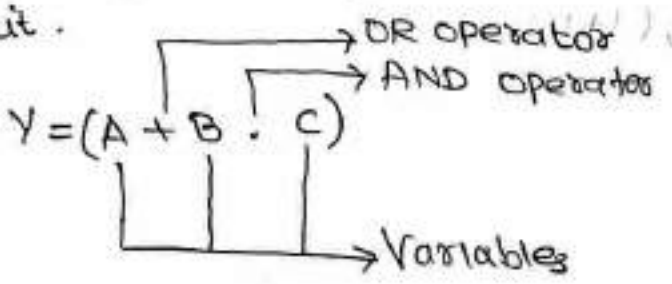
Unit-2

Combinational Circuits

Boolean Algebra – A system of mathematical logic which can be used to simplify the design of logic circuit.



Laws in Boolean Algebra

1. Commutative law ⟶ $A+B=B+A$, $A \cdot B = B \cdot A$
2. Associative law ⟶ $A+(B+C)=(A+B)+C$, $(AB)C = A(BC)$
3. Distributive law ↓ $A(B+C) = AB+AC$

Demorgan's Theorem – important part of Boolean algebra.

(i) $\overline{AB} = \bar{A}+\bar{B}$

(ii) $\overline{A+B} = \bar{A}\,\bar{B}$

Consensus Theorem

$$AB + \bar{A}C + BC = AB + \bar{A}C$$

↓

Redundant term

* Redundant term can be eliminated to form the equivalent expression $AB+\bar{A}C$

Proof: $AB+\bar{A}C+BC$

$= AB+\bar{A}C+(A+\bar{A})BC$

$= AB+\bar{A}C+ABC+\bar{A}BC$

$= AB+ABC+\bar{A}C+\bar{A}BC$

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

$= AB+\bar{A}C$

$\left[\begin{array}{l}1+C=1\\1+B=1\end{array}\right.$

Principle of Duality

1. Changing each OR sign to an AND sign
2. Changing each AND sign to an OR sign
3. Complementing any 0 or 1 appearing in expression

$\left.\begin{array}{l}A+\bar{A}=1\\A.\bar{A}=0\end{array}\right]$ example for principle of duality expression

$\left.\begin{array}{ll}A+0=A & A.0=0\\A+1=1 & A.1=A\\A+A=A & A.A=A\end{array}\right]$ another examples

Minimization of Boolean Expression

Rule 1: $0+A=A$
$A+0=A$

Rule 2: $1+A=1$
$A+1=1$

Rule 3: $A+A=A$

Rule 4: $A+\bar{A}=1$

Rule 5: $0 \cdot A = 0$
$A \cdot 0 = 0$

Rule 6: $1 \cdot A = A$
$A \cdot 1 = A$

Rule 7: $A \cdot A = A$

Rule 8: $A \cdot \bar{A} = 0$

Rule 9: $\bar{\bar{A}} = A$

Rule 10: $A + AB = A$

$LHS = A(1+B)$
$= A.$
$LHS = RHS$

Rule 11: $A + \bar{A}B = A + B$

$LHS = A + \bar{A}B$
$= A + AB + \bar{A}B$ (apply Rule 10)
$= A + B(A + \bar{A})$ (apply Rule 4)

$RHS = A + B$

Rule 12: $(A+B)(A+C) = ABC$

$LHS = (A+B)(A+C)$
$= AA + AC + AB + BC$ (apply rule 7)
$= A + AC + AB + BC$
$= A(1+C) + AB + BC$ (apply rule 2)
$= A + AB + BC$
$= A(1+B) + BC$
$RHS = A + BC$

Simplify the Boolean expression

(a) $ABCD + A\bar{B}CD$

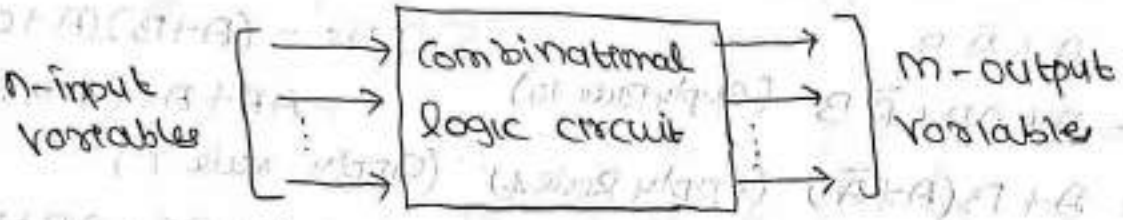
$ABCD + A\bar{B}CD$
$= ACD(B + \bar{B})$
$= ACD$

(b) $AB + ABC + AB(D+E)$

$= AB + ABC + ABD + ABE$
$= AB(1+C) + ABD + ABE$ (Rule 2)
$= AB + ABD + ABE$
$= AB(1+D) + ABE$ (Rule 2)
$= AB + ABE$
$= AB(1+E)$ (Rule 2)
$= AB$

Combinational logic circuit: The logic circuit whose output at any instant of time entirely depend upon the input signal present at the time are known as combinational circuit.

Combinational circuit does not contain memory elements.



Representation of logic function

Standard SOP form → Minterm

Standard POS form → max term

2^n minterms (or) 2^n max terms

where n = Variable logic

Variables A B C	Minterms m_i	Max terms M_i
0 0 0	$\bar{A}\bar{B}\bar{C}$ (m_0)	$A+B+C = M_0$
0 0 1	$\bar{A}\bar{B}C$ (m_1)	$A+B+\bar{C} = M_1$
0 1 0	$\bar{A}B\bar{C}$ (m_2)	$A+\bar{B}+C = M_2$
0 1 1	$\bar{A}BC$ (m_3)	$A+\bar{B}+\bar{C} = M_3$
1 0 0	$A\bar{B}\bar{C}$ (m_4)	$\bar{A}+B+C = M_4$
1 0 1	$A\bar{B}C$ (m_5)	$\bar{A}+B+\bar{C} = M_5$
1 1 0	$AB\bar{C}$ (m_6)	$\bar{A}+\bar{B}+C = M_6$
1 1 1	ABC (m_7)	$\bar{A}+\bar{B}+\bar{C} = M_7$

$F(A,B,C) = \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}C + AB\bar{C}$

$= m_0 + m_4 + m_2 + m_5$

$F(A,B,C,D) = \sum m(0,2,4,5)$

$\sum$ represent SOP (sum of product)

$F(A,B,C) = (A+B+C)(A+\bar{B}+C)(\bar{A}+\bar{B}+\bar{C})$

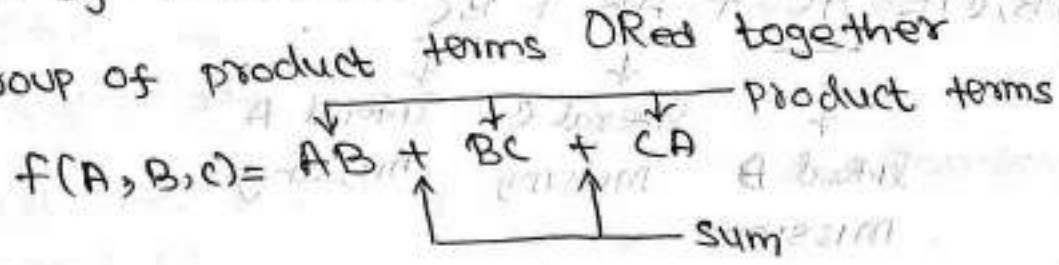
$= M_0 . M_2 . M_7$

$F(A,B,C) = \Pi_M(0,2,7)$

Π denotes POS (product of sum)

Sum of product (SOP) form

* Group of product terms ORed together



* Also called as disjunctive normal formula (or) disjunctive normal form

Standard SOP form or minterm canonical form

In each term of SOP form contain all the literals, then the SOP form is called as Canonical SOP form / minterm canonical form.

(Eg): $f(A,B,C) = ABC + \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C}$.

Converting expressions to standard SOP form

(i) Find the missing literals in each product term if any

(ii) AND each product term having missing literals with terms form by ORing the literals and its complement

(iii) expand the terms by applying distributive law and re-order the literals in product terms

(iv) reduce the expression by omitting repeated product terms.

Convert the expression $f(A,B,C) = AB + BC + AC$ into standard SOP form

$f(A,B,C) = \underline{AC} + \underline{AB} + \underline{BC}$

$\underline{AC}$ → literal B missing

$\underline{AB}$ → literal C missing

$\underline{BC}$ → literal A missing

$f(A,B,C) = AC(B+\bar{B}) + AB(C+\bar{C}) + BC(A+\bar{A})$

$= ABC + A\bar{B}C + ABC + AB\bar{C} + ABC + \bar{A}BC$

omit the repeated product terms

$f(A,B,C) = ABC + A\bar{B}C + AB\bar{C} + \bar{A}BC$

Obtain the canonical SOP form for the function

$Y(A,B,C) = A + BC + \bar{A}BC$

Ans: $Y(A,B,C) = ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + \bar{A}BC$

Standard POS form or maxterm canonical form

Each term in pos form contain all the literals then the pos form is known as canonical pos form.

(eg): $f(A,B,C) = (A+\bar{B}+C)(A+B+C)$

Converting expression to standard POS form

(i) find the missing literals in each sum term
(ii) OR each sum term having missing literals with term form by ANDing the literal and its complement
(iii) expand the term by applying distributive law and reorder the literals
(iv) Omit the repeated sum terms and reduce the expression.

Convert the following function into canonical product of Max-term

$F(A,B,C) = (A+\bar{B})(B+C)(A+\bar{C})$

$F(A,B,C) = \underbrace{(A+\bar{B})}_{\text{missing term - 'C'}}\ \underbrace{(B+C)}_{\text{missing term - A}}\ \underbrace{(A+\bar{C})}_{\text{missing term - B}}$

$F(A,B,C) = (A+\bar{B}+C\cdot\bar{C})\ (A\cdot\bar{A}+B+C)(A+B\bar{B}+C)$

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

$F(A,B,C) = F\,(A+\bar{B}+C)(A+\bar{B}+\bar{C})\ (A+B+C)(\bar{A}+B+C)(A+\bar{B}+C)$

Convert the given expression in standard POS form

$$f = A \cdot (A+B+C)$$

$$f(A,B,C) = \underline{A} \cdot (A+B+C)$$

↓

B & C are missing

$$f(A,B,C) = (A+B\bar{B}+C\bar{C})\,(A+B+C)$$

$$= (A+B+C\bar{C})\,(A+\bar{B}+C\bar{C})\,(A+B+C)$$

$$= (A+B+C)\,(A+B+\bar{C})\,(A+\bar{B}+C)\,(A+\bar{B}+\bar{C})\,(A+B+C)$$

$$f(A,B,C) = (A+B+C)(A+B+\bar{C})(A+\bar{B}+C)(A+\bar{B}+\bar{C})$$

Convert the expression into POS form

Ans: $Y(A,B,C) = (A+B+C)(A+B+\bar{C})(\bar{A}+B+C)(A+\bar{B}+C)$

Karnaugh Map Representation

* Systematic approach for simplifying a Boolean expression.
* A 'n' variable K-map contains 2^n boxes.
* Also known as Veitch diagram.

Two variable K-map $2^2 = 4$ cells

A \ B	$\bar{B}$ (0)	B (1)
$\bar{A}$ (0)	$\bar{A}\bar{B}_0$	$\bar{A}B_1$
A (1)	$A\bar{B}_2$	AB_3

Three Variable K-map $2^3 = 8$ cells

A \ BC	$\bar{B}\bar{C}$ 00	$\bar{B}C$ 01	BC 11	$B\bar{C}$ 10
$\bar{A}$ 0	$\bar{A}\bar{B}\bar{C}$ 0	$\bar{A}\bar{B}C$ 1	$\bar{A}BC$ 3	$\bar{A}B\bar{C}$ 2
A 1	$A\bar{B}\bar{C}$ 4	$A\bar{B}C$ 5	ABC 7	$AB\bar{C}$ 6

Four variable K-map $2^4 = 16$ cells

AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	$\bar{A}\bar{B}\bar{C}\bar{D}$ 0	$\bar{A}\bar{B}\bar{C}D$ 1	$\bar{A}\bar{B}CD$ 3	$\bar{A}\bar{B}C\bar{D}$ 2
$\bar{A}B$ 01	$\bar{A}B\bar{C}\bar{D}$ 4	$\bar{A}B\bar{C}D$ 5	$\bar{A}BCD$ 7	$\bar{A}BC\bar{D}$ 6
AB 11	$AB\bar{C}\bar{D}$ 12	$AB\bar{C}D$ 13	$ABCD$ 15	$ABC\bar{D}$ 14
$A\bar{B}$ 10	$A\bar{B}\bar{C}\bar{D}$ 8	$A\bar{B}\bar{C}D$ 9	$A\bar{B}CD$ 11	$A\bar{B}C\bar{D}$ 10

Minimization using K-map

(i) Grouping two adjacent ones (pair)

(ii) Grouping four adjacent ones (Quad)

(iii) Grouping eight adjacent ones (octet)

Simplification using K-map

(a) Simplification of SOP expression or minterms using K-map

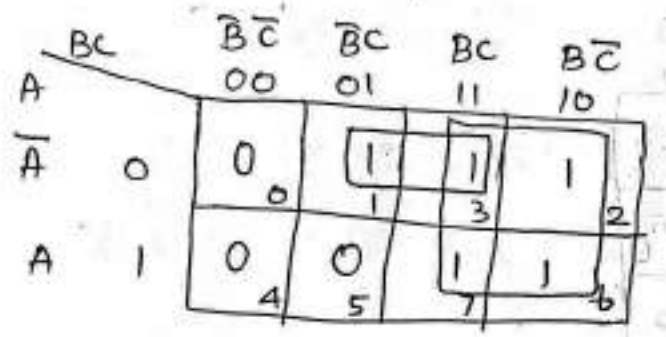
1. plot the expression and place 1's in those cells corresponding to the 1's in the truth table.
2. Encircle the Isolated 1's,

3. Check for those 1's which are adjacent to only one other 1 and encircle such pairs.

4. Check for quads and octets grouping.

5. Make sure that all 1's are grouped or encircled.

Minimize the expression $Y = \bar{A}\bar{B}C + \bar{A}BC + \bar{A}B\bar{C} + ABC + AB\bar{C}$



I^st pair: $\bar{A}\bar{B}C + \bar{A}BC$

$= \bar{A}C$

Quad pair

$\bar{A}BC + \bar{A}B\bar{C} + ABC + AB\bar{C}$

$= BC(A+\bar{A}) + B\bar{C}(A+\bar{A})$

$= BC + B\bar{C}$

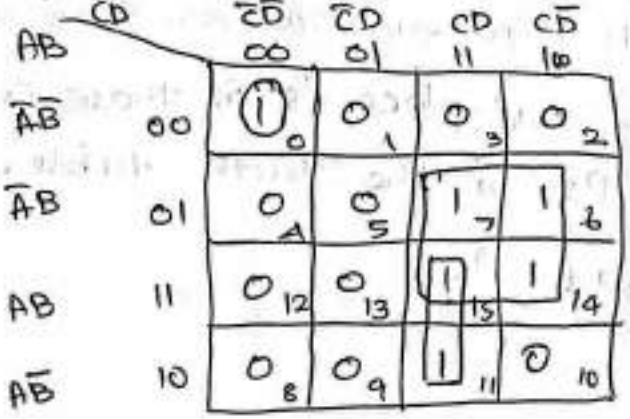
$= B(C+\bar{C})$

$= B$

$\therefore \boxed{Y = \bar{A}C + B}$

Minimize the expression using K-map

$Y = \bar{A}\bar{B}C\bar{D} + \bar{A}BCD + \bar{A}BC\bar{D} + ABCD + ABC\bar{D} + AB\bar{C}D$



Isolated 1: $\bar{A}\bar{B}\bar{C}\bar{D}$

Pair: $ABCD + A\bar{B}CD$

$= ACD(B+\bar{B})$

$= ACD$

Quad

$\bar{A}BCD + \bar{A}BC\bar{D} + ABCD + ABC\bar{D}$

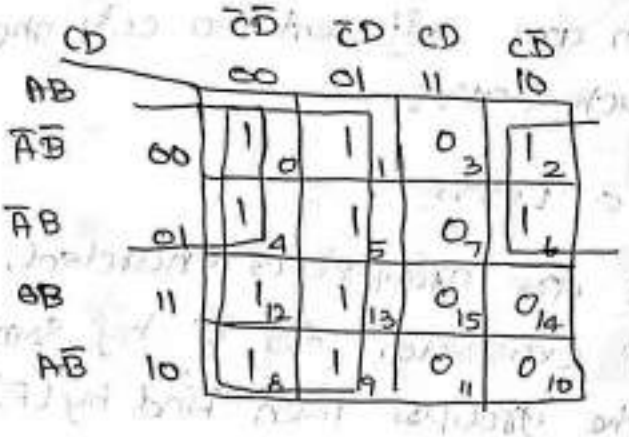
$= \bar{A}BC(D+\bar{D}) + ABC(D+\bar{D})$

$= BC(A+\bar{A})$

$= BC$

$$\boxed{Y = \bar{A}\bar{B}\bar{C}\bar{D} + ACD + BC}$$

Simplify the Boolean function $F = \Sigma m(0,1,2,4,5,6,8,9,12,13)$ using K-map method.



Octet combination

0, 1, 4, 5, 12, 13, 8, 9

Quad combination

0, 4, 2, 6

The simplified Boolean expression $\boxed{Y = \bar{A}\bar{D} + \bar{C}}$

Simplify the following Boolean expressions

(i) $Y = \bar{A}\bar{B}\bar{C}D + \bar{A}B\bar{C}D + \bar{A}BCD + \bar{A}BC\bar{D} + AB\bar{C}\bar{D} + AB\bar{C}D + ABCD + A\bar{B}CD$

Ans: $Y = \bar{A}\bar{C}D + \bar{A}BC + ACD + AB\bar{C}$

(ii) $Y = \bar{A}\bar{B}CD + ABC\bar{D} + A\bar{B}C\bar{D} + A\bar{B}CD + A\bar{B}\bar{C}\bar{D} + AB\bar{C}\bar{D} + \bar{A}\bar{B}CD + \bar{A}B\bar{C}\bar{D}$

Ans: $Y = \bar{B}C + \bar{B}\bar{D} + A\bar{D}$

Reduce the following function using K-map

(i) $f(w,x,y,z) = \Sigma m(0,1,4,8,9,10)$ Ans: $Y = \bar{w}\bar{y}\bar{z} + w\bar{x}\bar{z} + \bar{x}\bar{y}$

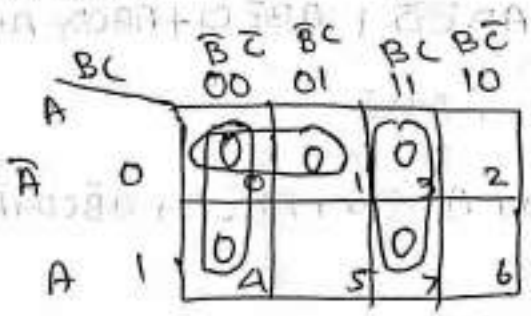
(ii) $Y = \Sigma(7,9,10,11,12,13,14,15)$ Ans: $Y = AB + AD + AC + BCD$

(b) Simplification of POS expression (or) Max-term using K-map

* Simplification of POS expression using K-map is similar to simplification of POS expression.
* Only difference is instead of making the group of ones, we have to make group of zeros.

1. Plot the K-map and place 0s in those cells corresponding to the 0's in the truth table.
2. Encircle the isolated 0's.
3. check for those 0's which are adjacent to only one other '0' and encircle such pairs.
4. check for quads and octets.
5. make sure that all 0's are grouped or encircled.
6. form the simplified SOP expression for $\bar{F}$ by summing product terms of all the groups. Then find by (F) demorgan's theorem

Minimize the expression $Y=(A+B+\bar{C})(A+\bar{B}+\bar{C})(\bar{A}+\bar{B}+\bar{C})(\bar{A}+B+C)(A+B+C)$



$Y = \prod_m (0, 1, 3, 4, 7)$

$\bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C}$ $\quad \bar{A}BC + ABC$ $\quad \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C$

$\bar{B}\bar{C}$ $\quad BC$ $\quad \bar{A}\bar{B}$

$\therefore \boxed{\bar{Y} = \bar{A}\bar{B} + BC + \bar{B}\bar{C}}$

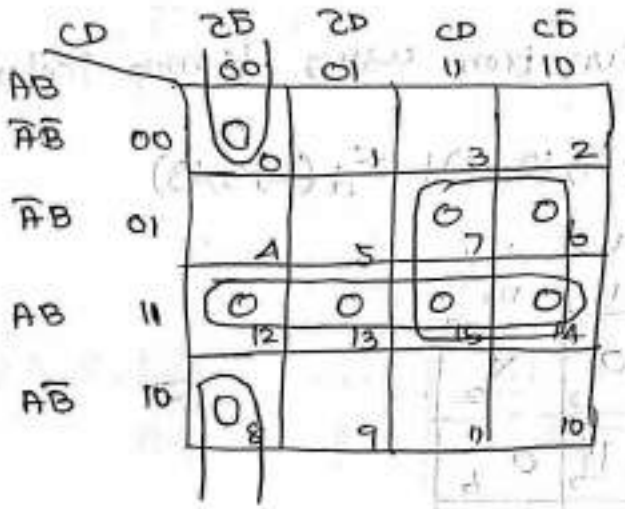
Apply demorgan's theorem

$$\overline{\overline{Y}} = \overline{\bar{B}\bar{C} + \bar{A}\bar{B} + BC}$$

$$\boxed{Y = (B+C)(A+\bar{B})(\bar{B}+\bar{C})}$$

Minimize the following expression in POS form

$$Y = (\bar{A}+\bar{B}+C+D)(\bar{A}+\bar{B}+\bar{C}+D)(\bar{A}+B+C+D)(\bar{A}+\bar{B}+C+\bar{D})$$
$$(A+\bar{B}+\bar{C}+D)(A+\bar{B}+\bar{C}+D)(A+B+C+D)(\bar{A}+\bar{B}+C+\bar{D})$$



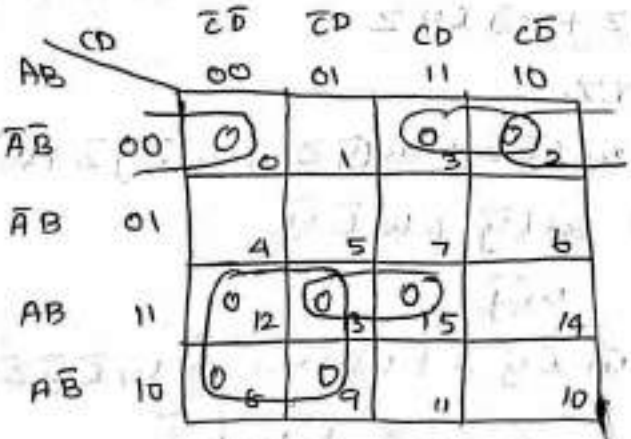
$$\bar{Y} = BC + AB + \bar{B}\bar{C}\bar{D}$$

$$\overline{\bar{Y}} = \overline{BC + AB + \bar{B}\bar{C}\bar{D}}$$

$$\boxed{Y = (\bar{B}+\bar{C})(\bar{A}+\bar{B})(B+C+D)}$$

Reduce the following function using K-map

$$F(A,B,C,D) = \Pi m\,(0,2,3,8,9,12,13,15)$$



$$\bar{F} = \bar{A}\bar{B}\bar{D} + \bar{A}BC + A\bar{C} + ABD$$

$$\overline{\bar{F}} = \overline{\bar{A}\bar{B}\bar{D} + \bar{A}BC + A\bar{C} + ABD}$$

$$\boxed{F = (A+B+D)(A+B+\bar{C})(\bar{A}+C)(\bar{A}+\bar{B}+\bar{D})}$$

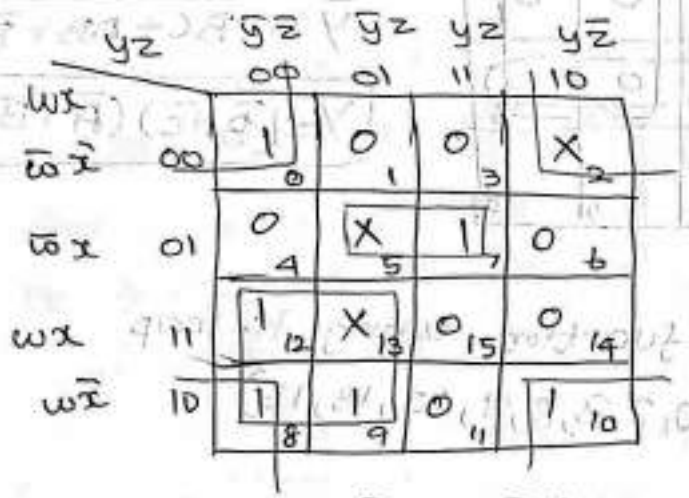
Don't care conditions

In some logic circuits, certain input conditions never occur, therefore the output of such inputs are indicated by "x" or don't care outputs.

A don't care can be considered as '0' or '1' inorder to produce the most simplified output expression.

Reduce the following function using K-map technique

(i) $F(w,x,y,z) = \Sigma_m(0,7,8,9,10,12) + \Sigma_d(2,5,13)$



cells (5,7) $\bar{w}x\bar{y}z + \bar{w}xyz$

$\bar{w}xz$

cells (12,13,8,9) $wx\bar{y}\bar{z} + wx\bar{y}z + w\bar{x}\bar{y}\bar{z} + w\bar{x}\bar{y}z$

$wx\bar{y} + w\bar{x}\bar{y}$

$w\bar{y}$

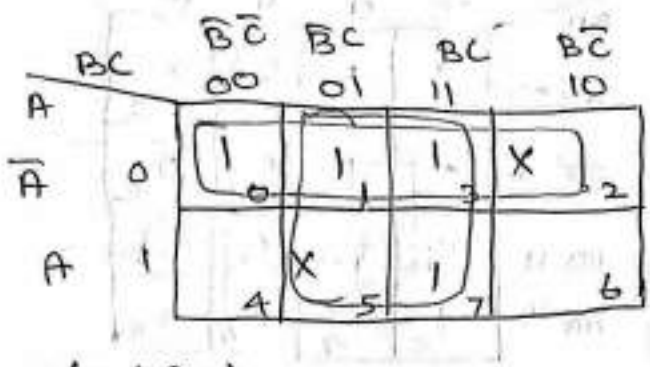
cells (0,2,8,10) $\bar{w}\bar{x}\bar{y}\bar{z} + \bar{w}\bar{x}yz + w\bar{x}\bar{y}\bar{z} + w\bar{x}yz$

$\bar{w}\bar{x}\bar{z} + w\bar{x}\bar{z}$

$\bar{x}\bar{z}$

$$\boxed{F = \bar{w}xz + w\bar{y} + \bar{x}\bar{z}}$$

(ii) $F(A,B,C) = \Sigma m(0,1,3,7) + \Sigma d(2,5)$



quad (0,1,3,2)

quad (1,3,5,7)

$\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}BC + \bar{A}B\bar{C}$

$\bar{A}\bar{B} + \bar{A}B$

$\bar{A}$

$\bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + ABC$

$\bar{A}C + AC$

C

$F(A,B,C) = \bar{A} + C$

(iii) $F(A,B,C,D) = \Sigma m(5,6,7,12,13) + \Sigma d(4,9,14,15)$

Ans: $F(A,B,C,D) = A$

(iv) $F(w,x,y,z) = \Sigma m(0,1,4,8,9,10) + \Sigma d(2,11)$

Ans: $F(w,x,y,z) = \bar{x}\bar{y} + w\bar{x} + \bar{w}\bar{y}z$

(v) $F(w,x,y,z) = \Sigma m(1,2,3,5,9,12,14,15) + \Sigma d(4,8,11)$

Ans: $F = \bar{w}\bar{x}y + \bar{w}\bar{y}z + wxy + \bar{x}z + wx\bar{z}$

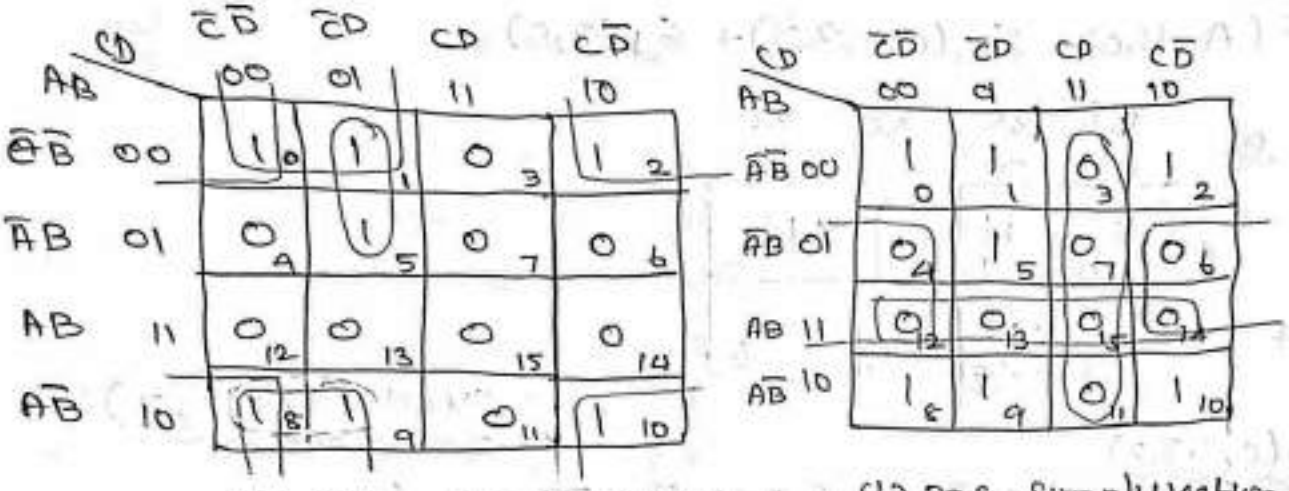
(vi) $f(w,x,y,z) = \Sigma m(0,1,3,9,10,12,13,14) + \Sigma d(2,5,6,11)$

Ans: $f = \bar{w}\bar{x} + \bar{y}z + y\bar{z} + wx\bar{y}$

(vii) $f(w,x,y,z) = \Sigma m(0,1,3,5,6,7,8,12,14) + \Sigma d(9,15)$

Ans: $\bar{w}z + xy + w\bar{y}\bar{z} + \bar{x}\bar{y}$

(viii) $F(A,B,C,D) = \Sigma m(0,1,2,5,8,9,10)$ in sum of product and product of sum.



