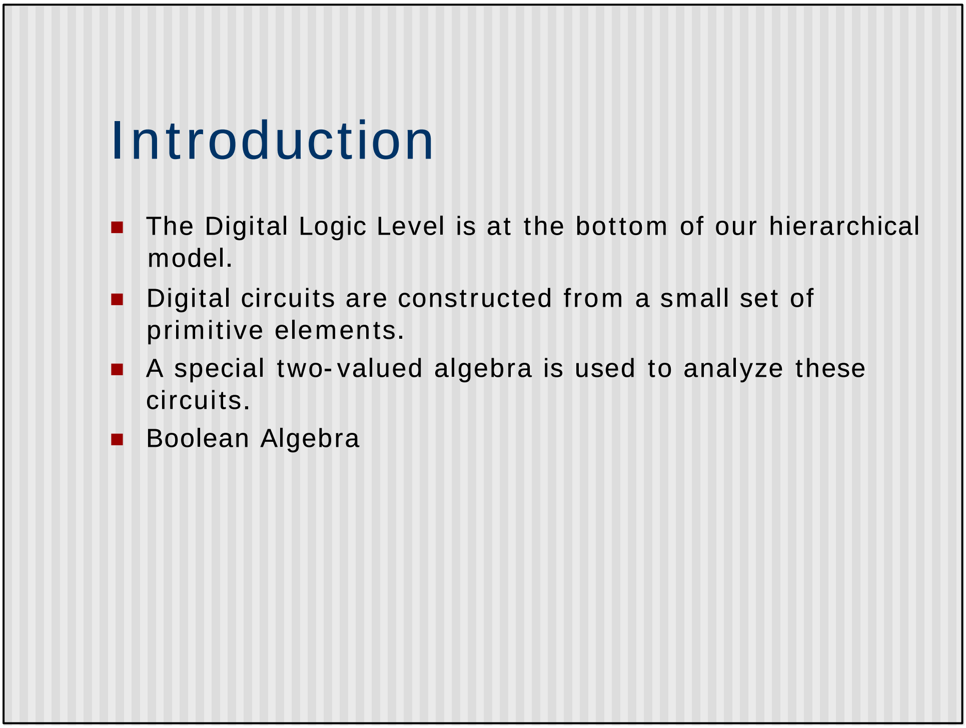
Lecture 2

Combinational Logic Circuits

Chapter 2

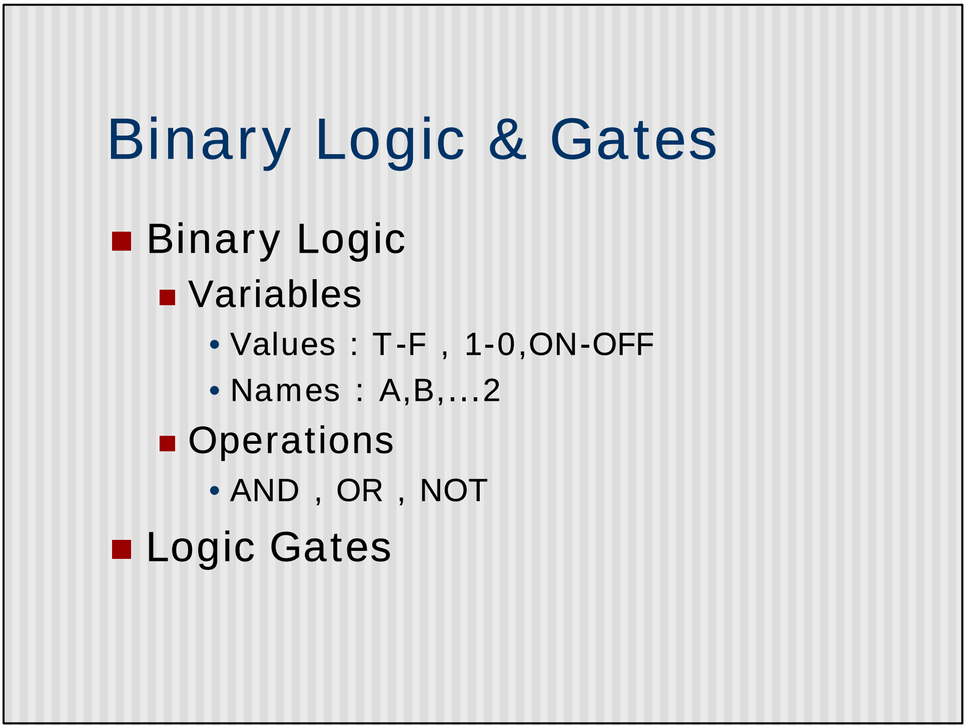
The Sum of Products Notation

- Binary Logic and Gates
- Boolean Algebra
- Standard Forms
- Implementations of Boolean Functions
- Using only NAND and NOR Gates
- Circuit Equivalence
- **Reading:** Chapter 2



Introduction

- The Digital Logic Level is at the bottom of our hierarchical model.
- Digital circuits are constructed from a small set of primitive elements.
- A special two-valued algebra is used to analyze these circuits.
- Boolean Algebra

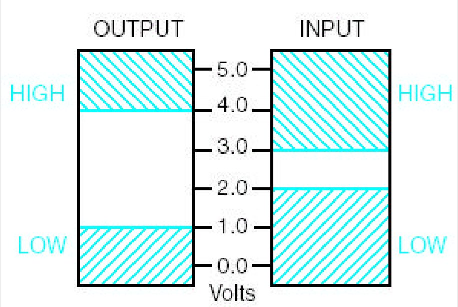


Binary Logic & Gates

- Binary Logic
  - Variables
    - Values : T-F , 1-0,ON-OFF
    - Names : A,B,...2
  - Operations
    - AND , OR , NOT
- Logic Gates

