Chapter 4

Combinational Logic Circuits

Dinh Duc Anh Vu
International University – VNU HCM

Outline

- Boolean algebra
  - Algebraic simplification
- Designing combinational circuits
- Karnaugh map simplification
- Useful combinational circuits

Objectives

- Understanding and applying Boolean algebra in designing logic circuits
- Convert a logic expression into a sum-of-products expression.
- Perform the necessary steps to reduce a sum-of-products expression to its simplest form.
- Use Boolean algebra and the Karnaugh map as tools to simplify and design logic circuits.
- Design simple logic circuits without the help of a truth table.
- Implement enable/disable circuits.

Boolean Algebra

- Algebra for Logic Circuits

The Duality Principle

- The dual of an expression is obtained by exchanging ($\cdot$ and $+$), and (1 and 0) in it, provided that the precedence of operations is not changed.
- Cannot exchange x with $\bar{x}$

- Example:
  - $F(x,y,z) = \bar{x}.y.\bar{z} + \bar{x}.\bar{y}.z$
  - Find $H(x,y,z)$, the dual of F

$$H(x,y,z) = (\bar{x} + y + \bar{z}).(\bar{x} + \bar{y} + z)$$

Boolean Theorems revisited (Single variable)

- Let X: boolean variable, and 0,1: constants

1. $X + 0 = X$ -- Zero Axiom
2. $X \cdot 1 = X$ -- Unit Axiom
3. $X + 1 = 1$ -- Unit Property
4. $X \cdot 0 = 0$ -- Zero Property
5. $X + X = X$ -- Idempotence
6. $X \cdot X = X$ -- Idempotence
7. $X + X' = 1$ -- Complement
8. $X \cdot X' = 0$ -- Complement

Power of Duality: Covering – Consensus theorem

- Covering theorem:
  - $x + x \cdot y = x$
  - $x \cdot (x + y) = x$
- Consensus theorem:
  - $x \cdot y + \bar{x} \cdot z + y \cdot z = x \cdot y + \bar{x} \cdot z$
  - $(x + y)(\bar{x} + z)(y + z) = (x + y)(\bar{x} + z)$

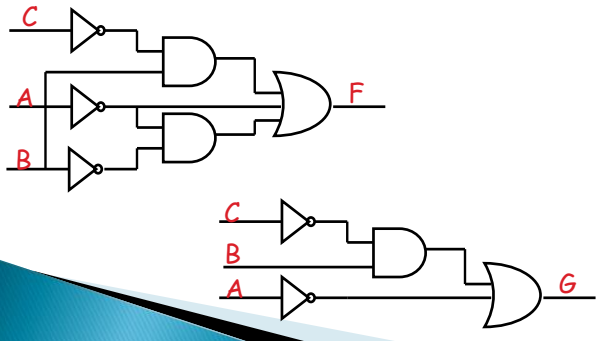
Boolean expressions–NOT unique

- Unlike truth tables, expressions representing a Boolean function are NOT unique.
- Example:
  - $F(x, y, z) = \bar{x}\bar{y}\bar{z} + \bar{x}y\bar{z} + xy\bar{z}$
  - $G(x, y, z) = \bar{x}\bar{y}\bar{z} + y\bar{z}$
- The corresponding truth tables for F() and G() are to the right. They are identical!
- Thus, F() = G()

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

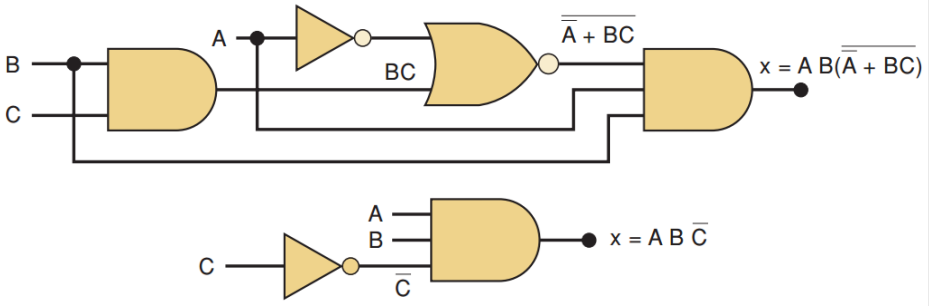
Simplifying Logic Circuit from Logic Function

- In order to design a cost-effective and efficient circuit, we must minimize the circuit's size (area) and propagation delay (time required for an input signal change to be observed at the output line)
- Observe the truth table of $F = \overline{A} + B\overline{C} + \overline{A}\overline{B}$ and $G = \overline{A} + B\overline{C}$
- Truth tables for F and G are identical, i.e. same function
- Use G to implement the logic circuit (less components)



A	B	C	F	G
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

Simplifying Logic Circuits

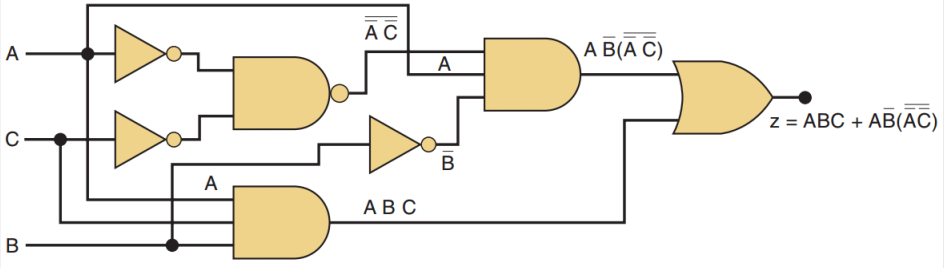


- Both circuits perform the same logic, so it should be obvious that the simpler circuit is more desirable because it contains fewer gates and will therefore be smaller and cheaper than the original.
- Furthermore, the circuit reliability will improve because there are fewer interconnections that can be potential circuit faults

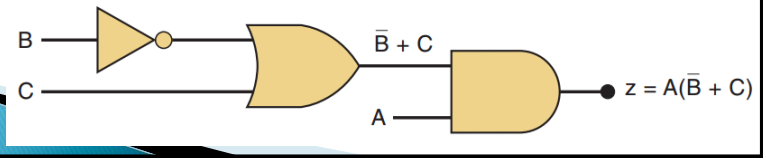
Algebraic Simplification

- Boolean algebra is a useful tool for simplifying digital circuits.
- Why do it? Simpler can mean cheaper, smaller, faster.
- It is not always obvious which theorems should be applied to produce the simplest result
- Two essential steps:
  - The original expression is put into SOP form by repeated application of De Morgan's theorems and multiplication of terms.
  - Once the original expression is in SOP form, the product terms are checked for common factors, and factoring is performed wherever possible. The factoring should result in the elimination of one or more terms.

