Digital Electronics

Karnaugh Maps
Universal gates

1st Year of 4-year B.Tech

Day 4

Karnaugh Maps

- K-map or Karnaugh is a graphical way of visualizing and then simplifying Boolean expressions
- A K-map is a matrix consisting of rows and columns that represent the output values of a Boolean function.
- The output values placed in each cell are derived from the minterms of a Boolean function.
- A *minterm* is a product term that contains all of the function's variables exactly once, either complemented or not complemented

- For example, the minterms for a function having the inputs *x* and *y* are: $\bar{x}\bar{y}, \bar{x}y, x\bar{y}$, and xy
- Consider the Boolean function, $F(x,y) = xy + x\bar{y}$
- Its minterms are:

Minterm	X	Y
$\bar{X}\bar{Y}$	0	0
$\bar{X}Y$	0	1
$X\bar{Y}$	1	0
XY	1	1

- Similarly, a function having three inputs, has the minterms that are shown in this diagram

Minterm	X	Y	Z
$\bar{X}\bar{Y}\bar{Z}$	0	0	0
$\bar{X}\bar{Y}Z$	0	0	1
$\bar{X}Y\bar{Z}$	0	1	0
$\bar{X}YZ$	0	1	1
$X\bar{Y}\bar{Z}$	1	0	0
$X\bar{Y}Z$	1	0	1
$XY\bar{Z}$	1	1	0
XYZ	1	1	1

- A Kmap has a cell for each minterm
- This means that it has a cell for each line for the truth table of a function
- The truth table for the function *F(x,y) = xy* is shown at the right along with its corresponding K-map

F(X,Y) = XY

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X \ Y	0	1
0	0	0
1	0	1

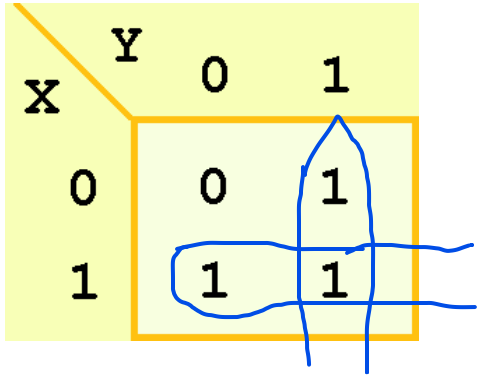
- As another example, we give the truth table and K-Map for the function, *F(x,y) = x + y* at the right

- This function is equivalent to the OR of all of the minterms that have a value of 1. Thus:

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

F(X,Y) = X+Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

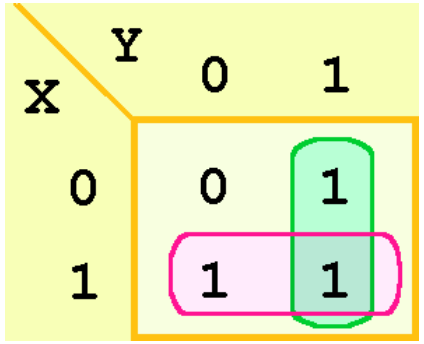


- The minterm function that we derived from our K-map was not in simplest terms

- We can, however, reduce our complicated expression to its simplest terms by finding adjacent 1s in the K-map that can be collected into groups that are powers of two

- In our example, we have two such groups.

- The best way of selecting two groups of 1s form our simple K-map is shown below
- We see that both groups are powers of two and that the groups overlap

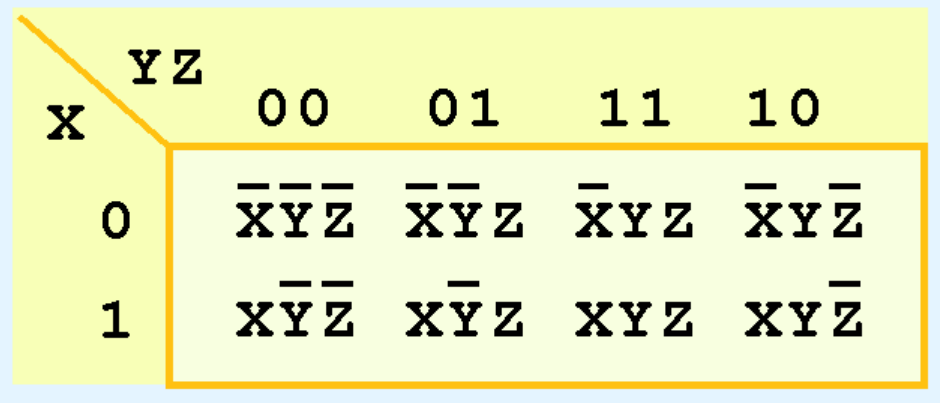


The rules of K-map simplification are:

- Groupings can contain only 1s; no 0s
- Groups can be formed only at right angles; diagonal groups are not allowed
- The number of 1s in a group must be a power of 2 – even if it contains a single 1
- The groups must be made as large as possible
- Groups can overlap and wrap around the sides of the K-map.

2^0 2^1 2^2 2^3

- A K-map for three variables is constructed as shown in the diagram below
- We have placed each minterm in the cell that will hold its value
  - Notice that the values for the *yz* combination at the top of the matrix form a pattern that is not a normal binary sequence.



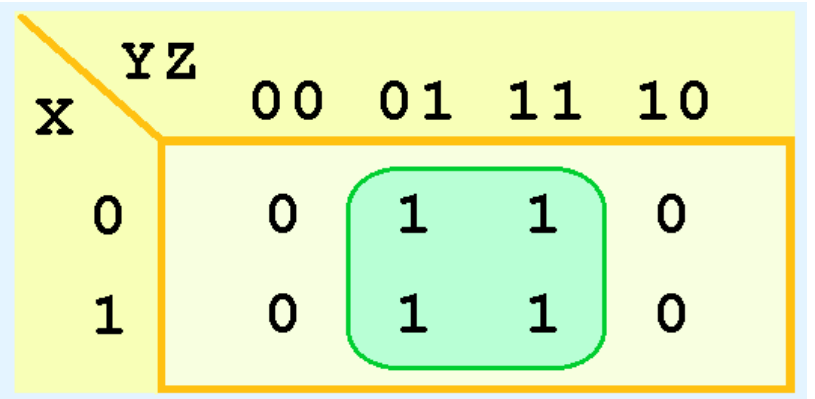
- Thus, the first row of the K-map contains all minterms where *x* has a value of zero

- The first column contains all minterms where *y* and *z* both have a value of zero

- Consider the function: $F(X,Y) = \bar{X}\bar{Y}Z + \bar{X}YZ + X\bar{Y}Z + XYZ$

- Its K-map is given below

X \ YZ	00	01	11	10
0	0	1	1	0
1	0	1	1	0

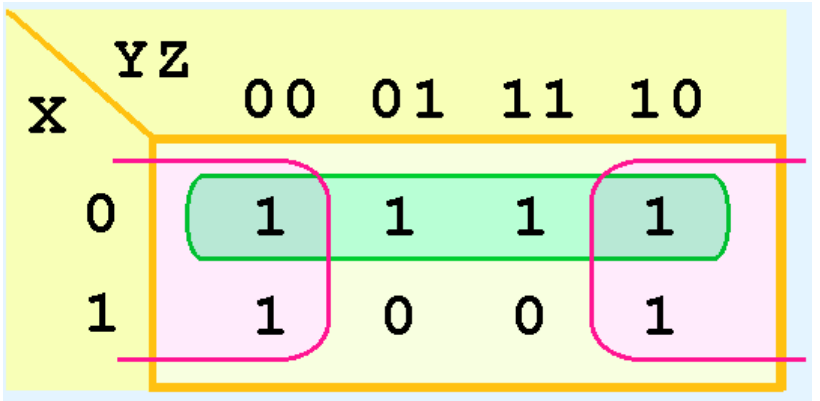


- This means that the function, reduces to *F(x) = Z*

- Now for a more complicated K-map
- Consider the function: $F(X,Y,Z) = \bar{X}\bar{Y}\bar{Z} + \bar{X}\bar{Y}Z + \bar{X}YZ + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XY\bar{Z}$

- Its K-map is shown below. There are (only) two groupings of 1s

X \ YZ	00	01	11	10
0	1	1	1	1
1	1	0	0	1



- In this K-map, we see an example of a group that wraps around the sides of a Kmap

- Our reduced function is: $F(X,Y,Z) = \bar{X} + \bar{Z}$

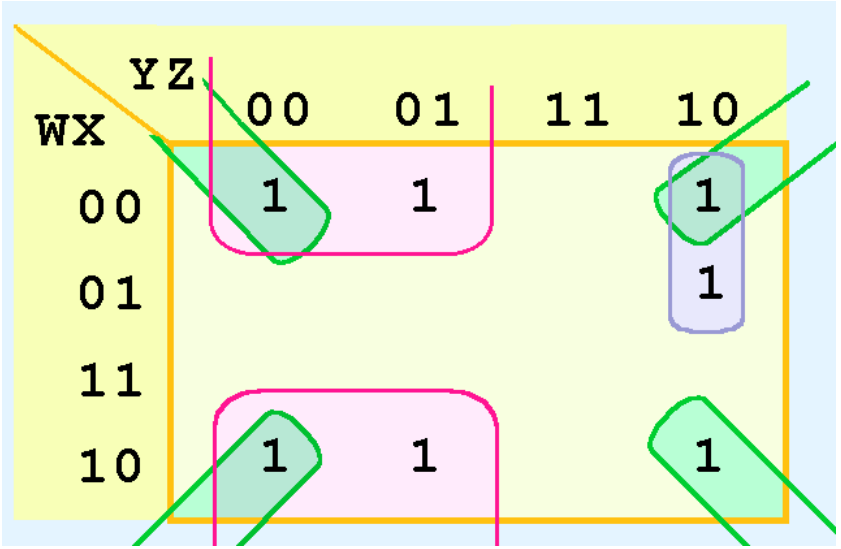
- Our model can be extended to accommodate the 16 minterms that are produced by a four-input function
- This is the format for a 16-minterm K-map

