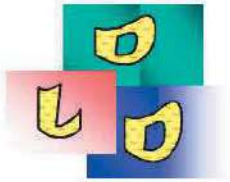
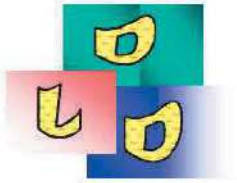


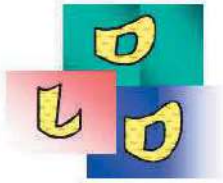
UNIT – II

GATE LEVEL MINIMIZATION



The Map Method

- The complexity of the digital logic gates that implement a Boolean function is directly related to the complexity of the algebraic expression from which the function is implemented.
- Although, the truth table representation of a function is unique, when expressed algebraically , it can appear in many different forms.
- The procedure of minimization of expressions by substituting equivalent rules of Boolean algebra is awkward because it lacks specific rules to predict each succeeding step in the manipulative process.
- The map method provides a simple straight forward procedure for minimizing Boolean functions. This method may be regarded as a pictorial form of a truth table.
- The Map method is also known as the Karnaugh map or K-map.

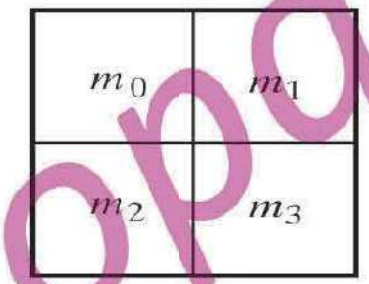


The Map Method - continued.,

- The map is a diagram made up of squares, with each square representing one minterm of the function. Since any Boolean function can be expressed as a sum of minterms, it follows that a Boolean function is recognized graphically in the map from the area enclosed by those squares whose minterms are included in the function.
- The map presents a visual diagram of all possible ways a function may be expressed in standard form.
- By recognizing various patterns, the user can derive alternative algebraic expressions for the same function, from which the simplest can be selected.
- The simplified expressions produced by the map are always in one of the two standard forms: sum of products or product of sums.

Two Variable Map

- The two variable map is used to represent a function with two variables.
- There are four minterms for two variables: hence the map consists of four squares, one for each minterm.

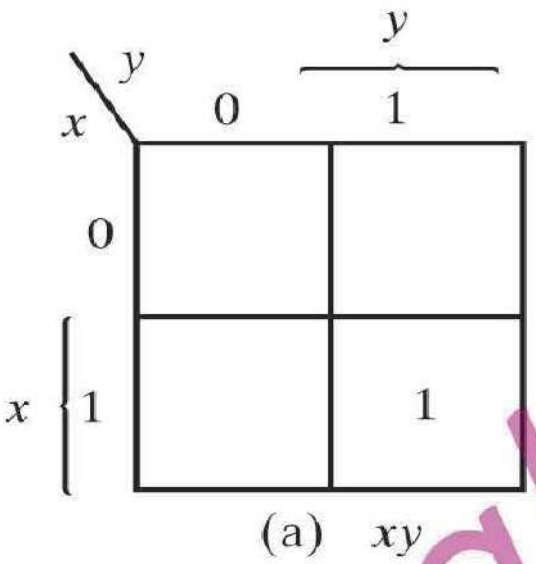


(a)

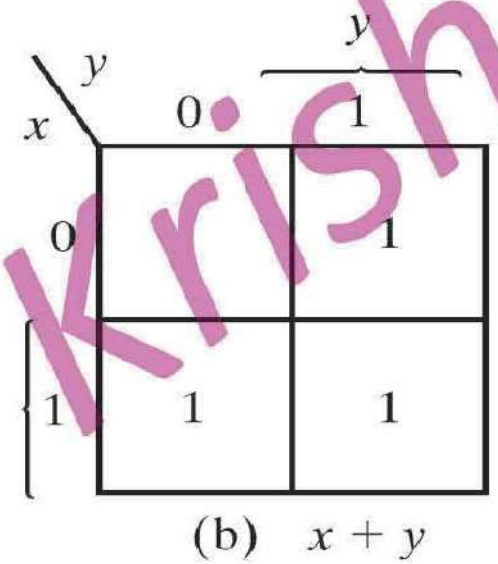
x \ y	0	1
0	$x'y'$	$x'y$
1	xy'	xy

(b)

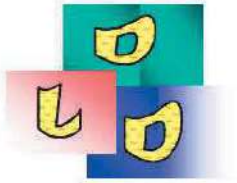
Representation of Functions in the Map



(a) xy



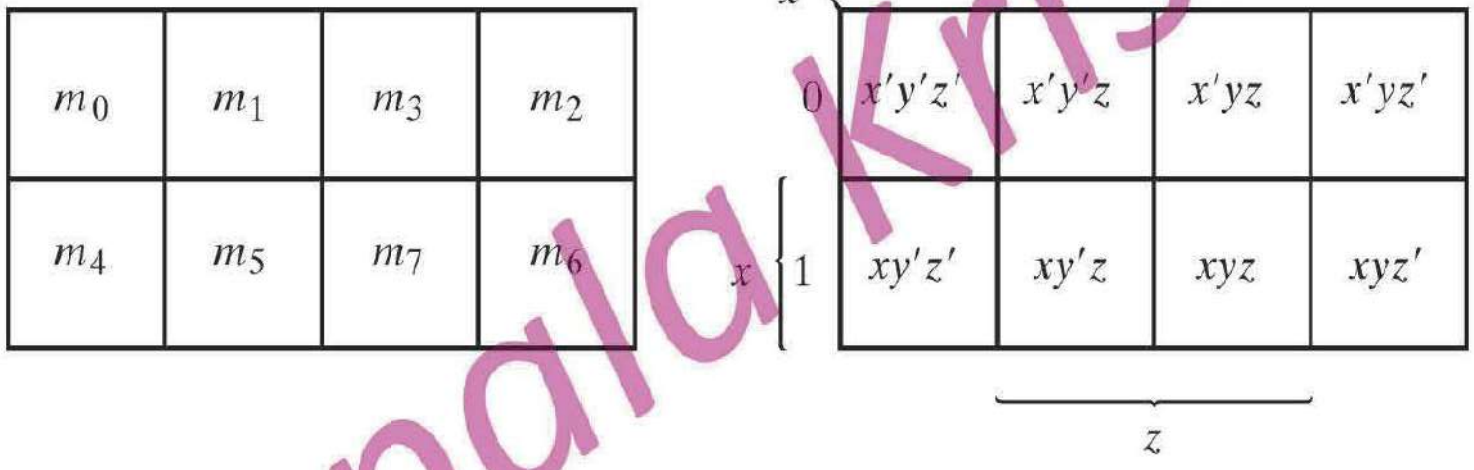
(b) $x + y$

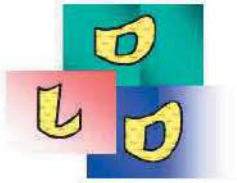


Three Variable Map

- The three variable map is used to represent a function with three variables.
- There are eight minterms for three binary variables. Therefore the map consists of eight squares.
- Here the minterms are not arranged in a binary sequence, but in a sequence similar to a Gray code. The characteristic of this sequence is that only bit changes in value from one adjacent column to the next.

Three Variable Map



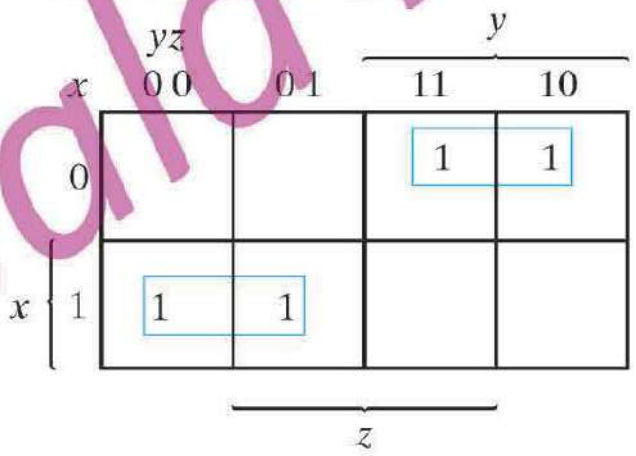


Minimization Procedure

- The combination of adjacent squares that is useful during the simplification process is easily determined from inspection of the three variable map:
- One square represents one minterm, giving a term of three literals.
- Two adjacent squares represent a term of two literals.
- Four adjacent squares represent a term of one literal.
- Eight adjacent squares represent the function equal to 1.

Example 1

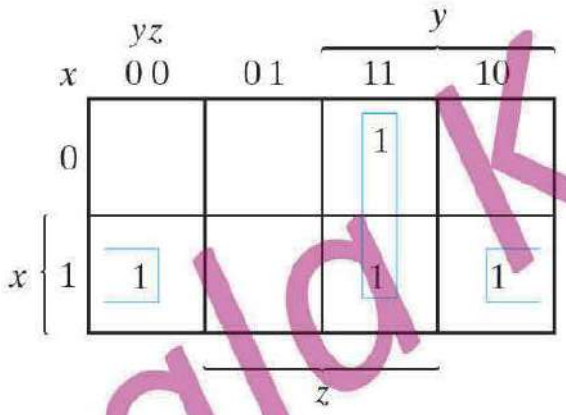
- Simplify the Boolean function $F(x,y,z)=\Sigma(2,3,4,5)$
- First, a 1 is marked in each minterm that represents the function.
- The next step is to find possible adjacent squares.