Algebraic Simplification – Example



$$z = ABC + A\overline{B}\left(\overline{\overline{A}\,\overline{C}}\right) = ABC + A\overline{B}\left(\overline{\overline{A}} + \overline{\overline{C}}\right)$$
$$= ABC + A\overline{B}(A + C) = ABC + A\overline{B}A + A\overline{B}C = ABC + A\overline{B} + A\overline{B}C$$
$$= AC\left(B + \overline{B}\right) + A\overline{B} = AC + A\overline{B} = A(C + \overline{B})$$



Algebraic Simplification – Example

- Simplify $F = \bar{x}yz + \bar{x}y\bar{z} + xz$

$$F = \bar{x}yz + \bar{x}y\bar{z} + xz$$
$$= \bar{x}y(z + \bar{z}) + xz$$
$$= \bar{x}y(1) + xz$$
$$= \bar{x}y + xz$$

- Simplify $F = xyz + x\bar{y}(\overline{\bar{x}\,\bar{z}})$

$$F = xyz + x\bar{y}\left(\overline{\bar{x}\,\bar{z}}\right) = xyz + x\bar{y}\left(\bar{\bar{x}} + \bar{\bar{z}}\right)$$
$$= xyz + x\bar{y}(x + z) = xyz + x\bar{y} + x\bar{y}z$$
$$= xz(y + \bar{y}) + x\bar{y} = xz + x\bar{y}$$
$$= x(z + \bar{y})$$

Algebraic Manipulation

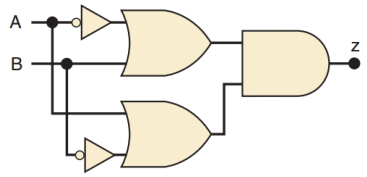
- Prove $\bar{x}\bar{y}\bar{z} + \bar{x}y\bar{z} + xy\bar{z} = \bar{x}\bar{z} + y\bar{z}$
- Proof:

$$\bar{x}\bar{y}\bar{z} + \bar{x}y\bar{z} + xy\bar{z}$$
$$= \bar{x}\bar{y}\bar{z} + \bar{x}y\bar{z} + \bar{x}y\bar{z} + xy\bar{z}$$
$$= \bar{x}\bar{z}(\bar{y} + y) + y\bar{z}(\bar{x} + x)$$
$$= \bar{x}\bar{z} + y\bar{z}$$

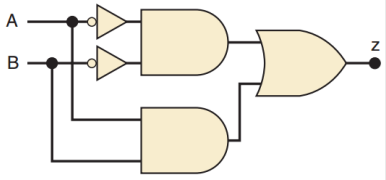
QED.

Review

- $z = \overline{A}C\left(\overline{\overline{A}BD}\right) + \overline{A}B\overline{C}\,\overline{D} + A\overline{B}C$
- $z = \overline{B}C + \overline{AD}(B + C)$



- $t = (\overline{A} + B)(A + \overline{B})$
- $t = \overline{AB} + AB$



- $x = (\overline{A} + B)(A + B + D)\overline{D}$
- $x = B\overline{D}$

Complement of a Function

- The complement of a function is derived by interchanging ($\cdot$ and $+$), and (1 and 0), and complementing each variable.
- Otherwise, interchange 1s to 0s in the truth table column showing F.
- The complement of a function IS NOT THE SAME as the dual of a function.

Complementation: Example

- Find the complement of $F = x\bar{y}\bar{z} + \bar{x}yz$
- G = F' $= \overline{x\bar{y}\bar{z} + \bar{x}yz}$
  $= \overline{x\bar{y}\bar{z}} \cdot \overline{\bar{x}yz}$ DeMorgan
  $= (\bar{x} + y + z)(x + \bar{y} + \bar{z})$ DeMorgan
- Note: The complement of a function can also be derived by finding the function's dual, and then complementing all of the literals

Definitions

- *Literal:* A variable or its complement
- *Product term:* literals connected by $\cdot$
- *Sum term:* literals connected by $+$
- *Minterm:* a product term in which all the variables appear exactly once, either complemented or uncomplemented
- *Maxterm:* a sum term in which all the variables appear exactly once, either complemented or uncomplemented

Forms of an expression

- A switching function can be represented by several different, but equivalent, algebraic expressions.
  - There are two standard forms
    - SOP (sum of product)
    - POS (product of sum)
  - The standard form is a unique algebraic representation of each function
    - Canonical form

Minterm

- For an n-variable function, a minterm is a product term of n variables in complemented or un-complemented form.
  - If the variable's value is 0 → complemented form.
  - If the variable's value is 1 → un-complemented form.
- With n variables → 2^n minterms
- Symbol for minterm: m_i , whereas i is the decimal equivalent of the minterm's corresponding binary combination (b_i)
  - Example: Assume 3 variables (A,B,C), and i=3. Then, $b_i = 011$ and its corresponding minterm is denoted by $m_i = \bar{A}BC$
- If A, B and C are the input variables, the minterms are: $\bar{A}\bar{B}\bar{C}$, $\bar{A}\bar{B}C$, $\bar{A}B\bar{C}$, $\bar{A}BC$, $A\bar{B}\bar{C}$, $A\bar{B}C$, $AB\bar{C}$, ABC

Represents exactly one combination in the truth table

Maxterm

- For an n-variable function, a maxterm is a sum term of n variables in complemented or un-complemented form.
  - If the variable's value is 1 → complemented form.
  - If the variable's value is 0 → un-complemented form.
- With n variables → 2^n maxterms
- Symbol for maxterm: M_i , whereas i is the decimal equivalent of the maxterm's corresponding binary combination (b_i)
  - Example: Assume 3 variables (A,B,C), and i=3. Then, b_i = 011 and its corresponding maxterm is denoted by $M_i = A + \bar{B} + \bar{C}$
- If A, B and C are the input variables, the maxterms are: $(\bar{A} + \bar{B} + \bar{C})$, $(\bar{A} + \bar{B} + C)$, $(\bar{A} + B + \bar{C})$, $(\bar{A} + B + C)$, $(A + \bar{B} + \bar{C})$, $(A + \bar{B} + C)$, $(A + B + \bar{C})$, $(A + B + C)$
- Represents exactly one combination in the truth table

Minterm vs Maxterm

A	B	C	Minterm		Maxterm	
			Expression	Symbol	Expression	Symbol
0	0	0	$\bar{A}\bar{B}\bar{C}$	m_0	$A+B+C$	M_0
0	0	1	$\bar{A}\bar{B}C$	m_1	$A+B+\bar{C}$	M_1
0	1	0	$\bar{A}B\bar{C}$	m_2	$A+\bar{B}+C$	M_2
0	1	1	$\bar{A}BC$	m_3	$A+\bar{B}+\bar{C}$	M_3
1	0	0	$A\bar{B}\bar{C}$	m_4	$\bar{A}+B+C$	M_4
1	0	1	$A\bar{B}C$	m_5	$\bar{A}+B+\bar{C}$	M_5
1	1	0	$AB\bar{C}$	m_6	$\bar{A}+\bar{B}+C$	M_6
1	1	1	ABC	m_7	$\bar{A}+\bar{B}+\bar{C}$	M_7

Minterm vs Maxterm

- The complement of minterm is maxterm and vice versa

$$\overline{m_i} = M_i$$
$$\overline{M_i} = m_i$$

Standard Sum of Product – SOP

- Express a Boolean function

$$F = \sum_{i=0}^{2^n - 1} m_i F_i$$

Whereas

- m_i is the ith minterm
- F_i is the F-function's value corresponding to the ith minterm

Canonical Sum Form
Sum of Products (OR of AND terms)

