Chapter 2

Combinational Logic Circuits

J.J. Shann

Chapter Overview

2-1 Binary Logic and Gates

- Digital circuits:
  - hardware components that manipulate binary information
  - are implemented using transistors and interconnections in integrated circuits.
- Logic gate:
  - basic ckt that performs a specific logical op

A. Binary Logic

- Binary logic:
  - deals w/ binary variables and the ops of mathematical logic applied to these variables.
    - binary variable: variable that take on two discrete values
  - resembles binary arithmetic, but should no be confused w/ each other.

Basic Logic Operations

- Basic logical ops: p.31, Table 2-1

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		

- **AND**: • , ∧
  - identical to binary multiplication
- **OR**: + , ∨
  - resemble binary addition

    In binary logic, 1 + 1 = 1

    In binary arithmetic: 1 + 1 = 10
- **NOT**: complement; ¯ , ′

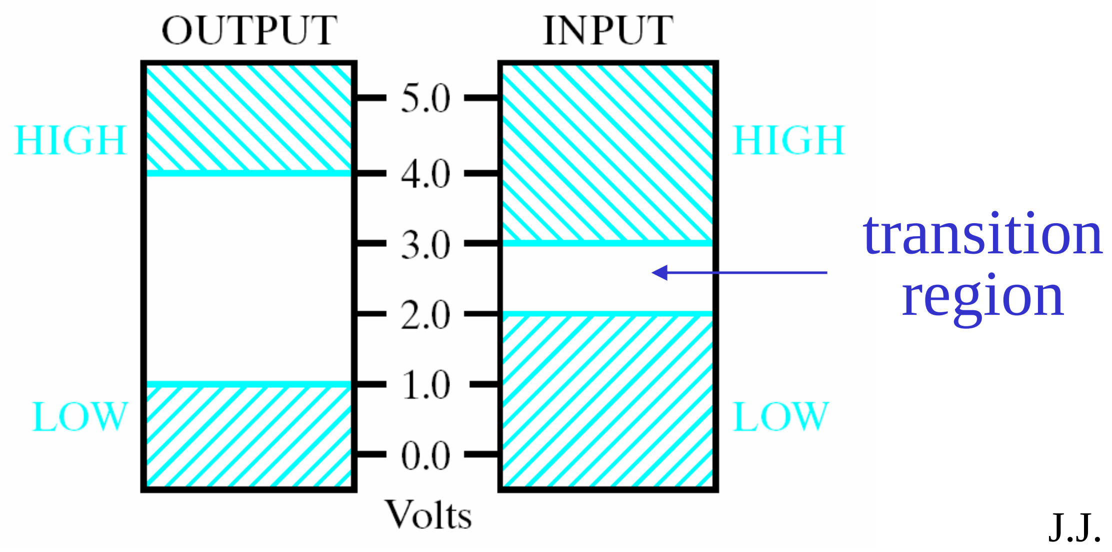
Truth Table

- Truth table:
  - a table of combinations of the binary variables showing the relationship b/t the values that the variables take on and the values of the result of the op.
  - 2^n rows, n: # variables
  - E.g.: truth table for AND op

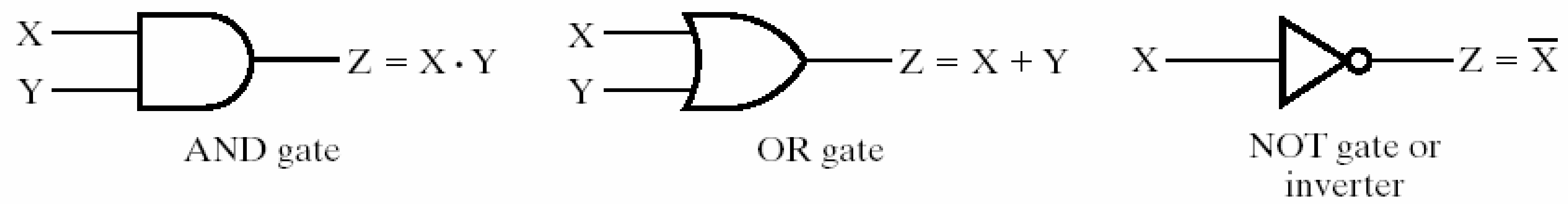
AND		
X	Y	$Z = X \cdot Y$
0	0	0
0	1	0
1	0	0
1	1	1

B. Logic Gates

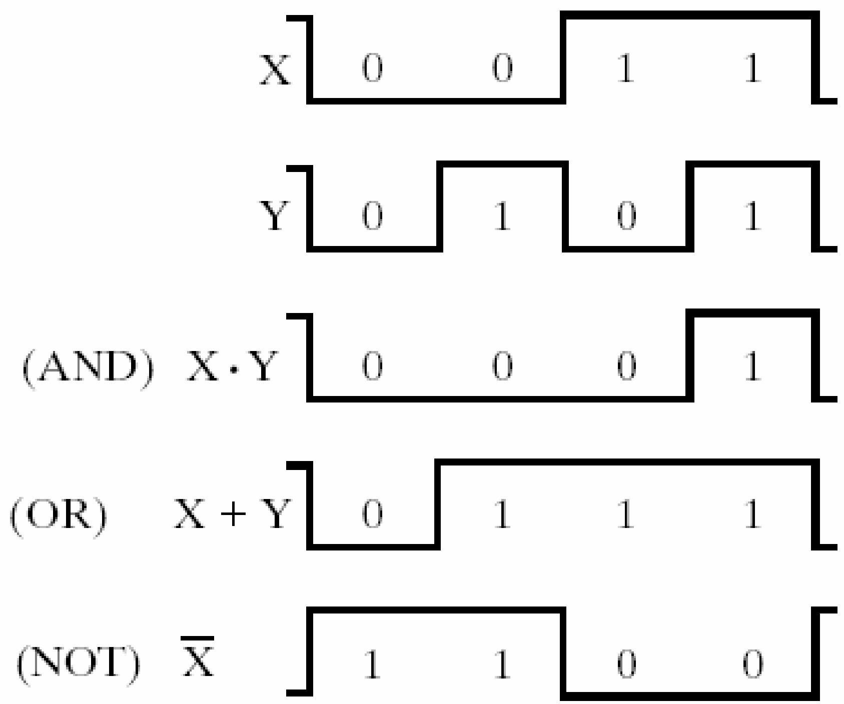
- Logic gate:
  - is an electronic ckt that operate on one or more input signals to produce an output signal.
  - The input terminals of logic gates accept binary signals within the allowable range and respond at the output terminals w/ binary signals that fall within a specified range.
    - E.g.:



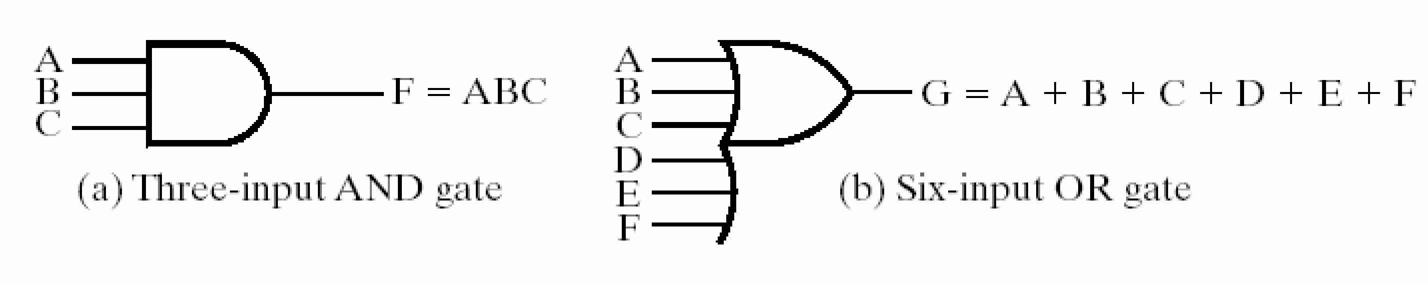
- Graphic symbols of 3 basic logic gates:



- Timing diagram:



- Multiple-input logic gates:
  - AND and OR gates may have ≥ 2 inputs.
  - E.g.:



2-2 Boolean Algebra

- Boolean algebra:
  - is an algebra dealing w/ binary variables and logic ops
    - binary variables: are designated by letters of the alphabet
    - logic ops: AND, OR, NOT
- Boolean expression:
  - an algebraic expression formed by using

    binary variables,

    the constants 0 and 1,

    the logic op symbols, and

    parentheses.

Boolean Function

- Boolean function:
  - can be described by

    a Boolean equation,

    a truth table, or

    a logic ckt diagram
  - Single-output Boolean function
  - Multiple-output Boolean function

Boolean Equation

- Boolean equation:
  - consists of a binary variable identifying the function followed by an equal sign and a Boolean expression.
  - expresses the logical relationship b/t binary variables
  - can be expressed in a variety of ways

    * Obtain a simpler expression for the same function.
  - E.g.:

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

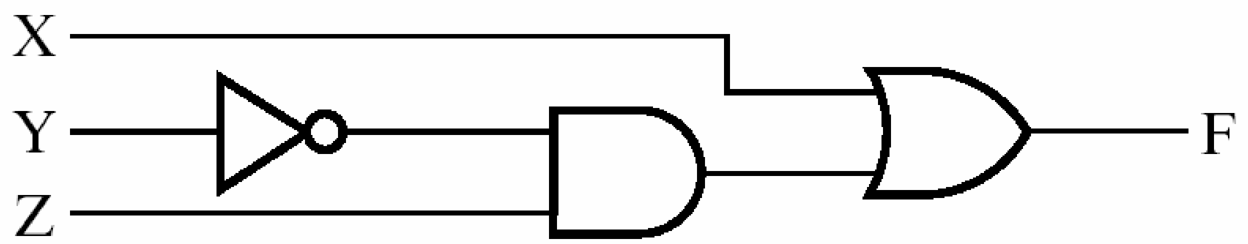
terms

- Truth table for a function: is unique
  - a list of all combinations of 1's and 0's that can be assigned to the binary variables and a list that shows the value of the function for each binary combination.
  - E.g.: $F(X,Y,Z) = X + \overline{Y}Z$

X	Y	Z	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

- Logic circuit diagram:
  - An algebraic expression for a Boolean function

    $\Rightarrow$ A ckt diagram composed of logic gates
  - Circuit gates are interconnected by wires that carry logic signals.
  - E.g.: $F(X,Y,Z) = X + \bar{Y}Z$



- Combinational logic circuits

Example

- Present a set of requirements under which an insurance policy will be issued:

The applicant must be

1. a married female 25 years old or over, or
2. a female under 25, or
3. a married male under 25 who has not been involved in a car accident, or
4. a married male who has been involved in a car accident, or
5. a married male 25 years or over who has not been involved in a car accident.

<Ans.>

- Define variables: 4; w, x, y, z

  $w = 1$ if applicant has been involved in a car accident

  $x = 1$ if applicant is married

  $y = 1$ if applicant is a male

  $z = 1$ if applicant is under 25

- Find a Boolean expression which assumes the value 1 whenever the policy should be issued:

$x y' z'$ 1. a married female 25 years old or over

$y' z$ 2. a female under 25

$w' x y z$ 3. a married male under 25 who has not been involved in a car accident

$w x y$ 4. a married male who has been involved in a car accident

$w' x y z'$ 5. a married male 25 years or over who has not been involved in a car accident

$$\Rightarrow F = x y' z' + y' z + w' x y z + w x y + w' x y z'$$

- Simplify the expression and suggest a simpler set of requirements:

$$F = x\,y'\,z' + y'\,z + w'\,x\,y\,z + w\,x\,y + w'\,x\,y\,z'$$
$$= x + y'\,z$$

$w = 1$ if applicant has been involved in a car accident
$x = 1$ if applicant is married
$y = 1$ if applicant is a male
$z = 1$ if applicant is under 25

The applicant must be

1. married or
2. a female under 25.

A. Basic Identities of Boolean Algebra

- Basic identities of Boolean Algebra:
  - *X*, *Y*, *Z*: binary variable

1.	$X+0=X$	2.	$X\cdot 1=X$		
3.	$X+1=1$	4.	$X\cdot 0=0$		
5.	$X+X=X$	6.	$X\cdot X=X$		
7.	$X+\overline{X}=1$	8.	$X\cdot\overline{X}=0$		
9.	$\overline{\overline{X}}=X$				
10.	$X+Y=Y+X$	11.	$XY=YX$	Commutative	
12.	$X+(Y+Z)=(X+Y)+Z$	13.	$X(YZ)=(XY)Z$	Associative	
14.	$X(Y+Z)=XY+XZ$	15.	$X+YZ=(X+Y)(X+Z)$	Distributive	
16.	$\overline{X+Y}=\overline{X}\cdot\overline{Y}$	17.	$\overline{X\cdot Y}=\overline{X}+\overline{Y}$	DeMorgan's	

dual

Duality Principle

- *Dual*:
  - The dual of an algebraic expression is obtained by interchanging OR and AND ops and replacing 1's by 0's and 0's by 1's (OR $\leftrightarrow$ AND, 0 $\leftrightarrow$ 1)
  - E.g.: $X + 0 \xrightarrow{dual} X \cdot 1$
- Duality principle:
  - A Boolean equation remains valid

    if we take the dual of the expressions on both sides of the equals sign.
  - E.g.:
    1. X + 0 = X
    2. X • 1 = X

Verification of Identities

- Verification of an identity:
  - by truth table or verified identities
  - E.g.: $X + 0 = X$

    $X = 0,\ X + 0 = 0 + 0 = 0$

    $X = 1,\ X + 0 = 1 + 0 = 1$

X	$X + 0$
0	$0 + 0 = 0$
1	$1 + 0 = 1$

  - E.g.: DeMorgan's theorem

    $(X + Y)' = X' \bullet Y'$

A)	X	Y	$X+Y$	$\overline{X+Y}$
	0	0	0	1
	0	1	1	0
	1	0	1	0
	1	1	1	0

B)	X	Y	$\overline{X}$	$\overline{Y}$	$\overline{X} \cdot \overline{Y}$
	0	0	1	1	1
	0	1	1	0	0
	1	0	0	1	0
	1	1	0	0	0

General DeMorgan's Theorem

- DeMorgan's theorem:

$$\overline{X+Y} = \overline{X} \cdot \overline{Y}$$
$$\overline{X \cdot Y} = \overline{X} + \overline{Y}$$

- General DeMorgan's theorem:

$$\overline{X_1 + X_2 + \ldots + X_n} = \overline{X}_1 \cdot \overline{X}_2 \cdot \ldots \cdot \overline{X}_n$$
$$\overline{X_1 \cdot X_2 \cdot \ldots \cdot X_n} = \overline{X}_1 + \overline{X}_2 + \ldots + \overline{X}_n$$

B. Algebraic Manipulation

- Simplification of Boolean functions:
  - Boolean algebra is a useful tool for simplifying digital ckts.

1.	$X+0=X$	2.	$X\cdot 1=X$	
3.	$X+1=1$	4.	$X\cdot 0=0$	
5.	$X+X=X$	6.	$X\cdot X=X$	
7.	$X+\overline{X}=1$	8.	$X\cdot\overline{X}=0$	
9.	$\overline{\overline{X}}=X$			
10.	$X+Y=Y+X$	11.	$XY=YX$	Commutative
12.	$X+(Y+Z)=(X+Y)+Z$	13.	$X(YZ)=(XY)Z$	Associative
14.	$X(Y+Z)=XY+XZ$	15.	$X+YZ=(X+Y)(X+Z)$	Distributive
16.	$\overline{X+Y}=\overline{X}\cdot\overline{Y}$	17.	$\overline{X\cdot Y}=\overline{X}+\overline{Y}$	DeMorgan's

  - E.g.: $F=\overline{X}YZ+\overline{X}Y\overline{Z}+XZ$

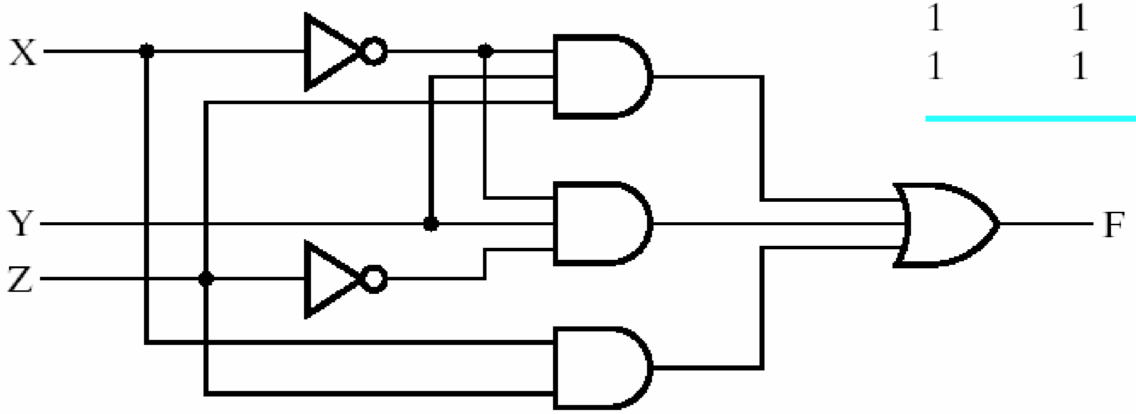
$$F=\overline{X}YZ+\overline{X}Y\overline{Z}+XZ$$

$$=\overline{X}Y(Z+\overline{Z})+XZ \qquad 14: x(y+z)=xy+xz$$

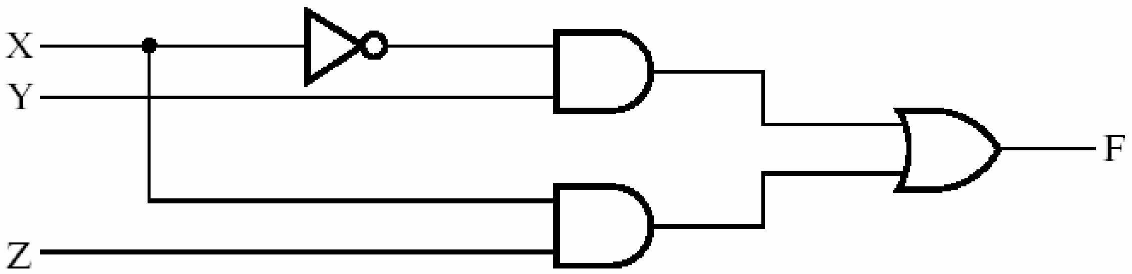
$$=\overline{X}Y\cdot 1+XZ \qquad 7: x+\overline{x}=1$$

$$=\overline{X}Y+XZ \qquad 2: x\cdot 1=x$$

X	Y	Z	(a) F	(b) F
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1



(a) $F = \overline{X}YZ + \overline{X}Y\overline{Z} + XZ$



(b) $F = \overline{X}Y + XZ$

Implementing Boolean Equations

- When a Boolean equation is implemented w/ logic gates:
  - each term requires a gate
  - each variable within the term designates an input to the gate
    - ***literal***: a single variable within a term that may or may not be complemented
  - E.g.: $F = \overline{X}YZ + \overline{X}Y\overline{Z} + XZ$ … 3 terms & 8 literals

    $F = \overline{X}Y + XZ$ … 2 terms & 4 literals

* Reduce # of terms, # of literals, or both in a Boolean expression ⇒ is often possible to obtain a simpler ckt

Reducing Expressions by Boolean Algebra

- Boolean algebra may be applied to reduce an expression for obtaining a simpler ckt:
  - Computer tools for synthesizing logic ckt
  - Manual method: cut-and-try
  - E.g.s:

$$1.\ X + XY = X(1+Y) = X$$

$$2.\ XY + X\overline{Y} = X(Y+\overline{Y}) = X$$

$$3.\ X + \overline{X}Y = (X+\overline{X})(X+Y) = X+Y$$

$\Downarrow$ dual

$$4.\ X(X+Y) = X + XY = X$$

$$5.\ (X+Y)(X+\overline{Y}) = X + Y\overline{Y} = X$$

$$6.\ X(\overline{X}+Y) = X\overline{X} + XY = XY$$

1.	$X+0=X$	2.	$X\cdot 1=X$
3.	$X+1=1$	4.	$X\cdot 0=0$
5.	$X+X=X$	6.	$X\cdot X=X$
7.	$X+\overline{X}=1$	8.	$X\cdot\overline{X}=0$
9.	$\overline{\overline{X}}=X$		
10.	$X+Y=Y+X$	11.	$XY=YX$
12.	$X+(Y+Z)=(X+Y)+Z$	13.	$X(YZ)=(XY)Z$
14.	$X(Y+Z)=XY+XZ$	15.	$X+YZ=(X+Y)(X+Z)$
16.	$\overline{X+Y}=\overline{X}\cdot\overline{Y}$	17.	$\overline{X\cdot Y}=\overline{X}+\overline{Y}$

Consensus Theorem

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

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

$$
\begin{aligned}
xy + x'z + yz &= xy + x'z + yz(x + x') \\
&= xy + x'z + xyz + x'yz \\
&= xy + xyz + x'z + x'yz \\
&= xy(1 + z) + x'z(1 + y) \\
&= xy + x'z
\end{aligned}
$$

  – can be used to eliminate redundant terms from Boolean expressions.

