Multilevel NAND Circuits (SOP)

Lecture-15

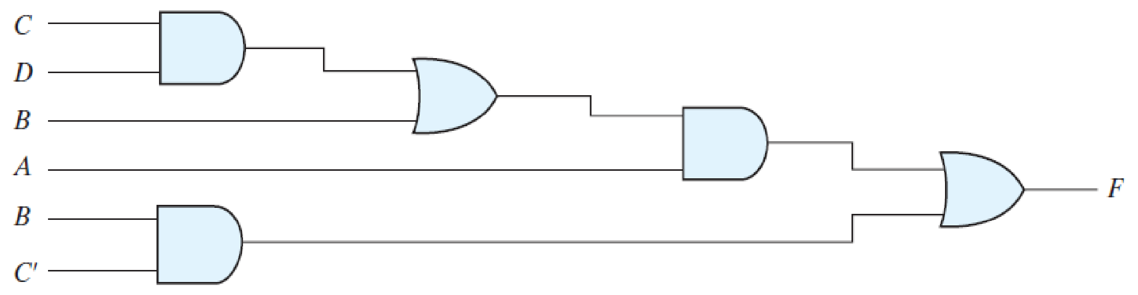
By

Bhagyalaxmi Behera

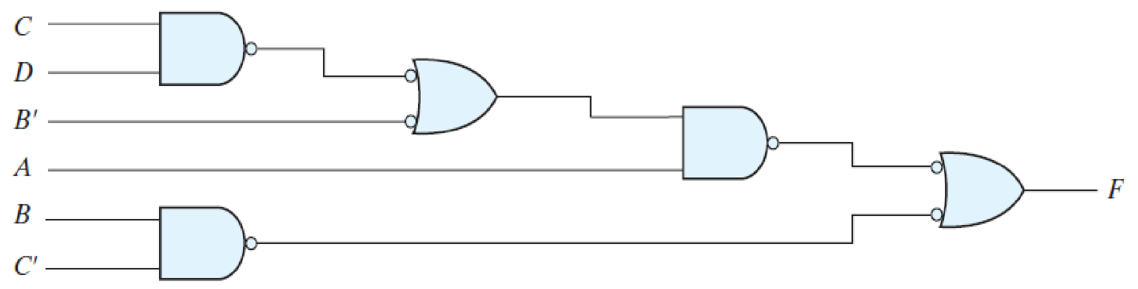
Asst. Professor (Dept. of ECE)

Multilevel NAND Circuits (SOP)

$F = A\,(CD + B) + BC'$



(a) AND–OR gates



(b) NAND gates

The general procedure for converting a multilevel AND–OR diagram into an all-NAND diagram using mixed notation is as follows:

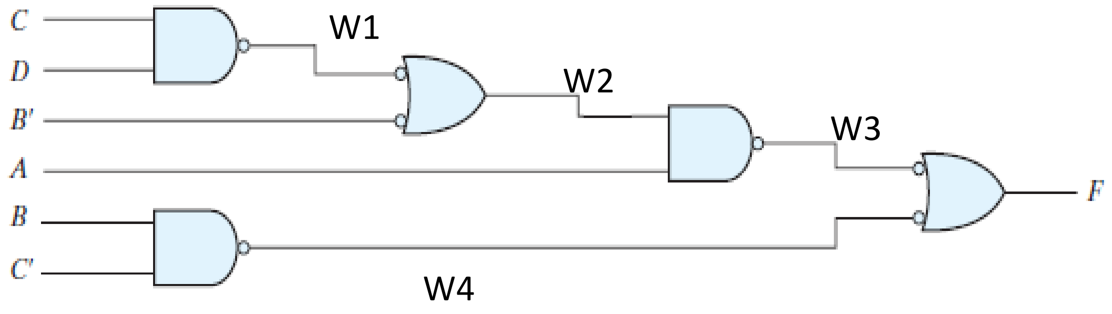
1. Convert all AND gates to NAND gates with AND-invert graphic symbols.

