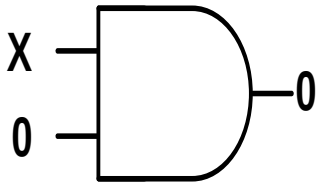
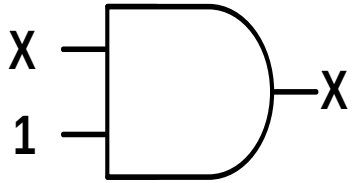
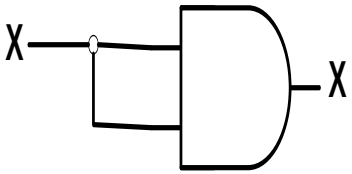
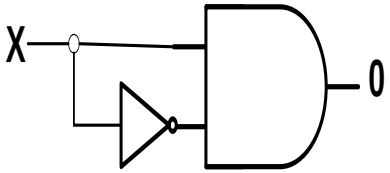
Chapter

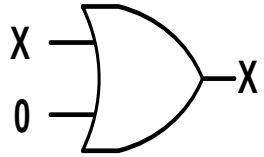
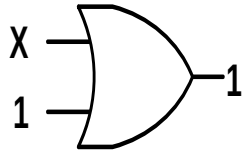
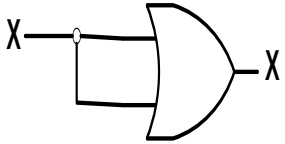
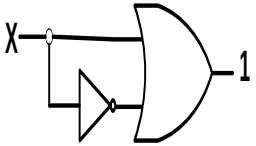
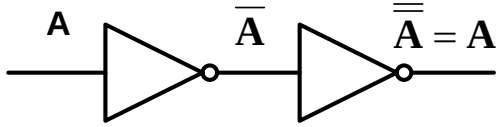
Boolean Algebra and Logic Simplification

Single Variable theorems

	THEOREMs	LOGIC DIAGRAM
1	$X \cdot 0 = 0$	
2	$X \cdot 1 =$ X	
3	$X \cdot X =$ X	
4	$X \cdot \overline{X} = 0$	

AND Laws

OR Laws

	THEOREMs	LOGIC DIAGRAM
5	$X + 0 = X$	
6	$X + 1 = 1$	
7	$X + X = X$	
8	$X + \overline{X} = 1$	
9	$\overline{\overline{A}} = A$	

OR Laws

Multivariable Theorems

The theorems presented below invoke more than one variable.

- **Commutative Laws: Commutative laws allow change in position of AND or OR variables. There are two commutative laws.**

10 $A \cdot B = B \cdot A$ Commutative law for AND operation

- **This law can be extended to any number of variables. For example,**

$$A \cdot B \cdot C = B \cdot C \cdot A = C \cdot A \cdot B = B \cdot A \cdot C$$

11 $A+B=B+A$ Commutative law for OR operation

- **This law can be extended to any number of variables. For example,**

$$A + B + C = B + C + A = C + A + B = B + A + C$$

- **Associative Laws : The associative laws allow grouping of variables. There are two associative laws.**

12 $A(BC)=(AB)C$ Associative law for AND operation

13 $A+(B+C)=(A+B)+C$ Associative law for OR operation

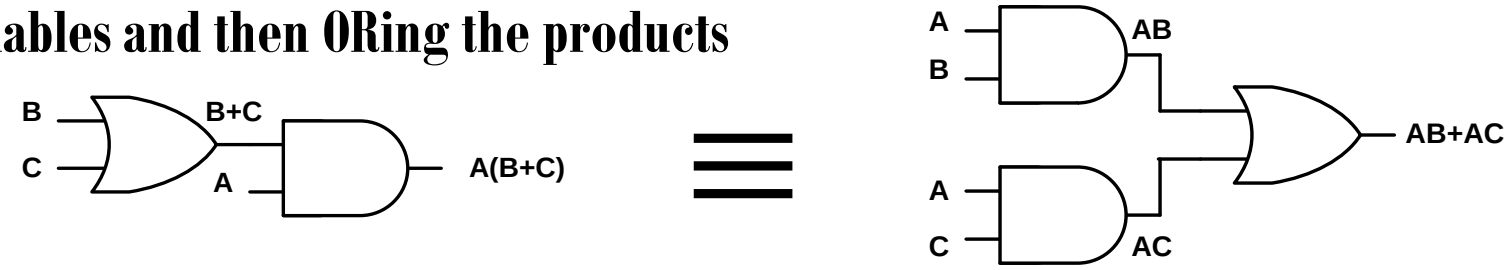
- **This law can be extended to any number of variables. For example,**

$$A +(B + C + D) = (A+ B +C) + D =(A + B) + (C + D)$$

❑ **Distributive Law:** The distributive laws allow factoring or multiplying out of expressions. There are two distributive laws.

14 $A(B+C) = AB + AC$

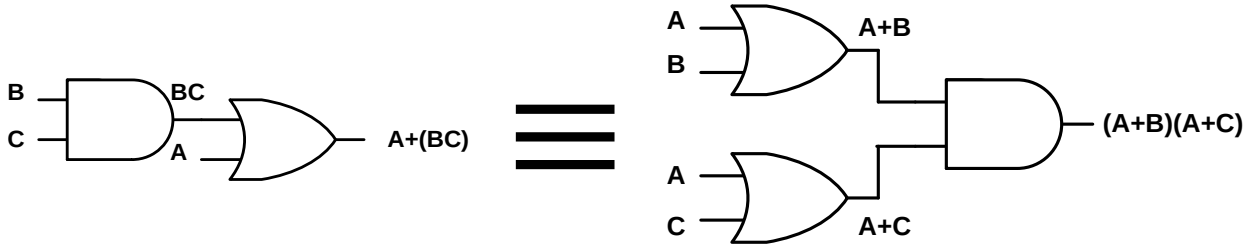
❑ This law states that ORing of several variables and ANDing the result with a single variable is equivalent to ANDing that single variable with each of the several variables and then ORing the products



15 $A+BC=(A+B)(A+C)$

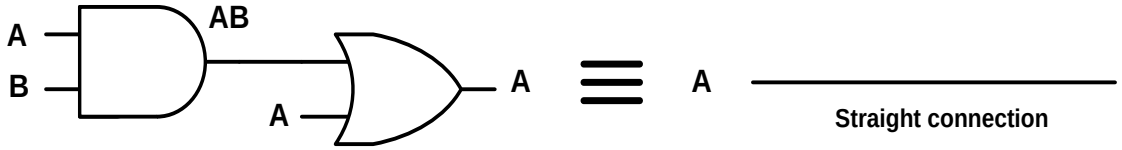
This can be proved algebraically as shown below.

RHS = (A + B) (A + C)
= AA + AC + BA + BC
= A + AC + AB + BC
= A(1 + C + B) + BC
= A · 1 + BC
= A + BC
= LHS



16 $A + A \cdot B = A$

- This law states that ORing of a variable (A) with the AND of that variable (A) and another variable (B) is equal to that variable itself (A).



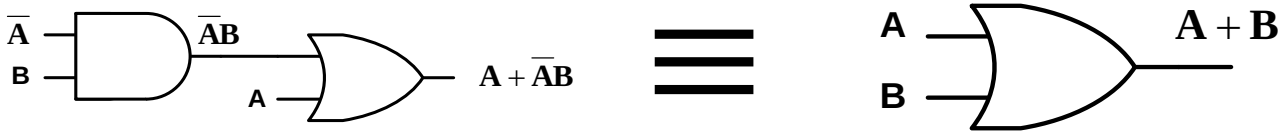
- Algebraically, we have

$$A + A \cdot B = A (1 + B) = A \cdot 1 = A$$

- Therefore, $A + A \cdot \text{Any term} = A$

17 $A + \overline{A}B = A + B$

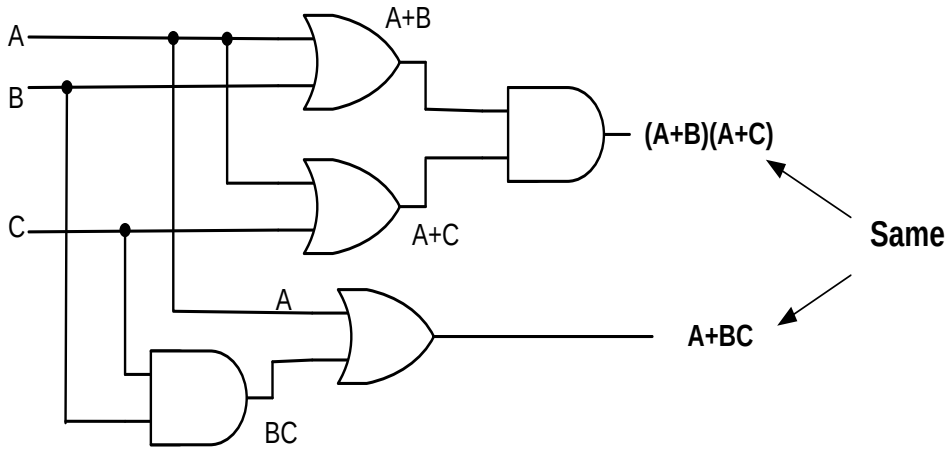
- This law states that ORing of a variable with the AND of the complement of that variable with another variable, is equal to the ORing of the two variables.