* Venn diagram

- E.g.: Simplify the expression (p.2-15)

$$F = x\,y'\,z' + y'\,z + w'\,x\,y\,z + w\,x\,y + w'\,x\,y\,z'$$

$$= x + y'\,z$$

C. Complement of a Function

- Complement of a function F:
  - interchange of 1's to 0's and 0's to 1's for the values of F in the truth table
  - can be derived algebraically by applying DeMorgan's theorem $\Rightarrow$ interchange AND and OR ops and complement each variable and constant
  - take the dual of the function equation and complement each literal

Examples

- Complementing functions by applying DeMorgan's theorem:
  - E.g.s:

$$F_1 = \overline{X}Y\overline{Z} + \overline{X}\overline{Y}Z \qquad \overline{F_1} = (\overline{X}Y\overline{Z} + \overline{X}\overline{Y}Z)'$$

$$= (\overline{X}Y\overline{Z})' \cdot (\overline{X}\overline{Y}Z)'$$

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

$$F_2 = X(\overline{Y}\overline{Z} + YZ) \qquad \overline{F_2} = (X(\overline{Y}\overline{Z} + YZ))'$$

$$= \overline{X} + (\overline{Y}\overline{Z} + YZ)'$$

$$= \overline{X} + (\overline{Y}\overline{Z})' \cdot (YZ)'$$

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

- Complementing functions by using duals:
  - E.g.s:

$$F_1 = \overline{X}Y\overline{Z} + \overline{X}\,\overline{Y}Z$$

$$\text{dual of } F_1 \Rightarrow (\overline{X} + Y + \overline{Z})\cdot(\overline{X} + \overline{Y} + Z)$$

$$\text{complement each literal} \Rightarrow (X + \overline{Y} + Z)\cdot(X + Y + \overline{Z})$$

$$= \overline{F_1}$$

$$F_2 = X(\overline{YZ} + YZ)$$

$$\text{dual of } F_2 \Rightarrow X + (\overline{Y} + \overline{Z})\cdot(Y + Z)$$

$$\text{complement each literal} \Rightarrow \overline{X} + (Y + Z)\cdot(\overline{Y} + \overline{Z})$$

$$= \overline{F_2}$$

2-3 Standard Forms

- Standard forms:
  - Contain *product terms* & *sum terms*
  - Two types:
    - *sum-of-products form*
    - *product-of-sums form*
- Product term:
  - a logical product consisting of an AND op among literals
  - E.g.: $XY'Z$
- Sum terms
  - a logical sum consisting of an OR op among literals
  - E.g.: $X+Y+Z'$

A. Minterms and Maxterms

- Minterm: m_j
  - a product term in which all the variables appear exactly once, either complemented or uncomplemented
  - represents exactly one combination of the binary variables in a truth table
  - has the value 1 for that combination and 0 for all others
  - There are 2^n distinct minterms for n variables.
  - Symbol: m_j,
    - j: denotes the decimal equivalent of the binary combination for which the minterm has the value 1
  - E.g.: Minterms for 3 variables

– E.g.: Minterms for 3 variables

X	Y	Z	Product Term	Symbol	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7
0	0	0	$\overline{X}\,\overline{Y}\,\overline{Z}$	m_0	1	0	0	0	0	0	0	0
0	0	1	$\overline{X}\,\overline{Y}Z$	m_1	0	1	0	0	0	0	0	0
0	1	0	$\overline{X}Y\overline{Z}$	m_2	0	0	1	0	0	0	0	0
0	1	1	$\overline{X}YZ$	m_3	0	0	0	1	0	0	0	0
1	0	0	$X\overline{Y}\,\overline{Z}$	m_4	0	0	0	0	1	0	0	0
1	0	1	$X\overline{Y}Z$	m_5	0	0	0	0	0	1	0	0
1	1	0	$XY\overline{Z}$	m_6	0	0	0	0	0	0	1	0
1	1	1	XYZ	m_7	0	0	0	0	0	0	0	1

Maxterms

- Maxterm: M_j
  - a sum term that contains all the variables in complemented or uncomplemented form
  - represents exactly one combination of the binary variables in a truth table
  - has the value 0 for that combination and 1 for all others
  - There are 2^n distinct maxterms for n variables.
  - Symbol: M_j,
    - j: denotes the decimal equivalent of the binary combination for which the maxterm has the value 0
  - E.g.: Maxterms for 3 variables

– E.g.: Maxterms for 3 variables

X	Y	Z	Sum Term	Symbol	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
0	0	0	$X+Y+Z$	M_0	0	1	1	1	1	1	1	1
0	0	1	$X+Y+\overline{Z}$	M_1	1	0	1	1	1	1	1	1
0	1	0	$X+\overline{Y}+Z$	M_2	1	1	0	1	1	1	1	1
0	1	1	$X+\overline{Y}+\overline{Z}$	M_3	1	1	1	0	1	1	1	1
1	0	0	$\overline{X}+Y+Z$	M_4	1	1	1	1	0	1	1	1
1	0	1	$\overline{X}+Y+\overline{Z}$	M_5	1	1	1	1	1	0	1	1
1	1	0	$\overline{X}+\overline{Y}+Z$	M_6	1	1	1	1	1	1	0	1
1	1	1	$\overline{X}+\overline{Y}+\overline{Z}$	M_7	1	1	1	1	1	1	1	0

Minterms vs. Maxterms

- A minterm = a function, not equal to 0, having the min # of 1's in its truth table
- A maxterm = a function, not equal to 1, having the max # of 1's in its truth table
- $M_j = (m_j)'$
  - E.g.:

    For 3 variables & $j = 3$

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

$$m_3 = \overline{X}YZ$$

$$\overline{m_3} = \overline{\overline{X}YZ} = X + \overline{Y} + \overline{Z} = M_3$$

Representing Boolean Function by m_j's or M_j's

- Given the truth table of a function

  $\Rightarrow$ a Boolean function in sum of minterms or

  in product of maxterms

  - Sum of minterms:
    - a logical sum of all the minterms that produce a 1 in the function.
  - Product of maxterms:
    - a logical product of all the maxterms that produce a 0 in the function.

- E.g.: Represent F and F' in sum-of-minterms form

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

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

$$= m_0 + m_2 + m_5 + m_7$$

$$= \sum m(0,2,5,7)$$

logical sum (Boolean OR)
of the minterms

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

$$= m_1 + m3 + m4 + m_6$$

$$= \sum m(1,3,4,6)$$

- E.g.: Represent F in product-of-maxterms form

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

$$\overline{F}(X,Y,Z) = \sum m(1,3,4,6)$$

$$\begin{aligned}
F &= \overline{m_1 + m_3 + m_4 + m_6} \\
&= \overline{m_1} \cdot \overline{m_3} \cdot \overline{m_4} \cdot \overline{m_6} \\
&= M_1 \cdot M_3 \cdot M_4 \cdot M_6 \quad (\because \overline{m_j} = M_j) \\
&= (X + Y + \overline{Z})(X + \overline{Y} + \overline{Z})(\overline{X} + Y + Z)(\overline{X} + \overline{Y} + Z) \\
&= \underline{\prod} M(1,3,4,6)
\end{aligned}$$

logical product (Boolean AND)
of the maxterms

Conversion to Sum-of-minterms Form

- Conversion:
  - by means of truth table or Boolean algebra
- E.g.:Convert $E = \overline{Y} + \overline{X}\,\overline{Z}$ to sum-of-minterms form
  - Method 1: by truth table

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

$\Downarrow$ truth table

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

$$\Rightarrow \quad = \sum m(0,1,2,4,5)$$

- E.g.:Convert $E = \overline{Y} + \overline{X}\,\overline{Z}$ to sum-of-minterms form
  - Method 2: by Boolean algebra

$$
\begin{aligned}
E &= \overline{Y} + \overline{X}\,\overline{Z} \\
&= \overline{Y}(X + \overline{X})(Z + \overline{Z}) + \overline{X}\,\overline{Z}(Y + \overline{Y}) \\
&= X\overline{Y}Z + X\overline{Y}\,\overline{Z} + \overline{X}\,\overline{Y}Z + \overline{X}\,\overline{Y}\,\overline{Z} + \overline{X}Y\overline{Z} + \overline{X}\,\overline{Y}\,\overline{Z} \\
&= \overline{X}\,\overline{Y}\,\overline{Z} + + \overline{X}\,\overline{Y}Z + \overline{X}Y\overline{Z} + X\overline{Y}\,\overline{Z} + X\overline{Y}Z \\
&= \sum m(0,1,2,4,5)
\end{aligned}
$$

Summary of the Properties of Minterms

- There are 2^n minterms for n Boolean variables.
  - These minterms can be evaluated from the binary numbers from 0 to $2^n - 1$.
- Any Boolean function can be expressed as a logical sum of minterms.
- The complement of a function contains those minterms not included in the original function.
- A function that includes all the 2^n minterms is equal to logic 1.
  - E.g.: $G(X, Y) = \Sigma m(0,1,2,3) = 1$

B. Sum of Products (SoP)

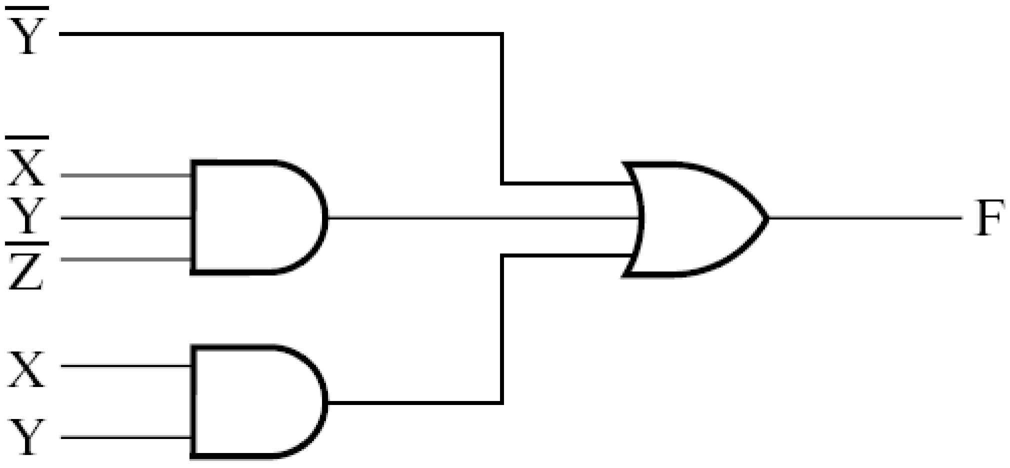
- Sum-of-products form:
  - a standard form that contains product terms w/ any number of literals. These AND terms are ORing together.
  - E.g.: $F(X,Y,Z) = \overline{Y} + \overline{X}Y\overline{Z} + XY$
- Sum-of-minterms form:
  - is a special case of SoP form
  - is obtained directly from a truth table
  - contains the max # of literals in each term and usually has more product terms than necessary

  $\Rightarrow$ simplify the expression to reduce the # of product terms and the # of literals in the terms

  $\Rightarrow$ a simplified expression in *sum-of-products* form

Logic Diagram for an SoP form

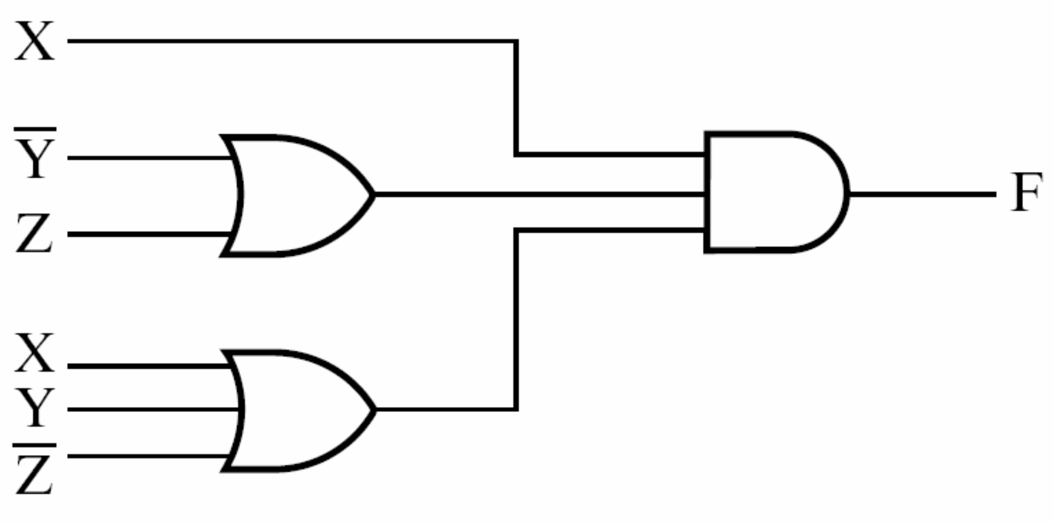
- Logic diagram for an SoP form:
  - consists of a group of AND gates followed by a single OR gate. ⇒ *2-level implementation* or *2-level ckt*
    - Assumption: The input variables are directly available in their complemented and uncomplemented forms. (⇒ Inverters are not included in the diagram)
  - E.g.: $F = \overline{Y} + \overline{X}Y\overline{Z} + XY$



C. Product of Sums (PoS)

- Product-of-sums form:
  - a logical product of sum terms & each logical sum term may have any # of distinct literals.

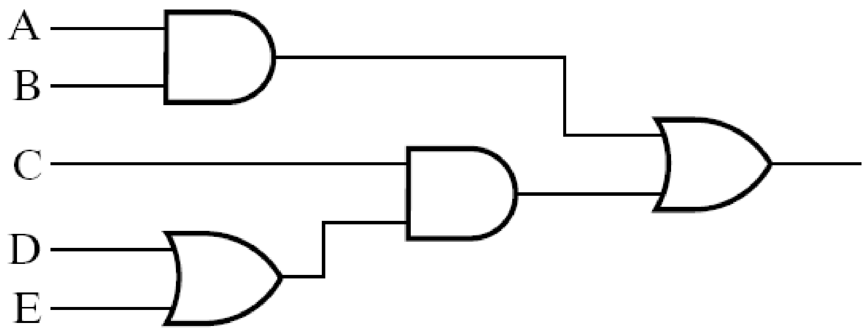
    $\Rightarrow$ 2-level gating structure
  - Special case: Product-of-maxterms form
  - E.g.: $F = X \cdot (\overline{Y} + Z) \cdot (X + Y + \overline{Z})$



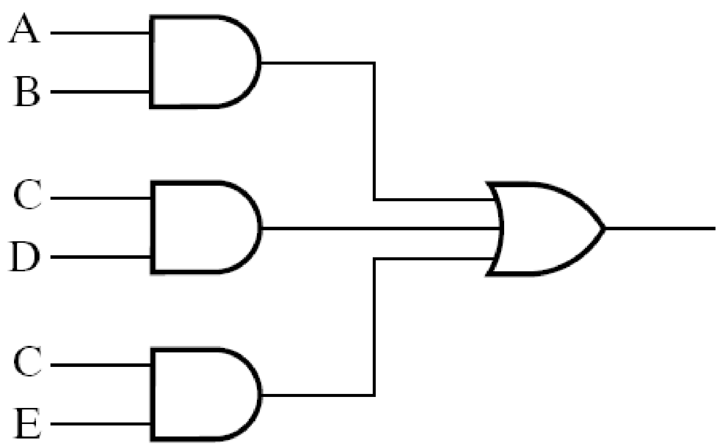
D. Nonstandard Form

- Nonstandard form:
  - E.g.: F = AB + C(D + E) → 3-level
- Nonstandard form → Standard form:
  - By using the distributive laws
  - E.g.: F = AB + C(D + E)

    = AB + CD + DE → 2-level



(a) AB + C(D + E)



(b) AB + CD + CE

2-4 Two-Level Circuit Optimization

- Representation of a Boolean function:
  - Truth table: unique
  - Algebraic expression: many different forms

    ⇒ digital logic circuit
- Minimization of Boolean function:
  - Algebraic manipulation: literal minimization (§2-2)
    - use the rules and laws of Boolean algebra
    - Disadv.: It lacks specific rules to predict each succeeding step in the manipulation process.
  - Map method: gate-level minimization (§2-4~2-5)
    - a simple straightforward procedure using *Karnaugh map* (*K-map*)
    - Disadv.: Maps for more than 4 variables are not simple to use.
  - Tabular method: Quine-McCluskey method (補充資料)

* The simplest algebraic expression: not unique

A. Cost Criteria

- Two cost criteria:
    i. Literal cost
        - the # of literal appearances in a Boolean expression
    ii. Gate input cost (✓)
        - the # of inputs to the gates in the implementation

Literal Cost

- Literal cost:
  - the # of literal appearances in a Boolean expression
  - E.g.: $F = AB + C(D+E) \;\rightarrow 5 \text{ literals}$

    $F = AB + CD + CE \;\rightarrow 6 \text{ literals}$
  - Adv.: is very simple to evaluate by counting literal appearances
  - Disadv.: does not represent ckt complexity accurately in all cases
    - E.g.:

$$G = ABCD + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \;\rightarrow 8 \text{ literals}$$

$$G = (\overline{A}+B)(\overline{B}+C)(\overline{C}+D)(\overline{D}+A) \;\rightarrow 8 \text{ literals}$$

Gate Input Cost

- Gate input cost:
  - the # of inputs to the gates in the implementation
  - For SoP or PoS eqs, the gate input cost can be found by the sum of
    - all literal appearances
    - the # of terms excluding terms that consist only of a single literal
    - the # of distinct complemented single literals (optional)
  - E.g.: p.2-46

$$G = ABCD + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \quad \rightarrow 8 + 2\,(+\,4) \text{ gate input counts}$$

$$G = (\overline{A} + B)(\overline{B} + C)(\overline{C} + D)(\overline{D} + A) \quad \rightarrow 8 + 4\,(+\,4)$$

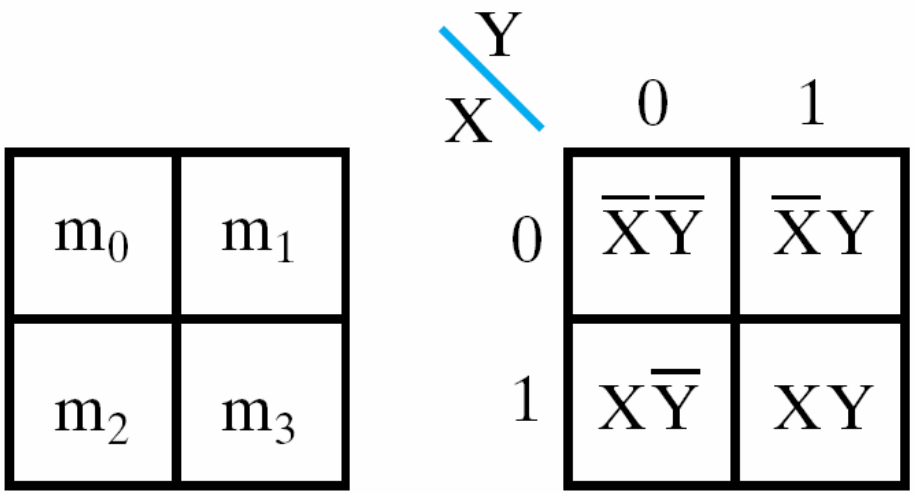
  - is a good measure for contemporary logic implementation
    - is proportional to the # of transistors and wires used in implementing a logic ckt. (especially for ckt ≥ 2 levels)

B. The Map Method

- Map method: Karnaugh map simplification
  - a simple straightforward procedure
  - K-map: a pictorial form of a truth table
    - a diagram made up of squares
    - Each square represents one minterm of the function.
  - The simplified expressions produced by the map are always in one of the two standard forms:
    - SOP (sum of products) or POS (product of sums)

(a) Two-Variable Map

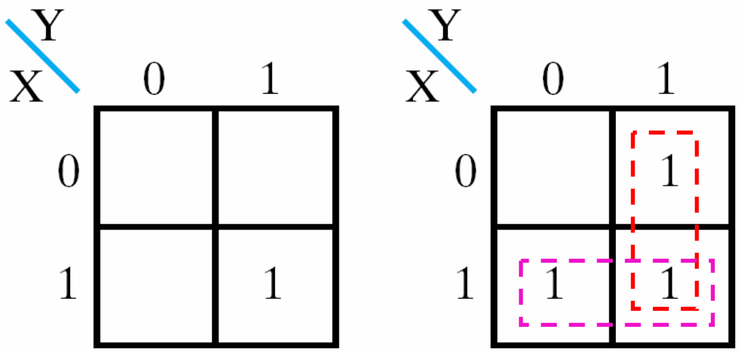
- Two-variable map:
  - 4 squares, one for each minterm.
  - A function of 2 variables can be represented in the map by marking the squares that correspond to the minterms of the function.



(a) (b)

Example

- E.g.:



(a) XY (b) X + Y

<Ans> $F_1 = m_3 = XY$

$$
\begin{aligned}
F_2 &= m_1 + m_2 + m_3 \\
&= \overline{X}Y + X\overline{Y} + XY \\
&= \overline{X}Y + X(\overline{Y} + Y) \\
&= \overline{X}Y + X \\
&= (\overline{X} + X)(Y + X) \\
&= X + Y
\end{aligned}
$$

by combining squares in the map

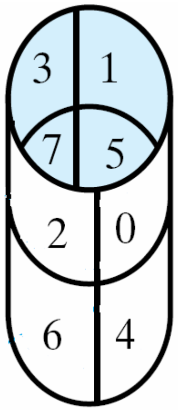
by applying Boolean algebra

(b) Three-Variable Map

- Three-variable map: 8 minterms $\Rightarrow$ 8 squares

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

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



- Only one bit changes in value from one adjacent column to the next
  - Any two adjacent squares in the map differ by only one variable, which is primed in one square and unprimed in the other.
  - E.g.: m_5 & m_7
- Note: Each square has 3 adjacent squares.
  - The right & left edges touch each other to form adjacent squares.
  - E.g.: $m_4 \rightarrow m_0, m_5, m_6$

Map Minimization of SOP Expression

- Basic property of adjacent squares:

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

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

Y (columns 11, 10); Z (columns 01, 11)

  - Any two adjacent squares in the map differ by only one variable: primed in one square and unprimed in the other
    - E.g.: $m_5 = X\overline{Y}Z,\ \ m_7 = XYZ$

$\Rightarrow$ Any two minterms in adjacent squares that are ORed together can be simplified to a single AND term w/ a removal of the different variable.

- E.g.: $m_5 + m_7 = X\overline{Y}Z + XYZ = XZ(\overline{Y} + Y) = XZ$

- Procedure of map minimization of SOP expression:

  i. A 1 is marked in each minterm that represents the function.

  - Two ways:

    (1) Convert each minterm to a binary number and then mark a 1 in the corresponding square.

    (2) Obtain the coincidence of the variables in each term.

  ii. Find possible adjacent 2^k squares:

  - 2 adjacent squares (i.e., minterms) $\rightarrow$ remove 1 literal
  - 4 adjacent squares (i.e., minterms) $\rightarrow$ remove 2 literal
  - 2^k adjacent squares (i.e., minterms) $\rightarrow$ remove k literal

  $\Rightarrow$ The larger the # of squares combined, the less the # of literals in the product (AND) term.

  * It is possible to use the same square more than once.

