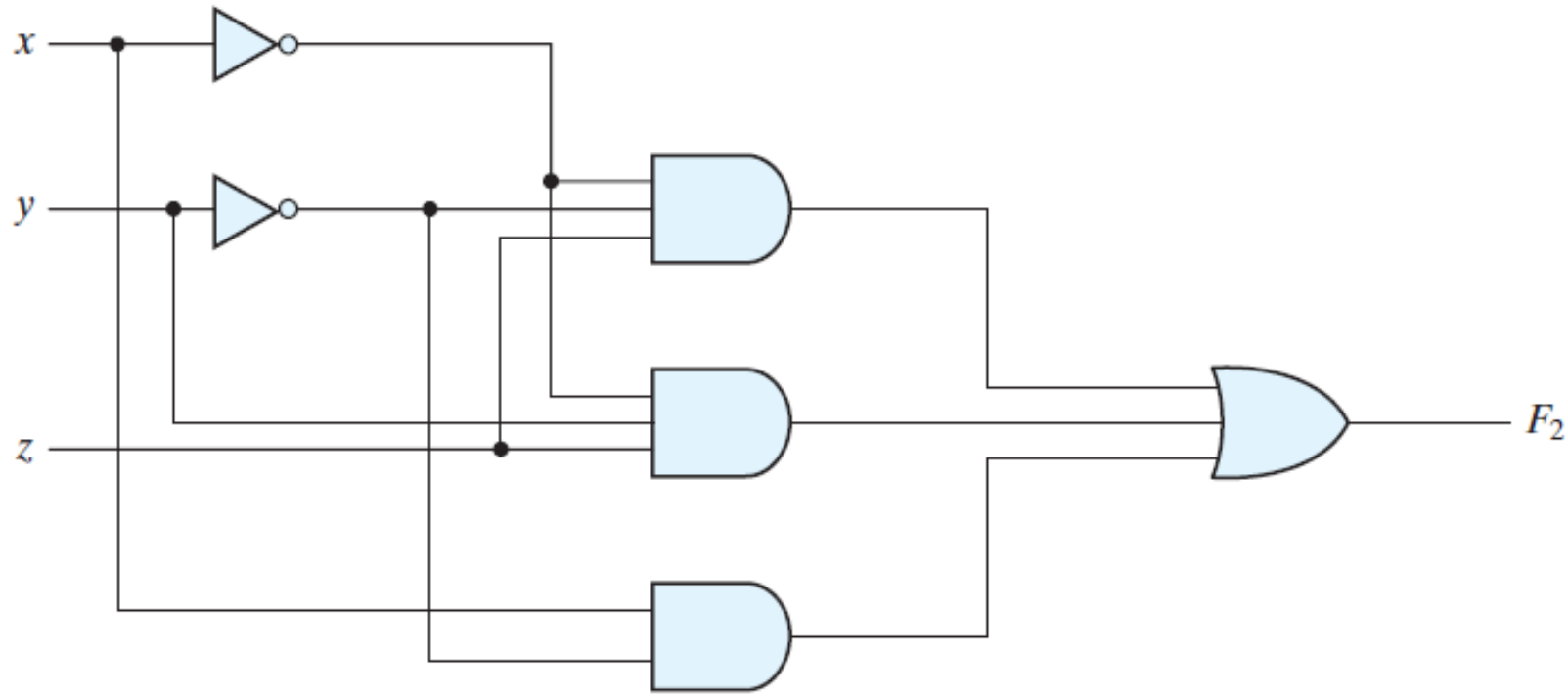
Boolean functions

Gate implementation of $F_1 = x + y'z$:



Boolean functions

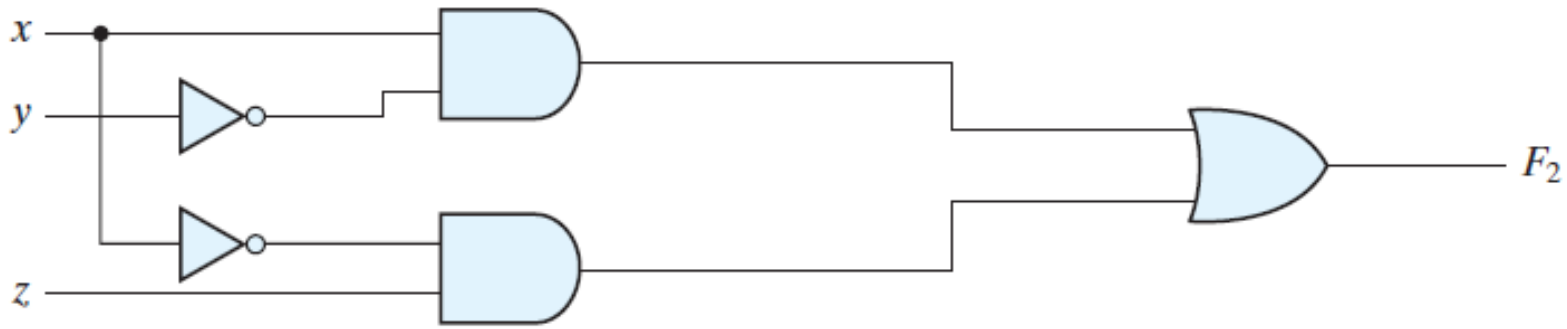
Gate implementation of F_2 :



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

Boolean functions

Gate implementation of F_2 :



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

Boolean functions (contd.)

Complement of a Function

- The complement of a function F is F' and is obtained from an interchange of 0's for 1's and 1's for 0's in the value of F.
- The complement of a function may be derived algebraically through DeMorgan's theorems

$$
\begin{aligned}
(A + B + C)' &= (A + x)' && \text{let } B + C = x \\
&= A'x' && \text{by theorem 5(a) (DeMorgan)} \\
&= A'(B + C)' && \text{substitute } B + C = x \\
&= A'(B'C') && \text{by theorem 5(a) (DeMorgan)} \\
&= A'B'C' && \text{by theorem 4(b) (associative)}
\end{aligned}
$$

Boolean functions (contd.)

Example-1

Find the complement of the functions $F_1 = x'yz' + x'y'z$ and $F_2 = x(y'z' + yz)$. By applying DeMorgan's theorems as many times as necessary, the complements are obtained as follows:

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

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

Boolean functions (contd.)

Example-1

Find the complement of the functions F_1 and F_2 of Example 2.2 by taking their duals and complementing each literal.

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

Minterms and Maxterms

- Two binary variables x and y combined with an AND operation. Since each variable may appear in either form, there are four possible combinations: x'y', x'y, xy', and xy. Each of these four AND terms is called a minterm, or a standard product.
- In a similar fashion, n variables forming an OR term, with each variable being primed or unprimed, provide 2^n possible combinations, called maxterms, or standard sums.
- A Boolean function can be expressed algebraically from a given truth table by forming a minterm for each combination of the variables that produces a 1 in the function and then taking the OR of all those terms.

Minterms and Maxterms(contd.)

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(contd.)

Example

- f1 = x'y'z + xy'z' + xyz = m1 + m4 + m7
- f2 = x'yz + xy'z + xyz' + xyz = m3 + m5 + m6 + m7

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

Sum of Minterms

- The minterms whose sum defines the Boolean function are those which give the 1's of the function in a truth table.
- Example

Express the Boolean function $F = A + B'C$ as a sum of minterms. The function has three variables: A, B, and C. The first term A is missing two variables; therefore,

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

This function is still missing one variable, so

$$\begin{aligned} A &= AB(C + C') + AB'(C + C') \\ &= ABC + ABC' + AB'C + AB'C' \end{aligned}$$

The second term $B'C$ is missing one variable; hence,

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

Combining all terms, we have

$$\begin{aligned} F &= A + B'C \\ &= ABC + ABC' + AB'C + AB'C' + A'B'C \end{aligned}$$

But $AB'C$ appears twice, and according to theorem 1 ($x + x = x$), it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

$$\begin{aligned} F &= A'B'C + AB'C + AB'C + ABC' + ABC \\ &= m_1 + m_4 + m_5 + m_6 + m_7 \end{aligned}$$

Minterms and Maxterms(contd.)

Truth Table

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

Minterms and Maxterms(contd.)