- This law can also be proved algebraically as shown below.

$$A + \overline{A}B$$

Proofing Theorem 15: (A+B) (A+C)= A+BC



$$(A + B)(A + C)$$

$$\Downarrow \qquad \text{Distributive Laws}$$

$$AA + AC + AB + BC$$

$$\Downarrow$$

$$A + AC + AB + BC$$

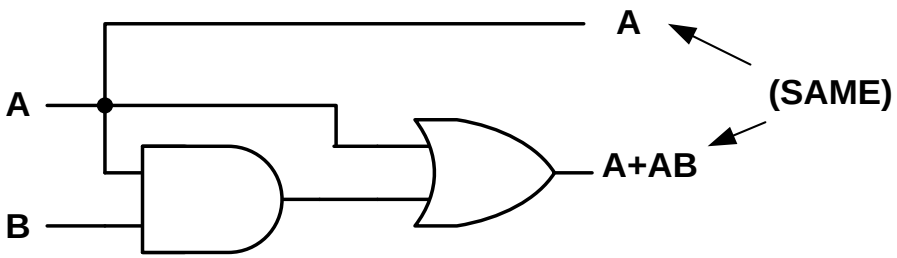
$$\Downarrow$$

$$A + AB + BC$$

$$\Downarrow$$

$$A + BC$$

Proofing: Theorem 16



$$A + AB$$

$\Downarrow$ Factoring x out of both terms

$$A(1 + B)$$

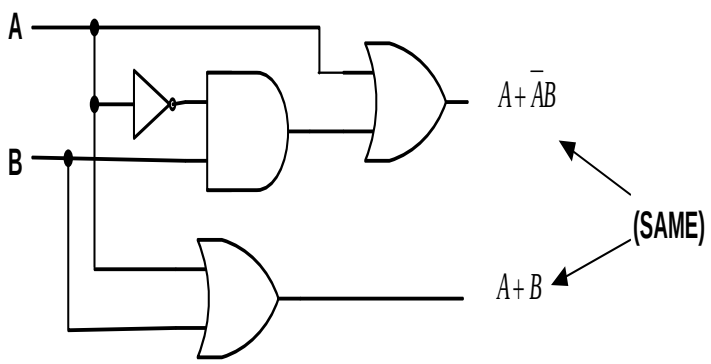
$\Downarrow$ Applying Theorem X+1=1

$$A(1)$$

$\Downarrow$ Applying Theorem X.1=X

$$A$$

Proovfing: Theorem 17



$$A + \overline{A}B$$

$\Downarrow$ Applying Theorem 16 to expand A term

$$A + AB + \overline{A}B$$

$\Downarrow$ Factoring out 2nd & 3rd terms

$$A + B\left(A + \overline{A}\right)$$

$\Downarrow$ Applying theorem $X + \overline{X} = 1$

$$A + B(1)$$

$\Downarrow$ Applying theorem $X \cdot 1 = X$

$$A + B$$

DeMorgan's Theorems

- A great mathematician named DeMorgan contributed two of the most important theorems of Boolean algebra.
- DeMorgan's theorems are extremely useful in simplifying expressions in which a product or sum of variables is inverted. The two theorems are

$$\overline{A + B} = \overline{A} \cdot \overline{B} \qquad 18$$

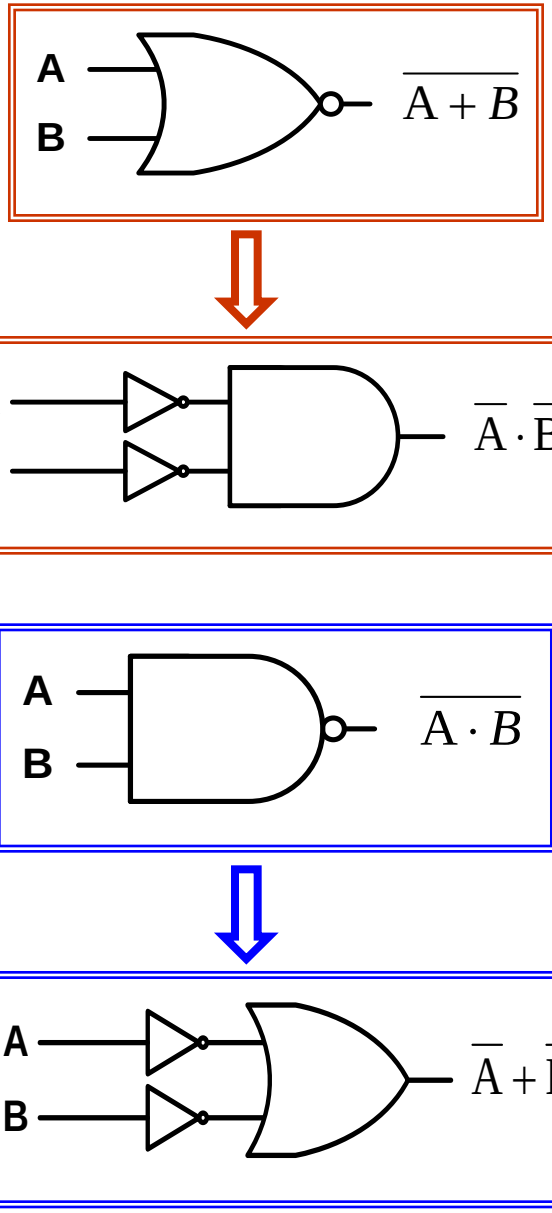
- Theorem (18) says that *when the OR sum of two variables is inverted*, this is the same as *inverting each variable individually and then ANDing these inverted variables*.

$$\overline{A \cdot B} = \overline{A} + \overline{B} \qquad 19$$

- Theorem (19) says that *when the And product of two variables is inverted* ,this is the same as *inverting each variable individually and then ORing them.*

Implications of De Morgan's Theorems

- Let us examine these theorem (18) and (19) from the standpoint of logic circuits.
- First, consider theorem (18) $\overline{A + B} = \overline{A} \cdot \overline{B}$
- The left-hand side of the equation can be viewed as the output of a NOR gate whose inputs are X and Y.
- The right-hand side of the equation, on the other hand, is a result of first inverting both X and Y and then putting them through an AND gate.
- What this means is that an AND gate with INVERTERS on each of its inputs is equivalent to a NOR gate.
- Now consider theorem (19) $\overline{A \cdot B} = \overline{A} + \overline{B}$
- The left hand side the equation can be implemented by a NAND gate with input X and Y.
- The right side can be implemented by first inverting inputs X and Y and then putting them through an OR gate.
- What this means is that an OR gate with INVERTERS on each of its inputs is equivalent to a NAND gate.



Standard Form of Boolean Expressions

- Boolean expression can be converted into one of 2 standards forms:
  - **The sum-of-products (SOP) form**
  - **The product-of-sums (POS) form**
- Standardization makes the evaluation, simplification, and implementation of Boolean expressions more systematic and easier
- **Product term** = a term with the product (Boolean multiplication) of literals
- **Sum term** = a term with the sum (Boolean addition) of literals

Sum- of- Products Form (Minterms)

- The methods of logic- circuit simplification and design that we will study require logic expression to be in a sum- of- products form (**SOP**).
- Some examples of this form are:

1. $ABC + \overline{A}B\overline{C}$
2. $AB + \overline{A}B\overline{C} + \overline{C}D + D$
3. $\overline{A}B + C\overline{D} + EF + GH$

- Each of this sum- of- product expression consists of two or more AND terms (products) that are ORed together.
- Each AND term consists of one or more variables appearing in either complemented or Uncomplemented form.
- Note that in a sum- of- products expression; one inversion sign can not cover more than one variable in a term.
- Example : we can not have

$$\overline{ABC} \text{ or } \overline{RST}$$

Product- of- Sums (Maxterms)

- There is another general form for logic expressions that is sometimes used in logic- circuit design.
- It is called the Product- of- sums form (**POS**), and it consists of two or more OR terms (sums) that are ANDed together.
- Each OR term contains one or more variables in complemented or uncomplemented form. Here are some product- of- sums expressions;

1. $(A + \overline{B} + C)(A + C)$
2. $(A + \overline{B})(\overline{C} + D)F$

- The methods of circuit simplification and design, which we will be using, are based on the sum- of- products form, so we will not be doing much with the products- of- sum form.
- It will, however, occur from time to time in some logic circuits, which have a particular structure.