WX \ YZ	00	01	11	10
00	$\bar{w}\bar{x}\bar{y}\bar{z}$	$\bar{w}\bar{x}\bar{y}z$	$\bar{w}\bar{x}yz$	$\bar{w}\bar{x}y\bar{z}$
01	$\bar{w}x\bar{y}\bar{z}$	$\bar{w}x\bar{y}z$	$\bar{w}xyz$	$\bar{w}xy\bar{z}$
11	$wx\bar{y}\bar{z}$	$wx\bar{y}z$	$wxyz$	$wxy\bar{z}$
10	$w\bar{x}\bar{y}\bar{z}$	$w\bar{x}\bar{y}z$	$w\bar{x}yz$	$w\bar{x}y\bar{z}$

- We have populated the K-map shown below with the nonzero minterms from the function:

WX \ YZ	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

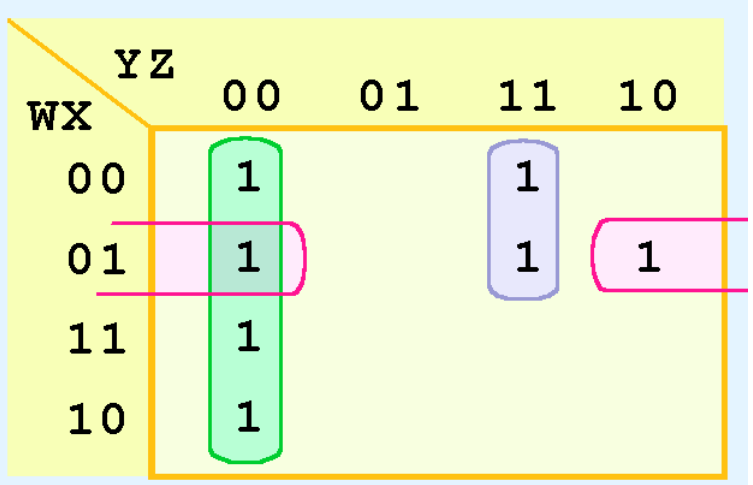
- Our three groups consist of:
  - A purple group entirely within the K-map at the right = $\overline{W}Y\overline{Z}$
  - A pink group that wraps the top and bottom = $\overline{X}\,\overline{Y}$
  - A green group that spans the corners = $\overline{X}\,\overline{Z}$

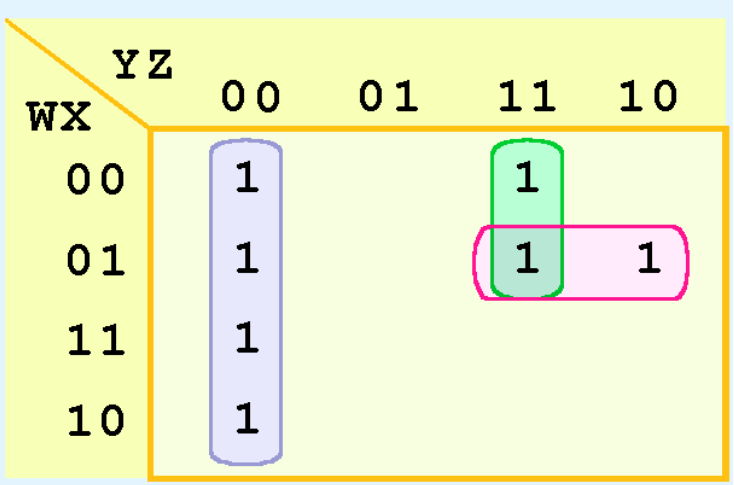


- Thus we have three terms in our final function:

$$F(W,X,Y,Z) = \overline{X}\overline{Y} + \overline{X}\overline{Z} + \overline{W}Y\overline{Z}$$

- It is possible to have a choice as to how to pick groups within a K-map, while keeping the groups as large as possible
- The (different) functions that result from the groupings below are logically equivalent





Don’t Care Conditions

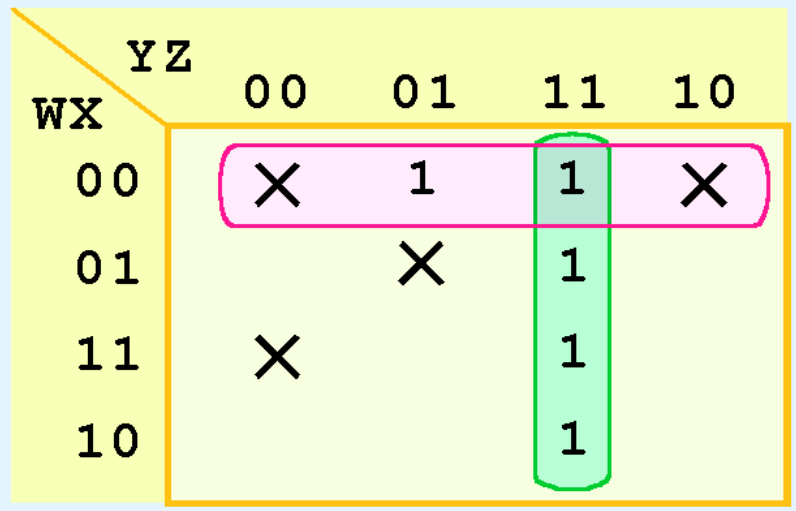
- Real circuits don’t always need to have an output defined for every possible input
  - For example, some calculator displays consist of 7-segment LEDs. These LEDs can display 2^7 -1 patterns, but only ten of them are useful
- If a circuit is designed so that a particular set of inputs can never happen, we call this set of inputs a *don’t care* condition
- They are very helpful to us in K-map circuit simplification
- In a K-map, a don’t care condition is identified by an *X* in the cell of the minterm(s) for the don’t care inputs, as shown below
- In performing the simplification, we are free to include or ignore the *X*’s when creating our groups

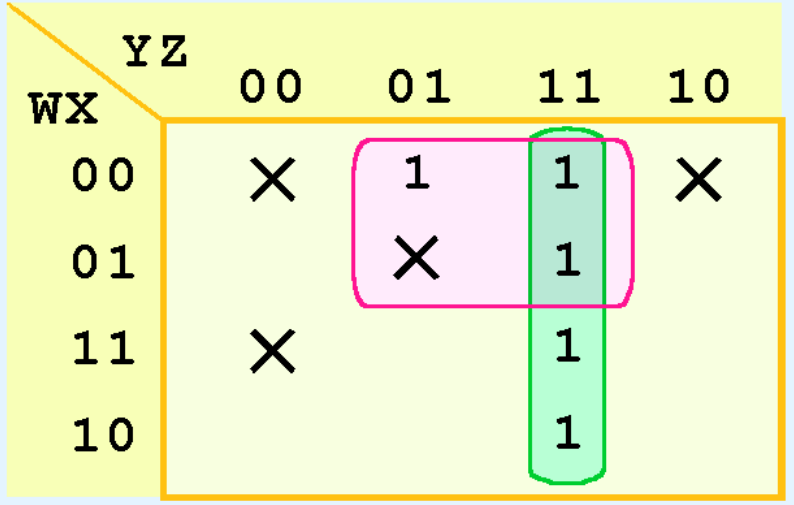
WX \ YZ	00	01	11	10
00	X	1	1	X
01		X	1	
11	X		1	
10			1	

- In one grouping in the K-map below, we have the function
- However, different grouping gives us different functions

$$F(W,X,Y,Z) = \bar{W}\bar{Y} + YZ$$

$$F(W,X,Y,Z) = \bar{W}Z + YZ$$





The truth table of both the function differ from each other.

However, the values for which they differ, are the inputs for which we have don't care conditions.

Example 2.26 Simplify the following expression using the Karnaugh map for the 4-variables A, B, C and D.

$$Y = m_1 + m_3 + m_5 + m_7 + m_8 + m_9 + m_{12} + m_{13}$$

Solution The K-map of the given equation is shown in Fig. E2.26. The expression is minimized using the following steps:

Step 1 Construct the K-map and enter 1 in the cells corresponding to the minterms present in the expression and 0 in the other cells.

Step 2 There are no 1s which are not adjacent to other 1s.