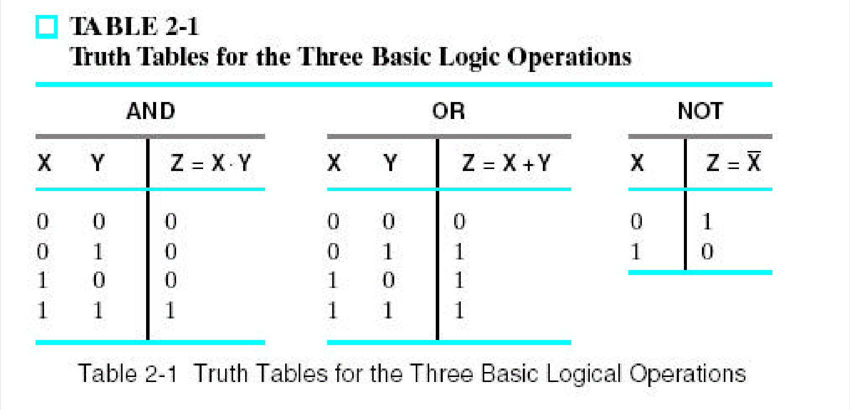
Gates

- A digital circuit is one in which only two logical values are present - 0 and 1.
- Typically, a signal between 0-1 volts represents a binary 0, a signal between 2-5 volts represents a binary 1.



- Tiny electronic devices, called gates, can compute various functions over these two-valued signals.
- These gates form the hardware basis on which all digital computers are built.

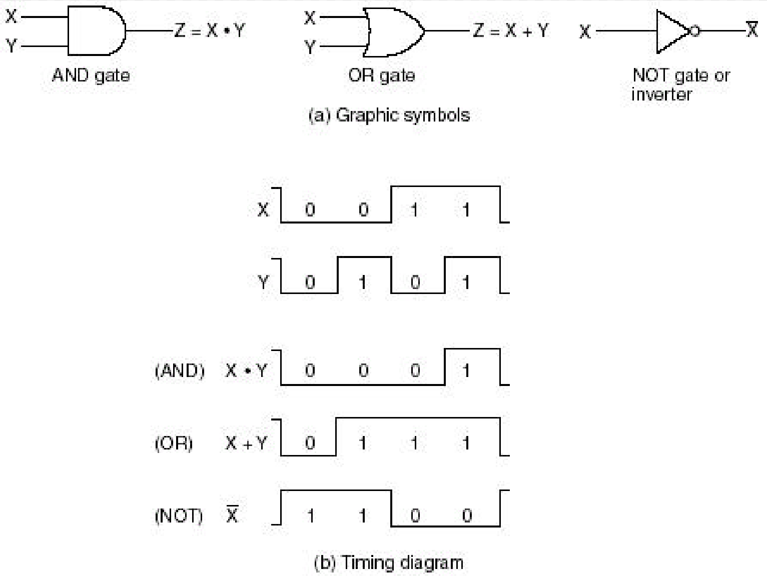
Simple Gates



□ **TABLE 2-1**
Truth Tables for the Three Basic Logic Operations

AND			OR			NOT	
X	Y	$Z = X \cdot Y$	X	Y	$Z = X + Y$	X	$Z = \overline{X}$
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

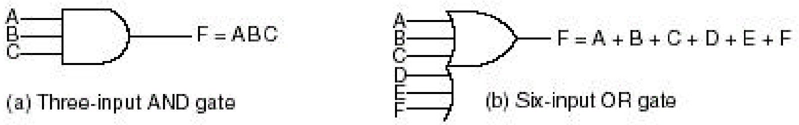
Table 2-1 Truth Tables for the Three Basic Logical Operations