$F(x,y,z)= \Sigma(2,3,4,5)=x^1y+xy^1$

Example 2

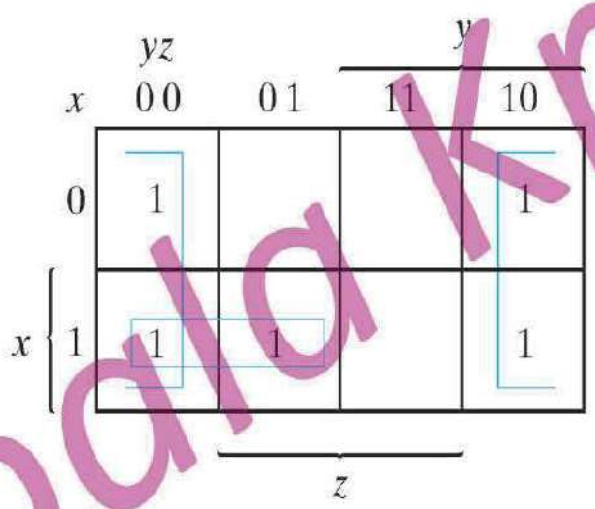
- Simplify the Boolean function $F(x,y,z)=\Sigma(3,4,6,7)$



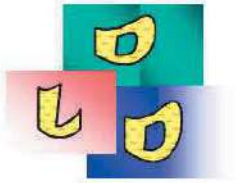
$F(x,y,z)= \Sigma(3,4,6,7)=yz+xz^1$

Example 3

- Simplify the Boolean function $F(x,y,z)=\Sigma(0,2,4,5,6)$



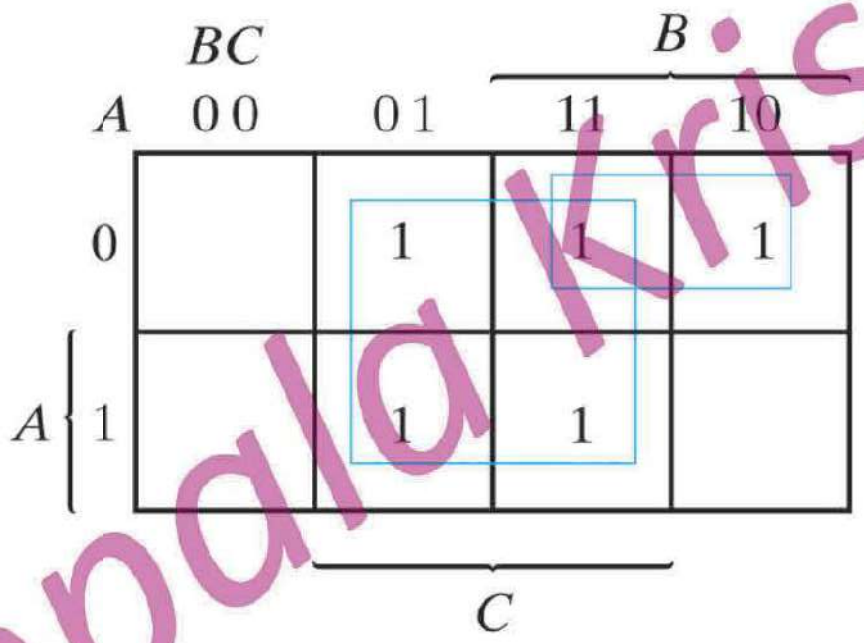
$$F(x,y,z)=\Sigma(0,2,4,5,6)=z^1+xy^1$$



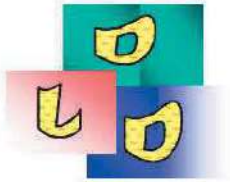
Example 4

- Given the Boolean function $F=A^1C+A^1B+AB^1C+BC$
  - Express it in terms of minterms
  - And find the minimal sum of products expression
- Three product terms in the expression have two literals and are represented in a three variable map by two squares each.
  - The two squares corresponding to the term A^1C are :001,011
  - The two squares corresponding to the term A^1B are :010,011
  - The square corresponding to the term AB^1C are :101
  - The two squares corresponding to the term BC are :011,111
- The function can be expressed in sum of minterms form:

$$F(A,B,C)=\Sigma(1,2,3,5,7)$$



$$F=A^1C+A^1B+AB^1C+BC=C+A^1B$$



Four Variable Map

- The four variable map is used to represent a function with four variables.
- There are sixteen minterms for four binary variables. Therefore the map consists of sixteen squares.
- The rows and columns are numbered in a Graycode sequence, with only one digit changing value between two adjacent rows and columns.
- The map minimization of four variables Boolean functions is similar to the method used to minimize three variable functions. Adjacent squares are defined to be squares next to each other. In addition, the map is considered to lie on a surface with the top and bottom edges, as well as the right and left edges, touching each other to form adjacent squares.

Four Variable Map

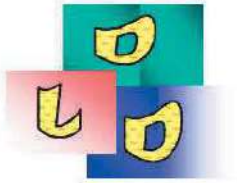
m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

(a)

wx \ yz	00	01	11	10
00	$w'x'y'z'$	$w'x'y'z$	$w'x'yz$	$w'x'yz'$
01	$w'xy'z'$	$w'xy'z$	$w'xyz$	$w'xyz'$
11	$wxy'z'$	$wxy'z$	$wxyz$	$wxyz'$
10	$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$

y, x, w, z

(b)

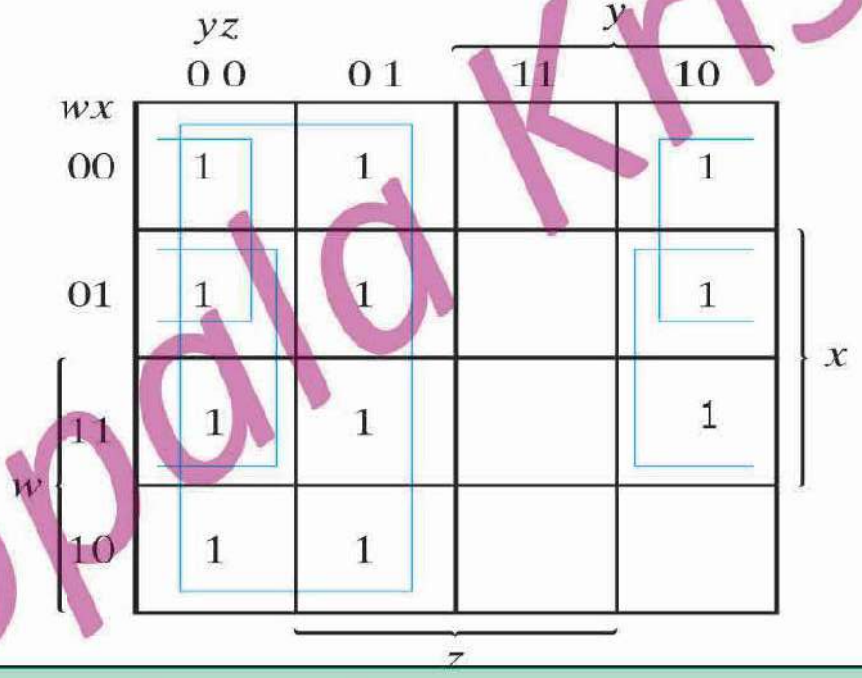


Minimization Procedure

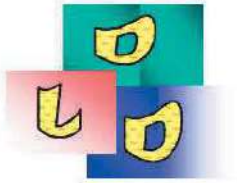
- The combination of adjacent squares that is useful during the simplification process is easily determined from inspection of the four variable map:
- One square represents one minterm, giving a term of four literals.
- Two adjacent squares represent a term of three literals.
- Four adjacent squares represent a term of two literals.
- Eight adjacent squares represent a term of one literal.
- Sixteen adjacent squares represent the function equal to 1.