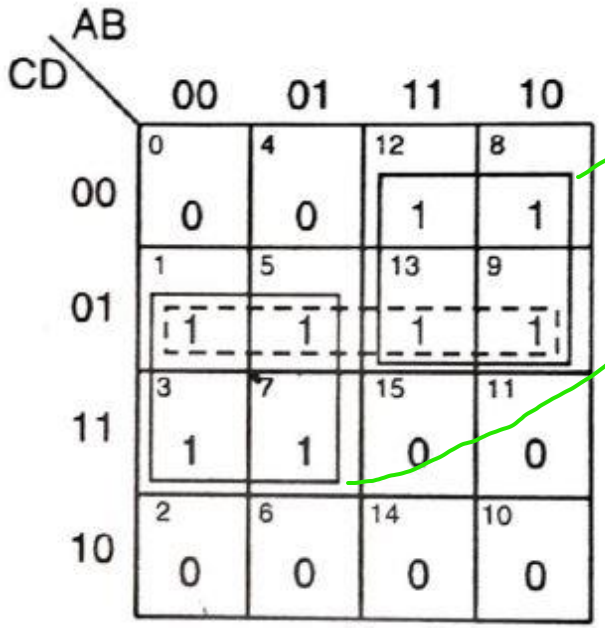
Step 3 There are no pairs which are not part of any larger groups.

Step 4 There are 2 quads. Cells 1, 3, 5 and 7 are grouped to form one quad and the second quad is made up of cells 12, 13, 8 and 9. The combinations corresponding to the cells in the first quad are $\overline{A}\,\overline{B}\,\overline{C}D$, $\overline{A}\,\overline{B}CD$, $\overline{A}B\overline{C}D$ and $\overline{A}BCD$. In the above group of four combinations, the variables $\overline{A}D$ are common in all the cells while B and C appear both in complemented and uncomplemented forms. From the preceding section, it is clear that only the variables that are the same in all the cells of the group must appear in the term corresponding to that group. Therefore, the minimized term for the first quad is $\overline{A}D$, and that of the second quad is $A\overline{C}$.

Step 5 There are no octets.

Step 6 All the 1s have already been grouped.

Step 7 The terms generated by the two groups are OR operated together to obtain the expression for Y as follows:



$A\bar{C} + \bar{A}D + \bar{C}D$

may ignore redundant

Fig. E2.26

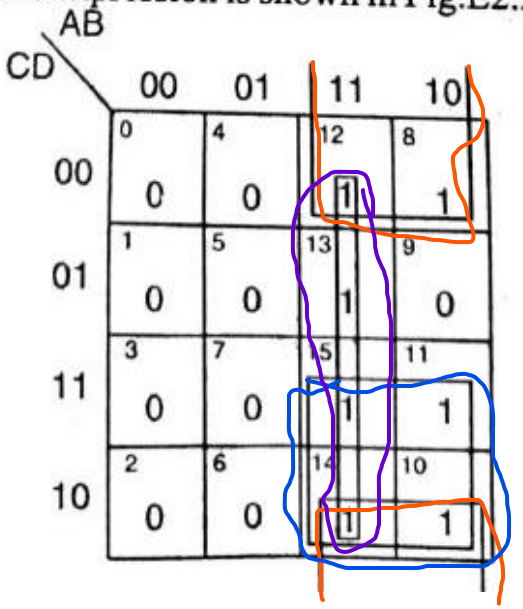
Note: In the above *K*-map, if a third quad is formed as shown by the dotted lines, it results in a redundant expression because the 1s to be covered by the third quad are already covered by quads 1 and 2.

Example 2.27 Plot the logical expression $ABCD + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}C + AB$ on a 4-variable K-map; obtain the simplified expression from the map.

Solution To enter into a K-map, a logic expression must be either in the canonical SOP form or in the canonical POS form. The canonical SOP form of the given expression can be obtained as follows:

$$
\begin{aligned}
Y &= ABCD + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}C + AB \\
&= ABCD + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}C(D+\overline{D}) + AB(C+\overline{C})(D+\overline{D}) \\
&= ABCD + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}CD + A\overline{B}C\overline{D} + (ABC + AB\overline{C})(D+\overline{D}) \\
&= ABCD + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}CD + A\overline{B}C\overline{D} + \cancel{ABCD} + ABC\overline{D} + AB\overline{C}D + AB\overline{C}\,\overline{D} \\
&= ABCD + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}CD + A\overline{B}C\overline{D} + ABC\overline{D} + AB\overline{C}D + AB\overline{C}\,\overline{D} \\
&= m_{15} + m_8 + m_{11} + m_{10} + m_{14} + m_{13} + m_{12} \\
&= \Sigma_m(8,10,11,12,13,14,15)
\end{aligned}
$$

The K-map for the above expression is shown in Fig.E2.27.



In the K map in Fig. E2.27, there are three quads; the minimised terms for them are AB, AC and $A\overline{D}$ and the simplified expression is:

$$Y = AB + AC + A\overline{D}$$

Example 2.28 Simplify the expression $Y = \Sigma_m(7,9,10,11,12,13,14,15)$, using the *K*-map method.

Solution The *K*-map for the above function is shown in Fig. E2.28.

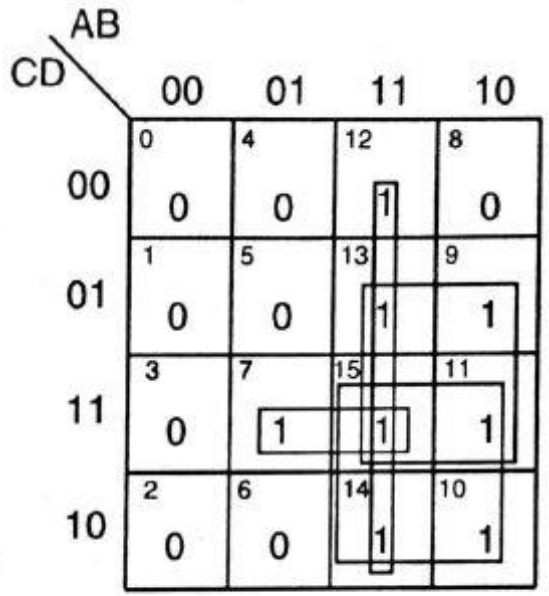


Fig. E2.28

In the given *K*-map, there are three quads and one pair; the corresponding simplified terms are *AB, AD, AC* and *BCD*.

Now, the simplified expression is

$$Y = AB + AD + AC + BCD$$

Since the quads and pair formed in the above *K*-map overlap, the expression can be further simplified using the Boolean algebra as follows:

$$Y = AB + AD + AC + BCD$$
$$= A(B + D + C) + BCD$$

Example 2.29 Simplify the expression $Y = m_1 + m_5 + m_{10} + m_{11} + m_{12} + m_{13} + m_{15}$, using the *K*-map method.

Solution The *K*-map for the above expression is shown in Fig. E2.29(a).

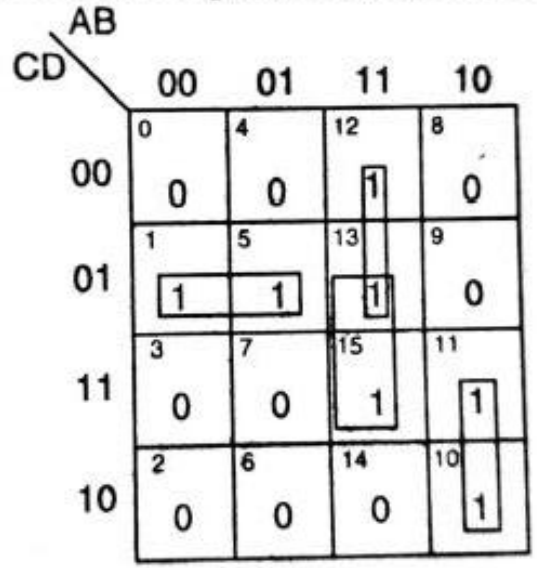


Fig. E2.29(a)

As shown in Fig. E2.29(a), the *K*-map contains four pairs but no quads or octets; the corresponding simplified expression is given by

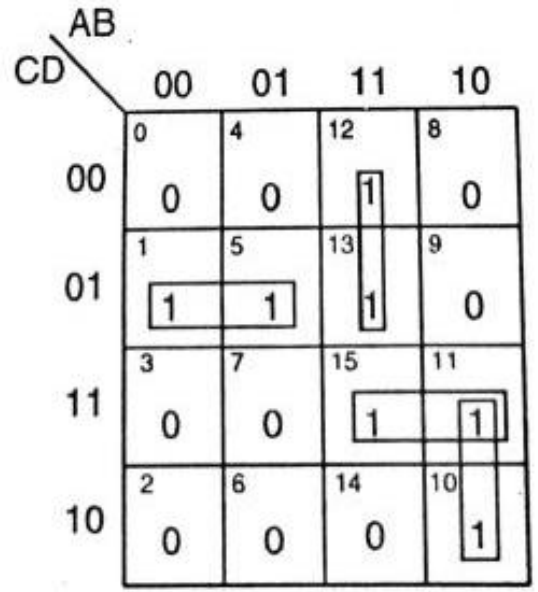
$$Y = \overline{A}\,\overline{C}D + AB\overline{C} + ABD + A\overline{B}C \quad (1)$$

It is important to note that the simplified expression obtained from the *K*-map is not unique. This can be explained by grouping the pairs in a different manner as shown in Fig. E2.29(b).

From the *K*-map shown in Fig. E2.29(b), the simplified expression can be written as:

$$Y = \overline{A}\,\overline{C}D + AB\overline{C} + ACD + A\overline{B}C \quad (2)$$

In equations (1) and (2) , the third term is different, due to the different groupings done in Fig. E2.29(b). Though the simplified expression for any given function is not unique, both the above expressions are logically equivalent. Two expressions are said to be logically equivalent if and only if both the expressions have the same value for every combination of input variables.