2. Convert all OR gates to NAND gates with invert-OR graphic symbols.

3. Check all the bubbles in the diagram. For every bubble that is not compensated by another small circle along the same line, insert an inverter (a one-input NAND gate) or complement the input literal.

HDL for the above Multilevel NAND Circuit (SOP)

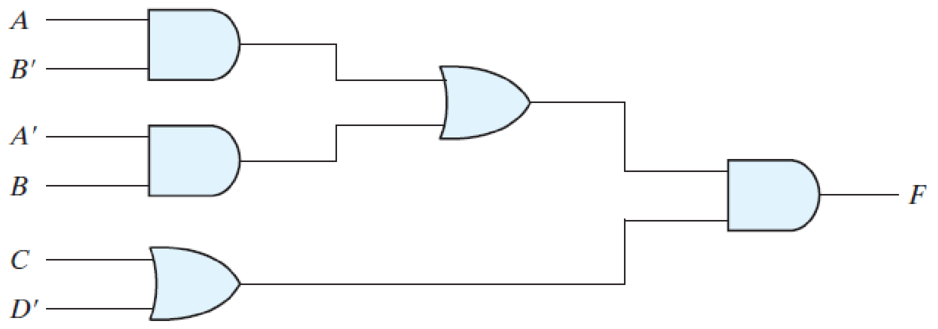
```
// Verilog model of Multilevel level
implementation circuit using NAND gates.
module  multi-level-gf (F, A, B, C, D);
output F;
input A, B, C, D;
wire B’, C’, w1, w2, w3, w4;
nand (B’, B, B),
        (C’, C, C);
nand G1 (w1, C, D),
        G2 (w2, w1,B’),
        G3 (w3, A, w2),
        G4 (w4, B, C’),
        G5 (F, w3, w4);
endmodule
```



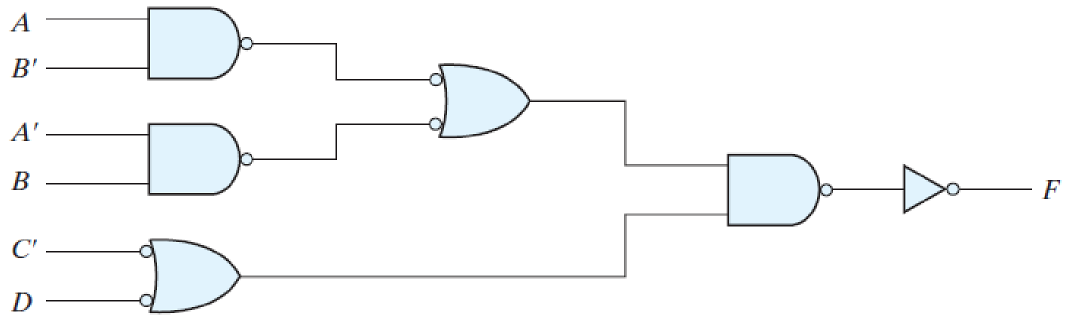
(b) NAND gates

Multilevel NAND Circuits (POS)

F = (AB' + A'B)(C + D')



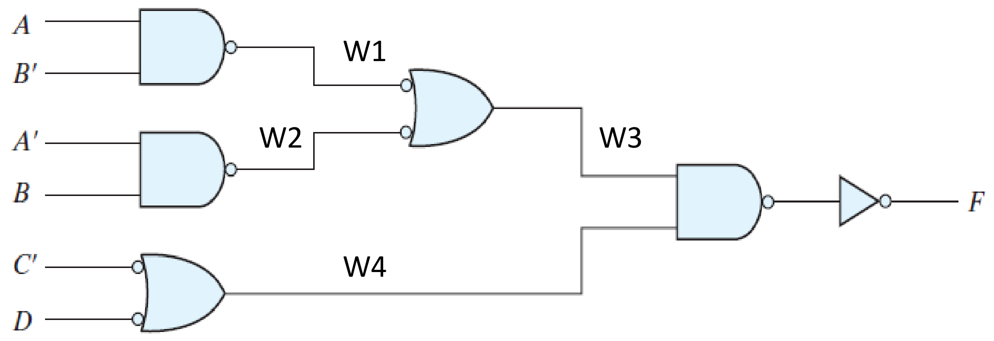
(a) AND–OR gates



(b) NAND gates

HDL for the above Multilevel NAND Circuit (POS)