(a) Graphic symbols

(b) Timing diagram

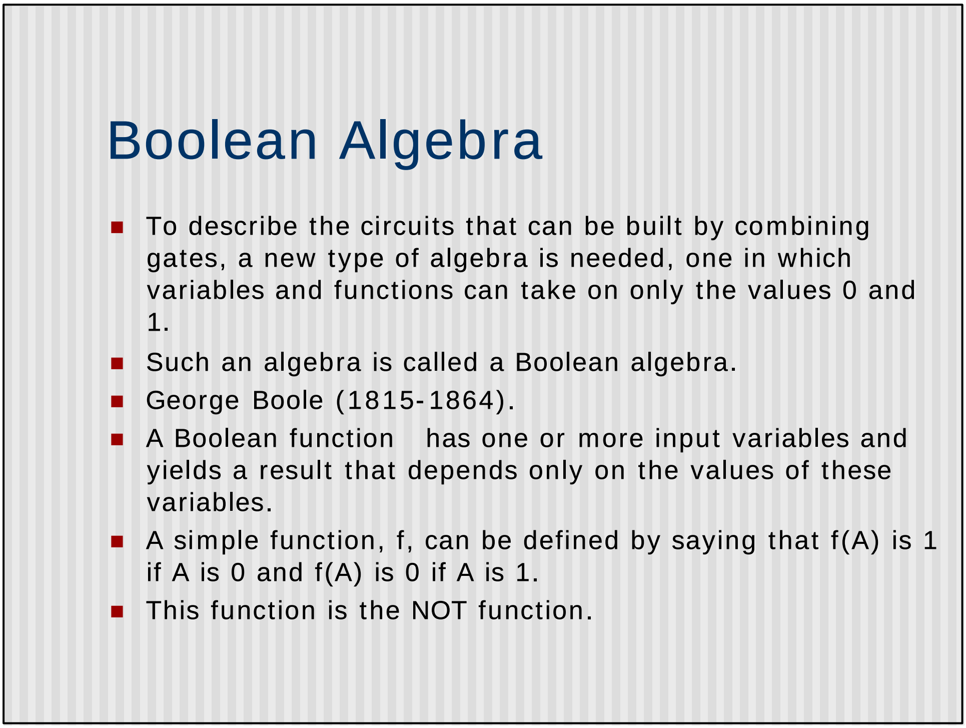
Fig. 2-1 Digital Logic Gates



(a) Three-input AND gate

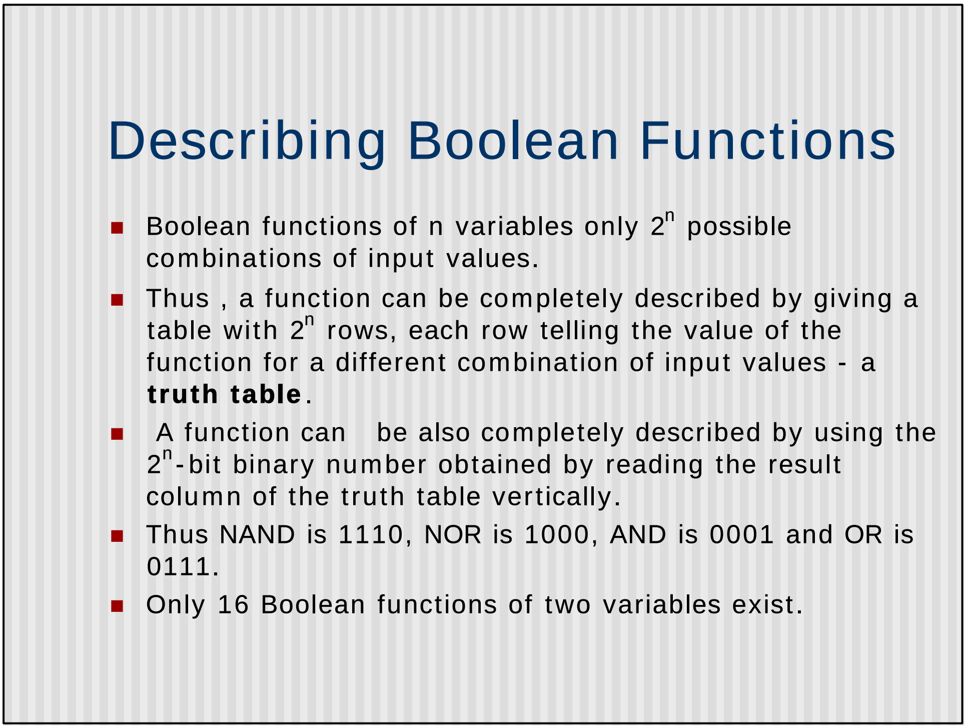
(b) Six-input OR gate

Fig. 2-2 Gates with More than Two Inputs



Boolean Algebra

- To describe the circuits that can be built by combining gates, a new type of algebra is needed, one in which variables and functions can take on only the values 0 and 1.
- Such an algebra is called a Boolean algebra.
- George Boole (1815-1864).
- A Boolean function has one or more input variables and yields a result that depends only on the values of these variables.
- A simple function, f, can be defined by saying that f(A) is 1 if A is 0 and f(A) is 0 if A is 1.
- This function is the NOT function.



Describing Boolean Functions

- Boolean functions of n variables only 2^n possible combinations of input values.
- Thus , a function can be completely described by giving a table with 2^n rows, each row telling the value of the function for a different combination of input values - a **truth table**.
- A function can be also completely described by using the 2^n-bit binary number obtained by reading the result column of the truth table vertically.
- Thus NAND is 1110, NOR is 1000, AND is 0001 and OR is 0111.
- Only 16 Boolean functions of two variables exist.

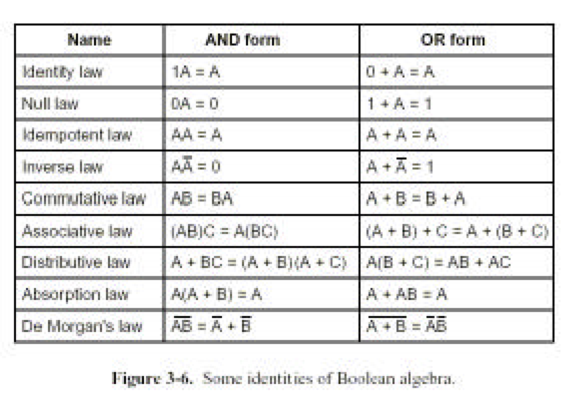
x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*
0	*1*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*
1	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*1*
1	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*	*0*	*1*
Operator Symbol			•	/		/	'	⊕	+	↓	⊙	'	⊂	'	⊃	↑	

Boolean Function	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⊕Y	Exclusive-OR	X or Y but not both
F_7 = X + Y	X+Y	OR	X or Y
F_8 = (X + Y)'	X↓Y	NOR	Not OR
F_9 = XY + X'Y'	X⊙Y	Equivalance*	X equals Y
F_{10} = Y'	Y'	Complement	Not Y
F_{11} = X + Y'	X⊂Y	Implication	If Y then X
F_{12} = X'	X'	Complement	Not X
F_{13} = X' + Y	X⊃Y	Implication	If X then Y
F_{14} = (XY)'	X↑Y	NAND	Not and
F_{15} = 1		Identity	Binary Constant 1

Laws of Boolean Algebra

- In general, a circuit designer starts with a Boolean function and then apply the laws of Boolean algebra to it in an attempt to find a simpler but equivalent one.
- From the final function, a circuit can be constructed.
- To use this approach, we need some identities from Boolean algebra.