Product of Maxterms

- Each of the 2^{2n} functions of n binary variables can be also expressed as a product of maxterms.
- To express a Boolean function as a product of maxterms, it must first be brought into a form of OR terms.
- Example

Express the Boolean function $F = xy + x'z$ as a product of maxterms. First, convert the function into OR terms by using the distributive law:

$$\begin{aligned} F &= xy + x'z = (xy + x')(xy + z) \\ &= (x + x')(y + x')(x + z)(y + z) \\ &= (x' + y)(x + z)(y + z) \end{aligned}$$

The function has three variables: x, y, and z. Each OR term is missing one variable; therefore,

$$\begin{aligned} x' + y &= x' + y + zz' = (x' + y + z)(x' + y + z') \\ x + z &= x + z + yy' = (x + y + z)(x + y' + z) \\ y + z &= y + z + xx' = (x + y + z)(x' + y + z) \end{aligned}$$

Minterms and Maxterms(contd.)

Product of Maxterms

- Example

Combining all the terms and removing those which appear more than once, we finally obtain

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

A convenient way to express this function is as follows:

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

The product symbol, Π, denotes the ANDing of maxterms; the numbers are the indices of the maxterms of the function.

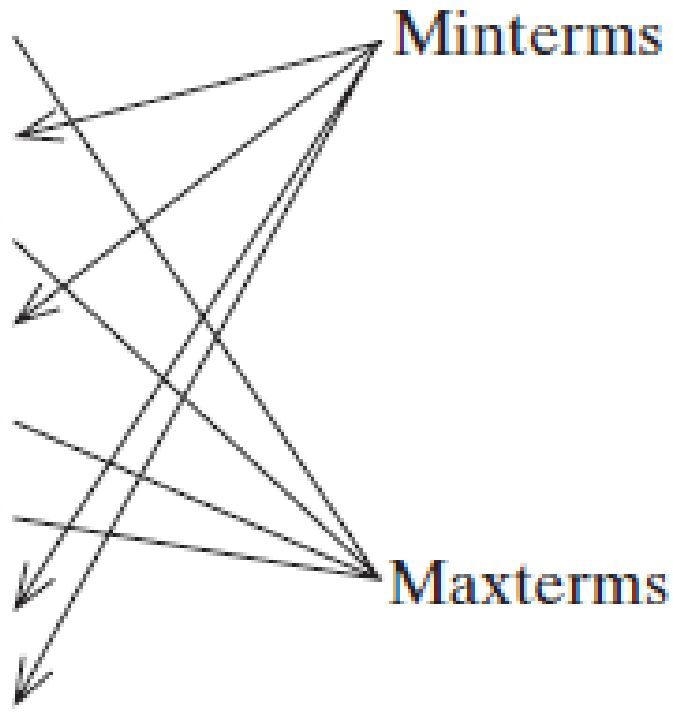
Conversion between Canonical Forms

- A Boolean function can be converted from an algebraic expression to a product of maxterms by means of a truth table and the canonical conversion procedure.
- Example
- The Boolean expression F = xy + x'z
- xy = 11 or xz = 01
- sum of minterms is F(x, y, z) = Σ(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) = Π(0, 2, 4, 5)

Conversion between Canonical Forms

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

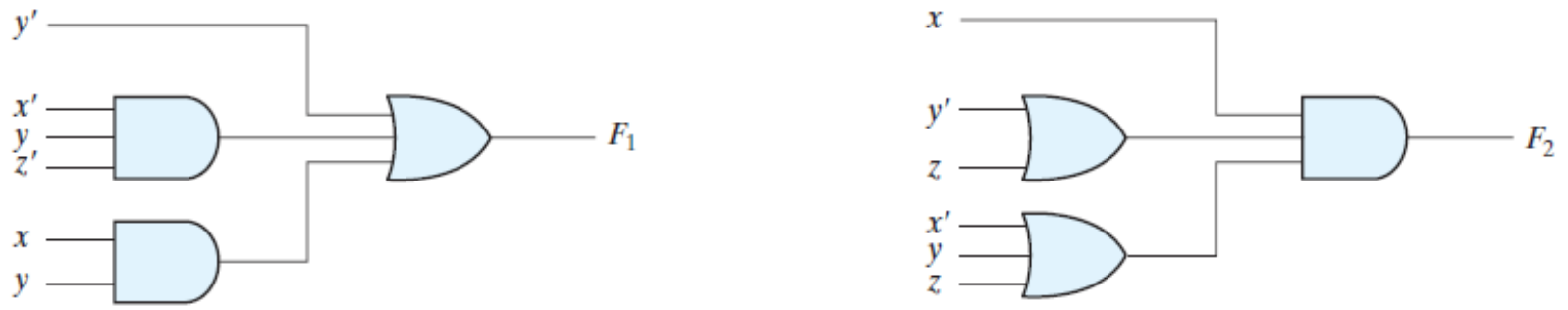
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Standard Forms