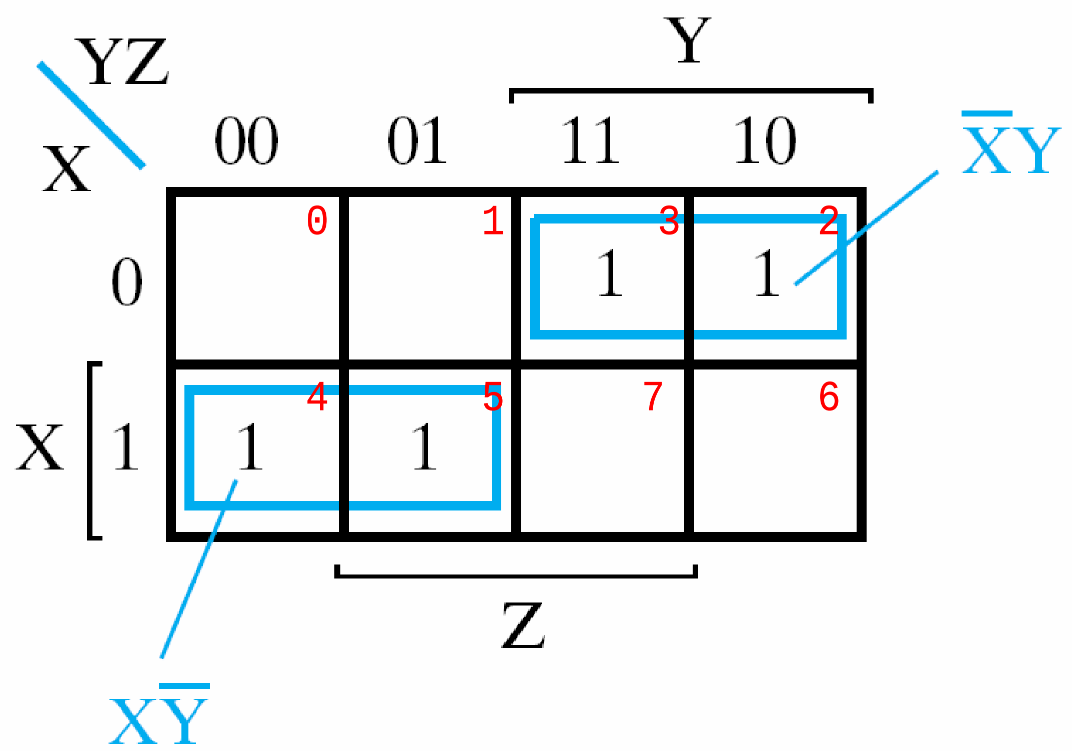
Example 2-3

- Example 2-3: Simplify the Boolean function

  $F(X, Y, Z) = \Sigma m(2,3,4,5)$

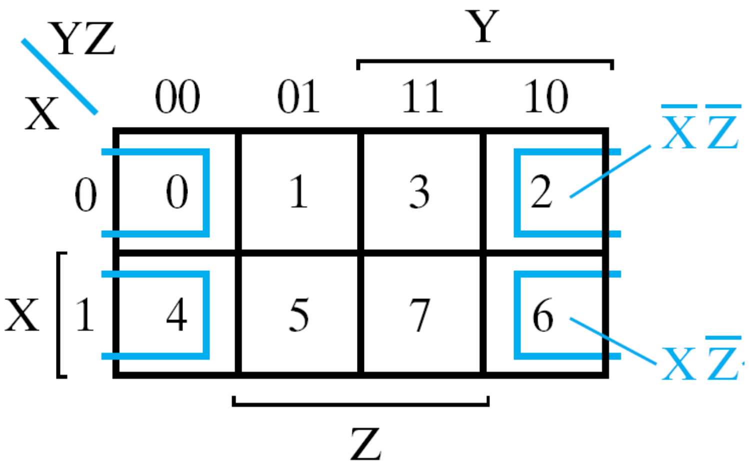
<Ans.>

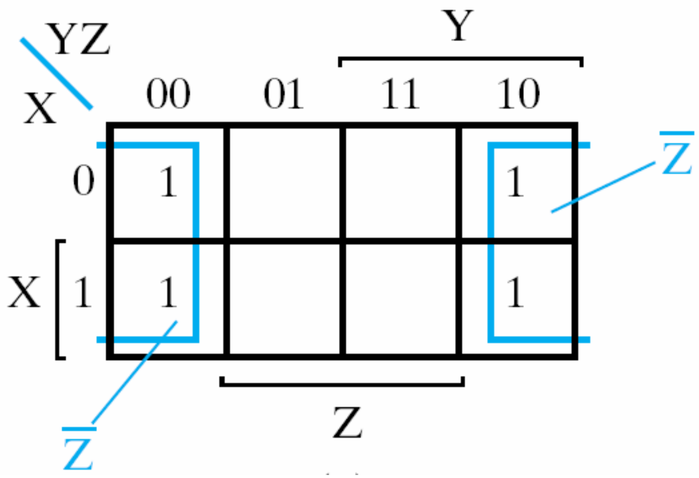
$$F = \overline{X}Y + X\overline{Y}$$



4-minterm Product terms

- Product terms using 4 minterms:
  - E.g.: F(X, Y, Z) = $\Sigma m(0,2,4,6)$



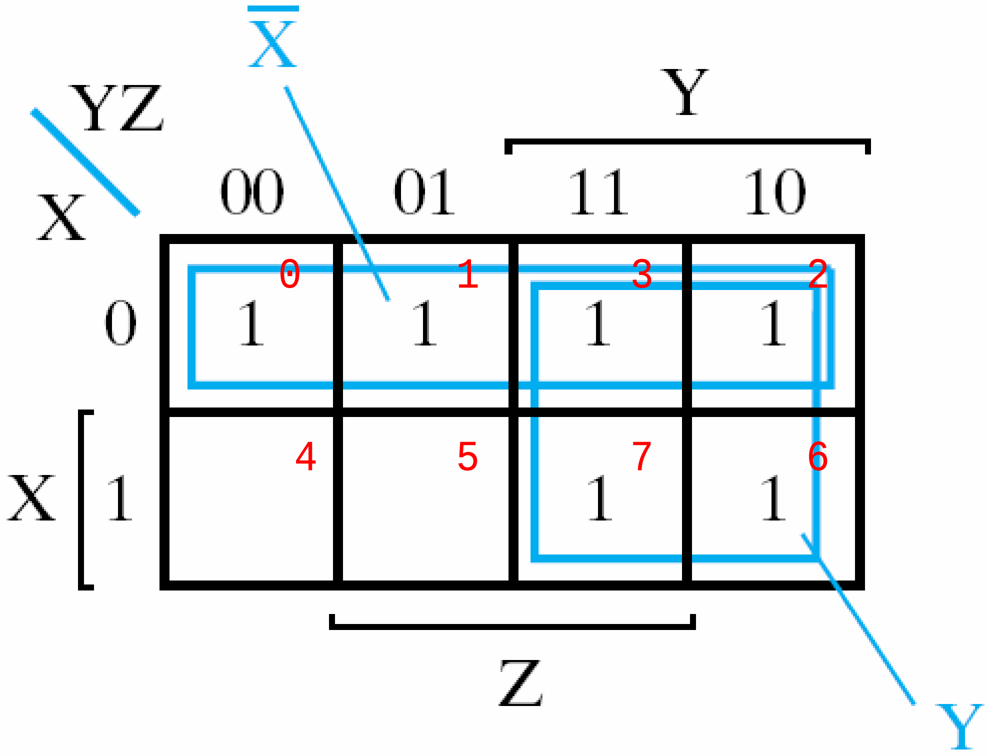


$$m_0 + m_2 + m_4 + m_6 = \overline{X}\,\overline{Y}\,\overline{Z} + \overline{X}Y\overline{Z} + X\overline{Y}\,\overline{Z} + XY\overline{Z}$$

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

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

– E.g.: F(X, Y, Z) = $\Sigma m(0,1,2,3,6,7)$



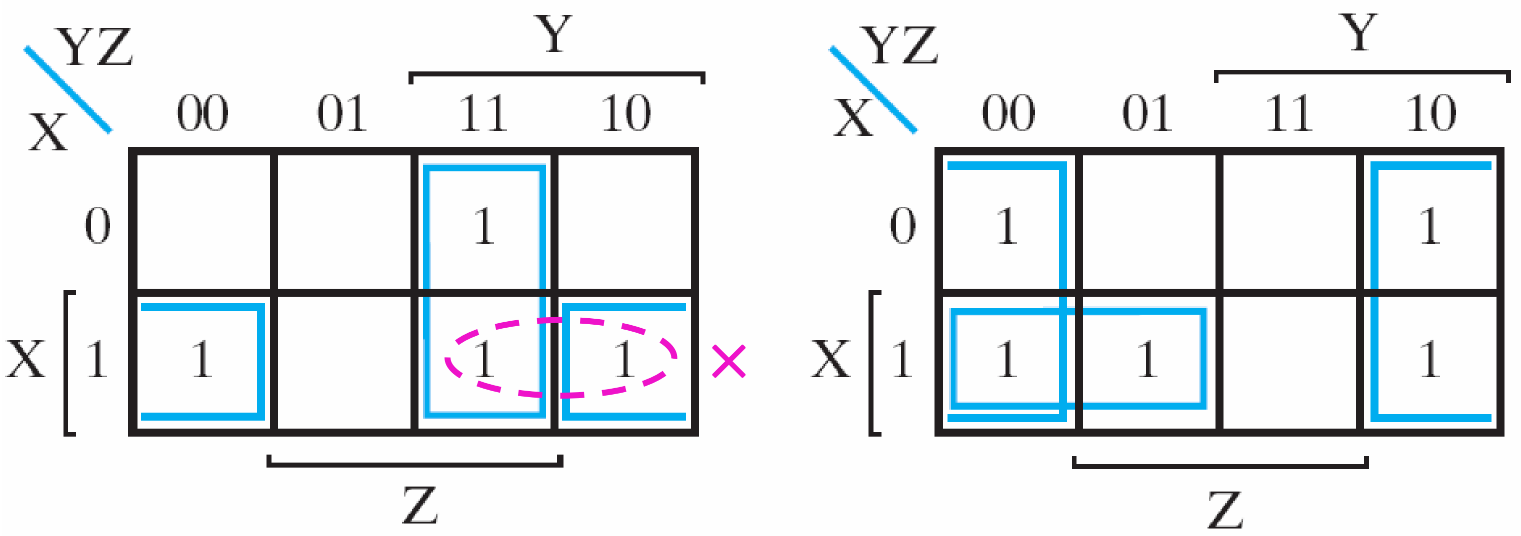
Example 2-4

- Example 2-4: Simplify the Boolean function

$$F_1(X,Y,Z) = \sum m(3,4,6,7)$$

$$F_2(X,Y,Z) = \sum m(0,2,4,5,6)$$

<Ans.>

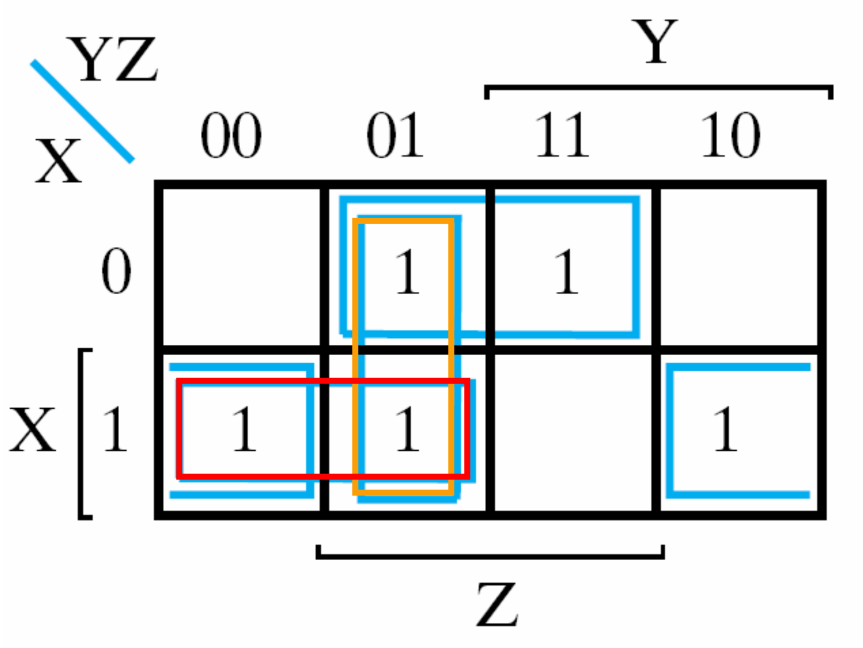


(a) $F_1(X,Y,Z) = \Sigma m(3,4,6,7) = YZ + X\overline{Z}$

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

Non-unique Optimized Expressions

- There may be alternative ways of combining squares to product equally optimized expressions:
  - E.g.: $F(X,Y,Z) = \sum m(1,3,4,5,6)$



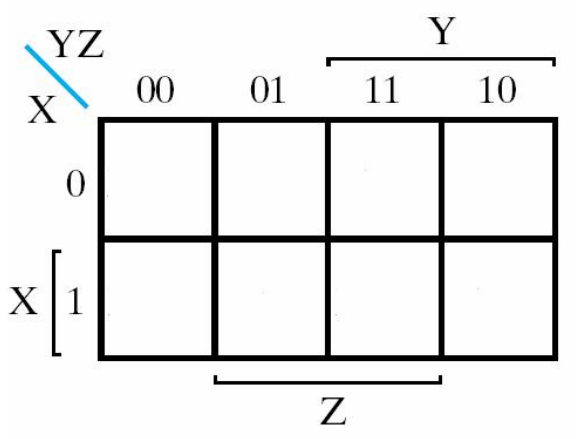
$$F(X,Y,Z) = \sum m(1,3,4,5,6)$$
$$= \overline{X}Z + X\overline{Z} + X\overline{Y}$$
$$= \overline{X}Z + X\overline{Z} + \overline{Y}Z$$

Simplifying Functions not Expressed as Sum-of-minterms Form

- If a function is not expressed as a sum of minterms:
  - use the map to obtain the minterms of the function & then simplify the function
  - E.g.: Given the Boolean function

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

<Ans.>

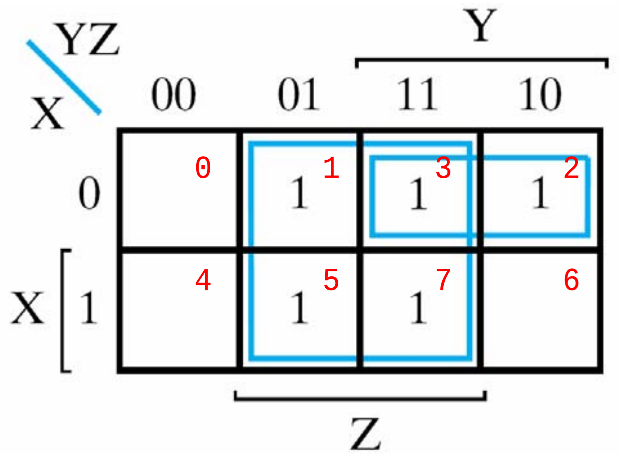


$$F = \overline{X}Z + \overline{X}Y + X\overline{Y}Z + YZ$$

$$\qquad 1,3 \qquad 2,3 \qquad 5 \qquad 3,7$$

$$= \sum m(1,2,3,5,7)$$

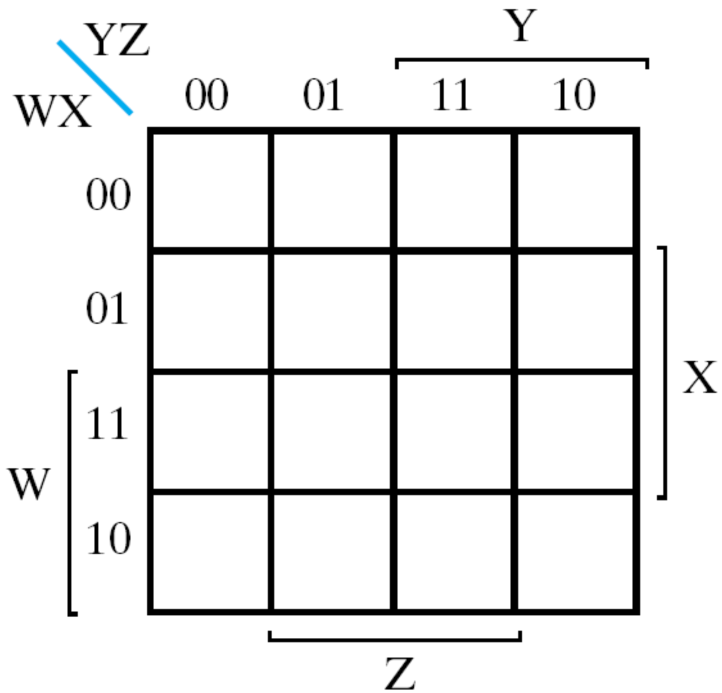
$$= Z + \overline{X}Y$$



(c) Four-Variable Map

- Four-variable map: 16 minterms ⇒ 16 squares

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}



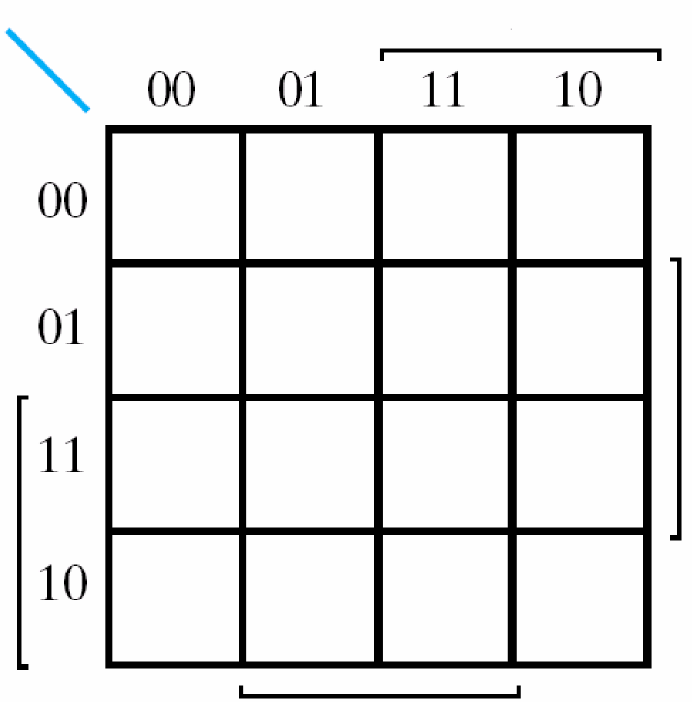
– Note: Each square has 4 adjacent squares.
  - The map is considered to lie on a surface w/ the top and bottom edges, as well as the right and left edges, touching each other to form adjacent squares.
  - E.g.: $m_8 \rightarrow m_0, m_9, m_{10}, m_{12}$

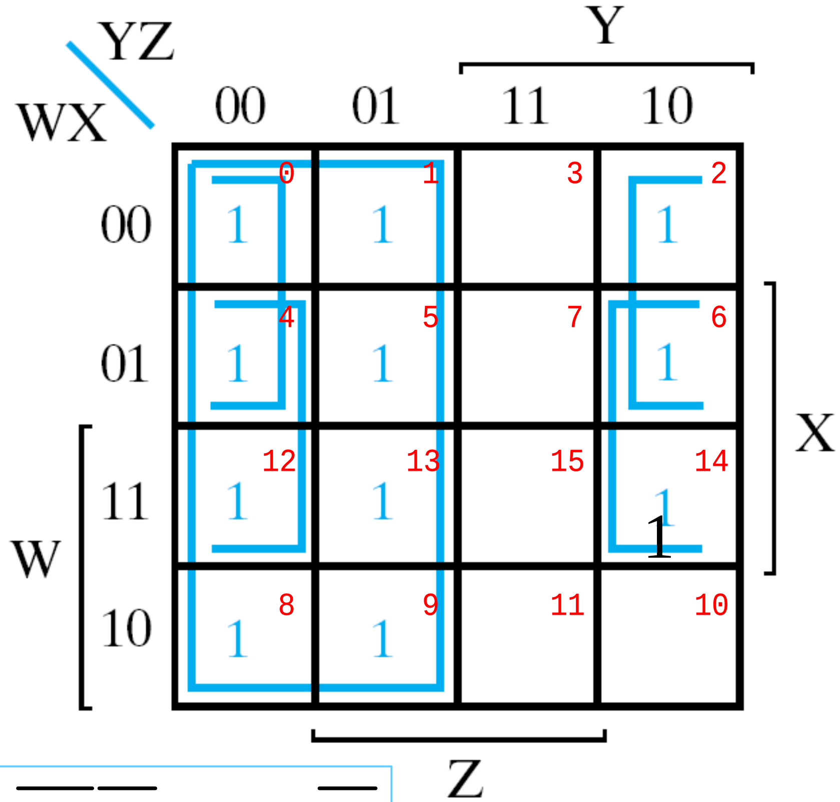
Example 2-5

- Simplify the Boolean function

$$F(W, X, Y, Z) = \Sigma(0,1,2,4,5,6,8,9,12,13,14)$$

<Ans.>





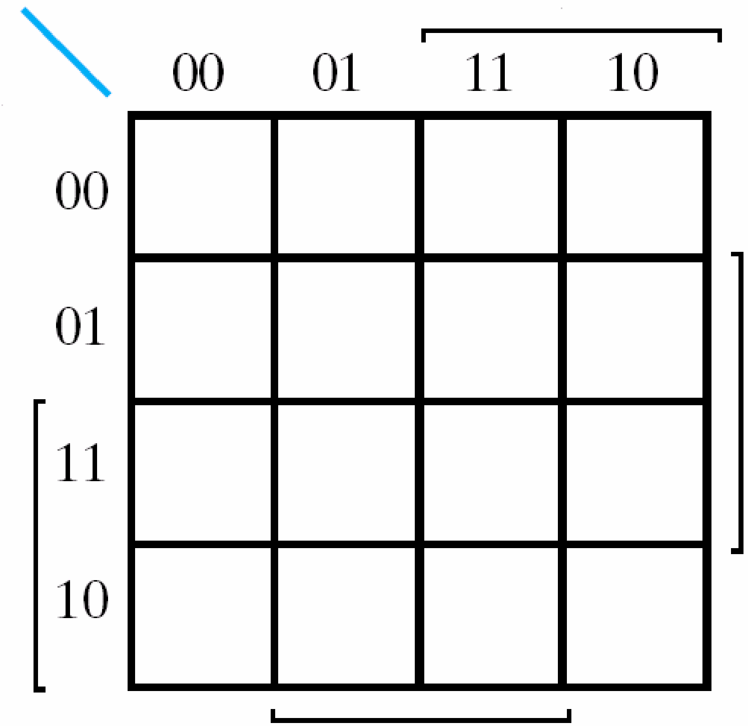
$$\boxed{F = \overline{Y} + \overline{W}\,\overline{Z} + X\,\overline{Z}}$$

