Combinational Logic Circuits

Instructor: Saraju P Mohanty

(Part 2)

- Map Manipulation
- NAND and NOR Gates
- Exclusive-OR (XOR) Gates

Map Manipulations

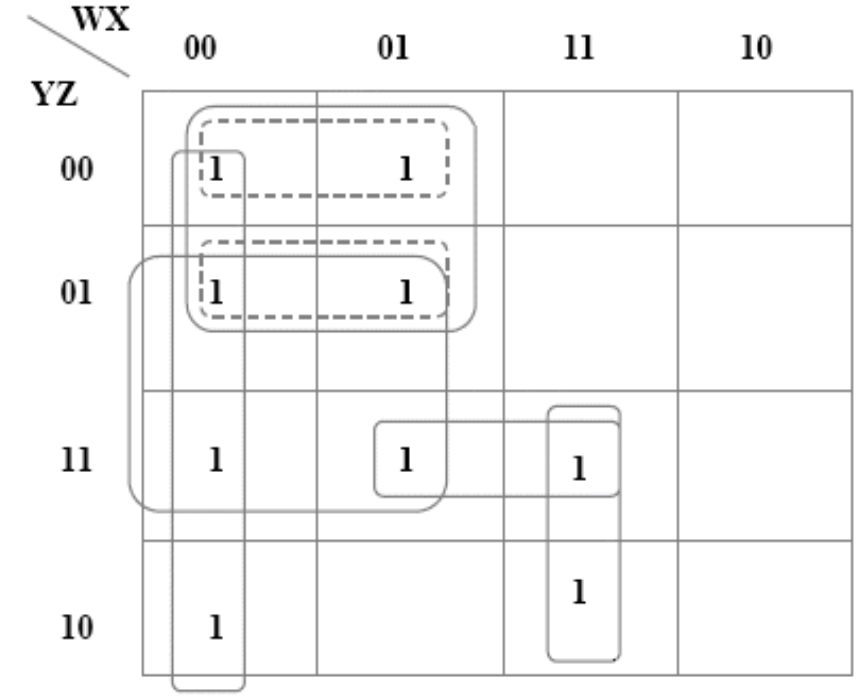
- While combining the squares in K-maps it is necessary to ensure that all the minterms are included.
- To minimize the number of terms in the simplified function any redundant term has to be avoided. So, the procedure of merging the squares need to be systematic.
- Three terms introduced to systematize the combining procedure are listed below.
  - Implicant
  - Prime Implicant
  - Essential Prime Implicant

Map Manipulations: Terminology

- An *implicant* is a product term (product of one or more literals) that could be used to cover minterms of the function. All the rectangles on a map made up of squares containing 1's correspond to implicants.
- A *prime implicant* is an implicant that is not a part of any other implicant of the function. Equivalently, prime implicant is a set of squares that is not a subset of any set containing larger number of squares.
- An *essential prime implicant* is a prime implicant that covers at least one minterm that is not covered by any other prime implicant.
- NOTE: Some of the implicants are prime implicants and some of the prime implicants are essential prime implicants.

Map Manipulations : Terminology Example

$F = \Sigma_{W,X,,Y,Z}\ (0,1,2,3,4,5,7,14,15)$



Prime implicants:
- W'X'
- W'Y'
- W'Z
- XYZ
- WXY

Essential prime implicants:
- W'X'
- W'Y'
- WXY

Note:
Implicants W'Y'Z' , W'Y'Z
- are not prime
- are contained in larger block W'Y'

NOTE: Implicants are, all single minterms + all groups of two / four / sixteen minterms.

Map Manipulations: Simplification using Prime Implicants

Rule: Determine all prime implicants. The simplified function is the logical sum of all the essential prime implicants and other prime implicants needed to include the remaining minterms not included in the essential prime implicant.

Example:

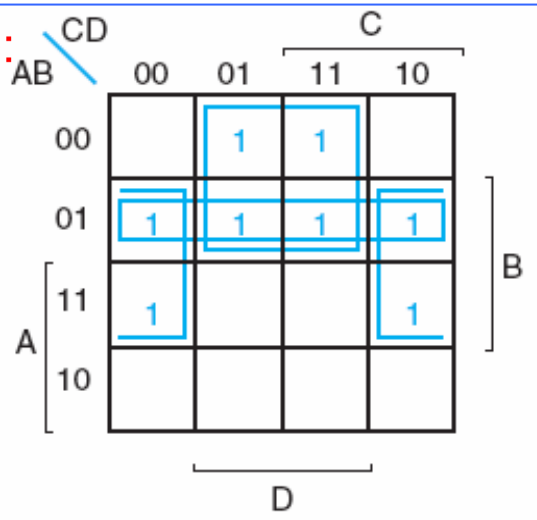


Fig. 2-21 Prime Implicants for Example 2-7: $\bar{A}D$, $B\bar{D}$, and $\bar{A}B$

The terms A'D and BD' are essential prime implicants since minterms 1 and 3 are covered by A'D only, and minterms 12 and 14 are covered by BD' only. As the minterms 4,5,6 and 7 are already included in A'D and BD', the remaining prime implicant A'B is not an essential prime implicant. The simplified expression is **F = A'D + BD'**.

NOTE: In this example, we do not need A'B since essential prime implicants cover all the minterms.

Map Manipulations: Simplification using Prime Implicants

The product terms A'B'C'D', BC'D, ABC', and AB'C are essential prime implicants as they are the only prime implicants covering 0, 5,12, and 10 respectively. Minterm 15 is included in two nonessential prime implicants.