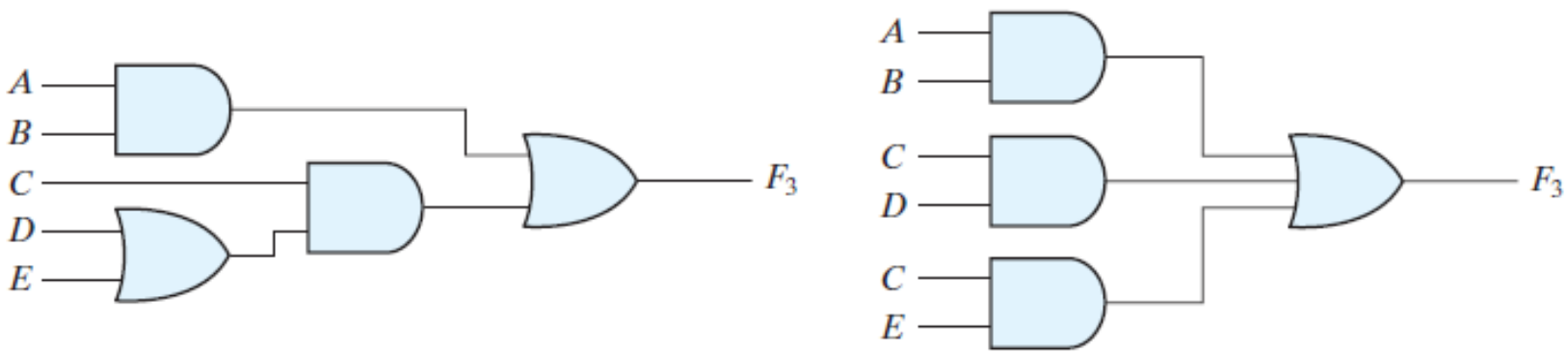
- Another way to express Boolean functions is in standard form.
- The sum of products is a Boolean expression containing AND terms, called product terms, with one or more literals each. The sum denotes the ORing of these terms.
- Sum of products(SOP): $F_1 = y' + xy + x'yz'$
- Product of sums(POS): $F_2 = x(y' + z)(x' + y + z')$

- Two-level implementation



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

- 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.
- Three-level implementation



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

Other logic operations

- There are 2^{2n} functions for n binary variables, for two variables, n=2, and the possible Boolean functions is 16.
- The 16 functions listed can be subdivided into three categories:
  - Two functions that produce a constant 0 or 1.
  - Four functions with unary operations: complement and transfer.
  - Ten functions with binary operators that define eight different operations: AND, OR, NAND, NOR, exclusive-OR, equivalence, inhibition, and implication.

Other logic operations

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

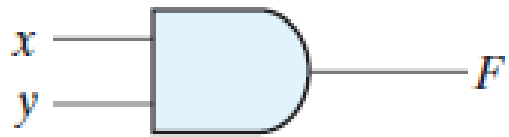
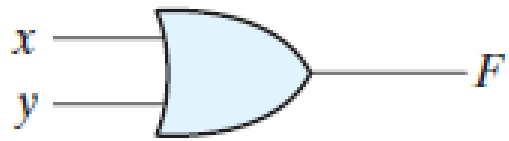
Other logic operations

Boolean Expressions for the 16 Functions of Two Variables

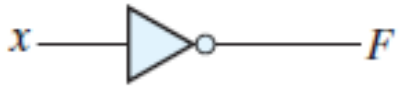
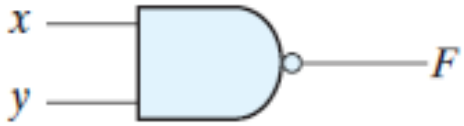
Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not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