(a) SOP Simplification

(b) POS Simplification

$F(A,B,C,D) = \bar{B}\bar{C} + \bar{B}\bar{D} + \bar{A}CD$

Pair combination (0,2,8,10)
(1,5)
(0,1,8,9)

$\bar{F} = CD + AB + B\bar{D}$

$\overline{\bar{F}} = \overline{CD + AB + B\bar{D}}$

$= (\bar{C}+\bar{D})(\bar{A}+\bar{B})(\bar{B}+D)$

Pair combination
(3,7,15,11) (4,12,6,14)
(12,13,14,15)

(ix) $F(A,B,C,D) = \Sigma m(0,2,3,6,7) + \Sigma d(8,10,11,15)$ in SOP and POS form

$F(A,B,C,D) = \bar{A}C + \bar{B}\bar{D} \rightarrow$ SOP

$\bar{F} = B\bar{C} + \bar{C}D + AC$

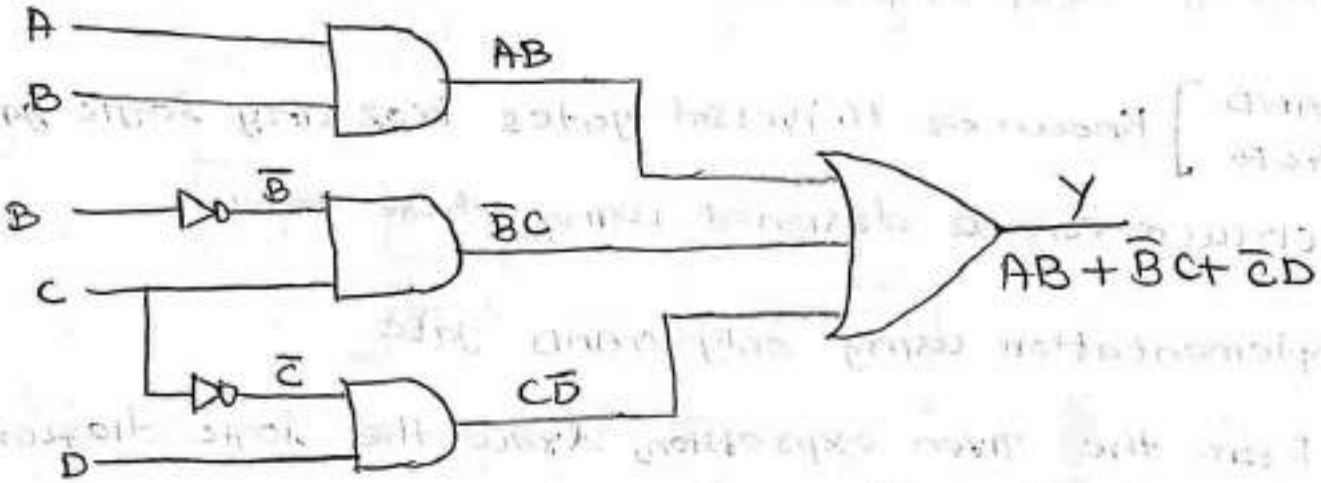
$F = (\bar{B}+C)(C+\bar{D})(\bar{A}+\bar{C}) \rightarrow$ POS

Implementation of SOP expression using logic gates

Consider the Boolean expression

$Y = AB + \bar{B}C + \bar{C}D$

* Three product term
* Use AND and OR gate



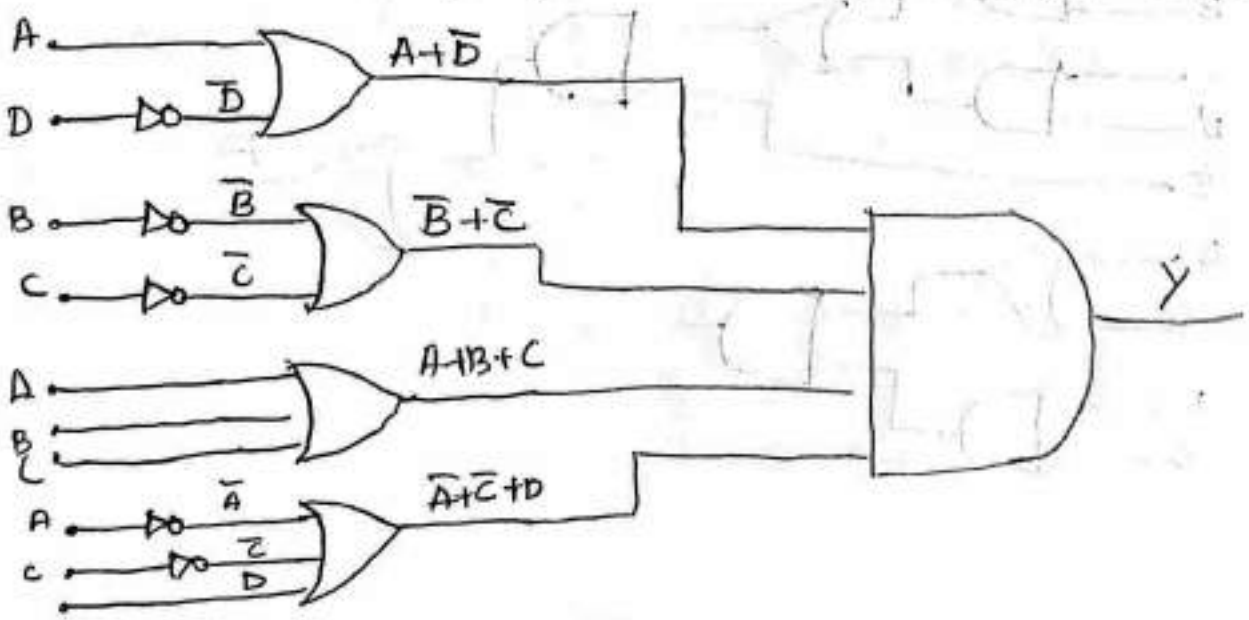
Implementation of POS expression using logic gates

Given $Y=\Pi(0,1,3,5,6,7,10,14,15)$. Draw the K-map and obtain the simplified expression. Realize the minimum expression using basic gates.

AB \ CD	$\bar{C}\bar{D}$ (00)	$\bar{C}D$ (01)	CD (11)	$C\bar{D}$ (10)
$\bar{A}\bar{B}$ (00)	0 (0)	0 (1)	0 (3)	(2)
$\bar{A}B$ (01)	(4)	0 (5)	0 (7)	0 (6)
AB (11)	(12)	(13)	0 (15)	0 (14)
$A\bar{B}$ (10)	(8)	(9)	(11)	0 (10)

By simplification, $\bar{Y} = \bar{A}D + BC + \bar{A}\bar{B}\bar{C} + A\bar{C}\bar{D}$

$$Y = (A+\bar{D})(\bar{B}+\bar{C})(A+B+C)(\bar{A}+\bar{C}+D)$$



NAND and NOR Implementation

NAND, NOR] Known as Universal gates bcoz any logic gates (or) circuits can be designed using these gates.

Implementation using only NAND gate

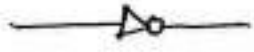
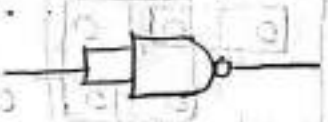
(1) From the given expression, draw the logic diagram using basic gates

(2) Add bubble to the output of each AND gate, also add bubble to the input side of all OR gate.

(3) Add an inverter on each line that received a bubble

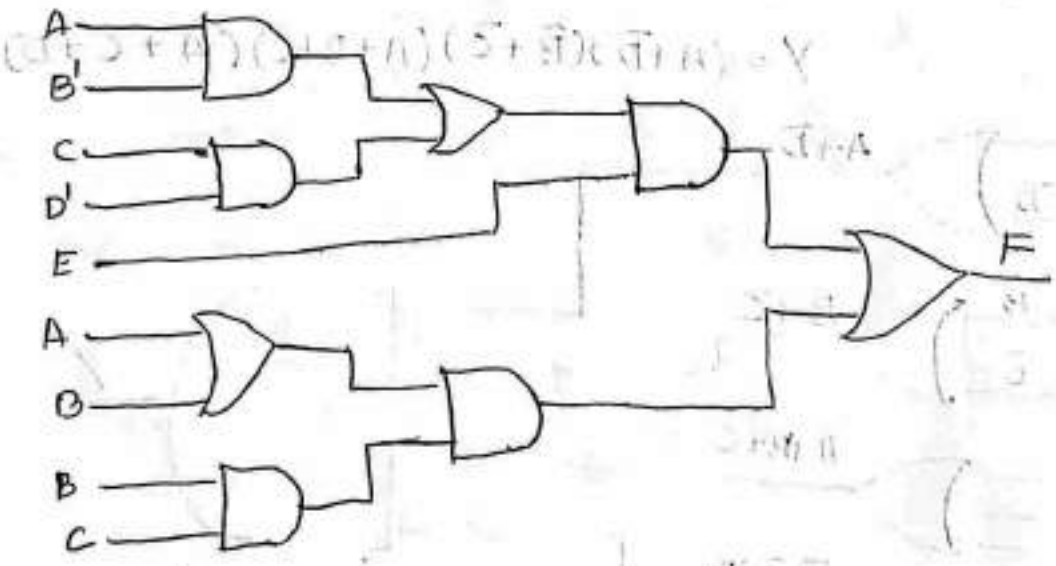
(4) Replace bubbled OR by NAND gate.

(5) Eliminate double inversions.

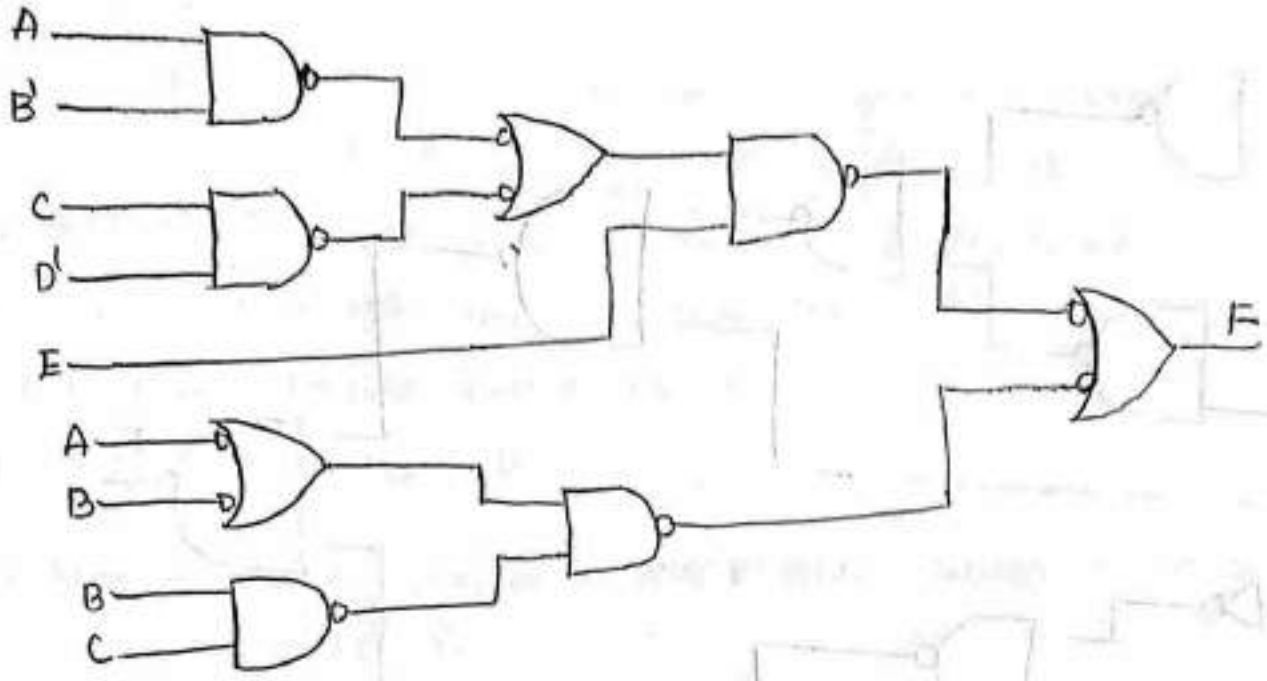
(6) Replace  by 

Draw the multiple-level two input NAND circuit for the following expression $F = (AB' + CD')E + BC(A+B)$

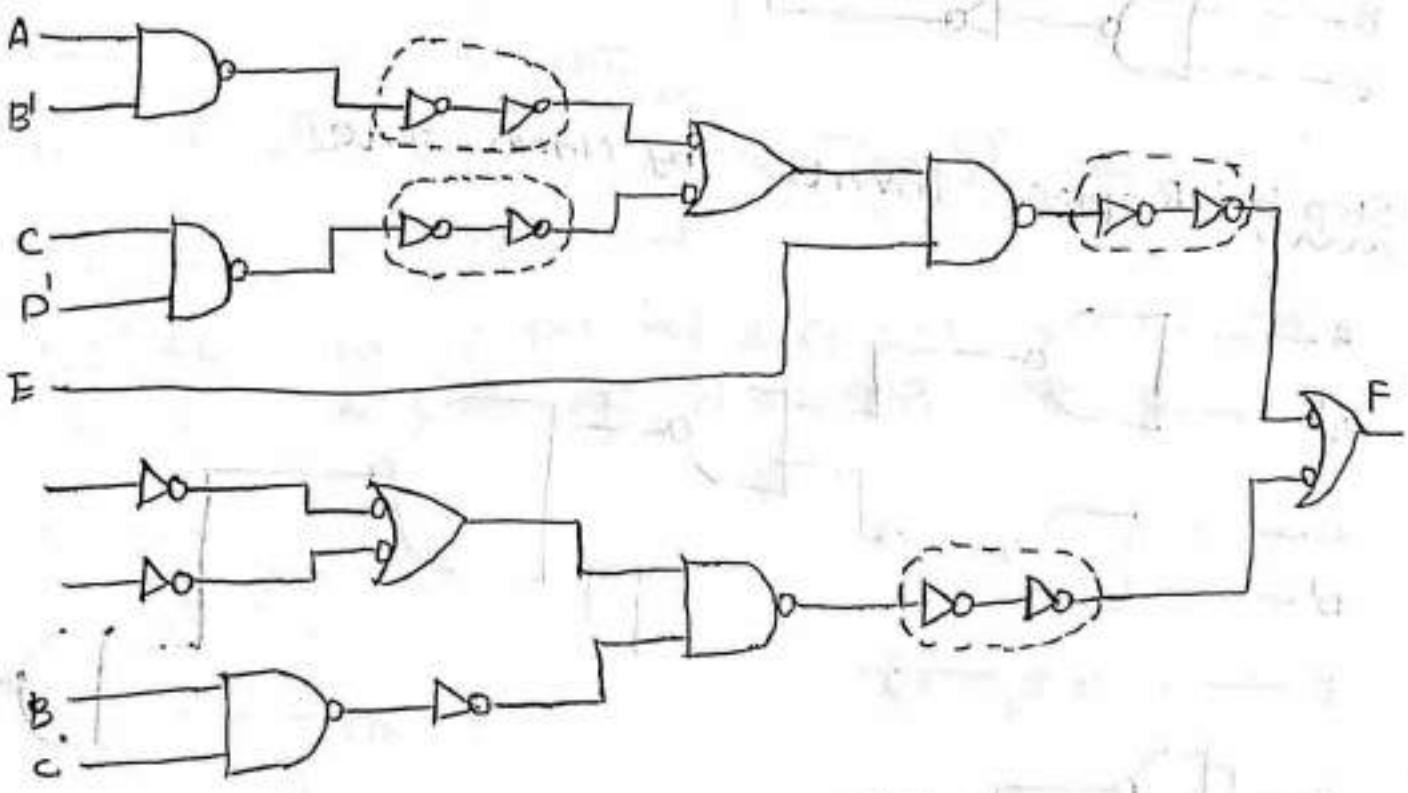
Step 1: Draw the logic diagram using basic gates



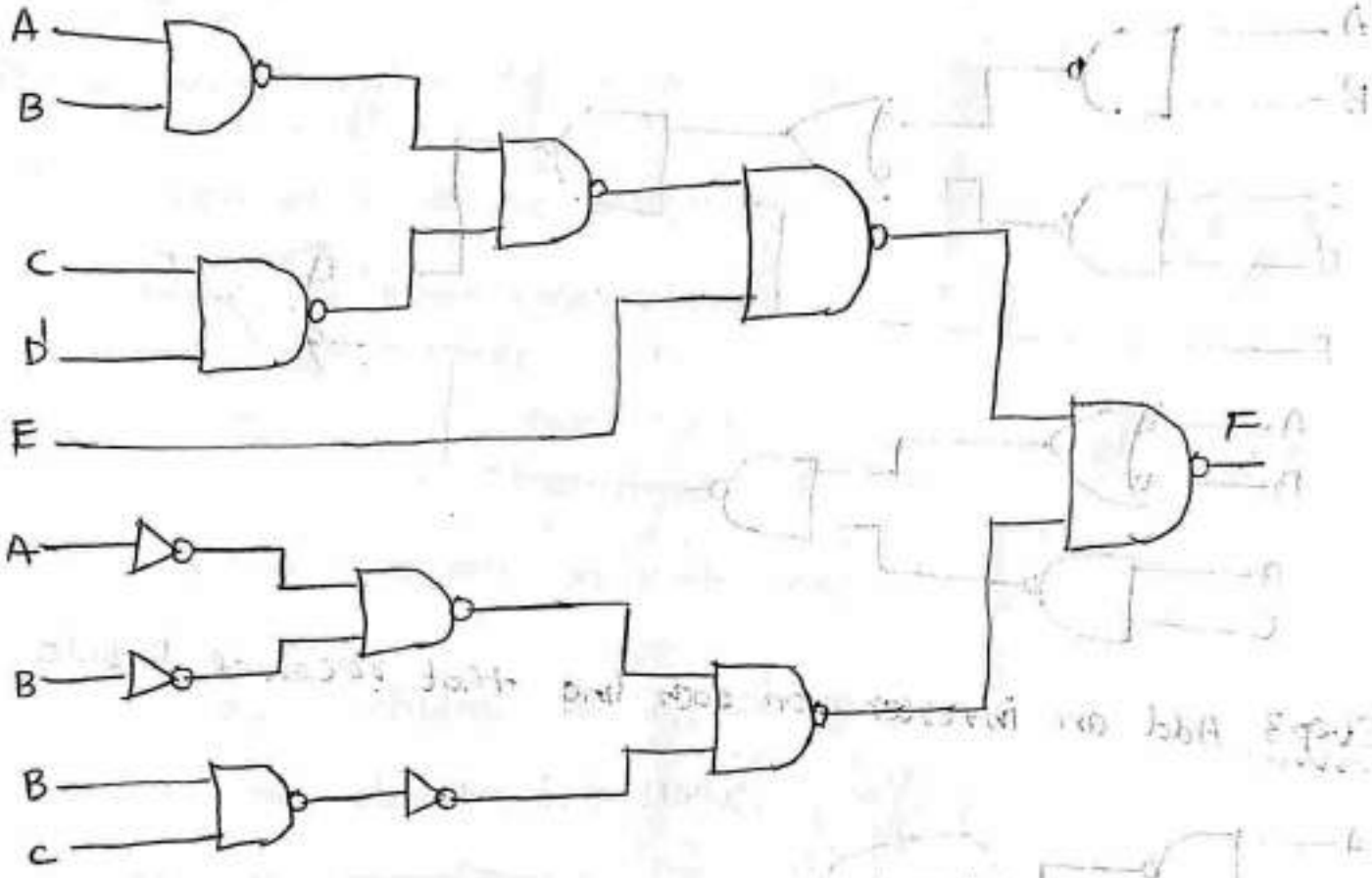
Step 2: Add bubbles on the output of each AND gate and on the inputs of each OR gates.



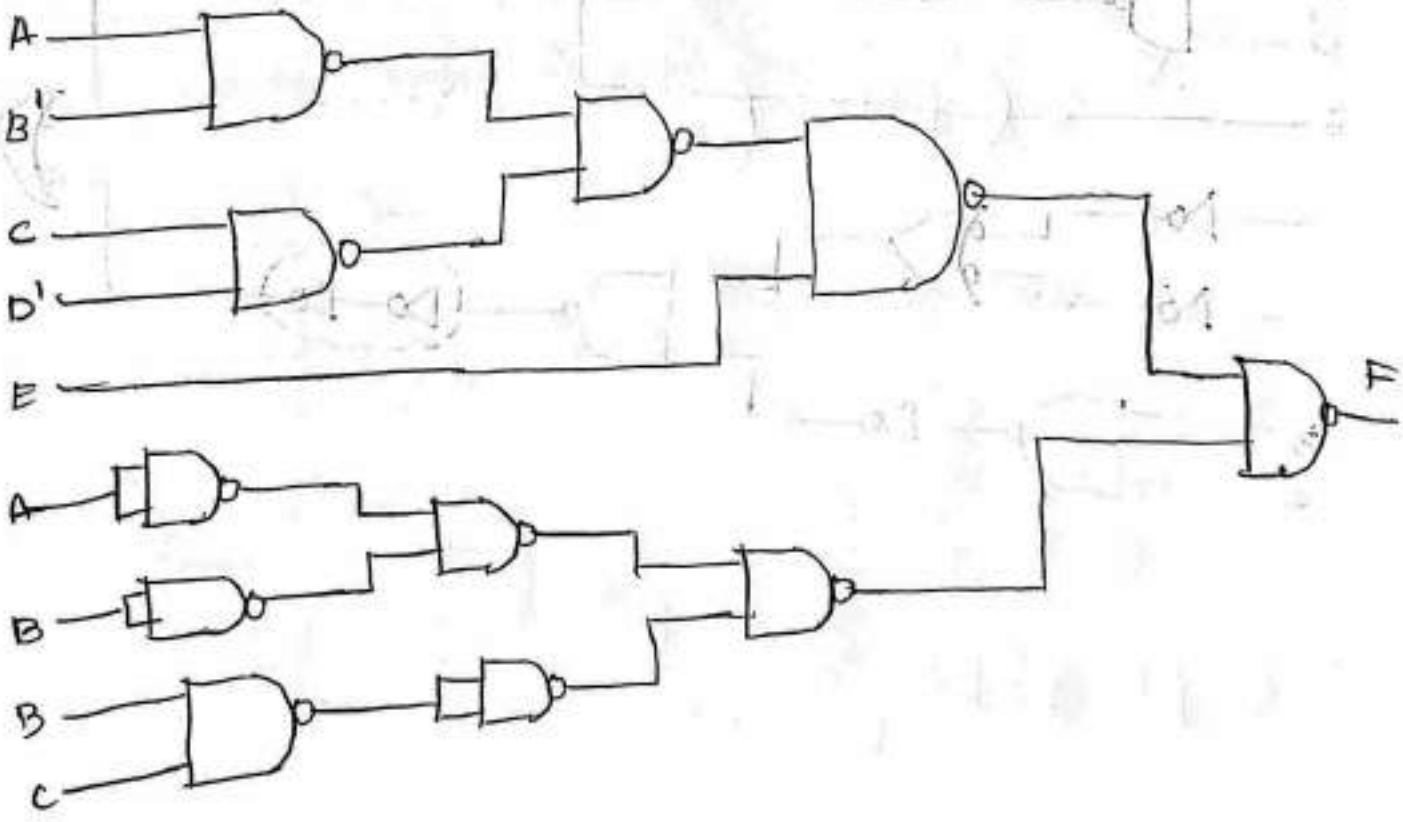
Step 3: Add an inverter on each line that received bubble



Step 4 & 5 : Replace bubbled OR by NAND gate.
Eliminated double inversion



Step 6 : Replace inverter by NAND gate



Implementation using only NOR gate

(1) From the given expression, draw the logic diagram using basic gates

(2) Add bubble to the output of each OR gate, also add bubble to the input side of all AND gates

(3) Add an inverter on each line that received a bubble.

(4) Replace bubbled AND by NOR gate.

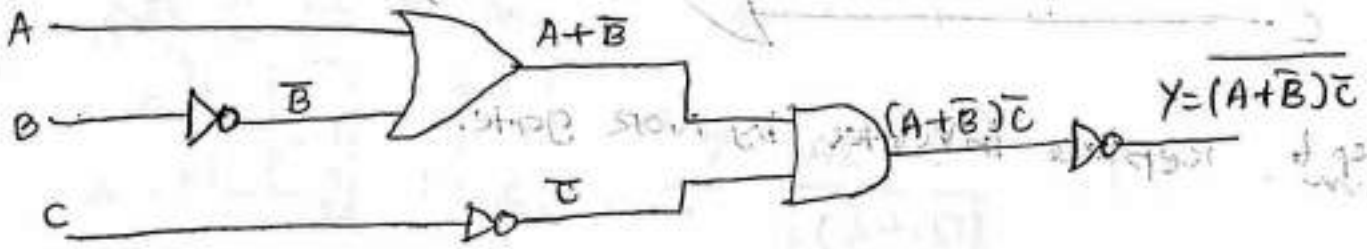
(5) Eliminate double inversions

(6) Replace  by 

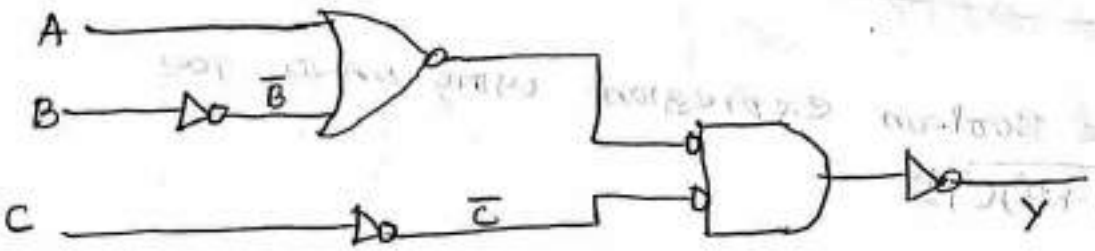
Implement the Boolean expression using NOR gate

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

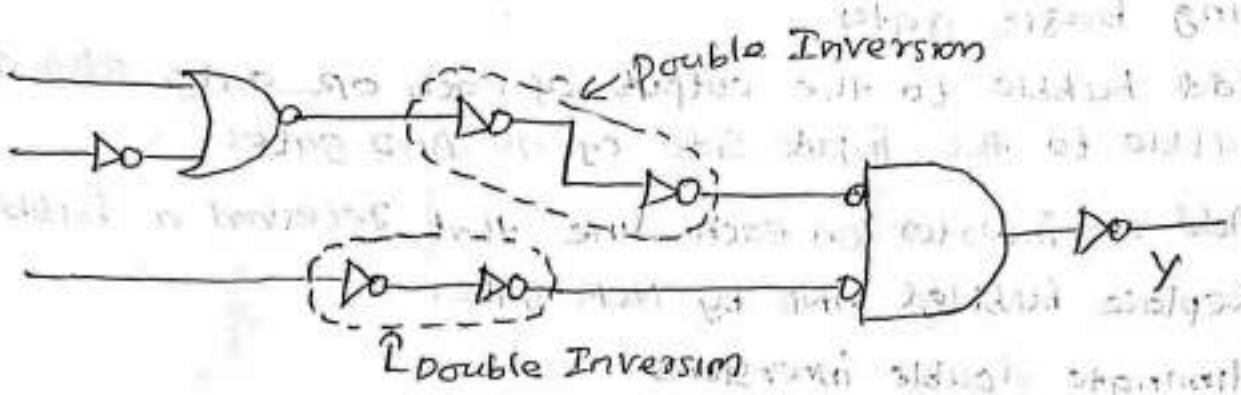
Step 1: Draw the logic diagram using logic gates



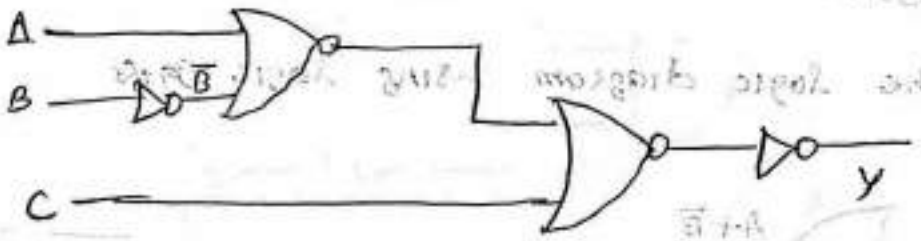
Step 2: Add bubble to the output of each OR gate, also add bubble to the input side of all AND gates.



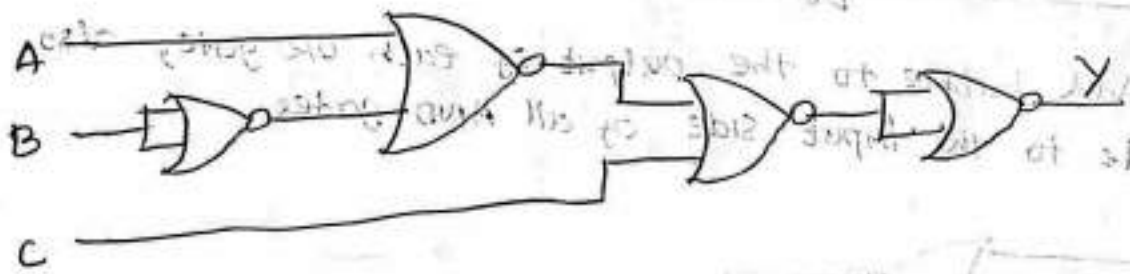
Step 3: Add an inverter on each line that received bubble



Step 4 & 5: Replace bubbled AND by NOR. Eliminate double inversion



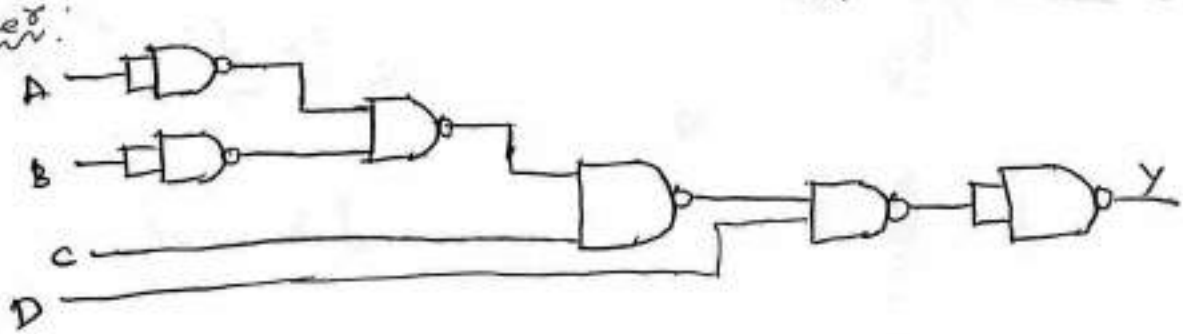
Step 6: Replace inverter by NOR gate.