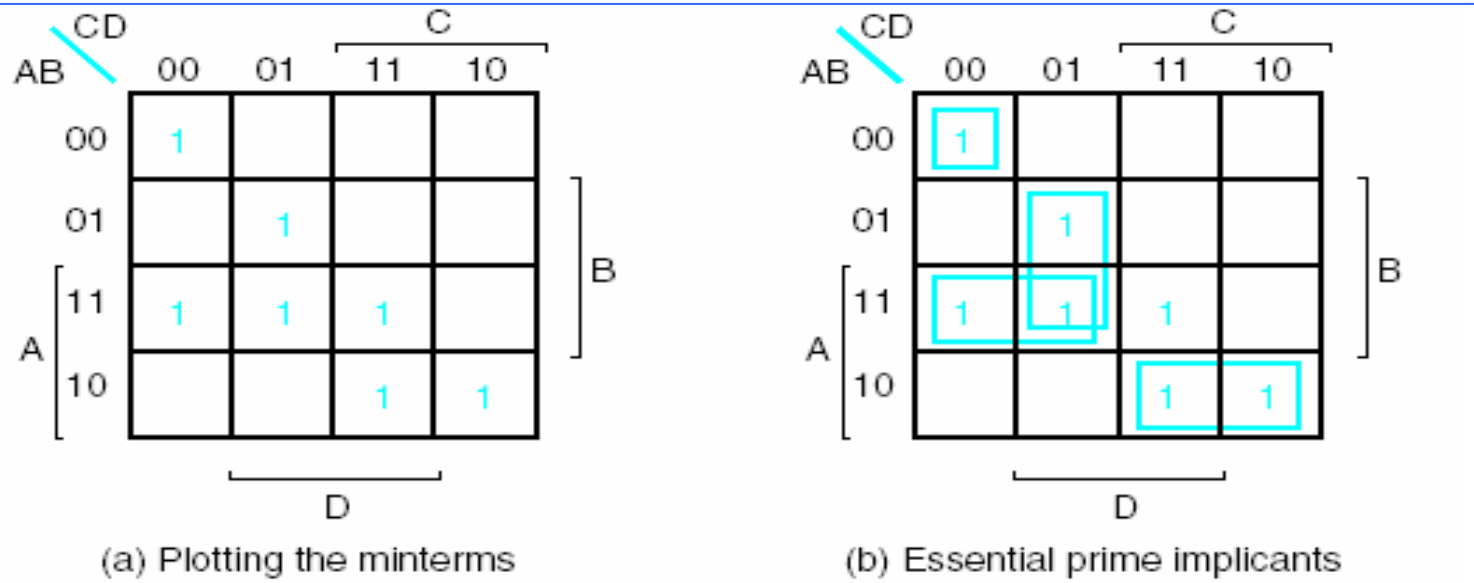


Fig. 2-22 Simplification with Prime Implicants in Example 2-8

The simplified expression is the logical sum of four essential prime implicants and one prime implicant that include 15.

$$F = A'B'C'D' + BC'D + ABC' + AB'C + ACD \quad \textbf{or}$$

$$F = A'B'C'D' + BC'D + ABC' + AB'C + ABD$$

Map Manipulations: Simplification using Selection Rule

- Use: The rule is used in selecting the nonessential prime implicants.
- Rule: Minimize the overlap among the prime implicants as much as possible. Make sure that each prime implicant selected includes at least one minterm not included in any other prime implicant selected.
- NOTE: The above rule will results in simplified, but not necessarily the minimum cost sum-of-products expression.

Map Manipulations: Selection Rule Example

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

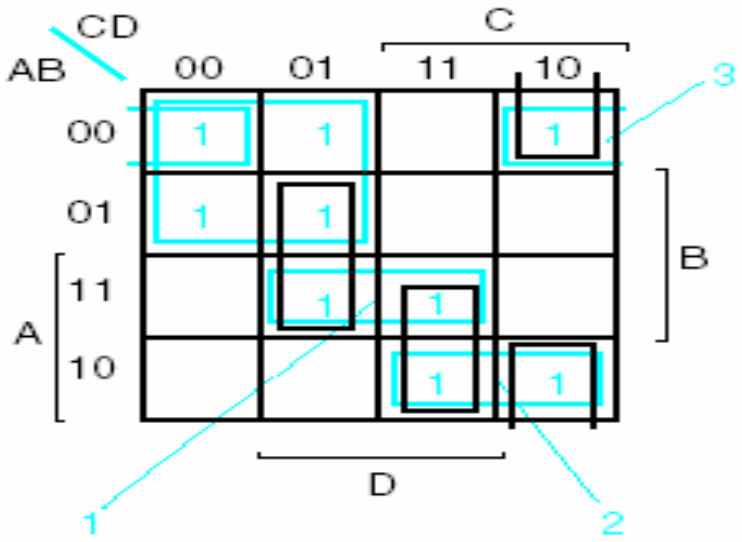


Fig. 2-23 Map for Example 2-9

The only essential prime implicant is A'C'. Using the selection rule, the nonessential prime implicants are chosen (shown as 1,2,3 in Fig.). Thus, the simplified expression is F = A'C' + ABD + AB'C + A'B'D'

Map Manipulations: Simplifying as product-of-sums

Mark the empty squares (not containing 1's) with 0's and combine them into valid rectangles to obtain an simplified expression for the complement of the function. Taking the complement of this results in the simplified SOP expression.

Example:

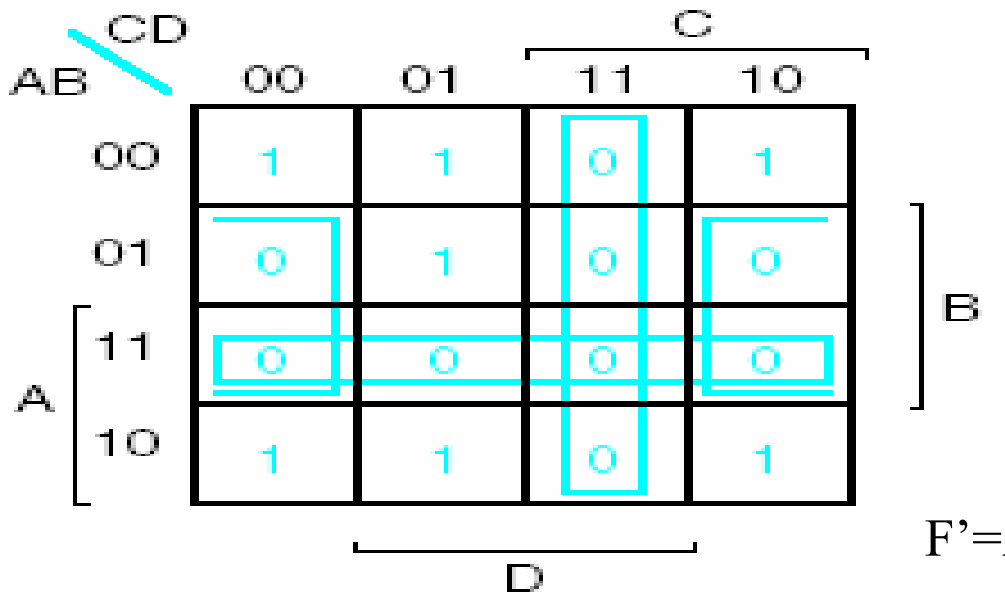


Fig. 2-24 Map for Example 2-10: $F = (\overline{A} + \overline{B})\,(\overline{C} + \overline{D})\,(\overline{B} + D)$

NOTE: Many more algorithms available.

Map Manipulations: Don't Care Conditions

We do not care for the values assumed by the function for the unspecified minterms (marked by x in the K-map). The don't-care minterms are chosen if at all it contribute to larger implicant.