CONVERSION FROM SOP to POS and vice-versa

- A standard SOP form can always be converted to a standard POS form, by treating missing minterms of the SOP form as the maxterms of the POS form.
- Similarly, a standard POS form can always be converted to a standard SOP form, by treating the missing maxterms of the POS form as the minterms of the corresponding SOP form.

EXPANSION OF A BOOLEAN EXPRESSION TO SOP FORM

- The following steps are followed for the expansion of a Boolean expression in SOP form in standard SOP form:
  1. Write down all the terms.
  2. If one or more variables are missing in any term, expand that term by multiplying it with the sum of each one of the missing variable and its complement
  3. Drop out the redundant terms
- *Also,* the given expression can be directly written in terms of its minterms by using the following procedure:
  1. Write down all the terms.
  2. Put Xs in terms where variables must be inserted to form a minterm.
  3. Replace the non-complemented variables by 1s and the complemented variables by 0s, and use all combinations of Xs in terms of 0s and 1s to generate minterms.
  4. Drop out all the redundant terms.

Example#1

Expand $\overline{A} + \overline{B}$ to minterms and maxterms

Solution

- The given expression is a two-variable function.
- In the first term $\overline{A}$, the variable B is missing; so, multiply it by $B + \overline{B}$.
- In the second term $\overline{B}$, the variable A is missing; so, multiply by $(A + \overline{A})$.
- Therefore,

$$\begin{aligned}
\overline{A} + \overline{B} &= \overline{A}(B + \overline{B}) + \overline{B}(A + \overline{A}) \\
&= \overline{A}B + \overline{A}\,\overline{B} + A\overline{B} + \overline{A}\,\overline{B} \\
&= \overline{A}B + \overline{A}\,\overline{B} + A\overline{B} \\
&= 01 + 00 + 10 \\
&= m_1 + m_0 + m_2 \\
&= \sum m(0,1,2)
\end{aligned}$$

- The minterm m_3 is missing in the SOP form.
- Therefore, the maxterm M_3 will be present in the POS form.
- Hence the POS form is M_3 **i.e.** $\overline{A} + \overline{B}$

2nd method

$$\overline{A} + \overline{B} = \overline{A}X + X\overline{B}$$

$$= 0X + X0$$

$$= 00 + 01 + 00 + 10$$

$$= 00 + 01 + 10$$

$$= m_0 + m_1 + m_2$$

$$= \sum m(0,1,2)$$

Exercise

- ✓ Expand $A + B\overline{C} + ABC$ to minterms and maxterms

EXPANSION OF A BOOLEAN EXPRESSION TO POS FORM

- The expansion of a Boolean expression to the standard POS form is conducted as follows:
  1. If one or more variables are missing in any sum term, expand that term by adding the products of each of the missing term and its complement.
  2. Drop out the redundant terms.
- The given expression can also be written in terms of maxterms by using the following procedure:

  1. Put Xs in terms wherever variables must be inserted to form a maxterm.
  2. Replace the complemented variables by 1 s and the non-complemented variables by 0s and use all combinations of Xs in terms of 0s and 1 s to generate maxterms.
  3. Drop out the redundant terms.

Example#2

Expand $A(\overline{B}+A)B$ to maxterms and minterms

Solution

- The given expression is a two-variable function in the POS form.
- The variable B is missing In the first term A. so, add $B\overline{B}$ to it.
- The second term contains all the variables. So leave it as it is.
- The variable A is missing in the third term B. So, add $A\overline{A}$ to it.
- Therefore,

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

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

or

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

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

$$= (00)\ (01)\ (10)$$

$$= M_0 \cdot M_1 \cdot M_2$$

$$= \prod M(0,1,2)$$

- The maxterm M_3 is missing in the POS form. So, the SOP form will contain only the minterm m_3 i.e. AB

- 2nd method

$$A \rightarrow OX = (00)(01) = M_0 \cdot M_1$$

$$(A + \overline{B}) \rightarrow (01) = M_1$$

$$B \rightarrow X0 = (00)(10) = M_0 \cdot M_2$$

Therefore,

$$A(A + \overline{B})B = \prod M(0,1,2)$$

Exercise

✓ Expand $A(\overline{A}+C)(A+\overline{B}+C)$ to maxterms and minterms

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

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

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

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

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

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

Hence,

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

$$= M_2 \cdot M_3 \cdot M_0 \cdot M_1 \cdot M_4 \cdot M_6$$ *Maxterms* $\prod M(0,1,2,3,4,6)$

Θ $\sum m(5,7)$..... minterms

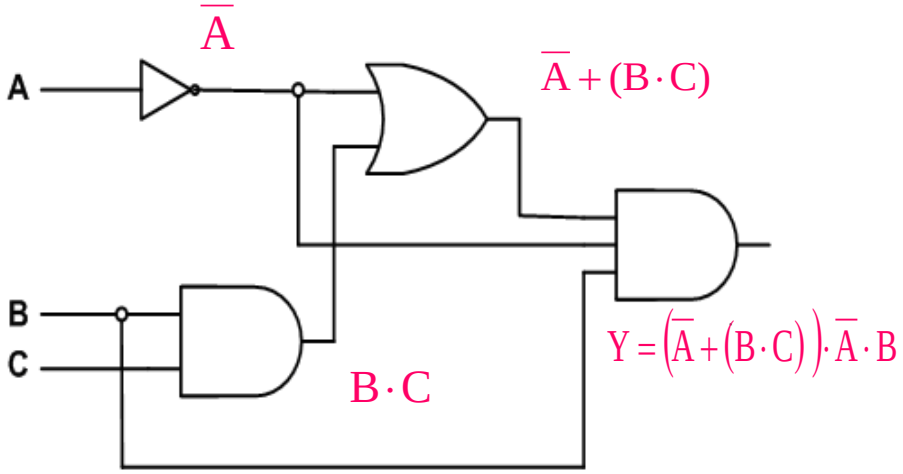
SIMPLIFYING LOGIC CIRCUITS

- Once the expression for a logic circuit has been obtained, we may be able to reduce it to a simpler form containing fewer terms of fewer variables in one or more terms.
- The new expression can then be used to implement a circuit that is equivalent to the original circuit but that contains fewer gates and connections.
- To illustrate, the circuit of fig (a) can be simplified to produce the circuit of fig (b).
- Since both circuits perform the same logic, it should be obvious that the simplest 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. In subsequent sections we will study two methods for simplifying logic circuits:

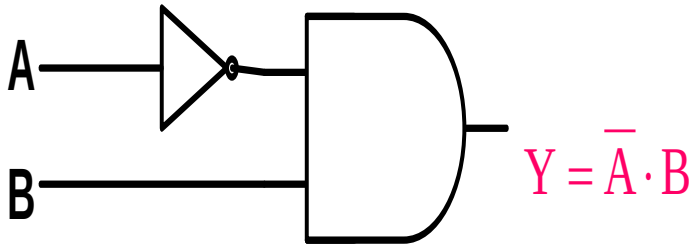
➢ One method will utilize the Boolean algebra theorems (Algebraic Method) and

➢ The other method will utilize Karnaugh mapping (K-map Method)

Why simplification?



(a) unsimplified circuit



(b) Simplified circuit

ALGEBRAIC SIMPLIFICATION

- The Boolean algebra theorems that we studied earlier can be used to help us simplify the expression for logic circuit.
- Unfortunately, it is not always obvious which theorems should be applied in order to produce the simplest result.
- Furthermore, there is no easy way to tell whether the simplified expression is in its simplest form or whether it could have been simplified further.
- Thus, ***algebraic simplification*** often becomes ***a process of trial and error***. With experience, however, one can become adept at obtaining reasonably good results.
- The examples that follow will illustrate many of the ways in which the Boolean theorems can be applied in trying to simplify an expression.
- You should notice that these examples contain two essential steps;

1. The original expression is put in to the sum- of- products form by repeated application of De Morgan's theorems and multiplication of terms.
2. Once it is in this form the product terms are checked for common factors, and factoring is performed wherever possible. Hopefully, the factoring results in the elimination of one or more terms.