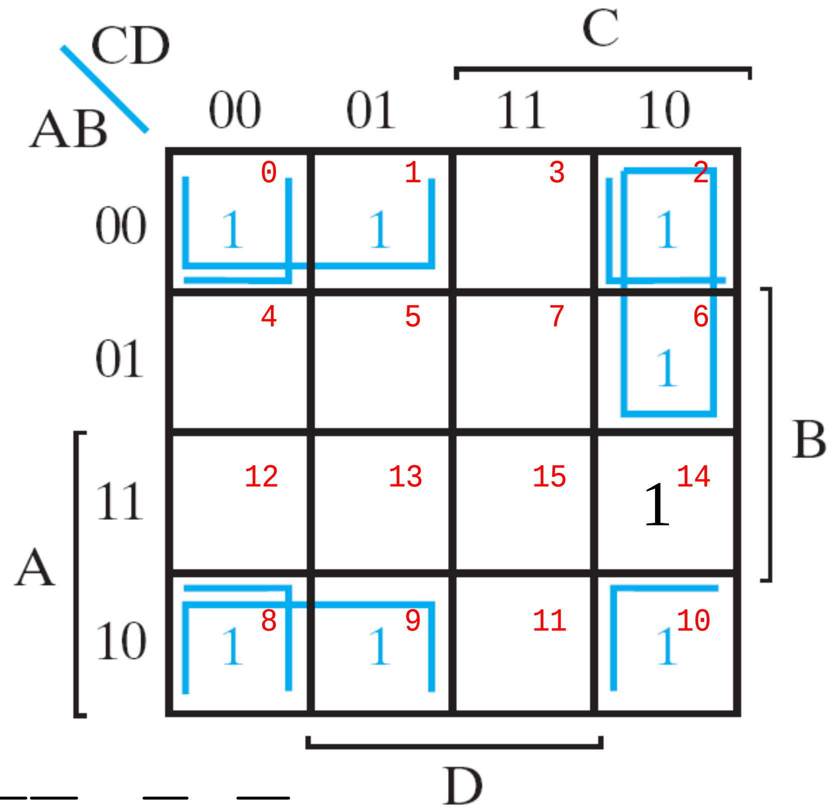
Example 2-6

- Simplify the Boolean function

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

<Ans.>

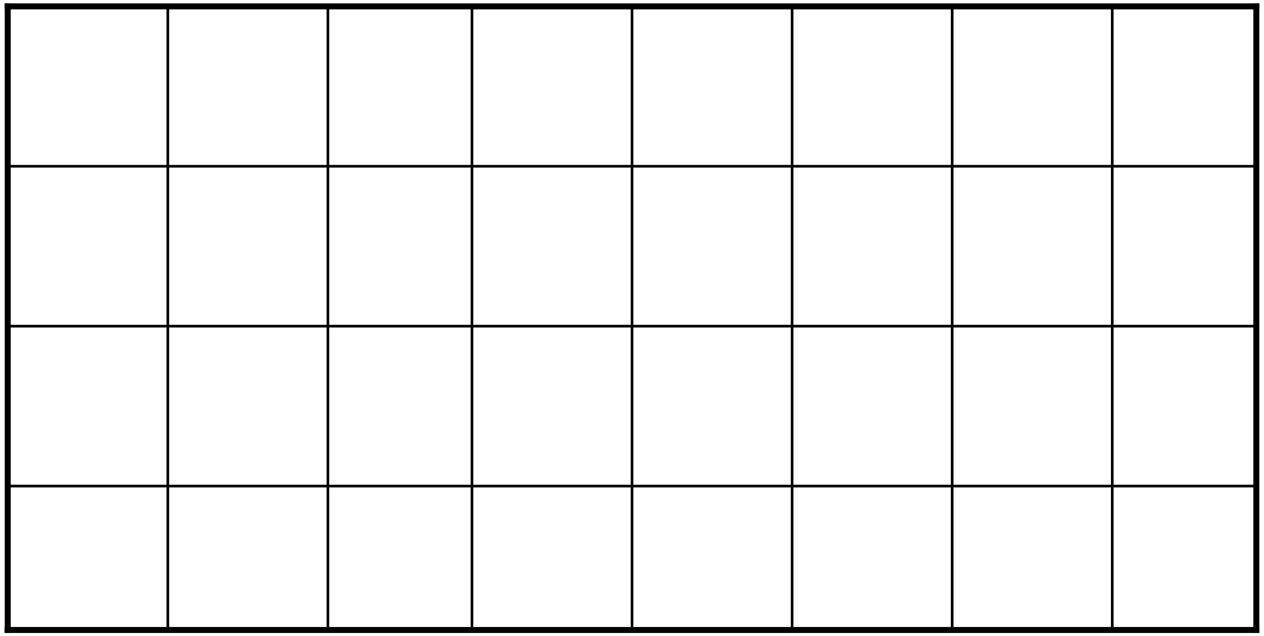




$$F = \overline{B}\,\overline{D} + \overline{B}\,\overline{C} + \overline{A}C\overline{D}$$

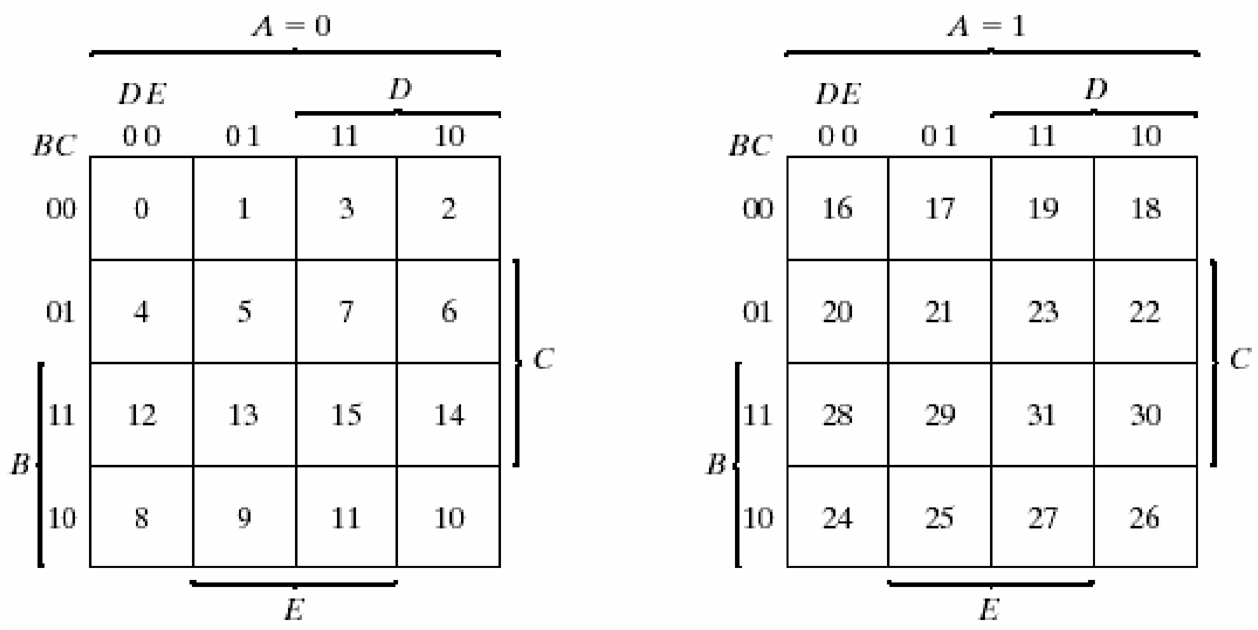
(d) Five-Variable Map (補充資料)

- Five-variable map: 32 minterms ⇒ 32 squares



  - Each square has 5 adjacent squares.

Alternatives: Five-variable map



- Each square has 5 adjacent squares.
  - Consider the two half maps as being one on top of the other.
  - Any 2 squares that fall one over the other are considered adjacent.
  - E.g.: $m_8 \rightarrow m_0, m_9, m_{10}, m_{12}, m_{24}$

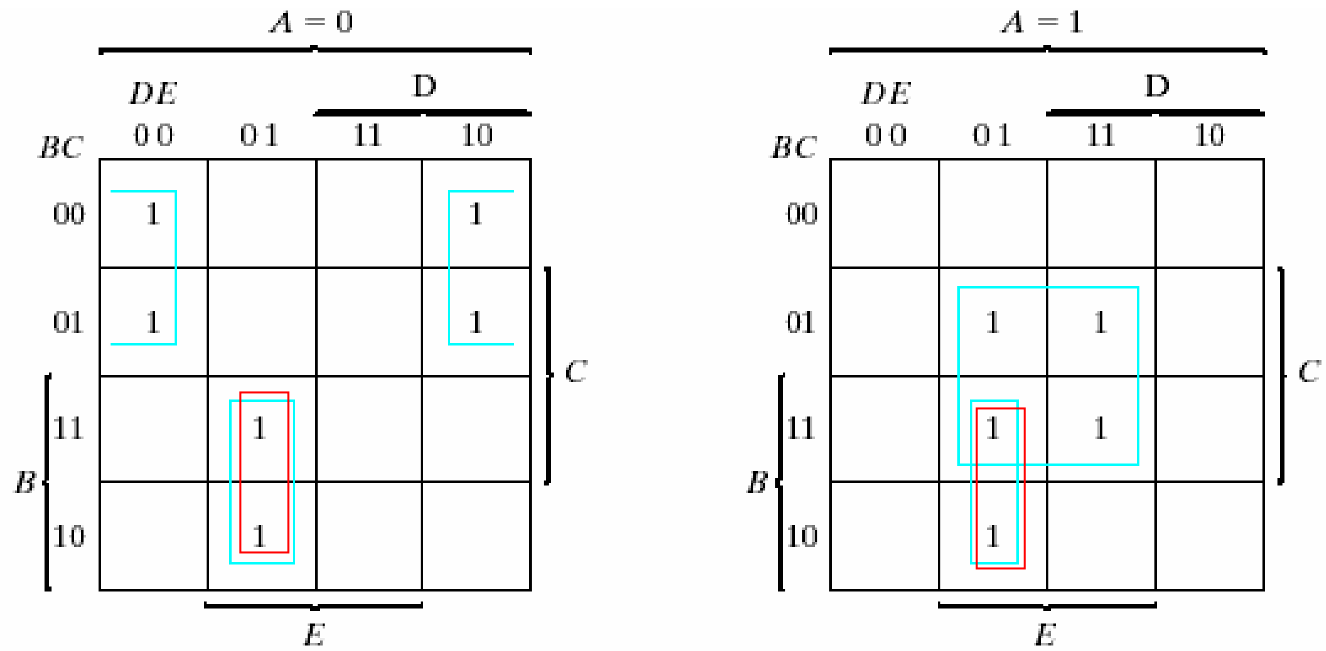
Six-variable map: 64 minterms $\Rightarrow$ 64 squares

Example

- Simplify the Boolean function

$$F(A,B,C,D,E) = \Sigma(0,2,4,6,9,13,21,23,25,29,31)$$

<Ans.>



F = A′B′E′ + BD′E + ACE

- Relationship b/t the # of adjacent squares and the # of literals in the term:
  - Any 2^k adjacent squares, for k = o, 1, …, n, in an n-variable map, will represent an area that gives a term of $n - k$ literals.

	Number of Adjacent Squares	Number of Literals in a Term in an n-variable Map			
K	2^k	$n = 2$	$n = 3$	$n = 4$	$n = 5$
0	1	2	3	4	5
1	2	1	2	3	4
2	4	0	1	2	3
3	8		0	1	2
4	16			0	1
5	32				0

- Maps for more than 4 variables are not as simple to use:
  - Employ computer programs specifically written to facilitate the simplification of Boolean functions w/ a large # of variables.

⇒ 補充 Quine-McCluskey Method (p.7-87~7-99)

2-5 Map Manipulation

- When choosing adjacent squares in a map:
  - Ensure that all the minterms of the function are covered when combining the squares.
  - Minimize the # of terms in the expression.
    - avoid any redundant terms whose minterms are already covered by other terms

A. Essential Prime Implicants

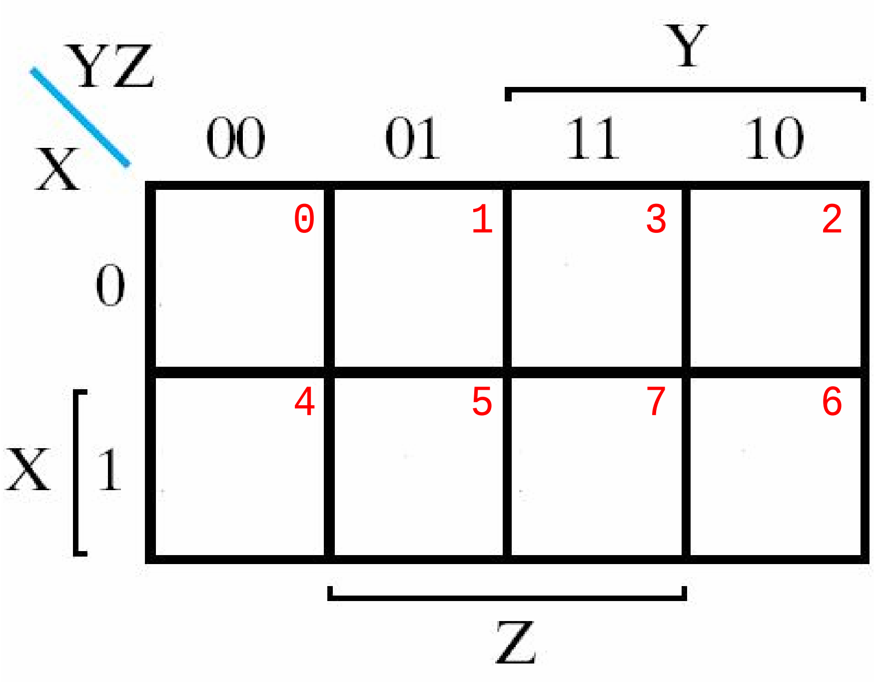
- *Implicant:*
  - A product term is an implicant of a function if the function has the value 1 for all minterms of the product term.
- *Prime implicant*: PI
  - a product term obtained by combining the max. possible # of adjacent squares in the map
- *Essential prime implicant*: EPI, must be included
  - If a minterm in a square is covered by only one PI, that PI is said to be essential.
    - Look at each square marked w/ a 1 and check the # of PIs that cover it.

Example: p.2-57

- Find the PIs and EPIs of the Boolean function

$$F(X, Y, Z) = \Sigma m(1,3,4,5,6)$$

<Ans.>



$$F(X,Y,Z) = \sum m(1,3,4,5,6)$$
$$= \overline{X}Z + X\overline{Z} + X\overline{Y}$$
$$= \overline{X}Z + X\overline{Z} + \overline{Y}Z$$

4 PIs: $\overline{X}Z,\ X\overline{Z},\ X\overline{Y},\ \overline{Y}Z$

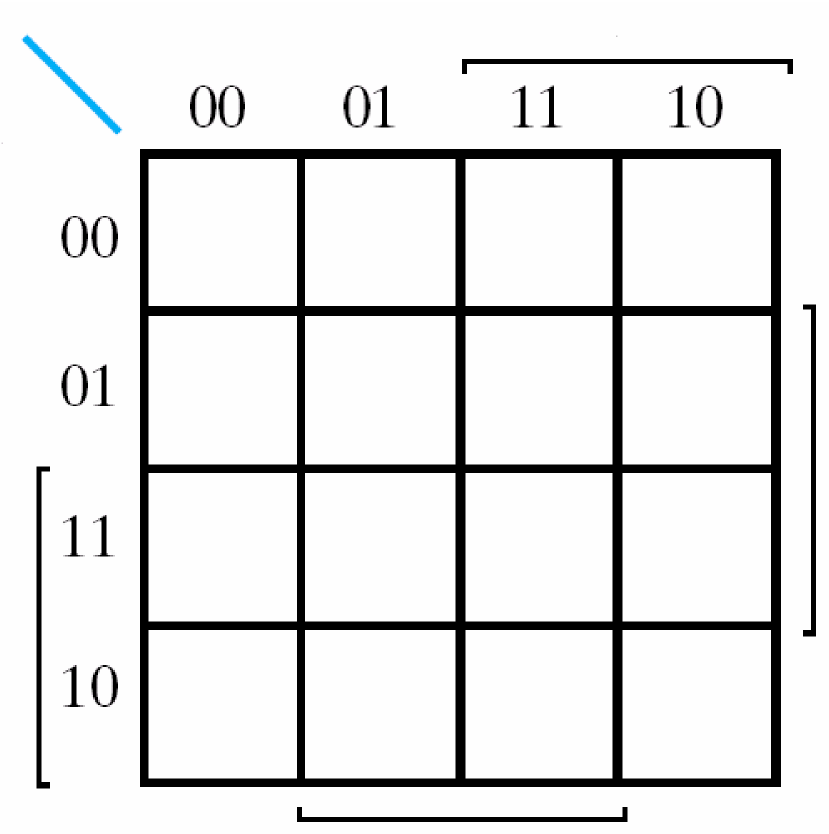
2 EPIs: $\overline{X}Z\ (\underset{*}{m_3}),\ X\overline{Z}\ (\underset{*}{m_6})$

Example

- Find the PIs and EPIs of the Boolean function

$$F(A,B,C,D) = \Sigma(0,2,3,5,7,8,9,10,11,13,15)$$

<Ans.>



6 PIs: BD, B'D',
CD, B'C,
AD, AB'

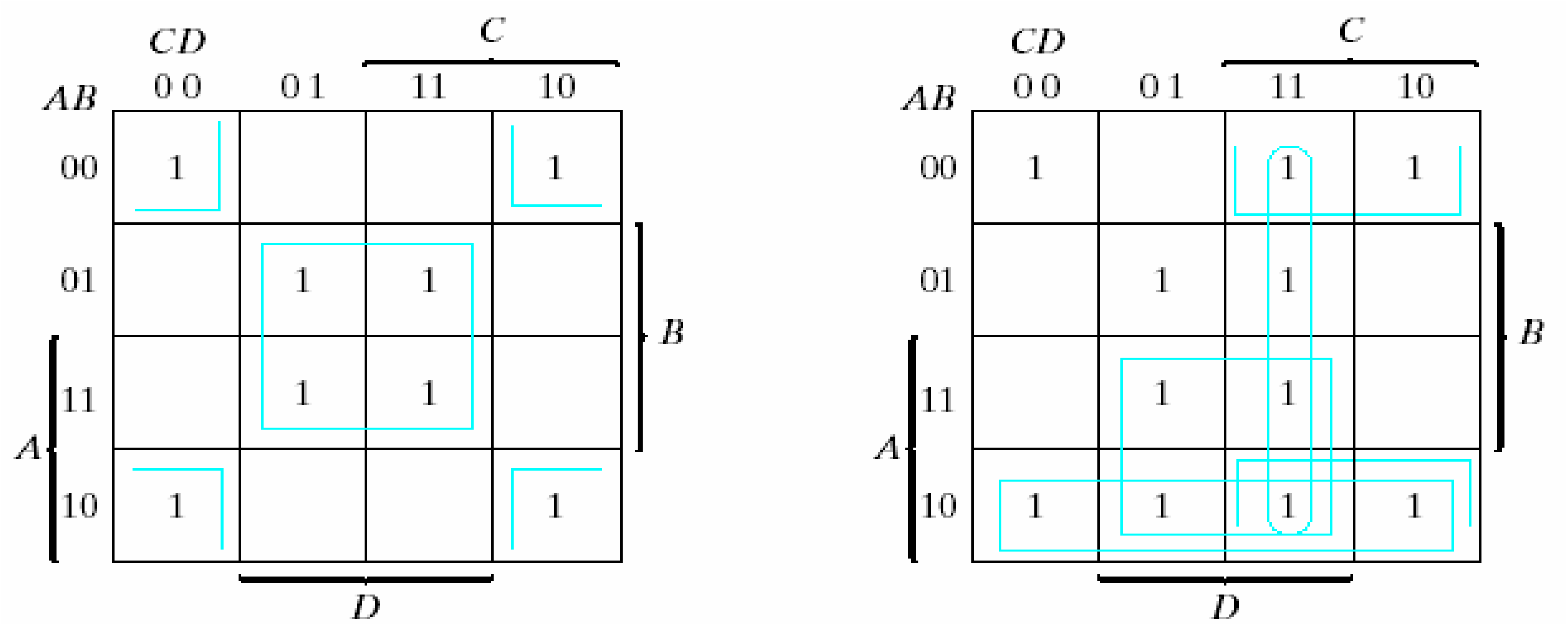
2 EPIs: BD (m5),
B'D' (m0)

Example (Cont’)

- Find the PIs and EPIs of the Boolean function

$$F(A,B,C,D) = \Sigma(0,2,3,5,7,8,9,10,11,13,15)$$

<Ans.>



(a) Essential prime implicants BD and B′D′

(b) Prime implicants CD, B′C AD, and AB′

6 PIs: BD, B'D', CD, B'C, AD, AB'

2 EPIs: BD (m_5), B'D' (m_0)

B. Finding the Simplified Expression

- Procedure for finding the simplified expression from the map: (SoP form)
  i. Determine all PIs.
  ii. The simplified expression is obtained from the logical sum of all the EPIs plus other PIs that may be needed to cover any remaining minterms not covered by the EPIs.

  – There may be more than one expression that satisfied the simplification criteria.

Example 2-7

- Simplify the Boolean function

$$F(A,B,C,D) = \Sigma(1,3,4,5,6,7,12,14)$$

<Ans.>

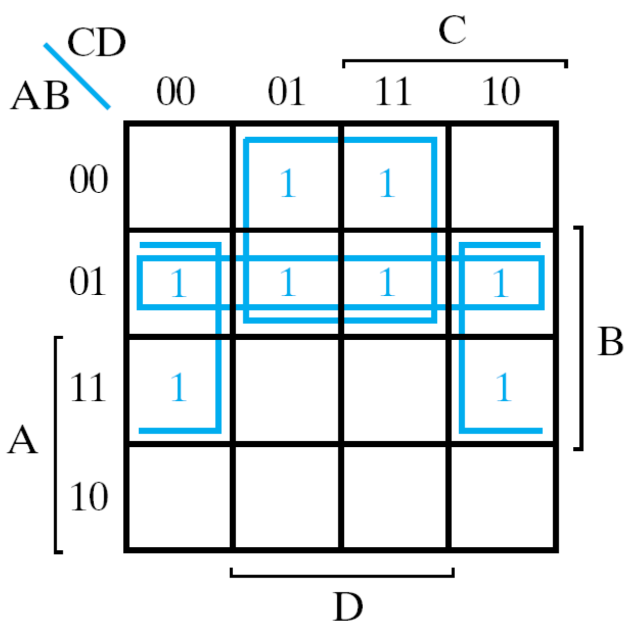
PIs: A'D, BD', A'B

EPIs: A'D (m_1, m_3), BD' (m_{12}, m_{14})

$\rightarrow m_1, m_3, m_5, m_7, m_4, m_6, m_{12}, m_{14}$

$\rightarrow m_1, m_3, m_4, m_5, m_6, m_7, m_{12}, m_{14}$

$$\Rightarrow F = \overline{A}D + B\overline{D}$$



Example 2-8

- Simplify the Boolean function

$$F(A,B,C,D) = \Sigma(0,5,10,11,12,13,15)$$

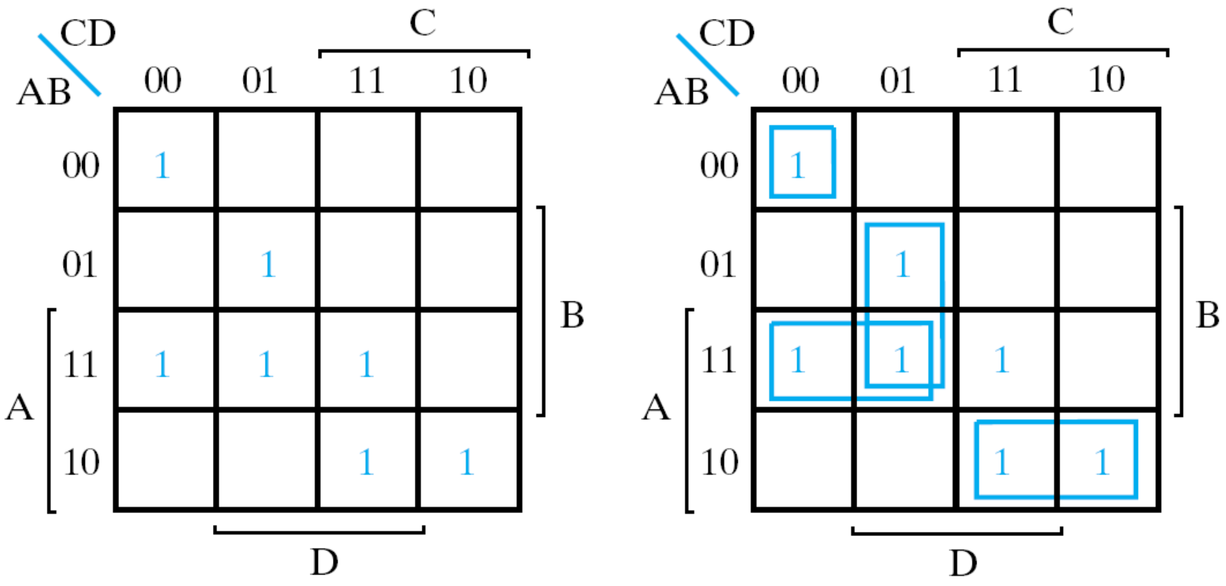
<Ans.>

EPIs: $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$

$B\overline{C}D$

$AB\overline{C}$

$A\overline{B}C$



(a) Plotting the minterms (b) Essential prime implicants

$\Rightarrow$ Combine PIs that contains m_{15}

$$\Rightarrow \; F = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + B\overline{C}D + AB\overline{C} + A\overline{B}C + \begin{cases} ACD \\ or \\ ABD \end{cases}$$

Selection Rule of Nonessential PIs

- Selection rule of Nonessential PIs:
  - Minimize the overlap among PIs as much as possible.
  - Make sure that each PI selected includes at least 1 minterm not included in any other PI selected.
    - It results in a simplified, although not necessarily minimum cost, SoP expression.