Identities of Boolean Algebra



Name	AND form	OR form
Identity law	1A = A	0 + A = A
Null law	0A = 0	1 + A = 1
Idempotent law	AA = A	A + A = A
Inverse law	$A\overline{A} = 0$	$A + \overline{A} = 1$
Commutative law	AB = BA	A + B = B + A
Associative law	(AB)C = A(BC)	(A + B) + C = A + (B + C)
Distributive law	A + BC = (A + B)(A + C)	A(B + C) = AB + AC
Absorption law	A(A + B) = A	A + AB = A
De Morgan's law	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$

Figure 3-6. Some identities of Boolean algebra.

Comments on The Identities

- It is interesting to note that each law has two forms that are **duals** of each other.
- By interchanging AND and OR and also 0 and 1, either form can be produced from the other one.
- All the laws can be easily proven by constructing their truth tables.
- Except for DeMorgan's law, the absorption law, and the AND form of the distributive law, the results are reasonably intuitive.
- DeMorgan's law can be ~~extended~~ ~~to more~~ than two variables, for example, *ABC* = *A* + *B* + *C* .

Consequences of DeMorgan’s Law

- DeMorgan’s law suggests an alternative notation.
- An OR gate with inverted inputs is equivalent to a NAND gate.
- A NOR gate can be drawn as AND gate with inverted inputs.
- Negating both forms of DeMorgan’s law also has interesting consequences - leads to equivalent representations of the AND and OR gates.

Consequences of DeMorgan’s Law

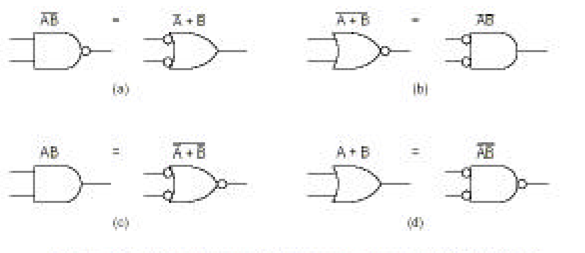

$\overline{AB}$
=
$\overline{A}+\overline{B}$
(a)
$\overline{A+B}$
=
$\overline{A}\,\overline{B}$
(b)
AB
=
$\overline{\overline{A}+\overline{B}}$
(c)
A + B
=
$\overline{\overline{A}\,\overline{B}}$
(d)

Figure 3-7. Alternative symbols for some gates: (a) NAND. (b) NOR. (c) AND. (d) OR.

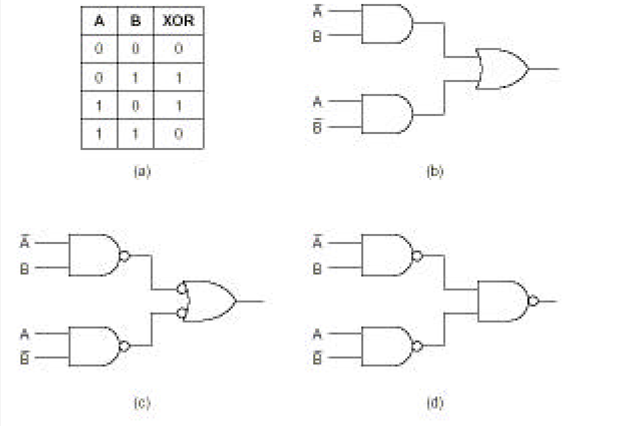
Using The Identities

- Using the identities it is easy to convert the sum-of-products representations of a truth table to pure NAND or pure NOR form.
- Example: consider the EXCLUSIVE-OR function

$$XOR = \bar{A}B + A\bar{B}$$

- How do we convert this to a completely NAND form?
- The standard sum-of-products circuit is shown in Fig. 3-8(b).
- Note that inversion bubbles can be moved along a line at will.

The EXCLUSIVE-OR Gate



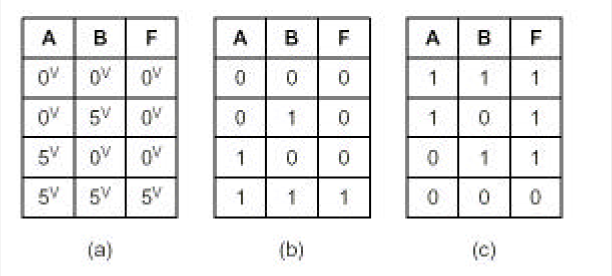
A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Figure 3-8. (a) The truth table for the XOR function. (b)-(d) Three circuits for computing it.

Positive and Negative Logic

- As a final note on circuit equivalence, we will now demonstrate the surprising result that the same physical gate can compute different functions , depending on the conventions used.
- If we adopt the convention that 0 volts is logical **0** and 5 volts is logical **1** , this is called **positive logic**.
- If, however, in **negative logic** , 0 volts denotes a logical **1** and 5 volts a logical **0**.
- What is the significance?

Positive and Negative Logic



A	B	F
0^V	0^V	0^V
0^V	5^V	0^V
5^V	0^V	0^V
5^V	5^V	5^V

(a)

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

(b)

A	B	F
1	1	1
1	0	1
0	1	1
0	0	0

(c)

Figure 3-9. (a) Electrical characteristics of a device. (b) Positive logic. (c) Negative logic.

- Thus, the convention chosen to map voltages onto logical values is critical.
- Except where otherwise specified, we will henceforth use positive logic, so the terms logical 1, true, and high are synonyms, as are logical 0, false, and low.

Example

- Simplify the given Boolean Function by algebraic manipulation.