Example # 1

Simplify the expression $Z = ABC + A\overline{B}(\overline{\overline{A}\,\overline{C}})$

- ✓ **It is usually a good idea to break down all large inverter signs using De Morgan's theorems and then multiply out all terms**

$$Z = ABC + A\overline{B}(\overline{A} + \overline{C}) \qquad \text{[theorem (19)]}$$

$$= ABC + A\overline{B}(A + C) \qquad \text{[cancel double inversions]}$$

$$= ABC + A\overline{B}A + A\overline{B}C \qquad \text{[multiply out]}$$

$$= ABC + A\overline{B} + A\overline{B}C \qquad \text{[A.A=A]}$$

- ✓ **With the expression now in sum-of-products form, we should look for common factors.**

∴ The first and third terms above have AC in common, which can be factored out:

$$Z = AC(B + \overline{B}) + A\overline{B} \qquad [\, B + \overline{B} = 1 \,]$$

$$Z = AC(1) + A\overline{B}$$

$$Z = AC + A\overline{B} \qquad \text{[factor out A]}$$

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

Example # 2

- ✓ Simplify the expression

$$Z = ABC + AB\overline{C} + A\overline{B}C$$

Solution:

- ✓ We will look at two different ways to arrive at the same result.

***Method 1*:**

- ✓ The first two terms have the product AB in common.

Thus,

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

$$= AB(1) + A\overline{B}C$$

$$= AB + A\overline{B}C$$

- ✓ We can factor the variable A from both terms

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

- ✓ Invoking theorem (15),

$$Z = A(B + C)$$

Method 2:

- ✓ The first two terms have AB in common.
- ✓ The first and the last terms have AC in common.
- ✓ How do we know whether to factor AB from the first two terms or AC from the two-end terms?
- ✓ Actually we can do both by using the ABC term twice.
- ✓ In other words, we can rewrite the expression as

$$Z = ABC + AB\overline{C} + A\overline{B}C + ABC$$

- ✓ The extra term ABC is valid and will not change the value of the expression, Since ABC +ABC = ABC [Theorem(7)]

$$Z = AB(C + \overline{C}) + AC(\overline{B} + B)$$

$$= AB \cdot 1 + AC \cdot 1$$

$$= A(B + C)$$

- ✓ This is the same result as method 1.
- ✓ In fact, the same term can be used more than twice if necessary.

KARNAUGH MAP METHOD

- The Karnaugh map is a graphical device used to simplify a logic equation or to convert a truth table to its corresponding logic circuit in a simple, orderly process.
- Although a Karnaugh map (henceforth abbreviated K map) can be used for problems involving any number of input variables, its practical usefulness is limited to six variables.
- The following discussion will be limited to problems with up to four inputs, since even five-and six- input problems are too involved and are best done by a computer program.

Karnaugh Map Format

1. The truth table gives the value of output X for each combination of input values. The K map gives the same information in a different format. Each case in the truth table corresponds to a square in the K map.
2. The K map squares are labeled so that horizontally adjacent square differs only in one variable.
3. In order for vertically and horizontally adjacent squares to differ in only one variable, the top-to-bottom labeling must be done in the order shown-.

$$\overline{A}\,\overline{B}, \overline{A}B, AB, A\overline{B}$$

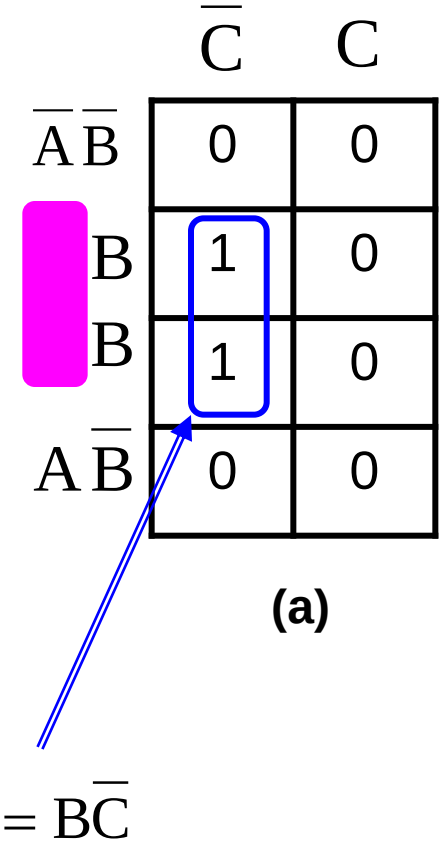
The same is true of the left-to-right labeling.

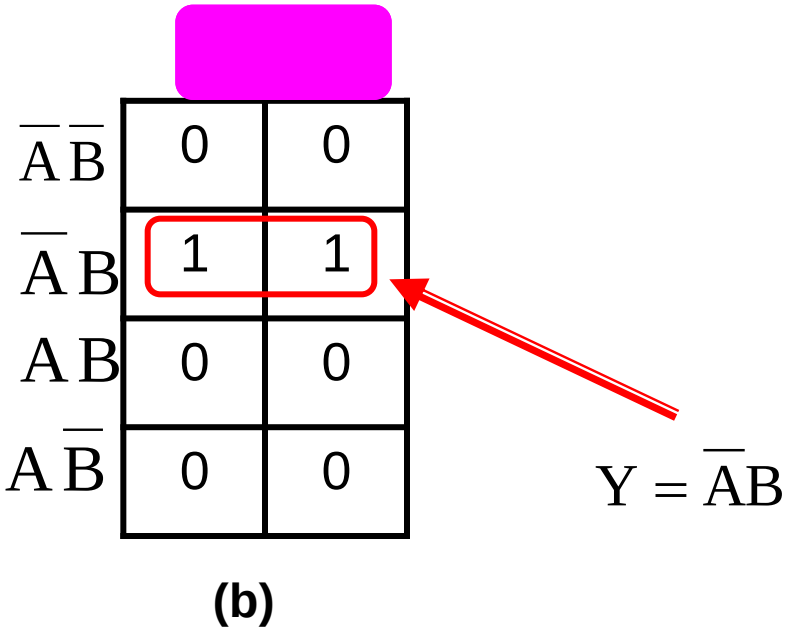
4. Once a K map has been filled with 0s and 1s, the sum-of-product expression for the output X can be obtained by ORing together those squares that contain a 1.

Looping

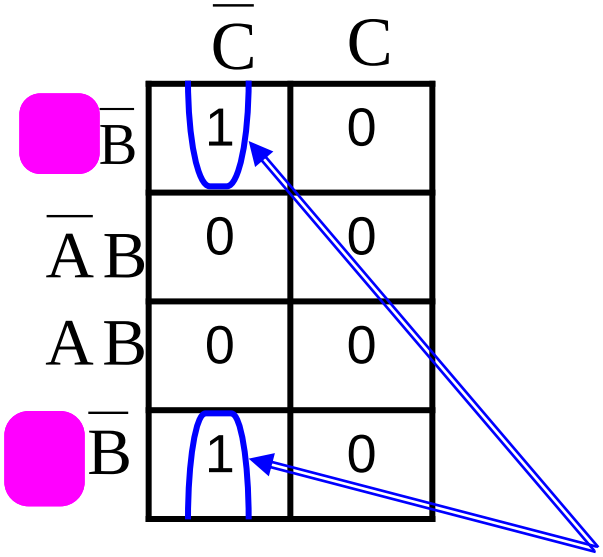
- The expression for output X can be simplified by properly combining those squares in the K map which contain 1s.
- The process for combining these 1s is called looping.

Looping Groups of Two (pairs)