Example 5

- Simplify the Boolean function F(w,x,y,z)=Σ(0,1,2,4,5,6,8,9,12,13,14)
-

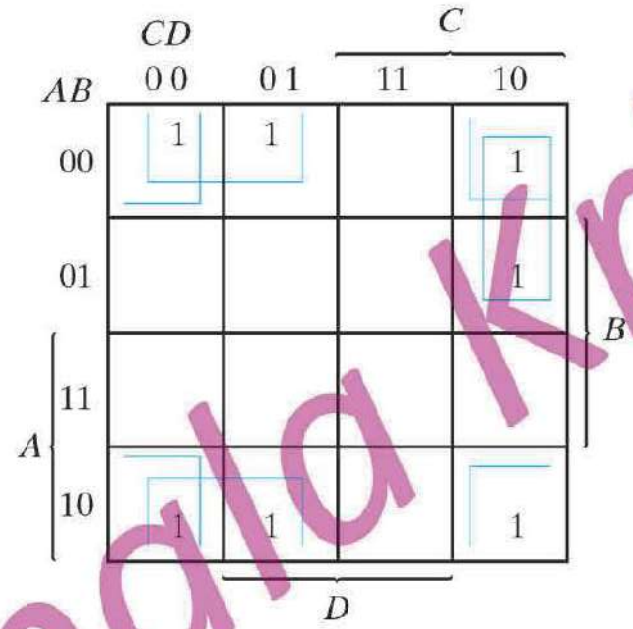


F(w,x,y,z)=Σ(0,1,2,4,5,6,8,9,12,13,14)=$y^1+w^1z^1+xz^1$

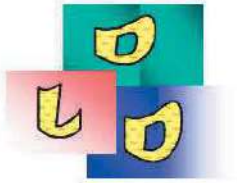


Example 6

- Simplify the Boolean function $F=A^1B^1C^1+B^1CD^1+A^1BCD^1+AB^1C^1$
- The function has four variables and consists of three terms with each three literals, and one term of four literals.
- Each term of three literals is represented in the map by two squares.
  - The two squares corresponding to the term $A^1B^1C^1$ are :0001,0000
  - The two squares corresponding to the term B^1CD^1 are :0010,1010
  - The two square corresponding to the term AB^1C^1 are :1000,1001
  - The one square corresponding to the term A^1BCD^1 are :0110



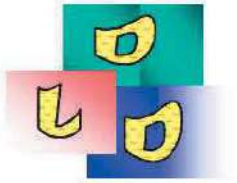
$$F=A^1B^1C^1+B^1CD^1+A^1BCD^1+AB^1C^1=B^1D^1+B^1C^1+A^1CD^1$$



Prime Implicants

- A prime implicant is a product term obtained by combining the maximum number of possible adjacent squares in the map.
- If a minterm in a square is covered by only on prime implicant, that prime implicant is called Essential prime implicant.

- Pair: pair is a group of two logical 1's in a Karnaugh map. A pair eliminates one variable in the output expression.
- Quad: Quad is a group of four logical 1's in a Karnaugh map. A quad eliminates two variable in the output expression.
- Octet: Octet is a group of eight logical 1's in a Karnaugh map. An octet eliminates three variable in the output expression.



Five Variable Map

- Maps for more than four variables are not as simple to use. A five variable map needs 32 squares and a six variable map needs 64 squares. When the number of variables becomes large, the number of squares becomes excessively large and the geometry of combining adjacent squares becomes more complicated.

Five Variable Map

- The five variable map can be shown as two four variable maps.

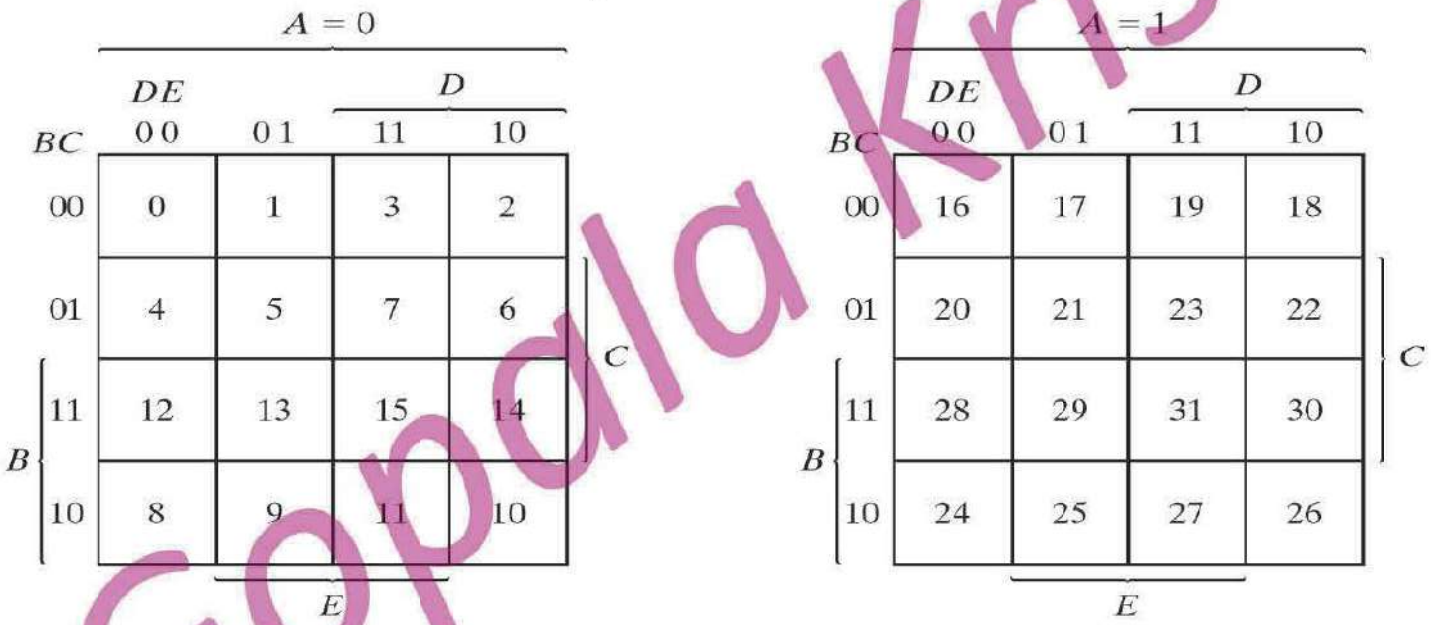
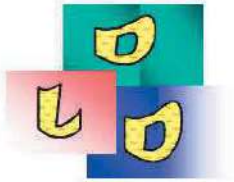


Fig. 3-12 Five-variable Map

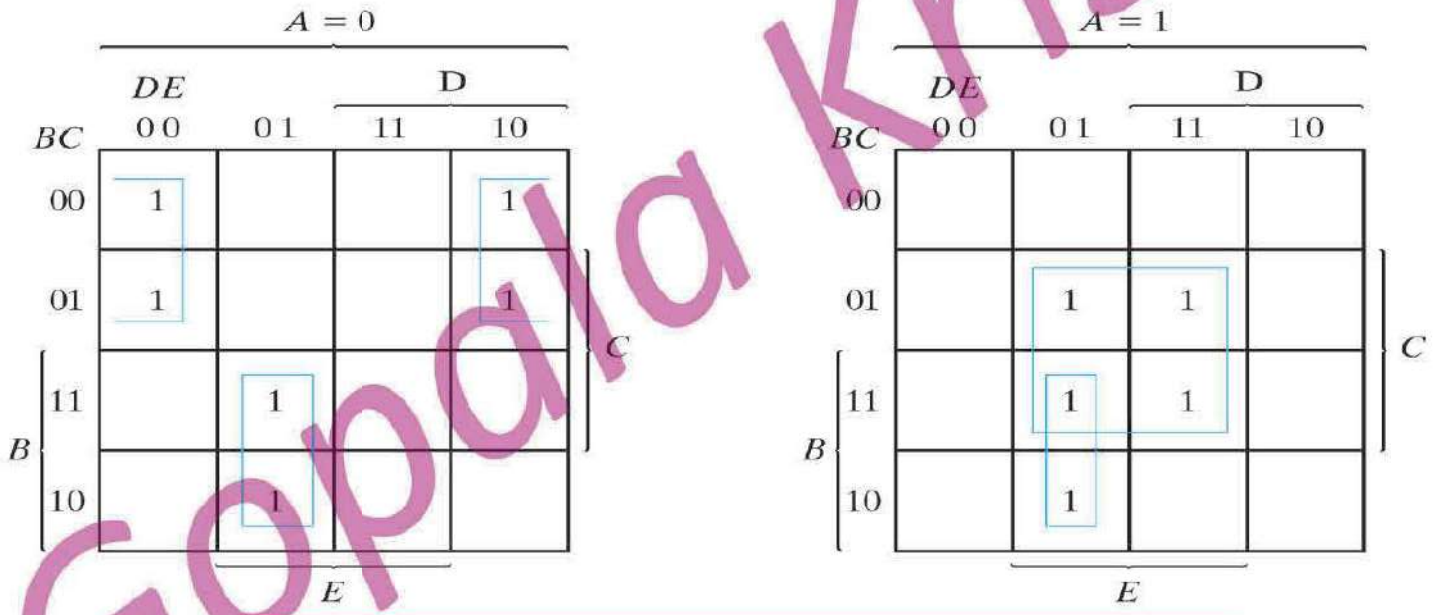


- It consists of two four variable maps with variables A,B,C,D and E. Variable A distinguishes between the two maps as indicated on the top of the diagram.
- Minterms 0 to 15 belong with A=0 and minterms 16 to 31 belong with A=1.
- Each four variable map retains its adjacency when taken separately.
- In addition each square in the A=0 map is adjacent to the corresponding square in A=1 map. For example, minterm 4 is adjacent to minterm 20 and minterm 15 to 31.
- The best way to visualize this new rule for adjacent squares is to consider the two half maps as being one on top of the other. Any two squares that fall one over the other are considered adjacent.