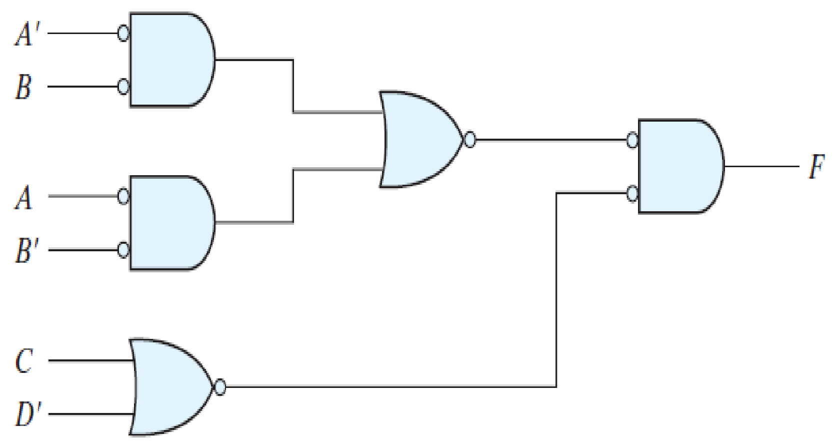
```
module Fig_3_23b_gates (F, A, A_bar, B, B_bar, C_bar, D);
output F;
input A, A_bar, B, B_bar, C_bar, D;
wire w1, w2, w3, w4;
nand (w1, A, B_bar);
nand (w2, A_bar, B);
not (w1_bar, w1);
not (w2_bar, w2);
or (w3, w1_bar, w2_bar);
or (w4, C, D_bar);
not (w5, C_bar);
not (w6, D);
nand (F_bar, w3, w4);
not (F, F_bar);
endmodule
```



(b) NAND gates

Multilevel NOR Implementation Example

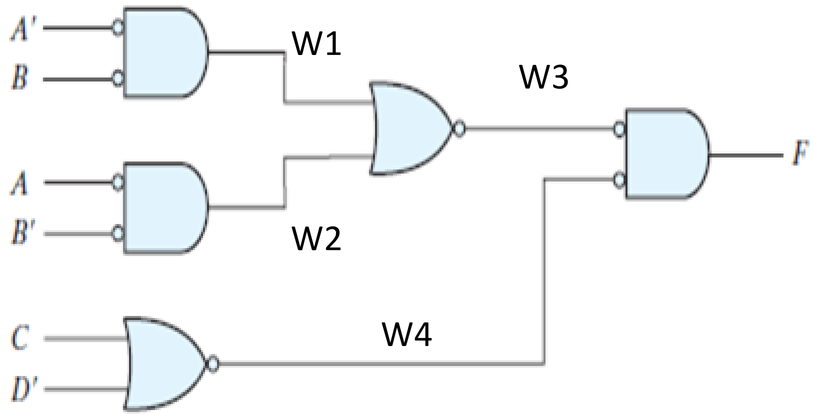
$F = (AB' + A'B)(C + D')$



HDL for the above Boolean function using NOR gates

$F = (AB' + A'B)(C + D')$

```
// Verilog model of Boolean function using
NOR gates.
module  bolean-usingnor-gf (F, A, B, C,
D);
output  F;
input A, B, C, D;
wire A', B', D', w1, w2, w3, w4;
nor  (A', A, A),
      (B', B, B),
      (D', D, D);
nor  G1 (w1, A', B),
       G2 (w2, A, B'),
       G3 (w4, C, D'),
       G4 (w3, w1, w2),
       G5 (F, w3, w4);
endmodule
```



Example

Draw a multilevel NOR circuit for the following expression:

(AB´ + CD ´) E + BC (A+B)