• Logic Gates Using Diodes

- In digital logic gate one that has a "LOW" level logic "0" is of 0 volts (ground) and a "HIGH" level logic "1" is of +5 volts
- Hence,

  0-0.2V is termed as '0' state or 'OFF' state

  3.8V – 5V is termed as '1' state or 'ON' state

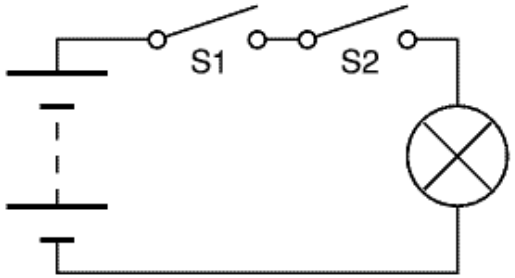
- Digital Logic Gates can be made from discrete components such as Resistors, Transistors and Diodes to form RTL (resistor-transistor logic) or DTL (diode-transistor logic) circuits, but today's modern digital 74xxx series integrated circuits are manufactured using TTL (transistor-transistor logic) based on NPN bipolar transistor technology or the much faster and low power CMOS based MOSFET transistor logic used in the 74Cxxx, 74HCxxx, 74ACxxx and the 4000 series logic chips

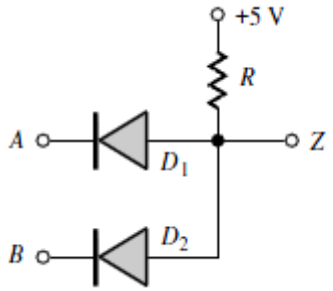
Pull-up and Pull-down Resistors

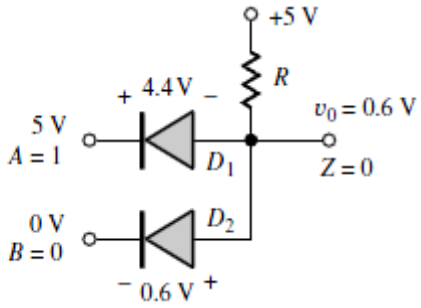
- When connecting together digital logic gates to produce logic circuits, any "unused" inputs to the gates must be connected directly to either a logic level "1" or a logic level "0" by means of a suitable "Pull-up" or "Pull-down" resistor (for example 1kΩ resistor) to produce a fixed logic signal. This will prevent the unused input to the gate from "floating" about and producing false switching of the gate and circuit.

Logic Gates: AND

The simples gates are AND and OR. They can be built from switches or using the simplest form of electronic logic - diode logic.







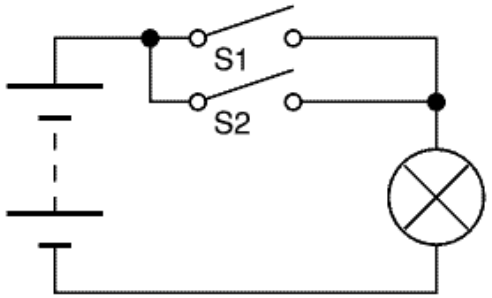
A = 0 , B = 0 → both diodes are forward biased → both diodes conduct →out is LOW → 0.

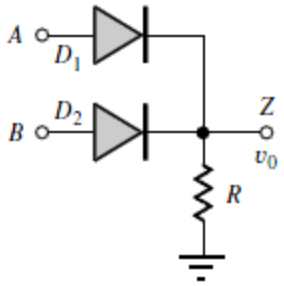
A = 0 , B = 1 → DB is reverse biased → does not conduct,
DA is forward biased → conducts →out is LOW → 0.

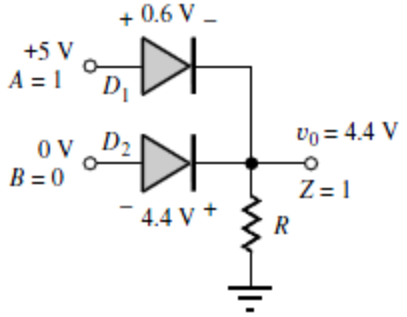
A = 1 , B = 0 → DA is reverse biased → does not conduct,
DB is forward biased → conducts →out is LOW → 0.

A = 1 , B = 1 → both diodes are reverse biased →
both the diodes do not conduct → out is HIGH → 1.

Logic Gates: OR







A = 0 , B = 0 → both diodes are off → does not conduct →out is LOW → 0.

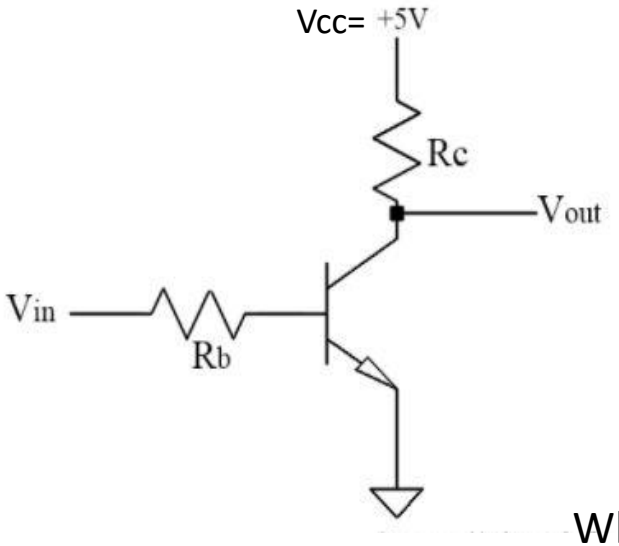
A = 0 , B = 1 → DA is off → does not conduct,
DB is forward biased → conducts →out is HIGH → 1.

A = 1 , B = 0 → DB is off → does not conduct,
DA is forward biased → conducts →out is HIGH → 1.

A = 1 , B = 1 → both diodes are forward biased→
both the diodes conduct → out is HIGH → 1.

Inverter - circuit

NOT gate It is also known as an inverter because the output is opposite to the input. It has one input and one output. Circuit diagram of NOT gate using transistor is shown below:



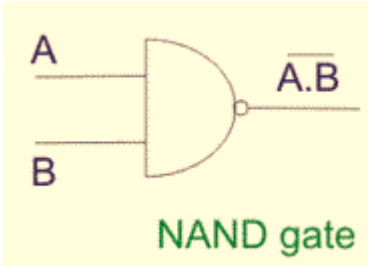
When the input V_{in} is 'LOW', the BE junction does not get the required V_{BE} to switch on the transistor. Transistor is in OFF state .Hence, output $V_{out} = V_{cc}$(HIGH)

When the input V_{in} is 'HIGH', the transistor is 'ON' and it conducts. Hence, $V_{out} = V_{CE(sat)}$ = 'LOW'

What are Universal Gates?

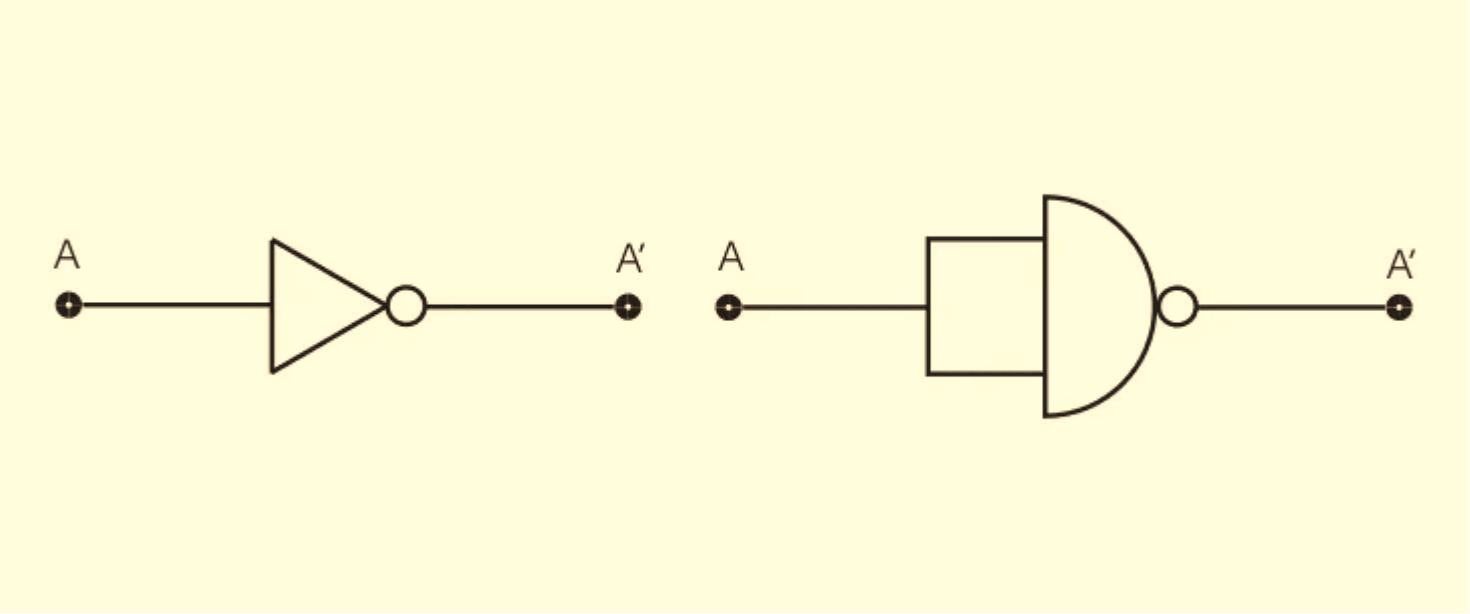
- A **universal gate** is a logic gate which can implement any Boolean function without the need to use any other type of logic gate
- The NOR gate and NAND gate are universal gates
- This means that you can create any logical Boolean expression using only NOR gates or only NAND gates
- Other logical gates – such as AND gates, NOT gates and OR gates – do not have this property of universality