$F = X'YZ + X'YZ' + XZ$

$= X'Y (Z + Z') + XZ$ Distributive Law

$= X'Y + XZ$ $Z + Z' = 1$

Complement of a Function

- Find the complement of a given function:

$F = X'YZ' + X'Y'Z$

$F' = (X'YZ' + X'Y'Z)' = (X'YZ')' (X'Y'Z)'$

$F' = (X + Y' + Z) (X + Y + Z')$

Complement of a Function

- A Simpler Method:
  - Take the dual
  - Complement each variable

    $F = X'YZ' + X'Y'Z$

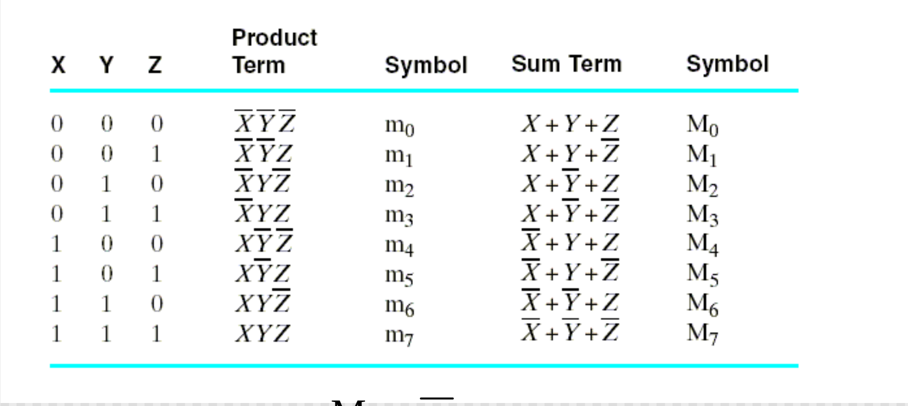
    $F_{\mathbf{dual}} = (X' + Y + Z')\,(X' + Y' + Z)$

    $F' = (X + Y' + Z)\,(X + Y + Z')$

Standard Forms

- Minterms & Maxterms
- Sum of Products
- Product of Sums

Minterms & Maxterms



X	Y	Z	Product Term	Symbol	Sum Term	Symbol
0	0	0	$\overline{X}\,\overline{Y}\,\overline{Z}$	m_0	$X+Y+Z$	M_0
0	0	1	$\overline{X}\,\overline{Y}Z$	m_1	$X+Y+\overline{Z}$	M_1
0	1	0	$\overline{X}Y\overline{Z}$	m_2	$X+\overline{Y}+Z$	M_2
0	1	1	$\overline{X}YZ$	m_3	$X+\overline{Y}+\overline{Z}$	M_3
1	0	0	$X\overline{Y}\,\overline{Z}$	m_4	$\overline{X}+Y+Z$	M_4
1	0	1	$X\overline{Y}Z$	m_5	$\overline{X}+Y+\overline{Z}$	M_5
1	1	0	$XY\overline{Z}$	m_6	$\overline{X}+\overline{Y}+Z$	M_6
1	1	1	XYZ	m_7	$\overline{X}+\overline{Y}+\overline{Z}$	M_7

$$M_j = \overline{m_j}$$

Example

- The Boolean Function can be expressed as sum of product terms or product of sum terms.

F = XYZ + XYZ + XYZ + XYZ

X	Y	Z	F	$\overline{F}$
0	0	0	1	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	1
1	0	1	1	0
1	1	0	0	1
1	1	1	1	0

$F = m_0 + m_2 + m_5 + m_7$

$\underline{F}\,(X,Y,Z) = \Sigma\, m\,(0,2,5,7)$

$F = \underline{m_1 + m_3 + m_4 + m_6}$

$F = \underline{m_1} + \underline{m_3} + \underline{m_4} + m_6$

$F = m_1 \;\; m_3 \;\; m_4 \;\; m_6$

$F = M_1 \;\; M_3 \;\; M_4 \;\; M_6$

$= \Pi\, M\ (\ 1,3,4,6)$

Example

- A function that is not expressed in terms of sum of minterms can be converted to sum of minterms by use of truth tables.

$$E = \overline{Y} + \overline{X}\,\overline{Z}$$

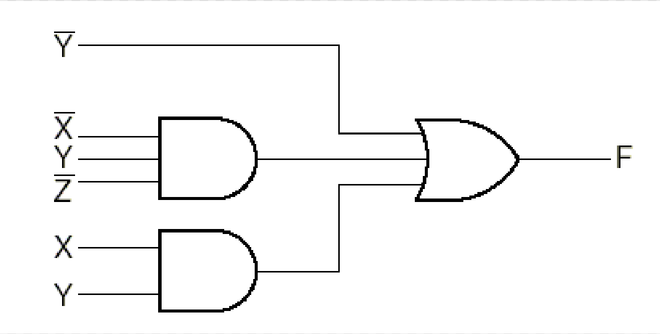
X	Y	Z	E
0	*0*	*0*	*1*
0	*0*	*1*	*1*
0	*1*	*0*	*1*
0	*1*	*1*	*0*
1	*0*	*0*	*1*
1	*0*	*1*	*1*
1	*1*	*0*	*0*
1	*1*	*1*	*0*

$E = \Sigma\ m\ (0,1,2,4,5)$

$\overline{E} = \Sigma\ m\ (3,6,7)$

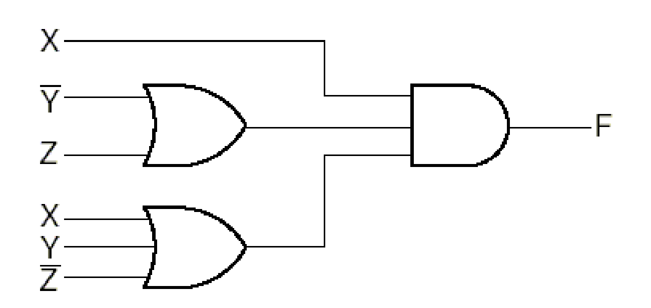
Sum of Products

$$F = \overline{Y} + \overline{X}Y\overline{Z} + XY$$



Product of Sums

$$F = X\,(\overline{Y} + Z)\,(X + Y + \overline{Z})$$



The Sum of Products Notation

- We will consider an example.
- We will consider the truth table for a Boolean function of three variables: *M = f(A,B,C).*
- In particular, we will consider **the majority logic function** , that is, it is 0 if a majority of its inputs are 0 and 1 if a majority of its inputs 1.
- Although any Boolean function can be fully specified by giving its truth table, as the number of variables increases, this notation becomes increasingly cumbersome.
- Instead , another notation is frequently used.