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

Minterms

SOP Examples

- Express the following functions in the Standard SOP
  - $F(A,B,C,D) = A\overline{B}CD + A\overline{B}\overline{C}D + ABC\overline{D} + \overline{A}BC\overline{D}$
    - Already in standard SOP form
  - $F(A,B,C) = AC + \overline{A}B$

$$F(A,B,C) = AC\left(B + \overline{B}\right) + \overline{A}B\left(C + \overline{C}\right)$$

$$F(A,B,C) = ABC + A\overline{B}C + \overline{A}BC + \overline{A}B\overline{C}$$

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

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

$$F(A,B,C,D) = AC + \overline{A}\,\overline{B} + \overline{B}C = \ldots$$

SOP Examples

- Write the expression form of the following functions
- $F(A,B,C) = \sum(1,4,5,6)$

$$F(A,B,C) = \overline{A}\,\overline{B}C + A\overline{B}\,\overline{C} + A\overline{B}C + AB\overline{C}$$

- $F(A,B,C,D) = \sum(1,4,5,6)$

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

Standard Product of Sum – POS

- Express a Boolean function

$$F = \prod_{i=0}^{2^n-1} (M_i + F_i)$$

whereas

- m_i is the ith maxterm
- F_i is the F–function's value corresponding to the ith maxterm

POS Examples

- Write the expression form of the following functions
- $F(A,B,C) = \prod(1,4,5,6)$

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

- $F(A,B,C,D) = \sum(1,4,5,6)$

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

Product of Sums – POS

- Transform the function F into the standard form 2
- $F(A,B,C) = \sum(0,2,3,7)$

$M_i = \overline{m_i}$

$$F(A,B,C) = \sum(0,2,3,7) = \prod(7,5,4,0)$$

Review: Minterm vs Maxterm

- For an n-variable function, a minterm is a product term of n variables in complemented or un-complemented form.
  - If the variable's value is 0 → complemented form.
  - If the variable's value is 1 → un-complemented form.
- Symbol for minterm: m_i , whereas i is the decimal equivalent of the minterm's corresponding binary combination (b_i)
- A maxterm is a sum term of n variables in complemented or un-complemented form.
  - If the variable's value is 1 → complemented form.
  - If the variable's value is 0 → un-complemented form.
- Symbol for maxterm: M_i , whereas i is the decimal equivalent of the maxterm's corresponding binary combination (b_i)
- The complement of minterm is maxterm and vice versa
  - $\overline{m_i} = M_i$
  - $\overline{M_i} = m_i$
- Represents exactly one combination in the truth table

Algebraic Expression into Truth Table

- Transform the expression form into Standard SOP.
- Fill values "1" in the rows having the binary combinations equals to the minterm indices of the expression. The other rows will have F=0.

Algebraic Expression into Truth Table – Example

- Write the truth table of the following function

$$F(A,B,C) = AC + \overline{A}B$$

We have

$$F(A,B,C) = AC + \overline{A}B$$
$$= AC\left(B + \overline{B}\right) + \overline{A}B\left(C + \overline{C}\right)$$
$$= ABC + A\overline{B}C + \overline{A}BC + \overline{A}B\overline{C}$$
$$= \sum(7,5,3,2)$$

→Fill 1 in the rows 2, 3, 5, 7 and 0 in the other rows.

A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Algebraic Expression into Truth Table

- Transform the expression form into Standard POS.
- Fill values "0" in the rows having the binary combinations equals to the maxterm indices of the expression. The other rows will have F=1.

Algebraic Expression into Truth Table – Example

- Write the truth table of the following function

$$F(A,B,C) = AC + \overline{A}B$$

$F(A,B,C) = AC + \overline{A}B$

$= (A + \overline{A})(A + B)(C + \overline{A})(C + B)$ (dual of the distributive law)

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

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

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

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

$= \prod(0,1,4,6)$

Algebraic Expression into Truth Table – Example

→Fill 0 in rows 0, 1, 4, 6 and 1 in the other rows.

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

Truth Table into the Algebraic Expression

- Example: Write the algebraic form of the following function

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

$$F(A,B,C)=\sum(0,1,3,5,6)$$
$$=m_0+m_1+m_3+m_5+m_6$$
$$=\overline{A}\,\overline{B}\,\overline{C}+\overline{A}\,\overline{B}C+\overline{A}B\overline{C}+A\overline{B}\,\overline{C}$$

Truth Table into the Algebraic Expression

- Example: Write the algebraic form of the following function

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

$$F(A,B,C)=\prod(2,4,7)$$
$$=(A+\overline{B}+C)(\overline{A}+B+C)(\overline{A}+\overline{B}+\overline{C})$$

"Don't care"

- In practical, there are some cases in which the binary combinations of variables does not exist. Thus, the output (the value of the function) corresponding to these binary combinations can be 0 or 1 (called "don't care"), symbol "d". When filling in the Truth Table for the "don't care" cases, the symbol "x" is used.

"Don't care"

- Example: build Truth table for F

$$F(A,B,C)=\sum(1,2,4,5)+d(0,6)$$

A	B	C	F
0	0	0	x
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	x
1	1	1	0

"Don't care"

- Example: build Truth table for F

$$F(A,B,C)=\prod(0,2,6).d(1,3,7)$$

A	B	C	F
0	0	0	0
0	0	1	x
0	1	0	0
0	1	1	x
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	x

Combinational Logic Circuits

Designing Combinational Logic Circuit

Any logic problem can be solved using the following step-by-step procedure.

1. Interpret the problem and set up a truth table to describe its operation.
2. Write the AND (product) term for each case where the output is 1.
3. Write the sum-of-products (SOP) expression for the output.
4. Simplify the output expression if possible.
5. Implement the circuit for the final, simplified expression.

Designing Combinational Logic Circuit – Example 1

- Problem: A logic circuit having 3 inputs, A, B, C will have its output HIGH only when a majority of the inputs are HIGH
- **Step 1** Set up the truth table
- **Step 2** Write the AND term for each case where the output is a 1.

A	B	C	x	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\rightarrow \bar{A}BC$
1	0	0	0	
1	0	1	1	$\rightarrow A\bar{B}C$
1	1	0	1	$\rightarrow AB\bar{C}$
1	1	1	1	$\rightarrow ABC$

Combinational Logic Circuit Design – Example 1

- **Step 3** Write the SOP form for the output

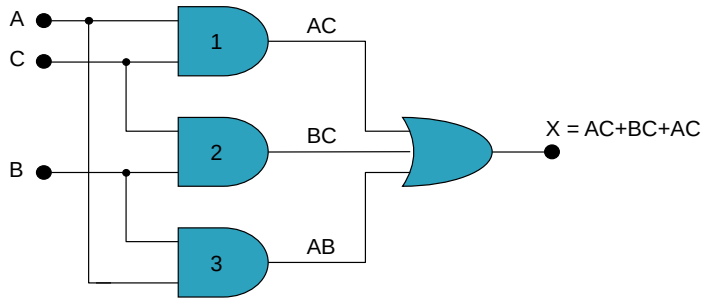
$$x = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