- NAND Gate: During the operation of the NAND gate, the inputs are first going through AND gate and after that, the output gets reversed, and we get the final output

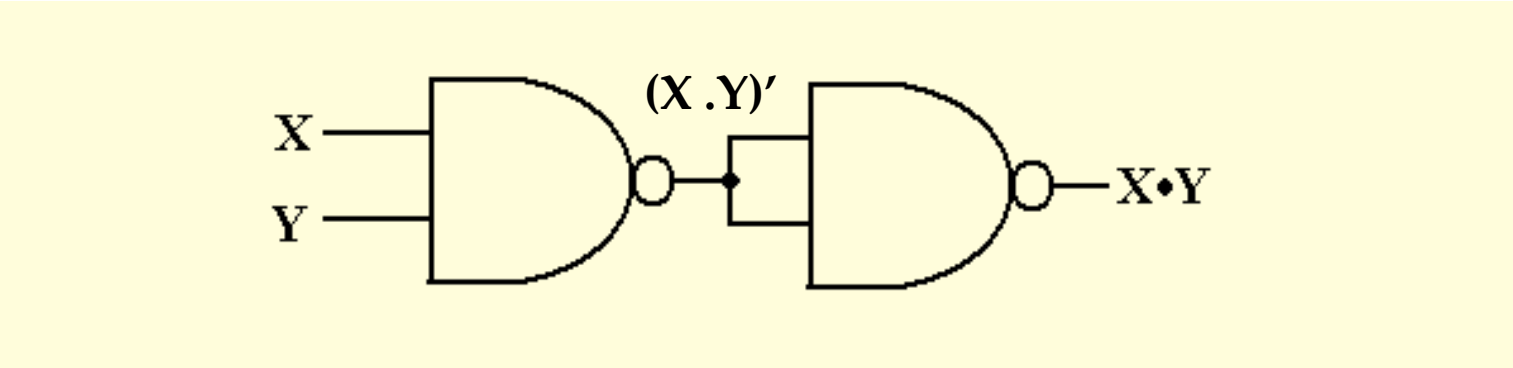


1. NAND gate as NOT gate:

This is the circuit diagram of a NAND gate used to make work like a NOT gate, the original logic gate diagram of NOT gate is given besides the circuit diagram below.

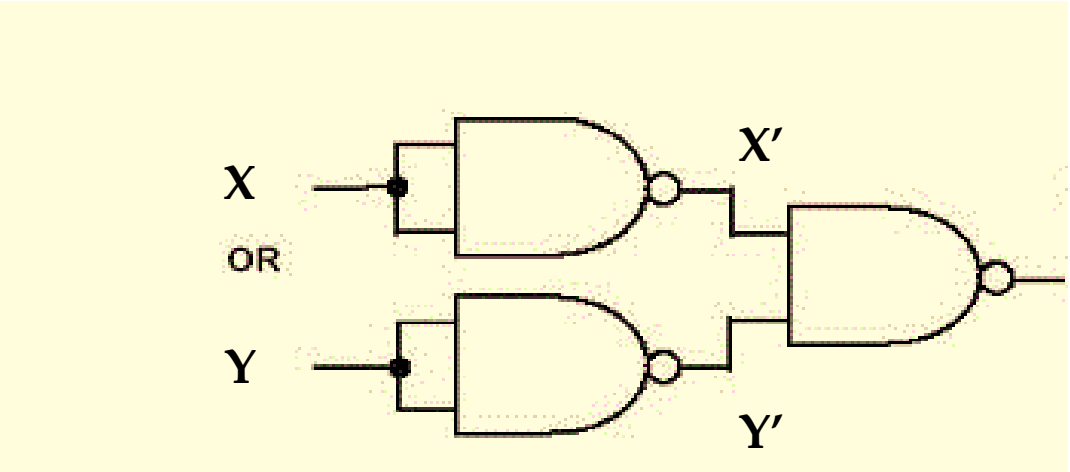


- NAND gate as AND gate:



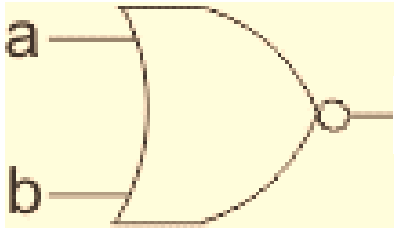
The above diagram is of an AND gate made from NAND gate

3 NAND gate as OR gate:



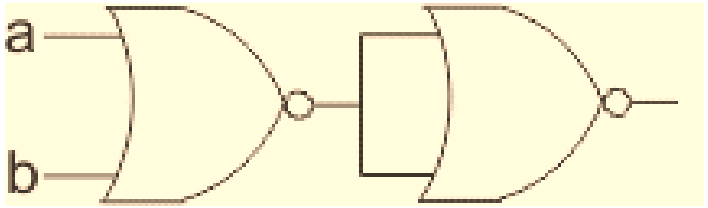
$$(X' \cdot Y')' = X'' + Y'' = X + Y$$

NOR gate: During the operation of the NOR gate, the inputs are first going through an OR gate and after that, the output gets reversed, and we get the final output



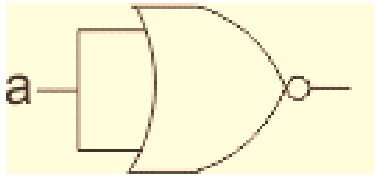
$\overline{a + b}$

- NOR gate as OR gate:



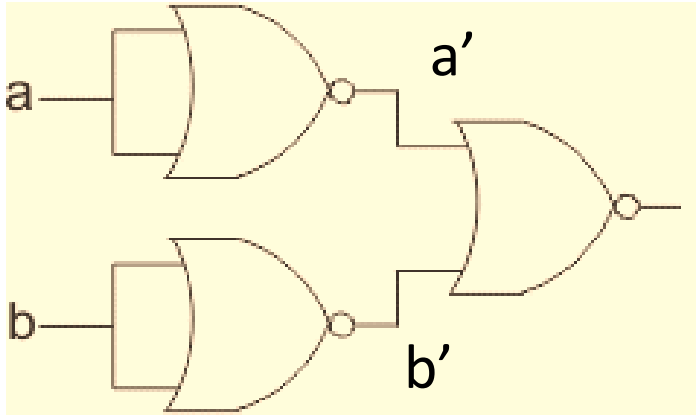
a + b

2. NOR gate as NOT gate



a’