Example 7

- Simplify the Boolean function

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



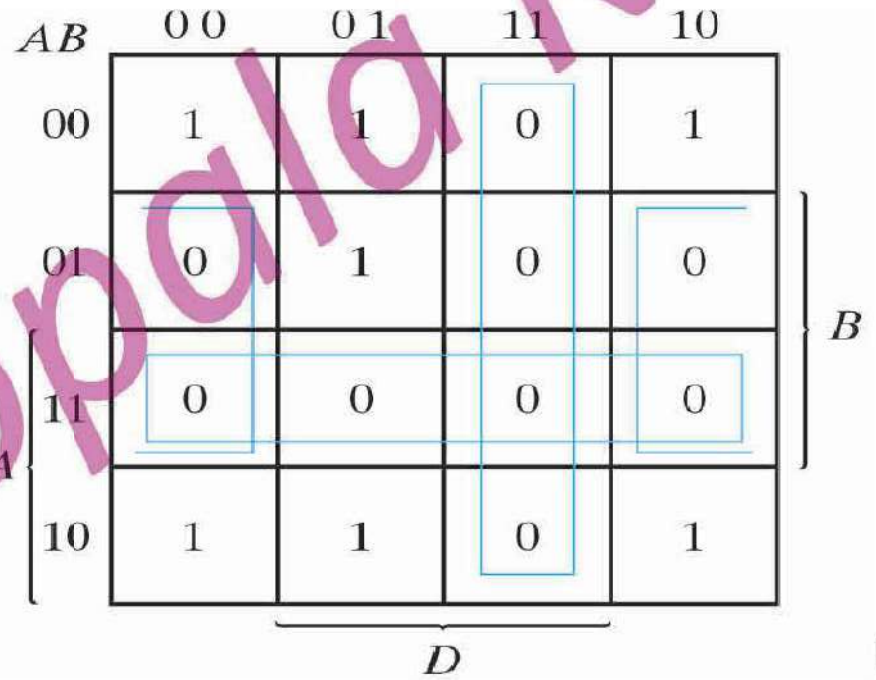
$$F=A^1B^1E^1+BD^1E+ACE$$

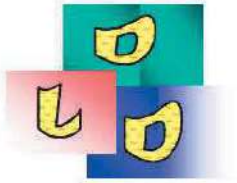
Product of Sums Simplification

- The procedure for obtaining a minimized function in product of sums follows from the basic properties of Boolean functions. The 1's placed in the squares of the map represent the minterms of the function.
- The minterms not included in the function denote the complement of the function.
- So, if we mark the empty squares by 0's and combine them into valid adjacent squares we obtain a simplified expression for F^1. the complement of F^1 gives us back the function F. because of the generalized DeMorgan's theorem, the function so obtained is automatically in the product of sums form.

Example 8

- Simplify the following Boolean function in Product of Sums form:
  $F(A,B,C,D)=\Sigma(0,1,2,5,8,9,10)$



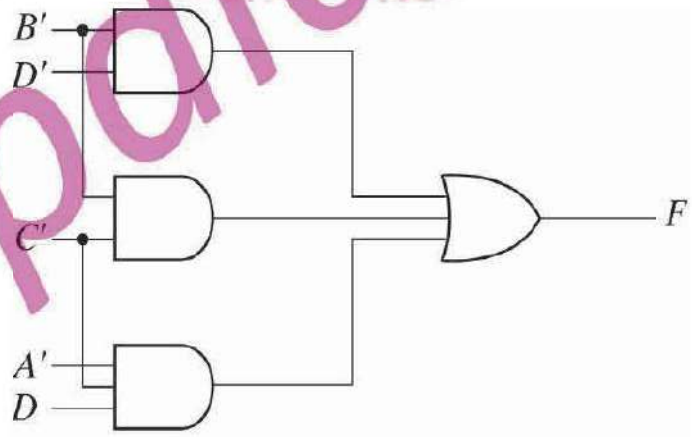


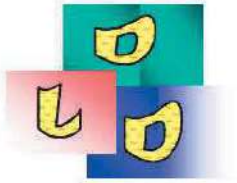
- If the squares marked with 0's are combined, we obtain the simplified complemented function: $F^1=AB+CD+BD^1$
- $(F^1)^1=(AB+CD+BD^1)^1$

  $=(A^1+B^1)(C^1+D^1)(B^1+D)$

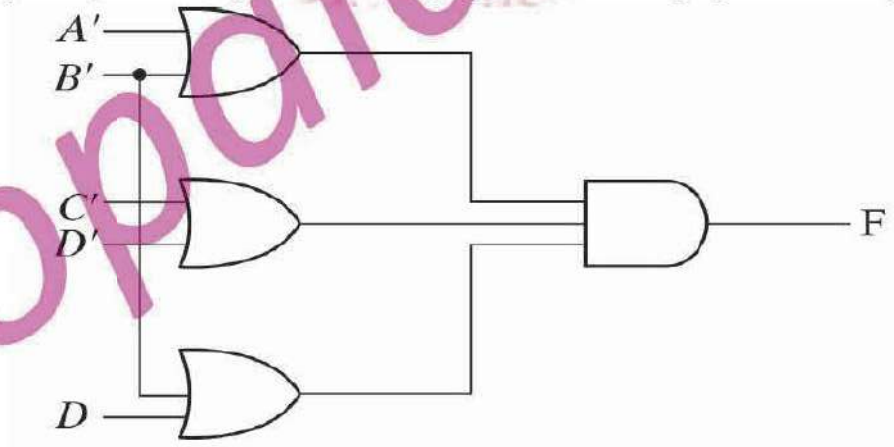
Two Level Implementation

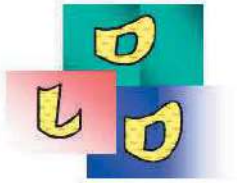
- The implementation of the simplified expression in sum of products form can be done as follows: with a group of AND gates, one for each AND term. The outputs of the AND Gates are connected to the inputs of a single OR Gate.
- For example, $F=B^1D^1+B^1C^1+A^1C^1D$





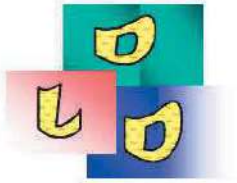
- The implementation of the simplified expression in products of Sums form can be done as follows: with a group of OR gates, one for each OR term. The outputs of the OR Gates are connected to the inputs of a single AND Gate.
- For Example, $F=(A^1+B^1)(C^1+D^1)(B^1+D)$





Don't Care Conditions

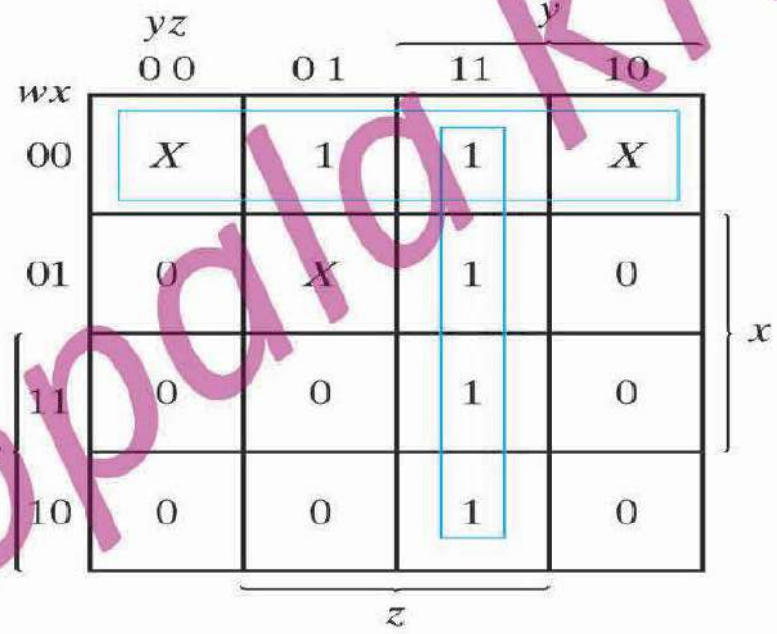
- Functions that have unspecified outputs for some input combinations are called incompletely specified functions.
- In most applications, we simply don't care what value is assumed by the function for the unspecified minterms. So it is customary to call the unspecified minterms of a function don't care conditions.
- These don't care conditions can be used on a map to provide further simplification of the Boolean expression.