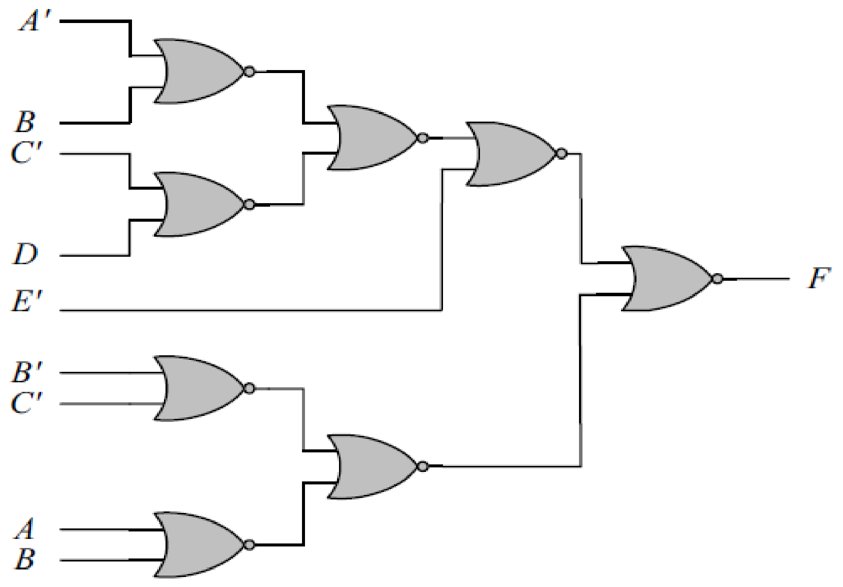
Multi-level NOR:

$F = (AB' + CD')E + BC(A + B)$

$F' = [(AB' + CD')E + BC(A + B)]'$

$F' = [\,[(AB' + CD')' + E']' + [\,(BC)' + (A + B)']'\,]'$

$F' = [\,[((A' + B)' + (C' + D)')' + E']' + [\,(B' + C')' + (A + B)']'\,]'$



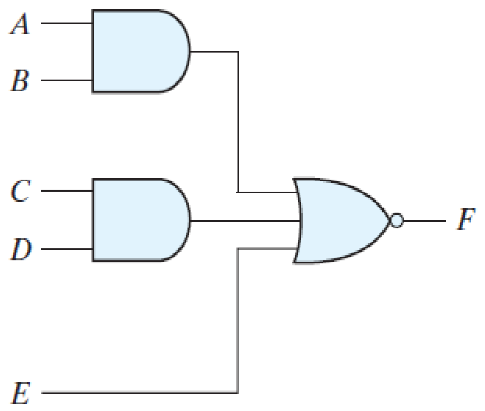
Nondegenerate Forms

The eight *nondegenerate forms produce an implementation in sum-of-products* form or product-of-sums form. The eight *nondegenerate forms are as follows:*