Looping Groups of Two (pairs) *(Continued)*

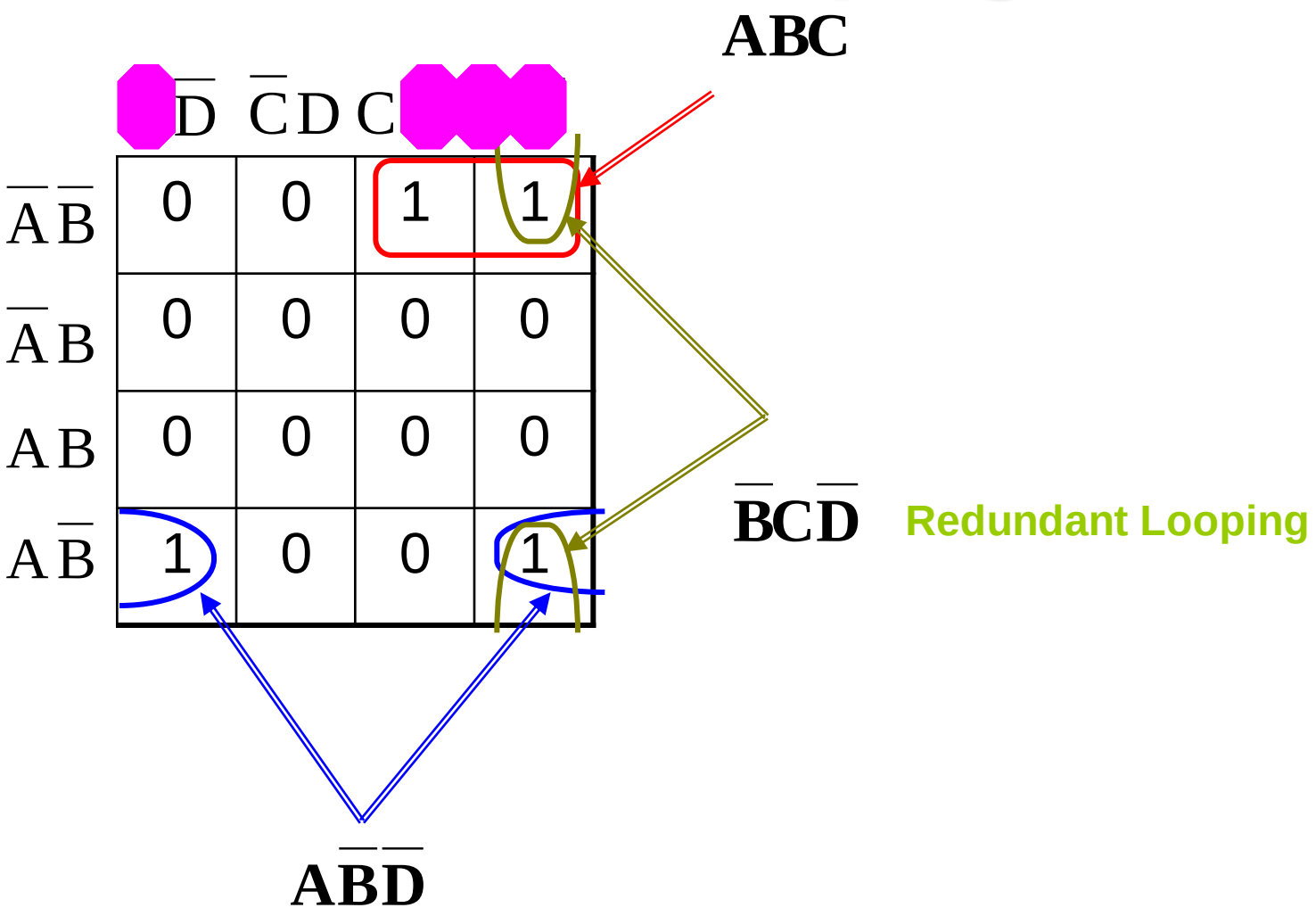


(c)

$$Y = \overline{B}\overline{C}$$

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

Redundant Looping



$$Y = \overline{A}\overline{B}C + A\overline{B}\overline{D}$$

Correct

$$Y = \overline{A}\overline{B}C + A\overline{B}\overline{D} + \overline{B}C\overline{D}$$

Incorrect

Looping Groups of Four (Quads):

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

(a)

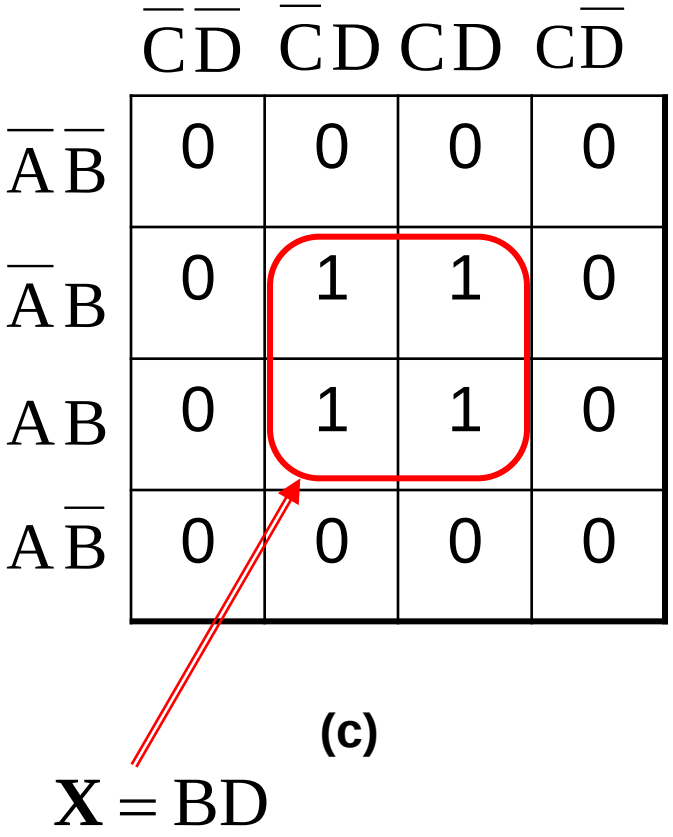
$$X = C\overline{D}$$

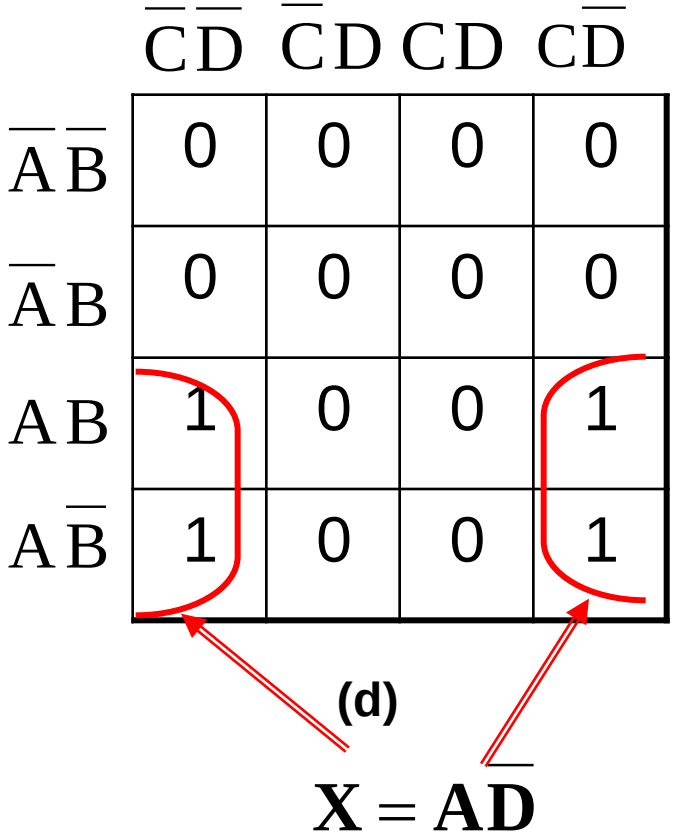
	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	0	0	0	0
$\overline{A}B$	0	0	0	0
AB	1	1	1	1
$A\overline{B}$	0	0	0	0

$$X = AB$$

(b)

Looping Groups of Four (Quads) (continued…)





Looping Groups of Four (Quads) (continued…)

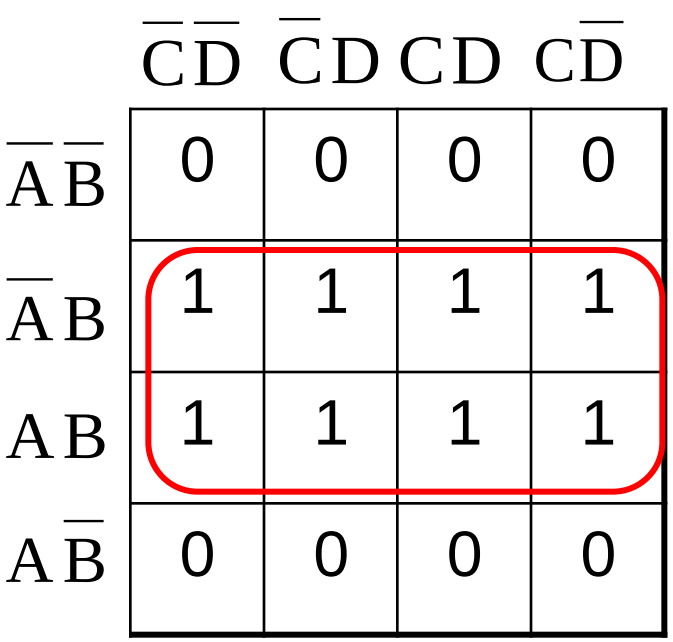
	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	1	0	0	1
$\overline{A}B$	0	0	0	0
AB	0	0	0	0
$A\overline{B}$	1	0	0	1

(e)

$$X = \overline{B}\,\overline{D}$$

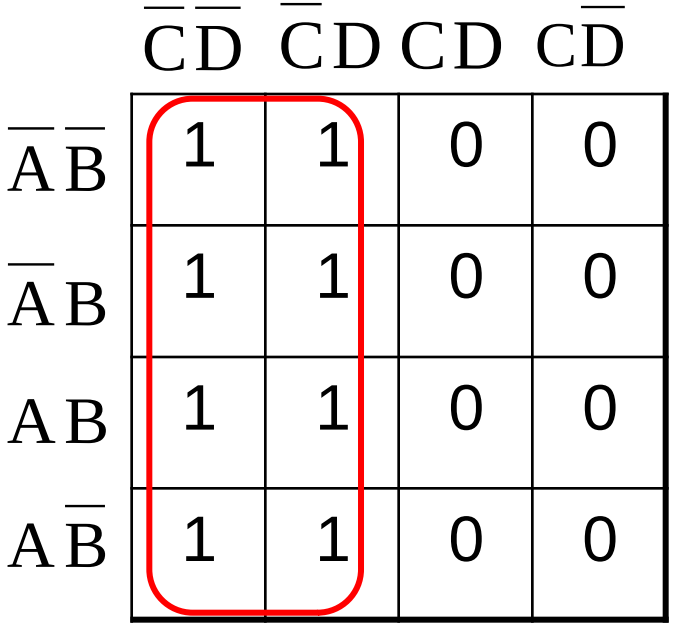
❖ Looping a quad of 1s eliminates the two variables that appear in both complemented and uncomplemented form.