- **Step 4** Simplify the output expression

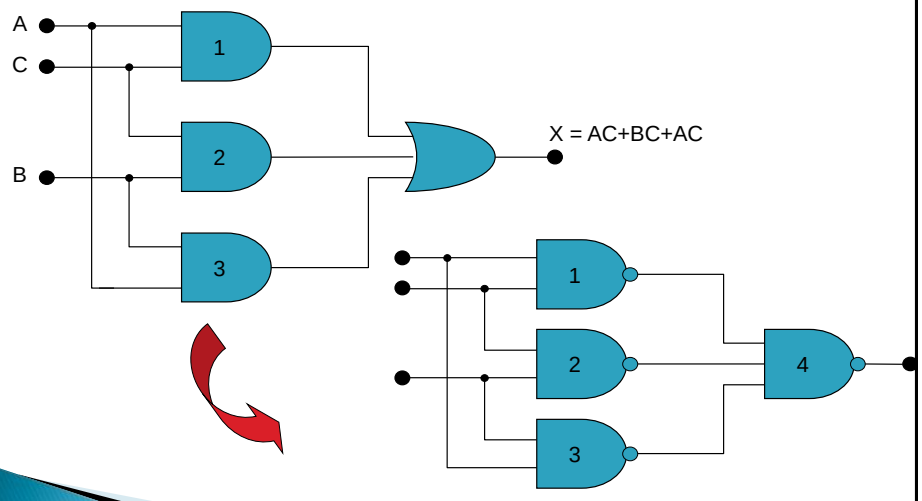
$$\begin{aligned} x &= \bar{A}BC + ABC + A\bar{B}C + ABC + AB\bar{C} + ABC \\ &= BC(\bar{A} + A) + AC(\bar{B} + B) + AB(\bar{C} + C) \\ &= BC + AC + AB \end{aligned}$$

Combinational Logic Circuit Design – Example 1

- **Step 5** Implement the circuit

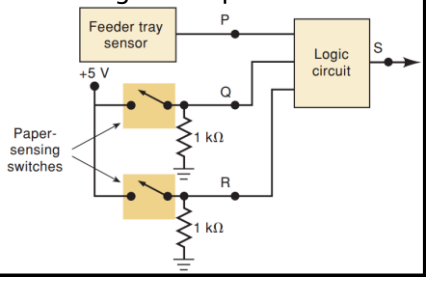


Implement the Circuit with NAND Gates



Combinational Logic Circuit Design – Example 2

- In a simple copy machine, a stop signal, S, is to be generated to stop the machine operation and energize an indicator light whenever either of the following conditions exists:
  - (1) there is no paper in the paper feeder tray; or
  - (2) the two microswitches in the paper path are activated, indicating a jam in the paper path.
- The presence of paper in the feeder tray is indicated by a HIGH at logic signal P. Each of the microswitches produces a logic signal (Q and R) that goes HIGH whenever paper is passing over the switch to activate it.
- Design the logic circuit to produce a HIGH at output signal S for the stated conditions, and implement it using two-input NAND gates.



Combinational Logic Circuit Design – Example 2

- **Step 1**. Setup the truth table
- **Step 2**. Write the AND term for each case where the output is a 1
- **Step 3**. Write the SOP form for the output

P	Q	R	S	
0	0	0	1	$\overline{P}\,\overline{Q}\,\overline{R}$
0	0	1	1	$\overline{P}\,\overline{Q}R$
0	1	0	1	$\overline{P}Q\overline{R}$
0	1	1	1	$\overline{P}QR$
1	0	0	0	
1	0	1	0	
1	1	0	0	
1	1	1	1	PQR

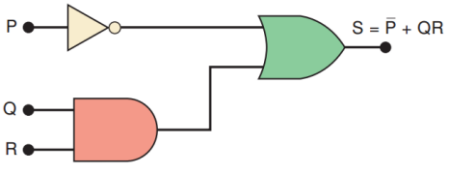
$$S = \overline{P}\,\overline{Q}\,\overline{R} + \overline{P}\,\overline{Q}R + \overline{P}Q\overline{R} + \overline{P}QR + PQR$$

Combinational Logic Circuit Design – Example 2

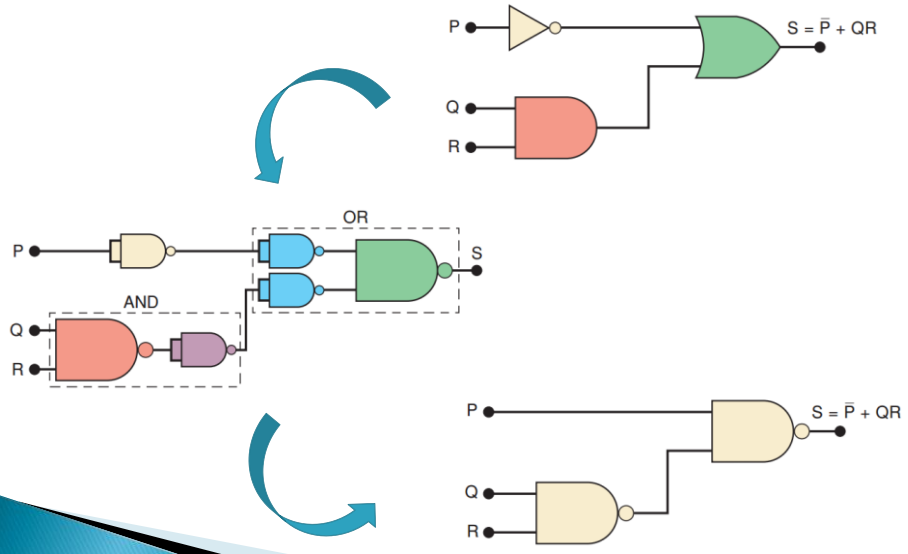
- **Step 4.** Simplify the output expression

$$S = \overline{PQ}R + \overline{PQ}R + \overline{P}Q\overline{R} + \overline{P}QR + PQR$$
$$= \overline{PQ}(\overline{R} + R) + \overline{P}Q(\overline{R} + R) + PQR$$
$$= \overline{PQ} + \overline{P}Q + PQR$$
$$= \overline{P}(\overline{Q} + Q) + PQR = \overline{P} + PQR$$
$$= \overline{P} + QR \text{ (by applying } \overline{x} + xy = \overline{x} + y)$$

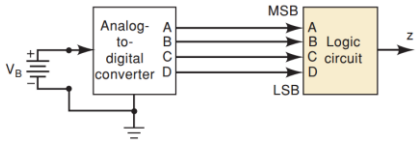
- **Step 5.** Implement the circuit



Implement the Circuit with NAND Gates

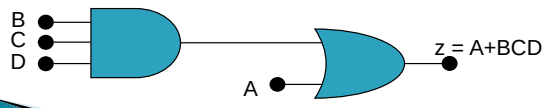


Example 3

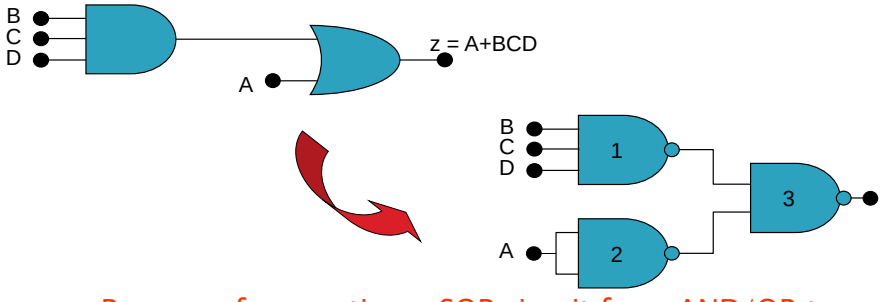


- Designing a circuit that has 4 logic signal lines A, B, C, D representing a 4-bit binary number with A as MSB and D as LSB. The circuit produces a HIGH output only when the binary number is greater than $0110_2 = 6_{10}$