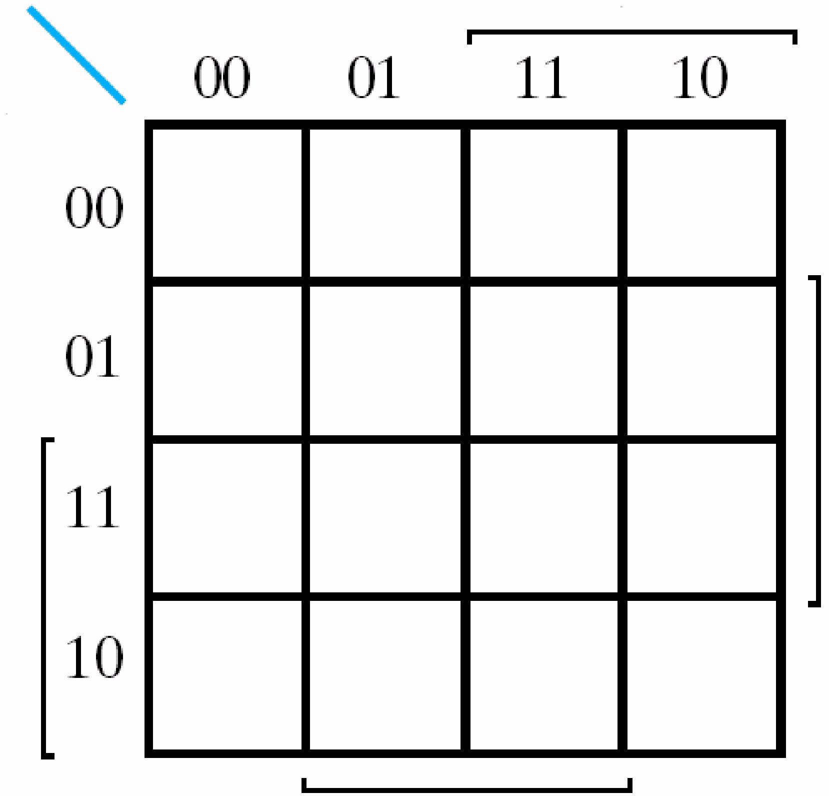
Example 2-9

- Simplify the Boolean function

  F(A,B,C,D) = Σm(0,1,2,4,5,10,11,13,15)

<Ans.>

EPI : $\overline{A}\,\overline{C}$ (m_1, m_4)



$$F(A,B,C,D)$$

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

1 2 3

Example: p.2-74

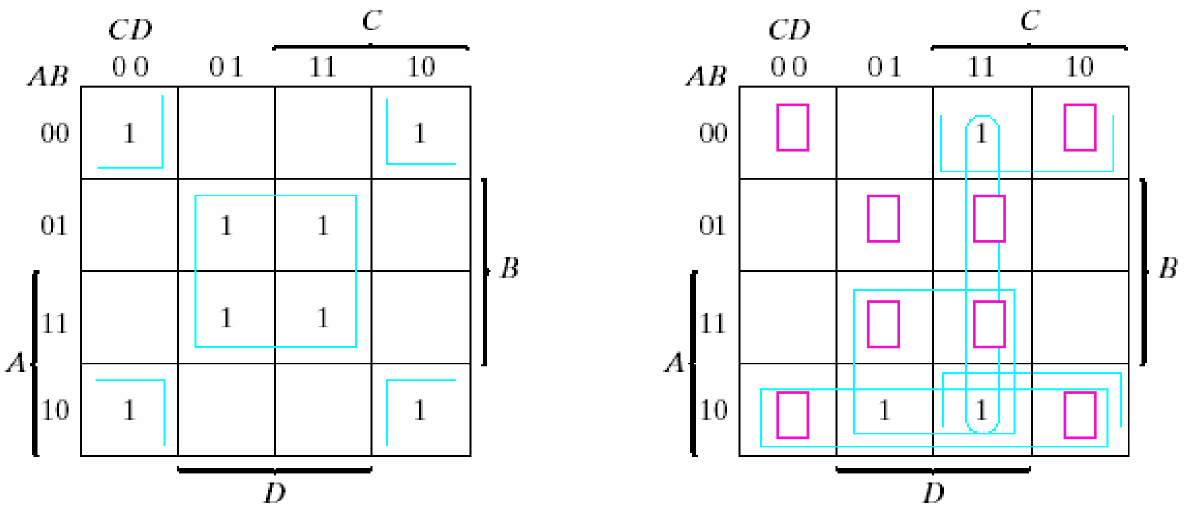
- Simplify the Boolean function

$$F(A,B,C,D) = \Sigma(0,2,3,5,7,8,9,10,11,13,15)$$

<Ans.>

6 PIs: BD, B'D', CD,
B'C, AD, AB'

2 EPIs: BD, B'D'



(a) Essential prime implicants BD and B'D'

(b) Prime implicants CD, B'C AD, and AB'

EPIs: BD, B'D' $\rightarrow$ $m_0, m_2, m_5, m_7, m_8, m_{10}, m_{13}, m_{15}$

$\Rightarrow$ Combine PIs that contains m_3, m_9, m_{11} (CD, B'C, AD, AB')

$\Rightarrow$ F = BD + B'D' + CD + AD

= BD + B'D' + CD + AB'

= BD + B'D' + B'C + AD

= BD + B'D' + B'C + AB'

C. Product-of-Sums (PoS) Optimization

- Approach 1:
  - Simplified F' in the form of sum of products
  - Apply DeMorgan's theorem $F = (F')'$

    F': sum of products $\Rightarrow$ F: product of sums
- Approach 2:
  - combinations of the maxterms of F

    i. A 0 is marked in each maxterm that represents the function.

    ii. Find possible adjacent 2^k squares and realize each set as a sum (OR) term, w/ variables being complemented.
  - E.g.:

$$M_0M_1 = (A+B+C+D)(A+B+C+D')$$
$$= (A+B+C)+(DD')$$
$$= A+B+C$$

AB \ CD	00	01	11	10
00	M_0	M_1	M_3	M_2
01	M_4	M_5	M_7	M_6
11	M_{12}	M_{13}	M_{15}	M_{14}
10	M_8	M_9	M_{11}	M_{10}

Example 3-8

- Simplify the Boolean function in (a) SoP and (b) PoS: F(A,B,C,D) = Σm(0,1,2,5,8,9,10)

<Ans.>

(a) F = B′D′ + B′C′ + A′C′D

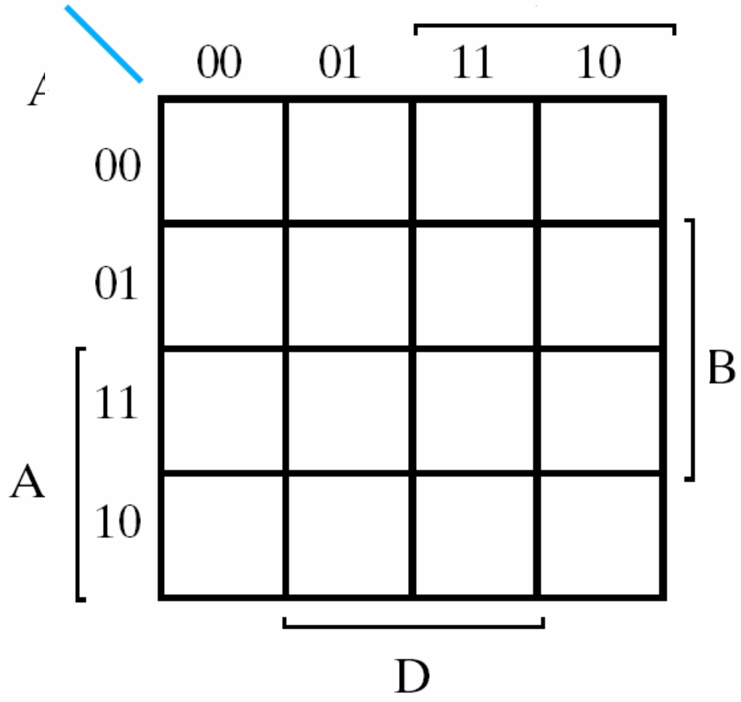
(b) Approach 1:

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

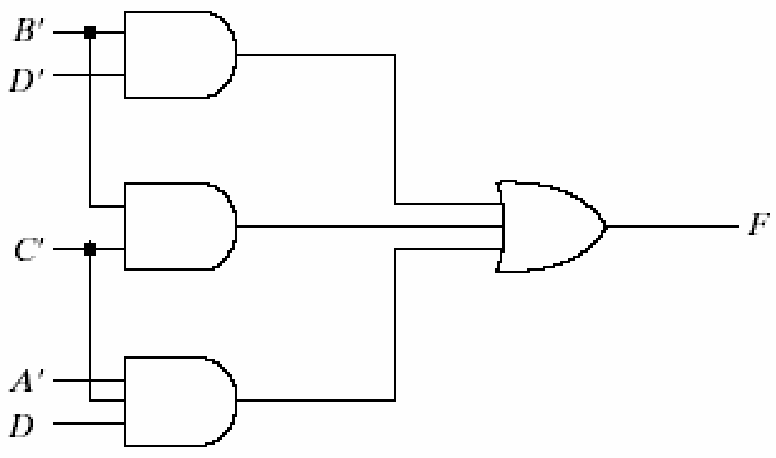
$$F = (\overline{F})' = (\overline{A} + \overline{B})(\overline{C} + \overline{D})(\overline{B} + D)$$

Approach 2:

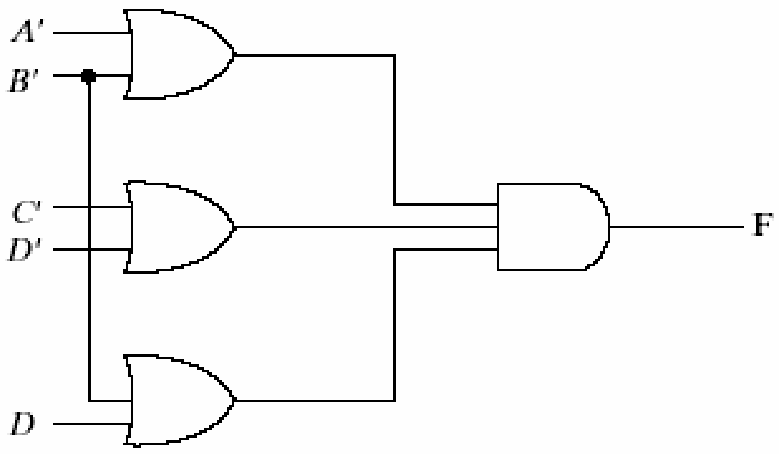
Think in terms of maxterms



– Gate implementation:



(a) F = B′D′ + B′C′ + A′C′D



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

- The implementation of a function in a standard form is said to be a two-level implementation.
  – Assumption: The input variables are directly available in their complement ⇒ Inverters are not needed.
- Determine which form will be best for a function.

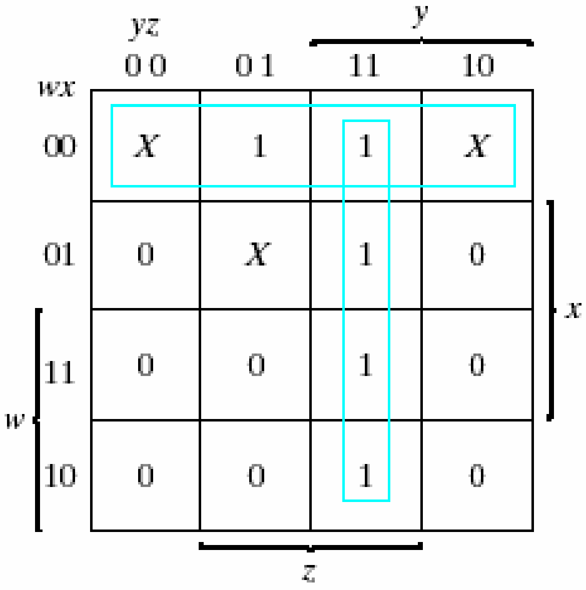
D. Don't-Care Conditions

- *Don't care condition*:
  - the unspecified minterms of a function
  - is represented by an $\times$
  - E.g.: A 4-bit decimal code has 6 combinations which are not used.
- *Incompletely specified function*:
  - has unspecified outputs for some input combinations
- Simplification of an incompletely specified function:
  - When choosing adjacent squares to simplify the function in the map, the $\times$'s may be assumed to be either 0 or 1, whichever gives the simplest expression.
  - An $\times$ need not be used at all if it does not contribute to covering a larger area.

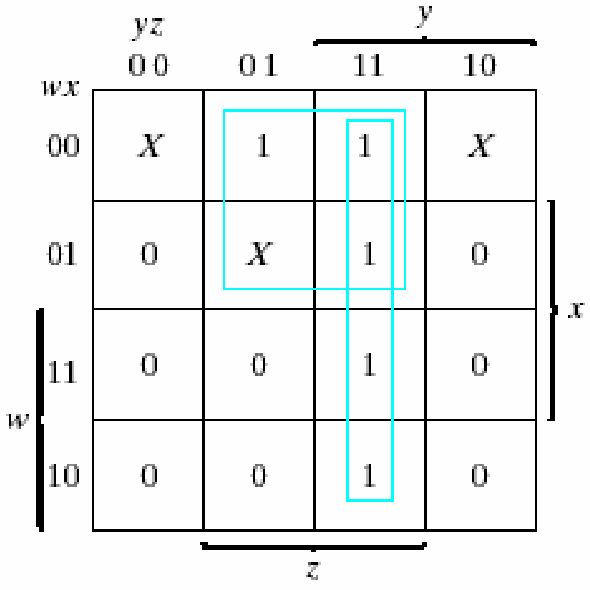
Example 2-11

- Simplify the Boolean function F $(w,x,y,z) = \Sigma m(1,3,7,11,15)$ which has the don't-care conditions d $(w,x,y,z) = \Sigma m(0,2,5)$.

<Ans.>



(a) $F = yz + w'x'$



(a) $F = yz + w'z$

* The outputs in a particular implementation of the function are only 0's and 1's.

(a) SOP: F $(w,x,y,z) = yz + w'x' = \Sigma m(0,1,2,3,7,11,15)$

F $(w,x,y,z) = yz + w'z = \Sigma m(1,3,5,7,11,15)$

(b) POS: F $(w,x,y,z) = z\,(w' + y) = \Sigma m(1,3,5,7,11,15)$

補充資料:

Quine-McCluskey Method

&

CAD Tools for Simplification

J.J. Shann

A. Quine-McCluskey Method (for SOP)

- Tabular method to systematically find all PIs

<u>Stage 1</u>: Find all prime implicants

Step 1: Fill Column 1 with minterm and don't-care condition indices.
Group by number of 1's.

Step 2: Apply Uniting Theorem ($xy + xy' = x$)
Compare elements of group w/ N 1's against those w/ N+1 1's.
Differ by one bit implies adjacent.
Eliminate variable and place in next column.
When used in a combination, mark with a check.
If cannot be combined, mark with a star.
These are the prime implicants (PI).

<u>Stage 2</u>: Find the minimum cover (SOP)

Stage 1: Finding All Prime Implicants (1/4)

$F(A,B,C,D) = \Sigma m(4,5,6,8,9,10,13) + \Sigma d(0,7,15)$

Stage 1: Find all prime implicants

Step 1: Fill Column 1 with minterm and don't-care condition indices. Group by number of 1's.

Implication Table	
Column 1	
0 **0000**	
4 **0100**	
8 **1000**	
5 **0101**	
6 **0110**	
9 **1001**	
10 **1010**	
7 **0111**	
13 **1101**	
15 **1111**	

Stage 1: Finding All Prime Implicants (2/4)

Step 1: Fill Column 1 with minterm and don’t-care condition indices.
Group by number of 1's.

Step 2: Apply Uniting Theorem
Compare elements of group w/ *N* 1's against those w/ *N*+1 1's.
Differ by one bit implies adjacent.
Eliminate variable and place in next column.

E.g., 0000 vs. 0100 yields 0-00
0000 vs. 1000 yields -000

When used in a combination, mark with a check.
If cannot be combined, mark w/ a star (i.e., PI).

Implication Table

Column 1	Column 2	
0000 √	0-00	
	-000	
0100 √		
1000 √	010-	
	01-0	
0101 √	100-	
0110 √	10-0	
1001		
1010 √	01-1	
	-101	
0111 √	011-	
1101 √	1-01	
1111 √	-111	
	11-1	

Repeat until no further combinations can be made.

Stage 1: Finding All Prime Implicants (3/4)

Step 1: Fill Column 1 with minterm and don't-care condition indices. Group by number of 1's.

Step 2: Apply Uniting Theorem
Compare elements of group w/ *N* 1's against those w/ *N*+1 1's.
Differ by one bit implies adjacent.
Eliminate variable and place in next column.

E.g., 0000 vs. 0100 yields 0-00
0000 vs. 1000 yields -000

When used in a combination, mark with a check.
If cannot be combined, mark w/ a star (i.e., PI).

Repeat until no further combinations can be made.

Implication Table

Column 1	Column 2	Column 3
0000 √	0-00 *	01- - *
	-000 *	
0100 √	010- √	-1-1 *
1000 √	01-0 √	
	100- *	
0101 √	10-0 *	
0110 √		
1001	01-1 √	
1010 √	-101 √	
	011- √	
0111 √	1-01 *	
1101 √		
	-111 √	
1111 √	11-1 √	

Stage 1: Finding All Prime Implicants (4/4)

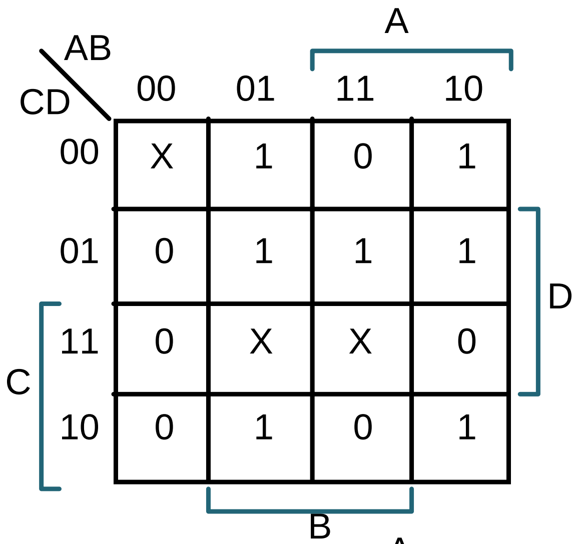
Implication Table

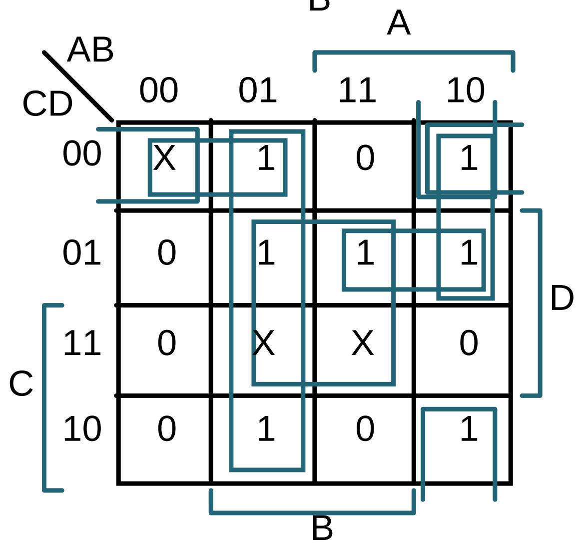
Column 1	Column 2	Column 3
0000 √	0-00 *	01- - *
	-000 *	
0100 √		-1-1 *
1000 √		
	010- √	
	01-0 √	
0101 √	100- *	
0110 √	10-0 *	
1001		
1010 √	01-1 √	
	-101 √	
0111 √	011- √	
1101 √	1-01 *	
1111 √	-111 √	
	11-1 √	

Prime Implicants:

- 0-00 = A' C' D'
- -000 = B' C' D'
- 100- = A B' C'
- 10-0 = A B' D'
- 1-01 = A C' D
- 01- - = A' B
- -1-1 = B D

Find all PIs by using K-map:





$$F(A,B,C,D) = \Sigma m(4,5,6,8,9,10,13) + \Sigma d(0,7,15)$$

Prime Implicants:

0-00 = A' C' D'

-000 = B' C' D'

100- = A B' C'

10-0 = A B' D'

1-01 = A C' D

01- - = A' B

-1-1 = B D

Stage 2: Finding the Minimum Cover

Stage 2: Find smallest set of prime implicants that cover the minterms.

Recall that essential prime implicants must be in all covers.

Another tabular method → the prime implicant chart

Prime Implicants:

0-00 = A' C' D'

-000 = B' C' D'

100- = A B' C'

10-0 = A B' D'

1-01 = A C' D

01- - = A' B

-1-1 = B D

Prime Implicant Chart

$F(A,B,C,D) = \Sigma m(4,5,6,8,9,10,13) + \Sigma d(0,7,15)$

rows = prime implicants

columns = minterms only

(don't-care conditions are not included)

Place an "X" if minterm is covered by the PI.