Digital Logic gates

- The graphic symbols and truth tables of the gates of the eight different operations are:

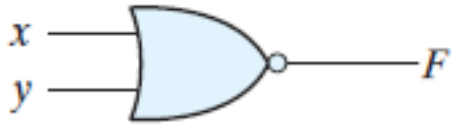
Name	Graphic symbol	Algebraic function	Truth table
AND		$F = x \cdot y$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1

Digital Logic gates

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

Digital Logic gates

NOR



$F = (x + y)'$

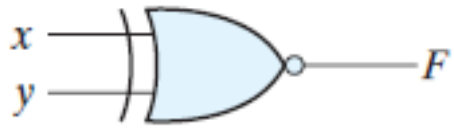
x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

Exclusive-OR (XOR)

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

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

Exclusive-NOR or equivalence

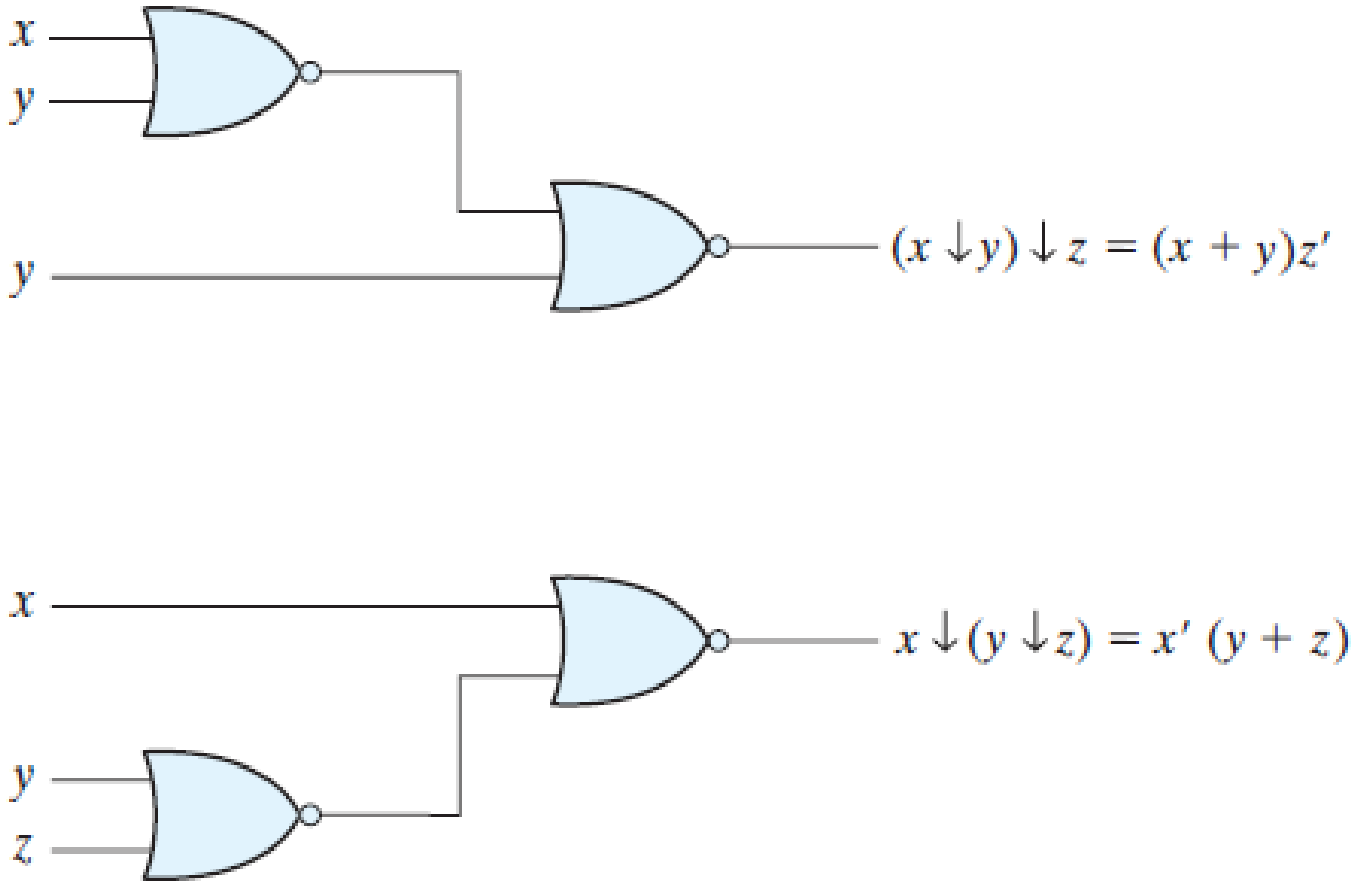


$F = xy + x'y'$
$= (x \oplus y)'$

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

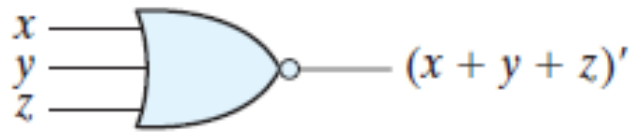