3. NOR gate as AND gate

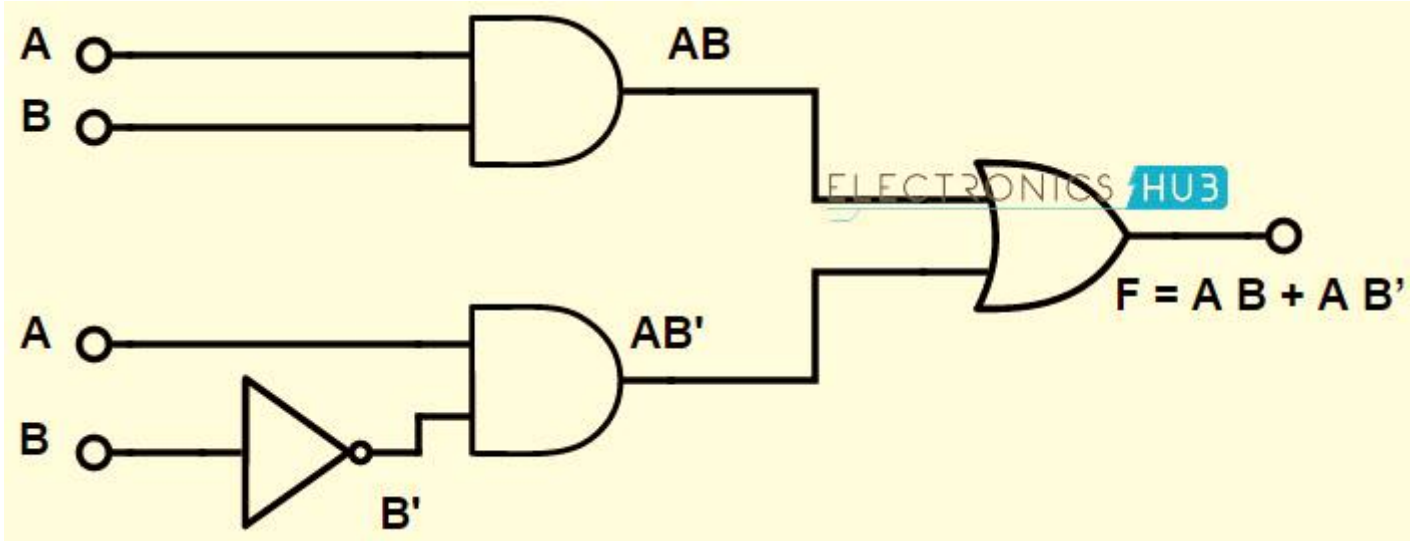


a. b

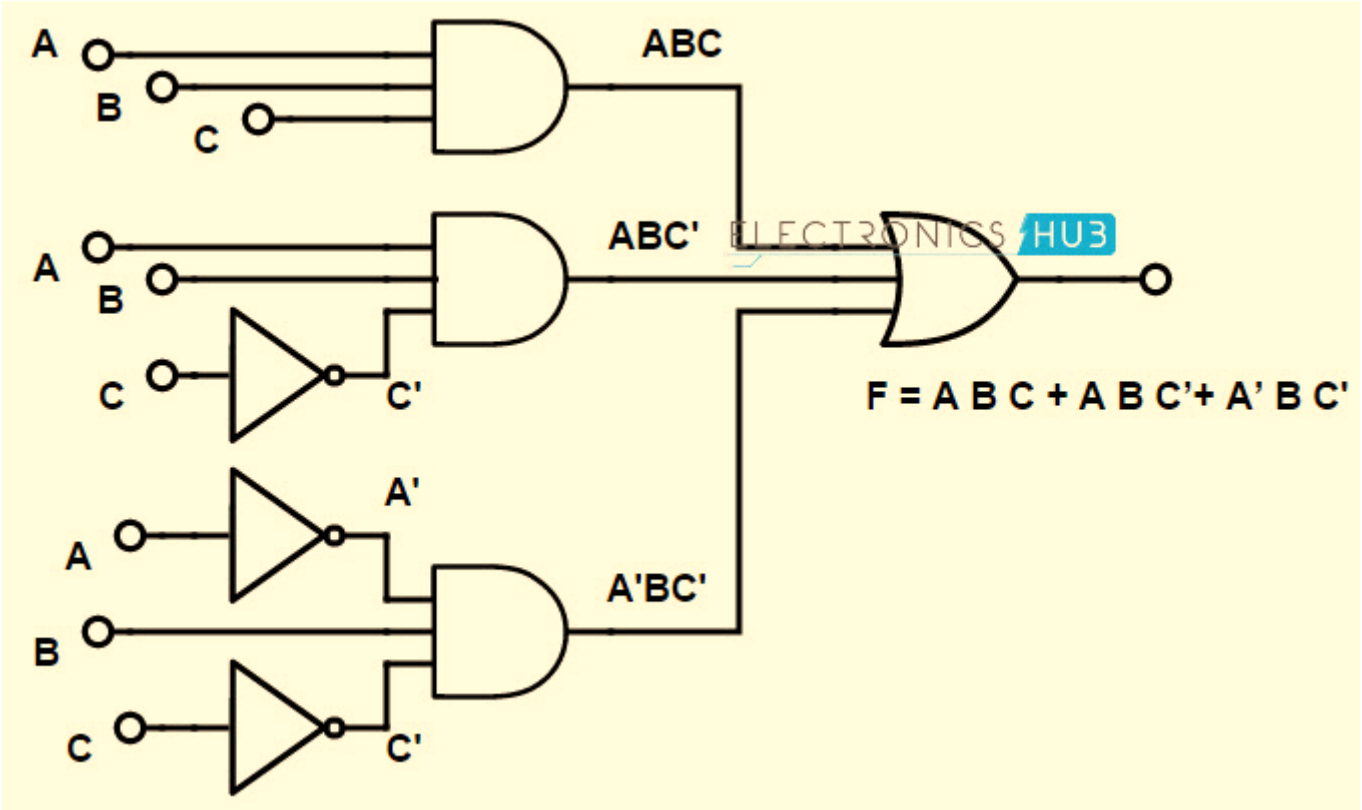
(a’ + b’)’ = a’’. b’’ = a.b

Implementing a Boolean function using logic gates

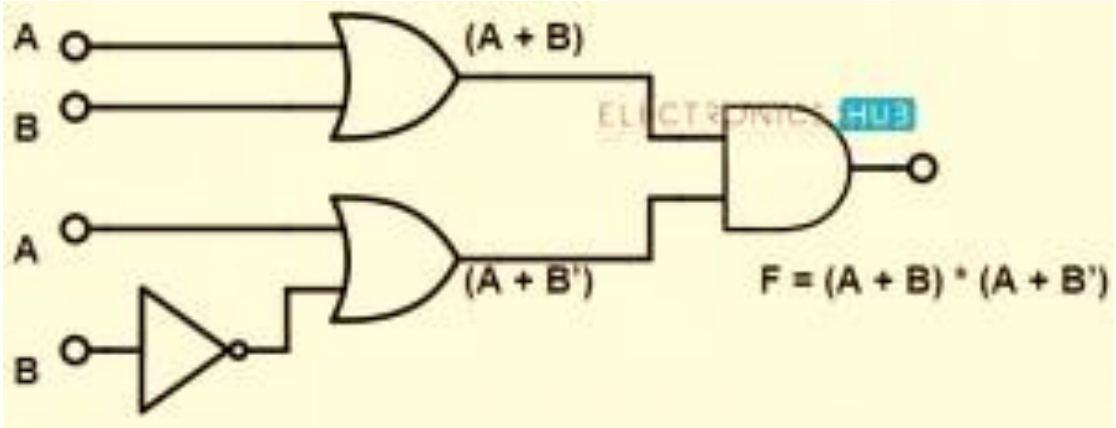
- $F = AB + A\overline{B}$



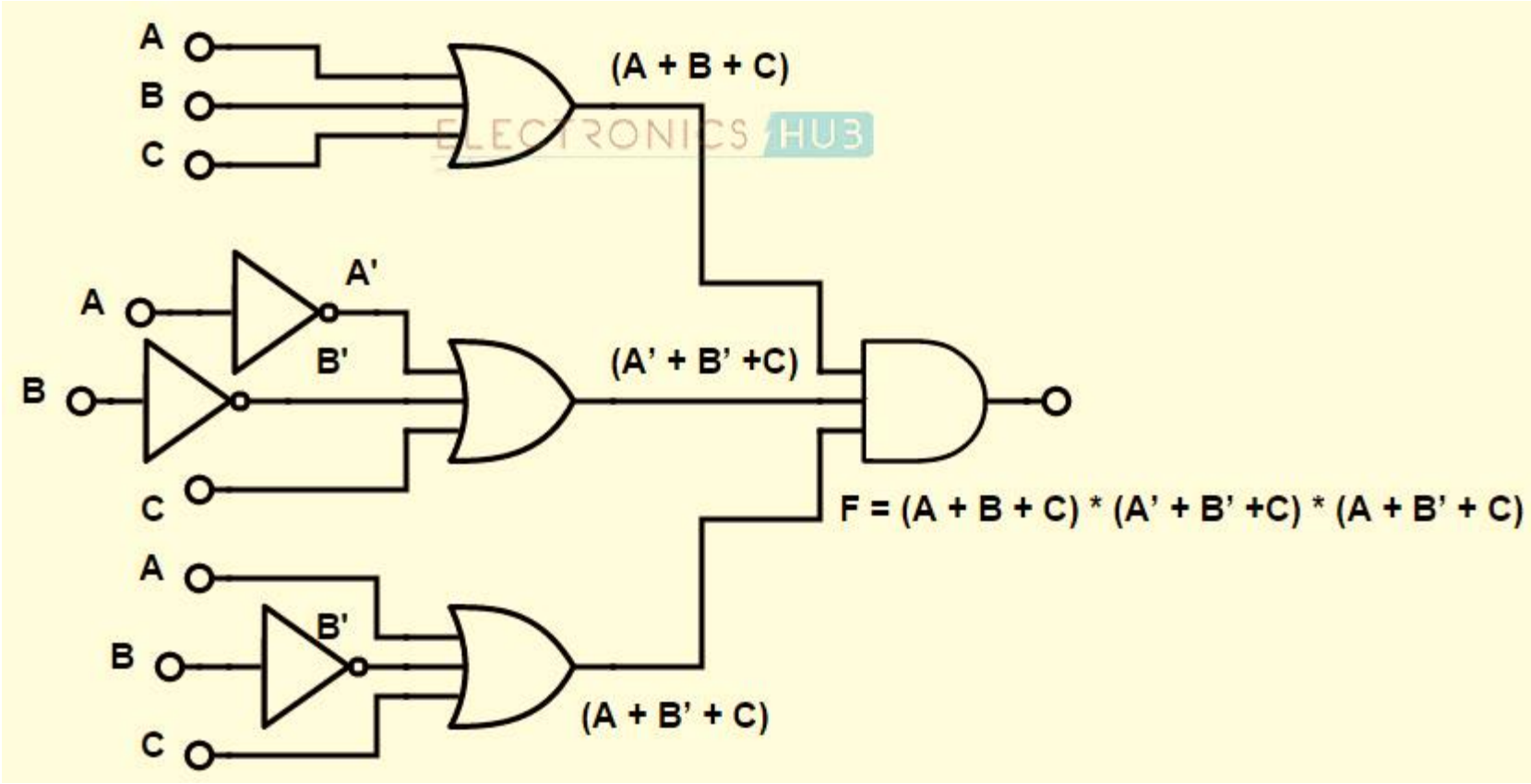
- $F = A\ B\ C + A\ B\ \overline{C} + \overline{A}\ B\ \overline{C}$