	4	5	6	8	9	10	13
0,4(0–00)	X						
0,8(–000)				X			
8,9(100–)				X	X		
8,10(10–0)				X		X	
9,13(1–01)					X		X
4,5,6,7(01– –)	X	X	X				
5,7,13,15(–1–1)		X					X

(a) Initial prime implicant chart

Prime Implicants:

0-00 = A' C' D'

-000 = B' C' D'

100- = A B' C'

10-0 = A B' D'

1-01 = A C' D

01- - = A' B

-1-1 = B D

Prime Implicant Chart

If column has a single X,
then the PI associated w/ the row
is essential. (EPI)
It must appear in minimum cover.

	4	5	6	8	9	10	13
0,4(0–00)	X						
0,8(–000)				X			
8,9(100–)				X	X		
8,10(10–0)				X		X	
9,13(1–01)					X		X
4,5,6,7(01– –)	X	X	X				
5,7,13,15(–1–1)		X					X

(b) Essential prime implicants

Prime Implicant Chart

Eliminate all columns covered by EPI.

	4	5	6	8	9	10	13
0,4(0–00)	X						
0,8(–000)				X			
8,9(100–)				X	X		
8,10(10–0)				X		X	
9,13(1–01)					X		X
4,5,6,7(01– –)	X	X	X				
5,7,13,15(–1–1)		X					X

(c) Covered minterms

Prime Implicant Chart

Find minimum set of rows that cover the remaining columns

F = A B' D' + A C' D + A' B

	4	5	6	8	9	10	13
0,4(0–00)	X						
0,8(–000)				X			
8,9(100–)				X	X		
8,10(10–0)				X		X	
9,13(1–01)					X		X
4,5,6,7(01– –)	X	X	X				
5,7,13,15(–1–1)		X					X

(d) Final configuration

B. CAD Tools for Simplification

ESPRESSO Method

Problem with Quine-McCluskey:

the # of prime implicants grows rapidly with the # of inputs

Upper bound: $3^n/n$, where n is the number of inputs

Finding a minimum cover is NP-complete, i.e., a computational expensive process not likely to yield to any efficient algorithm.

Espresso: trades solution speed for minimality of answer

don't generate *all* prime implicants (Quine-McCluskey Stage 1)

judiciously select a subset of primes that still covers the ON-set

operates in a fashion not unlike a human finding primes in a K-map

Espresso Inputs and Outputs

F(A,B,C,D) = Σm(4,5,6,8,9,10,13) + d(0,7,15)

Espresso Input

```
.i 4            -- # inputs
.o 1            -- # outputs
.ilb a b c d    -- input names
.ob f           -- output name
.p 10           -- number of product terms
0100   1        -- A'BC'D'
0101   1        -- A'BC'D
0110   1        -- A'BCD'
1000   1        -- AB'C'D'
1001   1        -- AB'C'D
1010   1        -- AB'CD'
1101   1        -- ABC'D
0000   -        -- A'B'C'D' don't care
0111   -        -- A'BCD don't care
1111   -        -- ABCD don't care
.e              -- end of list
```

Espresso Output

```
.i 4
.o 1
.ilb a b c d
.ob f
.p 3
1-01   1
10-0   1
01--   1
.e
```

F = A C' D + A B' D' + A' B

2-6 Multiple-Level Circuit Optimization

- 2-level ckt optimization:
  - can reduce the cost of combinational logic ckts
- Multi-level ckts:
  - ckts w/ more than 2 levels
  - There are often additional cost saving available

Example

- E.g.: $G = ABC + ABD + E + ACF + ADF$

<Ans.>

2-level implementation: gate-input cost = 17

Multi-level implementation: distributive law

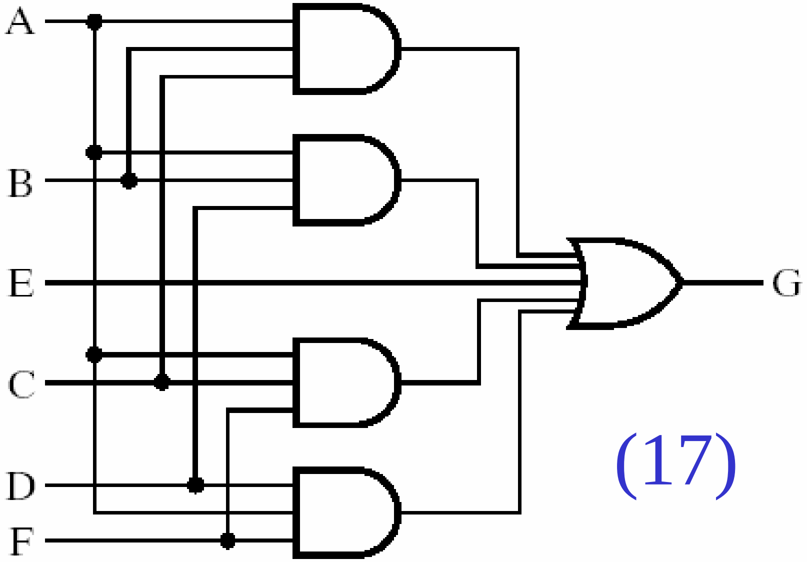
$$G = \underline{ABC + ABD} + E + \underline{ACF + ADF} \quad \rightarrow 17$$

$$= AB(\underline{C + D}) + E + AF(\underline{C + D}) \quad \rightarrow 13$$

$$= (\underline{A}B + \underline{A}F)(C + D) + E \quad \rightarrow 11$$

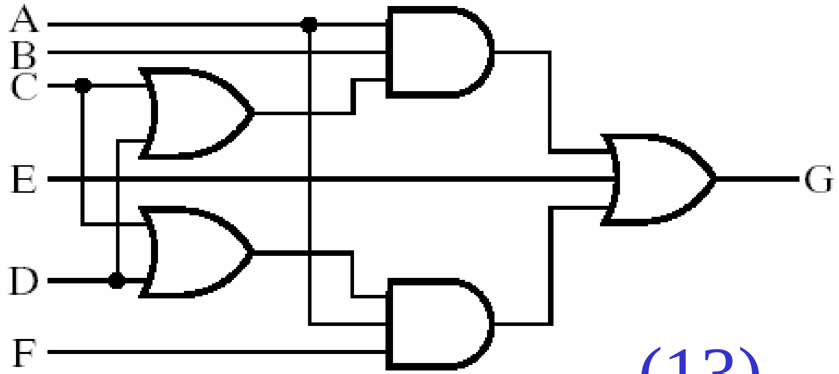
$$= A(B + F)(C + D) + E \quad \rightarrow 9$$

Gate input count (GIC)



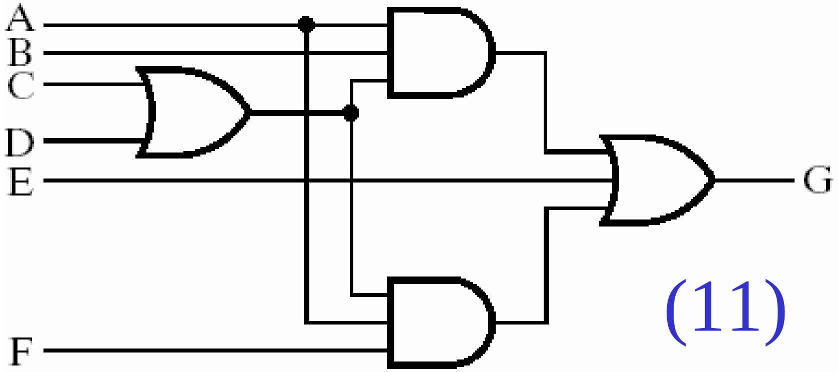
(17)

$$G = ABC + ABD + E + ACF + ADF$$



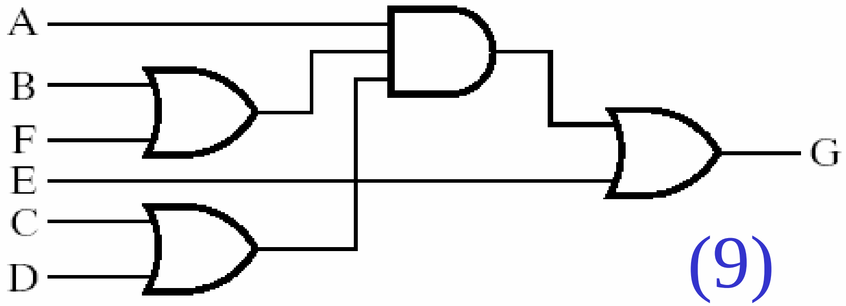
(13)

$$G = AB(C + D) + E + A(C + D)F$$



(11)

$$G = (AB + AF)(C + D) + E$$



(9)

$$G = A(B + F)(C + D) + E$$

Multiple-level Optimization

- Multiple-level ckt optimization (simplification):
  - is based on the use of a set of transformations that are applied in conjunction w/ cost evaluation to find a good, but not necessarily optimum solution.

Multilevel Optimization Transformations

- Transformations:
  - *Factoring*:
    - is finding a factored form from either a SoP or PoS expression for a function
  - *Decomposition*:
    - is the expression of a function as a set of new functions
  - *Extraction*:
    - is the expression of multiple functions as a set of new functions
  - *Substitution* of a function G into a function F:
    - is expressing F as a function of G and some or all of the original variables of F
  - *Elimination*: *flattening* or *collapsing*
    - is the inverse of substitution
    - function G in an expression for function F is replaced by the expression for G

for GIC ↓

for delay ↓

Example 2-12

- E.g. 2-12: Multilevel optimization transformations

$$G = A\overline{C}E + A\overline{C}F + A\overline{D}E + A\overline{D}F + BCD\overline{E}\,\overline{F}$$

$$H = \overline{A}BCD + ABE + ABF + BCE + BCF$$

<Ans.>

GIC

26 $G = A\overline{C}E + A\overline{C}F + A\overline{D}E + A\overline{D}F + BCD\overline{E}\,\overline{F}$

$= A(\overline{C}E + \overline{C}F + \overline{D}E + \overline{D}F) + BCD\overline{E}\,\overline{F}$ — factoring

$= A(\overline{C}(E+F) + \overline{D}(E+F)) + BCD\overline{E}\,\overline{F}$ — factoring

18 $= A(\overline{C} + \overline{D})(E+F) + BCD\overline{E}\,\overline{F}$ — factoring

$= A(\overline{C} + \overline{D})X_2 + BX_1\overline{E}\,\overline{F}$ — decomposition & substitution

14 $= A\overline{X_1}X_2 + BX_1\overline{X_2}$

$$\boxed{\begin{aligned} X_1 &= CD \\ X_2 &= E + F \end{aligned}}$$

$$G = A\overline{X_1}X_2 + BX_1\overline{X_2}$$
$$X_1 = CD$$
$$X_2 = E + F$$

(Cont'd)

$$H = \overline{A}BCD + ABE + ABF + BCE + BCF$$

factoring

$$= B(\overline{A}CD + AE + AF + CE + CF)$$

factoring

$$= B(\overline{A}CD + A(E+F) + C(E+F))$$

factoring

$$= B(\overline{A}(CD) + (A+C)(E+F))$$

$\Rightarrow$ extraction of G & H:

$$X_1 = CD$$
$$X_2 = E + F$$
$$X_3 = A + C$$

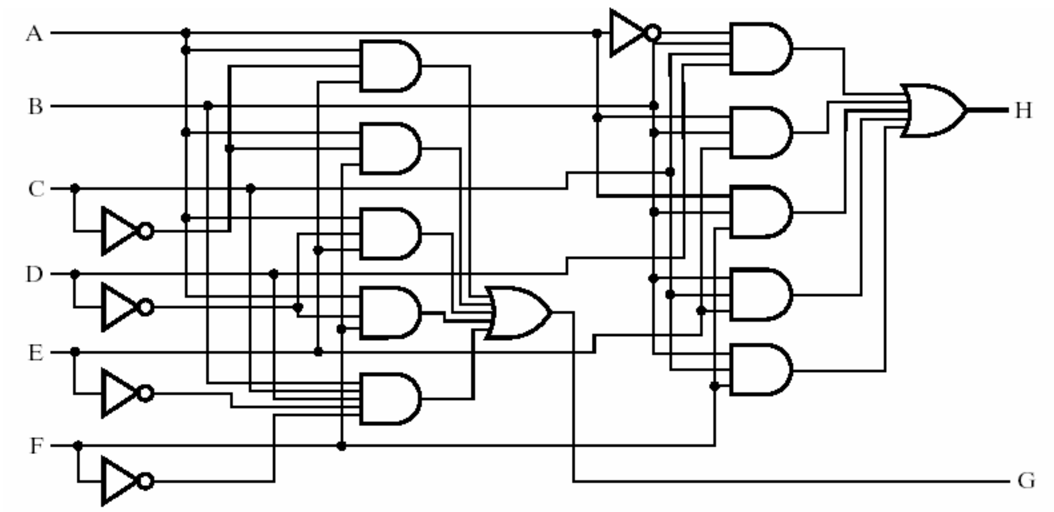
$\Rightarrow$ substitution:

$$G = A\overline{X_1}X_2 + BX_1\overline{X_2}$$
$$H = B(\overline{A}X_1 + X_3X_2)$$

$$G = A\bar{C}E + A\bar{C}F + A\bar{D}E + A\bar{D}F + BCD\bar{E}\bar{F}$$

$$H = \bar{A}BCD + ABE + ABF + BCE + BCF$$

GIC = 48



$$G = A\overline{X_1}X_2 + BX_1\overline{X_2}$$

$$H = B(\bar{A}X_1 + X_3X_2)$$

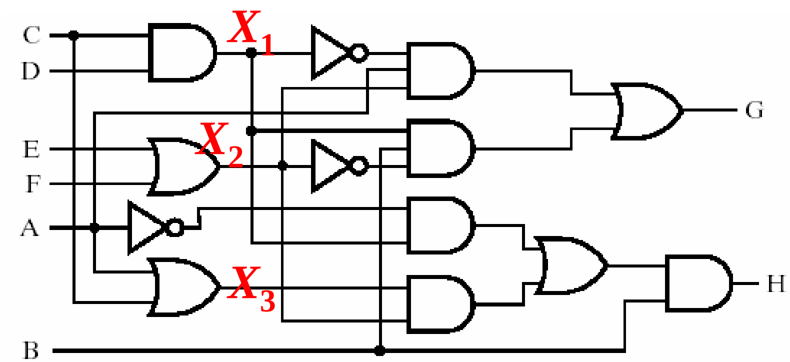
$$X_1 = CD$$

$$X_2 = E + F$$

$$X_3 = A + C$$

GIC = 25 (w/ shared terms)

(= 31 w/o shared terms)



- Key to successful transformations:
  - is the determination of the factors to be used in decomposition or extraction and choice of the transformation sequence to apply

Transformation for Delay Reduction

- Path delay:
  - the length of time it takes for a change in a signal to propagate down a path through the gates
- Critical path:
  - the longest path(s) through a ckt
- In a large proportion of designs, the length of the longest path(s) through the ckt is often constrained.

  $\Rightarrow$ The # of gates in series may need to be reduced.

  $\Rightarrow$ Elimination transformation

Elimination transform:

- replaces intermediate variables, *Xi*, w/ the expressions on their right hand sides or removes other factoring of some variables

  $\Rightarrow$ Reduces the # of gates in series
- Determination of factor or combination of factors should be eliminated:
  - look at the effect of gate input count
  - The increase in gate input count for the combinations of eliminations that reduce the problem path lengths by at lease one gate are of interest.

Example 2-13

- E.g.: Transformation for delay reduction
  - Assumption: Ignore the delay of NOT gate.

$$G = A\overline{X}_1X_2 + BX_1\overline{X}_2$$

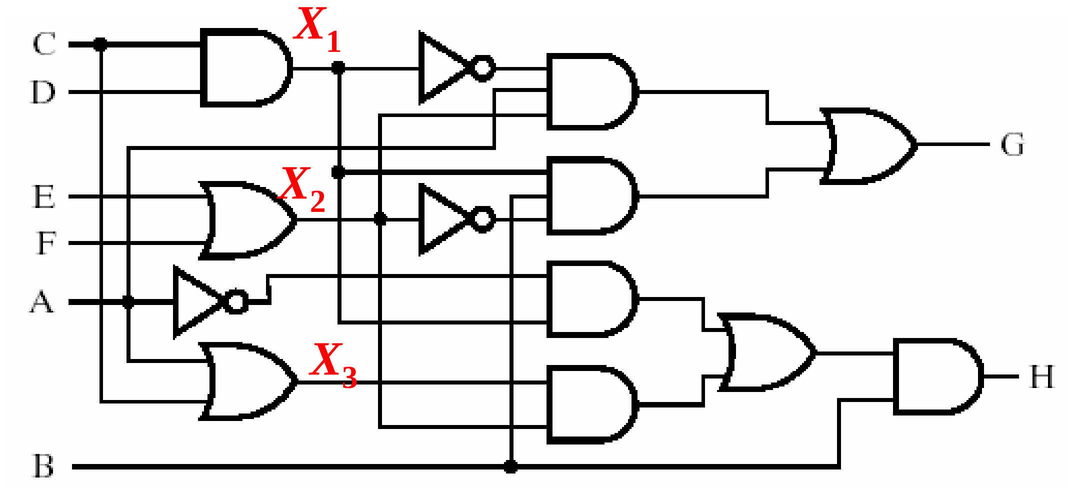
$$H = B(\overline{A}X_1 + X_3X_2)$$

$$X_1 = CD$$

$$X_2 = E + F$$

$$X_3 = A + C$$

GIC = 25



$$G = A\overline{X}_1 X_2 + BX_1\overline{X}_2$$

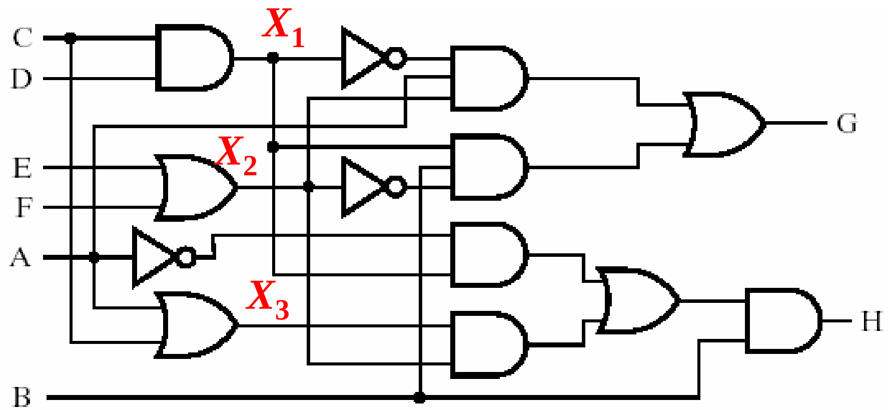
$$H = B(\overline{A}X_1 + X_3 X_2)$$

$$X_1 = CD$$

$$X_2 = E + F$$

$$X_3 = A + C$$

GIC = 25



<Ans.>

3 combinations of eliminations that reduce ≥ 1 gate delay:

i. removal of the factor B in H

$\Rightarrow$ GIC increases = 0

$$G = A\overline{X}_1 X_2 + BX_1\overline{X}_2$$

$$H = B\overline{A}X_1 + BX_3 X_2$$

$$X_1 = CD \quad X_2 = E + F \quad X_3 = A + C$$

ii. removal of the intermediate variables X_1, X_2, X_3

$\Rightarrow$ GIC increases = 12 (?)

$$G = A\overline{C}E + A\overline{C}F + A\overline{D}E + A\overline{D}F + BCD\overline{E}\,\overline{F} \quad 26$$

$$H = B(\overline{A}CD + AE + AF + CE + CF) \quad 18$$

iii. removal of B, X_1, X_2, X_3

$\Rightarrow$ GIC increases = 12 (?)

$$G = A\overline{C}E + A\overline{C}F + A\overline{D}E + A\overline{D}F + BCD\overline{E}\,\overline{F} \quad 26$$

$$H = \overline{A}BCD + ABE + ABF + BCE + BCF \quad 22$$

$$G = A\overline{X_1}X_2 + BX_1\overline{X_2}$$

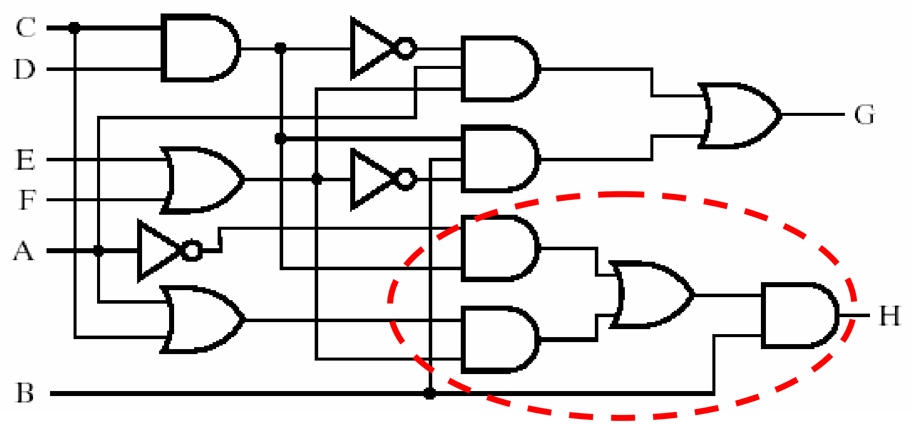
$$H = B(\overline{A}X_1 + X_3X_2)$$

$$X_1 = CD$$

$$X_2 = E + F$$

$$X_3 = A + C$$

GIC = 25

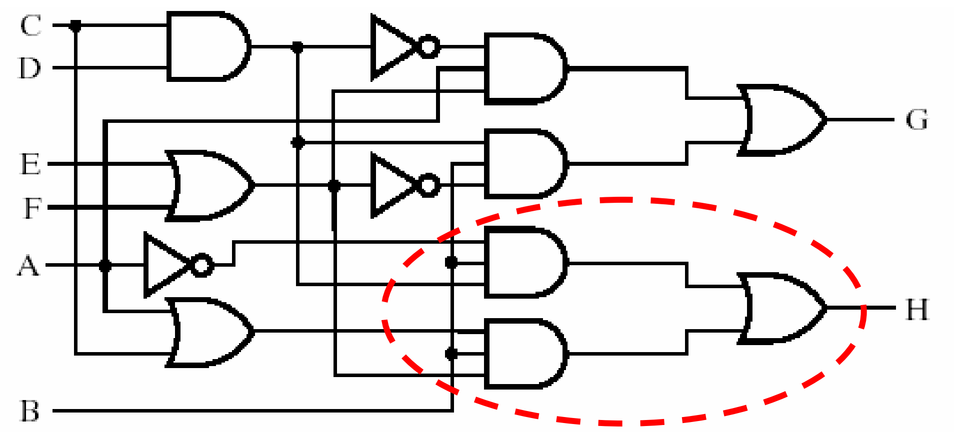


$\Downarrow$ Eliminate the factor B

$$G = A\overline{X_1}X_2 + BX_1\overline{X_2}$$

$$H = B\overline{A}X_1 + BX_3X_2$$

GIC = 25



2-7 Other Gate Types

- Basic logic operations: AND, OR, NOT
- All possible functions of *n* binary variables:

  n binary variables $\rightarrow 2^n$ distinct minterms

  $\rightarrow 2^{2^n}$ possible functions
- E.g.: $n = 2 \rightarrow 4$ minterms $\rightarrow 16$ possible functions

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

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 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

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

– The 16 functions can be subdivided into 3 categories:

1. 2 functions that produce a constant 0 or 1.
2. 4 functions w/ unary operations complement (NOT) and transfer.
3. 10 functions w/ binary operators that define eight different operations AND, OR, NAND, NOR, exclusive-OR, equivalence, inhibition, and implication.