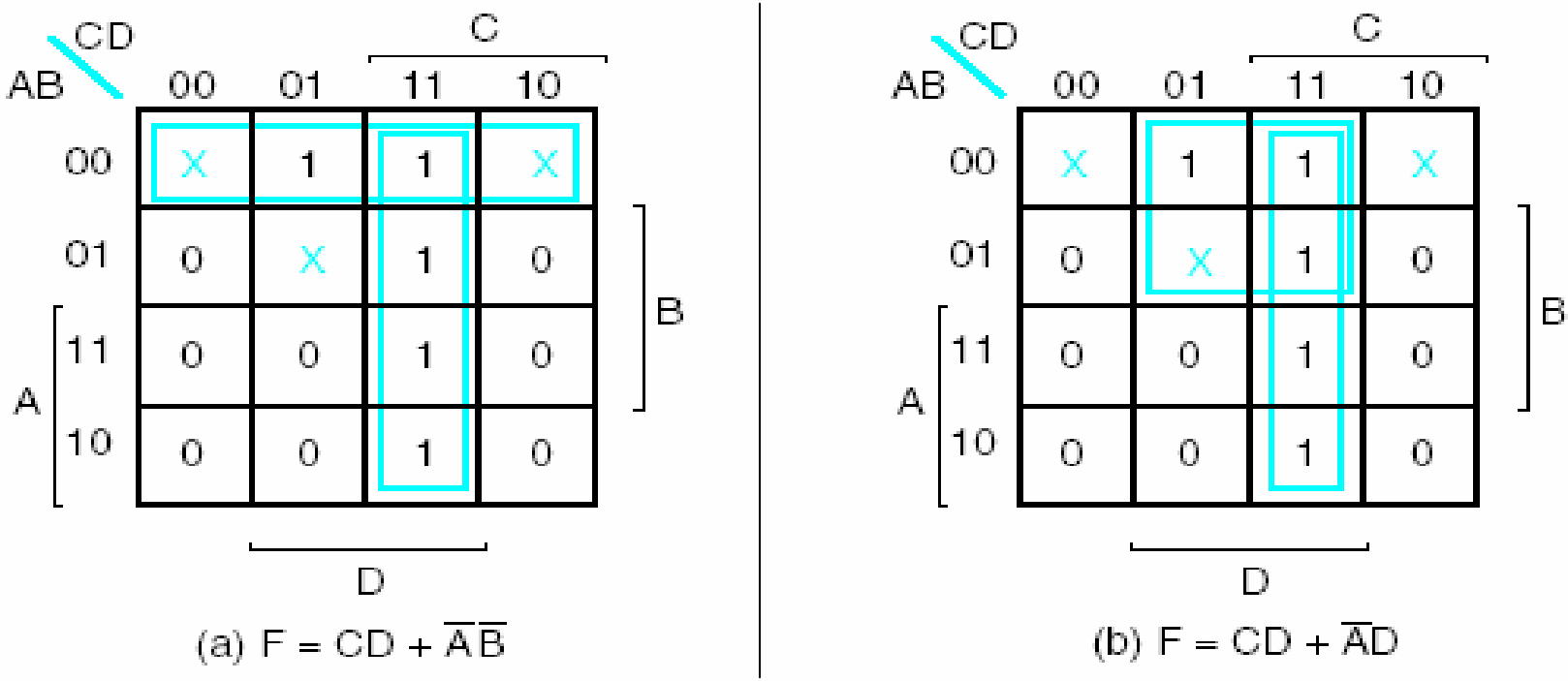


Fig. 2-25 Example with Don't-Care Conditions

don't-care minterms 0 and 2 are included.

don't-care minterm 5 included.

What Gates to be used in design ??

- OR, AND, NOT are basic gates and can implement all logic functions. NAND and NOR are universal gates and can also implement all types of logic functions.
- What is a good choice ??

  The deciding factors are :
  - Cost
  - Possibility of extending the gate to more than two inputs
  - ..and so on.
- NAND and NOR are more popular choices.

IEEE Standard Graphics Symbols

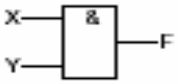
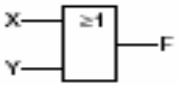
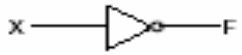
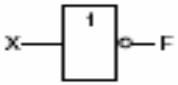
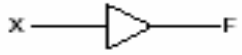
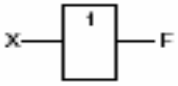
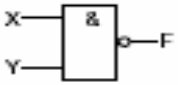
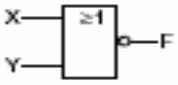
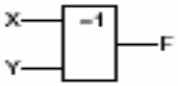
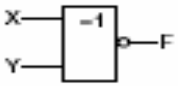
Name	Commonly used	IEEE	Algebraic equation	Truth table
AND			$F = XY$	X Y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR			$F = X + Y$	X Y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT (inverter)			$F = \overline{X}$	X | F 0 | 1 1 | 0
Buffer			$F = X$	X | F 0 | 0 1 | 1
NAND			$F = \overline{X \cdot Y}$	X Y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR			$F = \overline{X + Y}$	X Y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive–OR (XOR)			$F = X\overline{Y} + \overline{X}Y$ $= X \oplus Y$	X Y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive–NOR (XNOR)			$F = XY + \overline{X}\overline{Y}$ $= \overline{X \oplus Y}$	X Y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Fig. 2-26 Digital Logic Gates