An Alternative Notation

- Note that any Boolean function can be specified by telling which combinations of input variables give an output value of 1.
- By convention, we will place a bar over an input variable to indicate that its value is inverted - the absence of a bar means that it is not inverted.
- We will use implied multiplication or a dot to mean the Boolean AND function and "+" to mean the Boolean OR function.
- Thus, for example, *AB'C* takes the value 1 only when A=1 and B=0 and C=1.
- Also, *AB'* + *BC'* is 1 only when (A=1 and B=0) or (B=1 and C=0).

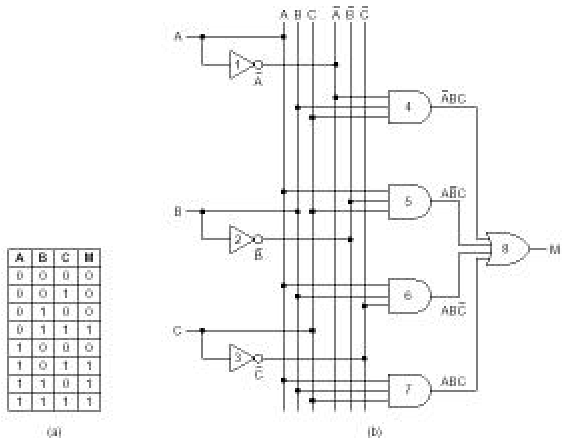
The Majority Function

- The majority function can be defined as follows:

$$M = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

- A function of *n* variables can thus be described by giving a "sum" of at most 2^n-variable "product" terms.
- This formulation is especially important, as we will see shortly, because it leads directly to an implementation of the function using standard gates.

The Majority Function



A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Figure 3-3. (a) The truth table for the majority function of three variables. (b) A circuit for (a).

Implementations of Boolean Functions

1. Write down the truth table for the function.
2. Provide inverters to generate the complement of each input.
3. Draw an AND gate for each term with a 1 in the result column.
4. Wire the AND gates to the appropriate inputs.
5. Feed the output of all the AND gates into an OR gate.

Using only NAND and NOR Gates

- It is often convenient to implement circuits using only a single type of gate.
- Both NAND and NOR gates are said to be complete, because any Boolean function can be computed using either of them.
- One way to implement a Boolean function using only NAND or only NOR gates is :

  1. Follow the procedure given above for constructing it with NOT, AND, and OR.

  2. Then replace the multi-input gates with equivalent circuits using two-input gates.

  3. Finally, the NOT, AND, and OR gates are replaced by the following circuits.

Using only NAND and NOR Gates

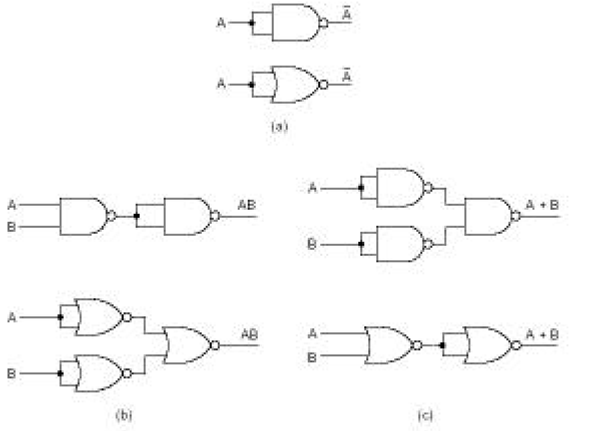


Figure 3-4. Construction of (a) NOT, (b) AND, and (c) OR gates using only NAND gates or only NOR gates.

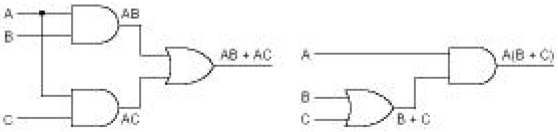
Circuit Equivalence

- Circuits with fewer and/or simpler gates (fewer inputs) are better.
- Boolean algebra can be a valuable tool for simplifying circuits.
- Example:

  $M = AB + AC$

- Many of the rules of **ordinary algebra** also hold for Boolean algebra.
- In particular, $AB + AC$ can be factored into $A(B+C)$ using the distributive law.
- Two functions are equivalent if and only if they have the same output for all possible inputs.
- Thus, $AB + AC$ is equivalent to $A(B+C)$.

Circuit Equivalence



A	B	C	AB	AC	AB + AC
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

(a)

A	B	C	A	B + C	A(B + C)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

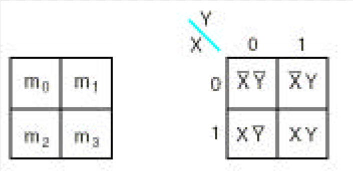
(b)

Figure 3-5. Two equivalent functions. (a) $AB + AC$. (b) $A(B + C)$.