Digital Logic Gates

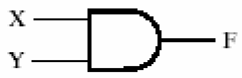
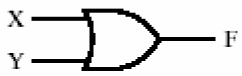
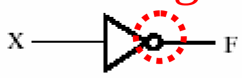
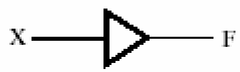
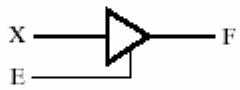
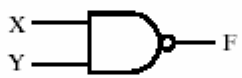
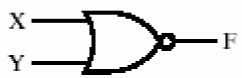
- Boolean expression:
  - AND, OR and NOT operations
  - It is easier to implement a Boolean function in these types of gates.
- Consider the construction of other types of logic gates:
  - the feasibility and economy of implementing the gate w/ electronic components
  - the basic properties of the binary operations
    - E.g.: commutativity, associativity, …
  - the ability of the gate to implement Boolean functions alone or in conjunction w/ other gates
  - the convenience of representing gate functions that are frequently used

Primitive Digital Logic Gates

- AND
- OR
- NOT (inverter)
- Buffer
  - amplify an electrical signal to permit more gates to be attached to the output or to decrease the time it takes for signals to propagate through the ckt
- 3-state Buffer (§2-9)
- NAND
- NOR

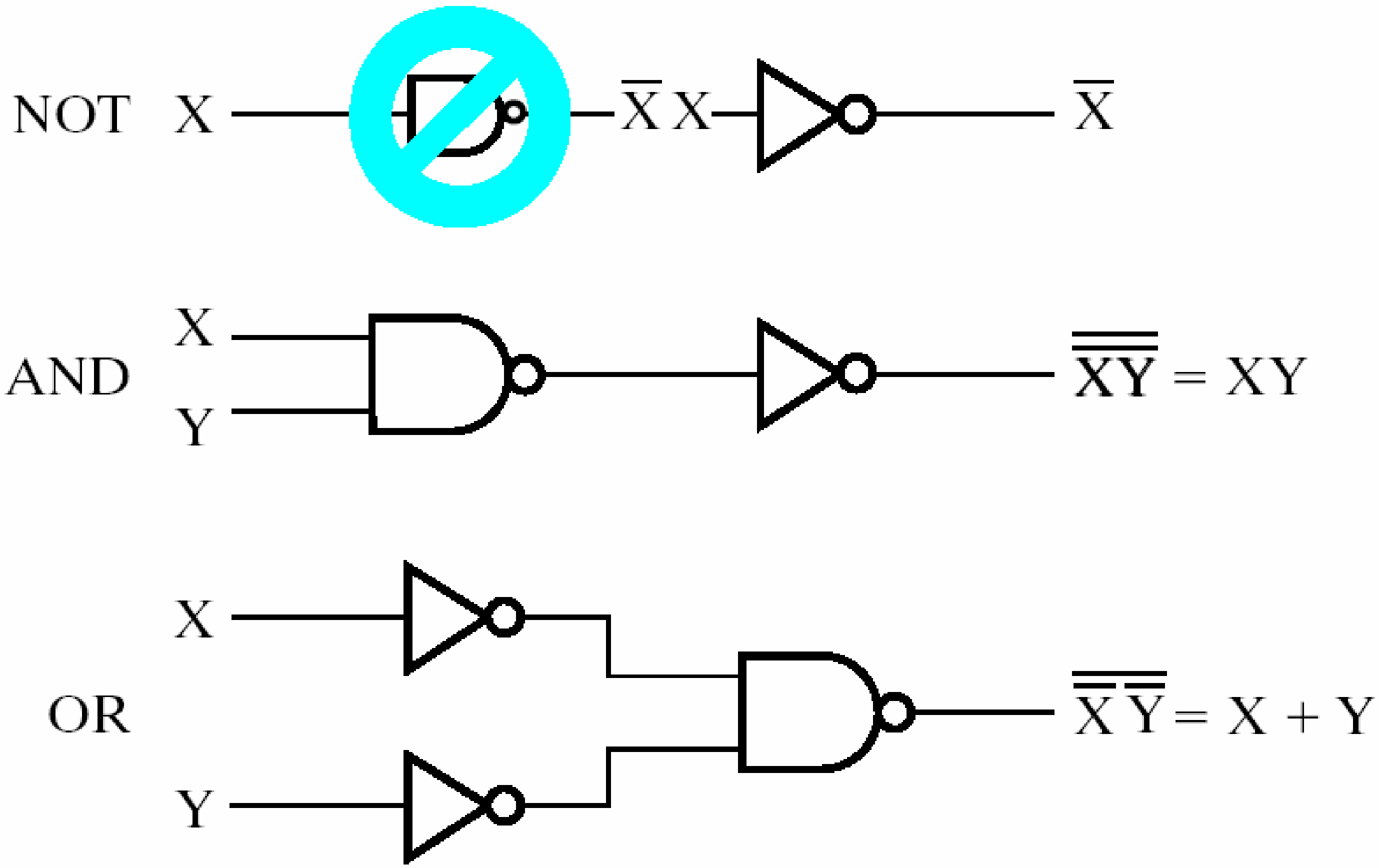
* NAND or NOR gates alone can implement any Boolean function

⇒ are much more widely used than AND and OR gates.

Name	Graphics Symbols: Distinctive shape	Graphics Symbols: Algebraic equation	Graphics Symbols: 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
NOT (inverter)	negation indicator 	$F = \overline{X}$	X | F 0 | 1 1 | 0
Buffer		$F = X$	X | F 0 | 0 1 | 1
3-State Buffer			E X | F 0 0 | Hi-Z 0 1 | Hi-Z 1 0 | 0 1 1 | 1
NAND		$F = \overline{X \cdot Y}$	X Y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = \overline{X + Y}$	X Y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0

- Universal gate:
  - a gate type that alone can be used to implement all Boolean functions
  - E.g.: NAND gate, NOR gate
- Proof of a universal gate:
  - Show that the logical ops of AND, OR, and NOT can be obtained w/ the universal gate only.
  - E.g.: Show that the NAND gate is a universal gate

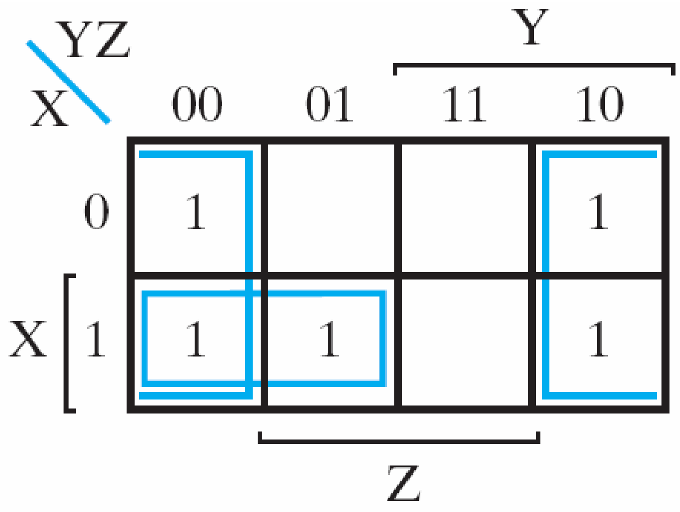
– E.g.: Show that the NAND gate is a universal gate



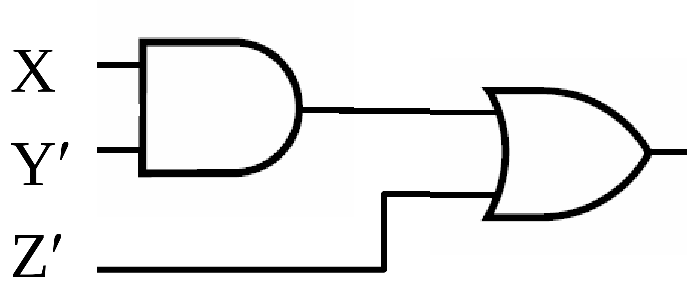
– E.g.: Simplify the Boolean function and implement it by NAND gates

$$F(X,Y,Z)=\sum m(0,2,4,5,6)$$

<Ans.> **NAND gates ← SoP form**

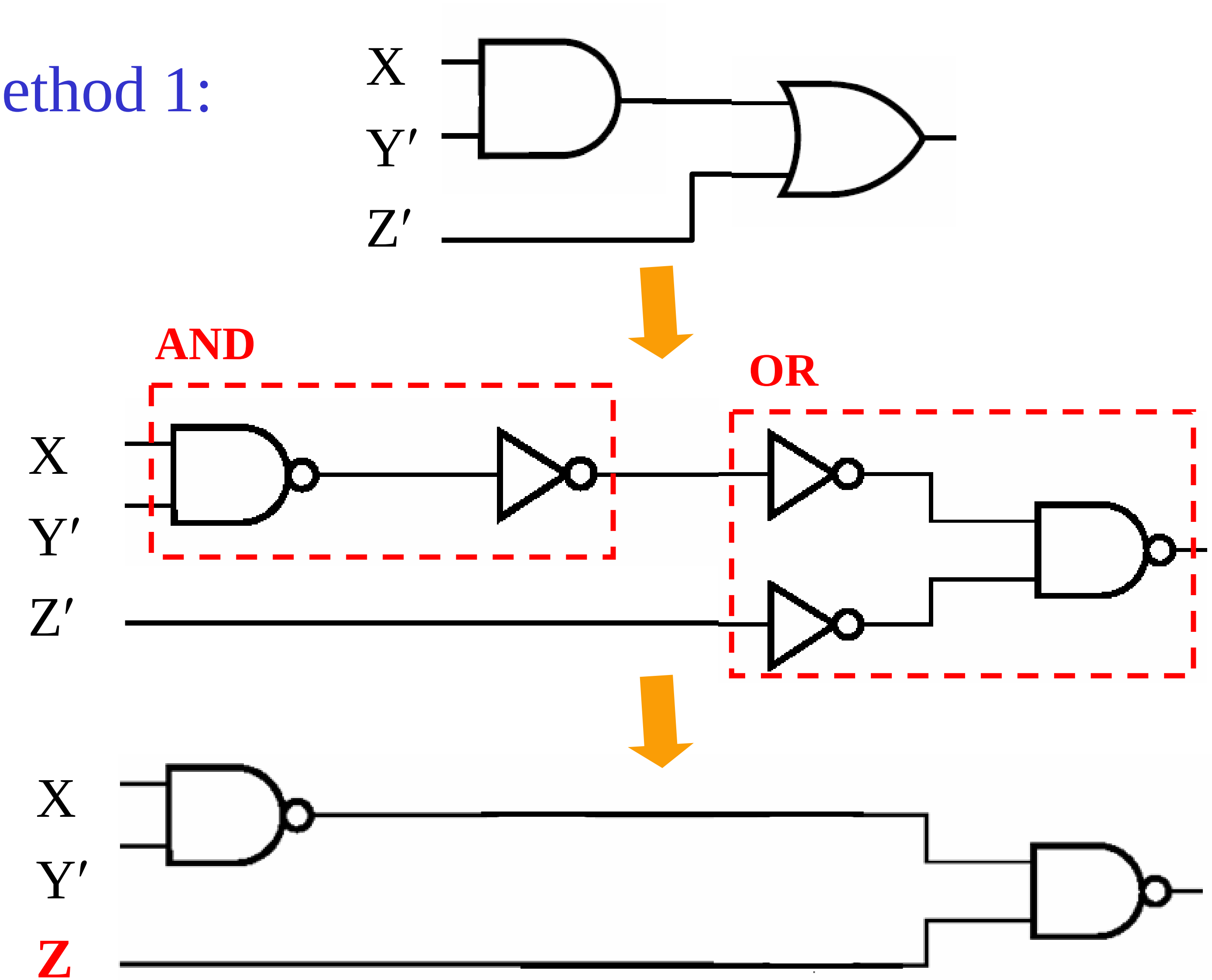


$$F(X,Y,Z)=\Sigma m(0,2,4,5,6)$$
$$=\overline{Z}+X\overline{Y}$$

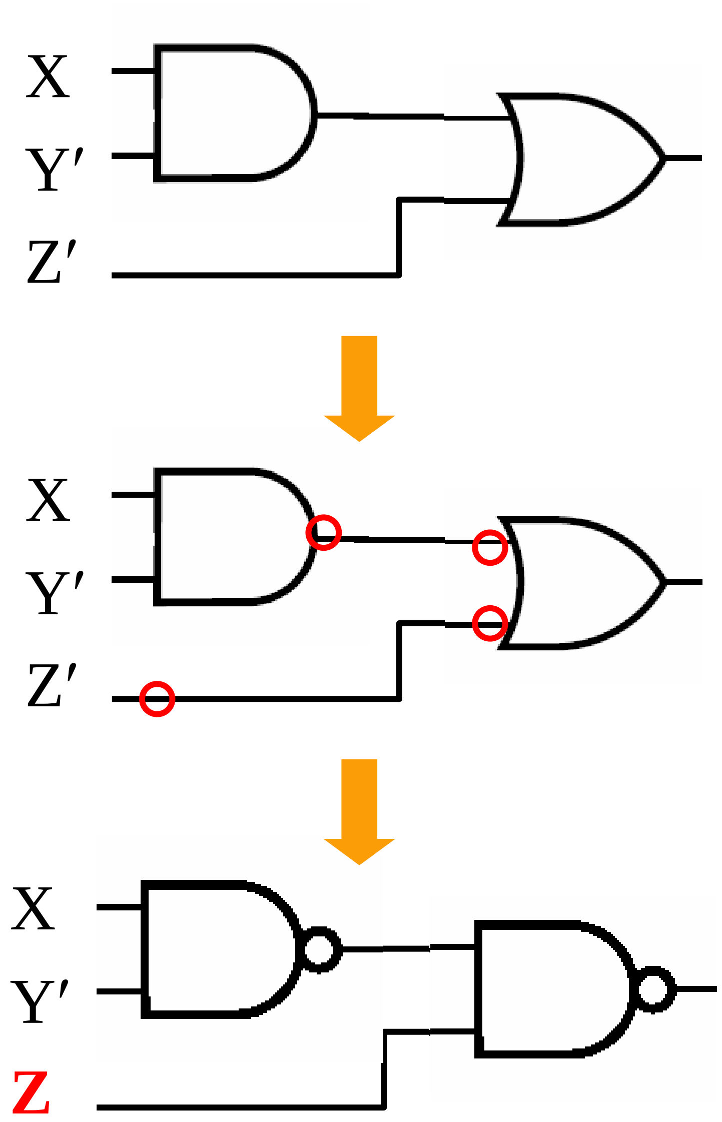


NAND gates

Method 1:



Method 2:



Complex Digital Logic Gates

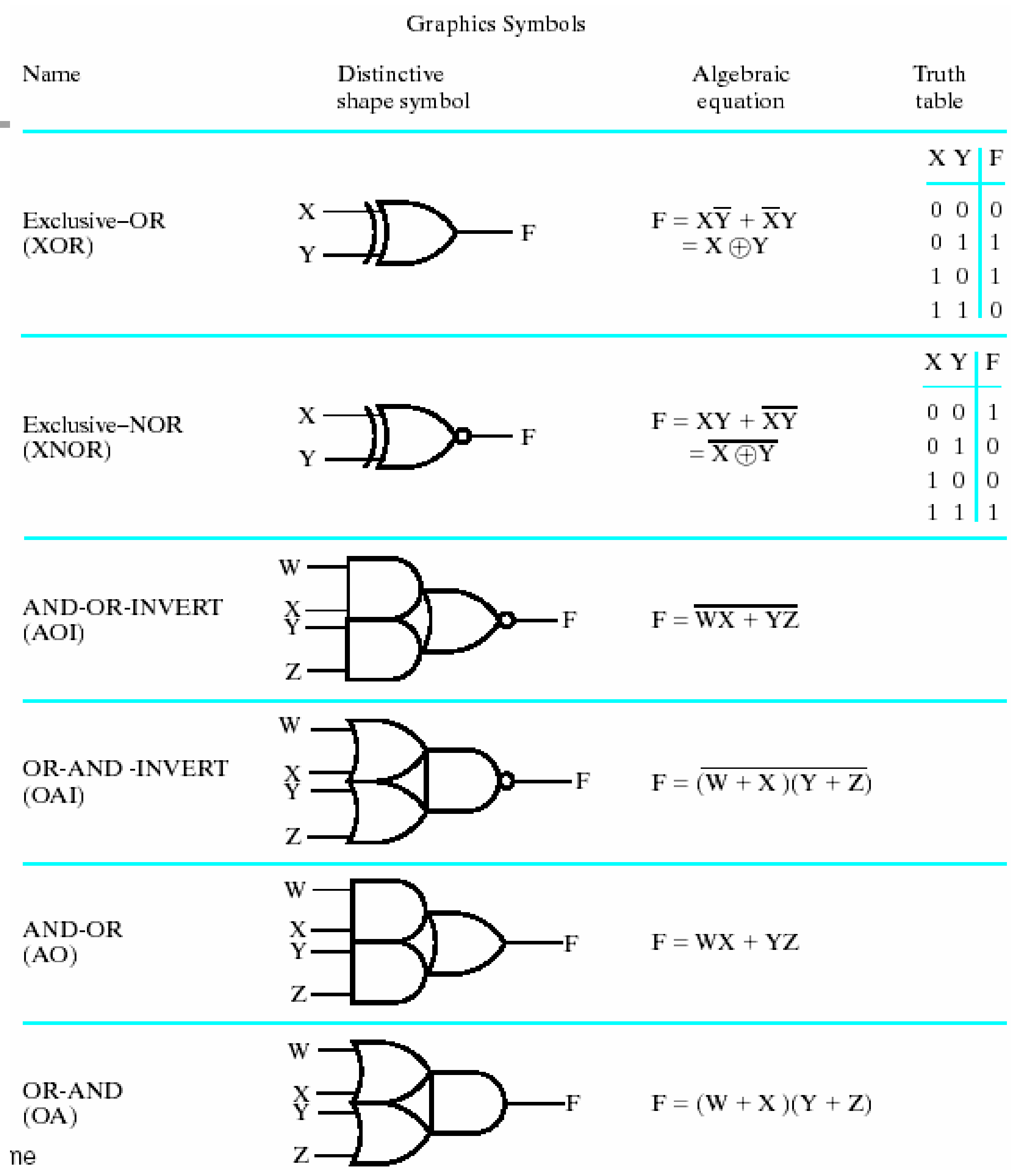
- Exclusive-OR (§2-8)
- XNOR (§2-8)
- AND-OR-Invert (AOI)
  - the complement of SoP
  - E.g.: 2-1 AOI

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

  - E.g.: 3-2-2 AOI

$$F = \overline{TUV + WX + YZ}$$

- OR-AND-Invert (OAI)
  - the dual of AOI
  - the complement of PoS
- AND-OR (AO)
- OR-AND (OA)



- Adv. of using complex gates:
  - reduce the ckt complexity needed for implementing specific Boolean functions in order to reduce integrated ckt (IC) cost
  - reduce the time required for signals to propagate through a ckt

2-8 Exclusive-OR Operator and Gates

- Exclusive-OR: XOR, ⊕

$$X \oplus Y = X\overline{Y} + \overline{X}Y$$

  - is equal to 1 if exactly one input variable is equal to 1
- Exclusive-NOR: XNOR, equivalence

$$\overline{X \oplus Y} = XY + \overline{X}\,\overline{Y}$$

  - is equal to 1 if both *X* and *Y* are equal to 1 or if both are equal to 0.
  - XOR & XNOR are the complement to each other.

$$\overline{X \oplus Y} = \overline{X\overline{Y} + \overline{X}Y} = (\overline{X} + Y)(X + \overline{Y}) = XY + \overline{X}\,\overline{Y}$$

- They are particularly useful in arithmetic operations and error-detection and correction ckts.