NAND Circuits ⟶ AND, OR, NOT

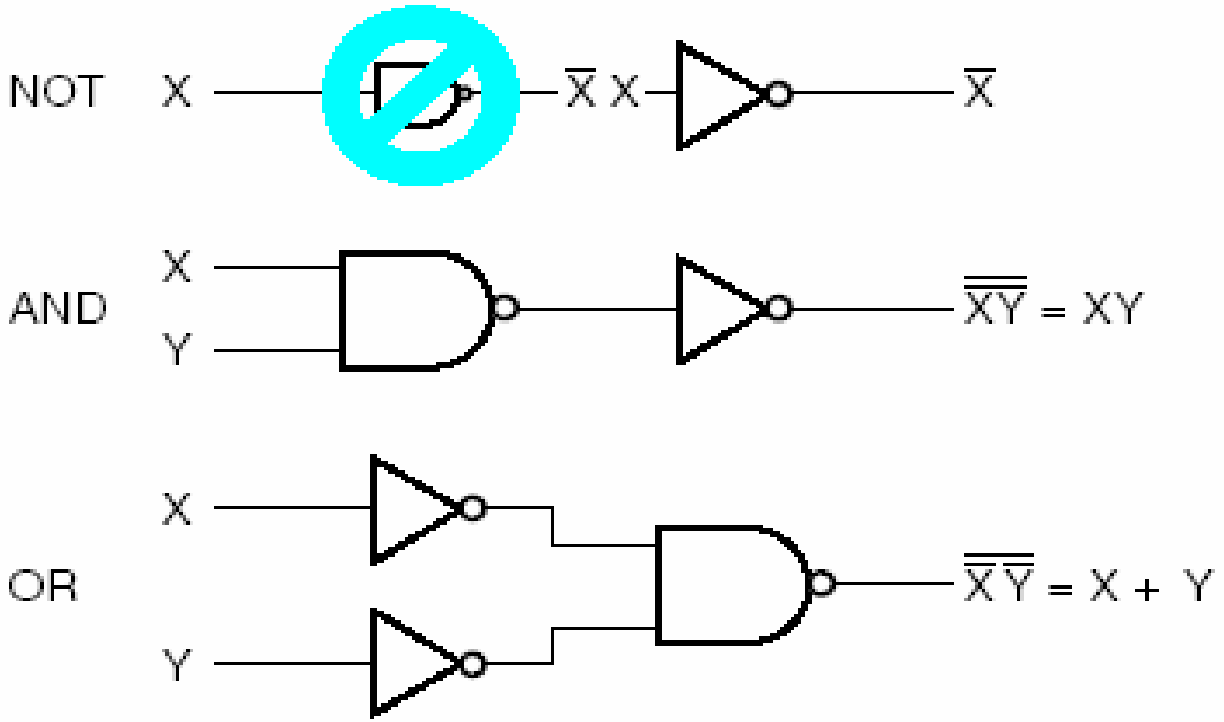


Fig. 2-27 Logical Operations with NAND Gates

NAND Circuits: Alternative Symbols

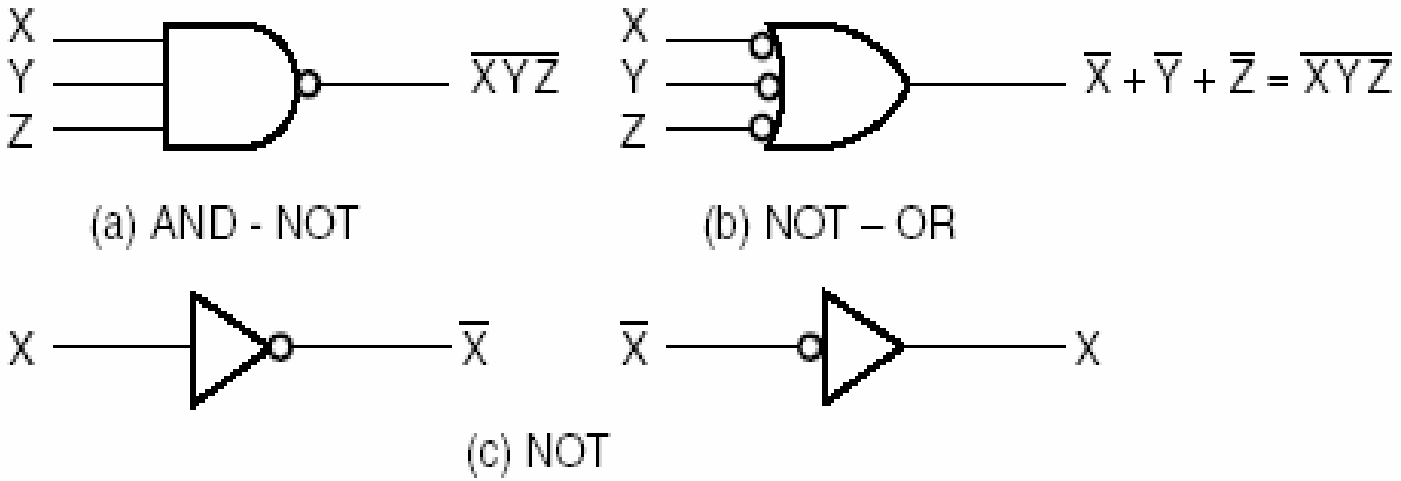


Fig. 2-28 Alternative Graphics Symbols for NAND and NOT Gates

NOTE: Helpful for logic conversion.

NAND Circuits: Two-Level Implementation

Sum-of-products form : AND gate is first level and OR gate is second level. SOP is more convenient for NAND implementations.

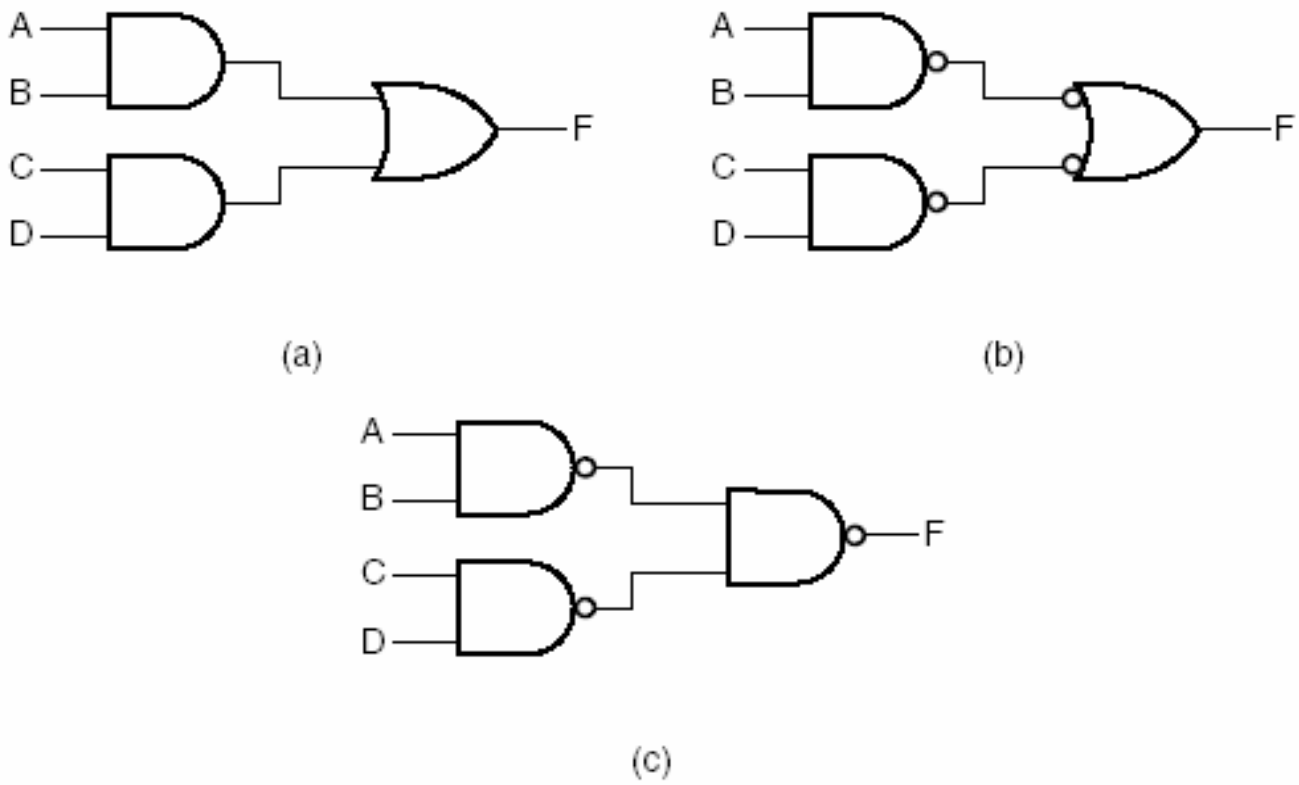


Fig. 2-29 Three Ways to Implement F = AB + CD

Two-Level NAND Implementation: General Procedure

- Simplify the function and express it in sum-of-products form.
- Draw a NAND gate for each product term of the expression that has at least two literals. The literals become the inputs for the NAND gates. This group of gates is the first level.
- Draw a single gate using AND-NOT or the NOT-OR configuration at the second level with inputs coming from the output of the first-level.
- A product term with single literal requires a NOT at the first level. If the single literal is a complement then it can be directly connected as the input pf the second-level NAND gate.