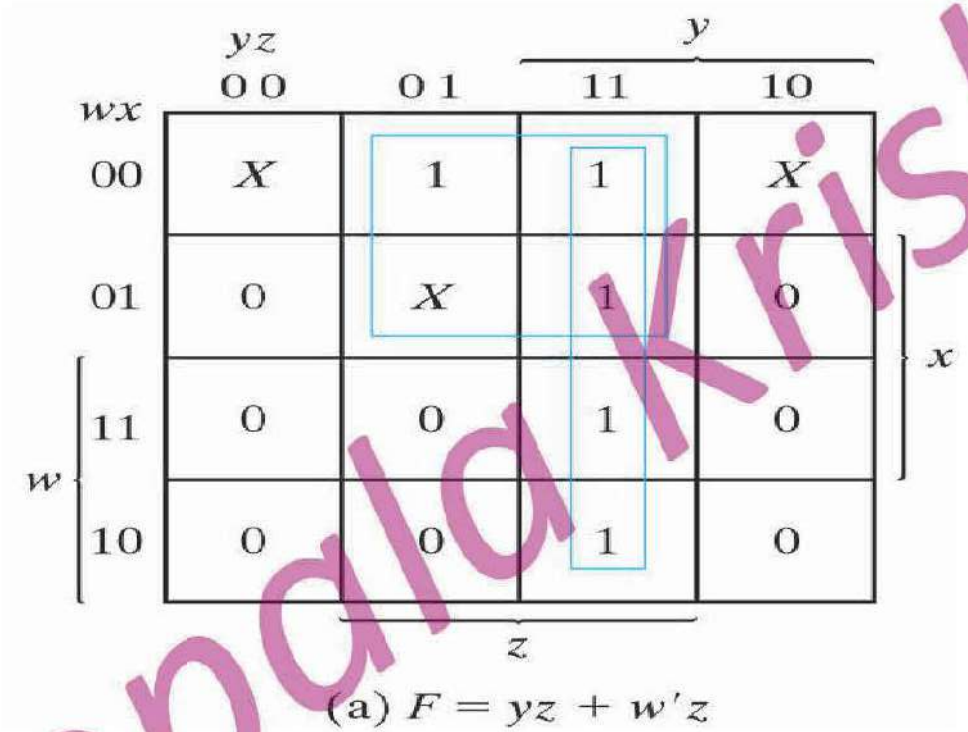
- A don't care minterm is a combination of variable whose logical value is not specified.
- To distinguish the don't care condition from 1's and 0's, an X used.
- Thus an X inside a square in the map indicates that we don't care whether the value of 0 or 1 is assigned to function for the particular minterm.

Example 9

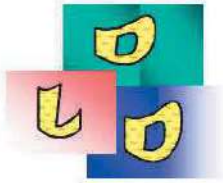
- Simplify the Boolean function F(w,x,y,z)=Σ(1,3,7,11,15) which has the don't care conditons d(w,x,y,z)= Σ(0,2,5)
-



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

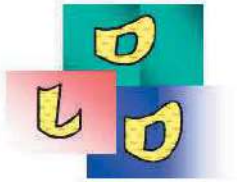


- Either one of the preceding two expressions satisfies the conditions stated for the example



NAND & NOR Implementation

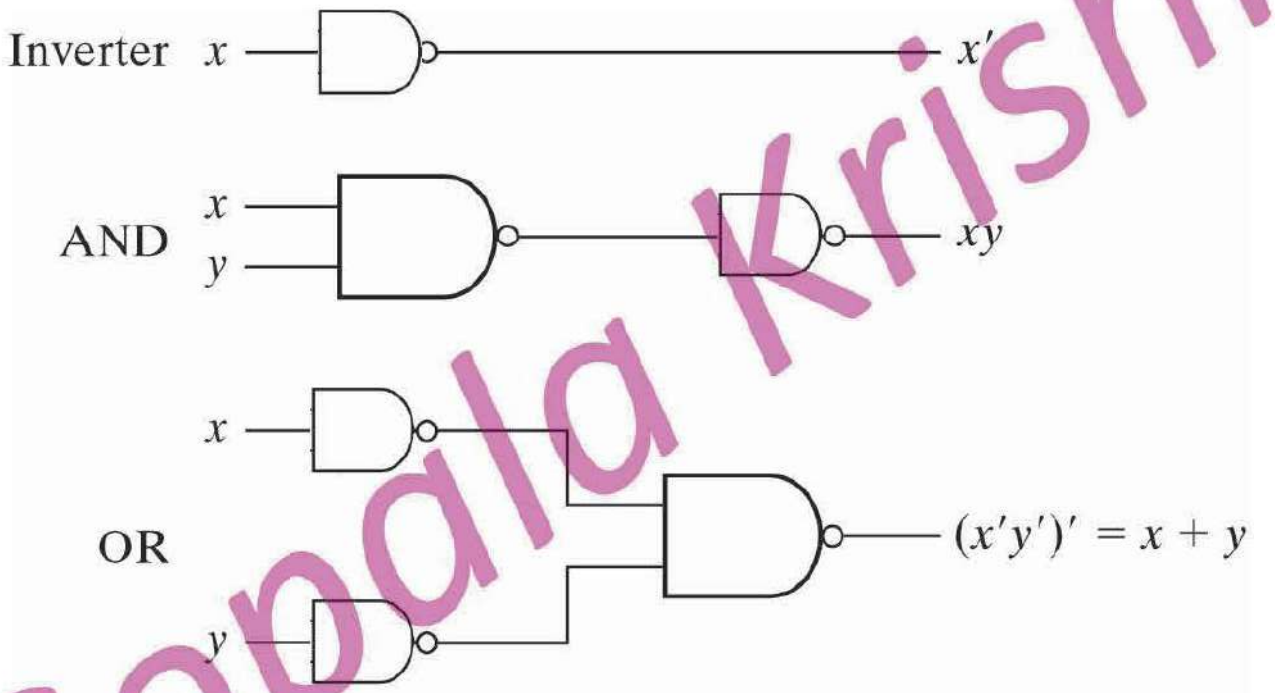
- Digital circuits are frequently constructed with NAND or NOR gates rather than with AND and OR gates.
- NAND and NOR gates are easy to fabricate with electronic components and are the basic gates used in all IC digital logic families.



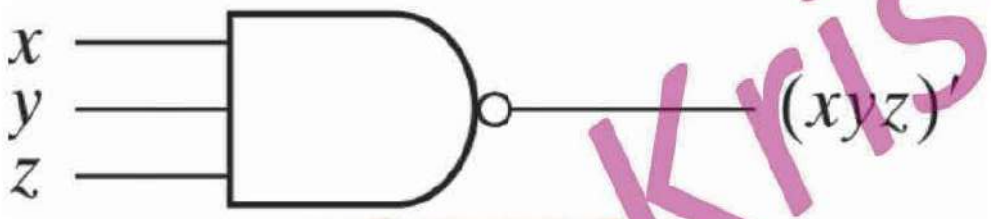
NAND Circuits

- The NAND gate is said to be universal gate because any digital system can be implemented with it.
- To show that any Boolean function can be implemented with NAND gates, we need only show that the logical operations of AND, OR and complement can be obtained with NAND gates only.