Properties of XOR

- Identities:

$$X \oplus 0 = X \qquad X \oplus 1 = \overline{X}$$

$$X \oplus X = 0 \qquad X \oplus \overline{X} = 1$$

$$X \oplus \overline{Y} = \overline{X \oplus Y} \qquad \overline{X} \oplus Y = \overline{X \oplus Y}$$

  – can be verified by using a truth table or by replacing the op by its equivalent Boolean expression

- Commutativity and associativity:

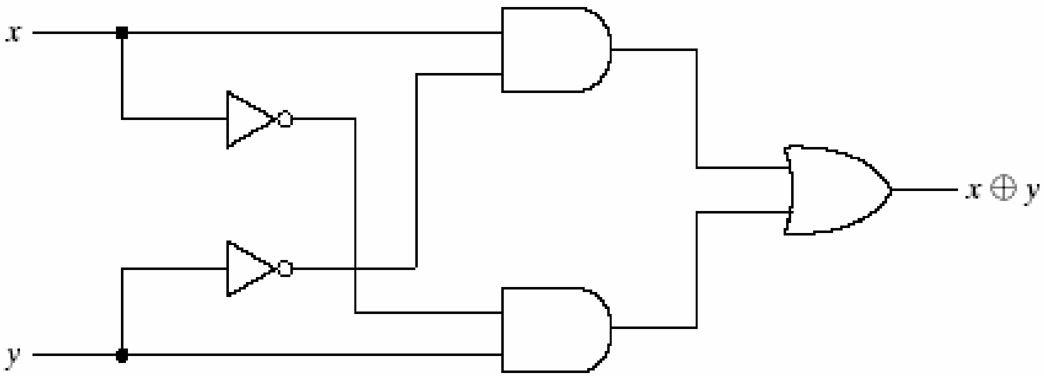
  $A \oplus B = B \oplus A$

  $(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$

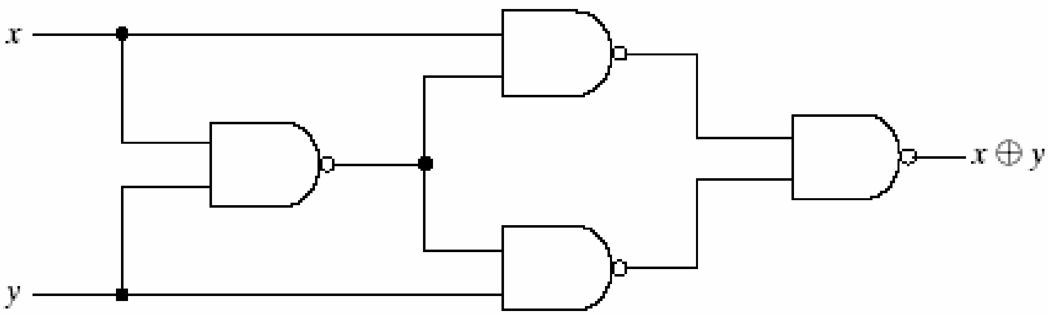
  $\Rightarrow$ XOR gates w/ three or more inputs

Implementations of XOR function

- XOR function is usually constructed w/ other types of gates:



(a) With AND-OR-NOT gates

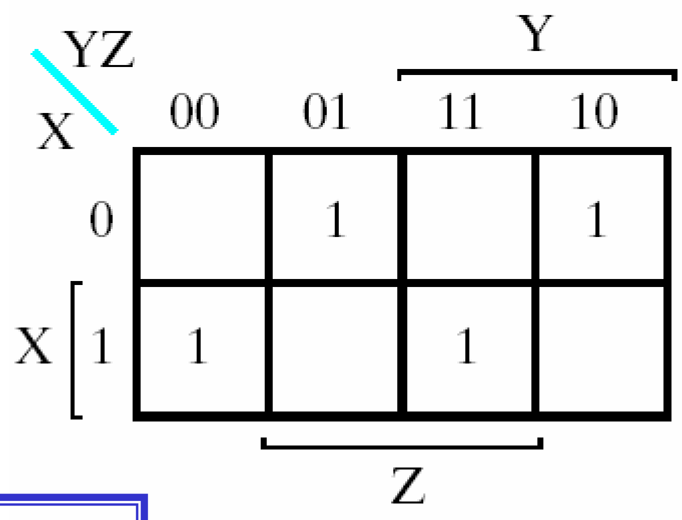


(b) With NAND gates

Odd Function

- Multiple-variable XOR operation: odd function
  - equal to 1 if the input variables have an odd # of 1's
  - E.g.: 3-variable XOR

$$
\begin{aligned}
X \oplus Y \oplus Z &= (X\overline{Y} + \overline{X}Y) \oplus Z \\
&= (X\overline{Y} + \overline{X}Y)\overline{Z} + (XY + \overline{X}\,\overline{Y})Z \\
&= X\overline{Y}\,\overline{Z} + \overline{X}Y\overline{Z} + \overline{X}\,\overline{Y}Z + XYZ \\
&= \sum m(1,2,4,7)
\end{aligned}
$$

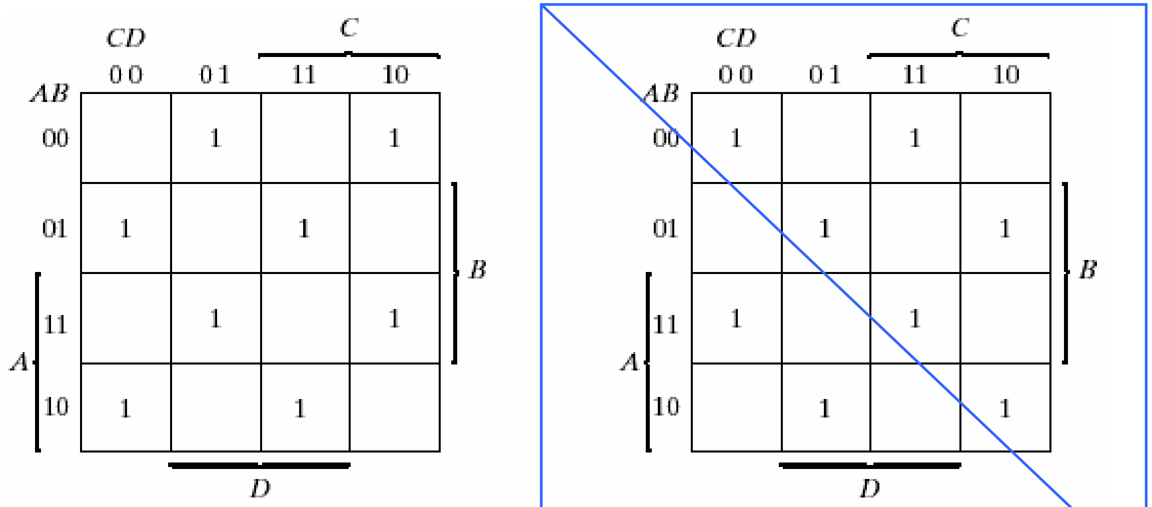


(a) $X \oplus Y \oplus Z$

$\Rightarrow$ is equal to 1 if only one variable is equal to 1 or if all three variables are equal to 1.

– E.g.: 4-variable XOR

$$A \oplus B \oplus C \oplus D = (AB' + A'B) \oplus (CD' + C'D)$$
$$= (AB' + A'B)(CD + C'D') + (AB + A'B')(CD' + C'D)$$
$$= \Sigma(1, 2, 4, 7, 8, 11, 13, 14)$$



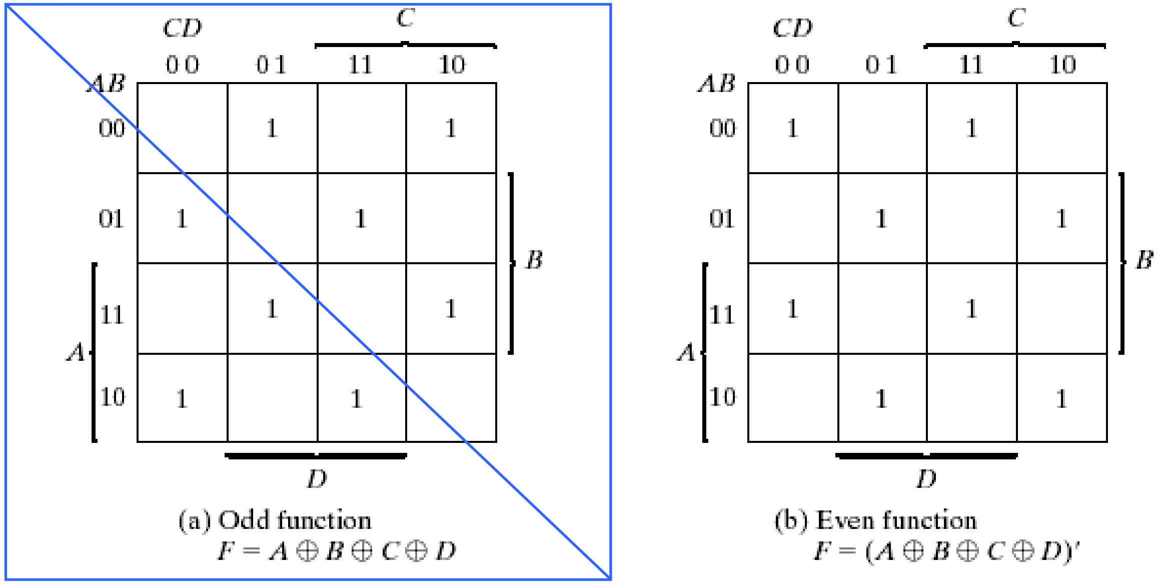
(a) Odd function
$F = A \oplus B \oplus C \oplus D$

(b) Even function
$F = (A \oplus B \oplus C \oplus D)'$

- *n*-variable XOR function: the logical sum of the $2^n/2$ minterms whose binary numerical values have an odd # of 1's.

Even Function

- Multiple-variable XNOR op: an even function
  - E.g.: 4-variable XNOR



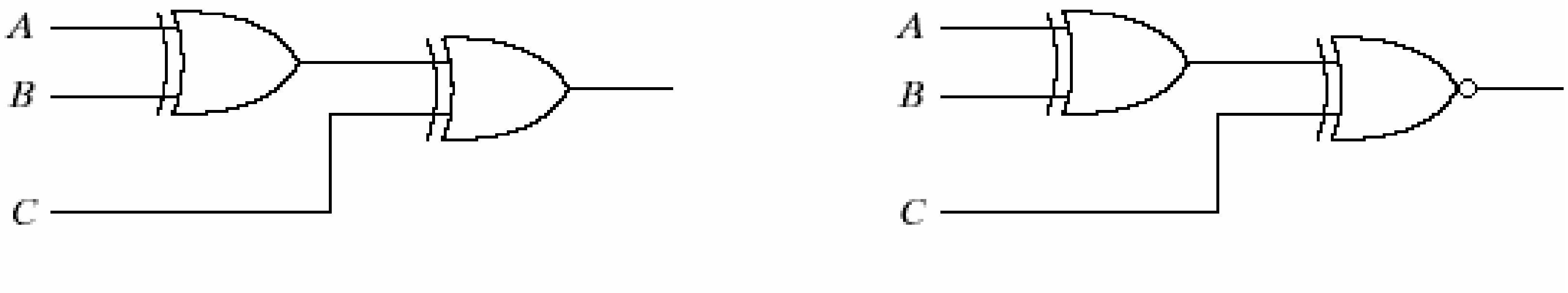
(a) Odd function
$F = A \oplus B \oplus C \oplus D$

(b) Even function
$F = (A \oplus B \oplus C \oplus D)'$

- n-variable XNOR function: the logical sum of the $2^n/2$ minterms whose binary numerical values have an even # of 1's.

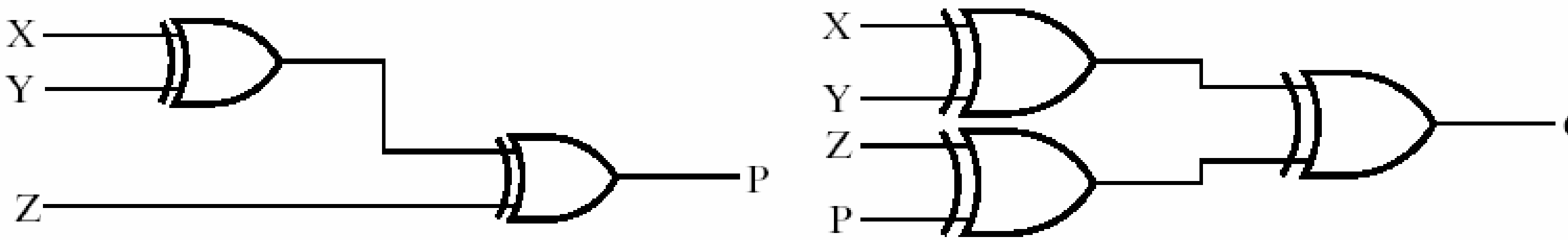
Logic Diagram of Odd & Even Functions

- Logic diagram of odd & even functions:



(a) 3-input odd function (b) 3-input even function

- Multiple-input Odd function:



(a) $P = X \oplus Y \oplus Z$ (b) $C = X \oplus Y \oplus Z \oplus P$

Example

- Simplify the Boolean function and implement it by XOR gates and other gates

$$F(W, X, Y, Z) = \Sigma(1, 4, 7, 8, 13)$$

<Ans.>

WX \ YZ	00	01	11	10
00		1		
01	1		1	
11		1		
10	1			

$$F = Y'(W \oplus X \oplus Z) + WZ'(X \oplus Y)'$$

or

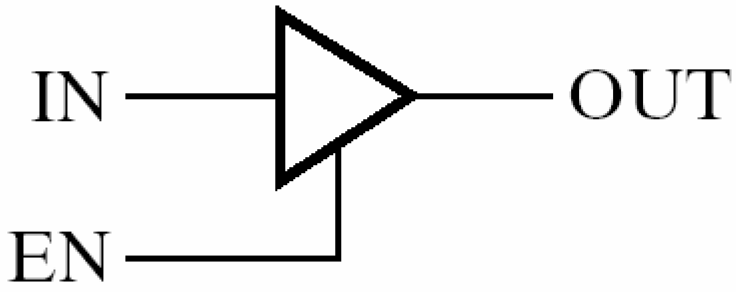
$$= Y'(W \oplus X \oplus Z) + W'XYZ$$

2-9 High-Impedance Outputs

- Output values of a gate:
  - 0, 1
  - Hi-Z: *high-impedance state*
    - behaves as an open ckt, i.e., looking back into the ckt, the output appears to be disconnected
  - * Gates w/ only logic 0 and 1 outputs cannot have their outputs connected together.
  - * Gates w/ Hi-Z output values can have their outputs connected together, provided that no 2 gates drive the line at the same time to opposite 0 and 1 values.
- Structures that provide Hi-Z:
  - 3-state buffers
  - transmission gates

Three-State Buffers

- 3-state buffer:

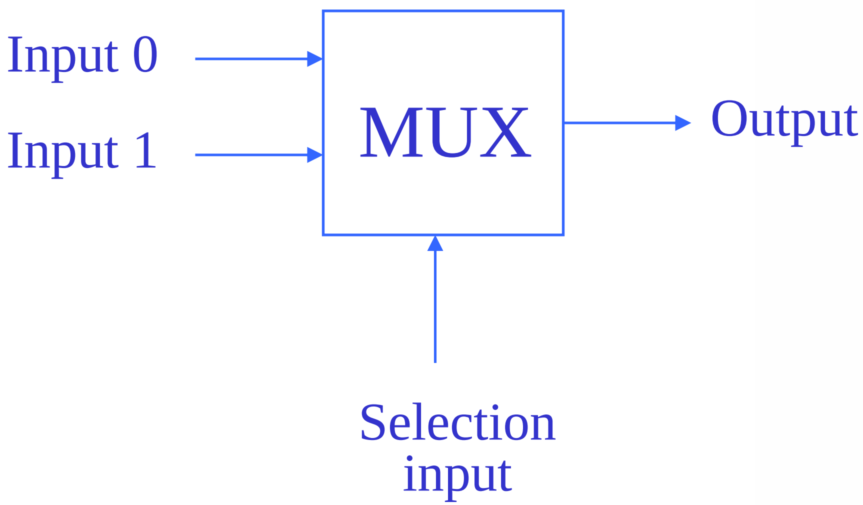


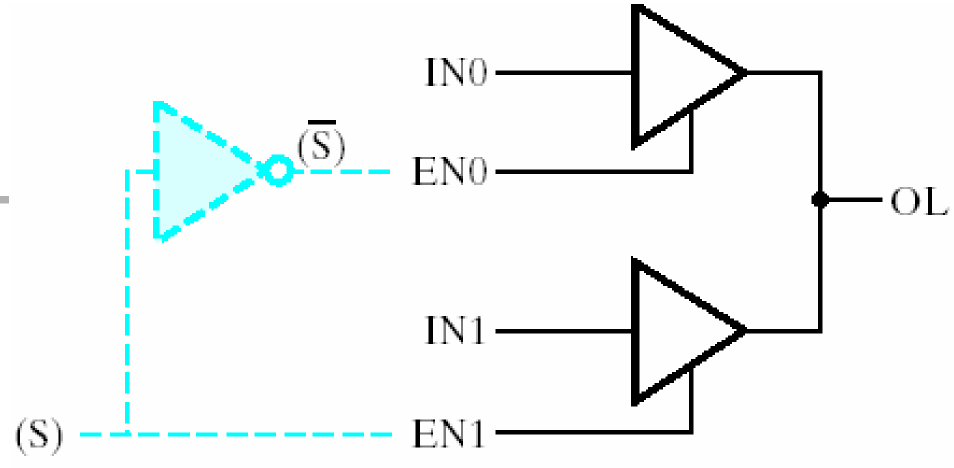
(a) Logic symbol

EN	IN	OUT
0	X	Hi-Z
1	0	0
1	1	1

(b) Truth table

- Multiplexer constructed by 3-state buffers:
  - E.g.: 2-to-1 MUX





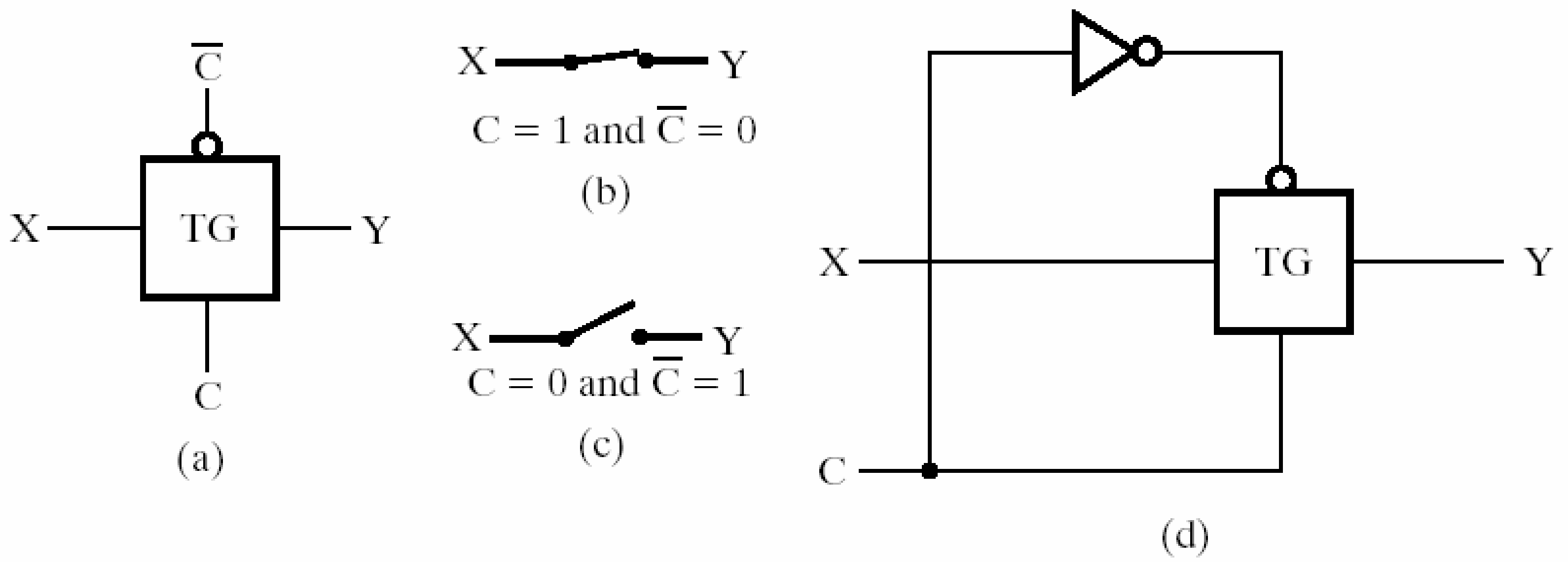
(a) Logic Diagram

EN1	EN0	IN1	IN0	OL
0	0	X	X	Hi-Z
(S) 0	($\overline{S}$) 1	X	0	0
0	1	X	1	1
1	0	0	X	0
1	0	1	X	1
1	1	0	0	0
1	1	1	1	1
1	1	0	1	
1	1	1	0	

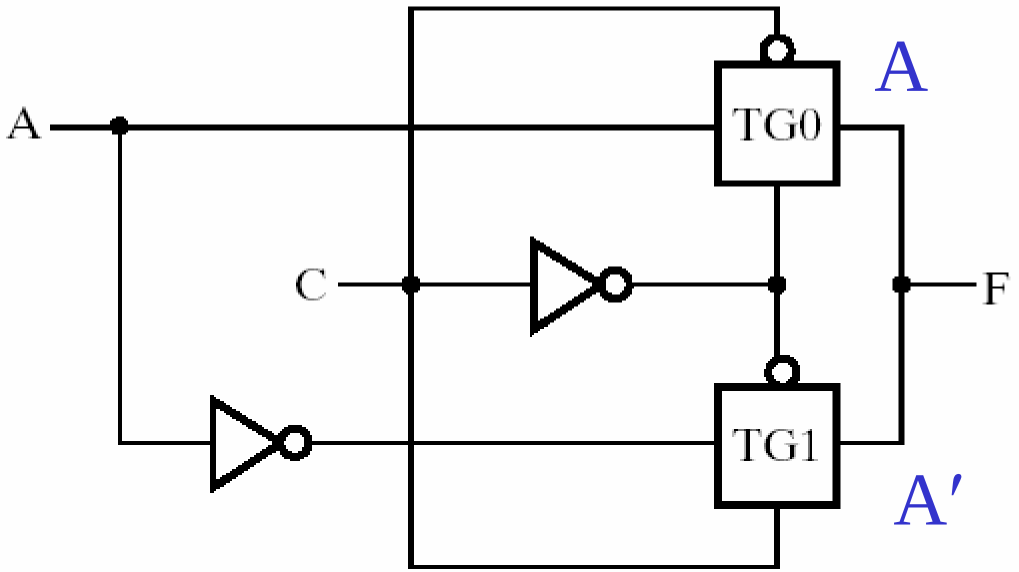
(b) Truth table

Transmission Gates

- Transmission gate (TG):



- XOR gate constructed from transmission gates:



(a)

A	C	TG1	TG0	F
0	0	No path	Path	0
0	1	Path	No path	1
1	0	No path	Path	1
1	1	Path	No path	0

(b)

$C = 0, F = A$

$C = 1, F = A'$

$F = AC' + A'C$

2-10 Chapter Summary

- Primitive logic ops: AND, OR, NOT
- Boolean algebra
- Minterm & maxterm standard forms
- SoP and PoS standard forms $\Rightarrow$ 2-level gate ckts
- 2 cost measures for ckt optimization:
  - # of input literals to a ckt
  - total # of inputs to the gates in a ckt

- Circuit optimization:
  - 2-level ckt optimization
    - Boolean algebra manipulation
    - K-map
    - Quine-McCluskey method
  - Multi-level ckt optimization
- Other logic gates:
  - NAND, NOR
  - XOR, XNOR
- Hi-Z outputs:
  - 3-state buffer
  - transmission gates

Problems

Sections	Exercises
§2-1	
§2-2	2-1 ~ 2-9
§2-3	2-10 ~ 2-13
§2-4	2-14 ~ 2-18
§2-5	2-19 ~ 2-26
§2-6	2-17 ~ 2-29
§2-7	2-30
§2-8	2-31
§2-9	2-32 ~ 2-35

Homework (1/2)

- Part I:
  1. 2-3 (a)與(c)用algebraic manipulation來證明
  2. 2-4 (證明兩個Boolean equations相等)
  3. 2-6(a) and (d) (化簡Boolean equation)
  4. 2-11(minterms and maxterms)
  5. 2-12 (sum-of-products and product-of-sums)
  6. 找出下列式子的complement (證明DeMorgan's law)

     (a) $x'y' + xy + x'y$

     (b) $ab(c'd+cd') + a'b'(c'+d)(c+d')$
  7. 用真值表驗證底下的式子

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

Homework (2/2)

- Part II:
  1. 使用K-Map化簡 $F(a,b,c,d) = b'cd' + abc + a'bc + abc'd$ -->給定布林方程式來做化簡
  2. 2-16(c) -->給定布林方程式來做化簡
  3. 2-21(a) -->給定布林方程式（輸入是sum of minterm）來做化簡，PoS form
  4. 2-22(c) -->給定布林方程式來做化簡，並轉成SoP and PoS forms
  5. 2-26(a) -->給定布林方程式與don't-card conditions來做化簡，並轉成SoP and PoS forms
  6. 2-27(a) -->decomposition
  7. 2-29(a) -->elimination
  8. 2-31 -->使用excusinve-OR與AND，並在最少# of gate inputs的前提下，實做一個布林方程式