Two-Level NAND Implementation: Example

Example : Implement F(X,Y,Z) = Σm(1,2,3,4,5,7) using NAND gates.

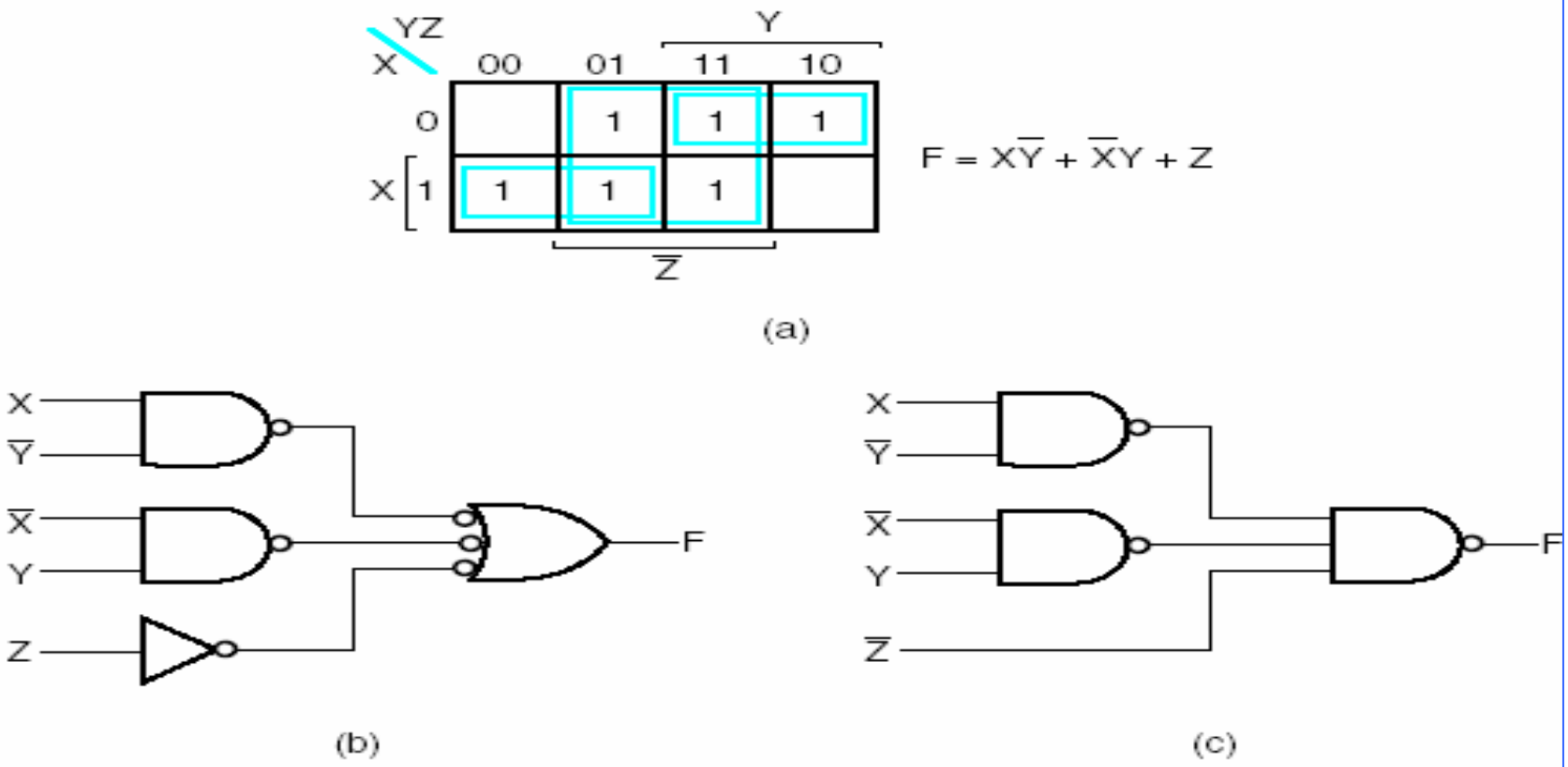
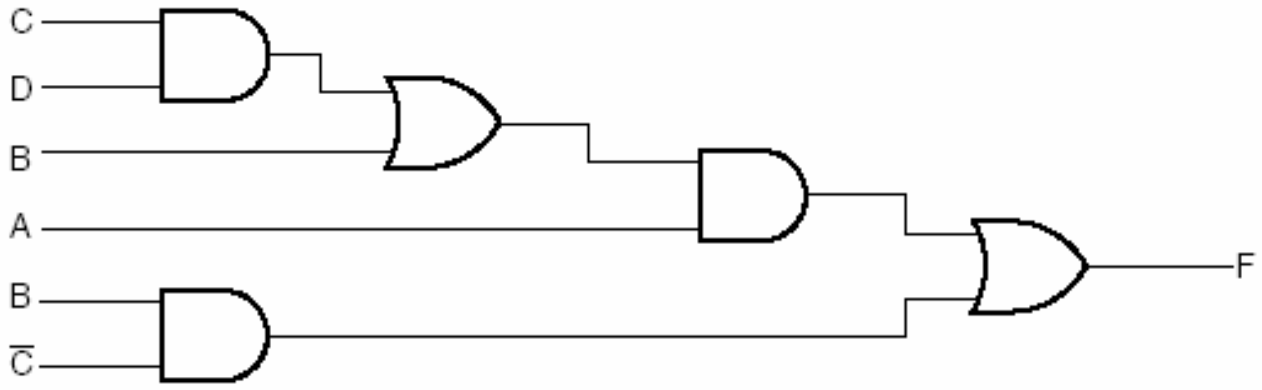