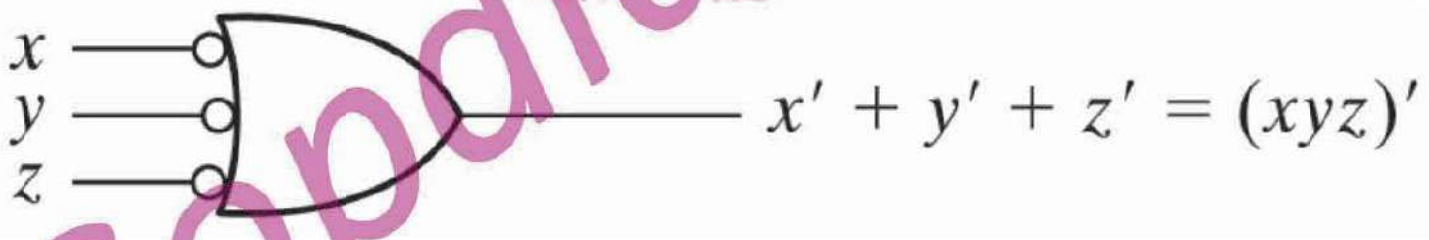
NAND Implementation



Graphic symbols for NAND Gates



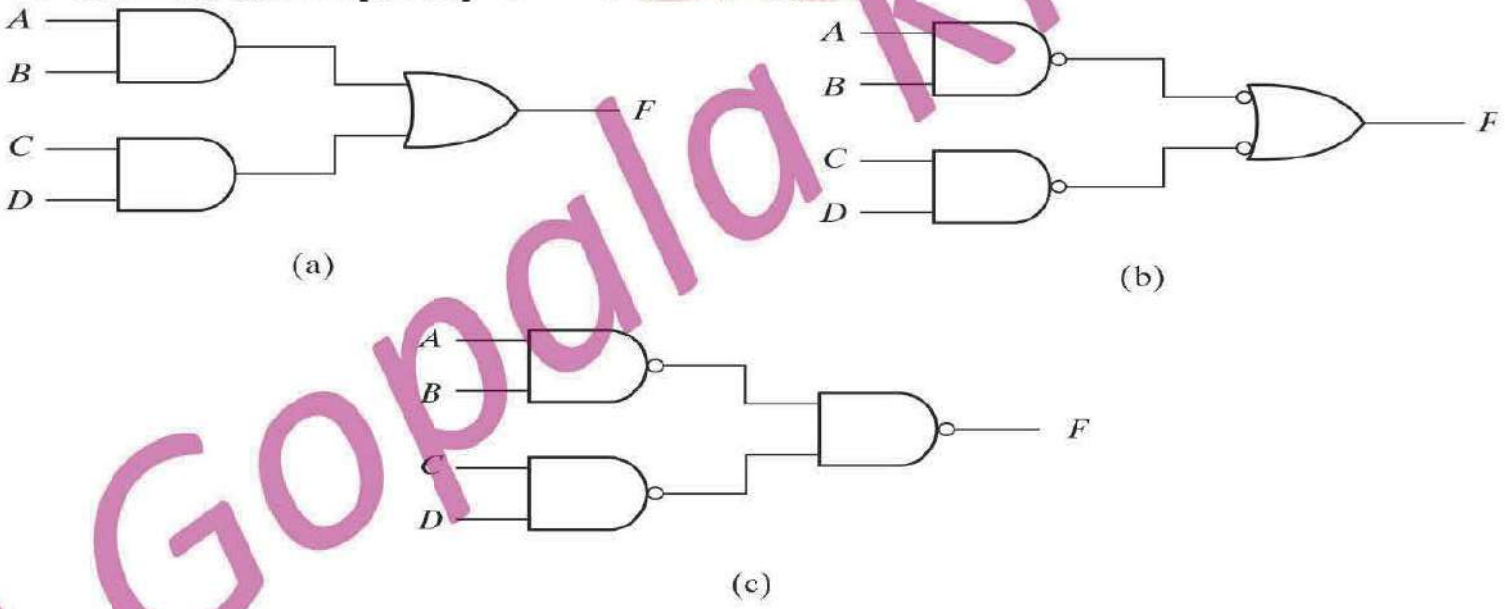
AND-INVERT

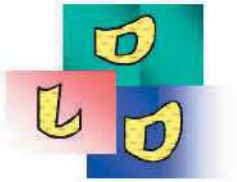


INVERT-OR

Two Level Implementation

- The implementation of Boolean functions with NAND gates requires that the function be in sum of products form.
- For example, F=AB+CD

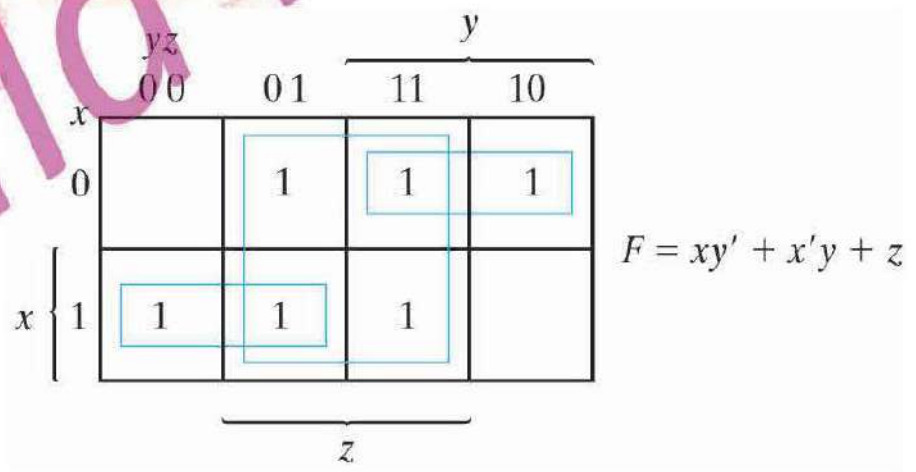


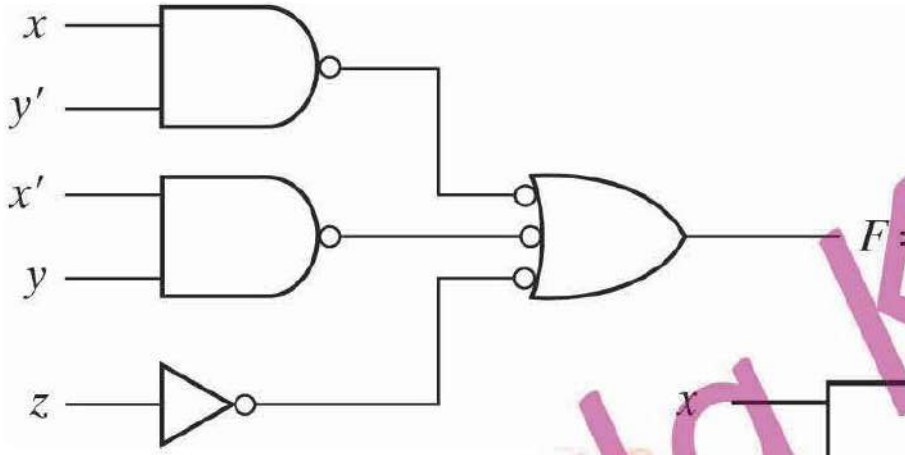


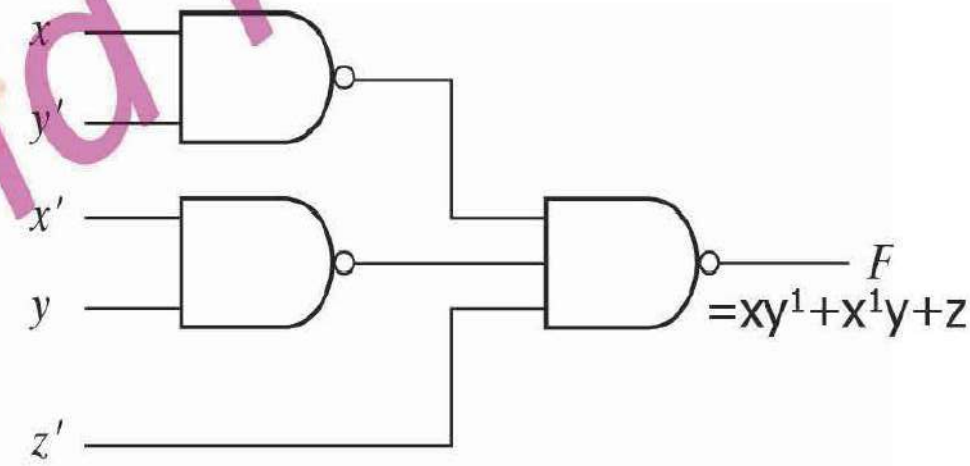
- The procedure for obtaining the logic diagram of Boolean function is as follows:
  - Simplify the function and express it in sum of products.
  - Draw a NAND gate for each product term of the expression that has atleast two literals. This constitutes the first level gates.
  - Draw single gate using the AND Invert or Invert OR graphic symbol in the second level, with inputs coming from outputs of first level gates.
  - A term with single literal requires an inverter in the first level. However, if the single literal is complemented, it can be connected directly to an input of the second level NAND Gate.

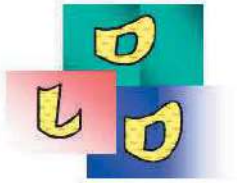
Example 10

- Implement the following Boolean function with NAND Gates: F(x,y,z)=(1,2,3,4,5,7)
- The first step is to simplify the function in sum of products. This is done by means of the map method.