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

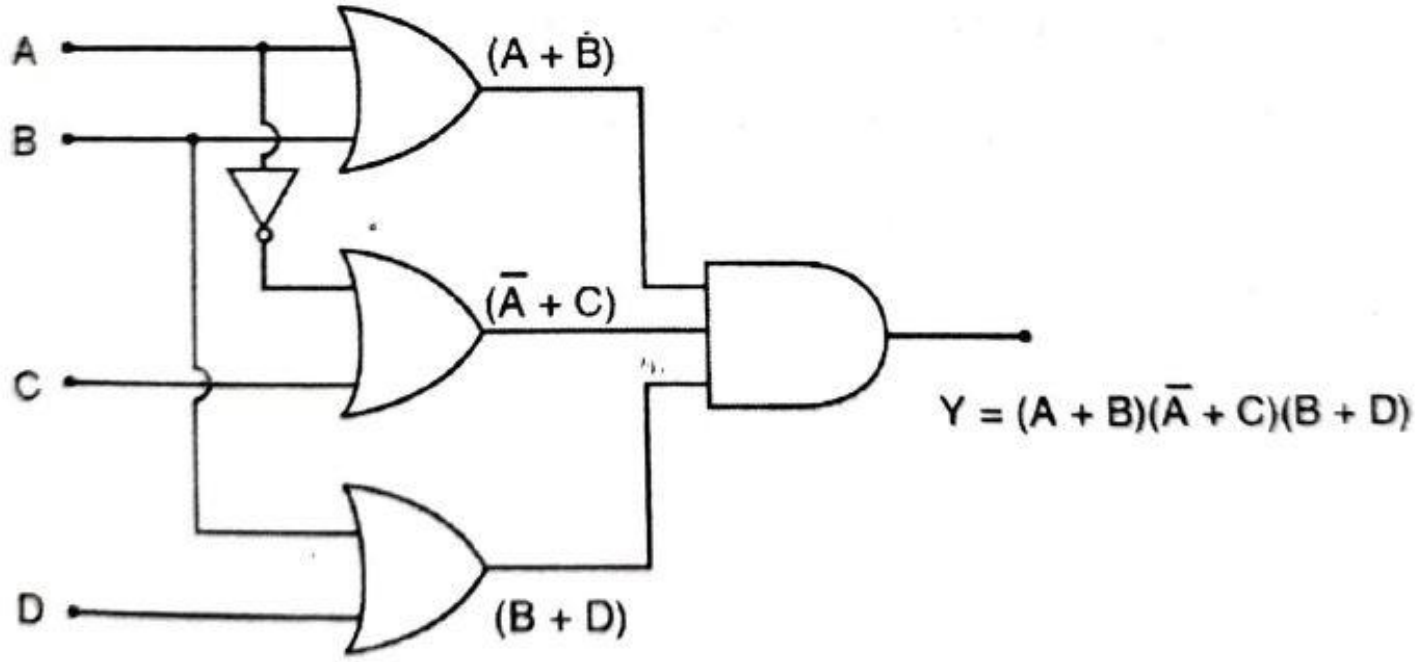


- $F = (A + B + C) \cdot (\overline{A} + \overline{B} + C) \cdot (A + \overline{B} + C)$



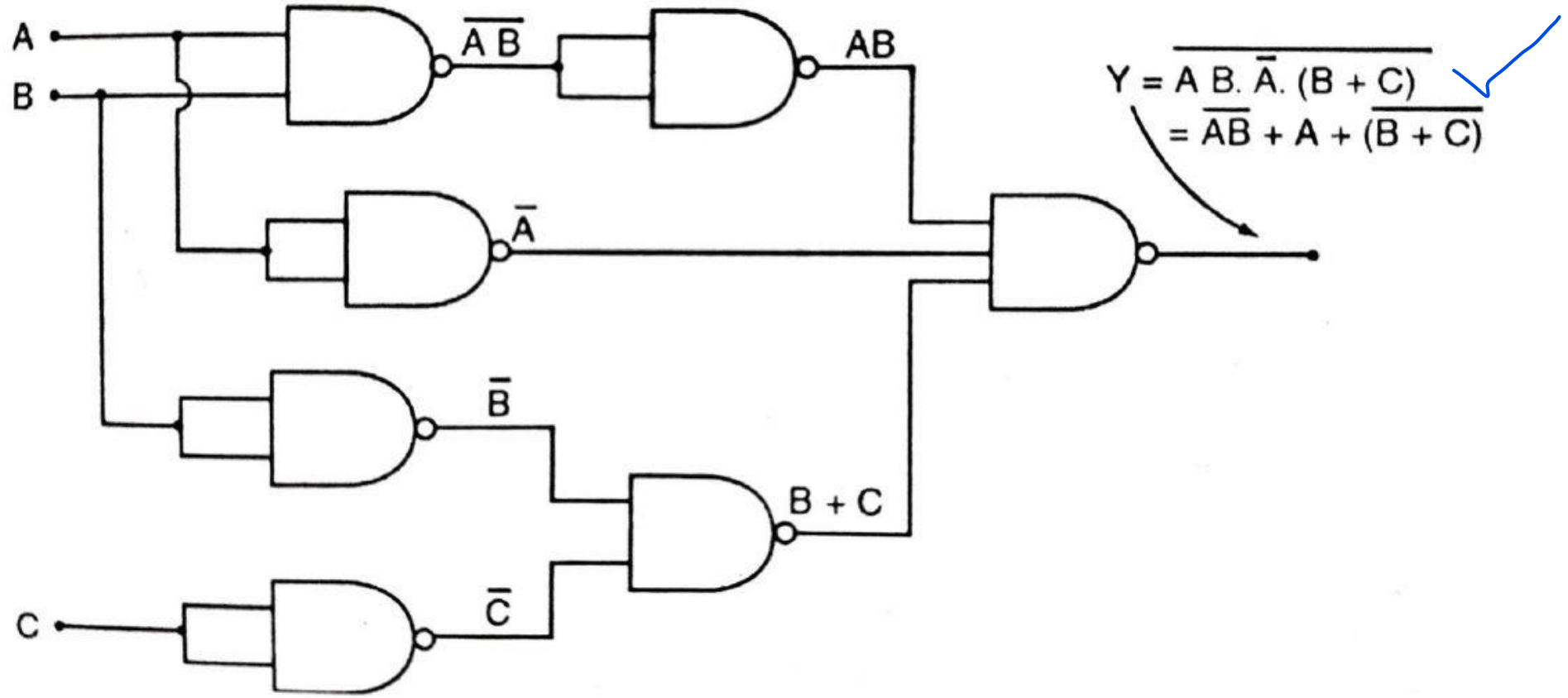
Example 3.2 Realise the logic expression $Y = (A + B)(\bar{A} + C)(B + D)$ using basic gates.

Solution In the given expression, there are 3 sum terms which can be implemented using three 2-input OR gates and their outputs are AND operated together by a 3-input AND gate. A NOT gate can be used to obtain the inverse of A. Now, the realised circuit is shown in Fig. E3.2.



Example 3.3 Implement $Y = \overline{AB} + A + (\overline{B+C})$ using NAND gates only.

Solution The implementation of the given function is shown in Fig. E3.3.



Example 3.4 Simplify the logic circuit shown in Fig. E3.4(a).

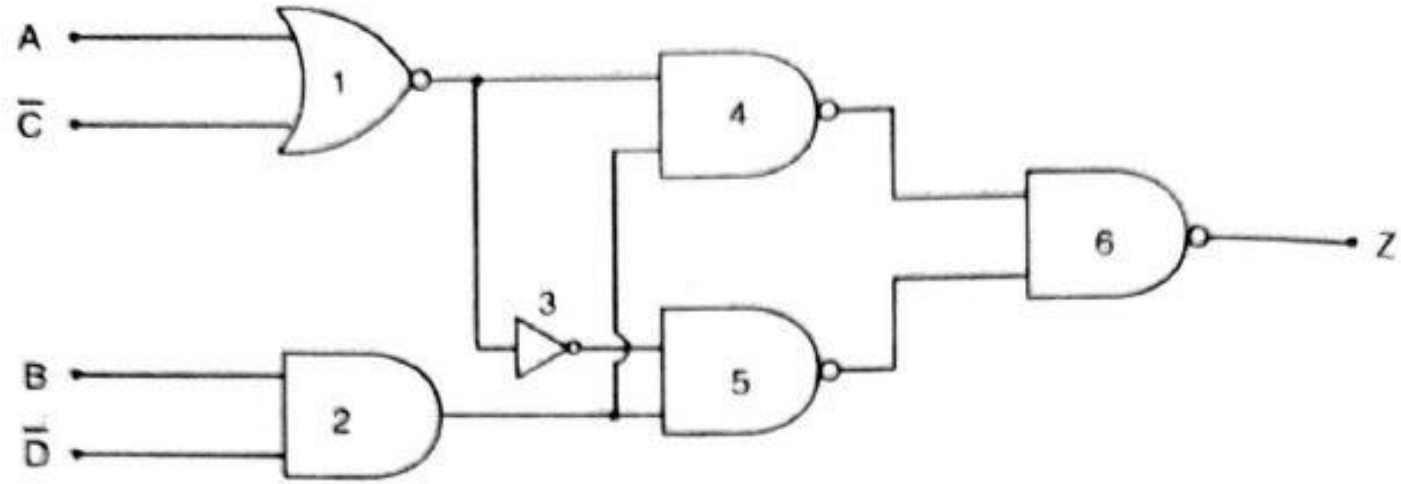


Fig. E3.4 (a)

Solution From the given logic circuit, the expression for Z can be written as

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

$$= \overline{[\overline{\overline{(A+\overline{C})}} + \overline{B\overline{D}}]\cdot[\overline{\overline{(A+\overline{C})}} + \overline{B\overline{D}}]}$$

$$= \overline{[(A+\overline{C}) + \overline{B\overline{D}}]\cdot[\overline{(A+\overline{C})} + \overline{B\overline{D}}]}$$

$$= \overline{B\overline{D}} + (A+\overline{C})\overline{(A+\overline{C})} \qquad [\because (A+B)(A+C) = A+BC]$$

$$= \overline{B\overline{D}} \qquad [\because A\overline{A} = 0]$$

$$= B\overline{D}$$

Therefore, the above logic circuit can be simplified as shown in Fig. E3.4(b).