A	B	C	D	z	
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	0	
0	1	0	0	0	
0	1	0	1	0	
0	1	1	0	0	
0	1	1	1	1	$\rightarrow \bar{A}BCD$
1	0	0	0	1	$\rightarrow A\bar{B}\bar{C}\bar{D}$
1	0	0	1	1	$\rightarrow A\bar{B}\bar{C}D$
1	0	1	0	1	$\rightarrow A\bar{B}C\bar{D}$
1	0	1	1	1	$\rightarrow A\bar{B}CD$
1	1	0	0	1	$\rightarrow AB\bar{C}\bar{D}$
1	1	0	1	1	$\rightarrow AB\bar{C}D$
1	1	1	0	1	$\rightarrow ABC\bar{D}$
1	1	1	1	1	$\rightarrow ABCD$



Implement the Circuit with NAND Gates



- Process of converting a SOP circuit from AND/OR to NAND gates as follows
  1. Replace each AND/OR/INVERTER gate by a single NAND gate
  2. Use a NAND gate to invert any single variable that is feeding the final OR gate

Example 4

- Design a logic circuit with inputs P, Q, R so that output S is HIGH whenever P=0 or whenever Q=R=1
  - $S = \overline{P} + QR$

Review Questions

- Write the SOP expression for a circuit with 4 inputs and an output that is to be HIGH only when input A is LOW at the same time that exactly two other inputs are LOW
- $x = \overline{A}BC\overline{D} + \overline{A}B\overline{C}\overline{D} + \overline{A}\overline{B}CD$
- Implement the expression of the above question using all NAND gates. How many gates are required?
- 8

Karnaugh Map

- Tool to simplify a logic expression

Karnaugh Map (K Map) Method

- K Map that shows the relationship between inputs & outputs, is a method used to simplify the Boolean function
- Each cell of K-map expresses a value of function F (0, 1 or x), corresponding to a row of the Truth Table
- Horizontally & vertically adjacent squares differ only in one variable
  - Note that each square in the top row is considered to be adjacent to a corresponding square in the bottom row.
- Once a K map has been filled with 0s and 1s, the SOP expression for the output can be obtained by ORing together those squares that contain a 1.

K-map for 2 and 3 variables

A	B	X
0	0	1 → $\bar{A}\bar{B}$
0	1	0
1	0	0
1	1	1 → AB

$$x = \bar{A}\bar{B} + AB$$

	$\bar{B}$	B
$\bar{A}$	1	0
A	0	1

A	B	C	X
0	0	0	1 → $\bar{A}\bar{B}\bar{C}$
0	0	1	1 → $\bar{A}\bar{B}C$
0	1	0	1 → $\bar{A}B\bar{C}$
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1 → $AB\bar{C}$
1	1	1	0

$$X = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + AB\bar{C}$$

	$\bar{C}$	C
$\bar{A}\bar{B}$	1	1
$\bar{A}B$	1	0
AB	1	0
$A\bar{B}$	0	0

K-map for 4 variables

A	B	C	D	X
0	0	0	0	0
0	0	0	1	1 → $\bar{A}\bar{B}\bar{C}D$
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1 → $\bar{A}B\bar{C}D$
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1 → $AB\bar{C}D$
1	1	1	0	0
1	1	1	1	1 → ABCD

$$X = \bar{A}\bar{B}\bar{C}D + \bar{A}B\bar{C}D + AB\bar{C}D + ABCD$$

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	1	0	0
$\bar{A}B$	0	1	0	0
AB	0	1	1	0
$A\bar{B}$	0	0	0	0

Fill values in K-map

- The function F is given
  - by the Truth table
  - in the Standard form 1 (Σ-form)
  - in the Standard form 2 (Π-form)
  - in algebraic expression

Fill values from Truth Table

- If the function F is based on the Truth Table
  - Fill "0", "1" or "x" in squares having the same binary combinations in Truth Table.

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	x
1	1	0	x
1	1	1	0

F: C \ AB	00	01	11	10
0	0 (0)	1 (2)	X (6)	1 (4)
1	0 (1)	0 (3)	0 (7)	X (5)

Fill values from standard SOP

- If the function F is given in the standard SOP form (Σ-form)
  - Fill "1" in squares corresponding to minterms,
  - Fill "x" in squares corresponding to "don't care"
  - Fill "0" in the remained squares
- We can fill only two symbols "1" and "x" in the K-map. The blank squares are implicit.

$$F(A,B,C,D) = \sum(2,3,5,8,11,13,14) + d(1,4,15)$$

F: CD \ AB	00	01	11	10
00	0 (0)	X (4)	0 (12)	1 (8)
01	X (1)	1 (5)	1 (13)	0 (9)
11	1 (3)	0 (7)	X (15)	1 (11)
10	1 (2)	0 (6)	1 (14)	0 (10)

Fill values from standard POS

- If the function F is given in the standard POS form (Π-form)
  - Fill "0" in squares corresponding to maxterms,
  - Fill "x" in squares corresponding to "don't care"
  - Fill "1" in the remained squares.
- We can fill only two symbols "0" and "x" in the K-map. The blank squares are implicit.

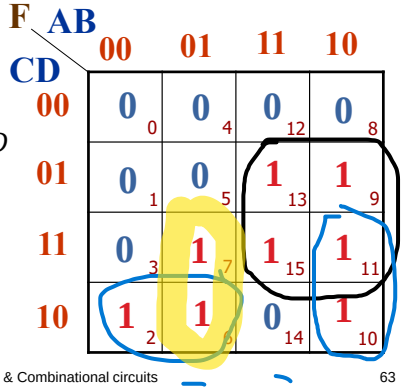
$$F(A,B,C,D) = \prod(4,5,12,14,15) \cdot d(3,7,8,11)$$

F: CD \ AB	00	01	11	10
00	1 (0)	0 (4)	0 (12)	X (8)
01	1 (1)	0 (5)	1 (13)	1 (9)
11	X (3)	X (7)	0 (15)	X (11)
10	1 (2)	1 (6)	0 (14)	1 (10)

Fill values from algebraic form

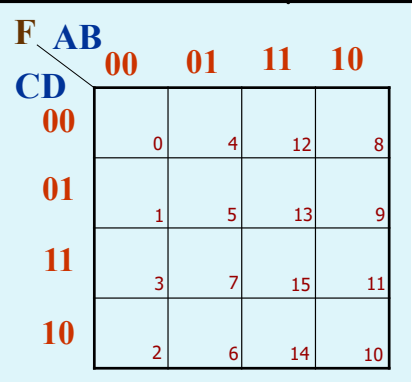
- If the function F is given in algebraic form
  - Convert it into Standard SOP or POS and then fill in the K-map.
  - If it has the form of SOP, transform it into standard SOP.
  - If it has the form of POS, transform it into standard POS.