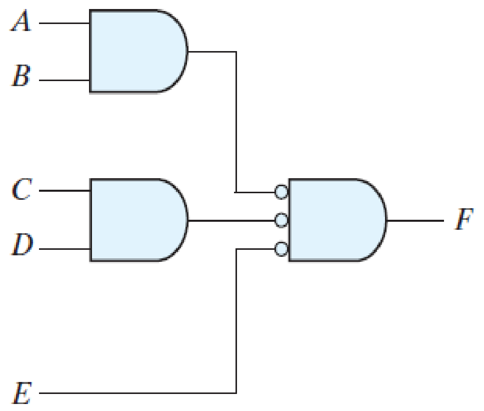
- AND–OR
- OR–AND
- NAND–NAND
- NOR–NOR
- NOR–OR
- NAND–AND
- OR–NAND
- AND–NOR

AND–OR–INVERT Implementation

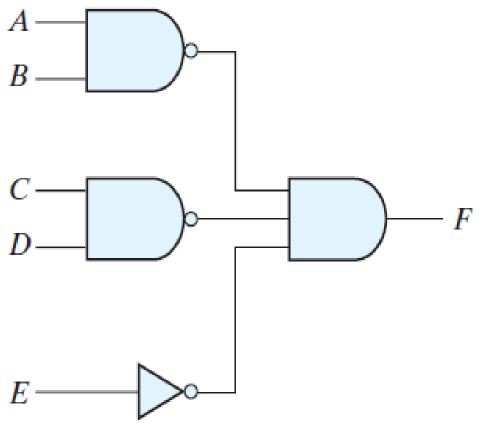
$F = (AB + CD + E)'$



(a) AND–NOR



(b) AND–NOR

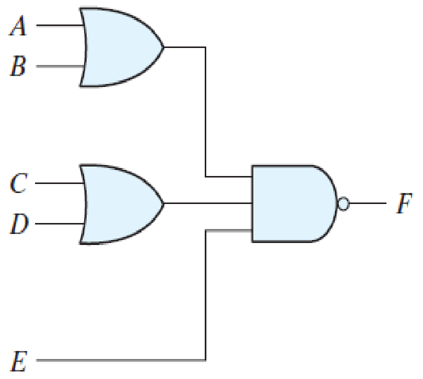


(c) NAND–AND

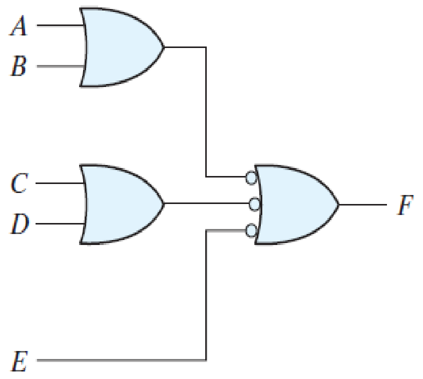
OR–AND–INVERT Implementation

The OR–NAND and NOR–OR forms perform the OR–AND–INVERT function.

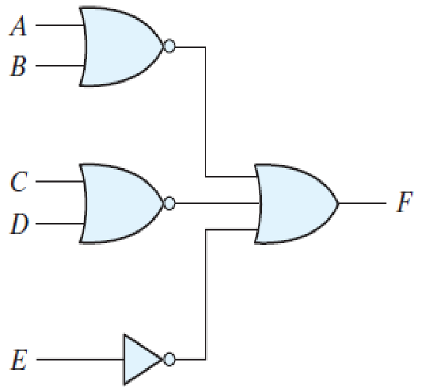
$$F = [(A + B)(C + D)E]'$$



(a) OR–NAND



(b) OR–NAND



(c) NOR–OR

HDL for the above AND-OR-INVERT & OR-AND-INVERT Circuit

$F = (AB + CD + E)'$

```
// Verilog model: Circuit with Boolean
expressions
module Circuit_Boolean_df (F, A, B, C,
D, E);
output  F;
input A, B, C, D, E;
assign F = !((A && B) || (C&& D) ||  E) ;
endmodule
```

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

```
// Verilog model: Circuit with Boolean
expressions
module Circuit_Boolean_df (F, A, B, C,
D, E);
output  F;
input A, B, C, D, E;
assign F = (A || B) && (C || D) && (!E) ;
endmodule
```

Exclusive OR and Equivalence Functions

- The exclusive-OR (XOR), denoted by the symbol $\oplus$,is a logical operation that performs the following Boolean operation:

$$x \oplus y = xy' + x'y$$

- It is particularly useful in arithmetic operations and error detection and correction circuits.
- The exclusive-OR is equal to 1 if only *x is equal to 1 or if only y is equal to 1 (i.e., x and y* differ in value), but not when both are equal to 1 or when both are equal to 0. The exclusive NOR, also known as equivalence, performs the following Boolean operation:

$$(x \oplus y)' = xy + x'y'$$

- The exclusive-NOR is equal to 1 if both *x and y are equal to 1 or if both are equal to 0.* The exclusive-NOR can be shown to be the complement of the exclusive-OR by means of a truth table or by algebraic manipulation:

$$(x \oplus y)' = (xy' + x'y)' = (x' + y)(x + y') = xy + x'y'$$

- The following identities apply to the exclusive-OR operation:

$$x \oplus 0 = x$$
$$x \oplus 1 = x'$$
$$x \oplus x = 0$$
$$x \oplus x' = 1$$
$$x \oplus y' = x' \oplus y = (x \oplus y)'$$