Looping Groups of Eight (octets):



(a)

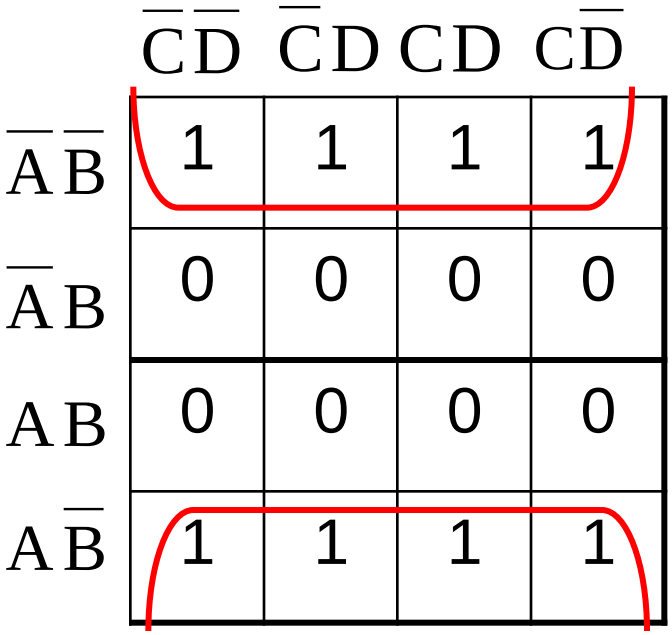
$$X = B$$



(b)

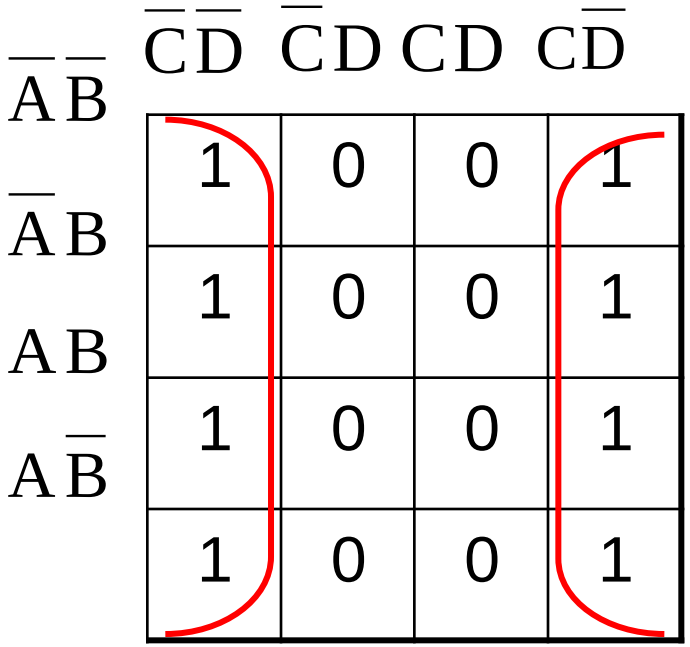
$$X = \overline{C}$$

Looping Groups of Eight (Octets) (cont'd…)



(c)

$$Y = \overline{B}$$



(d)

$$Y = \overline{D}$$

❖ **Looping an octet of 1s eliminates the three variables that appear in both complemented and uncomplemented form.**

Rule for loops of any size

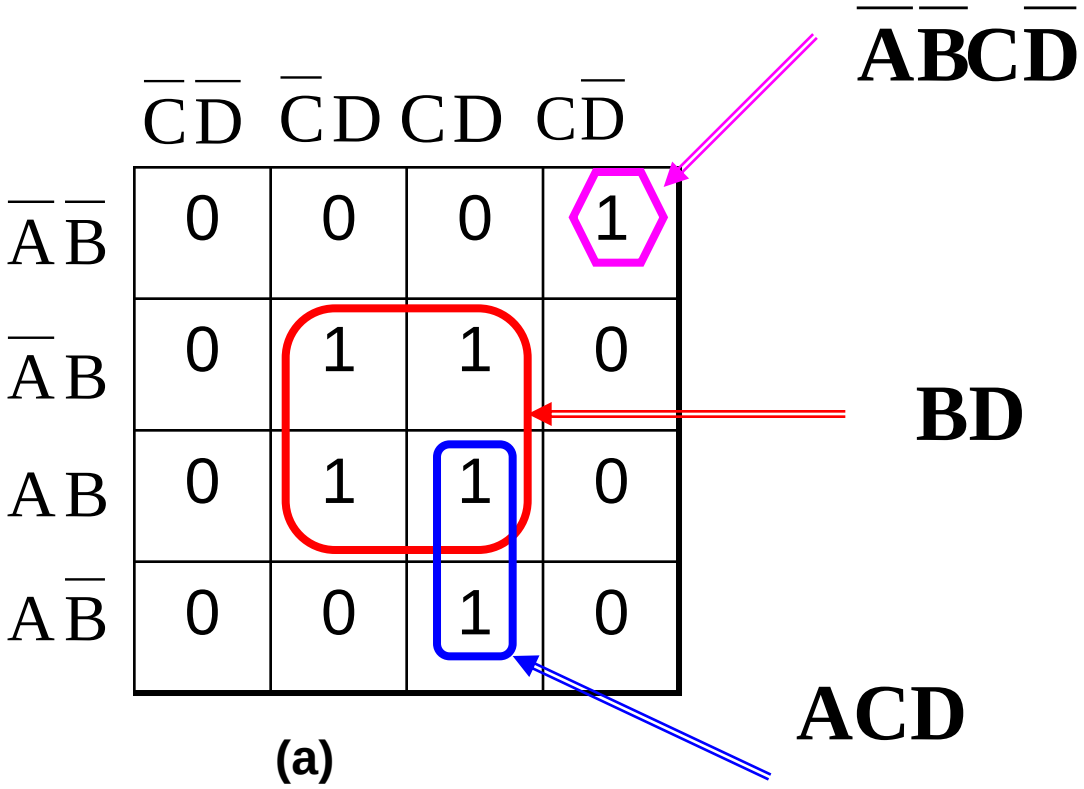
We can summarize the rule for loops of any size:

- When a variable appears in both complemented and 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.
- It should be clear that a larger loop of 1s eliminates more variables. To be exact, a loop of two eliminates one variable, a loop of four eliminates two, and a loop of eight eliminates three.

Complete Simplification Process

1. Construct the K map and place 1s in those squares corresponding to the 1s in the truth table. Place 0s in the other squares.
2. Examine the map for adjacent 1s and loop those 1s, which are not adjacent to any other 1s. These are called isolated 1s.
3. Next, look for those 1s, which are adjacent to only one other 1. Loop any pair containing such a 1.
4. Loop any octet even it contains some 1s that have already been looped.
5. Loop any quad that contains one or more 1s, which have not already been looped, is 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.