Fig. 2-30 Solution to Example 2-12

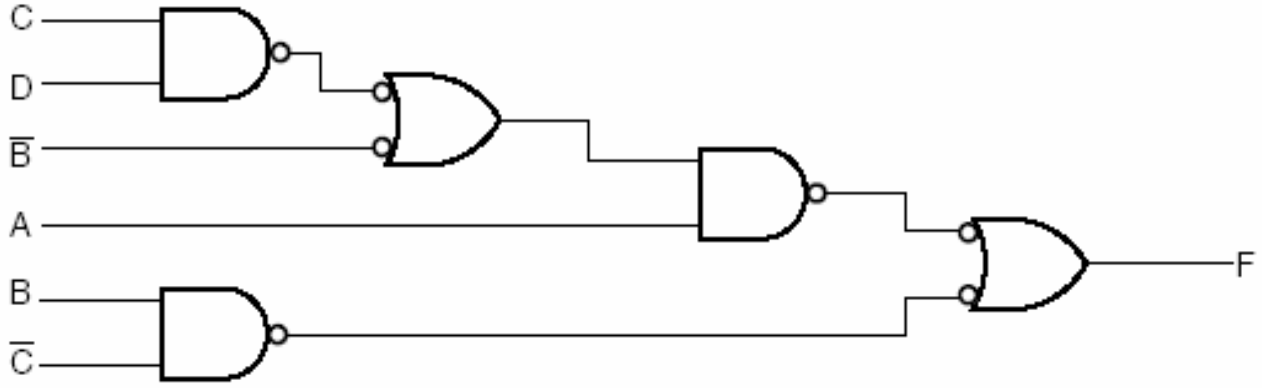
Multilevel NAND Implementation: General Procedure

- Simplify the function and express it in sum-of-products.
- Implement the function using AND, OR, and NOT gates.
- Use the following steps to convert from AND-OR implementation to NAND-NAND implementation.
- Convert all AND gates to NAND gates with AND-NOT graphic symbols.
- Convert all OR gates to NAND gates with NOT-OR graphic symbols.
- Check all the bubbles in the diagram. For every bubble that is not counteracted by another bubble along the same line, insert a NOT gate or complement the input literal from its original appearance.