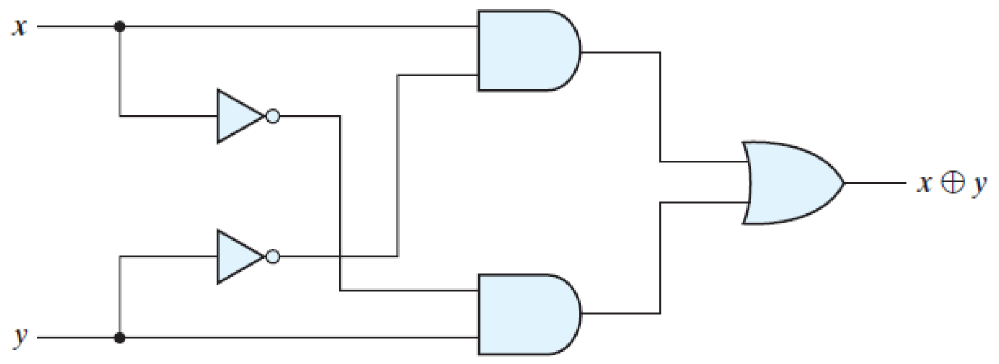
- Also, it can be shown that the exclusive-OR operation is both commutative and associative; that is,

$$A \oplus B = B \oplus A$$

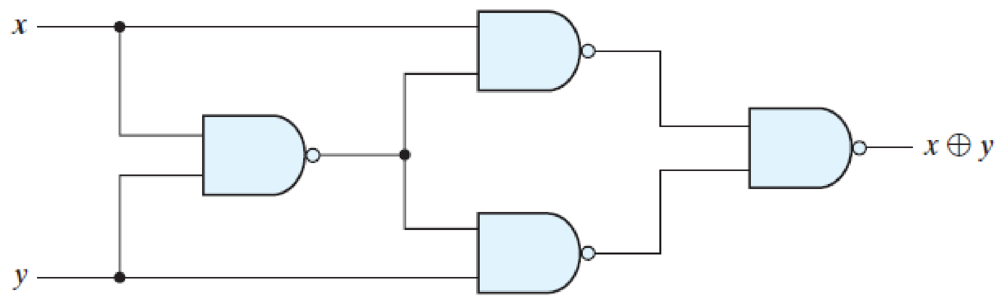
$$(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$$

- This means that the two inputs to an exclusive-OR gate can be interchanged without affecting the operation.

Construction of a two-input exclusive-OR function:



(a) Exclusive-OR with AND–OR–NOT gates

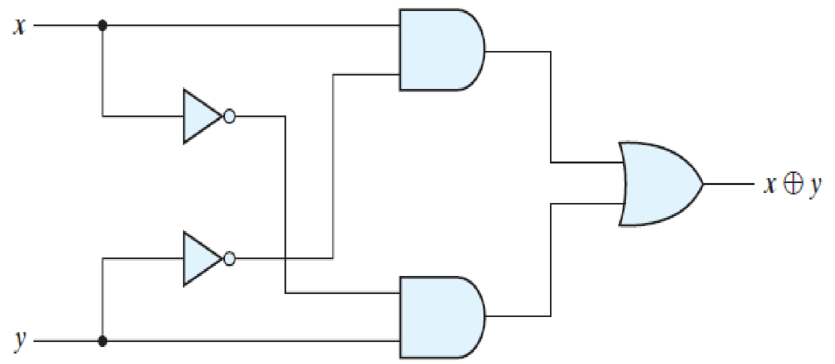


(b) Exclusive-OR with NAND gates

HDL For XOR Gate

```
// Verilog model for XOR gate.
module  exclusiveor-df (F, x, y);
output F;
input x, y;
assign F =  (x && (!y)) || ((!x) && y);
endmodule
```