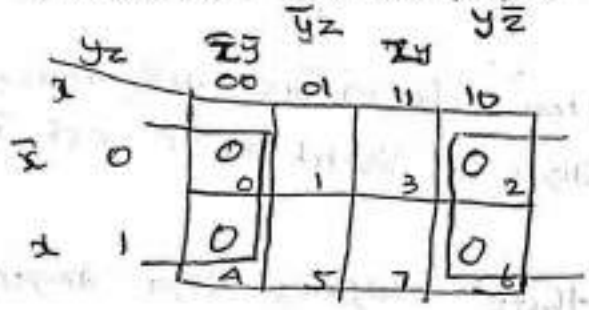
Implement the Boolean expression using NAND gate

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

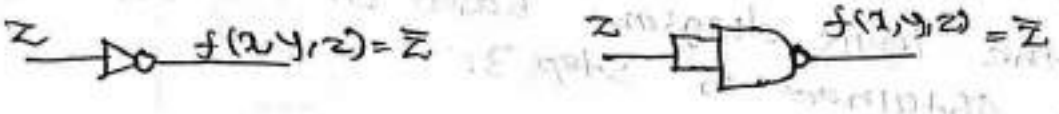
Answer:



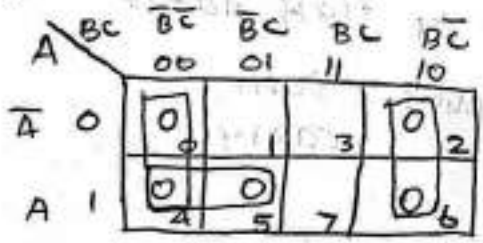
Implement the following function using only NAND gate. $f(x,y,z) = \sum m(0,2,4,6)$



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

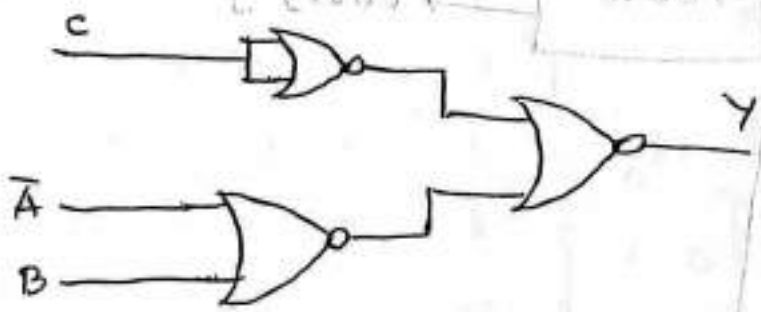


Implement the expression $Y(A,B,C) = \Pi(0,2,4,5,6)$ using only NOR-NOR logic



$$\bar{Y} = A\bar{B} + \bar{C}$$

$$\boxed{Y = C(\bar{A}+B)}$$

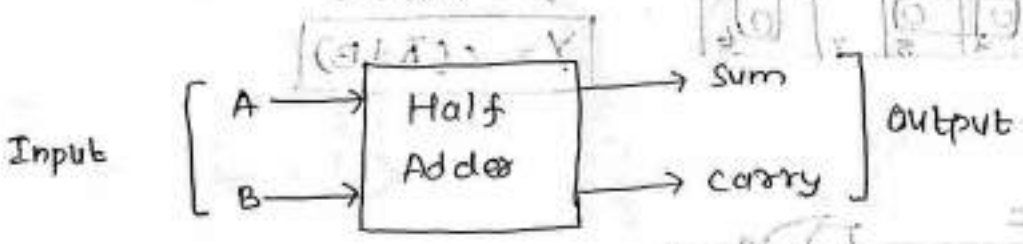


Design procedure in Combinational circuit

1. From the given specification determine the number of inputs and outputs, Assign a symbol to each input and output
2. Derive the truth table that defines the required relationship between inputs and outputs
3. Obtain the Boolean function for each output as a function of the input variable using K-map.
4. Draw the logic diagram based on the Boolean function obtained in Step 3.

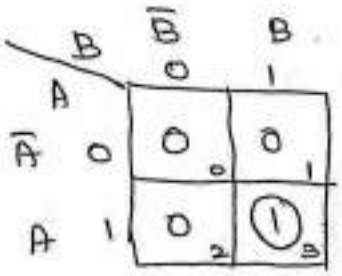
Half Adder

* Addition of two bit is called as Half Adder
* Two inputs A and B; Two outputs Sum carry



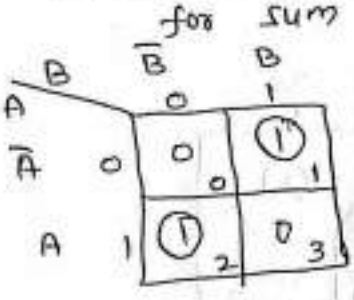
Inputs		Outputs	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

K-map Simplification for carry

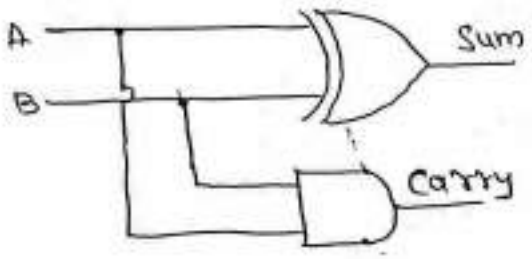


Carry = AB

K-map Simplification for sum



Sum = $A\bar{B} + \bar{A}B = A \oplus B$

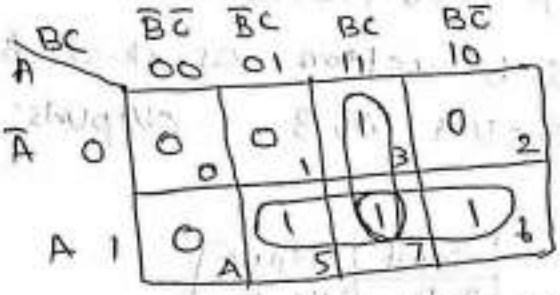


Full Adder

* Addition of three input bits
* Input: A, B, C Output: Sum Carry

Inputs			outputs	
A	B	C	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

K-map Simplification for Carry



Carry = $\bar{A}BC + ABC + AB\bar{C} + A\bar{B}C + ABC + ABC$

$= BC + AC + AB$

K-map simplification for Sum

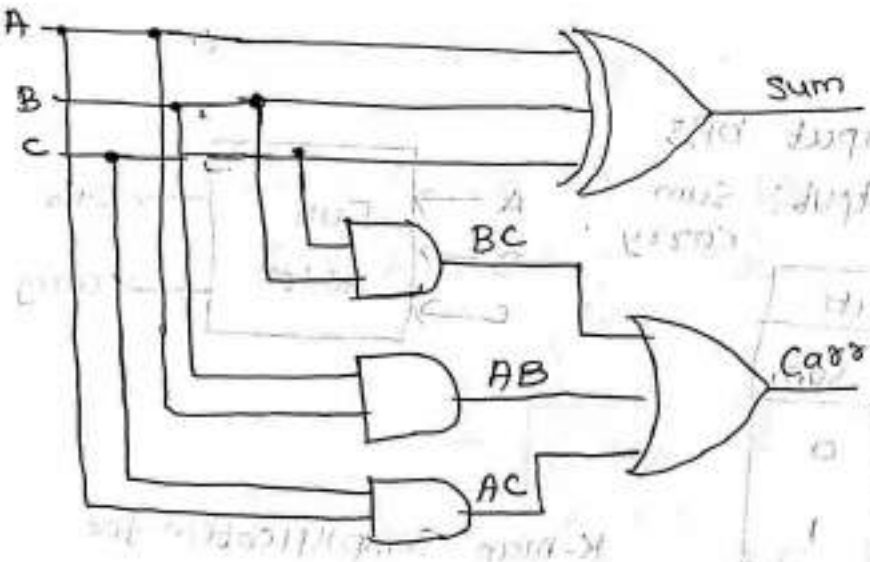
A \ BC	$\bar{B}\bar{C}$ 00	$\bar{B}C$ 01	BC 11	$B\bar{C}$ 10
$\bar{A}$ 0	0 (0)	1 (1)	0 (3)	1 (2)
A 1	1 (4)	0 (5)	1 (7)	0 (6)

$$Sum = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$

$$= \bar{A}(\bar{B}C + B\bar{C}) + A(\bar{B}\bar{C} + BC)$$

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

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

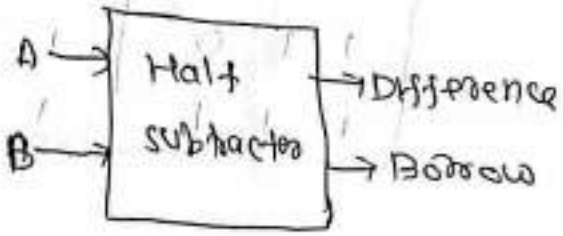


Half Subtractor

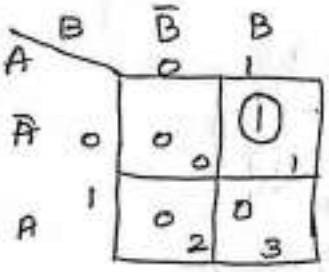
* Subtraction of two bits
* Inputs A, B Outputs: Difference, Borrow

A → Minuend
B → Subtrahend

Input		output	
A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

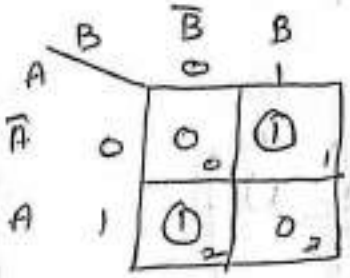


K-map Simplification for Borrow



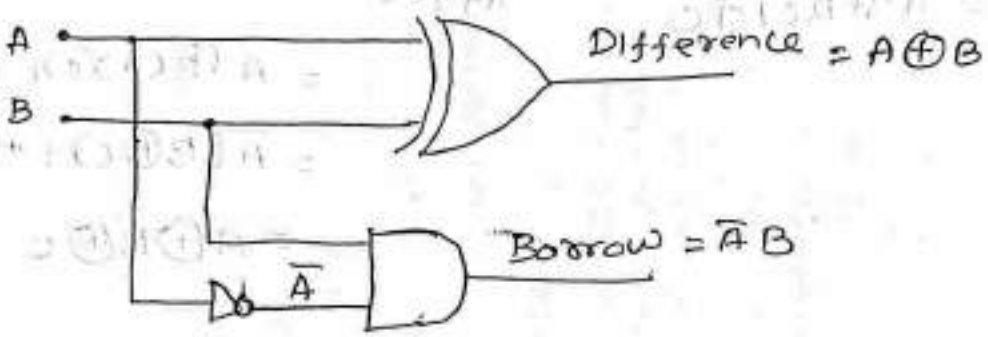
Borrow $\bar{A}B$

K-map Simplification for Difference



Difference $= \bar{A}B + A\bar{B}$

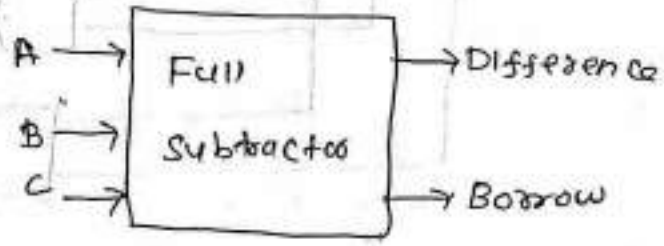
$= A \oplus B$



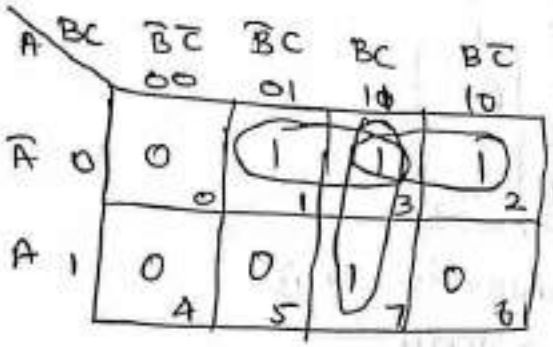
Full Subtractor

* Subtraction between three bits
* Inputs A, B, C Output: Difference, Borrow

Inputs			Outputs	
A	B	C	Diff	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

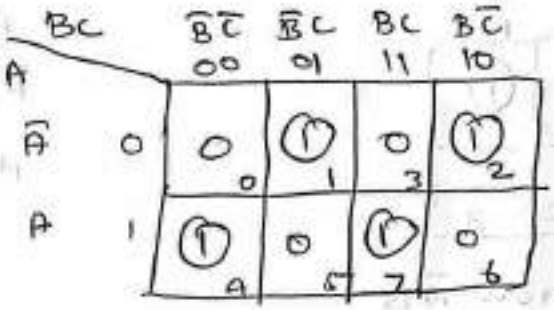


K-map Simplification for Borrow



Borrow = $\bar{A}B + BC + \bar{A}C$

K-map Simplification for difference

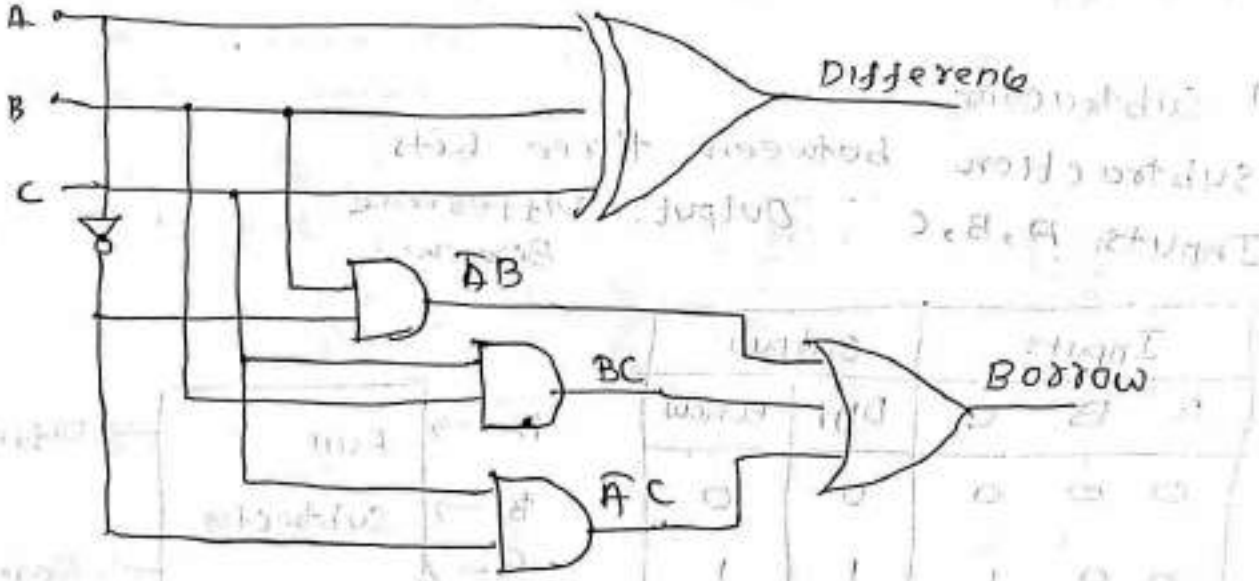


Difference = $\bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$

$= \bar{A}(\bar{B}C + B\bar{C}) + A(\bar{B}\bar{C} + BC)$

$= \bar{A}(B \oplus C) + A(B \odot C)$

$= A \oplus B \oplus C$



2.10 PARALLEL BINARY SUBTRACTOR

The parallel subtractor performs the subtraction operation based on 2's complement subtraction. The subtraction A-B can be done by taking the 2's complement of 'B' and adding it to 'A'. The 2's complement of 'B' can be obtained by taking the 1's complement of 'B' and adding 1 to the least significant bit.

The 1's complement can be implemented with inverters and a 1 can be added to the sum through the input carry C_0, by keeping $C_0 = 1$.

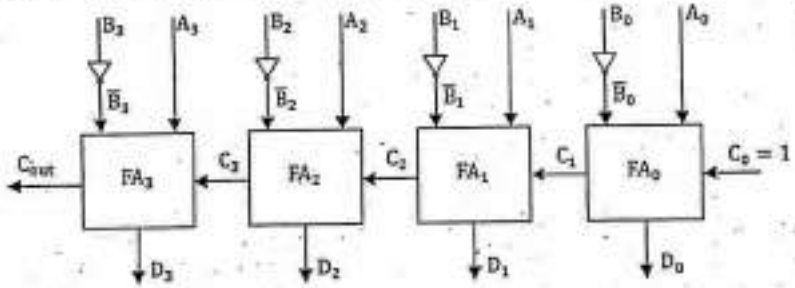


Figure 2.163 4 bit parallel subtractor

The 4-bit parallel subtractor produces the output difference D (D_3, D_2, D_1 and D_0). Consider the binary numbers A=1100 and B=0101. Here $C_0 = 1$. The 4-bit subtractor performs the subtraction as follows.

Minuend	A=1100
Subtrahend	B=0101

	C_{out}	C_3	C_2	C_1	C_0
Carry C	1	0	0	0	1
A		1	1	0	0
$\bar{B}$	+	1	0	1	0
Difference D		0	1	1	1
		D_3	D_2	D_1	D_0

Here the 4 bit subtractor produces the difference 0111 at the output.

2.11 PARALLEL BINARY ADDER/ SUBTRACTOR

The addition and subtraction operations can be performed in one common circuit known as binary adder/ subtractor. This can be done by including an XOR gate as shown in figure 2.164. The mode input 'M' control the operation of the circuit. When M=0, $C_0 = 0$ also the full adders gets the inputs A and B. Hence addition operation is performed. When M=1, $C_0 = 1$ also the full adders gets the inputs A and $\bar{B}$. Hence subtraction operation is performed.

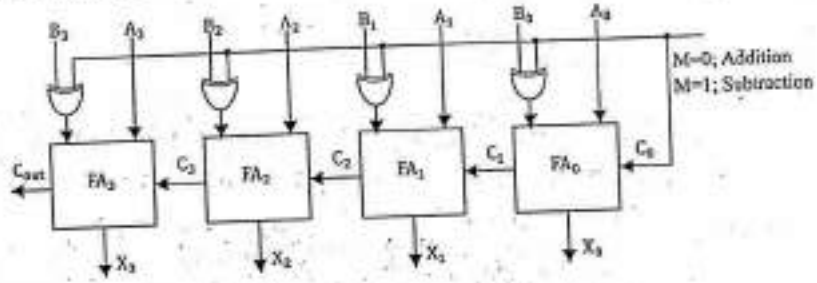


Figure 2.164 4-bit adder/ subtractor.

2.12 FAST ADDER

In the parallel adder, the carry outputs of each full adder is connected to the carry input of the next higher order full adder. Therefore the higher order full adders cannot perform addition without the carry output of the previous stage full adder. This causes a delay in addition using parallel adder which is known as carry propagation delay. So inorder to speedup the addition process fast adders are used.

2.12.1 Carry look ahead adder

One of the method of making the addition process faster is carry look ahead addition which eliminates the ripple carry delay. This method is based on the carry generating and the carry propagating functions of the full adder

(e.g.)

	1	1	0	1
+	1	1	0	0
1	1	0	0	1

In the above example, if each full adder have a propagation delay of 25ns. The carry propagation delay of 4-bit parallel adder is 100ns. Therefore the total time required to produce the output 11001 is 100ns.

The speed of addition process can be increased by using carry look ahead adder.

Here we define two functions carry generate G_i and carry propagate P_i

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

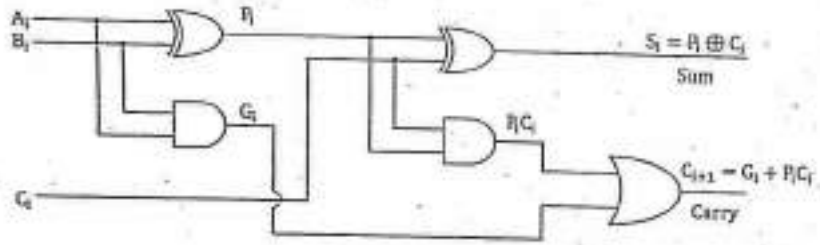


Figure 2.165 Full adder circuit

The output Sum and Carry is given by

$$\text{Sum, } S_i = P_i \oplus C_i$$
$$\text{Carry, } C_{i+1} = G_i + P_iC_i$$

The carry output of each stage can be written as

$C_1 = G_0 + P_0C_0$ where C_0 is the input carry

$$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$$
$$C_3 = G_2 + P_2C_2 = G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_0)$$
$$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$

From the above expression of C_1, C_2, C_3 it can be seen that C_3 does not have to wait for C_2 and C_1 to propagate. C_3 is estimated at the same time as C_2 and C_1 are estimated.

The carry propagate and carry generate functions are given by,

$$P_0 = A_0 \oplus B_0,\ G_0 = A_0B_0$$
$$P_1 = A_1 \oplus B_1,\ G_1 = A_1B_1$$
$$P_2 = A_2 \oplus B_2,\ G_2 = A_2B_2$$
$$P_3 = A_3 \oplus B_3,\ G_3 = A_3B_3$$

The sum output of each stage is given by

$$S_0 = P_0 \oplus C_0;\ S_1 = P_1 \oplus C_1$$
$$S_2 = P_2 \oplus C_2;\ S_3 = P_3 \oplus C_3$$

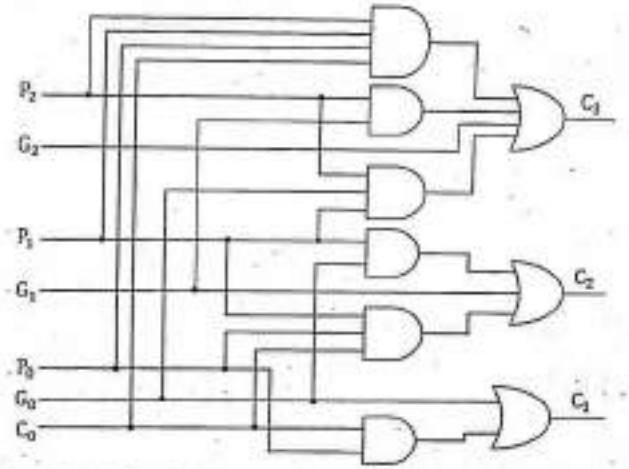


Figure 2.166 Logic diagram of carry look ahead generator

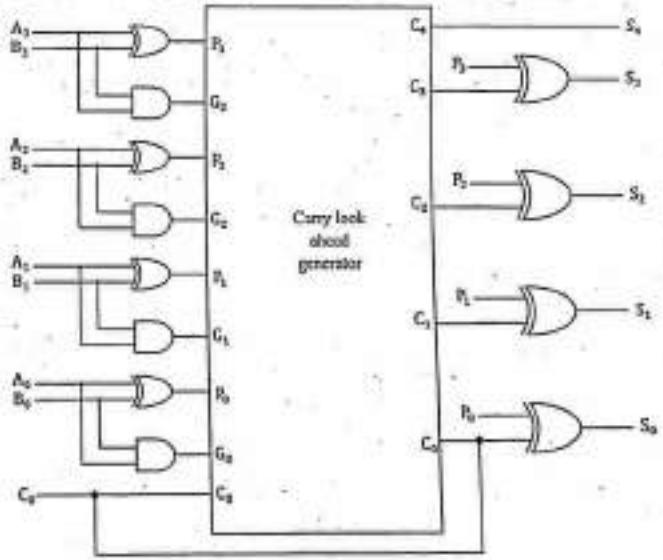


Figure 2.167 4-bit parallel adder with look ahead carry generator

The output of first XOR gate generates P_i and the AND gate generates G_i from the inputs A_i and B_i. The carries are generated using the carry look ahead generator with P_i, G_i and C_0 as inputs. Finally the sum is generated using the second XOR gate with P_i and carries as inputs.

2.13 SERIAL ADDER/SUBTRACTOR

Serial adder can add numbers stored in the right shift register A and B serially. The full adder is used to perform bit by bit addition and a D-flip flop is used to store the carry output generated after addition. This carry output stored in D-flip flop is used as carry input for the next addition. Initially the D-flip flop is cleared and addition starts with the LSB bit of register A and B. The full adder adds the LSB bit of A and B and provides the output sum and carry. After each clock pulse data present in the right shift registers are shifted right by 1-bit and also the previous carry is fed as input 'C' to the full adder. Figure 2.168 shows a block diagram of serial adder.

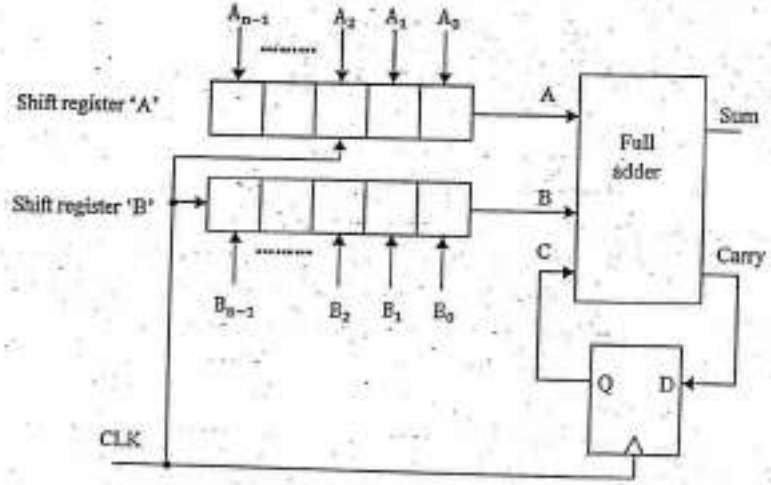


Figure 2.168 Block diagram of serial adder

The full adder provides the output sum and carries on receipt of each clock pulse (CLK) until the two numbers stored initially in shift register A and B have been added and the resulting sum has been clocked back into the shift register A. We can implement serial subtractor by replacing full subtractor instead of full adder.

2.14 BCD ADDER

A BCD adder is a circuit which performs the BCD addition as follows.

i. Add the two BCD numbers to obtain sum.
ii. If the binary sum is greater than 1001_2, add 0110_2 to obtain the sum.

So a BCD adder consists of a

i. 4-bit adder for initial addition.
ii. Circuit to detect whether sum is greater than 1001_2.
iii. 4-bit adder to add 0110_2 with the sum if the sum is greater than 1001_2 or carry is 1.

Let $S_3S_2S_1S_0$ be the sum of the two BCD numbers $B_3B_2B_1B_0$ and $A_3A_2A_1A_0$. The logic circuit to detect sum greater than 9 can be designed by obtaining the Boolean expression from the following truth table shown in table 2.16. If the decimal equivalent of sum $S_3S_2S_1S_0$ is greater than 9, then Y=1

Sum outputs				Logic circuit output
S_3	S_2	S_1	S_0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table 2.16

In the above block diagram when the output carry 'C' is equal to 0, nothing is added to the binary sum ($S_3S_2S_1S_0$). When the output carry is equal to 1, 0110_2 is added to the binary sum ($S_3S_2S_1S_0$) to produce the final BCD sum ($X_3X_2X_1X_0$).

2.15 BINARY MULTIPLIER

Binary multiplication is performed similar to decimal multiplication. The multiplicand is multiplied by each bit of the multiplier starting from the LSB bit. Consider the binary numbers A=110 and B=101. The binary multiplication is performed as follows.

Multiplicand	A →	1	1	0	
Multiplier	B →	1	0	1	
		1	1	0	
	0	0	0	0	
+	1	1	0	0	0
	1	1	1	1	0
	↓	↓	↓	↓	↓
	S_4	S_3	S_2	S_1	S_0

Consider the multiplicand $A = A_1A_0$ and multiplier $B = B_1B_0$. The 2×2 multiplication is carried out as follows.

Multiplicand	A →		A_1	A_0	
Multiplier	B →	×	B_1	B_0	
			B_0A_1	B_0A_0	
+		B_1A_1	B_1A_0	0	
	S_3	S_2	S_1	S_0	

$S_0 = B_0A_0$

$S_1 = B_0A_1 + B_1A_0$

$S_2 = B_1A_1$ + carry out of S_1

S_3 = carry out of S_2

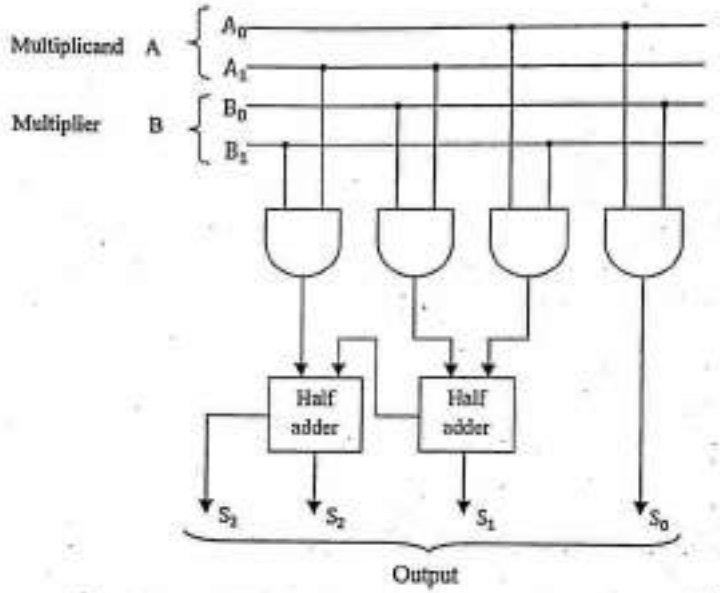


Figure 2.170 2-bit by 2-bit binary multiplier

2.16 BINARY DIVIDER

Binary Divider performs binary Division. Here the dividend is stored in dividend register $X(X_3X_2X_1X_0)$, the divisor is stored in the divisor register $Y(Y_3Y_2Y_1Y_0)$, and initially the Z register $(Z_3Z_2Z_1Z_0)$ is cleared.

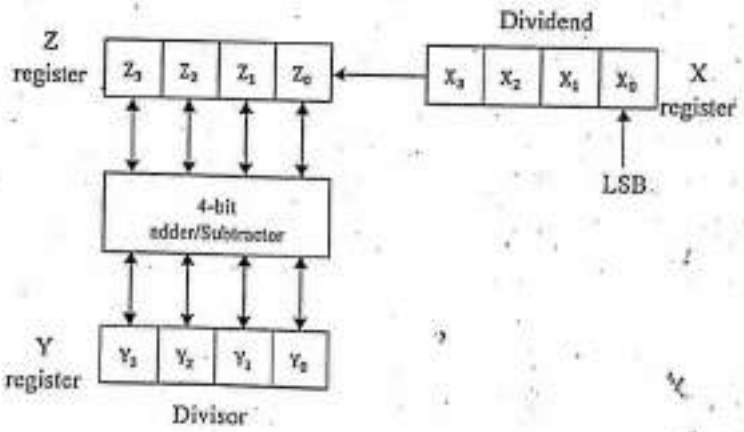


Figure 2.171 Block diagram of Binary Divider

The procedure for division operation using the circuit shown in figure 2.171 is explained as follows.

Step 1: Shift the combined contents of Z and Y register to the left by one bit.

Step 2: Subtract the content of Y register from Z register.

Step 3: If there is no borrow in step 2, put '1' in the LSB of X register, also replace the content of Z register by the difference obtained in step 2.

If there is a borrow in step 2, replace the content of Z register by adding the difference obtained in step 2 with the Y register content.

Step 4: Repeat steps 1 to 3 for 'n' times. Where 'n' is the number of bits in the dividend, for a 4-bit division $n = 4$.

Finally the quotient will be available in the X register and the remainder will be available in the Z register. Consider the division of 1010 (10_{10}) by 0011 (3_{10}). Initially the dividend and divisor are stored in X and Y registers respectively, and the Z register is initially cleared to 0. Therefore

$$X_3X_2X_1X_0 = 1010$$

$$Y_3Y_2Y_1Y_0 = 0011$$

$$Z_3Z_2Z_1Z_0 = 0000$$

I Cycle:

Step 1: Shift the combined contents of Z and X to the left by one bit. Therefore

$$ZX = 0001\ 0100$$

Step 2: Subtract dividend 0011 from register Z.

$$\begin{array}{l} Z = 0\ 0\ 0\ 1 \\ (-)\ Y = 0\ 0\ 1\ 1 \\ \hline \text{Difference} = 1\ 1\ 1\ 0 \end{array} \quad \text{with Borrow '1'}$$

Step 3: Since borrow = 1, replace the content of Z register by adding the difference obtained in step 2 with Y register content.

$$\begin{array}{l} \text{Difference} = 1\ 1\ 1\ 0 \\ (+)\ Y = 0\ 0\ 1\ 1 \\ \hline Z = 0\ 0\ 0\ 1 \end{array}$$

$$ZX = 0001\ 0100$$

Step 2: Subtract dividend 0011 from register Z.

$$\begin{array}{l} Z = 0\ 1\ 0\ 0 \\ (-)\ Y = 0\ 0\ 1\ 1 \\ \hline \text{Difference} = 0\ 0\ 0\ 1 \end{array}$$ with Borrow '0'

Step 3: Since borrow = 0, put '1' in the LSB of X register (X_0), also replace the content of Z register by the difference obtained in step 2.