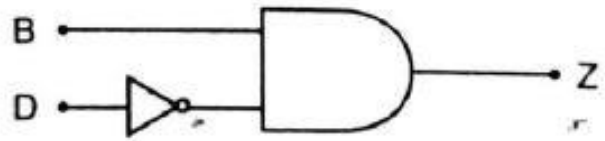


Fig. E3.4 (b)

Instead of using Boolean algebra, the logic circuit can be simplified directly as shown below. In the given logic circuit shown in Fig. E3.4 (a), the NAND gate (6) can be replaced by an OR gate with a bubble at its inputs as shown in Fig.E3.4(c).

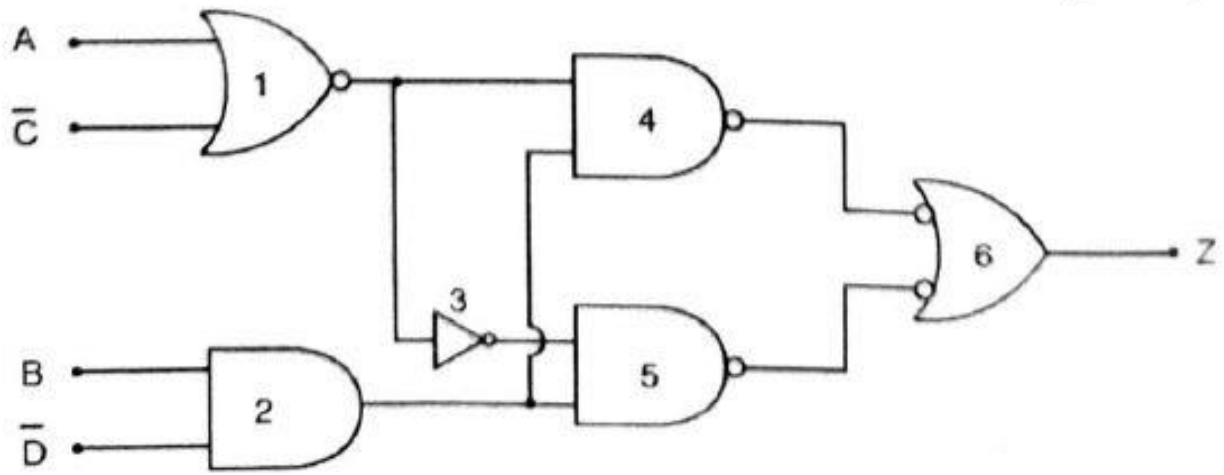


Fig. E3.4 (c)

Now, using $\overline{\overline{A}} = A$, the bubble at the outputs of gates 4 and 5 get cancelled with the bubble at the inputs of gate 6 as shown in Fig. E3.4(d).

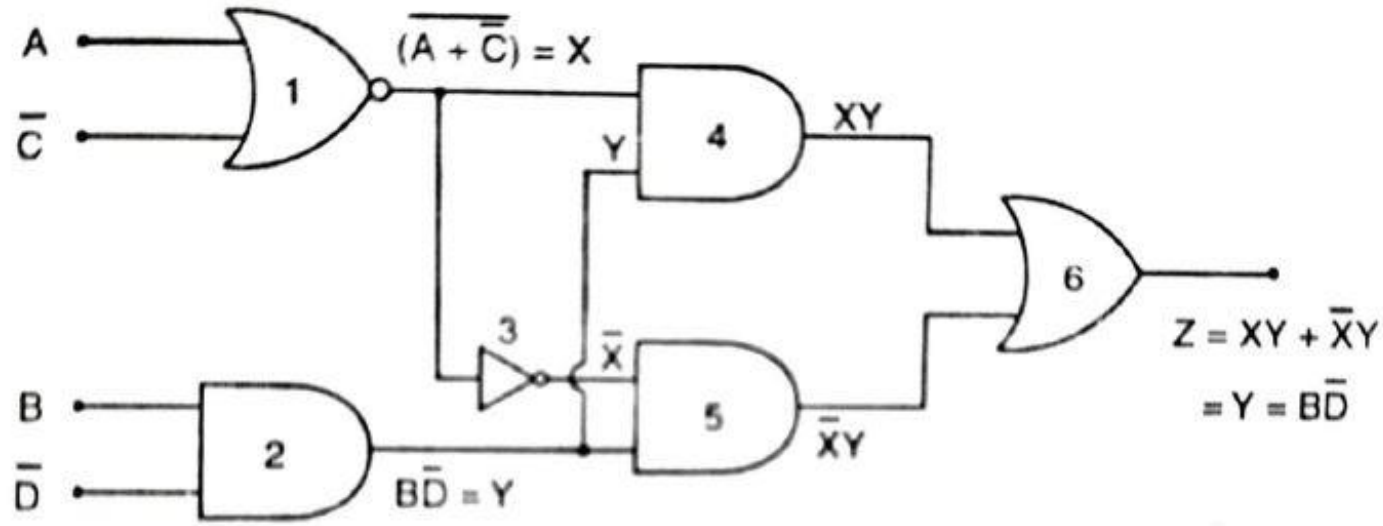
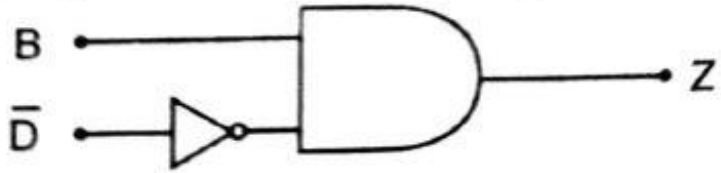


Fig. E3.4(d)

In the above figure, if we assume the output of gate 1 $\overline{(A+\overline{C})} = X$ and the output of gate 2 $B\overline{D} = Y$, then the output of gate 4 is XY and the output of gate 5 is $\overline{X}Y$. If XY and $\overline{X}Y$ are OR operated in gate 6, then $XY + \overline{X}Y = Y(X + \overline{X}) = Y = B\overline{D}$. Therefore, the above circuit can be simplified as shown in Fig. E3.4(e).



Example 3.5 Realise (a) $Y = A + BC\overline{D}$ using NAND gates and

(b) $Y = (A+C)(A+\overline{D})(A+B+\overline{C})$ using NOR gates.

Solution Using NAND gates

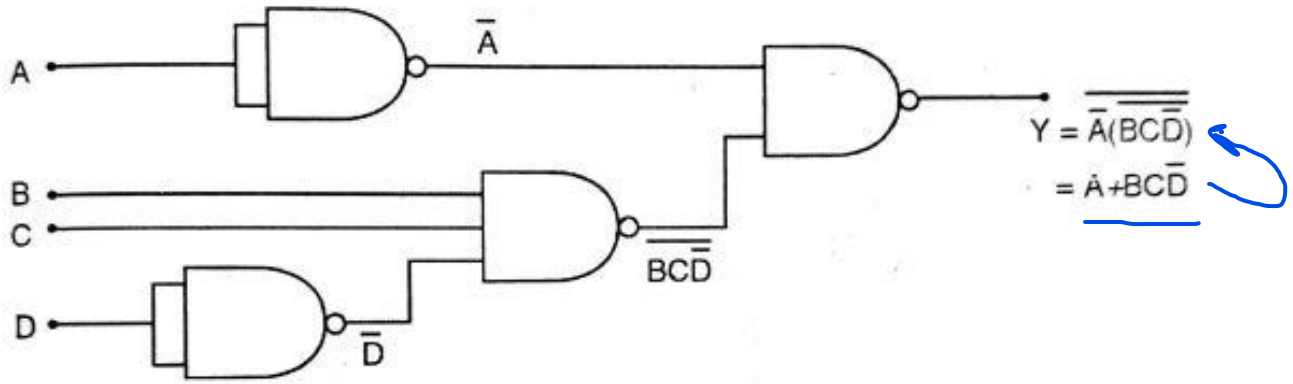


Fig. E3.5(a)

(b) Using NOR gates