Multilevel NAND Implementation: Example1



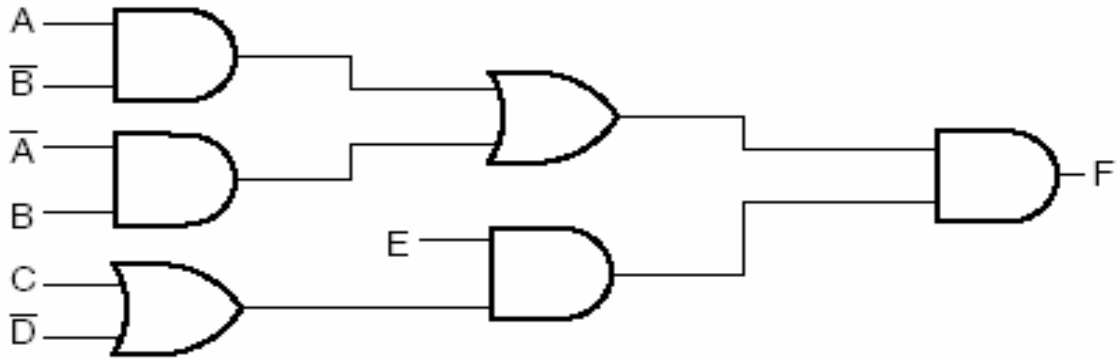
(a) AND – OR gates



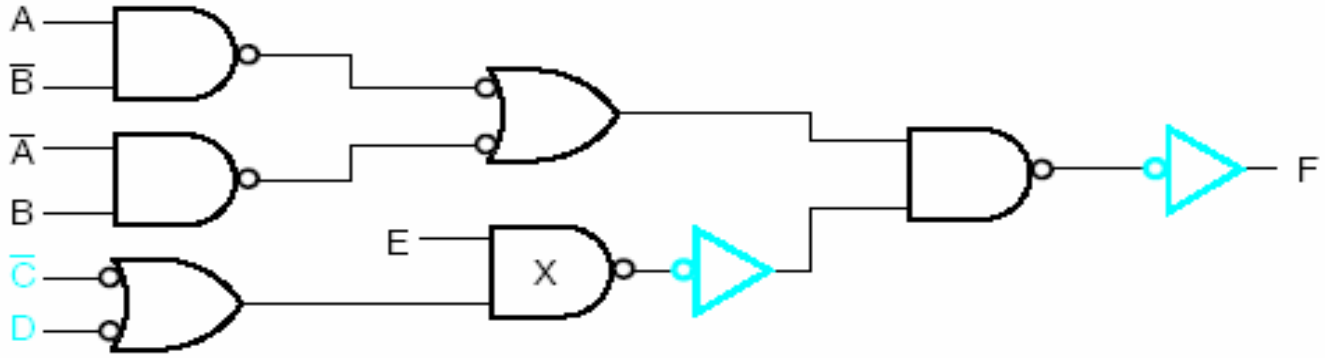
(b) NAND gates

Fig. 2-31 Implementing $F = A\,(CD + B) + B\overline{C}$

Multilevel NAND Implementation: Example2



(a) AND – OR gates



(b) NAND gates

Fig. 2-32 Implementing $F = (A\overline{B} + \overline{A}B)E(C + \overline{D})$

NOR Circuits ⟶ AND, OR, NOT

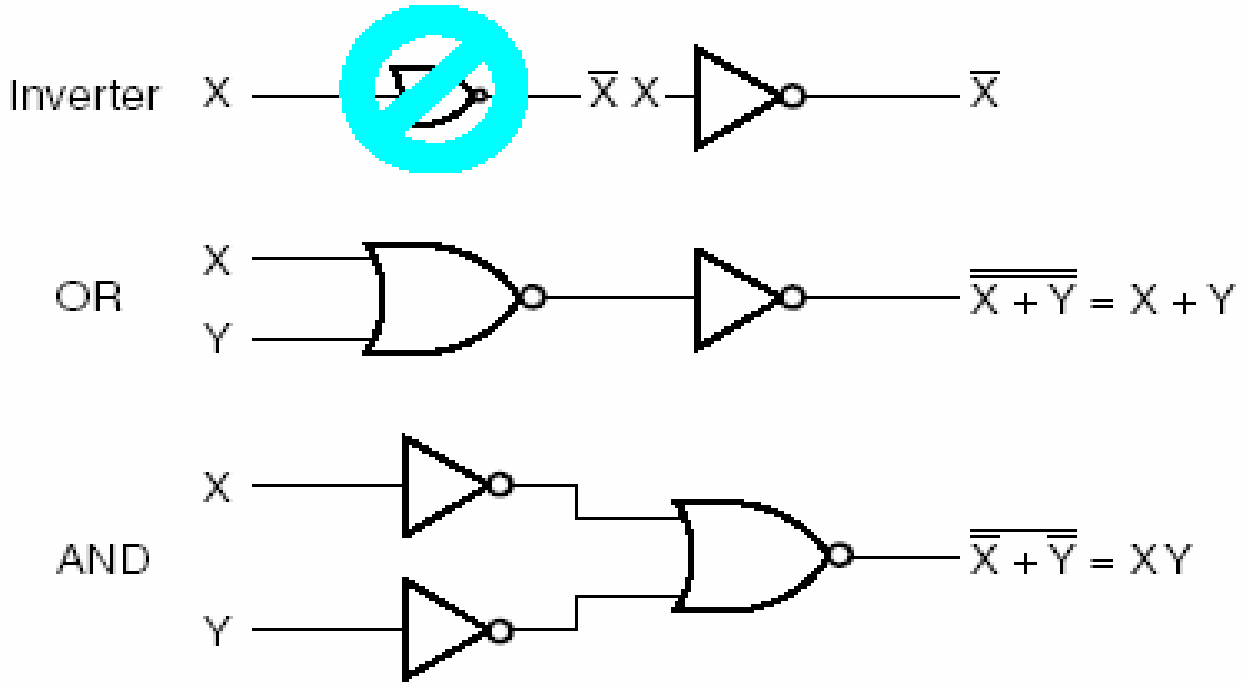


Fig. 2-33 Logic Operations with NOR Gates

NOR Circuits: Alternative Symbols

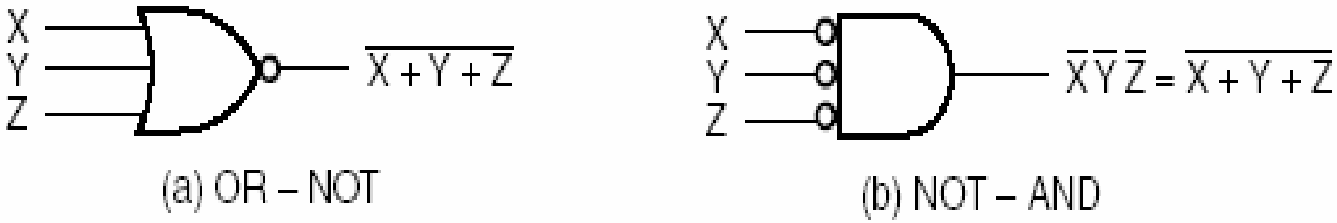


Fig. 2-34 Two Graphic Symbols for NOR Gate

NOR Implementation: General Procedure

- Simplify the function and express it in product-of-sums.
- Implement the function using OR, AND, and NOT gates.
- Use the following steps to convert from OR-AND implementation to NOR-NOR implementation.
- Convert all OR gates to NOR gates with OR-NOT graphic symbols.
- Convert all AND gates to NOR gates with NOT-AND graphic symbols.
- Check all the bubbles in the diagram. For every bubble that is not counteracted by another bubble along the same line, insert a NOT gate or complement the input literal from its original appearance.

NOR Implementation: Example 1

First obtain the OR-AND implementation and then convert it to NOR implementation.

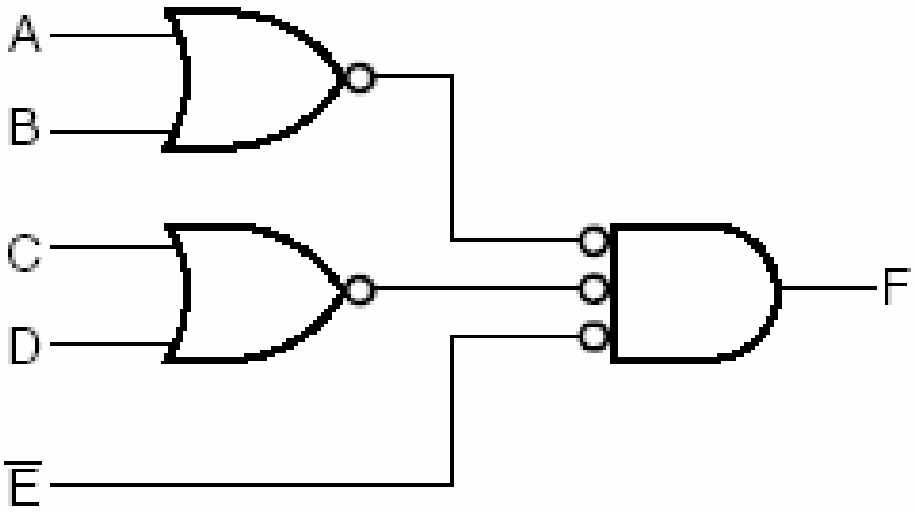


Fig. 2-35 Implementing $F = (A + B)(C + D)E$ with NOR Gates

NOR Implementation: Example 2

First obtain the OR-AND implementation and then convert it to NOR implementation using the steps mentioned before.

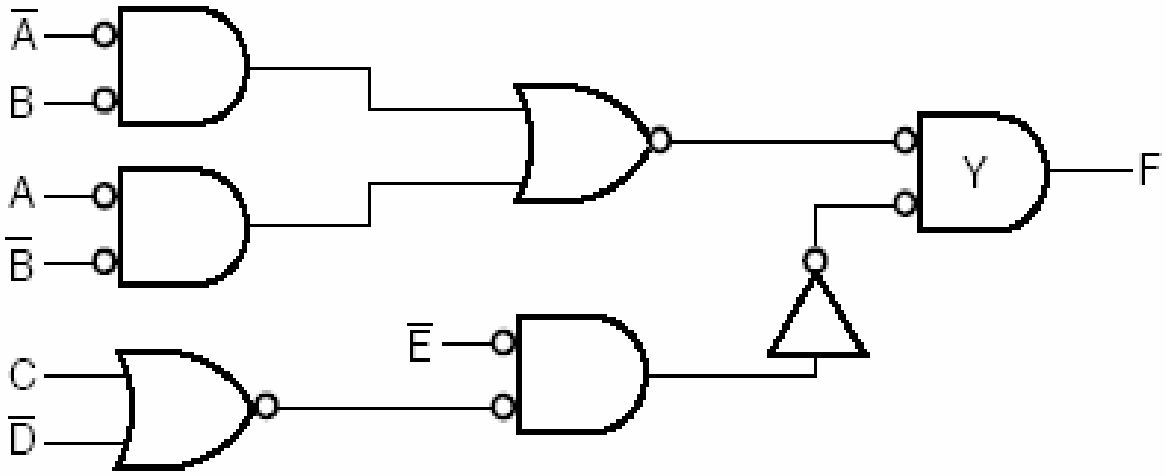


Fig. 2-36 Implementing $F = (A\overline{B} + \overline{A}B)\,E\,(C + \overline{D})$ with NOR Gates

Exclusive-OR Gates

- XOR performs the function F=XY'+X'Y.
- The truth table is 1 when one of the inputs is 1.
- The complement of XOR is XNOR; that is F'=XY+X'Y'.
- XNOR is 1 when both inputs are 1 or both inputs are 0.