$F(A,B,C,D) = A\overline{B}CD + \overline{A}BC + \overline{B}C\overline{D} + AD$
$= \sum(2,6,7,9,10,11,13,15)$



$\overline{A}C\overline{D} + AB\overline{C} + AD + AB\overline{C}$

K-map – Looping



- Looping is a process combinin
  contain 1s. The output expres
  looping
- Adjacent squares
  - 2 squares are adjacent if they lie c
    symmetric through axis. The featu
    they are corresponding to two minterms (or maxterms) which are different in only 1 bit
  - 4 squares are adjacent if they consist of 2 adjacent squares and each square of this group is adjacent to one square of the other
  - The above definition is applied similarly for 8 adjacent squares and 2^n adjacent squares
  - The adjacent squares are grouped if they have the same 1 or 0

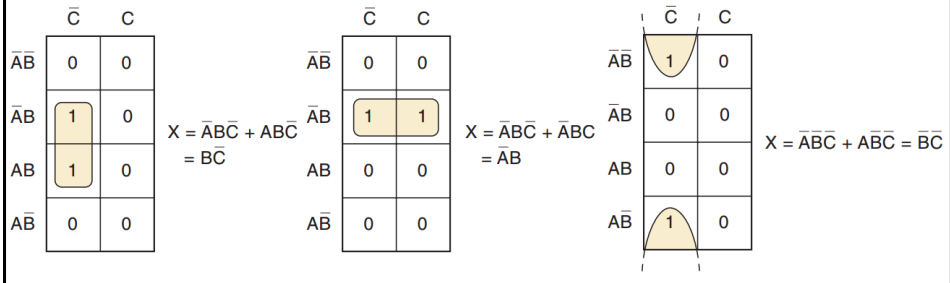
Loop of 2 adjacent squares "1"

- When grouping the adjacent squares having the same value "1", we obtain a product of variables having the same value:
  - 0 → variables are in complemented.
  - 1 → variables are not complemented.

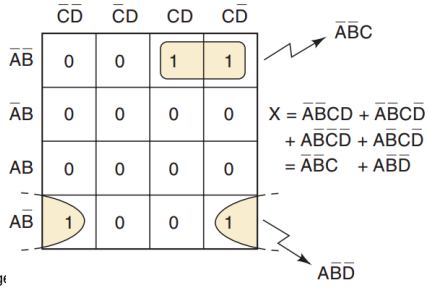
  (Variable with different values are omitted.)
- Because two adjacent squares have the binary combinations which are different in one variable

⇒ grouping two adjacent squares can remove one variable.

Grouping 2 adjacent squares "1"



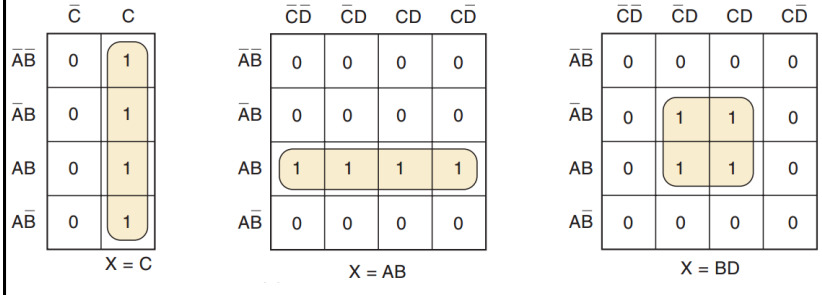
Looping a pair of adjacent 1s in a K map eliminates the variable that appears in complemented and uncomplemented form.



Grouping the adjacent cells

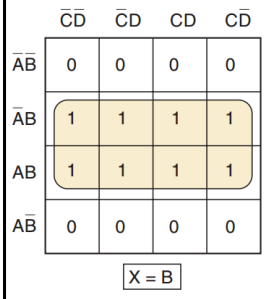
- Similarly,
  - When grouping 4 adjacent cells ⇒ remove 2 variables.
  - When grouping 8 adjacent cells ⇒ remove 3 variables.
- Generally,
  - When grouping 2^n adjacent cells ⇒ remove n variables

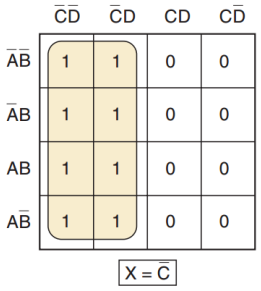
Grouping 4 adjacent squares "1"

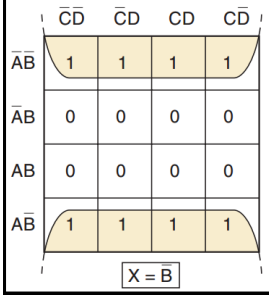


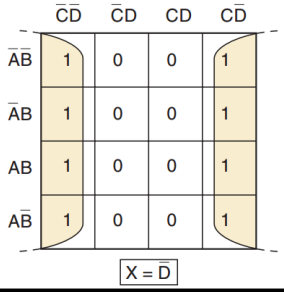
Looping a quad of adjacent 1s eliminates two variables that appears in both complemented and uncomplemented form

Grouping 8 adjacent squares "1"









Looping an octet of adjacent 1s eliminates three variables that appears in both complemented and uncomplemented form

Rule for loops of any size

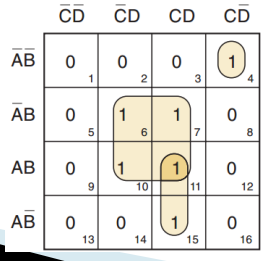
- When a variable appears in both complemented & uncomplemented form within a loop, that variable is eliminated from the expression.
- Variables that are the same for all squares of the loop must appear in the final expression

Complete Simplification Process

1. Construct the K map and place 1s and 0s in the squares according to the truth table
2. Loop the isolated 1s which are not adjacent to any other 1s (single loops)
3. Loop any pair which contains a 1 adjacent to only one other 1 (double loops)
4. Loop any octet even if it contains one or more 1s that have already been looped
5. Loop any quad that contains one or more 1s that have not already been looped, **making sure to use the minimum number of loops**
6. Loop any pairs necessary to include any 1s that have not yet been looped, **making sure to use the minimum number of loops**
7. Form the OR sum of all the terms generated by each loop

Example 1

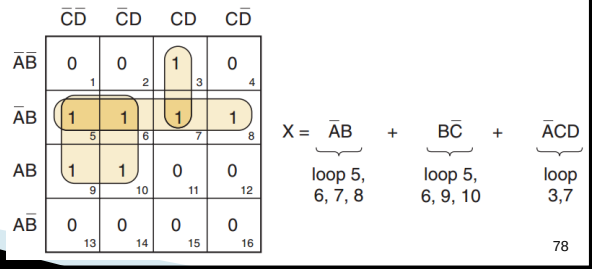
- **Step 1.** the map was obtained from the problem truth table
- **Step 2.** Square 4 is the only square containing a 1 that is not adjacent to any other 1. It is looped and is referred to as loop 4.
- **Step 3.** Square 15 is adjacent **only** to square 11. This pair is looped and referred to as loop 11, 15.
- **Step 4.** There are no octets.
- **Step 5.** Squares 6, 7, 10, and 11 form a quad. This quad is looped (loop 6, 7, 10, 11). Note that square 11 is used again, even though it was part of loop 11, 15.
- **Step 6.** All 1s have already been looped.
- **Step 7.** Each loop generates a term in the expression for X.



$$X = \underbrace{\bar{A}\bar{B}C\bar{D}}_{\text{loop 4}} + \underbrace{ACD}_{\text{loop 11, 15}} + \underbrace{BD}_{\text{loop 6, 7, 10, 11}}$$

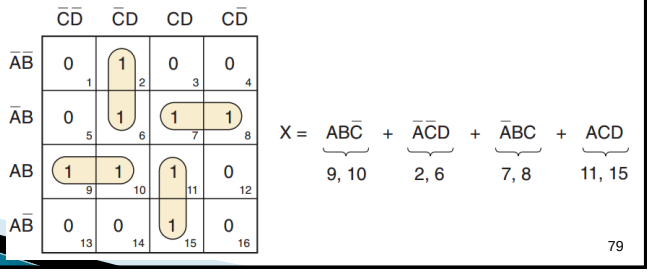
Example 2

- **Step 1.** the map was obtained from the problem truth table
- **Step 2.** There are no isolated 1s.
- **Step 3.** The 1 in square 3 is adjacent only to the 1 in square 7. Looping this pair (loop 3, 7) produces the term $\overline{A}CD$.
- **Step 4.** There are no octets.
- **Step 5.** There are two quads. Squares 5, 6, 7, and 8 form one quad. Looping this quad produces the term $\overline{A}B$. The second quad is made up of squares 5, 6, 9, and 10. This quad is looped because it contains two squares that have not been looped previously. Looping this quad produces $B\overline{C}$.
- **Step 6.** All 1s have already been looped.
- **Step 7.** Each loop generates a term in the expression for X.