X = 0011

Z = 0001

ZX = 0001 0011

Now the Quotient $0011(3_{10})$ is available in the X register and the remainder $0001(1_{10})$ is available in the Z register.

2.17 MULTIPLEXER

Multiplexer is a digital circuit that allows the digital information from several sources to be routed on a single output line. A multiplexer consists of '2^n' number of input lines, 'n' selection line and only one output line.

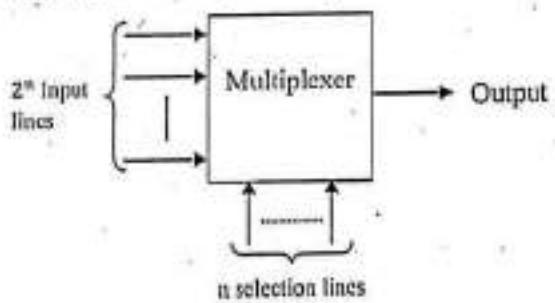


Figure 2.172 Block Schematic of Multiplexer

In this section, we can see

i. 4:1 multiplexer
ii. 8:1 multiplexer

2.17.1 4:1 multiplexer

A 4:1 multiplexer has 4 inputs (D_3, D_2, D_1 and D_0), two selection lines (S_1 and S_0) and single output line 'Y'.

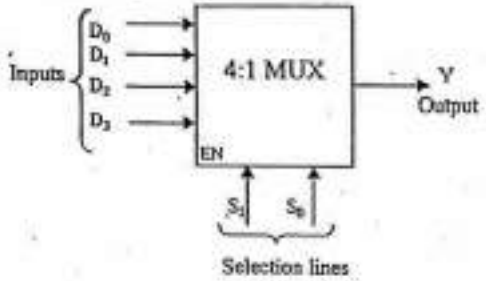


Figure 2.173 Block schematic of 4:1 Multiplexer

Selection lines		Output
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Table 2.17 Truth table of 4:1 Multiplexer

When $S_1S_0 = 00$, the output is $Y=D_0$. When $S_1S_0 = 01$, the output is $Y=D_1$. When $S_1S_0 = 10$, the output is $Y=D_2$. When $S_1S_0 = 11$, the output is $Y=D_3$. The truth table of a 4:1 multiplexer is shown in table 2.17. The logic expression for 4:1 Multiplexer can be written as,

$$Y = \bar{S}_1\bar{S}_0D_0 + \bar{S}_1S_0D_1 + S_1\bar{S}_0D_2 + S_1S_0D_3$$

Logic diagram

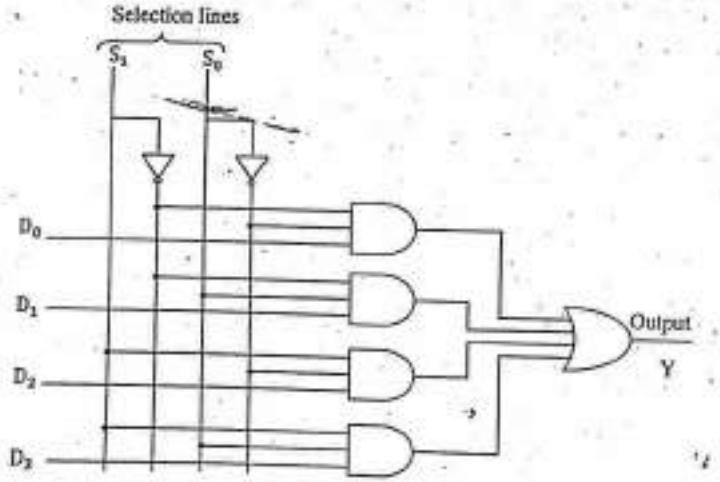


Figure 2.174 Logic diagram of 4:1 multiplexer

When $S_1S_0 = 00$, the AND gate associated with data input D_0 has two of its inputs equal to 1 and the third input is connected to D_0. The other three AND gates have atleast one input equal to 0, which makes their outputs equal to 0. The OR gate output is now equal to D_0. Thus if $S_1S_0 = 00$, the output is D_0. Similarly if $S_1S_0 = 01$, the output is D_1, if $S_1S_0 = 10$, the output is D_2 and if $S_1S_0 = 11$, the output is D_3.

2.17.2 8:1 multiplexer

A 8:1 multiplexer has 8 inputs (D_7, D_6, D_5, D_4, D_3, D_2, D_1, D_0), 3 selection lines (S_2, S_1, S_0) and single output line 'Y'.

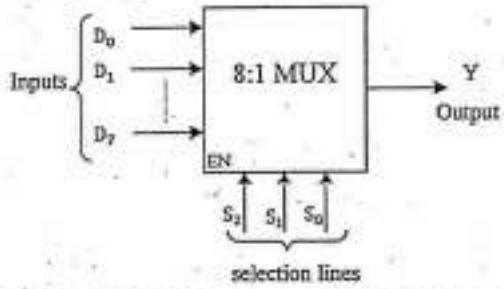


Figure 2.175 Block schematic of 8:1 Multiplexer

When $S_2S_1S_0 = 000$, the output is $Y=D_0$. When $S_2S_1S_0 = 001$, the output is $Y=D_1$. When $S_2S_1S_0 = 010$, the output is $Y=D_2$. When $S_2S_1S_0 = 011$, the output is $Y=D_3$. When $S_2S_1S_0 = 100$, the output is $Y=D_4$. When $S_2S_1S_0 = 101$, the output is $Y=D_5$. When $S_2S_1S_0 = 110$, the output is $Y=D_6$. When $S_2S_1S_0 = 111$, the output is $Y=D_7$.

The truth table of a 8:1 multiplexer is shown in table 2.18.

Selection lines			Output
S_2	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

Table 2.18 Truth table of 8:1 Multiplexer

The logic expression for 8:1 Multiplexer can be written as,

$$Y = \bar{S}_2\bar{S}_1\bar{S}_0D_0 + \bar{S}_2\bar{S}_1S_0D_1 + \bar{S}_2S_1\bar{S}_0D_2 + \bar{S}_2S_1S_0D_3 + S_2\bar{S}_1\bar{S}_0D_4 + S_2\bar{S}_1S_0D_5 + S_2S_1\bar{S}_0D_6 + S_2S_1S_0D_7$$

Logic diagram

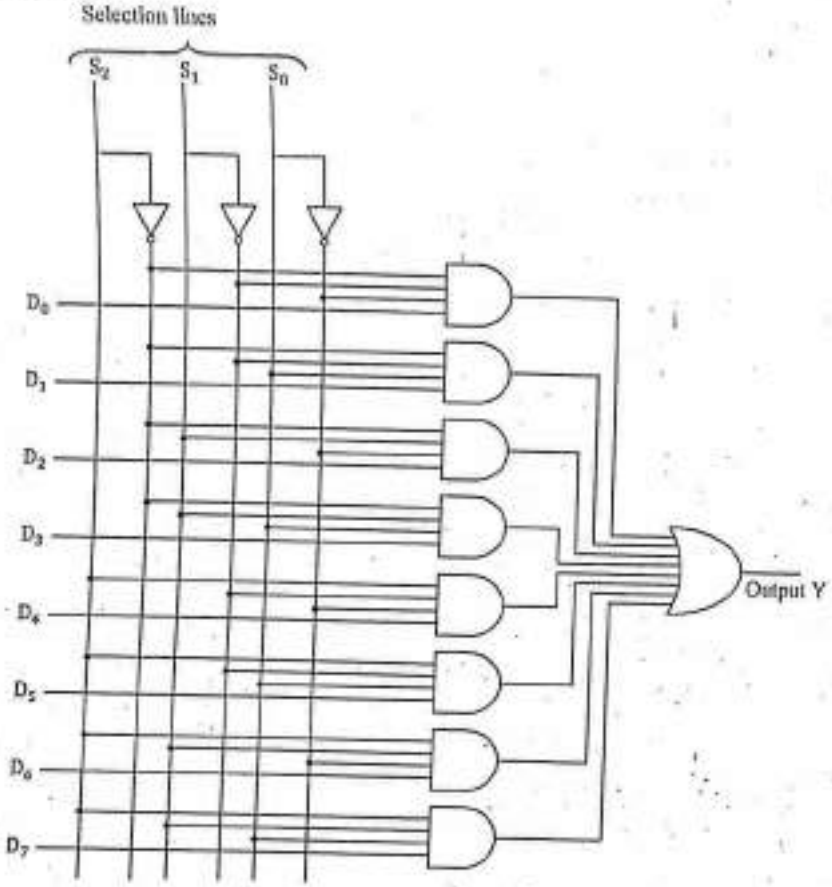


Figure 2.176 Logic diagram of 8:1 Multiplexer

When $S_2S_1S_0 = 011$, the AND gate associated with data input D_3 has three of its inputs equal to 1 and the fourth input is connected to D_3. The other 7 AND gates have atleast one input equal to 0, which makes their outputs equal to 0. The OR gate output is now equal to D_3. Thus if $S_2S_1S_0 = 011$, the output is D_3.

Example 2.50: Implement the following Boolean function using 4:1 multiplexer

$$f(A,B) = \sum_m (0,2)$$

Solution:

The given function $f(A,B)$ can be implemented using a 4:1 multiplexer. A 4:1 multiplexer has 4 inputs (D_3, D_2, $D_1$and D_0), two selection lines (S_1 and S_0) and single output 'Y'. Connect the minterm inputs (D_0, D_2) to high and others (D_1, D_3)to low.

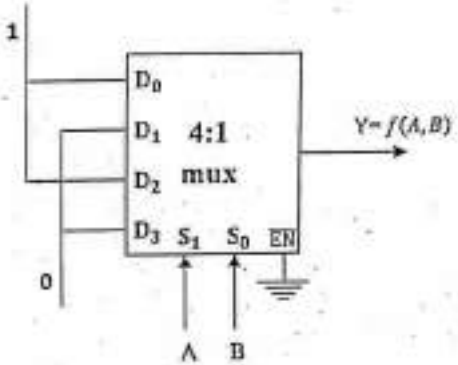


Figure 2.177

Here the variables A and B are applied to the select lines S_1 and S_0 respectively.

Example 2.51: Implement the following Boolean function using 8:1 multiplexer.

$$f(A,B,C) = \sum_m (1,4,5,6).$$

Solution:

The given function $f(A,B,C)$ can be implemented using a 8:1 multiplexer. A 8:1 multiplexer has 8 inputs (D_7, D_6, D_5, D_4, D_3, D_2, D_1, D_0), 3 selection lines (S_2, S_1, S_0) and single output 'Y'. Connect the given minterms D_1, D_4, D_5, D_6 to logic high and other inputs D_0, D_2, D_3, D_7 to logic low.

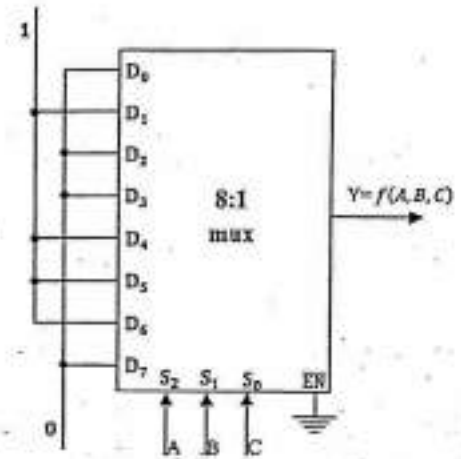


Figure 2.178

Here the variables A, B, C are applied to the select lines S_2, S_1 and S_0 respectively.

Example 2.52: Implement the following Boolean function using 4:1 multiplexer $f(A,B,C) = \sum_m (1,3,5,6)$.

Solution:

In order to implement a 8 input multiplexer function using a 4:1 multiplexer, the 8 input multiplexer function is first converted to 4 input multiplexer function (D_3, D_2, D_1 and D_0). A 4:1 multiplexer has 4 inputs (D_3, D_2, D_1 and D_0), 2 select lines and one output Y.

The given 8 input multiplexer function is converted to 4 inputs (D_3, D_2, D_1 and D_0) as follows.

	D_0	D_1	D_2	D_3
$\overline{A}$	0	(1)	2	(3)
A	4	(5)	(6)	7
	0	1	A	$\overline{A}$

Table 2.19 Implementation table

- Encircle the given minterms.
- If the two minterms in a column are not circled, 0 is applied to the corresponding input.
- If the two minterms in a column are circled, 1 is applied to the corresponding input.
- If the minterm in the first row is circled, $\overline{A}$ is applied to the corresponding input.
- If the minterm in the second row is circled, A is applied to the corresponding input.

Here the column corresponding to D_0 have no circled minterms. Therefore 0 is applied to D_0. The column corresponding to D_1 have two circled minterms. Therefore 1 is applied to D_1. The column corresponding to D_2 have circled minterm on second row. Therefore A is applied to D_2. The column corresponding to D_3 have circled minterm on first row. Therefore $\overline{A}$ is applied to D_3 as shown in figure 2.179.

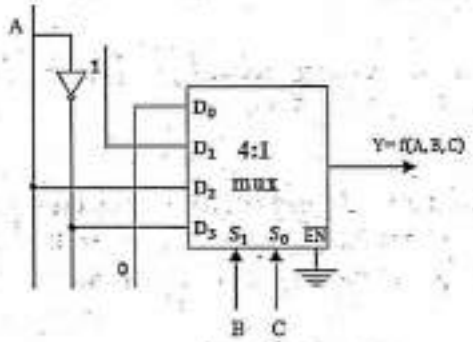


Figure 2.179

Since the 4:1 multiplexer have two selection lines, the last two variables B and C are applied to selection lines S_1 and S_0 respectively.

Example 2.53:Implement the following Boolean function using 8:1 multiplexer.
$f(w,x,y,z) = \sum_m (0,1,3,4,8,9,15)$

Solution:

In order to implement a 16 input multiplexer function using a 8:1 multiplexer, the 16 input multiplexer function is first converted to 8 inputs (D_7, D_6, D_5, D_4, D_3, D_2, D_1, D_0) as shown in table 2.20.

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{w}$	0	1	2	3	4	5	6	7
w	8	9	10	11	12	13	14	15
	1	1	0	$\bar{w}$	$\bar{w}$	0	0	w

Table 2.20 Implementation table

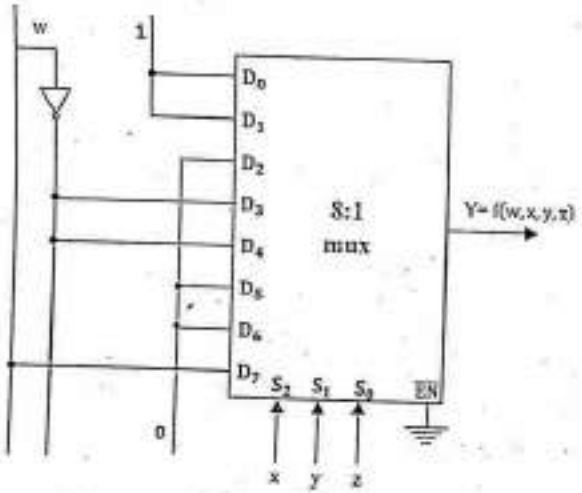


Figure 2.180

Since 8:1 mux have three selection lines, the last 3 variables x, y and z are applied to the selection lines S_2, S_1 and S_0 respectively.

Example 2.54: Implement the following Boolean function using multiplexers.

$$F(A,B,C,D) = \sum_m(0,1,3,4,8,9,15)$$

Solution:

Let $F(A,B,C,D) = \sum_m(0,1,3,4,8,9,15)$. Let us design the given Boolean function using 8:1 mux. In order to implement a 16 input multiplexer function using a 8:1 mux, the 16 inputs are first converted to 8 inputs (D_0, D_1, D_2..... D_7).

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
	1	1	0	$\bar{A}$	$\bar{A}$	0	0	A

Table 2.21 Implementation table

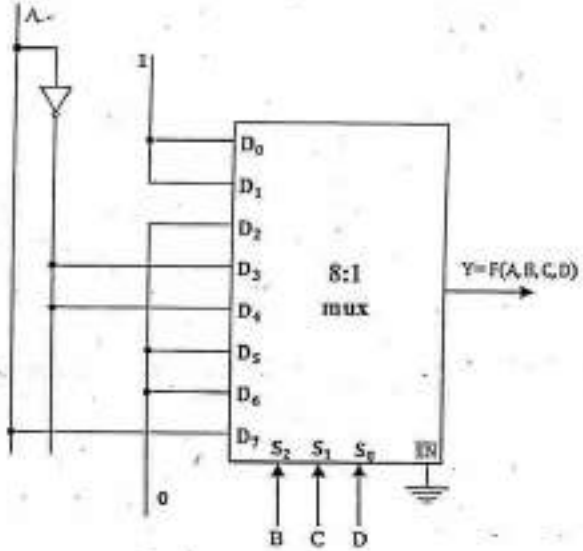


Figure 2.181

Since 8:1 mux have three selection lines, the last 3 variables B, C and D are applied to the selection lines S_2, S_1 and S_0 respectively.

Example 2.55 : Implement the following Boolean function using 4:1 mux.

$$f(w,x,y,z) = \sum_m (0,1,2,4,6,9,12,14)$$

Solution:

The given function $f(w,x,y,z)$ has four variables. To implement this, we require one 8:1 multiplexer or two 4:1 multiplexer. The concept of designing a 8:1 multiplexer is followed here, but the 8:1 multiplexer is replaced by two 4:1 multiplexer and one OR gate.

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{w}$	0	1	2	3	4	5	6	7
w	8	9	10	11	12	13	14	15
	$\bar{w}$	1	w	0	1	0	1	0

Table 2.22 Implementation table

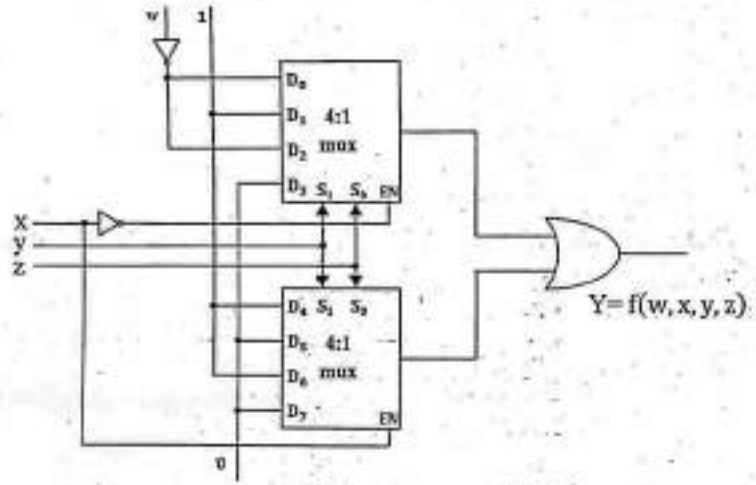


Figure 2.182

Example 2.56: Implement the following Boolean function using 8:1 multiplexer. $f(A,B,C,D) = \bar{A}B\bar{D} + ACD + \bar{B}CD + \bar{A}\bar{C}D$

Solution:

The given Boolean function is not in canonical SOP form. Let us first convert the given expression in canonical SOP form.

Literal B is missing — Literal B is missing

$$f(A,B,C,D) = \bar{A}B\bar{D} + ACD + \bar{B}CD + \bar{A}\bar{C}D$$

Literal C is missing — Literal A is missing

$$f(A,B,C,D) = \bar{A}B\bar{D}(C+\bar{C}) + ACD(B+\bar{B}) + \bar{B}CD(A+\bar{A}) + \bar{A}\bar{C}D(B+\bar{B})$$

$$f(A,B,C,D) = \bar{A}BC\bar{D} + \bar{A}B\bar{C}\bar{D} + ABCD + A\bar{B}CD + A\bar{B}CD + \bar{A}\bar{B}CD + \bar{A}B\bar{C}D + \bar{A}\bar{B}\bar{C}D$$

$$f(A,B,C,D) = \bar{A}BC\bar{D} + \bar{A}B\bar{C}\bar{D} + ABCD + A\bar{B}CD + \bar{A}\bar{B}CD + \bar{A}B\bar{C}D + \bar{A}\bar{B}\bar{C}D$$

Minterms	Binary				Decimal
$\bar{A}BC\bar{D}$	0	1	1	0	6
$\bar{A}B\bar{C}\bar{D}$	0	1	0	0	4
$ABCD$	1	1	1	1	15
$A\bar{B}CD$	1	0	1	1	11
$\bar{A}\bar{B}CD$	0	0	1	1	3
$\bar{A}B\bar{C}D$	0	1	0	1	5
$\bar{A}\bar{B}\bar{C}D$	0	0	0	1	1

Table 2.23 Truth table

From the above truth table the function f(A, B, C, D) can be written as $f(A,B,C,D) = \sum_m (1,3,4,5,6,11,15)$

The 8:1 mux inputs are D_7, D_6, D_5, D_4, D_3, D_2, D_1 and D_0.

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
	0	$\bar{A}$	0	1	$\bar{A}$	$\bar{A}$	$\bar{A}$	A

Table 2.24 Implementation table

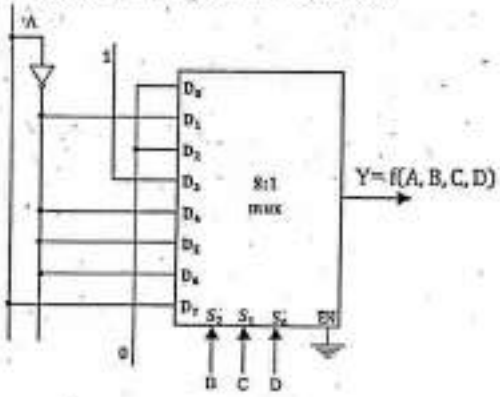


Figure 2.183

Example 2.57 : Implement the following Boolean function with 8:1 mux.
$$f(A,B,C,D) = \Pi M(0,3,5,8,9,10,12,14)$$

Solution:

In the given problem the maxterms are specified instead of minterms. Thus we have to circle the minterms which are not included in the given Boolean function.

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	(1)	(2)	3	(4)	5	(6)	(7)
A	8	9	10	(11)	12	(13)	14	(15)
	0	$\bar{A}$	$\bar{A}$	A	$\bar{A}$	A	$\bar{A}$	1

Table 2.25 Implementation table

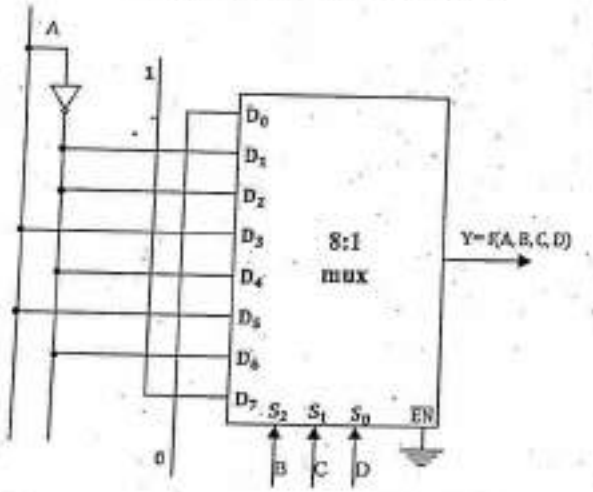


Figure 2:184

Example 2.58 Implement the following Boolean function with 8:1 multiplexer.
$$f(A,B,C,D) = \sum_m (0,2,6,10,11,12,13) + d(3,8,15)$$

Solution:

If don't cares are given in the function, we can include the don't cares, so that we can replace A and $\bar{A}$ by 0 or 1.

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	(0)	1	(2)	3	4	5	(6)	7
A	8	9	(10)	(11)	(12)	(13)	14	15
	$\bar{A}$	0	1	A	A	A	$\bar{A}$	0

Table 2.26 Implementation table without don't cares

We have the don't cares (3,8,15). If we apply the don't care 3 in above table we get D_3 as '1'. If we apply the don't care 8, we get D_0 as '1'. But if we apply the don't care 15 we get D_7 as A. So the don't cares we need to apply are (3,8).

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	
$\bar{A}$	(0)	1	(2)	(3)	4	5	(6)	7	
A	(8)	9	(10)	(11)	(12)	(13)	14	15	← Unused don't care
	1	0	1	1	A	A	$\bar{A}$	0	

Table 2.27 Implementation table with necessary don't cares

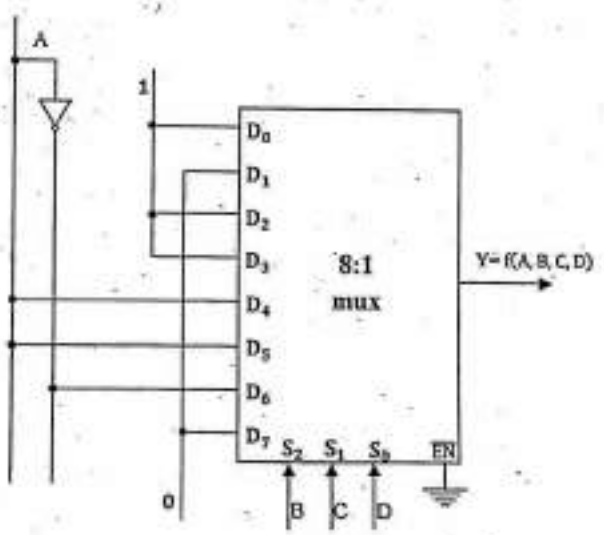


Figure 2.185

Example 2.59: Implement a full adder with two 4×1 multiplexers

Solution:

The truth table of a full adder can be written as shown in table 2.28

Inputs			Outputs	
A	B	C	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 2.28 Truth table of Full adder

From the truth table, the Boolean function of 'Carry' can be written as

$$\text{Carry} = \sum_m(3,5,6,7)$$

The Boolean function of 'Sum' can be written as

$$\text{Sum} = \sum_m(1,2,4,7)$$

The multiplexer circuit for the function 'Carry = $\sum_m(3,5,6,7)$' can be drawn as follows.

	D_0	D_1	D_2	D_3
$\bar{A}$	0	1	2	(3)
A	4	(5)	(6)	(7)
	0	A	A_2	1

Table 2.29 Implementation table for 'Carry'

The 4:1 mulitplexer has two selection lines S_1 and S_0. The last two variables 'B' and 'C' are applied to the selection lines S_1 and S_0 respectively.

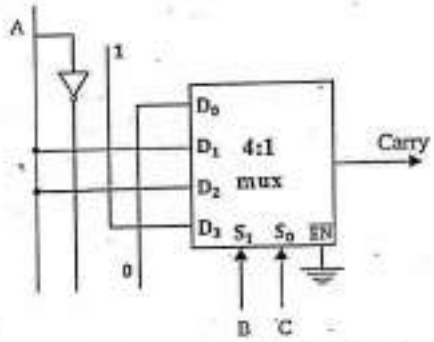


Figure 2.186 4:1 multiplexer for the function 'carry'

The multiplexer circuit for the function 'Sum = $\sum_m(1,2,4,7)$' can be drawn as follows.

	D_0	D_1	D_2	D_3
$\bar{A}$	0	(1)	(2)	3
A	(4)	5	6	(7)
	A	$\bar{A}$	$\bar{A}$	A

Table 2.30 Implementation table for 'Sum'.

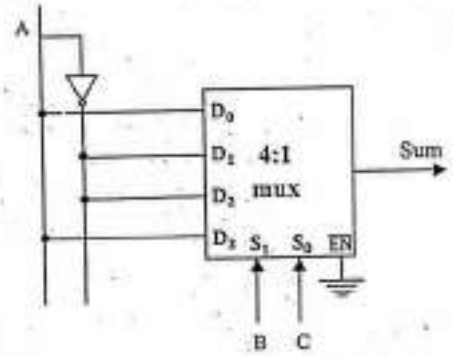


Figure 2.187 4:1 Multiplexer for the function 'Sum'

Merging figure 2.186 and figure 2.187 we can draw the circuit as shown in figure 2.188

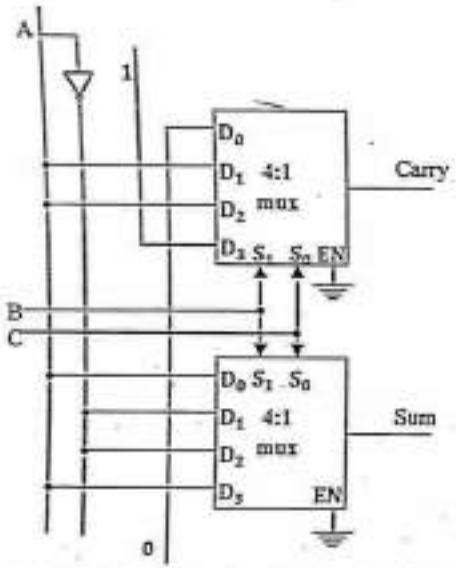


Figure 2.188 Full adder circuit with two 4:1 multiplexers

Example 2.60: Implement the given Boolean function using 4×1 multiplexer
$F(x,y,z) = \sum_m(1,2,6,7)$

Solution:

In order to implement a 8 input multiplexer function using a 4:1 multiplexer, the 8 input multiplexer function is first converted to 4 inputs (D_0, D_1, D_2, D_3). A 4:1 multiplexer has 4 inputs (D_0, D_1, D_2, D_3), two selection lines (S_1 and S_0) and one output 'Y'.

The given 8 input multiplexer function is converted to 4:1 multiplexer inputs as follows.

	D_0	D_1	D_2	D_3
$\bar{x}$	0	(1)	(2)	3
x	4	5	(6)	(7)
	0	$\bar{x}$	1	x

Table 2.31 Implementation table

The 4:1 multiplexer for the above implementation table is as follows.

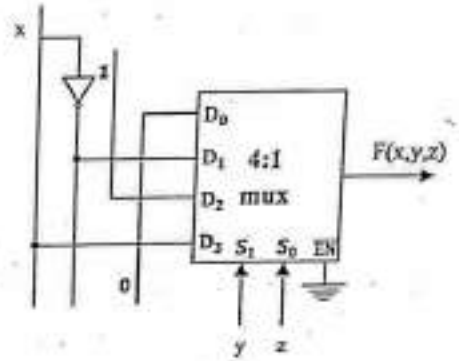


Figure 2.189

Example 2.61: Implement the following function using a suitable multiplexer
$f(a,b,c) = \sum m(3,7,4,5)$.

Solution:

The implementation table is shown in table 2.32

	D_0	D_1	D_2	D_3
$\bar{a}$	0	1	2	(3)
a	(4)	(5)	6	(7)
	a	$\bar{a}$	0	1

Table 2.32 Implementation table

The 4:1 multiplexer for the above implementation table is as follows.

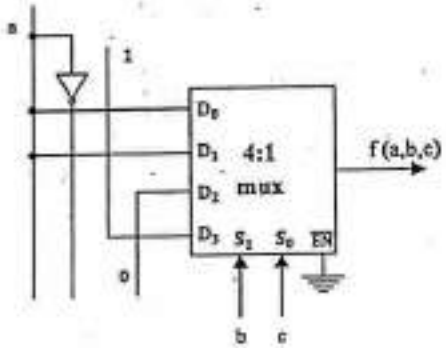


Figure 2.190 Implementation using 4:1 mux

Example 2.62: Implement the switching function $F = \sum_m(0, 1, 3, 4, 12, 14, 15)$ *using an 8 input MUX.*

Solution:

Let $F(A, B, C, D) = \sum_m(0,1,3,4,12,14,15)$

In order to implement a 16 input multiplexer function using a 8:1 mux, the 16 inputs are first converted to 8 inputs (D_0, D_1, D_2..... D_7).