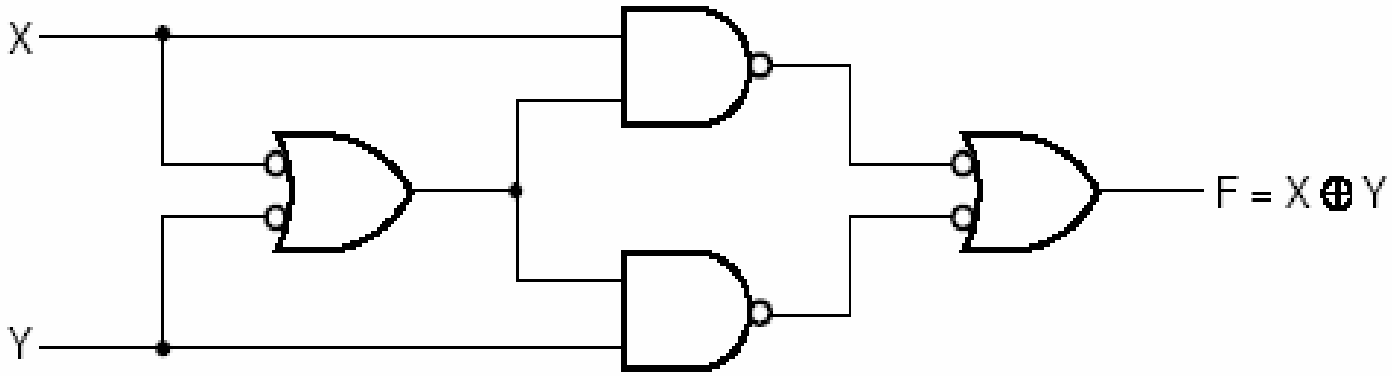


Fig. 2-37 Exclusive-OR Constructed with NAND Gates

- XNOR is also known as the equivalence.

Exclusive-OR : Identities

1. X XOR 0 = X
2. X XOR 1 = X'
3. X XOR X = 0
4. X XOR X' = 1
5. X XOR Y' = (X XOR Y)'
6. X' XOR Y = (X XOR Y)'
7. X XOR Y = Y XOR X
8. (X XOR Y) XOR Z = X XOR (Y XOR Z) = X XOR Y XOR Z

Exclusive-OR : ODD Function

- In general sense the XOR function is known as odd function.
- The term "exclusive-OR" is more appropriate for two-variable case.
- Why odd function ? The binary values of all the minterms have odd number of 1's.
- Even function: Binary values of all minterms have even number of 1's.

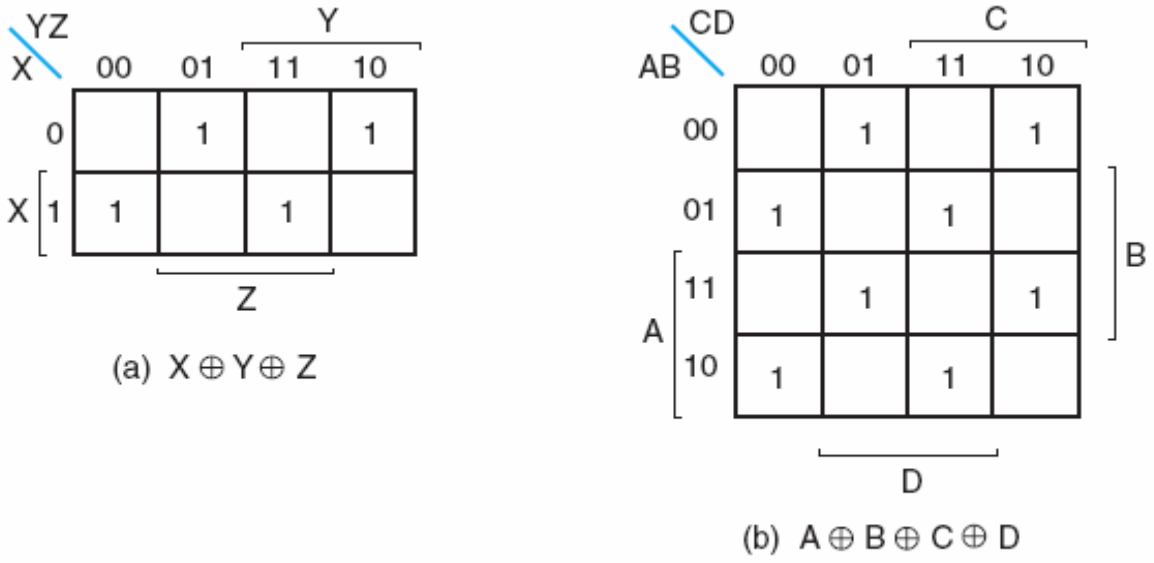


Fig. 2-38 Maps for Multiple-Variable Odd Functions

XOR : Parity Generation and Checking

- As discussed before, parity bit is an extra bit added to the sequence to make it either even or odd parity.
- Used for error detection during transmission of binary information.
- Odd (and even) functions are used for parity bit generation and checking.
- Parity Generator is a circuit that generates parity bit at the transmitter.
- Parity Checker is a circuit that checks parity bit at the receiver.

Parity Generator Circuit using XOR

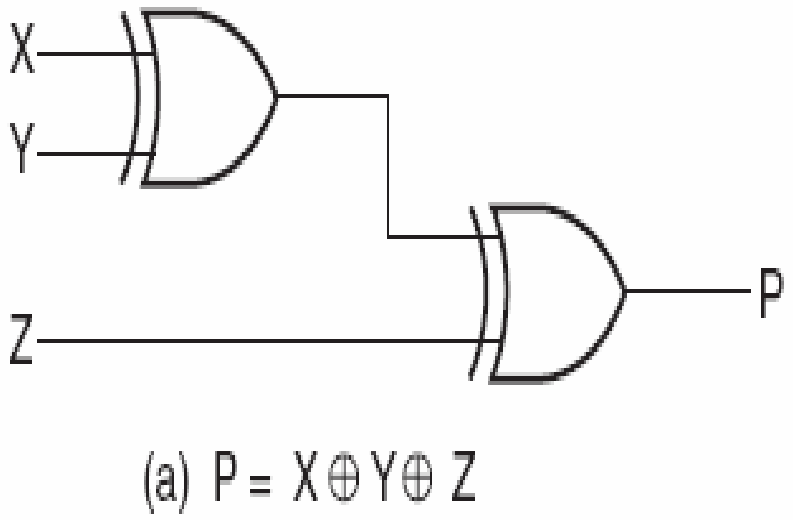
Design of a even parity generator is discussed. The input message is 3-bits. The parity bit generated as 1 if the number of 1's in the 3-bit message is odd, else the generated bit is 0.

Similarly, the design of a parity checker can be considered.

TABLE 2-9
Truth Table for an Even Parity Generator

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

Table 2-9 Truth Table for an Even Parity Generator



Logic Diagram