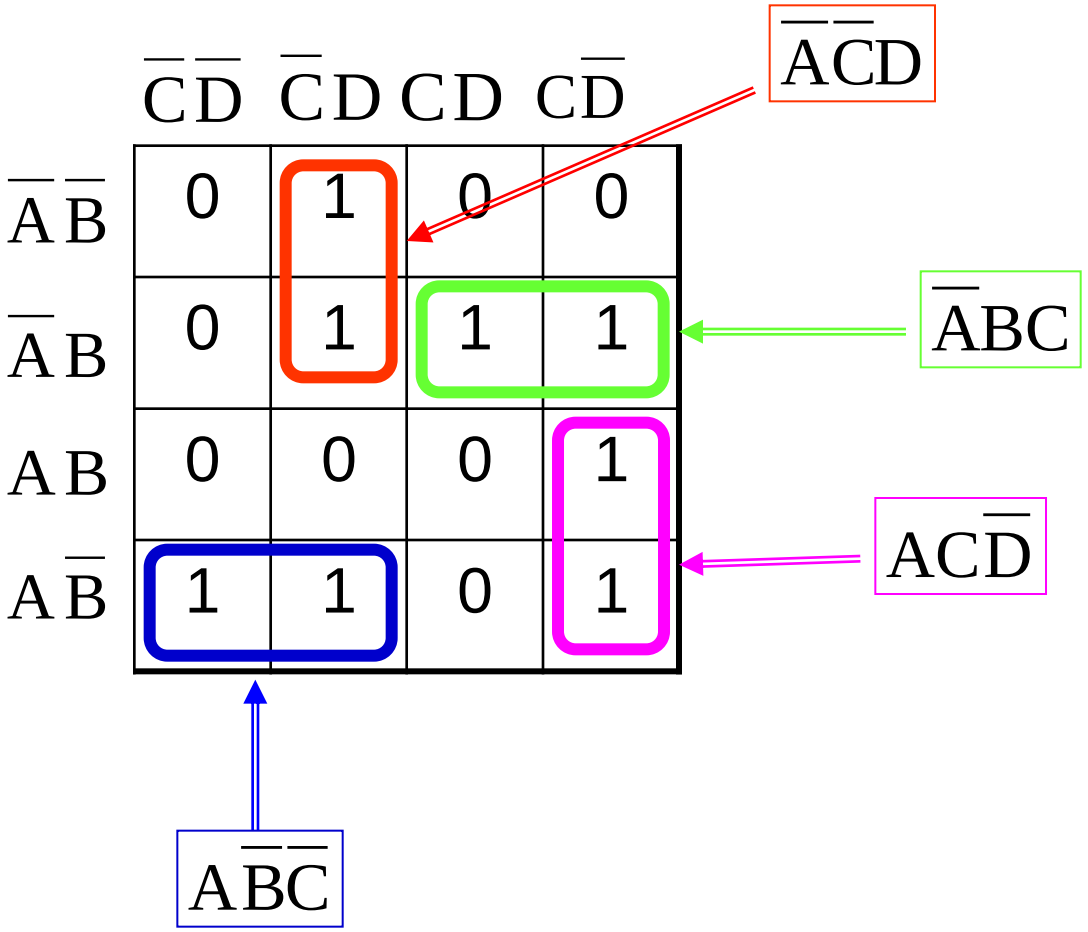
Example#1



$$X = \overline{A}\,\overline{B}\overline{C}\,\overline{D} + ACD + BD$$

Example#2

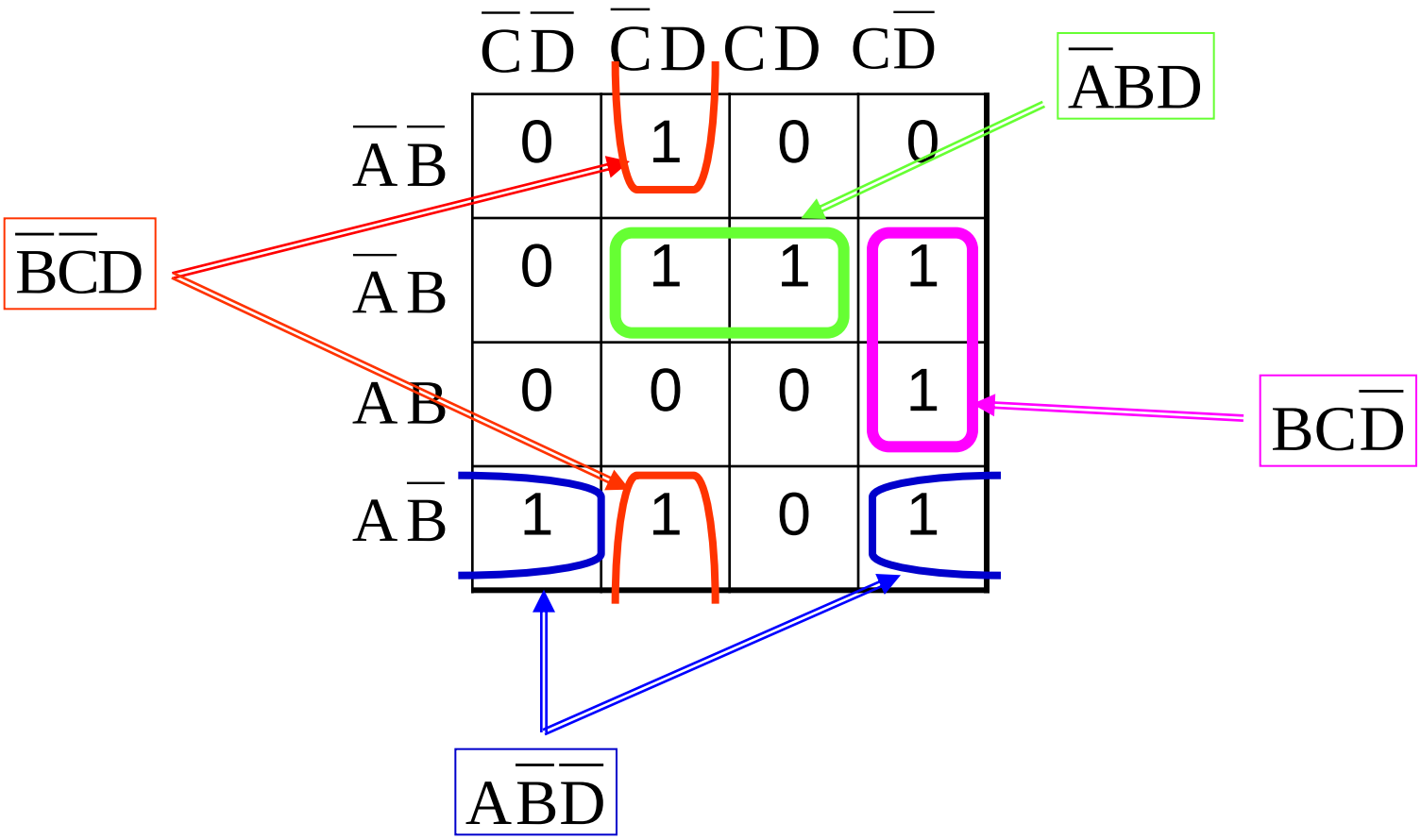
Option #1



$$X = \overline{A}\overline{C}D + \overline{A}BC + AC\overline{D} + A\overline{B}\overline{C}$$

Example#

Option # 2



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

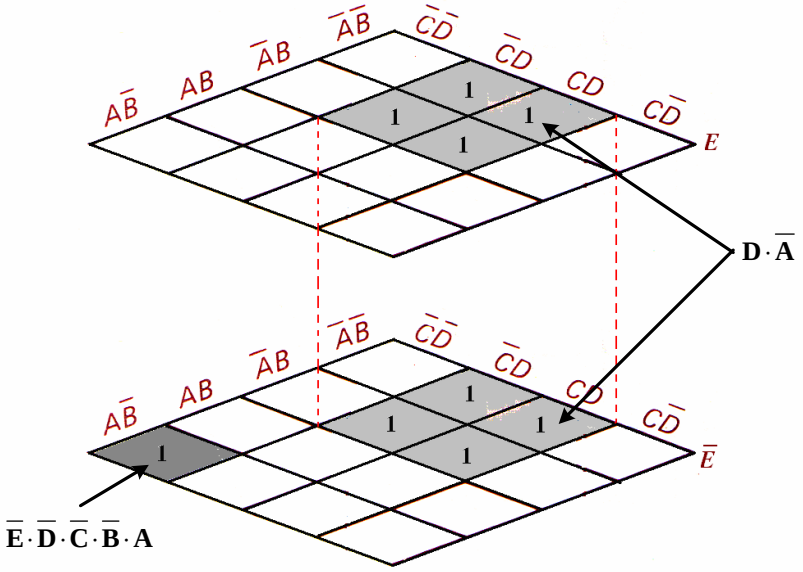
A Five Variable Karnaugh Map

- The Karnaugh map becomes three-dimensional when solving logic problems with more than four variables. A three-dimensional Karnaugh map will be used in this section.
- Consider the truth table shown in Figure below (a). The truth table must have 32 (2^5) rows to show all the combinations for five variables.
- The unsimplified Boolean expression for the truth table can be written as

INPUTS					OUTPUT	INPUTS					OUTPUT
E	D	C	B	A	Y	E	D	C	B	A	Y
0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	1	1
0	0	0	1	0	0	1	0	0	1	0	0
0	0	0	1	1	0	1	0	0	1	1	0
0	0	1	0	0	0	1	0	1	0	0	1
0	0	1	0	1	1	1	0	1	0	1	0
0	0	1	1	0	0	1	0	1	1	0	0
0	0	1	1	1	0	1	0	1	1	1	0
0	1	0	0	0	1	1	1	0	0	0	0
0	1	0	0	1	0	1	1	0	0	1	0
0	1	0	1	0	0	1	1	0	1	0	0
0	1	0	1	1	0	1	1	0	1	1	0
0	1	1	0	0	1	1	1	1	0	0	1
0	1	1	0	1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	1	1	1	0	0
0	1	1	1	1	0	1	1	1	1	1	0