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
	$\bar{A}$	$\bar{A}$	0	$\bar{A}$	1	0	A	A

Table 2.33 Implementation table

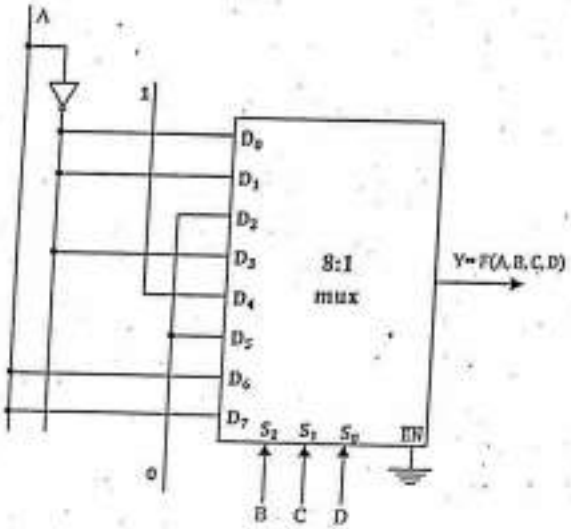


Figure 2.191

Since 8:1 mux have three selection lines, the last 3 variables B, C and D are applied to the selection lines S_2, S_1 and S_0 respectively.

Example 2.63: Implement $F(A, B, C, D) = \sum_m(1, 3, 4, 11, 12, 13, 14, 15)$ *using 8×1 Multiplexer*

Solution:

In order to implement a 16 input mux function using a 8:1 mux, the 16 input multiplexer function is first converted to 8 inputs (D_0, D_1, D_2..... D_7).

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
	0	$\bar{A}$	0	1	1	A	A	A

Table 2.34 Implementation table

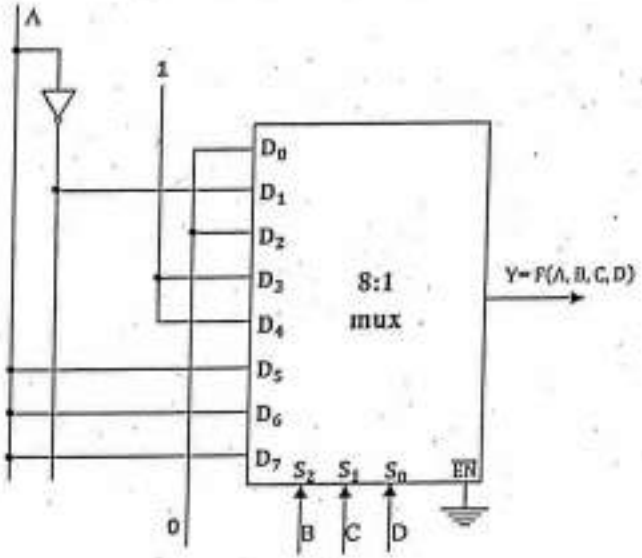


Figure 2.192

Since 8:1 mux have three selection lines, the last 3 variables B, C and D are applied to the selection lines S_2, S_1 and S_0 respectively.

2.18 DEMULTIPLEXER

A demulitplexer is a digital circuit that receives information on a single line and transmits the information on one of the several output lines. A demultiplexer consists of one input line, 'n' selection lines and '2^n' output lines.

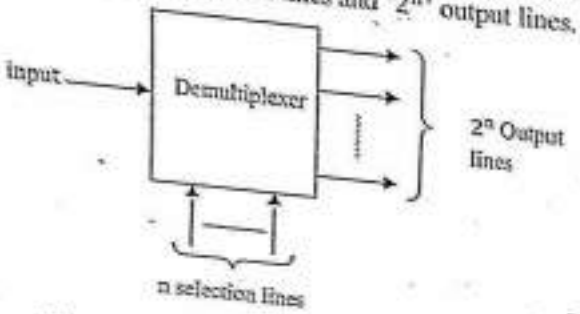


Figure 2.193 Block schematic of Demultiplexer

In this section, we can see

i. 1:4 Demultiplexer
ii. 1:8 Demultiplexer

2.18.1 1:4 Demultiplexer

A 1:4 Demultiplexer has single input D_{in}, 4 outputs (Y_0, Y_1, Y_2, Y_3) and 2 selection lines (S_1, S_0).

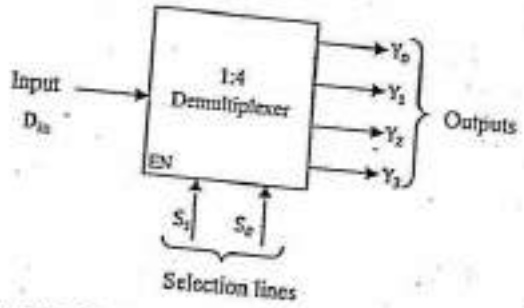


Figure 2.194 Block schematic of 1:4 Demultiplexer

When $S_1S_0 = 00$, the output $Y_0=D_{in}$ and the remaining outputs are '0'. When $S_1S_0 = 01$, the output $Y_1=D_{in}$ and the remaining outputs are '0'. When $S_1S_0 = 10$, the output $Y_2=D_{in}$ and the remaining outputs are '0'. When $S_1S_0 = 11$, the output $Y_3=D_{in}$ and the remaining outputs are '0'.

When $S_1S_0 = 00$, the AND gate associated with data output Y_0 has two of its inputs equal to 1 and the third input is connected to D_{in}. The other three AND gates have atleast one input equal to 0, which makes their outputs equal to 0. Thus if $S_1S_0 = 00$, the output $Y_0=D_{in}$. Similarly if $S_1S_0 = 01$, the output $Y_1=D_{in}$, if $S_1S_0 = 10$, the output $Y_2=D_{in}$ and if $S_1S_0 = 11$, the output $Y_3=D_{in}$.

Logic diagram

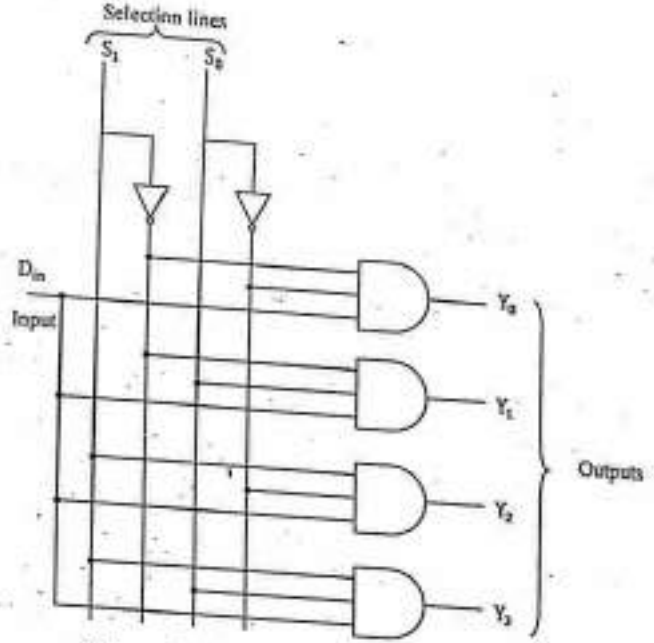


Figure 2.195 Logic diagram of 1:4 demultiplexer

The truth table of a 1:4 demultiplexer can be written as shown in table 2.35.

Selection lines		Outputs			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	D_{in}
0	1	0	0	D_{in}	0
1	0	0	D_{in}	0	0
1	1	D_{in}	0	0	0

Table 2.35 Truth table of 1:4 Demultiplexer

2.18.2 1:8 Demultiplexer

A 1:8 Demultiplexer has single input 'D_{in}', 8 outputs ($Y_0, Y_1, \ldots, Y_7$) and 3 selection lines (S_2, S_1 and S_0).

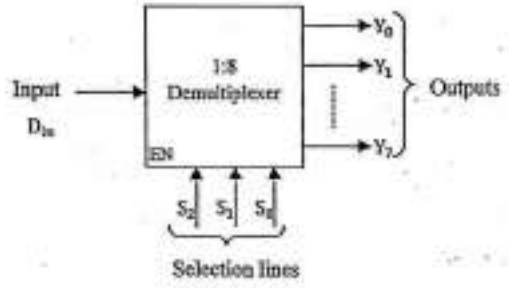


Figure 2.196 Block schematic of 1:8 Demultiplexer

When $S_2S_1S_0 = 000$, the output $Y_0=D_{in}$ and the remaining outputs are '0'.
When $S_2S_1S_0 = 001$, the output $Y_1=D_{in}$ and the remaining outputs are '0'.
When $S_2S_1S_0 = 010$, the output $Y_2=D_{in}$ and the remaining outputs are '0'.
When $S_2S_1S_0 = 011$, the output $Y_3=D_{in}$ and the remaining outputs are '0'.
When $S_2S_1S_0 = 100$, the output $Y_4=D_{in}$ and the remaining outputs are '0'.
When $S_2S_1S_0 = 101$, the output $Y_5=D_{in}$ and the remaining outputs are '0'.
When $S_2S_1S_0 = 110$, the output $Y_6=D_{in}$ and the remaining outputs are '0'.
When $S_2S_1S_0 = 111$, the output $Y_7=D_{in}$ and the remaining outputs are '0'.
The truth table of a 8:1 multiplexer can be written as

Selection lines			Outputs							
S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	D_{in}
0	0	1	0	0	0	0	0	0	D_{in}	0
0	1	0	0	0	0	0	0	D_{in}	0	0
0	1	1	0	0	0	0	D_{in}	0	0	0
1	0	0	0	0	0	D_{in}	0	0	0	0
1	0	1	0	0	D_{in}	0	0	0	0	0
1	1	0	0	D_{in}	0	0	0	0	0	0
1	1	1	D_{in}	0	0	0	0	0	0	0

Table 2.36 Truth table of 1:8 Demultiplexer

Logic diagram

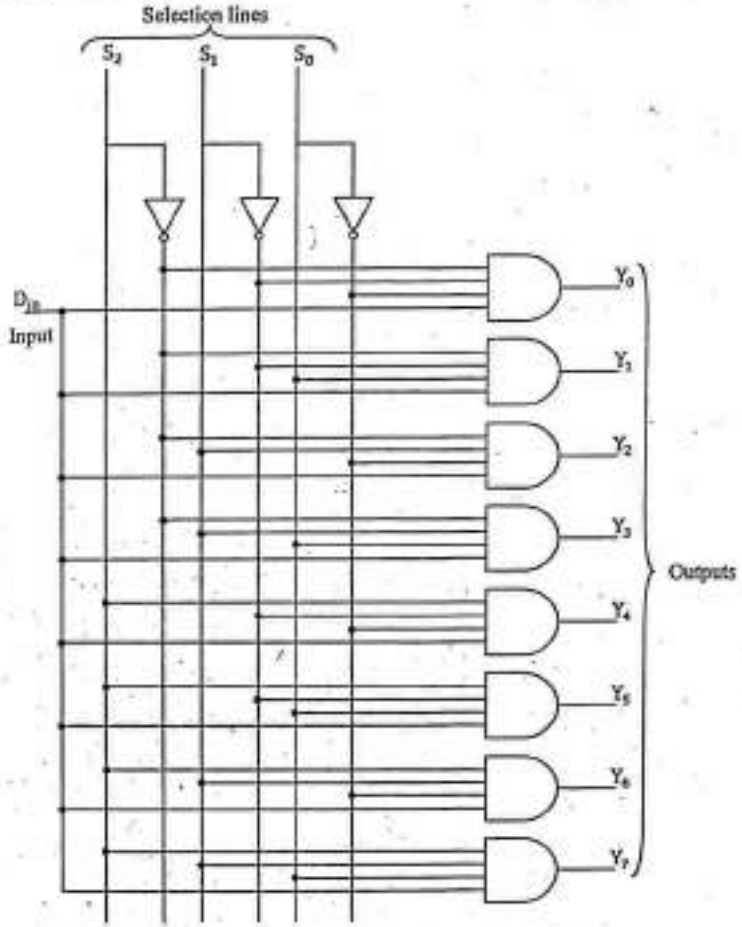


Figure 2.197 Logic diagram of 1:8 Demultiplexer

Example 2.64: Design 1:8 demultiplexer using two 1:4 demultiplexer

Solution:

A 1:8 Demultiplexer has single input 'D_{in}', 8 outputs ($Y_0, Y_1, \ldots, Y_7$) and 3 selection lines (S_2, S_1 and S_0). Two 1:4 demultiplexers are cascaded to form 1:8 demultiplexer as shown in figure 2.198.

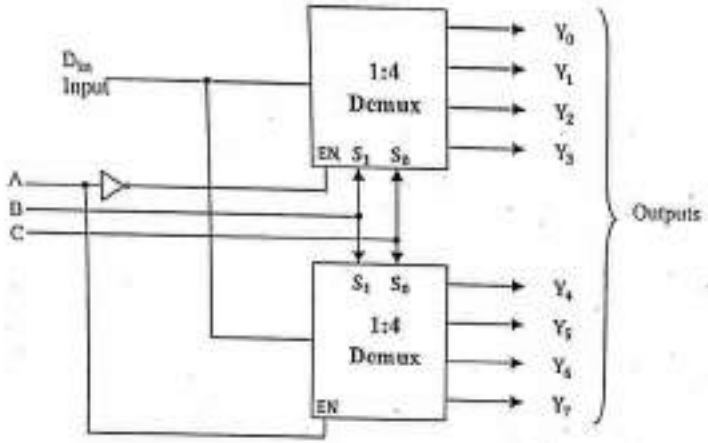


Figure 2.198 1:8 Demultiplexer using two 1:4 Demultiplexer

Example 2.65: Design a full subtractor using demultiplexer.

Solution :

A full subtractor performs the subtraction between three bits and produces the output 'Difference' and 'borrow'. Let the inputs be A, B and C.

Inputs			Outputs	
A	B	C	Borrow	Difference
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Table 2.37 Truth table for full subtractor

From the above truth table the functions for 'Borrow' and 'difference' can be written as

$$\text{Borrow} = \sum_m(1,2,3,7)$$

$$\text{Difference} = \sum_m(1,2,4,7)$$

The full subtractor can be implemented using a demultiplexer as shown in figure 2.199.

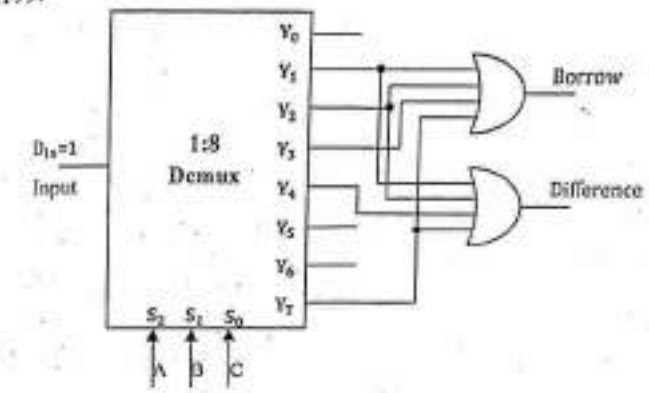


Figure 2.199 Full subtractor using Demultiplexer

2.19 DECODER

A decoder is a combinational circuit that converts 'n' number of input lines to maximum of '2^n' number of output lines. The decoders given here are n-to-m line decoders, where $m \leq 2^n$. Based on the number of outputs and inputs, decoders can be classified as.

(i). 2-to-4 line decoder.
(ii). 3-to-8 line decoder.
(iii). 4-to-16 line decoder.

2.19.1 2-to-4 line decoder

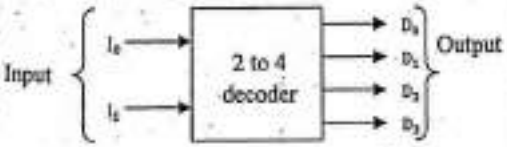


Figure 2.200 2-to-4 line decoder.

Inputs		Outputs			
I_1	I_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Table 2.38 Truth table of 2-to-4 line decoder

From the above truth table shown in table 2.38

$D_0 = \bar{I}_1\bar{I}_0$; $D_1 = \bar{I}_1 I_0$

$D_2 = I_1\bar{I}_0$; $D_3 = I_1 I_0$

Logic diagram

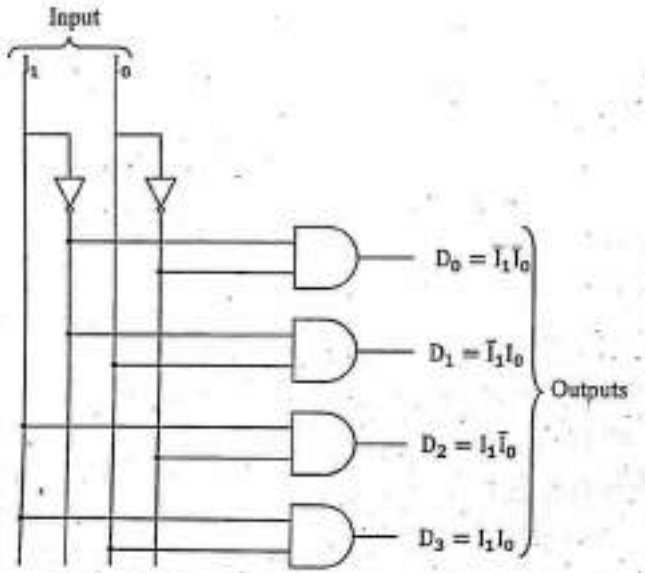


Figure 2.201 Logic diagram of a 2-to-4 line decoder

A 2 to 4 decoder consists of 2 input lines I_1 and I_0, four AND gates and four output lines D_3, D_2, D_1 and D_0.

The output D_0 is active when the inputs $I_1 = I_0 = 0$. The output D_1 is active when the inputs $I_1 = 0$ and $I_0 = 1$. The output D_2 is active when the inputs $I_1 = 1$ and $I_0 = 0$. The output D_3 is active when the inputs $I_1 = 1$ and $I_0 = 1$.

2.19.2 3-to-8 line decoder (Binary to Octal decoder)

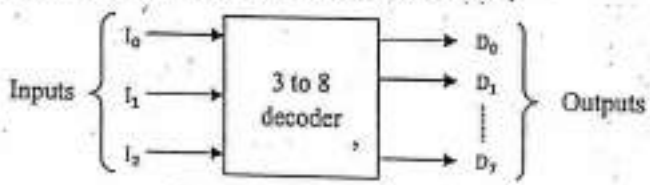


Figure 2.202 3-to-8 line decoder.

Inputs			Outputs							
I_2	I_1	I_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Table 2.39 Truth table of 3-to-8 line decoder

Logic diagram

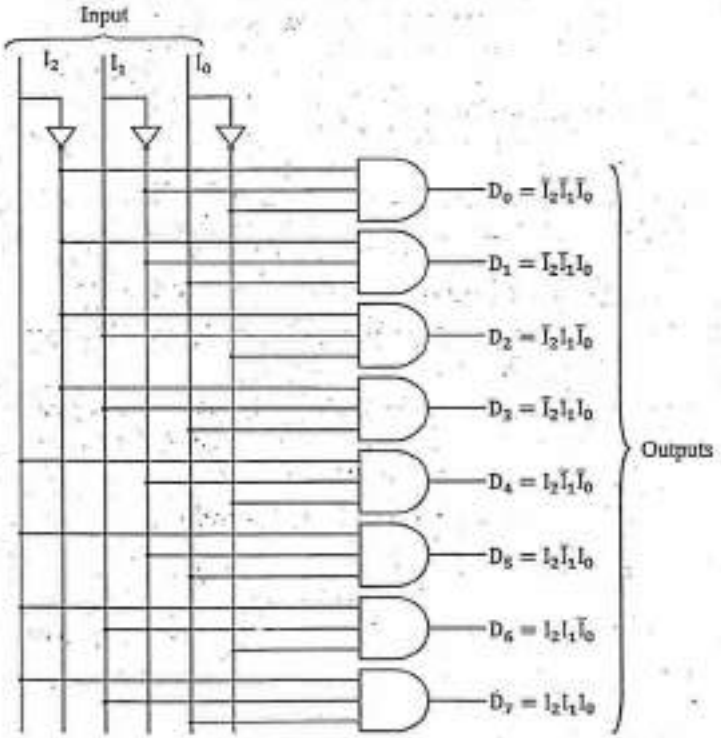


Figure 2.203 Logic diagram of a 3-to-8 line decoder

In 3 to 8 decoder the number of inputs is 3. Hence the number of outputs is $2^n = 8$.

$D_0 = \bar{I}_2\bar{I}_1\bar{I}_0$; $D_1 = \bar{I}_2\bar{I}_1 I_0$;

$D_2 = \bar{I}_2 I_1\bar{I}_0$; $D_3 = \bar{I}_2 I_1 I_0$

$D_4 = I_2\bar{I}_1\bar{I}_0$; $D_5 = I_2\bar{I}_1 I_0$;

$D_6 = I_2 I_1\bar{I}_0$; $D_7 = I_2 I_1 I_0$

2.19.3 4-to-16 line decoder (Binary to Hexadecimal decoder)

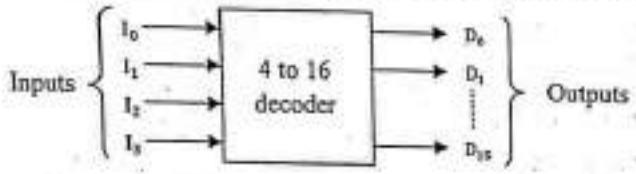


Figure 2.204 4-to-16 line decoder.

Inputs				Outputs															
I_3	I_2	I_1	I_0	D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.40 4-to-16 line decoder

From the truth table shown in table 2.40, we can write,

$D_0 = \bar{I}_3\bar{I}_2\bar{I}_1\bar{I}_0$; $D_1 = \bar{I}_3\bar{I}_2\bar{I}_1 I_0$; $D_2 = \bar{I}_3\bar{I}_2 I_1\bar{I}_0$; $D_3 = \bar{I}_3\bar{I}_2 I_1 I_0$;

$D_4 = \bar{I}_3 I_2\bar{I}_1\bar{I}_0$; $D_5 = \bar{I}_3 I_2\bar{I}_1 I_0$; $D_6 = \bar{I}_3 I_2 I_1\bar{I}_0$; $D_7 = \bar{I}_3 I_2 I_1 I_0$;

$D_8 = I_3\bar{I}_2\bar{I}_1\bar{I}_0$; $D_9 = I_3\bar{I}_2\bar{I}_1 I_0$; $D_{10} = I_3\bar{I}_2 I_1\bar{I}_0$; $D_{11} = I_3\bar{I}_2 I_1 I_0$;

$D_{12} = I_3 I_2\bar{I}_1\bar{I}_0$; $D_{13} = I_3 I_2\bar{I}_1 I_0$; $D_{14} = I_3 I_2 I_1\bar{I}_0$; $D_{15} = I_3 I_2 I_1 I_0$;

Logic diagram

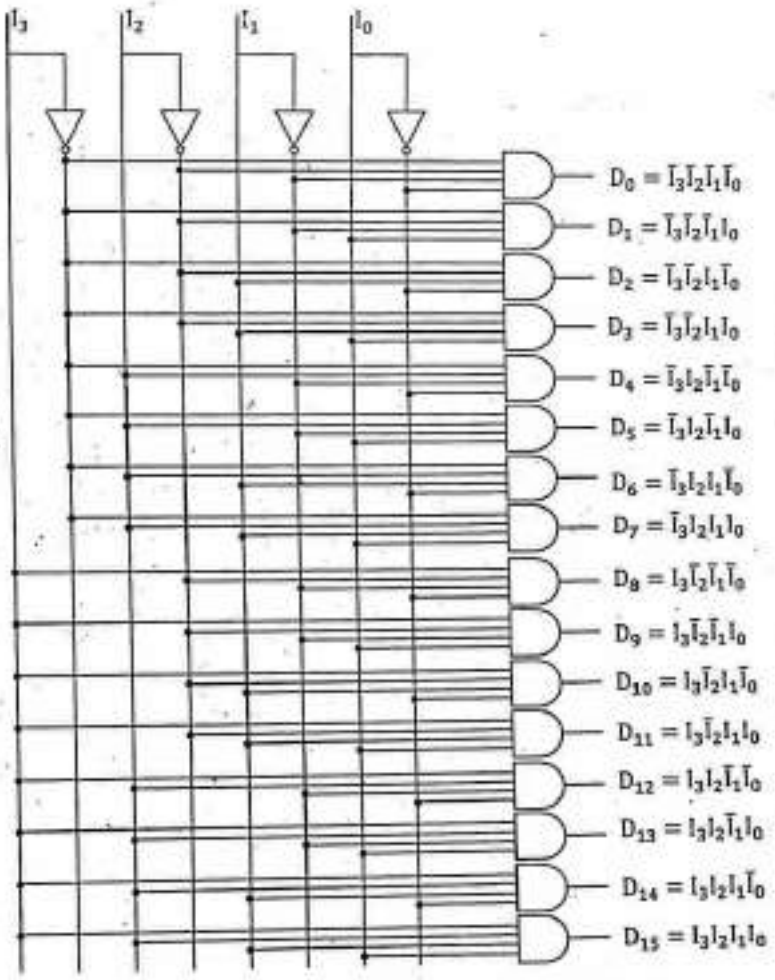


Figure 2.205 Logic diagram of 4-to-16 line decoder

2.19.4 BCD to seven segment Display decoder

The BCD to 7-segment decoder converts the input BCD code into its equivalent signal suitable for driving a seven segment display.

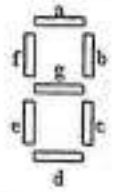
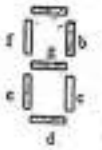
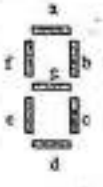


Figure 2.206 Seven segment display

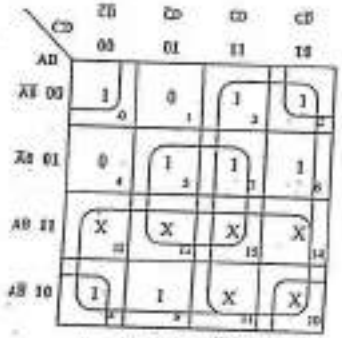
Digit	Display	Glowing segments
0		a, b, c, d, e, f
1		b, c
2		a, b, d, e, g
3		a, b, c, d, g

4		b, c, f, g
5		a, c, d, f, g
6		a, c, d, e, f, g
7		a, b, c
8		a, b, c, d, e, f, g
9		a, b, c, d, f, g

Digit	BCD input				Segment output						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

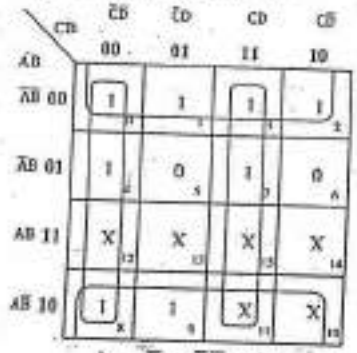
Table 2.41 Truth table

K-map simplification for 'a'



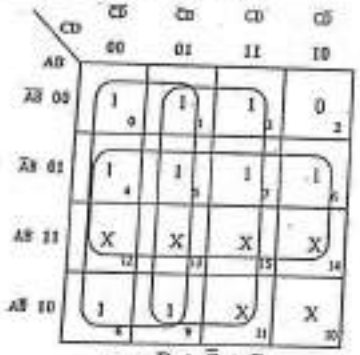
$a = A + C + \overline{B}\overline{D} + BD$

K-map simplification for 'b'



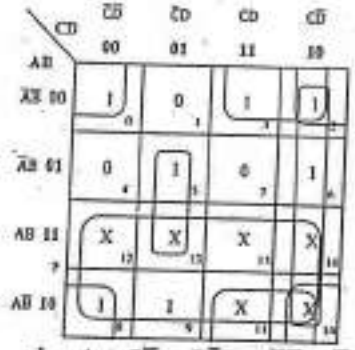
$b = \overline{B} + \overline{C}\overline{D} + CD$

K-map simplification for 'c'



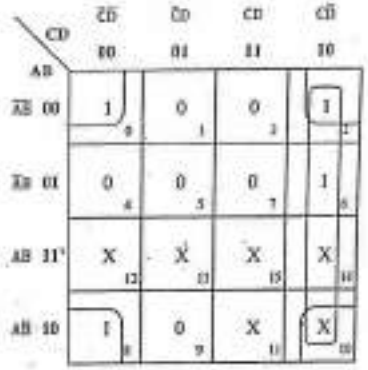
$c = B + \overline{C} + D$

K-map simplification for 'd'



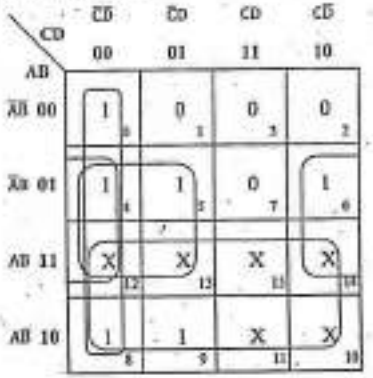
$d = A + C\overline{D} + B\overline{C}D + \overline{B}\overline{D} + \overline{B}C$

K-map simplification for 'e'



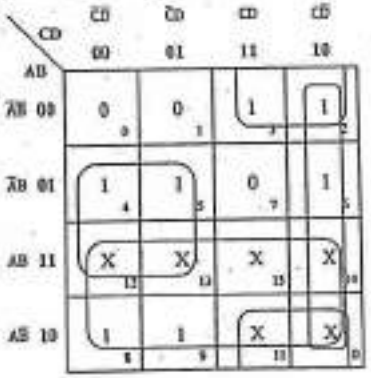
$e = C\overline{D} + \overline{B}\overline{D}$

K-map simplification for 'f'



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

K-map simplification for 'g'



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

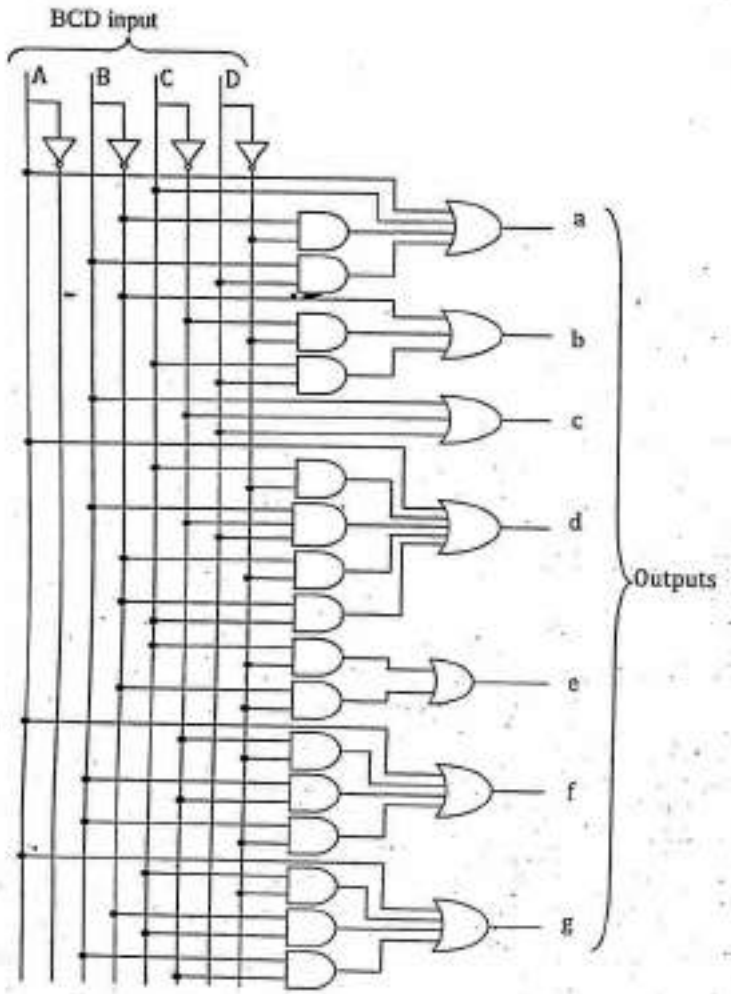


Figure 2.207 Logic diagram of BCD to 7-segment decoder

2.19.5 BCD to decimal decoder

BCD to decimal decoders has four inputs to represent the BCD numbers from 0000 to 1001. It has 10 outputs represented as $D_0, D_1 \ldots\ldots D_9$. The four bit BCD input (b_3, b_2, b_1 and b_0) is decoded to activate one of the ten outputs. The truth table of a BCD to decimal decoder is shown in table 2.42.