Extension to Multiple Inputs

- Demonstrating the non associativity of the NOR operator:$(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$.

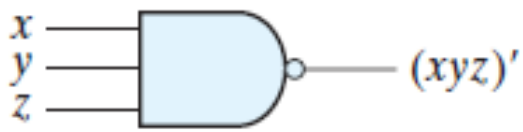


Cascaded of NAND gates

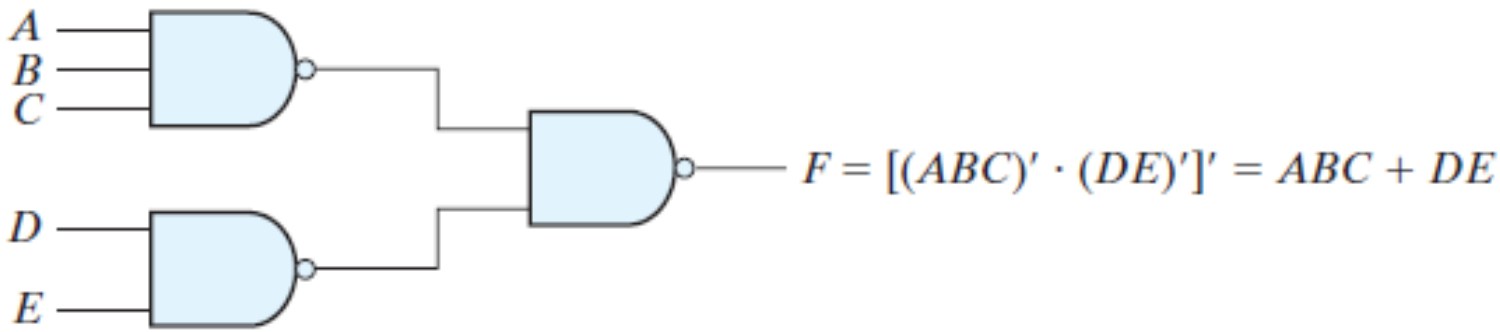
- In writing cascaded NOR and NAND operations, one must use the correct parentheses to signify the proper sequence of the gates.



(a) 3-input NOR gate



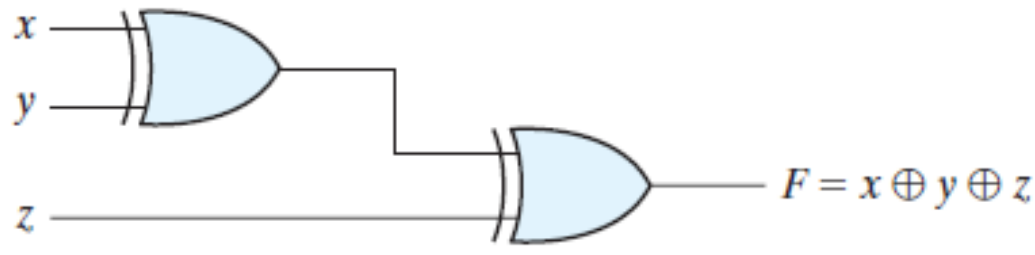
(b) 3-input NAND gate



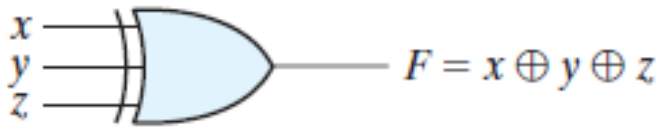
(c) Cascaded NAND gates

properties of X-OR gates

- The exclusive-OR and equivalence gates are both commutative and associative and can be extended to more than two inputs.
- Exclusive-OR is an odd function (i.e., it is equal to 1 if the input variables have an odd number of 1's).