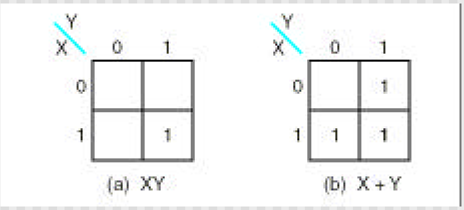
Map Simplification

- Two Variable Map
- Three Variable Map
- Four Variable Map

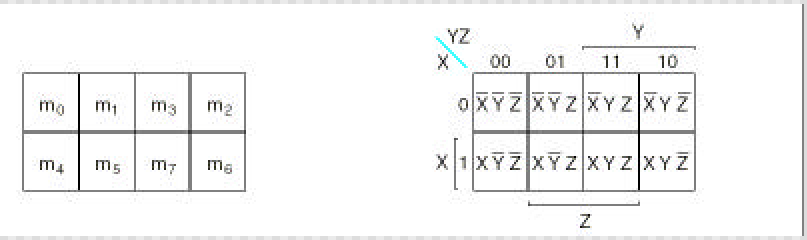
Two Variable Map



•Representations of Functions In the map.



Three Variable Map



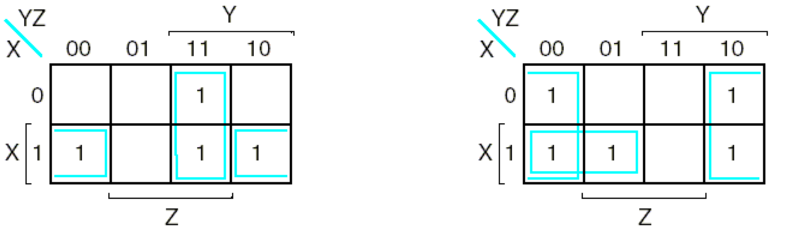
m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

X \ YZ	00	01	11	10
0	$\bar{X}\bar{Y}\bar{Z}$	$\bar{X}\bar{Y}Z$	$\bar{X}YZ$	$\bar{X}Y\bar{Z}$
1	$X\bar{Y}\bar{Z}$	$X\bar{Y}Z$	XYZ	$XY\bar{Z}$

The adjacent squares to m_5 (101) are m_7 (111), m_4 (100), m_1 (001).

$$m_5 + m_7 = X\bar{Y}Z + XYZ$$
$$= XZ(\bar{Y} + Y) = XZ$$

Examples



X \ YZ	00	01	11	10
0			1	
1	1		1	1

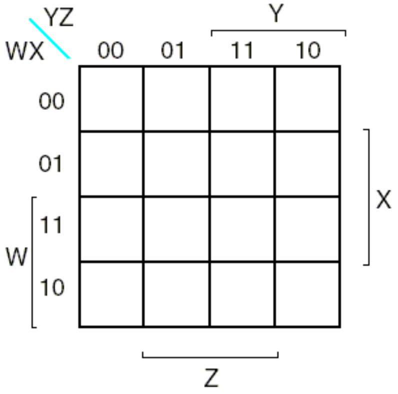
(a) $F_1(X, Y, Z) = \Sigma m(3, 4, 6, 7)$
$= YZ + X\bar{Z}$

X \ YZ	00	01	11	10
0	1			1
1	1	1		1

(b) $F_2(X, Y, Z) = \Sigma m(0, 2, 4, 5, 6)$
$= \bar{Z} + X\bar{Y}$

Four Variable Map

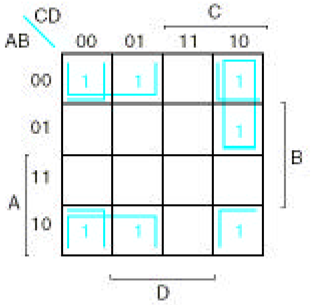
m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}



Example:

- Simplify the Boolean function:

$$F = \overline{A}\,\overline{B}\,\overline{C} + \overline{B}C\overline{D} + \overline{A}BC\overline{D} + A\overline{B}\,\overline{C}$$

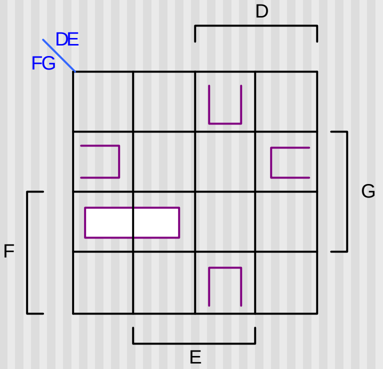


$$F = \overline{B}\,\overline{D} + \overline{B}\,\overline{C} + \overline{A}C\overline{D}$$

Example:

- Simplify the function:

  $F = \Sigma\ m\ (1,3,7,9,12,14)$

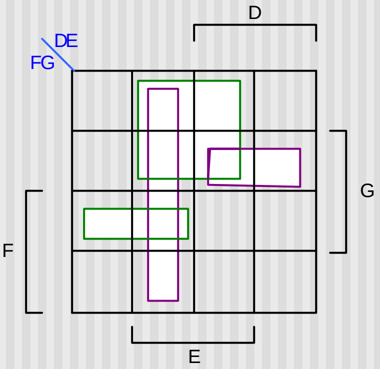


$F = AB\overline{D} + \overline{A}CD + \overline{B}\,\overline{C}D$

Example:

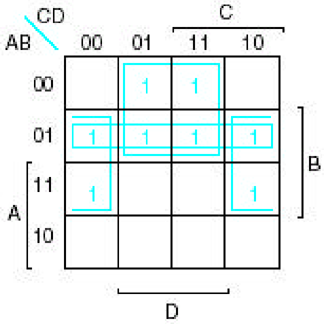
- Simplify the function:

  $F = \Sigma\ m\ (3,4,5,6,7,9,12,13)$



$F = \overline{A}B + B\overline{C} + A\overline{C}D + \overline{A}CD$

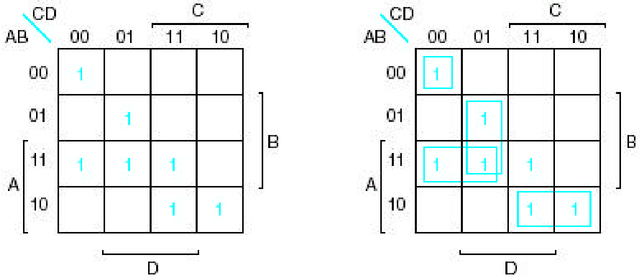
Prime Implicants



AB \ CD	00	01	11	10
00		1	1	
01	1	1	1	1
11	1			1
10				

Fig. 2-21 Prime Implicants for Example 2-7: $\bar{A}D$, $B\bar{D}$, and $\bar{A}B$

Prime Implicants



AB \ CD	00	01	11	10
00	1			
01		1		
11	1	1	1	
10			1	1

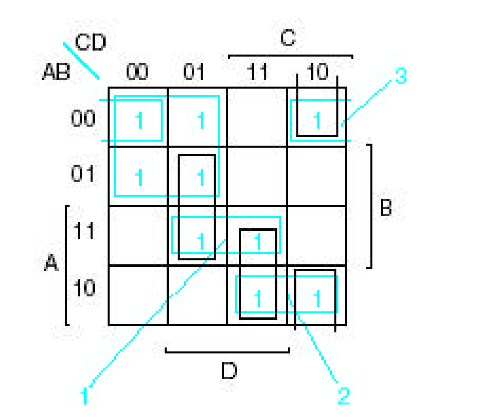
(a) Plotting the minterms

AB \ CD	00	01	11	10
00	1			
01		1		
11	1	1	1	
10			1	1

(b) Essential prime implicants

Fig. 2-22 Simplification with Prime Implicants in Example 2-8

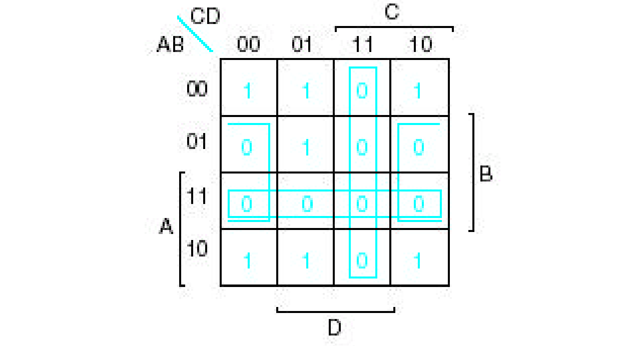
Prime Implicants



AB \ CD	00	01	11	10
00	1	1		1
01	1	1		
11		1	1	
10			1	1

Fig. 2-23 Map for Example 2-9

Simplifying a Product of Sums Form

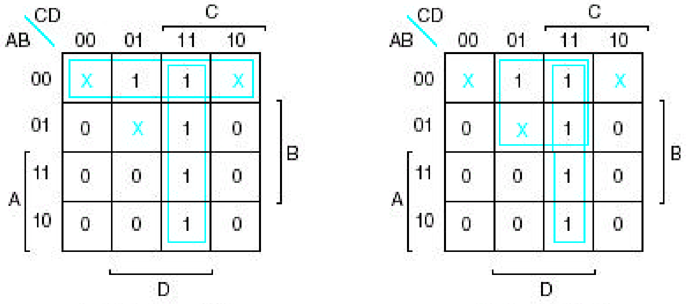


AB \ CD	00	01	11	10
00	1	1	0	1
01	0	1	0	0
11	0	0	0	0
10	1	1	0	1

Fig. 2-24 Map for Example 2-10: $F = (\bar{A} + \bar{B})(\bar{C} + \bar{D})(\bar{B} + D)$

$$\overline{F} = AB + CD + B\overline{D}$$

Don’t Care Conditions



AB \ CD	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

(a) $F = CD + \overline{A}\,\overline{B}$

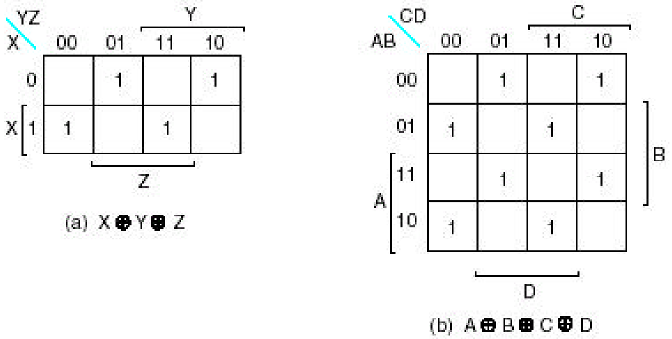
AB \ CD	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

(b) $F = CD + \overline{A}D$

Fig. 2-25 Example with Don't-Care Conditions

Exclusive-Or Gate

Odd Function

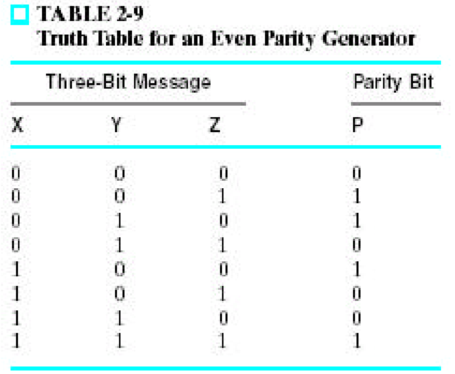


(a) X ⊕ Y ⊕ Z

(b) A ⊕ B ⊕ C ⊕ D

Fig. 2-38 Maps for Multiple-Variable Odd Functions

Parity Generation and Checking



□ **TABLE 2-9**
Truth Table for an Even Parity Generator

Three-Bit Message			Parity Bit
X	Y	Z	P
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

Table 2-9 Truth Table for an Even Parity Generator