BCD inputs				Outputs									
b_3	b_2	b_1	b_0	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0	0

Table 2.42 Truth table of BCD to decimal decoder

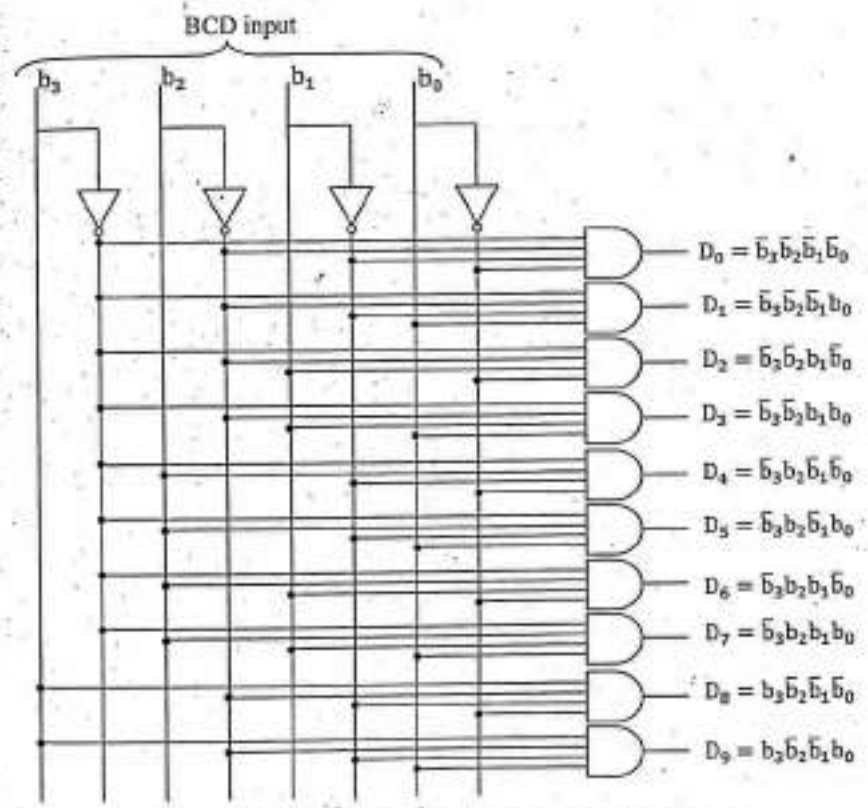


Figure 2.208 Logic diagram of BCD to decimal decoder

From the truth table shown in table 2.42, we can write

$D_0 = \bar{b}_3\bar{b}_2\bar{b}_1\bar{b}_0$; $D_1 = \bar{b}_3\bar{b}_2\bar{b}_1 b_0$; $D_2 = \bar{b}_3\bar{b}_2 b_1\bar{b}_0$

$D_3 = \bar{b}_3\bar{b}_2 b_1 b_0$; $D_4 = \bar{b}_3 b_2\bar{b}_1\bar{b}_0$; $D_5 = \bar{b}_3 b_2\bar{b}_1 b_0$

$D_6 = \bar{b}_3 b_2 b_1\bar{b}_0$; $D_7 = \bar{b}_3 b_2 b_1 b_0$; $D_8 = b_3\bar{b}_2\bar{b}_1\bar{b}_0$

$D_9 = b_3\bar{b}_2\bar{b}_1 b_0$

2.19.6 Applications of decoders

Decoders are used in counter systems, analog to digital converters. It can be used to drive a seven segment display system.

2.20 ENCODER

Encoder performs the inverse operation of a decoder. An encoder is a combinational circuit that converts '2^n' number of input lines to 'n' number of output lines. Based on the number of outputs and inputs, encoders can be classified as.

i. 4-to-2 line encoder.
ii. 8-to-3 line encoder.
iii. 16-to-4 line encoder.

2.20.1 4-to-2 line encoder

4-to-2 line encoder has 4 input lines and 2 output lines. Based on the 4 input it produces 2 bit output.

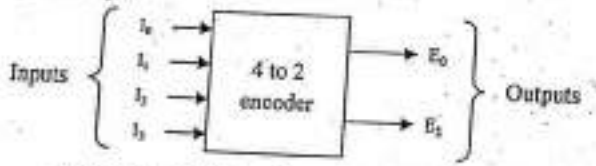


Figure 2.209 Block schematic of 4-to 2 line Encoder

Let the 4 bit inputs be I_0, I_1, I_2 and I_3. The two bit outputs be E_0 and E_1. The truth table of 4-to-2 line encoder is shown in table 2.43.

Inputs				Outputs	
I_3	I_2	I_1	I_0	E_1	E_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Table 2.43 Truth table of 4-to-2 line Encoder

From the above truth table the logic expression can be written as

(LSB) $E_0 = I_1 + I_3$

(MSB) $E_1 = I_2 + I_3$

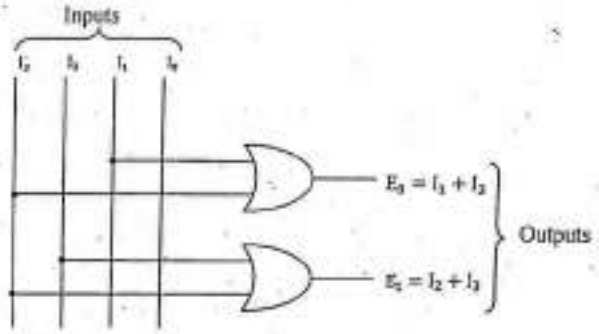


Figure 2.210 Logic diagram of 4-to-2 line encoder.

2.20.2 8-to-3 line encoder (Octal to Binary encoder)

It have eight inputs (I_0, I_1,..... I_7) and three outputs (E_0, E_1, E_2). Based on the eight inputs it produces a 3 bit output. The truth table of 8-to-3 line encoder is shown in table 2.44.

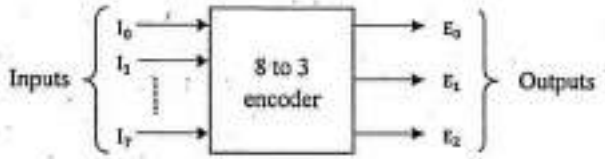


Figure 2.211 Block schematic of 8-to-3 line Encoder

Inputs								Outputs		
I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	E_2	E_1	E_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Table 2.44 Truth table of 8-to-3 line Encoder

From the above truth table, the logic expression for E_0, E_1 and E_2 can be written as

(LSB) $E_0 = I_1 + I_3 + I_5 + I_7$

$E_1 = I_2 + I_3 + I_6 + I_7$

(MSB) $E_2 = I_4 + I_5 + I_6 + I_7$

Logic diagram

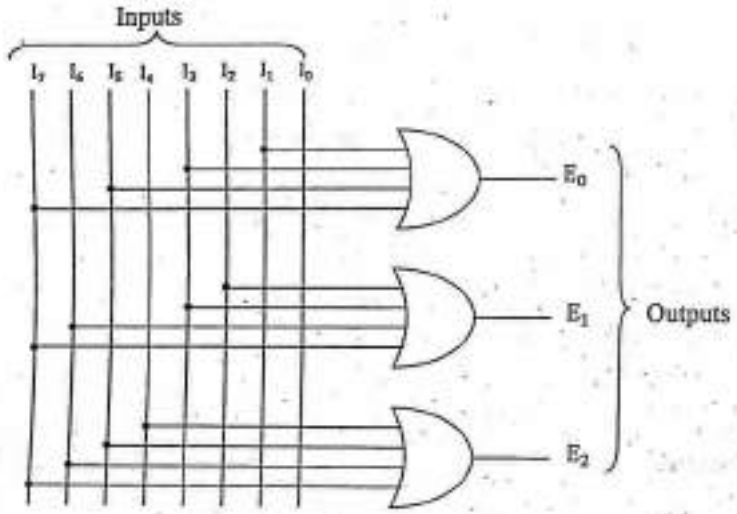


Figure 2.212 Logic diagram of 8-to-3 line encoder.

2.20.3 16-to-4 line encoder (Hexadecimal to Binary encoder)

It have 16 inputs (I_0, I_1..... I_{15}) and 4 outputs (E_0, E_1, E_2, E_3). Based on the 16 inputs it produces a 4 bit output.

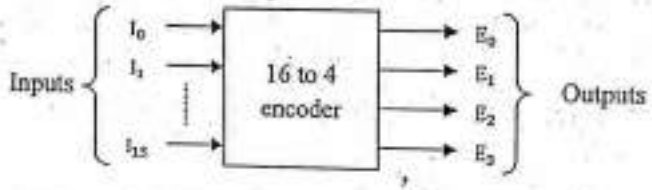


Figure 2.213 Block schematic of 16-to-4 line Encoder

The truth table of 16-to-4 line encoder is shown in table 2.45

Inputs																Outputs			
I_{15}	I_{14}	I_{13}	I_{12}	I_{11}	I_{10}	I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Table 2.45 Truth table of 16-to-4 line encoder

Logic diagram

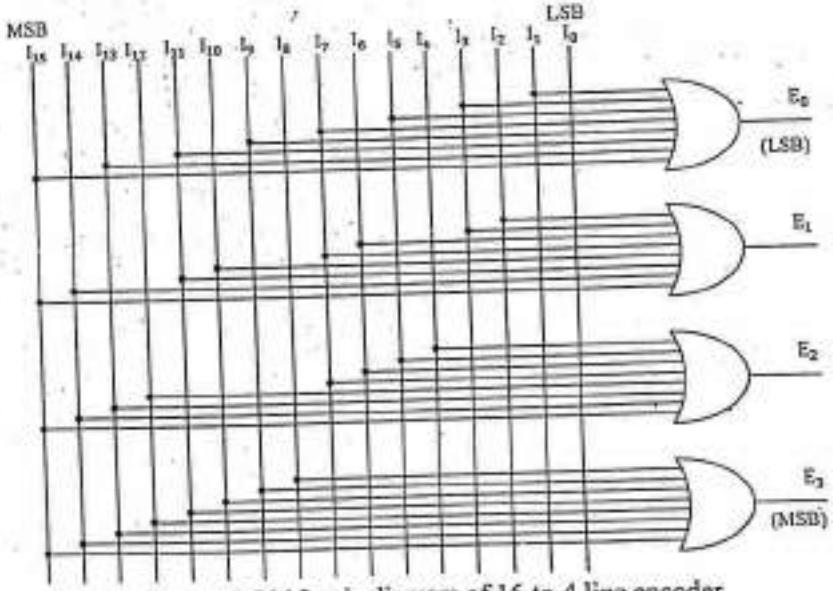


Figure 2.214 Logic diagram of 16-to-4 line encoder

From the truth table of 16-to-4 line encoder, the logic expression for E_0, E_1, E_2 and E_3 can be written as,

(LSB) $E_0 = I_1 + I_3 + I_5 + I_7 + I_9 + I_{11} + I_{13} + I_{15}$

$E_1 = I_2 + I_3 + I_6 + I_7 + I_{10} + I_{11} + I_{14} + I_{15}$

$E_2 = I_4 + I_5 + I_6 + I_7 + I_{12} + I_{13} + I_{14} + I_{15}$

(MSB) $E_3 = I_8 + I_9 + I_{10} + I_{11} + I_{12} + I_{13} + I_{14} + I_{15}$

2.20.4 Decimal to BCD encoder

Decimal to BCD encoder performs the inverse operation of a BCD to decimal decoder. It has 10 inputs D_0, D_1..... D_9 and four outputs b_3, b_2, b_1 and b_0. The truth table for a decimal to BCD encoder is shown in table 2.46.

Inputs										Outputs			
D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1

Table 2.46 Truth table of Decimal to BCD Encoder

From the above truth table, the logic expression for b_3, b_2, b_1 and b_0 can be written as

(LSB) $b_0 = D_1 + D_3 + D_5 + D_7 + D_9$

$b_1 = D_2 + D_3 + D_6 + D_7$

$b_2 = D_4 + D_5 + D_6 + D_7$

(MSB) $b_3 = D_8 + D_9$

Logic diagram

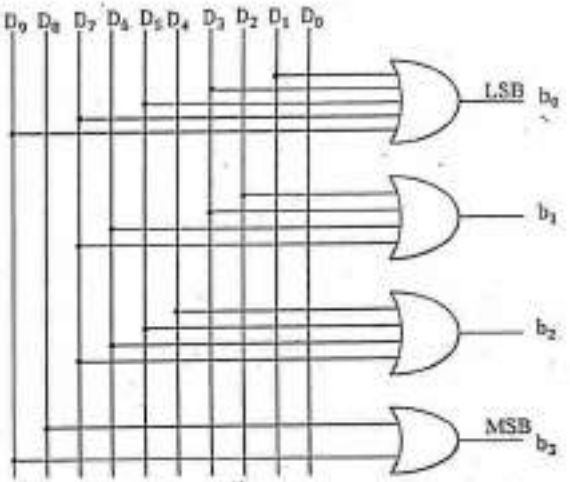


Figure 2.215 Logic diagram of decimal to BCD encoder

2.20.5 Priority encoder

Inputs				Outputs		
D_3 (highest priority)	D_2	D_1	D_0 (lowest priority)	Y_1 (MSB)	Y_0 (LSB)	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	0	1	1	0	1	1
0	1	0	0	1	0	1
0	1	0	1	1	0	1
0	1	1	0	1	0	1
0	1	1	1	1	0	1
1	0	0	0	1	1	1
1	0	0	1	1	1	1
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

Table 2.47 Truth table of Priority Encoder

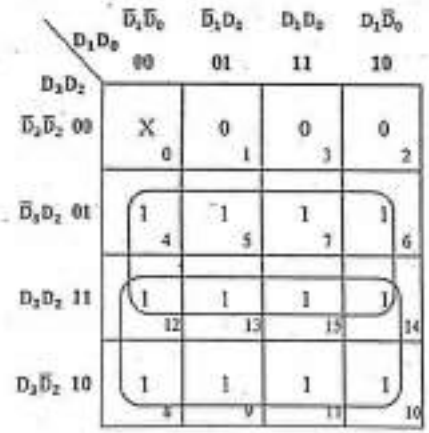
A priority encoder is an encoder circuit that includes the priority function. The function of the priority encoder is such that if two or more inputs are equal to 1, the input having the highest priority will take precedence.

The truth table of a priority encoder is shown in table 2.47. In priority encoder, D_3 is assigned with highest priority and D_0 is assigned with lowest priority. Hence D_2 has less priority than D_3, D_1 has less priority than D_3 and D_2, D_0 has less priority than D_3, D_2 and D_1.

The outputs are Y_1, Y_0 and V. Here V is known to be a valid output indicator. If V=0 it indicates that no priority is assigned to anyone of the inputs. For example if $D_3 = D_2 = D_1 = D_0 = 0$ indicates that no priority is assigned to D_3, D_2, D_1 or D_0. Hence V=0.

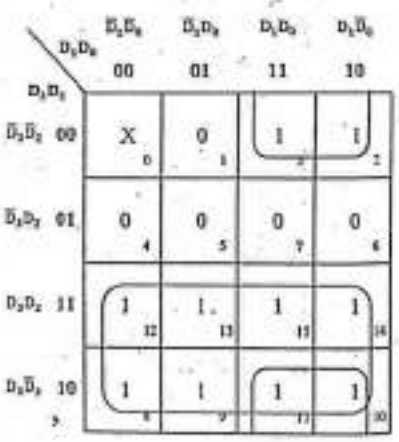
If V=1, it indicates that priority is assigned to one of the inputs. For example if $D_3 = D_2 = D_0 = 0$ and $D_1 = 1$, it indicates that priority is assigned to D_1, hence $Y_1Y_0 = 01$ and V=1. If $D_3 = D_2 = D_1 = D_0 = 1$, it indicates that priority is assigned to all the inputs, but we have to consider the input with high priority (here D_3 have high priority), hence $Y_1Y_0 = 11$ and V=1.

K-map for Y_1



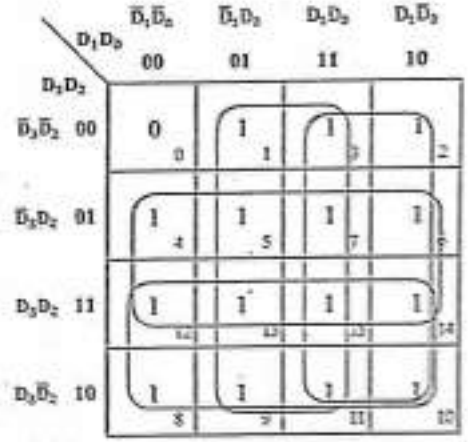
$Y_1 = D_2 + D_3$

K-map for Y_0



$Y_0 = D_3 + \bar{D}_2D_1$

K-map for V



$V = D_3 + D_2 + D_1 + D_0$

Logic diagram

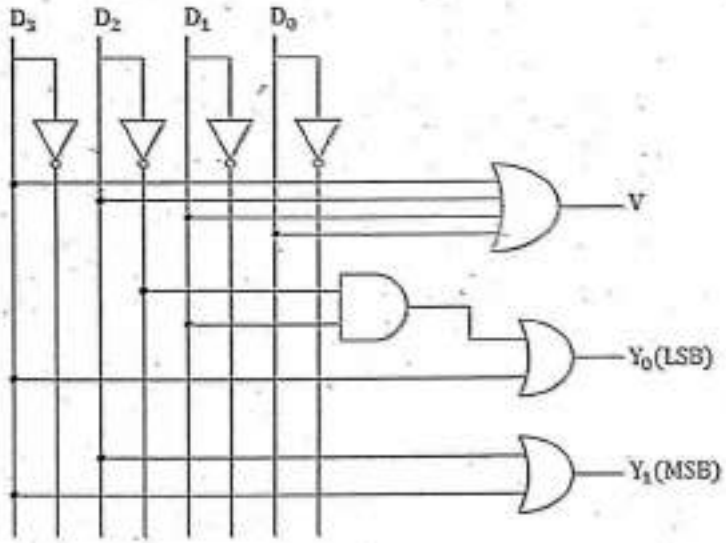


Figure 2.216 Logic diagram of Priority Encoder

2.21 PARITY GENERATOR / PARITY CHECKER

In the transmission of binary information, a parity bit is used for detecting the errors in the received binary information. A parity bit is an extra bit included with the binary message at the transmitting section to make the number of 1's either odd or even. The message including the parity bit is then transmitted. The received binary message along with the parity bit is checked 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 and the circuit that checks the parity in the receiver section is called a parity checker. The parity generator circuit and parity checker circuit is designed below.

In even parity the added parity bit will make the total number of 1's an even amount. In odd parity the added parity bit will make the total number of 1's an odd amount.

3-bit message			Parity bit (P)	
A	B	C	Odd parity bit	Even parity bit
0	0	0	1	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Table 2.48 Truth table of parity generator

Here we have designed even parity generator

K-map for even parity bit

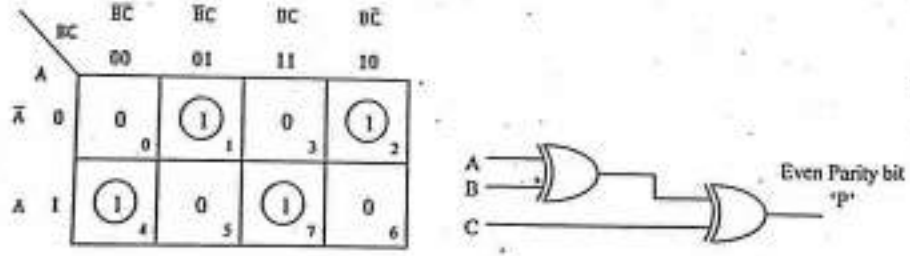


Figure 2.217 Logic diagram for even parity generator

The logic expression for even parity bit is given by,

$$P = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$
$$= \bar{A}(\bar{B}C + B\bar{C}) + A(\overline{B\bar{C}} + BC)$$
$$= \bar{A}(B \oplus C) + A(B \odot C)$$
$$= \bar{A}(B \oplus C) + A(\overline{B \oplus C})$$
$$= A \oplus (B \oplus C) = A \oplus B \oplus C$$

Received bits along with parity bit				Parity error check bit
A	B	C	D	PEC
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

Table 2.49 Truth table of Even parity checker

The three bit message together with the parity bits is transmitted to the destination. In the receiver section the received message together with the parity bits are fed to the parity checker input. The parity checker circuit checks for possible errors in the transmission. Since the information was transmitted with even parity, the four bits received must have an even number of 1's. If the received four bits have an odd number of 1's, it indicates that an error have occurred during the transmission. The output of Parity checker is denoted by PEC (Parity Error Check). If the parity error check bit is 1, then error has been occurred. If the parity error check bit is 0, then there is no error in the received information.

K-map for PEC

AB \ CD	$\overline{C}\overline{D}$ 00	$\overline{C}D$ 01	CD 11	$C\overline{D}$ 10
$\overline{A}\overline{B}$ 00	0 (0)	1 (1)	0 (3)	1 (2)
$\overline{A}B$ 01	1 (4)	0 (5)	1 (7)	0 (6)
AB 11	0 (12)	1 (13)	0 (15)	1 (14)
$A\overline{B}$ 10	1 (8)	0 (9)	1 (11)	0 (10)

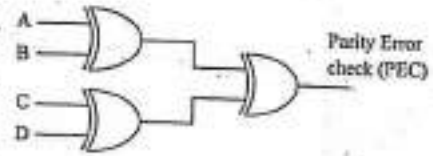


Figure 2.218 Logic diagram of parity error check (PEC)

The logic expression for parity error checker output is given by,

$$PEC = \overline{A}\overline{B}CD + \overline{A}\overline{B}C\overline{D} + \overline{A}B\overline{C}\overline{D} + \overline{A}BCD + AB\overline{C}D + ABC\overline{D} + A\overline{B}\overline{C}\overline{D} + A\overline{B}CD$$

$$PEC = \overline{A}\overline{B}(\overline{C}D + C\overline{D}) + \overline{A}B(\overline{C}\overline{D} + CD) + AB(\overline{C}D + C\overline{D}) + A\overline{B}(\overline{C}\overline{D} + CD)$$

$$PEC = \overline{A}\overline{B}(C \oplus D) + \overline{A}B(\overline{C \oplus D}) + AB(C \oplus D) + A\overline{B}(\overline{C \oplus D})$$

$$PEC = (C \oplus D)(\overline{A}\overline{B} + AB) + (\overline{C \oplus D})(\overline{A}B + A\overline{B})$$

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

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

2.22 CODE CONVERTERS

The digital systems use a wide variety of binary codes such as BCD, Excess-3, Gray codes etc. A code conversion circuit must be inserted between the two systems, if each system uses different codes for the same information.

This section explains the following code converters.

1. Binary to BCD code converter.
2. BCD to Binary code converter.
3. BCD to Excess-3 code converter.
4. Excess-3 to BCD code converter.
5. Binary to Gray code converter.
6. Gray to Binary code converter.
7. BCD to Gray code converter.
8. Gray to BCD code converter.

2.22.1 Binary to BCD code converter

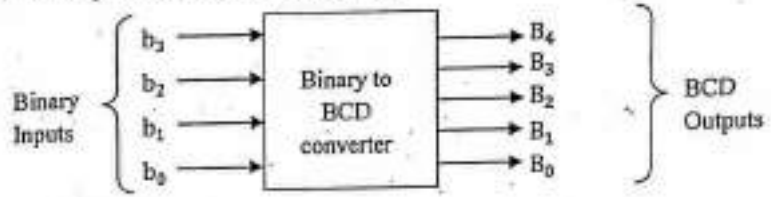
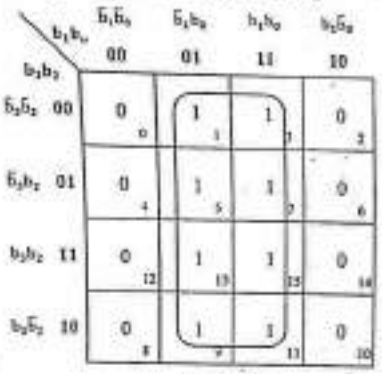


Figure 2.219 Block diagram of Binary to BCD converter

Input Binary code				Output BCD code				
b_3	b_2	b_1	b_0	B_4	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	1	0
0	0	1	1	0	0	0	1	1
0	1	0	0	0	0	1	0	0
0	1	0	1	0	0	1	0	1
0	1	1	0	0	0	1	1	0
0	1	1	1	0	0	1	1	1
1	0	0	0	0	1	0	0	0
1	0	0	1	0	1	0	0	1
1	0	1	0	1	0	0	0	0
1	0	1	1	1	0	0	0	1
1	1	0	0	1	0	0	1	0
1	1	0	1	1	0	0	1	1
1	1	1	0	1	0	1	0	0
1	1	1	1	1	0	1	0	1

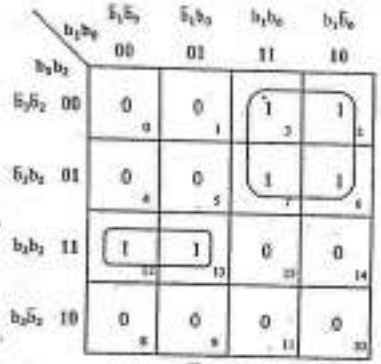
Table 2.50 Truth table for binary to BCD converter

K-map simplification for B_0



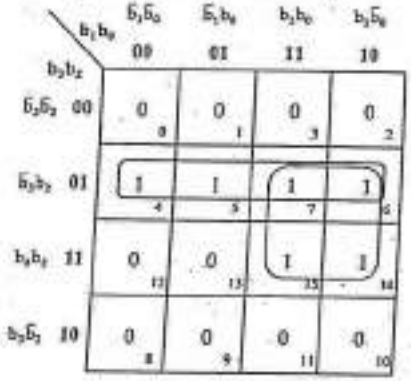
$B_0 = b_0$

K-map simplification for B_1



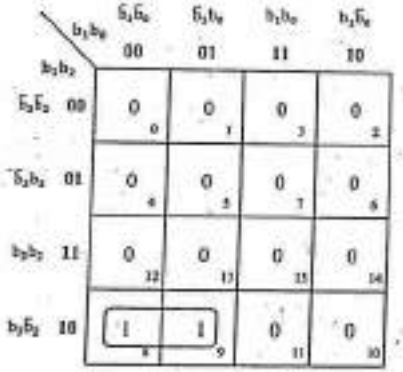
$B_1 = \bar{b}_3 b_1 + b_3 b_2 \bar{b}_1$

K-map simplification for B_2



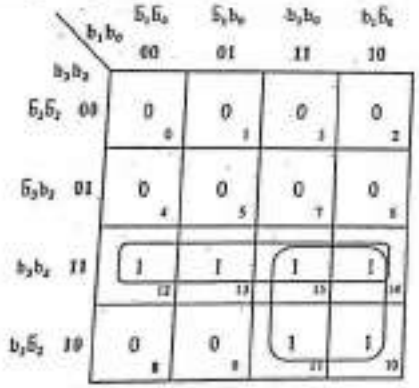
$B_2 = \bar{b}_3 b_2 + b_2 b_1$

K-map simplification for B_3



$B_3 = b_3 \bar{b}_2 \bar{b}_1$

K-map simplification for B_4



$B_4 = b_3 b_2 + b_3 b_1$

Logic diagram

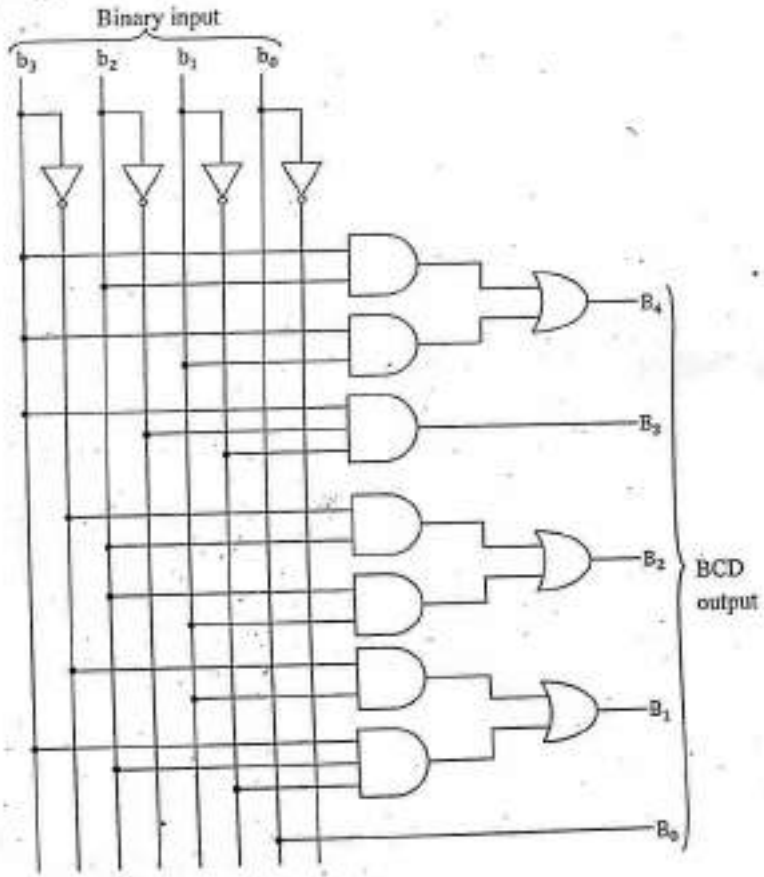


Figure 2.220 Logic diagram of Binary to BCD converter

2.22.2 BCD to Binary code converter

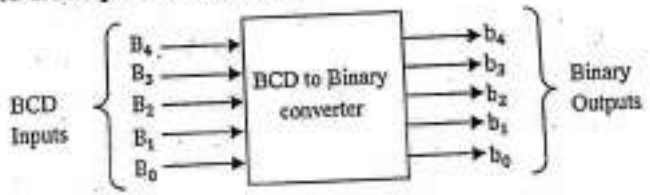


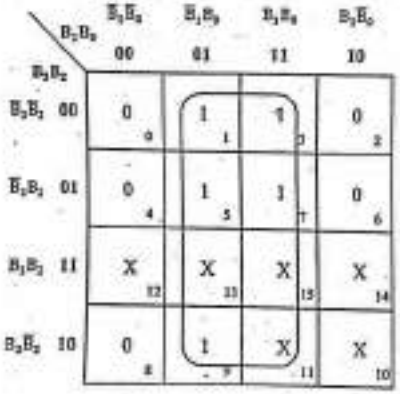
Figure 2.221 Block diagram of BCD to Binary converter