(a) Using 2-input gates



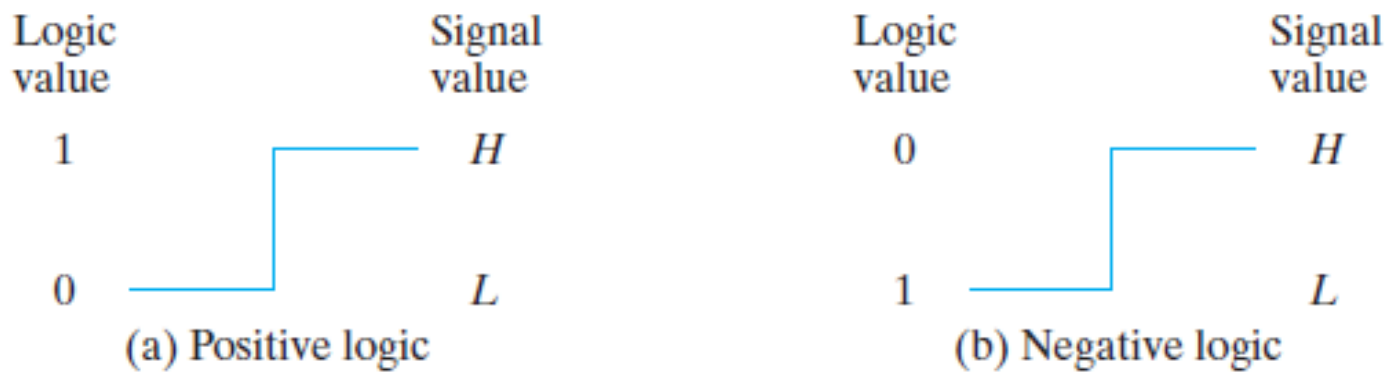
(b) 3-input gate

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

Positive and Negative Logic

- The binary signal at the inputs and outputs of any gate has one of two values, except during transition.
- One signal value represents logic 1 and the other logic 0.
- The higher signal level is designated by H and the lower signal level by L. Choosing the high-level H to represent logic 1 defines a positive logic system. Choosing the low-level L to represent logic 1 defines a negative logic system.



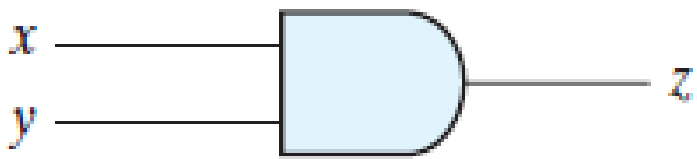
Positive and Negative Logic

Positive Logic

- Hardware digital gates are defined in terms of signal values such as H and L. It is up to the user to decide on a positive or negative logic polarity.
- Assumes a positive logic assignment, with H = 1 and L = 0. This truth table is the same as the one for the AND operation. The graphic symbol for a positive logic AND gate is shown here:

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

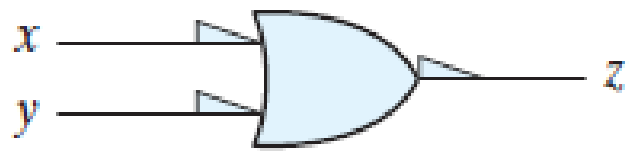
Positive and Negative Logic

Negative Logic

- 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 in both the inputs and the output of a gate.
- Since this operation produces the dual of a function, the change of all terminals from one polarity to the other results in taking the dual of the function.

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

IC

- An integrated circuit (IC) is fabricated on a die of a silicon semiconductor crystal, called a chip, containing the electronic components for constructing digital gates.
- The various gates are interconnected inside the chip to form the required circuit.
- The chip is mounted in a ceramic or plastic container, and connections are welded to external pins to form the integrated circuit.
- The number of pins may range from 14 on a small IC package to several thousand on a larger package.
- Each IC has a numeric designation printed on the surface of the package for identification.