$$x \oplus y = xy' + x'y$$



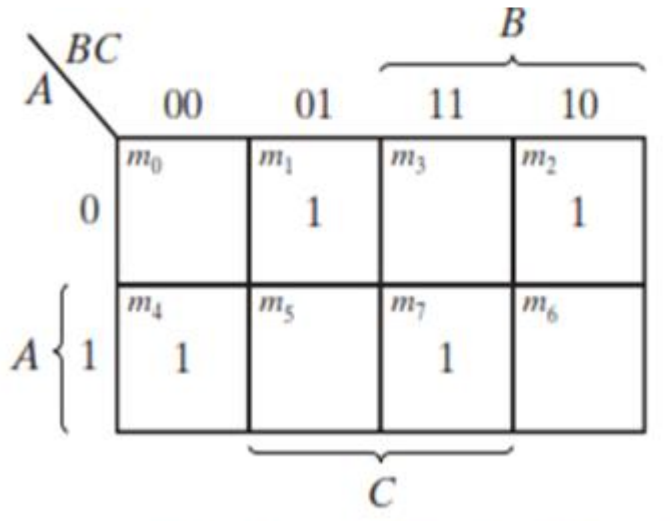
Odd Function

The exclusive-OR operation with three or more variables can be converted into an ordinary Boolean function by replacing the $\oplus$ symbol with its equivalent Boolean expression. In particular, the three-variable case can be converted to a Boolean expression as follows:

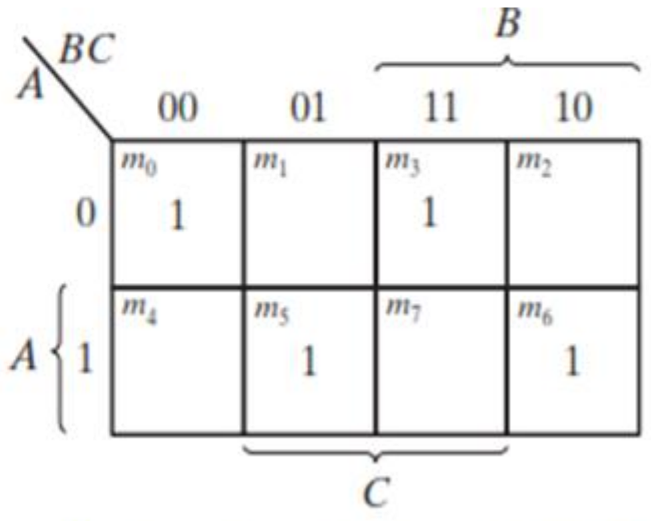
$$\begin{aligned} A \oplus B \oplus C &= (AB' + A'B)C' + (AB + A'B')C \\ &= AB'C' + A'BC' + ABC + A'B'C \\ &= \Sigma(1, 2, 4, 7) \end{aligned}$$

The multiple-variable exclusive-OR operation is defined as an ***odd function*** because in the case of three or more variables the requirement is that an odd number of variables be equal to 1.

- In general, an *n -variable exclusive-OR* function is an odd function defined as the logical sum of the $2^n/2$ minterms whose binary numerical values have an odd number of 1's.
- The definition of an odd function can be clarified by plotting it in a map.



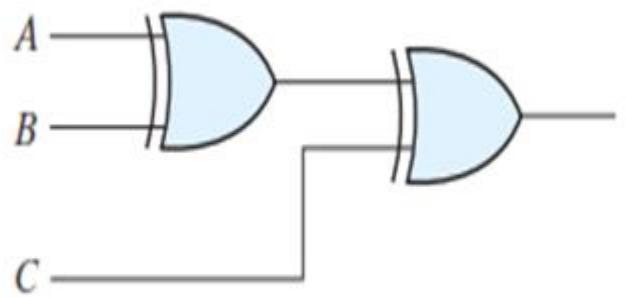
(a) Odd function $F = A \oplus B \oplus C$



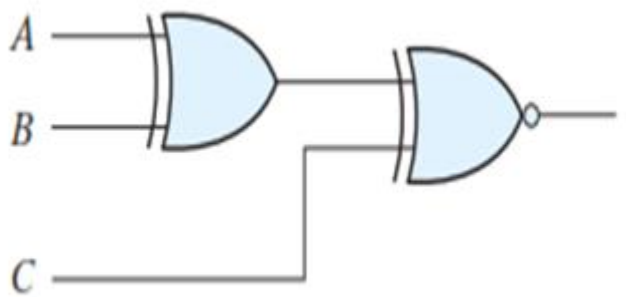
(b) Even function $F = (A \oplus B \oplus C)'$