Input BCD code					Output Binary code				
B_4	B_3	B_2	B_1	B_0	b_4	b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	0	0	1	0
0	0	0	1	1	0	0	0	1	1
0	0	1	0	0	0	0	1	0	0
0	0	1	0	1	0	0	1	0	1
0	0	1	1	0	0	0	1	1	0
0	0	1	1	1	0	0	1	1	1
0	1	0	0	0	0	1	0	0	0
0	1	0	0	1	0	1	0	0	1
1	0	0	0	0	0	1	0	1	0
1	0	0	0	1	0	1	0	1	1
1	0	0	1	0	0	1	1	0	0
1	0	0	1	1	0	1	1	0	1
1	0	1	0	0	0	1	1	1	0
1	0	1	0	1	0	1	1	1	1
1	0	1	1	0	1	0	0	0	0
1	0	1	1	1	1	0	0	0	1
1	1	0	0	0	1	0	0	1	0
1	1	0	0	1	1	0	0	1	1

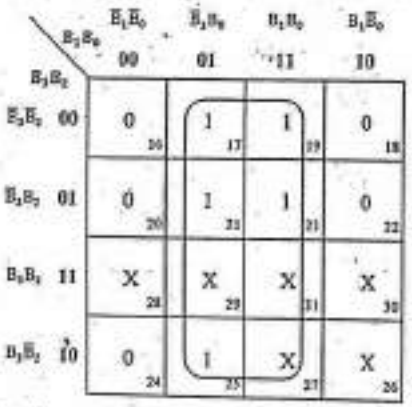
Table 2.51 Truth table for BCD to Binary converter

K-map simplification for b_0

$B_4 = 0$



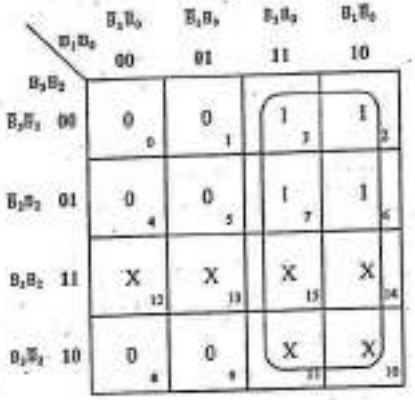
$B_4 = 1$



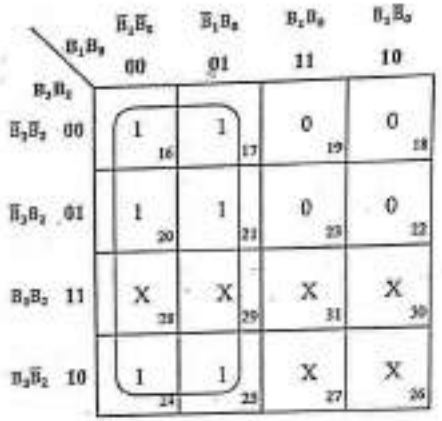
$$b_0 = B_0$$

K-map simplification for b_1

$B_4 = 0$



$B_4 = 1$

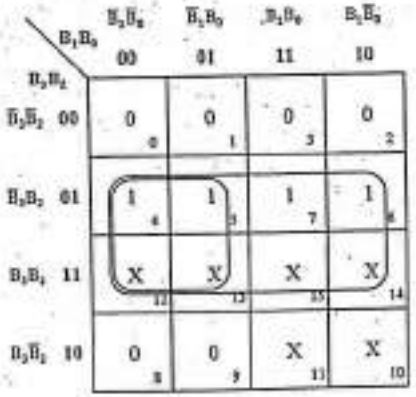


$$b_1 = B_1\overline{B}_4 + \overline{B}_1B_4$$

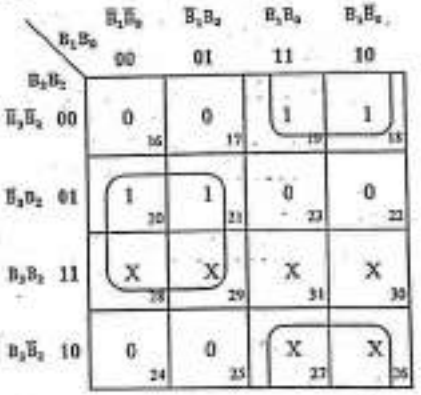
$$b_1 = B_1 \oplus B_4$$

K-map simplification for b_2

$B_4 = 0$



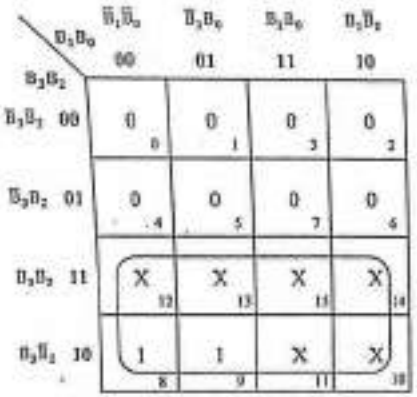
$B_4 = 1$



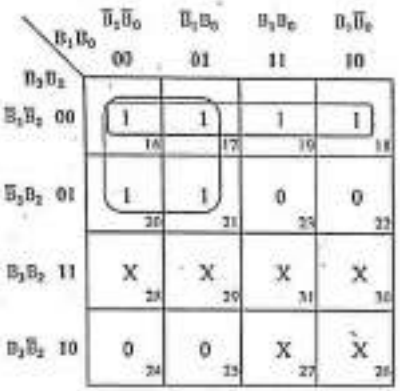
$$b_2 = \overline{B}_4B_2 + B_2\overline{B}_1 + B_4\overline{B}_2B_1$$

K-map simplification for b_3

$B_4 = 0$



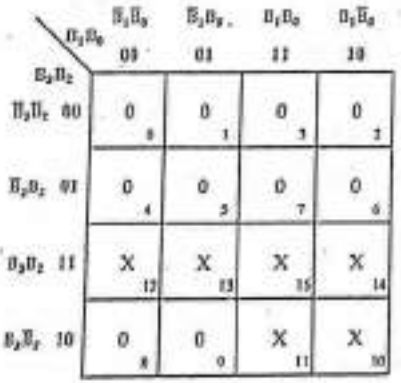
$B_4 = 1$



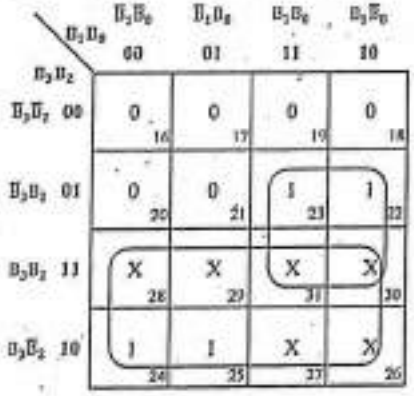
$$b_3 = \overline{B}_4B_3 + B_4\overline{B}_3\,\overline{B}_2 + B_4\overline{B}_2\,\overline{B}_1$$

K-map simplification for b_4

$B_4 = 0$



$B_4 = 1$



$$b_4 = B_4B_3 + B_4B_2B_1'$$

Logic diagram

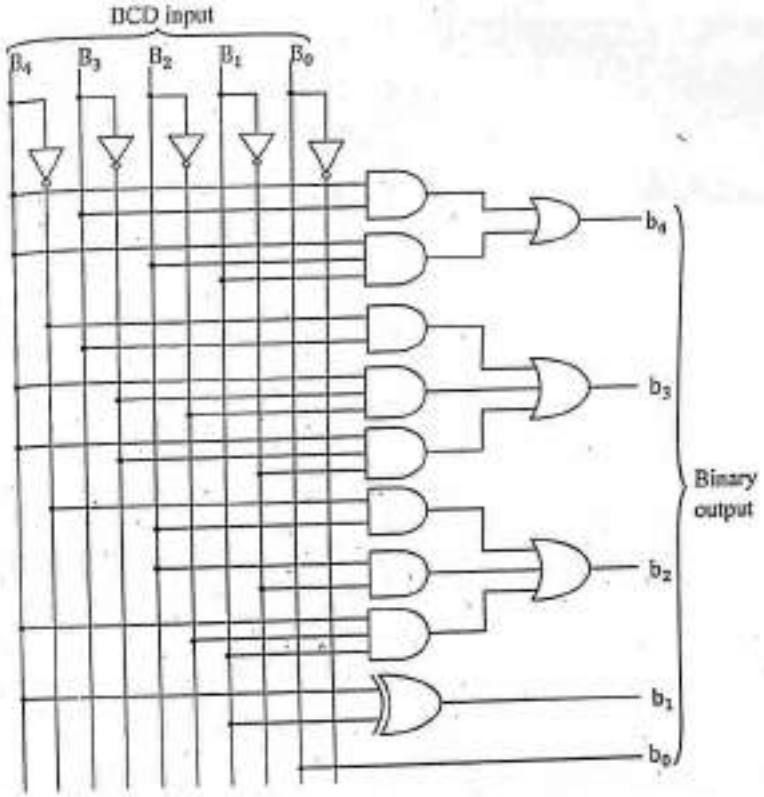


Figure 2.222 Logic diagram of BCD to Binary converter

2.22.3 BCD to Excess-3 code converter

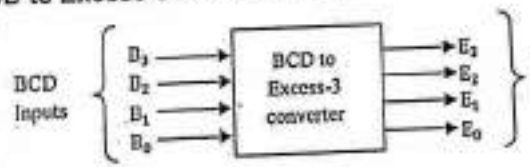
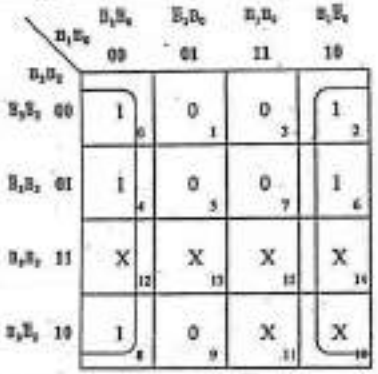


Figure 2.223 Block diagram of BCD to Excess-3 code converter

Input BCD code				Output Excess-3 code			
B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

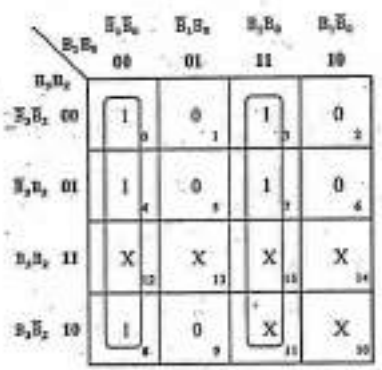
Table 2.52 Truth table for BCD to Excess-3 converter

K-map simplification for E_0



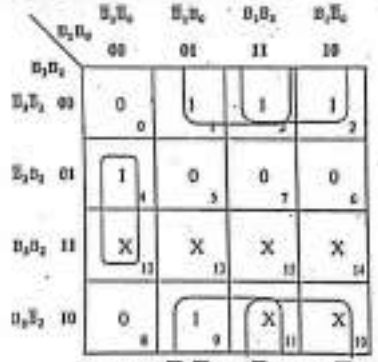
$E_0 = \overline{B_0}$

K-map simplification for E_1



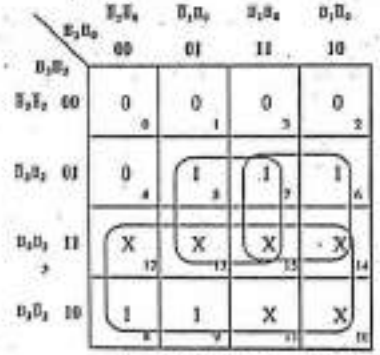
$E_1 = \overline{B_1}B_0 + B_1\overline{B_0} = B_1 \odot B_0$

K-map simplification for E_2



$E_2 = B_2\overline{B_1}\,\overline{B_0} + \overline{B_2}B_0 + \overline{B_2}B_1$

K-map simplification for E_3



$E_3 = B_3 + B_2B_0 + B_2B_1$

Logic diagram

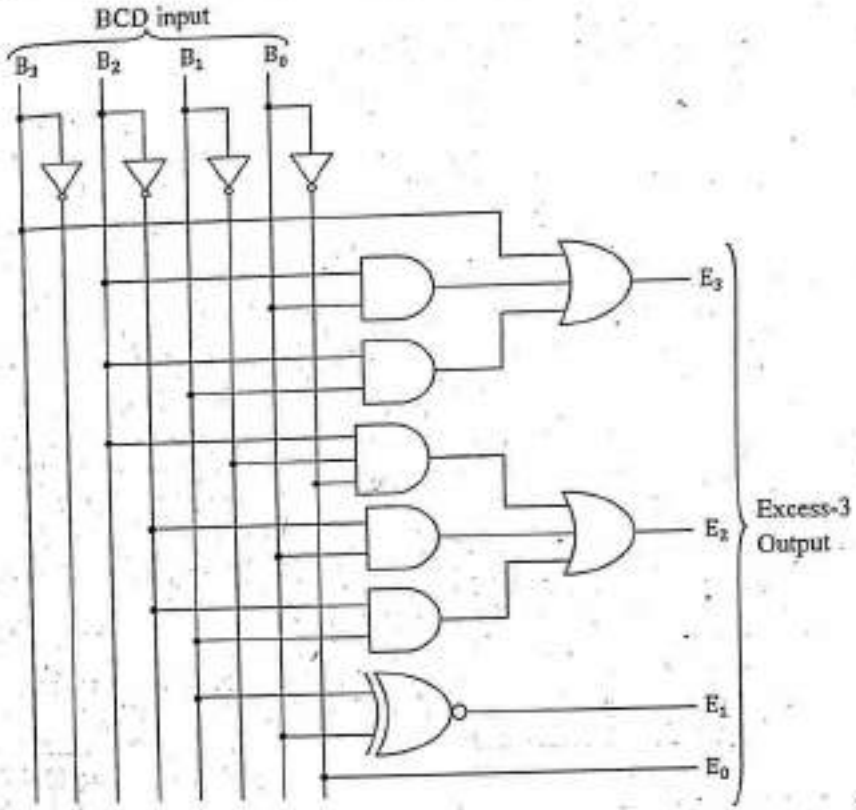


Figure 2.224 Logic diagram of BCD to Excess-3 converter

2.22.4 Excess-3 to BCD code converter

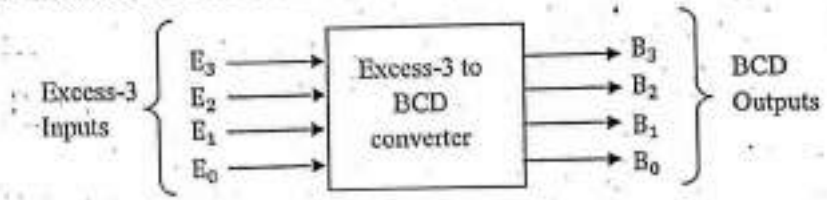
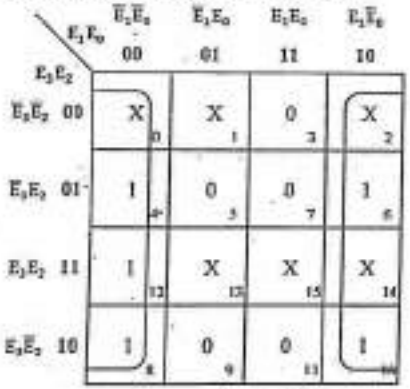


Figure 2.225 Block diagram of Excess-3 to BCD code converter

Input Excess-3 code				Output BCD code			
E_3	E_2	E_1	E_0	B_3	B_2	B_1	B_0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	0
1	0	0	0	0	1	0	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	1	1
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	1

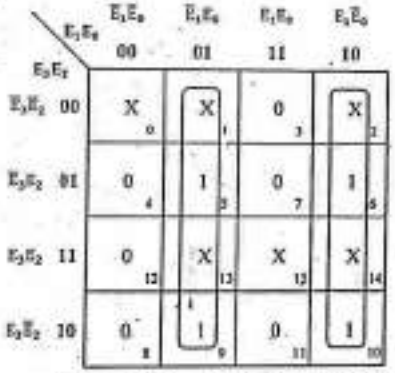
Table 2.53 Truth table for Excess-3 to BCD converter

K-map simplification for B_0



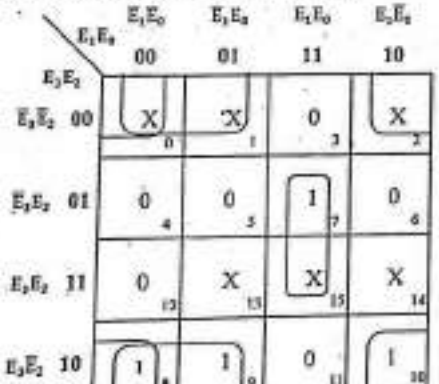
$$B_0 = \overline{E}_0$$

K-map simplification for B_1



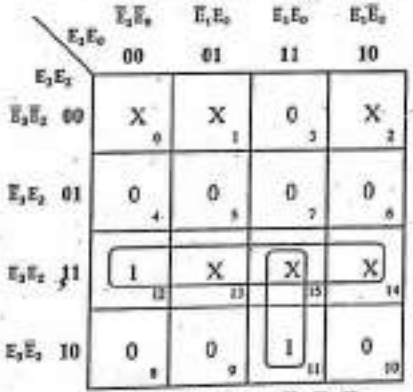
$$B_1 = \overline{E}_1 E_0 + E_1 \overline{E}_0 = E_1 \oplus E_0$$

K-map simplification for B_2



$$B_2 = \overline{E}_2 \overline{E}_1 + \overline{E}_2 \overline{E}_0 + E_2 E_1 E_0$$

K-map simplification for B_3



$$B_3 = E_3 E_2 + E_3 E_1 E_0$$

Logic diagram

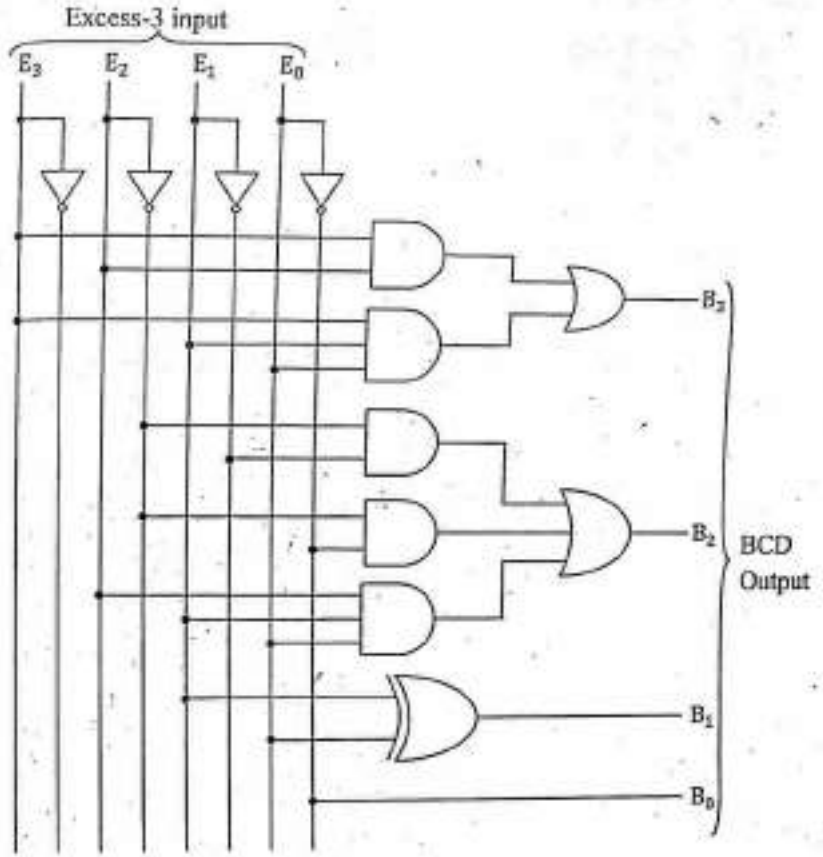


Figure 2.226 Logic diagram of Excess-3 to BCD converter

2.22.5 Binary to Gray code converter

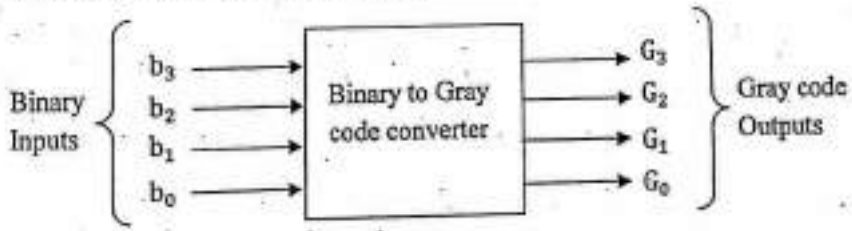
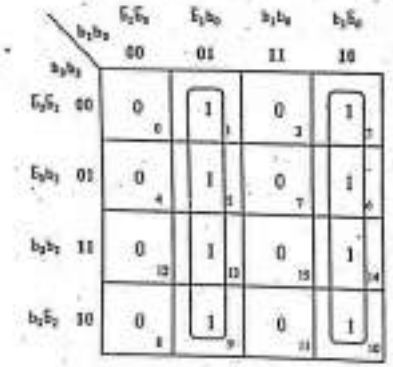


Figure 2.227 Block schematic of Binary to Gray code converter

Input Binary code				Output Gray code			
b_3	b_2	b_1	b_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

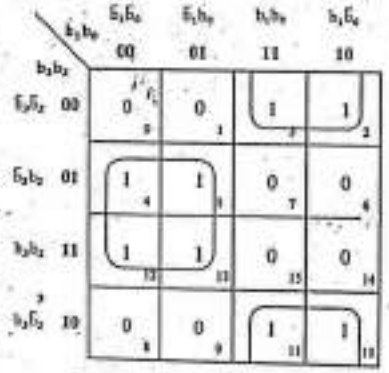
Table 2.54 Truth table for Binary to Gray code converter

K-map simplification for G_0



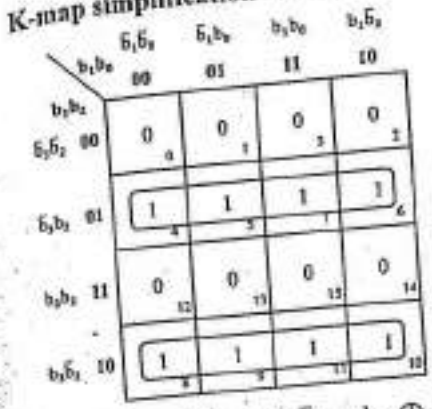
$$G_0 = \bar{b}_1 b_0 + b_1 \bar{b}_0 = b_1 \oplus b_0$$

K-map simplification for G_1



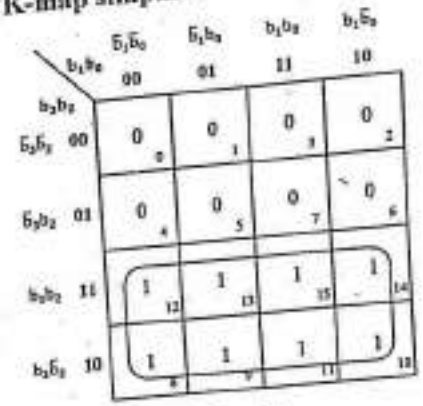
$$G_1 = b_2 \bar{b}_1 + \bar{b}_2 b_1 = b_2 \oplus b_1$$

K-map simplification for G_2



$$G_2 = \bar{b}_3 b_2 + b_3 \bar{b}_2 = b_3 \oplus b_2$$

K-map simplification for G_3



$$G_3 = b_3$$

Logic diagram

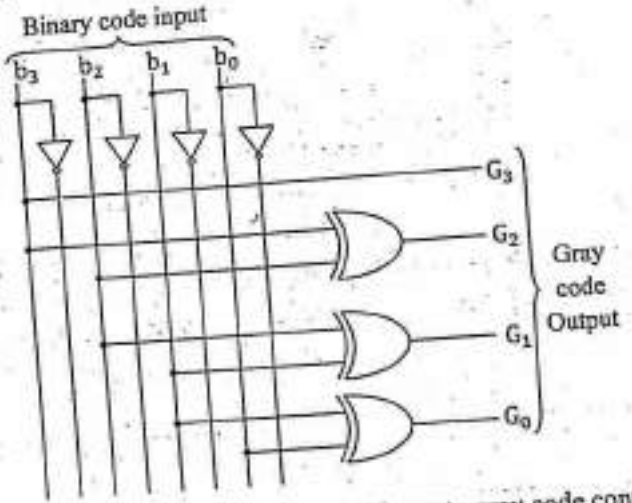


Figure 2.228 Logic diagram of binary to gray code converter

2.22.6 Gray to Binary code converter

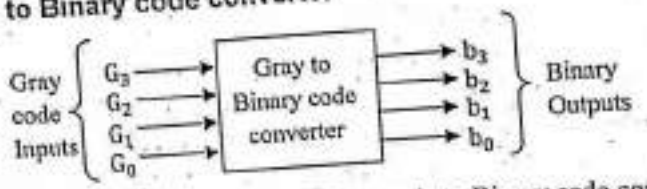
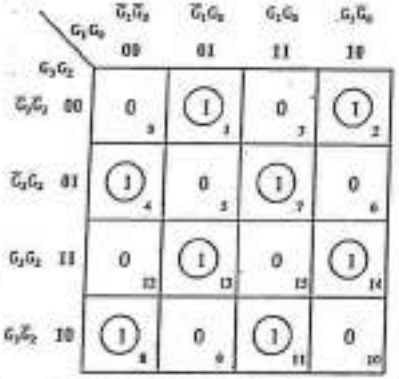


Figure 2.229 Block schematic of Gray code to Binary code converter

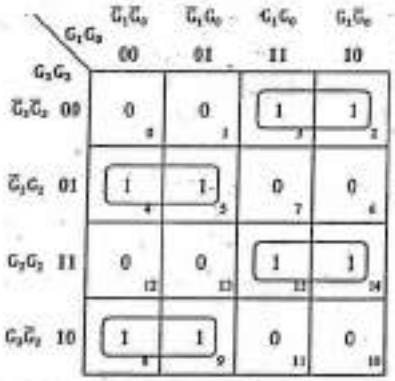
Input Gray code				Output Binary code			
G_3	G_2	G_1	G_0	b_3	b_2	b_1	b_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

Table 2.55 Truth table for Gray code to Binary converter

K-map simplification for b_0



K-map simplification for b_1



$$b_0 = \overline{G}_3\overline{G}_2\overline{G}_1G_0 + \overline{G}_3\overline{G}_2G_1\overline{G}_0 + \overline{G}_3G_2\overline{G}_1\overline{G}_0 + \overline{G}_3G_2G_1G_0 + G_3G_2\overline{G}_1G_0 + G_3G_2G_1\overline{G}_0 + G_3\overline{G}_2\overline{G}_1\overline{G}_0 + G_3\overline{G}_2G_1G_0$$

$$b_0 = \overline{G}_3\overline{G}_2(\overline{G}_1G_0 + G_1\overline{G}_0) + \overline{G}_3G_2(\overline{G}_1\overline{G}_0 + G_1G_0) + G_3G_2(\overline{G}_1G_0 + G_1\overline{G}_0) + G_3\overline{G}_2(\overline{G}_1\overline{G}_0 + G_1G_0)$$

$$b_0 = \overline{G}_3\overline{G}_2(G_1 \oplus G_0) + \overline{G}_3G_2(G_1 \odot G_0) + G_3G_2(G_1 \oplus G_0) + G_3\overline{G}_2(G_1 \odot G_0)$$

$$b_0 = \overline{G}_3\overline{G}_2(G_1 \oplus G_0) + \overline{G}_3G_2\overline{(G_1 \oplus G_0)} + G_3G_2(G_1 \oplus G_0) + G_3\overline{G}_2\overline{(G_1 \oplus G_0)}$$

$$b_0 = (G_1 \oplus G_0)(\overline{G}_3\overline{G}_2 + G_3G_2) + \overline{(G_1 \oplus G_0)}(\overline{G}_3G_2 + G_3\overline{G}_2)$$

$$b_0 = (G_1 \oplus G_0)(G_3 \odot G_2) + \overline{(G_1 \oplus G_0)}(G_3 \oplus G_2)$$

$$b_0 = (G_1 \oplus G_0)\overline{(G_3 \oplus G_2)} + \overline{(G_1 \oplus G_0)}(G_3 \oplus G_2)$$

Let $X = G_1 \oplus G_0$ and $Y = G_3 \oplus G_2$

Therefore $b_0 = X\overline{Y} + \overline{X}Y = X \oplus Y$

Substituting X and Y we get

$$b_0 = (G_1 \oplus G_0) \oplus (G_3 \oplus G_2)$$

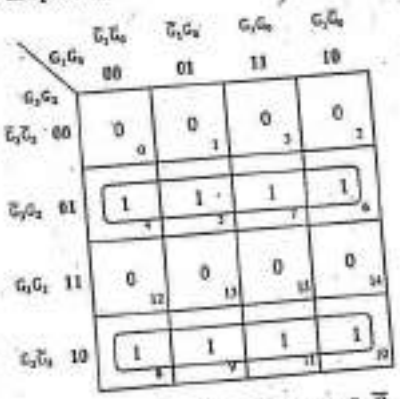
Also, $b_1 = \overline{G}_3\overline{G}_2G_1 + \overline{G}_3G_2\overline{G}_1 + G_3G_2G_1 + G_3\overline{G}_2\overline{G}_1$

$$b_1 = \overline{G}_3(\overline{G}_2G_1 + G_2\overline{G}_1) + G_3(G_2G_1 + \overline{G}_2\overline{G}_1)$$

$$b_1 = \overline{G}_3(G_2 \oplus G_1) + G_3\overline{(G_2 \oplus G_1)}$$

$$b_1 = G_3 \oplus (G_2 \oplus G_1)$$

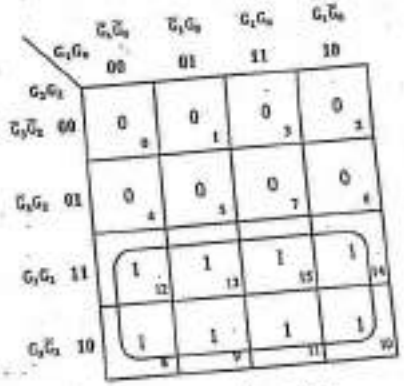
K-map simplification for b_2



$$b_2 = \overline{G}_3G_2 + G_3\overline{G}_2$$

$$b_2 = G_3 \oplus G_2$$

K-map simplification for b_3



$$b_3 = G_3$$

Logic diagram

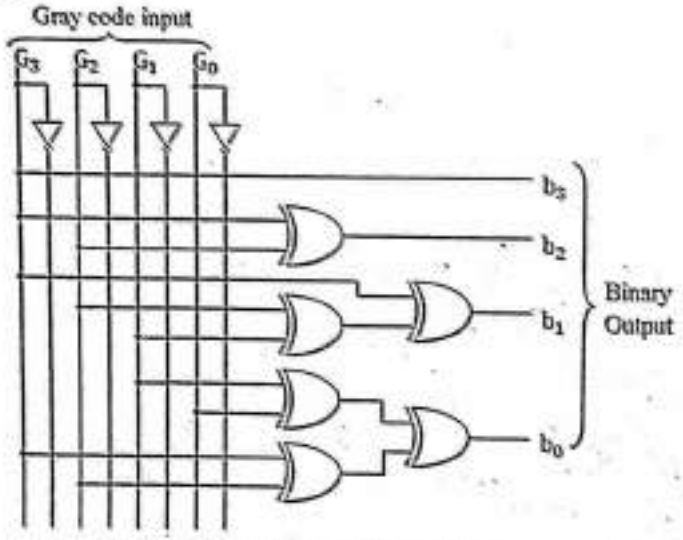


Figure 2.230 Logic diagram of Gray to Binary converter

2.22.7 BCD to Gray code converter

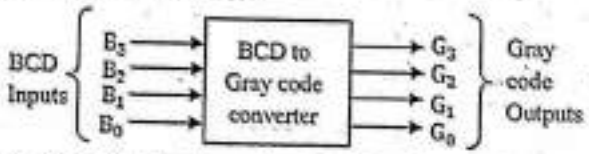
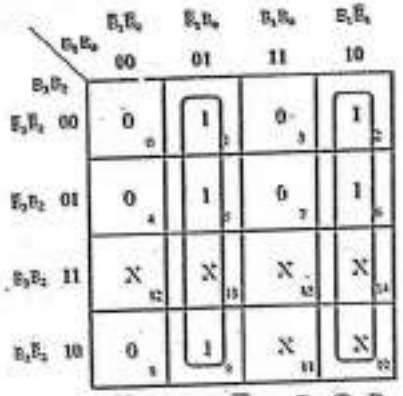