Levels of Integration

- Small-scale integration (SSI)
  - It contains several independent gates in a single package.
  - The inputs and outputs of the gates are connected directly to the pins in the package.
  - The number of gates is usually fewer than 10 and is limited by the number of pins available in the IC.
- Medium-scale integration (MSI)
  - Medium-scale integration (MSI) devices have a complexity of approximately 10 to 1,000 gates in a single package.
  - They usually perform specific elementary digital operations.

Levels of Integration

- Large-scale integration (LSI)
  - Large-scale integration (LSI) devices contain thousands of gates in a single package.
  - They include digital systems such as processors, memory chips, and programmable logic devices.
- Very large-scale integration (VLSI)
  - Very large-scale integration (VLSI) devices now contain millions of gates within a single package.
  - Examples are large memory arrays and complex microcomputer chips. Because of their small size and low cost, VLSI devices have revolutionized the computer system design technology, giving the designer the capability to create structures that were previously uneconomical to build.

Digital Logic Families

- Digital integrated circuits are classified not only by their complexity or logical operation, but also by the specific circuit technology to which they belong. The circuit technology is referred to as a digital logic family.
- Each logic family has its own basic electronic circuit upon which more complex digital circuits and components are developed.
- The basic circuit in each technology is a NAND, NOR, or inverter gate.
- Many different logic families of digital integrated circuits have been introduced commercially. The following are the most popular:
  - TTL (transistor–transistor logic)
  - ECL (emitter-coupled logi)
  - MOS (metal-oxide semiconductor)
  - CMOS (complementary metal-oxide semiconductor)

Digital Logic Families

- TTL (transistor–transistor logic) is a logic family that has been in use for 50 years and is considered to be standard.
- The ECL has an advantage in systems requiring high-speed operation.
- The MOS is suitable for circuits that need high component density.
- The CMOS is preferable in systems requiring low power consumption, such as digital cameras, personal media players, and other hand-held portable devices.
- Low power consumption is essential for VLSI design, therefore, CMOS has become the dominant logic family, while TTL and ECL continue to decline in use.

Digital Logic Families

Fan-out and Fan-in

- Fan-out specifies the number of standard loads that the output of a typical gate can drive without impairing its normal operation. A standard load is usually defined as the amount of current needed by an input of another similar gate in the same family.
- Fan-in is the number of inputs available in a gate.

Digital Logic Families

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 a signal to propagate from input to output. For example, if the input of an inverter switches from 0 to 1, the output will switch from 1 to 0, but after a time determined by the propagation delay of the device. The operating speed is inversely proportional to the propagation delay.
- 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 of VLSI Circuits

- The design of digital systems with VLSI circuits containing millions of transistors and gates is an enormous and formidable task.
- Systems of this complexity are usually impossible to develop and verify without the assistance of computer-aided design (CAD) tools, which consist of software programs that support computer-based representations of circuits and aid in the development of digital hardware by automating the design process.
- There are a variety of options available for creating the physical realization of a digital circuit in silicon.
- The designer can choose between an application-specific integrated circuit (ASIC), a field-programmable gate array (FPGA), a programmable logic device (PLD), and a full-custom IC.
- An important development in the design of digital systems is the use of a hardware description language (HDL).

Computer-Aided Design of VLSI Circuits

- HDL-based models of a circuit or system are simulated to check and verify its functionality before it is submitted to fabrication, thereby reducing the risk and waste of manufacturing a circuit that fails to operate correctly.
- In tandem with the emergence of HDL-based design languages, tools have been developed to automatically and optimally synthesize the logic described by an HDL model of a circuit.
- These two advances in technology have led to an almost total reliance by industry on HDL-based synthesis tools and methodologies for the design of the circuits of complex digital systems.
- Two HDLs—Verilog and VHDL—have been approved as standards by the Institute of Electronics and Electrical Engineers (IEEE) and are in use by design teams worldwide.

References I

[1] M. M. Mano and M. D. Ciletti, *Digital Design (4th Edition)*. USA: Prentice-Hall, Inc., 2006.


Thank
you!