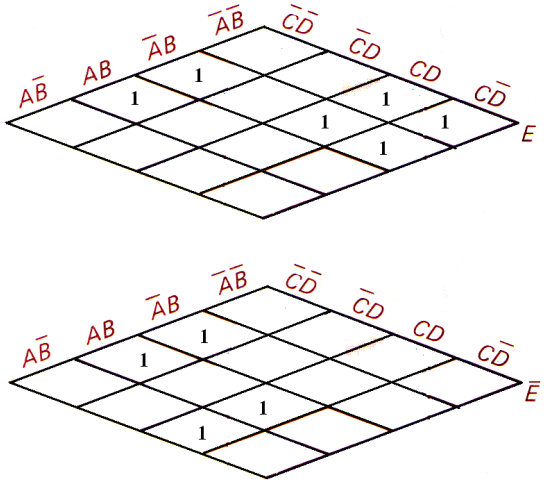
$$Y = \overline{E}\cdot\overline{D}\cdot\overline{C}\cdot\overline{B}\cdot A + \overline{E}\cdot D\cdot\overline{C}\cdot\overline{B}\cdot\overline{A} + \overline{E}\cdot D\cdot\overline{C}\cdot B\cdot\overline{A} + \overline{E}\cdot D\cdot C\cdot\overline{B}\cdot\overline{A} + \overline{E}\cdot D\cdot C\cdot B\cdot\overline{A} + E\cdot D\cdot\overline{C}\cdot\overline{B}\cdot\overline{A} + E\cdot D\cdot\overline{C}\cdot B\cdot\overline{A} + E\cdot D\cdot C\cdot\overline{B}\cdot\overline{A} + E\cdot D\cdot C\cdot B\cdot\overline{A}$$

INPUTS					OUTPUT	INPUTS					OUTPUT
E	D	C	B	A	Y	E	D	C	B	A	Y
0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	1	1
0	0	0	1	0	0	1	0	0	1	0	0
0	0	0	1	1	0	1	0	0	1	1	0
0	0	1	0	0	0	1	0	1	0	0	1
0	0	1	0	1	1	1	0	1	0	1	0
0	0	1	1	0	0	1	0	1	1	0	0
0	0	1	1	1	0	1	0	1	1	1	0
0	1	0	0	0	1	1	1	0	0	0	0
0	1	0	0	1	0	1	1	0	0	1	0
0	1	0	1	0	0	1	1	0	1	0	0
0	1	0	1	1	0	1	1	0	1	1	0
0	1	1	0	0	1	1	1	1	0	0	1
0	1	1	0	1	0	1	1	1	0	1	1
0	1	1	1	0	0	1	1	1	1	0	0
0	1	1	1	1	0	1	1	1	1	1	0

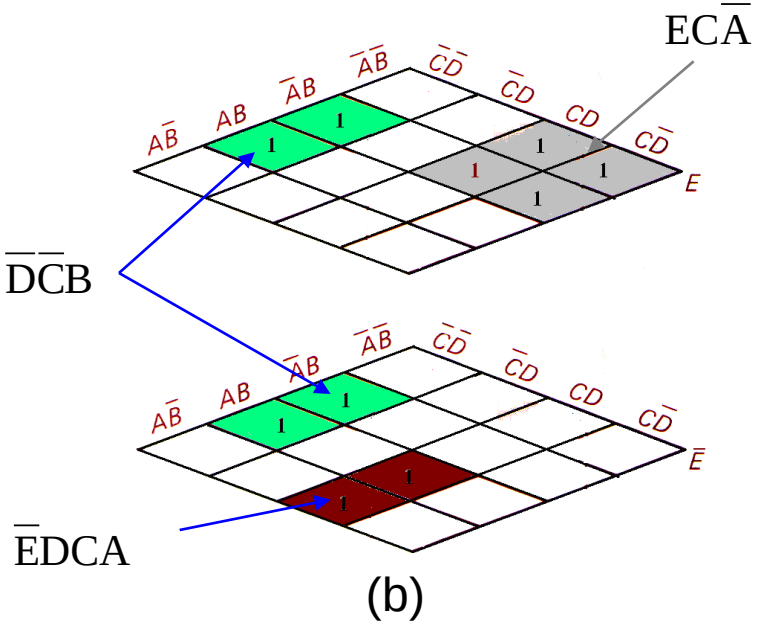


Example

- Simplify the following 5-input variables k-map shown in Figure (a).
- The simplified expression taken from the map of Figure (a) is developed and shown in Figure (b).
- Combining these AND terms using OR gate yields



(a)



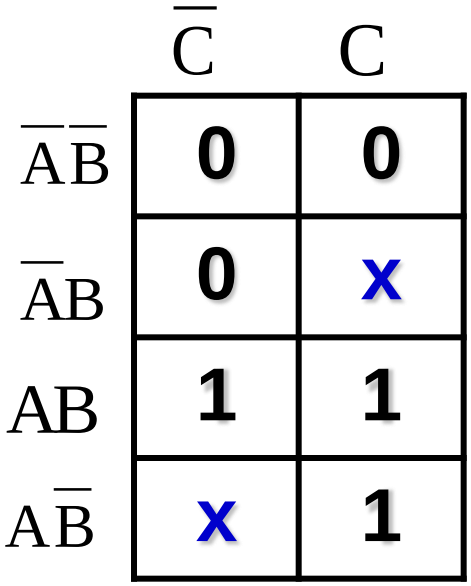
(b)

$$Y = \overline{D} \cdot \overline{C} \cdot B + E \cdot C \cdot \overline{A} + \overline{E} \cdot D \cdot C \cdot A$$

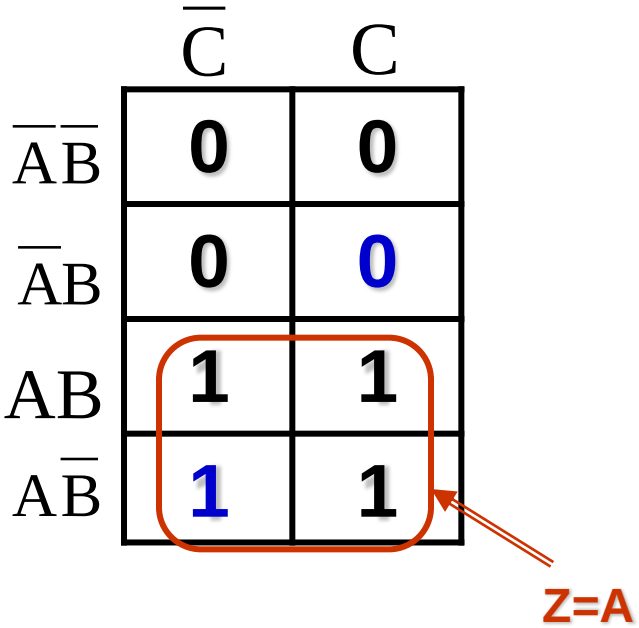
“Don’t Care” Conditions

- Some logic circuits can be designed so that there are certain input conditions for which there are no specified output levels, usually because these input conditions will never occur.
- In other words, there will be certain combinations of input levels where we ‘’don’t care’’ whether the output is HIGH or LOW.
- This is illustrated in the truth table of figure (a).

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



(b)



(c)

- Whenever "don't care" conditions occur, we have to decide which ones to change to 0 and which to 1 to produce the best K-map looping (i.e., the simplest expression).
- This decision is not always an easy one.

Example on “Don’t Care”

AB \ CD	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	0	0	0	0
$\overline{A}B$	1	0	0	X
AB	1	1	1	1
$A\overline{B}$	0	X	X	0

Redundant loop

$$Y = B\overline{D} + AD$$

~~$$Y = B\overline{D} + AD + AB$$~~

Summary

The K-map process has several advantages over the algebraic method.

- K mapping is a more orderly process with well-defined steps as compared with the trial- and error process sometimes used in algebraic simplification.
- K mapping usually requires fewer steps, especially for expressions containing many terms, and it always produces a minimum expression.
- Nevertheless, some instructors prefer the algebraic method because it requires a thorough knowledge of Boolean algebra and is not simply a mechanical procedure. Each method has its advantages, and though most logic designers are adept at both, being proficient in one method is all that is necessary to produce acceptable results.
- There are other more complex techniques that designers use to minimize logic circuits. These techniques are especially suited for circuits with large numbers of inputs where algebraic and k-mapping methods are not feasible.
- Most of these techniques can be translated into a computer program, which will perform the minimization from input data that supply the truth table or unsimplified expression.