Figure 2.231 Block schematic of BCD to Gray code converter

Input BCD code				Output Gray code			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1

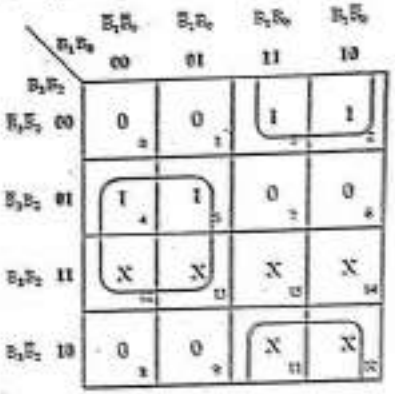
Table 2.56 Truth table for BCD to Gray code converter

K-map simplification for G_0



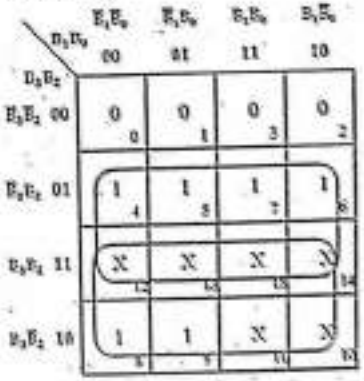
$G_0 = \overline{B}_1B_0 + B_1\overline{B}_0 = B_1 \oplus B_0$

K-map simplification for G_1



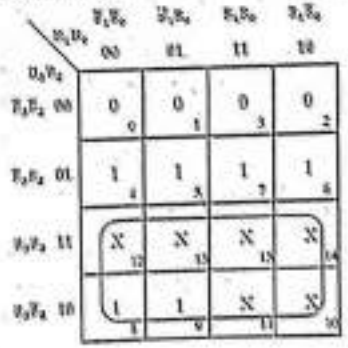
$G_1 = B_2\overline{B}_1 + \overline{B}_2B_1 = B_2 \oplus B_1$

K-map simplification for G_2



$G_2 = B_2 + B_3$

K-map simplification for G_3



$G_3 = B_3$

Logic diagram

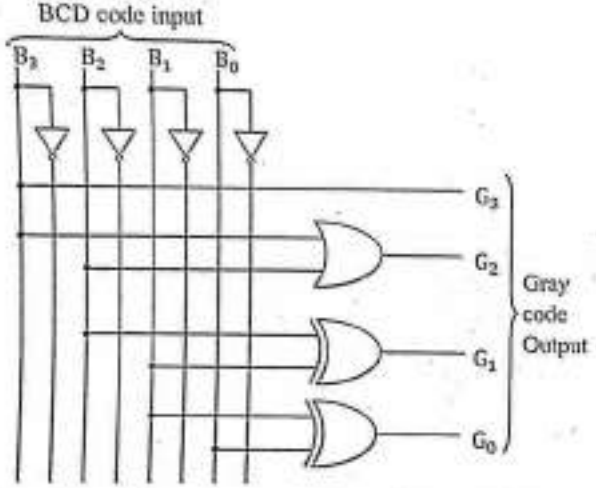


Figure 2.232 Logic diagram of BCD to Gray code Converter

2.22.8 Gray to BCD code converter

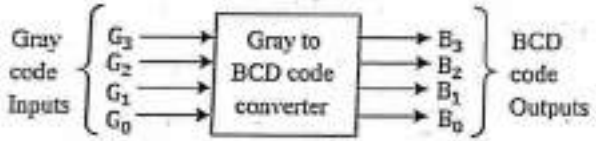
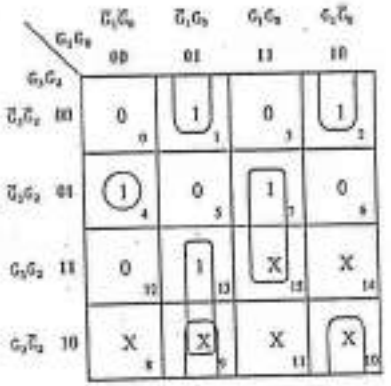


Figure 2.233 Block schematic of Gray code to BCD converter

Input Gray code				Output BCD code			
G_3	G_2	G_1	G_0	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1

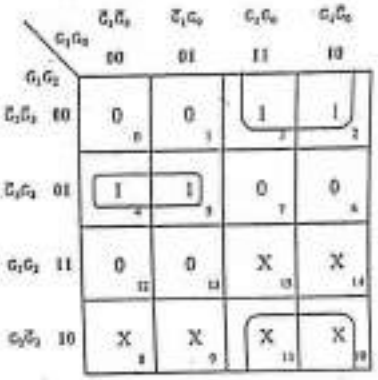
Table 2.57 Truth table for Gray to BCD code converter

K-map simplification for B_0



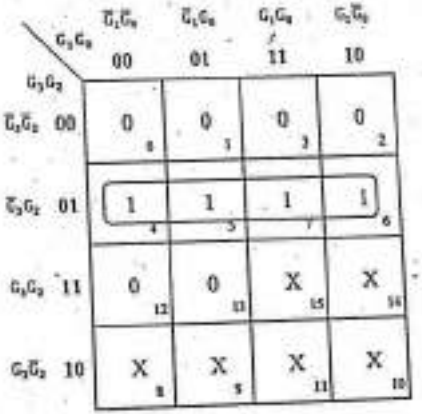
$$B_0 = G_3\overline{G}_1G_0 + \overline{G}_2\overline{G}_1G_0 + G_2G_1G_0 + \overline{G}_3G_2\overline{G}_1\overline{G}_0 + \overline{G}_2G_1\overline{G}_0$$

K-map simplification for B_1



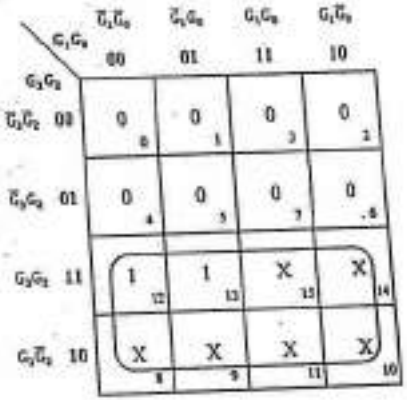
$$B_1 = \overline{G}_3G_2\overline{G}_1 + \overline{G}_2G_1$$

K-map simplification for B_2



$$B_2 = \overline{G}_3G_2$$

K-map simplification for B_3



$$B_3 = G_3$$

Logic diagram

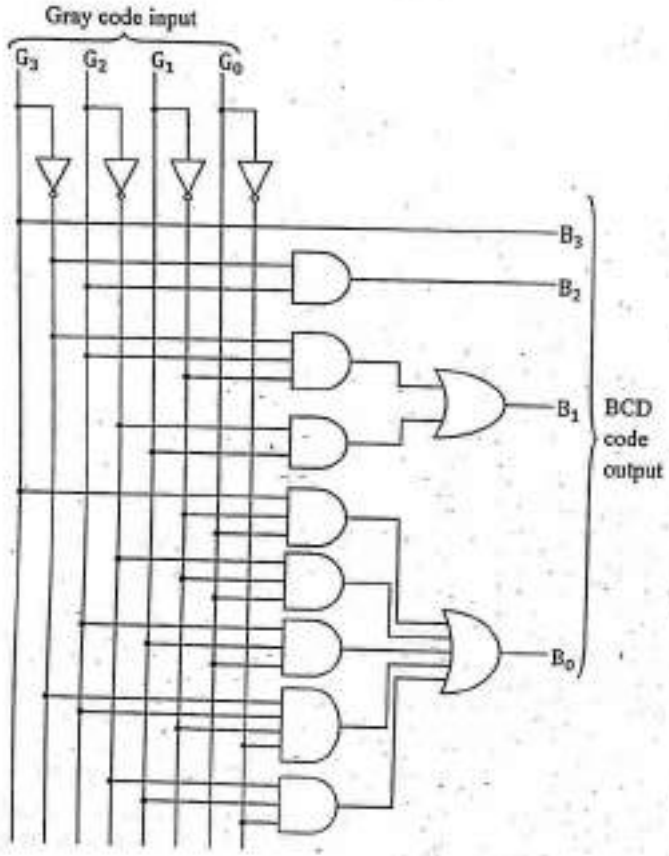


Figure 2.234 Logic diagram of Gray to BCD code converter

2.23 MAGNITUDE COMPARATOR

A comparator is a combinational circuit used to compare the relative magnitude of two binary numbers A and B. It receives two n-bit numbers A and B as inputs and produces the outputs A>B, A<B and A=B. Depending on the relative magnitude of the two number one of the outputs will be high and the other two outputs will be low.

2.23.1 2 Bit Magnitude Comparator

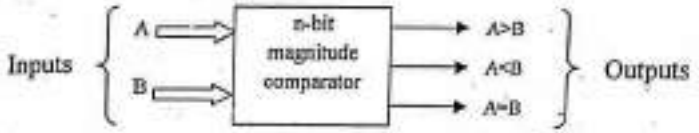


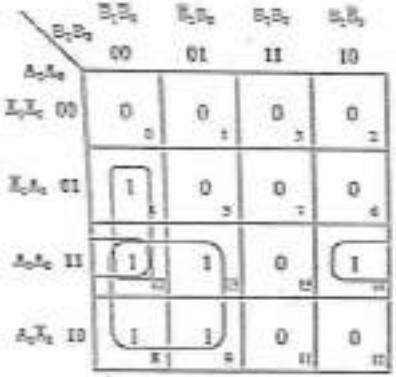
Figure 2.235 Block diagram of a magnitude comparator

A 2-bit magnitude comparator consists of two inputs A and B, each input is a combination of two bits. Hence A is a combination of A_1 and A_0, B is a combination of B_1 and B_0. The truth table of a 2-bit magnitude comparator is shown in table 2.58.

Inputs				Outputs		
A_1	A_0	B_1	B_0	A>B	A<B	A=B
0	0	0	0	0	0	1
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	1
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	0	1
1	0	1	1	0	1	0
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	0	1

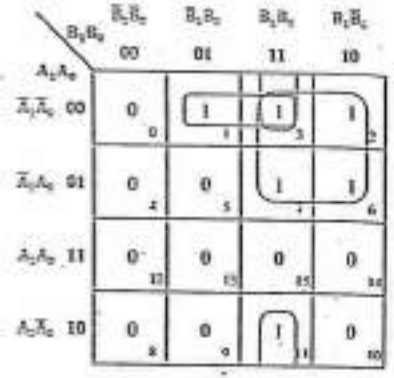
Table 2.58 Truth table of a 2-bit magnitude comparator

K-map for A>B



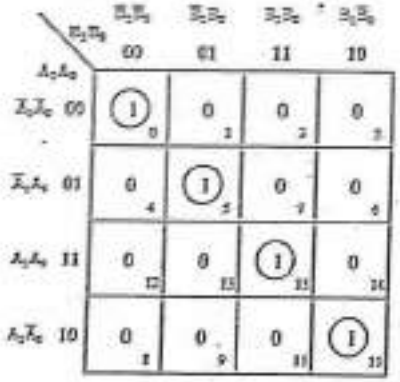
$(A > B) = A_1\overline{B}_1 + A_0\overline{B}_1\overline{B}_0 + A_1A_0\overline{B}_0$

K-map for A<B



$(A < B) = \overline{A}_1B_1 + \overline{A}_1\overline{A}_0B_0 + \overline{A}_0B_1B_0$

K-map for A=B



$(A = B) = \overline{A}_1\overline{A}_0\overline{B}_1\overline{B}_0 + \overline{A}_1A_0\overline{B}_1B_0 + A_1A_0B_1B_0 + A_1\overline{A}_0B_1\overline{B}_0$

$(A = B) = \overline{A}_1\overline{B}_1(\overline{A}_0\overline{B}_0 + A_0B_0) + A_1B_1(A_0B_0 + \overline{A}_0\overline{B}_0)$

$(A = B) = \overline{A}_1\overline{B}_1(A_0 \odot B_0) + A_1B_1(A_0 \odot B_0)$

$(A = B) = (A_0 \odot B_0)(\overline{A}_1\overline{B}_1 + A_1B_1)$

$(A = B) = (A_0 \odot B_0)(A_1 \odot B_1)$

Logic diagram

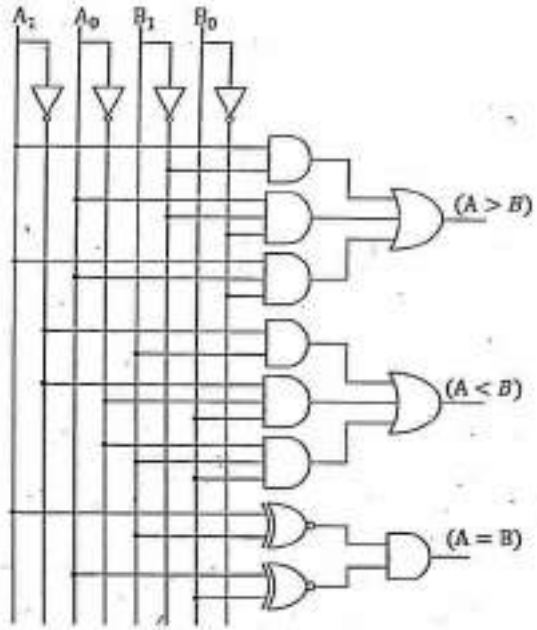


Figure 2.236 Logic diagram of a 2-bit magnitude comparator

2.23.2 4-Bit Magnitude comparator

A 4-bit magnitude comparator consists of two 4-bit numbers A and B. Hence A is a combination of 4 bits A_3, A_2 A_1, A_0 and B is a combination of 4 bits B_3, B_2 B_1, B_0. The two numbers A and B are equal, only if all the pairs of the significant digits are equal i.e., if $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$ and $A_0 = B_0$. The equality relation of each pair of bits can be expressed by the function as

$$X_n = A_nB_n + \overline{A}_n\overline{B}_n = A_n \odot B_n = \overline{A_n \oplus B_n} = \overline{A_nB_n + A_n\overline{B}_n}$$

Where n=0,1,2,3. When A_n and B_n are equal then $X_n = 1$ else $X_n = 0$. The magnitude comparator have 3 outputs (i) (A=B) (ii) (A>B) (iii) (A<B)

Case (i): (A=B)

The output (A=B) will be 1, if A and B are equal. For (A=B)=1 all X_n variables must be equal to 1. Therefore (A=B)= $X_3X_2X_1X_0$

Case (ii): (A>B)

Determine the magnitude of the bits in their relative positions of the two numbers starting from MSB. If two bits are equal, compare the bits in the next lower significant position. This procedure is continued until a pair of unequal bit is reached. Then if the corresponding bit is A=1 and B=0, we conclude that A>B. The Boolean function for (A>B) is

$$(A > B) = A_3\overline{B}_3 + X_3A_2\overline{B}_2 + X_3X_2A_1\overline{B}_1 + X_3X_2X_1A_0\overline{B}_0$$

Case (iii): (A<B)

Determine the magnitude of the bits in their relative positions of the two numbers starting from MSB. If two bits are equal, compare the bits in the next lower significant position. This procedure is continued until a pair of unequal bit is reached. Then if the corresponding bit is A=0 and B=1, we conclude that A<B. The Boolean function for (A<B) is

$$(A < B) = \overline{A}_3B_3 + X_3\overline{A}_2B_2 + X_3X_2\overline{A}_1B_1 + X_3X_2X_1\overline{A}_0B_0.$$

The logic diagram of 4-bit magnitude comparator is shown in figure 2.237.

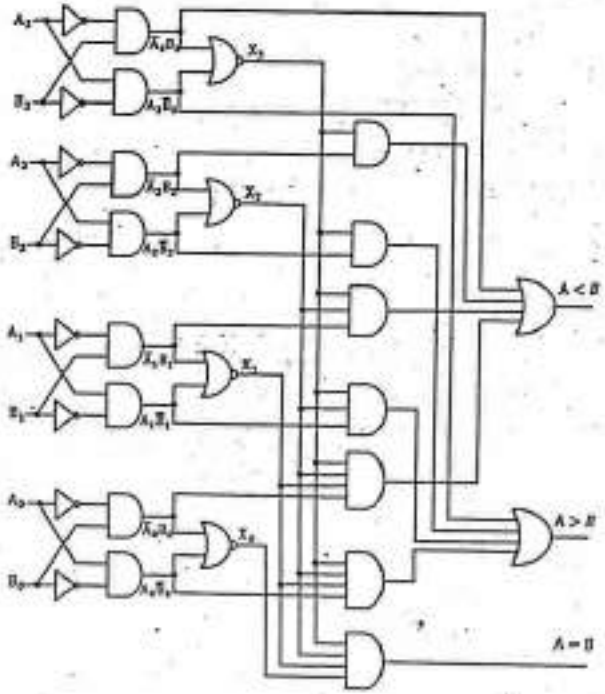


Figure 2.237 Logic Diagram of 4-bit magnitude comparator

2.24 QUINE-MCCLUSKEY METHOD OF MINIMIZATION (TABULATION METHOD)

As the number of variables increases, the K-map method become more complex. The tabulation method overcomes this difficulty. The tabulation method was first formulated by Quine and later improved by McCluskey. So tabular method is also known as Quine-McCluskey method.

The steps to be followed in the Quine-McCluskey method are,

1. List all the minterms in the binary form.
2. Arrange the minterms according to the number of 1's.
3. Compare each binary number with every term in the adjacent next higher category and if they differ only by one position, put a check mark and copy the term in the next column with '–' in the position that they differed.
4. Apply the same process described in Step 3 for the resultant column and continue these cycles until a single pass through cycle yields no further elimination of literate.
5. List all prime implicants.
6. Select the minimum number of prime implicants which must cover all the minterms.

Example 2.66: Simplify the following Boolean function by using the Quine-McCluskey method.

$$F(A,B,C,D) = \Sigma_m (0,2,3,6,7,8,10,12,13)$$

Solution :

Step 1: List all the minterms in the binary form.

Minterm	Binary representation ABCD
m_0	0 0 0 0
m_2	0 0 1 0
m_3	0 0 1 1
m_6	0 1 1 0
m_7	0 1 1 1
m_8	1 0 0 0
m_{10}	1 0 1 0
m_{12}	1 1 0 0
m_{13}	1 1 0 1

Table 2.59

Step 2: Arrange the minterms according to the number of 1's.

Number of 1's	Minterms	Binary representation ABCD
0	m_0✓	0 0 0 0
1	m_2✓	0 0 1 0
	m_8✓	1 0 0 0
2	m_3✓	0 0 1 1
	m_6✓	0 1 1 0
	m_{10}✓	1 0 1 0
	m_{12}✓	1 1 0 0
3	m_7✓	0 1 1 1
	m_{13}✓	1 1 0 1

Table 2.60

Step 3: Compare each binary number with every term in the adjacent next higher category and if they differ only by one position, put a check mark (✓) against the minterms and copy the term in the next column with '–' in the position that they differed.

Minterms	Binary representation ABCD
(0,2) ✓	0 0 – 0
(0,8) ✓	– 0 0 0
(2,3) ✓	0 0 1 –
(2,6) ✓	0 – 1 0
(2,10) ✓	– 0 1 0
(8,10) ✓	1 0 – 0
(8,12)	1 – 0 0
(3,7) ✓	0 – 1 1
(6,7) ✓	0 1 1 –
(12,13)	1 1 0 –

Table 2.61

Step 4: Apply the same process described in Step 3 for the resultant column and continue these cycles until a single pass through cycle yields no further elimination.

Minterms	Binary representation ABCD
(0,2,8,10)	– 0 – 0
(2,3,6,7)	0 – 1 –

Table 2.62

Step 5: List the prime implicants

	Minterms	Binary representation ABCD
a	(0,2,8,10)	– 0 – 0
b	(2,3,6,7)	0 – 1 –
c	(8,12)	1 – 0 0
d	(12,13)	1 1 0 –

Table 2.63

Step 6: Select the minimum number of prime implicants which must cover all the minterms.

	Prime implicants
*a	0, 2, 8, 10
*b	2, 3, 6, 7
~~c~~	~~8, 12~~
*d	12, 13

Table 2.64

First arrange the prime implicants based on number of minterms in each group. Here we are having 4 sets of prime implicants. Compare each set of prime implicants with other set, starting form 'd' to 'a'.

Compare 'd' with c,b and a. If both 12 and 13 is available in any one set c,b or a, then strike set 'd'. Else mark it by (*). Here '12' is available in 'c', but '13' is not available. So mark 'd' by (*). Similarly compare 'c' with a, b and d. If both 8 and 12 is available in any one set a, b or d, then strike 'c'. Here '8' and '12' are available in set 'a' and 'd', so strike 'c'. Similarly compare 'b' with 'a' and 'd' (no need to compare with c). Here all the minterms 2,3,6,7 are not available in set 'a' and 'd'. So

mark it by (*). Compare 'a' with set 'b' and 'd'. Here all the minterms (0,2,8,10) are not available in set 'b' or 'd'. So mark it by (*). So set a, b and d are minimum prime implicants. So the simplified expression can be written from table 2.163.

$$F(A,B,C,D) = (-0-0) + (0-1-) + (110-)$$

$$F(A,B,C,D) = \overline{B}\overline{D} + \overline{A}C + AB\overline{C}$$

Example 2.67: Minimize the expression using Quine-McCluskey method.

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

Solution:

$\overline{A}B\overline{C}\overline{D} = m_4$; $\overline{A}B\overline{C}D = m_5$; $AB\overline{C}\overline{D} = m_{12}$;

$AB\overline{C}D = m_{13}$; $A\overline{B}\overline{C}D = m_9$; $\overline{A}\overline{B}C\overline{D} = m_2$

The given expression can also be written as $Y(A,B,C,D) = \sum_m(2,4,5,9,12,13)$

Step 1: List all the minterms in the binary form.

Minterm	Binary representation
	ABCD
m_2	0010
m_4	0100
m_5	0101
m_9	1001
m_{12}	1100
m_{13}	1101

Table 2.65

Step 2: Arrange the minterms according to the number of 1's.

Number of 1's	Minterms	Binary representation
		ABCD
1	m_2	0010
	m_4✓	0100
2	m_5✓	0101
	m_9✓	1001
	m_{12}✓	1100
3	m_{13}✓	1101

Table 2.66

Step 3: Compare each binary number with every term in the adjacent next higher category and if they differ only by one position, put a check mark (✓) against the minterms and copy the term in the next column with '–' in the position that they differed.

Minterms	Binary representation
	ABCD
(4,5) ✓	010–
(4,12) ✓	–100
(5,13) ✓	–101
(9,13)	1–01
(12,13) ✓	110–

Table 2.67

Step 4: Apply the same process described in Step 3 for the resultant column and continue these cycles until a single pass through cycle yields no further elimination.

Minterms	Binary representation
	ABCD
(4,5,12,13)	–10–

Table 2.68

Step 5: List the prime implicants

	Minterms	Binary representation
		ABCD
a	(4,5,12,13)	–10–
b	(9,13)	1–01
c	2	0010

Table 2.69

Step 6: Select the minimum number of prime implicants which must cover all the minterms.

	Prime implicants
*a	4, 5, 12, 13
*b	9,13
* c	2

Table 2.70

First arrange the prime implicants based on number of minterms in each group. Here there are 3 sets of prime implicants a,b and c. Compare each set of prime implicants with other set, starting form 'c' to 'a'.Compare 'c' with 'a' and 'b'. Here minterm 2 is not available in 'a' or 'b'. So mark it by (*).Compare 'b' with 'a' and 'c', here minterm '13' is available in 'a', but '9' is not available in 'a' or 'c'. So mark 'b' with (*). Compare 'a' with 'b' and 'c'. Here the minterm '13' is available in 'b', but 4,5,12 is not available in 'b' or 'c'. So mark 'a' with (*). So sets a,b and c are minimum prime implicants. So the simplified expression can be written as.

$$Y = (-10-) + (1-01) + (0010)$$

$$Y = B\bar{C} + A\bar{C}D + \bar{A}BC\bar{D}$$

Example 2.68: Simplify the following using tabulation method

$$F(A, B, C, D) = \sum_m(1,2,3,5,9,12,14,15) + \sum_d(4,8,11)$$

Solution:

Step 1: List all the minterms in the binary form.

Minterm	Binary representation
	ABCD
m_1	0001
m_2	0010
m_3	0011
m_4	0100
m_5	0101
m_8	1000
m_9	1001
m_{11}	1011
m_{12}	1100
m_{14}	1110
m_{15}	1111

Table 2.71

Step 2: Arrange the minterms according to the number of 1's.

Number of 1's	Minterms	Binary representation
		ABCD
1	m_1✓	0001
	m_2✓	0010
	m_4✓	0100
	m_8✓	1000
2	m_3✓	0011
	m_5✓	0101
	m_9✓	1001
	m_{12}✓	1100
3	m_{11}✓	1011
	m_{14}✓	1110
4	m_{15}✓	1111

Table 2.72

Step 3: Compare each binary number with every term in the adjacent next higher category and if they differ only by one position, put a check mark (✓) against the minterms and copy the term in the next column with '–' in the position that they differed.

Minterms	Binary representation
	ABCD
(1,3) ✓	00–1
(1,5)	0–01
(1,9) ✓	–001
(2,3)	001–
(4,5)	010–
(4,12)	–100
(8,9)	100–
(8,12)	1–00
(3,11) ✓	–011
(9,11) ✓	10–1
(12,14)	11–0
(11,15)	1–11
(14,15)	111–

Table 2.73

Step 4: Apply the same process described in Step 3 for the resultant column and continue these cycles until a single pass through cycle yields no further elimination.

Minterms	Binary representation
	ABCD
(1,3,9,11)	−0−1

Table 2.74

Step 5: List the prime implicants

	Minterms	Binary representation
		ABCD
a	1,3,9,11	−0−1
b	1,5	0−01
c	2,3	001−
d	4,5	010−
e	4,12	−100
f	8,9	100−
g	8,12	1−00
h	12,14	11−0
i	11,15	1−11
j	14,15	111−

Table 2.75

Step 6: Select the minimum number of prime implicants which must cover all the minterms, except don't care

	Prime implicants
*a	1, 3, 9
*b	1, 5
*c	2, 3
*h	12, 14
*j	14, 15
d	5
e	12
f	9
g	12
i	15

Table 2.76

First arrange the prime implicants based on number of minterms in each group without including the don't cares. Here there are 10 sets of prime implicants without don't cares. Compare each set of prime implicats with other set from bottom to top. Compare 'i' with other sets, here '15' is found in 'j', so strike 'i'. Compare 'g' with other sets except 'i'. '12' is found in 'e' and 'h'. So strike 'g'. Repeat the same process. '9' in set 'f' is found in 'a'. So strike 'f'.'12' in set 'e' is found in 'h'. So strike 'e'. '5' in set 'd' is found in 'b', so strike 'd'. Compare 'j' with other active sets. '14' is found in 'h', but '15' is not available in a,b,c or h. So mark 'j' with (*). Compare 'h' with other active sets a,b,c and j. Here 14 is available in 'j', but '12' is not available in any other active sets. So mark 'h' with (*). Similarly compare c, b and a with other active sets. So sets a,b,c,h and j are minimum prime implicants.

$$F(A, B, C, D) = (-0-1) + (0-01) + (001-) + (11-0) + (111-)$$

$F(A, B, C, D) = \overline{B}D + \overline{A}CD + \overline{A}B\overline{C} + AB\overline{D} + ABC$ is the simplified Boolean expression.

Example 2.69: Simplify the logic function of equation

$$F(A, B, C, D) = \sum_m(1,3,7,11,15) + \sum_d(0,2,5)$$

Solution:

Step 1: List all the minterms in the binary form.

Minterm	Binary representation
	ABCD
m_0	0000
m_1	0001
m_2	0010
m_3	0011
m_5	0101
m_7	0111
m_{11}	1011
m_{15}	1111

Table 2.77

Step 2: Arrange the minterms according to the number of 1's.

Number of 1's	Minterms	Binary representation
		ABCD
0	m_0✓	0000
1	m_1✓	0001
	m_2✓	0010
2	m_3✓	0011
	m_5✓	0101
3	m_7✓	0111
	m_{11}✓	1011
4	m_{15}✓	1111

Table 2.78

Step 3: Compare each binary number with every term in the adjacent next higher category and if they differ only by one position, put a check mark (✓) against the minterms and copy the term in the next column with '–' in the position that they differed.

Minterms	Binary representation
	ABCD
(0,1)✓	000–
(0,2)✓	0.0–0
(1,3)✓	00–1
(1,5)✓	0–01
(2,3)✓	001–
(3,7)✓	0–11
(3,11)✓	–011
(5,7)✓	01–1
(7,15)✓	–111
(11,15)✓	1–11

Table 2.79

Step 4: Apply the same process described in Step 3 for the resultant column and continue these cycles until a single pass through cycle yields no further elimination.

Minterms	Binary representation
	ABCD
(0,1,2,3)	00––
(1,5,3,7)	0––1
(3,7,11,15)	––11

Table 2.80

Step 5: List the prime implicants

	Minterms	Binary representation
		ABCD
a	0,1, 2, 3	00––
b	1, 5, 3, 7	0––1
c	3, 7 11, 15	––11

Table 2.81

Step 6: Select the minimum number of prime implicants which must cover all the minterms, except don't care

	Prime implicants
*c	3, 7, 11, 15
*b	1, 3, 7
✗	✗,✗

Table 2.82

First arrange the prime implicants based on number of minterms in each group without including the don't cares. Here there are 3 sets of prime implicants without don't cares. Compare each set of prime implicants with other set from bottom to top.

Compare 'a' with 'b' and 'c'. Here the minterm '1' is available in 'b' and the minterm '3' is available in 'c' and 'b'. So strike 'a'.

Compare 'b' with 'c'. The minterm 3 and 7 is available in 'c'. But the minterm '1' is not available in 'c'. So mark it with (*). In 'c', the minterms '15' and '11' is not available in other active set 'b'. So mark it with (*). So sets 'c' and 'b' are minimum prime implicants.

By using the table 2.81 and 2.82

$$F(A, B, C, D) = (--11) + (0--1)$$

$F(A, B, C, D) = CD + \bar{A}D$ is the simplified Boolean expression.