Example 3

- **Step 1.** the map was obtained from the problem truth table
- **Step 2.** There are no isolated 1s.
- **Step 3.** The 1 in square 2 is adjacent only to the 1 in square 6. This pair is looped to produce $\overline{A}CD$. Similarly, square 9 is adjacent only to square 10. Looping this pair produces $\overline{A}B\overline{C}$. Likewise, loop 7, 8 and loop 11, 15 produce the terms $\overline{A}BC$ and ACD, respectively.
- **Step 4.** There are no octets.
- **Step 5.** There is one quad formed by squares 6, 7, 10, and 11. This quad, however, is **not** looped because all the 1s in the quad have been included in other loops.
- **Step 6.** All 1s have already been looped.
- **Step 7.** Each loop generates a term in the expression for X.

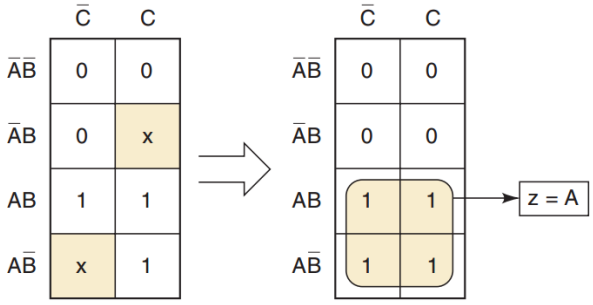


Don't-care Condition

- Don't-Care Conditions are certain input conditions for which there are no specified output levels
- Don't-care conditions should be changed to either 0 or 1 to produce K-map looping that yields the simplest expression
- Example

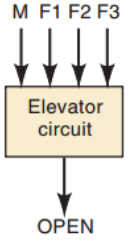
A	B	C	z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	x
1	0	0	x
1	0	1	1
1	1	0	1
1	1	1	1

"don't care"



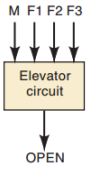
Example

- Let's design a logic circuit that controls an elevator door in a three-story building.
  - The circuit has four inputs.
  - M is a logic signal that indicates when the elevator is moving (M=1) or stopped (M=0).
  - F1, F2, and F3 are floor indicator signals that are normally LOW, and they go HIGH only when the elevator is positioned at the level of that particular floor.
    - For example, when the elevator is lined up level with the second floor, F2=1 and F1=F3=0.
  - The circuit output is the OPEN signal, which is normally LOW and will go HIGH when the elevator door is to be opened.



Elevator example

- Because the elevator cannot be lined up with more than one floor at a time, only one of the floor inputs can be HIGH at any given time.
- When M=1 the elevator is moving, so OPEN must be a 0 because we do not want the elevator door to open.
- When M=0 (elevator stopped) we want OPEN=1 provided that one of the floor inputs is 1.



M	F1	F2	F3	OPEN
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	X
0	1	0	0	1
0	1	0	1	X
0	1	1	0	X
0	1	1	1	X
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	X
1	1	0	0	0
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

	$\overline{F2}\,\overline{F3}$	$\overline{F2}F3$	$F2F3$	$F2\overline{F3}$
$\overline{M}\,\overline{F1}$	0	1	X	1
$\overline{M}F1$	1	X	X	X
$M\,F1$	0	X	X	X
$M\,\overline{F1}$	0	0	X	0

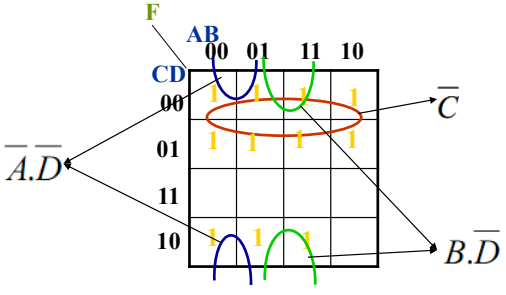
	$\overline{F2}\,\overline{F3}$	$\overline{F2}F3$	$F2F3$	$F2\overline{F3}$
$\overline{M}\,\overline{F1}$	0	1	1	1
$\overline{M}F1$	1	1	1	1
$M\,F1$	0	0	0	0
$M\,\overline{F1}$	0	0	0	0

OPEN = $\overline{M}$ (F1 + F2 + F3)

Simplify the Boolean function using K-map

Simplify the following function:

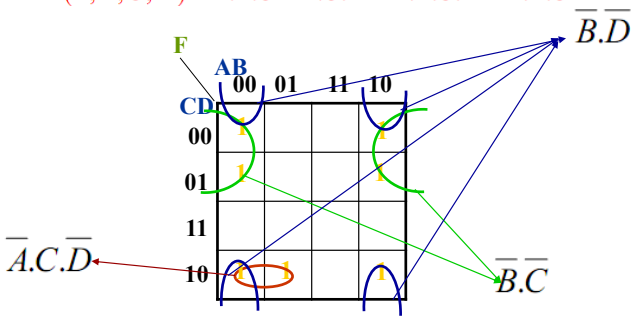
$$F(A,B,C,D)=\sum(0,1,2,4,5,6,8,9,12,13,14)$$



Simplify the Boolean function using K-map

Simplify the following function:

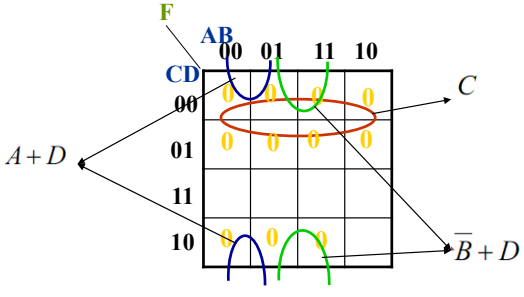
$$F(A,B,C,D) = \overline{A}.\overline{B}.\overline{C} + \overline{B}.C.\overline{D} + \overline{A}.B.C.\overline{D} + A.\overline{B}.\overline{C}$$



Simplify the Boolean function using K-map

Simplify the following function:

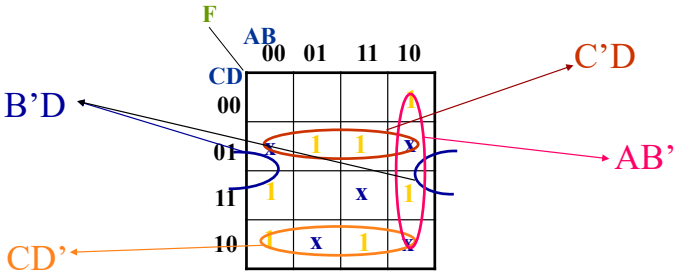
$$F(A,B,C,D) = \prod(0,1,2,4,5,6,8,9,12,13,14)$$