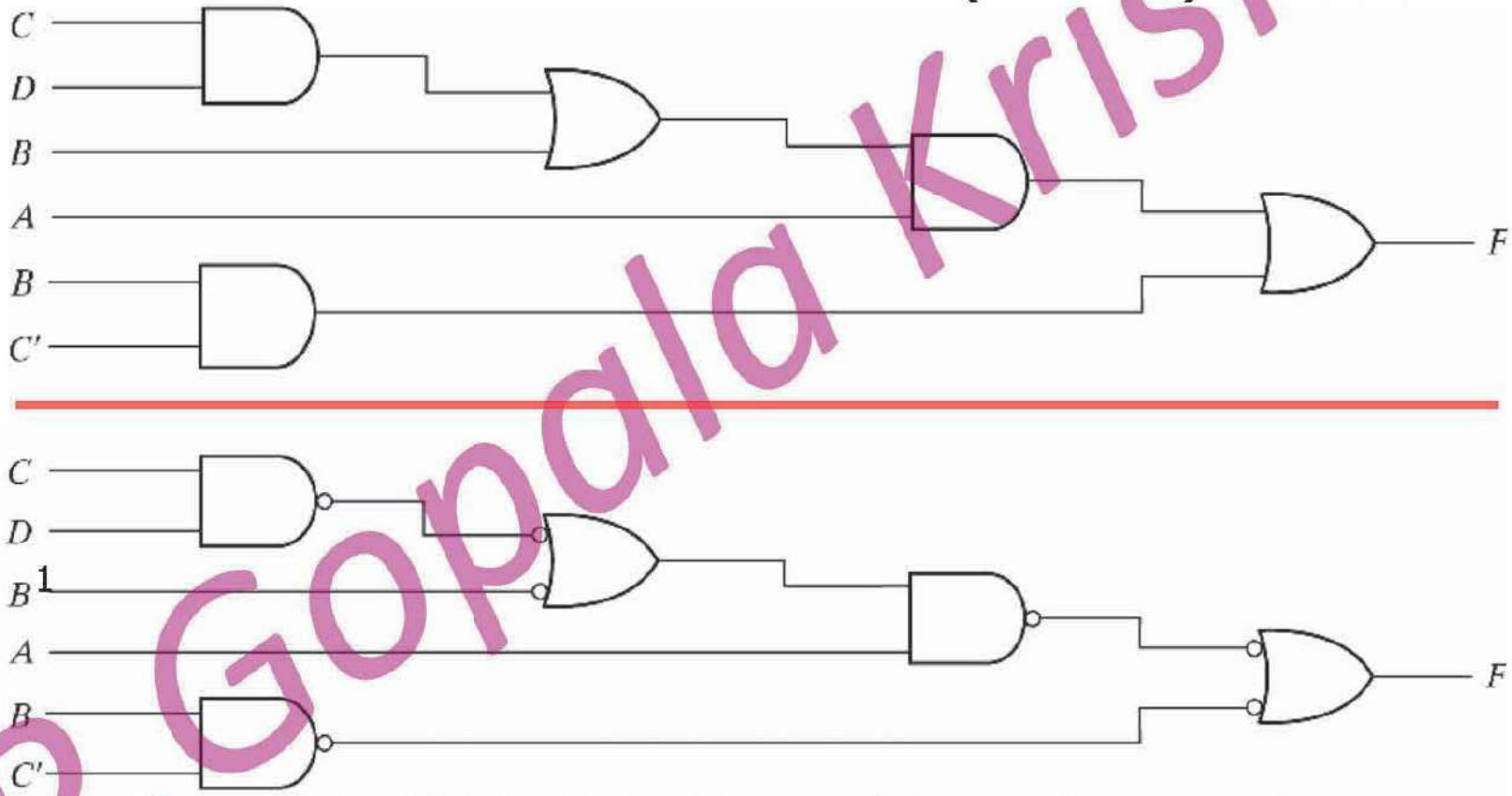


Multilevel NAND Circuits

- The general procedure for converting a multilevel AND-OR diagram into an all NAND diagram using mixed notation is as follows:
  - Convert all AND gates to NAND gates with AND-invert graphic symbol.
  - Convert all OR gates to NAND gates with invert-OR graphic symbols.
  - Check all the bubbles in the diagram. For every bubble that is compensated by another small circle along the same line, insert an inverter or complement the input literal.

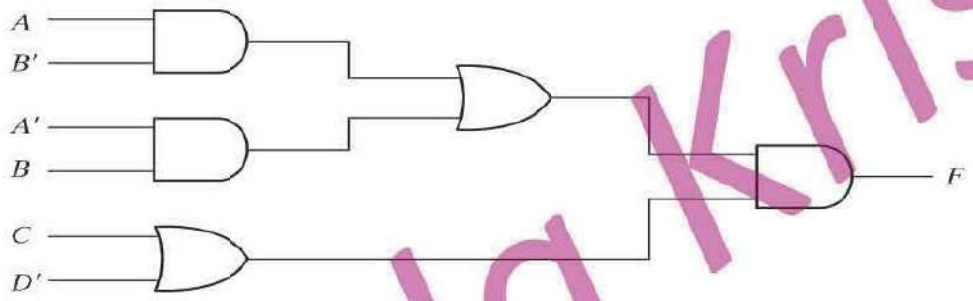
Example 11

- Implement the following function in multilevel NAND form $F=A(CD+B)+BC^1$

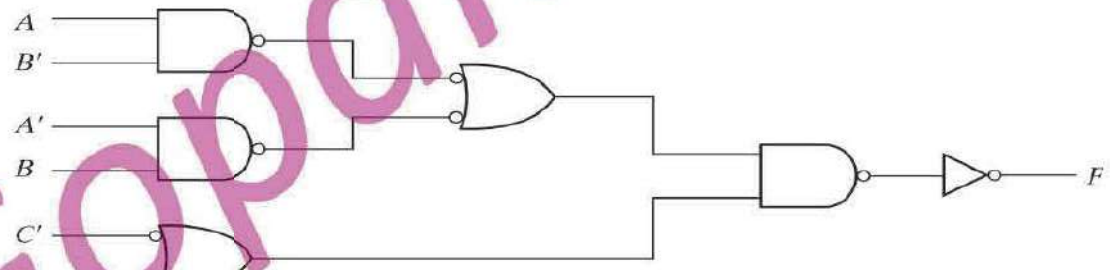


Example 12

As another example, consider the multilevel Boolean function $F=(AB^1+A^1B)(C+D^1)$



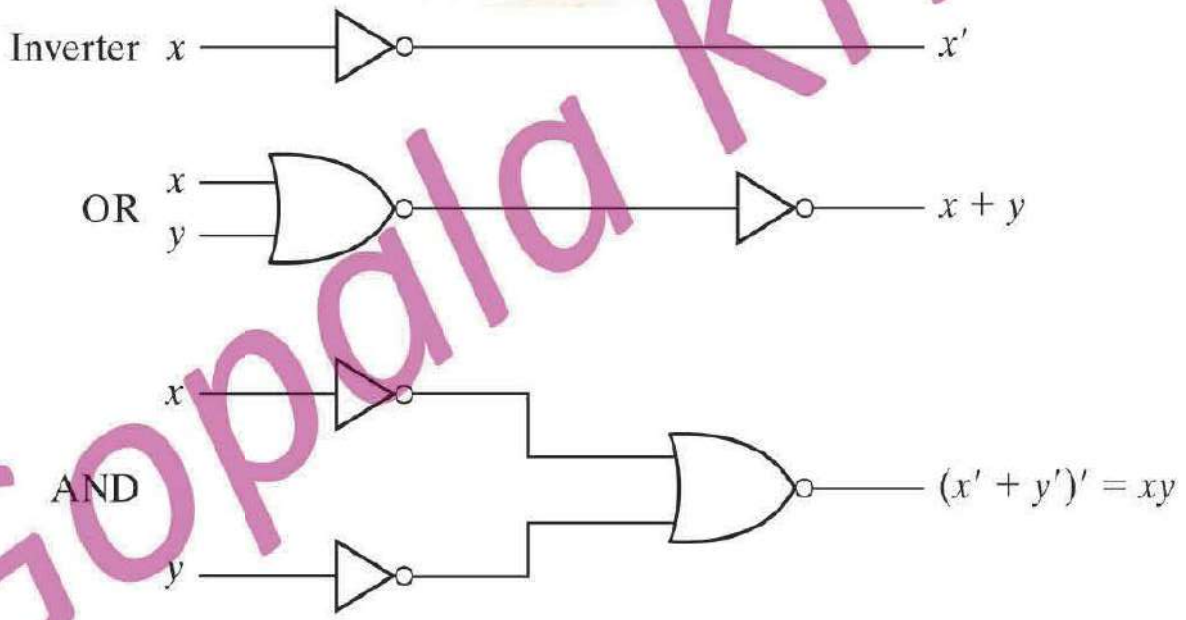
(a) AND-OR gates



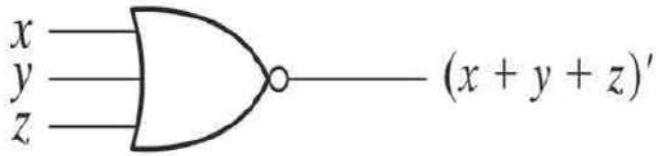
(b) NAND gates

NOR Implementation

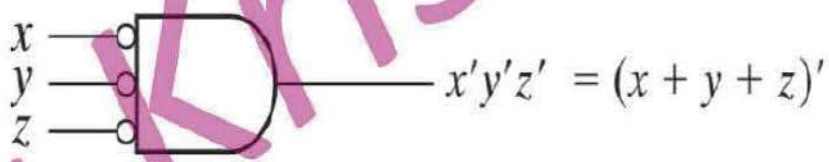
- The NOR Gate is another universal logic gate that can be used to implement any Boolean functions.