Map for a three-variable exclusive-OR function

Logic diagram of odd and even functions



(a) 3-input odd function



(b) 3-input even function

The three-input odd function is implemented by means of two-input exclusive-OR gates, as shown in Fig (a). The complement of an odd function is obtained by replacing the output gate with an exclusive-NOR gate, as shown in Fig.(b).

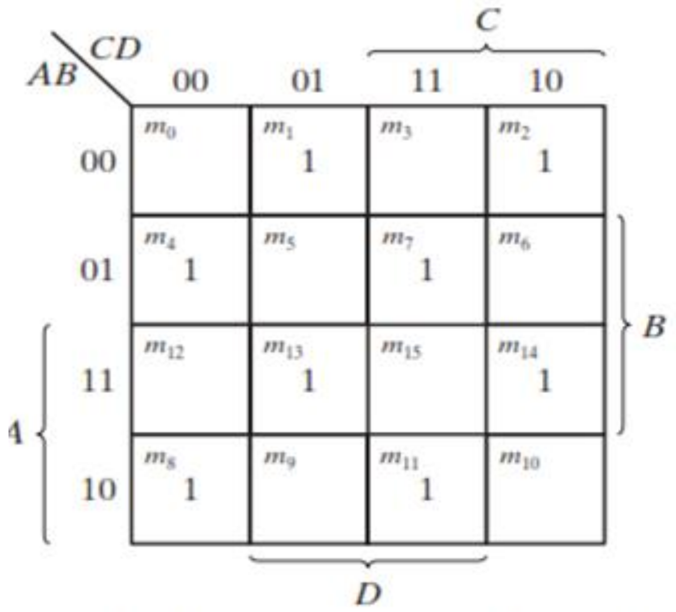
- Consider now the four-variable exclusive-OR operation. By algebraic manipulation, we can obtain the sum of minterms for this function:

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

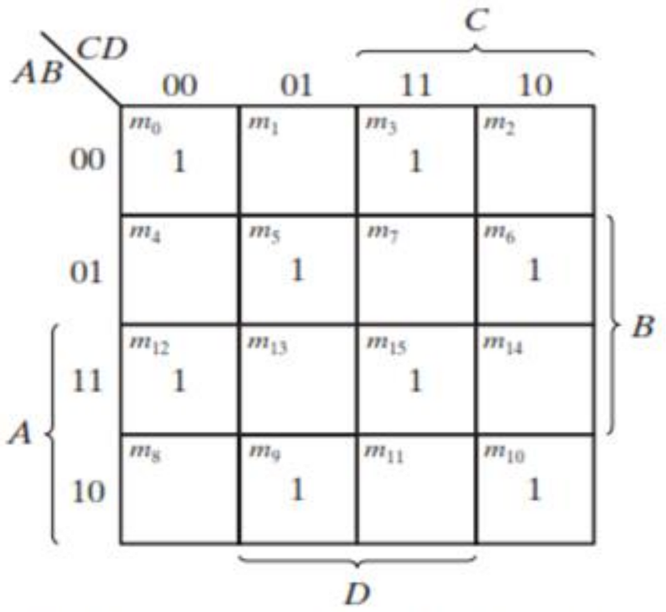
There are 16 minterms for a four-variable Boolean function. Half of the minterms have binary numerical values with an odd number of 1’s; the other half of the minterms have binary numerical values with an even number of 1’s.

- In plotting the function in the map, the binary numerical value for a minterm is determined from the row and column numbers of the square that represents the minterm.

- The map of Fig.(a) is a plot of the 4-variable exclusive-OR function. This is an odd function because the binary values of all the minterms have an odd number of 1's. The complement of an odd function is an even function. As shown in Fig.(b),the 4-variable even function is equal to 1 when an even number of its variables is equal to 1.



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



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