Simplify the Boolean function using K-map

Simplify the following function:

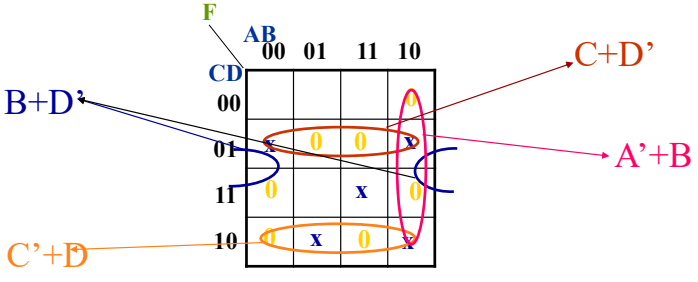
F(A,B,C,D) = Σ(2,3,5,8,11,13,14) + d(1,6,9,10,15)



Simplify the Boolean function using K-map

Simplify the following function:

F(D,C,B,A) = Π(2,3,5,8,11,13,14).d(1,6,9,10,15)



Review

Simplify the following functions:

F

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

F

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

F

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

Review

Simplify the following functions:

F

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

F

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

F

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

Review

Simplify the following functions:

F

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

F

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

F

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

Review

Simplify the following functions:

F

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

F

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

F

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

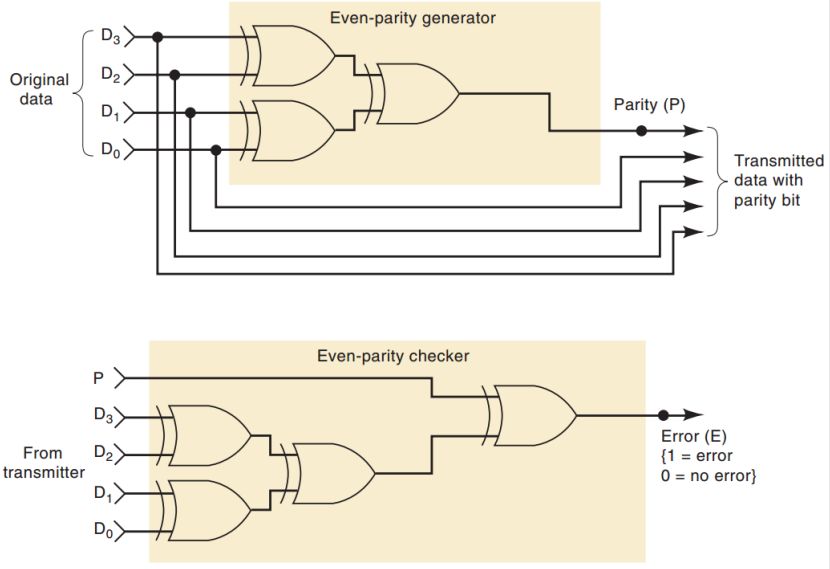
K Map Method – Summary

- Compared to the algebraic method, the K-map process is a more orderly process requiring fewer steps and always producing a minimum expression
  - Algebraic simplification is considered as the trial-and-error process
- For the circuits with large numbers of inputs (larger than four), other more complex techniques are used

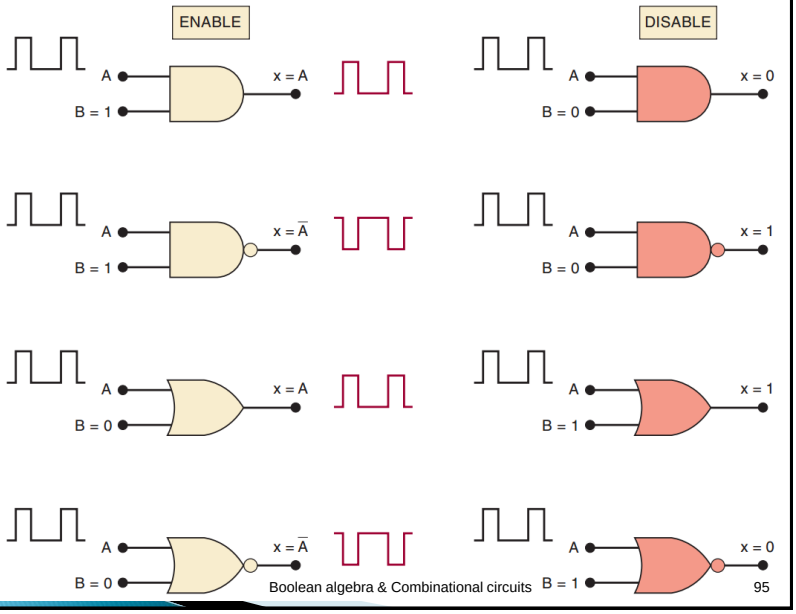
Useful Combinational Logic Circuits

- Parity generator and checker
- Enable/Disable circuits

Parity Generator and Checker

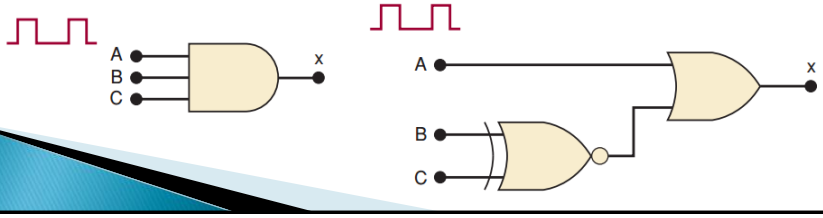


Enable/Disable Circuits



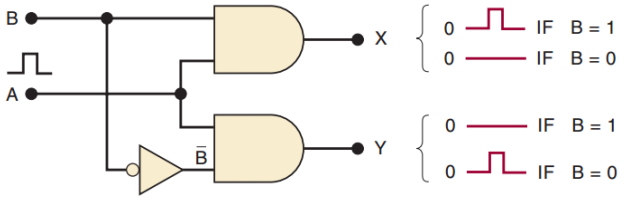
Enable/Disable Circuits – Example

- Design a logic circuit that will allow a signal to pass to the output only when control inputs B and C are both HIGH; otherwise, the output will stay LOW
  - An AND gate should be used because the signal is to be passed without inversion, and the disable output condition is a LOW. Because the enable condition must occur only when B=C=1, a three-input AND gate is used
- Design a logic circuit that allows a signal to pass to the output only when one, but not both, of the control inputs are HIGH; otherwise, the output will stay HIGH
  - An OR gate is used because we want the output disable condition to be a HIGH, and we do not want to invert the signal. Control inputs B and C are combined in an XNOR gate. When B and C are different, the XNOR sends a LOW to enable the OR gate. When B and C are the same, the XNOR sends a HIGH to disable the OR gate.



Enable/Disable Circuits – Example

- Designing a logic circuit with input signal A, control input B and outputs X and Y to operate as follows:
  - When B = 1, X = A and Y = 0
  - When B = 0, X = 0 and Y = A
  - Pulse-steering circuit



Summary

- Boolean algebra
  - SOP and POS
  - Simplifying Boolean expression
- Design of a combinational Logic circuit
  - construct its truth table,
  - convert it to a SOP
  - simplify using Boolean algebra or K mapping,
  - implement
- K map: a graphical method for representing a circuit's truth table and generating a simplified expression
- Some useful logic circuits
  - Parity generator and checker
  - Enable/disable: Each of the basic gates can be used to enable or disable the passage of an input signal to its output