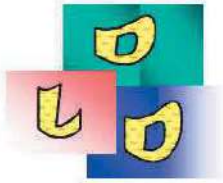
Two Graphic Symbols of NOR Gate



OR–invert



Invert–AND

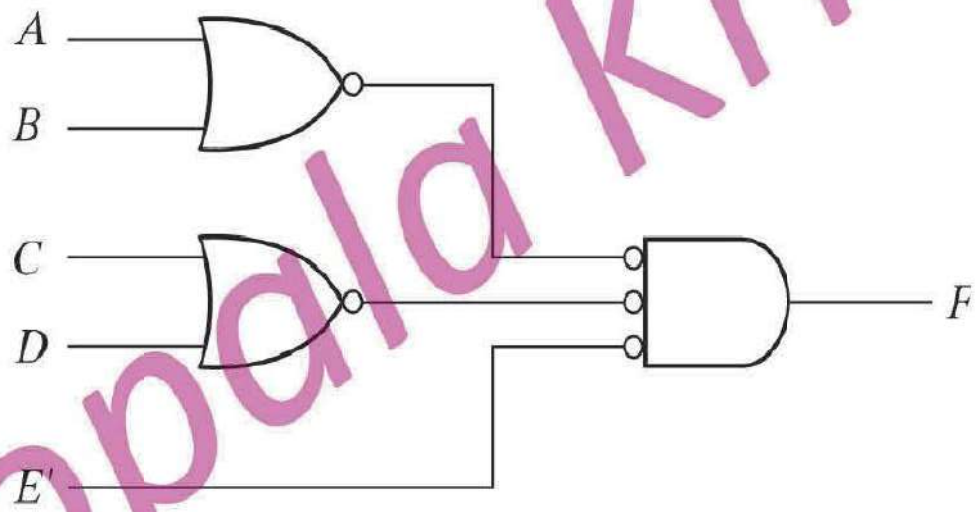


Two level NOR implementation

- A two level implementation with NOR gates requires that the function be simplified in product of sums.
- Remember that the simplified product of sums expression is obtained from the map by combining the 0's and complementing.
- A product of sums expression is implemented with a first level of OR gates that produce the sum terms followed by a second level AND gate to produce the product.
- The transformation from the OR-AND diagram to a NOR diagram is achieved by changing the OR gates to NOR gates with OR- invert graphic symbols and the AND gate with an invert-AND graphic symbol.
- A single literal term going into the second level gate must be complemented.

Example 13

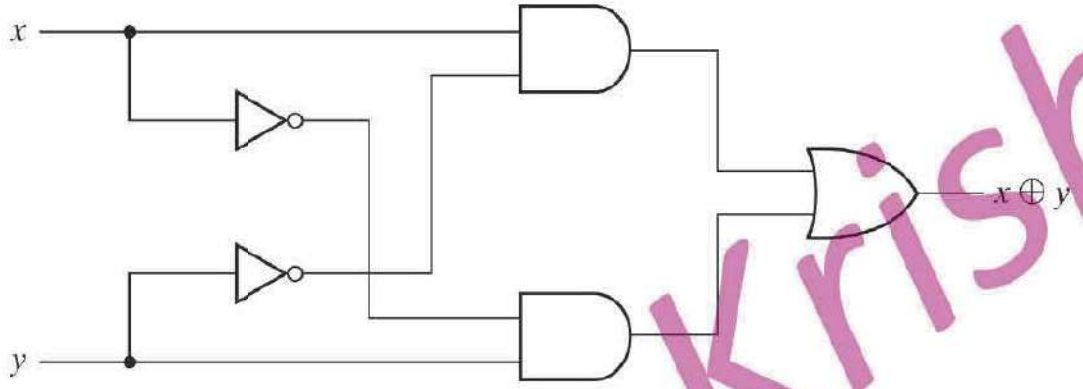
- Show the two level implementation of the Boolean Function F=(A+B)(C+D)E



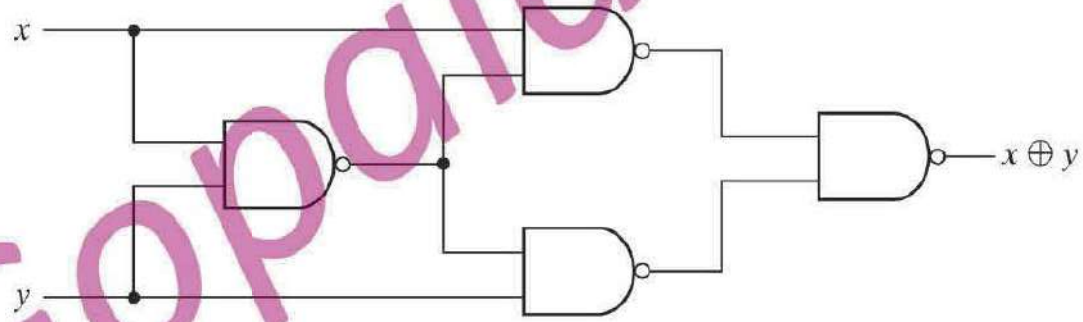
Exclusive – OR Function

- The exclusive – OR function (XOR), denoted by the symbol $\oplus$, is a logical operation that performs the following Boolean operation: $x \oplus y = xy^1 + x^1y$
- It is equal to 1 if only x is equal to 1 or if only y is equal to 1, but not when both are equal to 1.
- The exclusive NOR also known as equivalence, performs the following boolean Operation: $(x \oplus y)^1 = xy + x^1y^1$

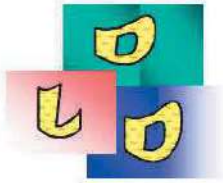
Logic Diagrams for XOR



(a) With AND-OR-NOT gates

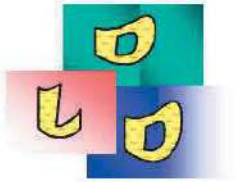


(b) With NAND gates



Parity Generation & Checking

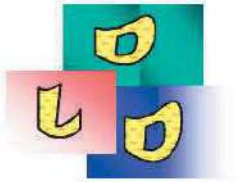
- A parity bit is used for detecting errors during transmissions of binary information.
- A parity bit is an extra bit included with a binary message to make the number of 1's either even or odd.
- The message, including the parity bit is transmitted and then checked at the receiving end for errors.
- An error is detected if the checked parity does not correspond with the one transmitted.
- The circuit that generates the parity bit in the transmitter is called a parity generator. The circuit that checks the parity in the receiver is called parity checker.



Parity Bit Generation

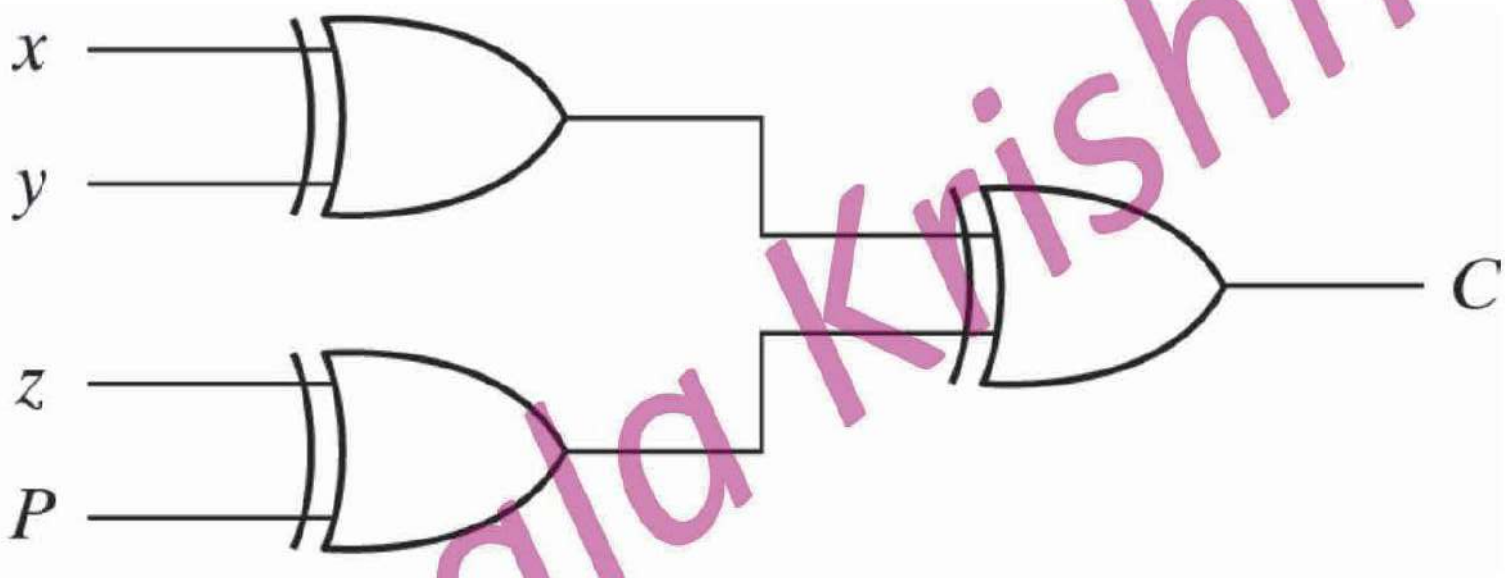
- As an example, consider a 3-bit message to be transmitted together with an even parity bit. For even parity the parity bit P is to be generated to make the number of 1's in the message even including the parity.

Three – Bit Message			Parity Bit
X	Y	Z	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



Parity Bit Checking

FOUR BITS RECIEVED				PARITY CHECK
X	Y	Z	P	C
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0



Four Bit Even Parity Checker