# Unit 2

## (A) Boolean Algebra :→

### Postulates and theorems of Boolean Algebra →

(i) $x+0=x$

(ii) $x+x'=1$

(iii) $x+x=x$

(iv) $x+1=1$

(v) $(x')'=x$

(vi) $x+y=y+x$

(vii) $x(y+z)=xy+xz$

(a) $x\cdot 1=x$

(b) $x\cdot x'=0$

(c) $x\cdot x=x$

(d) $x\cdot 0=0$

(e) $xy=yz$

(f) $x(yz)=(xy)z$

### Demorgon's theorem →

(viii) $(x+y)'=x'\cdot y'$

(ix) $x+xy=x$

$=x(1+y)$

$=x$

(x) $x(x+y)=x$

(g) $(x\cdot y)'=x'+y'$

(h) $x+yz$

$=(x+y)(x+z)$

# Simplify the Boolean functions.

(I) $f = x + x'y$

Ans → 1

(II) $f = x(x'+y)$

Ans → $xy$

(III) $f = x'y'z + x'yz$

Ans → $x'z$

(IV)

$f = xy + x'z + yz$

$= xy + x'z + yz(x+x')$

$= xy + x'z + xyz + x'yz$

$= xy(1+z) + x'z(1+y)$

$= xy + x'z$

(V) $f = (A'+B)(A+B+D)D'$

Ans → $BD'$

(VI) $(\overline{A+\overline{B}C})(A\overline{B} + ABC) = 0$

(VIII) $f = \overline{(\overline{A} + \overline{A+B})(\overline{B} + \overline{B+C})}$

Ans → $A+B$

(IX) $f = (A+AB)(B+BC)(C+AB)$

Ans → $AB$

(8) Minterm and Maxterm of three binary number (SOP) (POS) :→

| x | y | z | Minterm (SOP) | Maxterm (POS) |
|---|---|---|---|---|
| 0 | 0 | 0 | $x'y'z' \rightarrow m_0$ | $x+y+z \rightarrow M_0$ |
| 0 | 0 | 1 | $x'y'z \rightarrow m_1$ | $x+y+z' \rightarrow M_1$ |
| 0 | 1 | 0 | $x'yz' \rightarrow m_2$ | $x+y'+z \rightarrow M_2$ |
| 0 | 1 | 1 | $x'yz \rightarrow m_3$ | $x+y'+z' \rightarrow M_3$ |
| 1 | 0 | 0 | $xy'z' \rightarrow m_4$ | $x'+y+z \rightarrow M_4$ |
| 1 | 0 | 1 | $xy'z \rightarrow m_5$ | $x'+y+z' \rightarrow M_5$ |
| 1 | 1 | 0 | $xyz' \rightarrow m_6$ | $x'+y'+z \rightarrow M_6$ |
| 1 | 1 | 1 | $xyz \rightarrow m_7$ | $x'+y'+z' \rightarrow M_7$ |

F(A, B, C)

(1) Express the Boolean function $F = A + B'C$ in SOP form (minterm).

Sol:→

$$F = A + B'C$$

$$= A(B+B')(C+C') + (A+A')B'C$$

$$= ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C$$

$$= ABC + ABC' + AB'C + AB'C' + A'B'C$$

| 111 | 110 | 101 | 100 | 001 |
|---|---|---|---|---|
| $m_7$ | $m_6$ | $m_5$ | $m_4$ | $m_1$ |

$$f = m_1 + m_4 + m_5 + m_6 + m_7$$

$$f(A, B, C) = \Sigma(1, 4, 5, 6, 7)$$

(2) Convert the Boolean function in POS form (Maxterm).

~~Sol:→ f A, P~~ $f(x, y, z) = (x+y)(x+z)(y+z)$

$$f = (x'+y+zz')(x+z+yy')(y+z+xx')$$

$$= (x'+y+z)(x'+y+z')(x+y+z)(x+y'+z)$$

$$(x+y+z)(x'+y+z)$$

$f = (x'+y+z)(x'+y+z')(x+y+z)$

100 $(M_4)$ 101 $(M_5)$ 000 $M_0$

$(x+y'+z)$

010 $(M_2)$

$f(x,y,z) = \pi(M_0, M_2, M_4, M_5)$

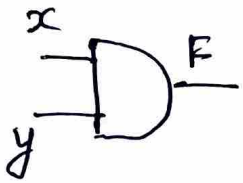
(3) Simplify $f(x,y,z) = (x+z)(x'+z')$ in POS form.

Ans → $\pi(0, 2, 5, 7)$

(C) **Digital logic Gates** → Since Boolean function are expressed in term of AND, OR and NOT ~~not~~ operations. It is easier to implement a boolean function with these types of gates.

| Name | Symbol | Function | Truth table |
|---|---|---|---|

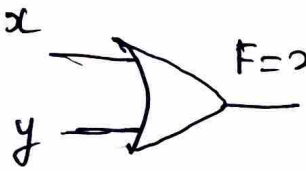
(1) AND




$F = xy$ (Multiplication)

| Input | | Output |
|---|---|---|
| x | y | F |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

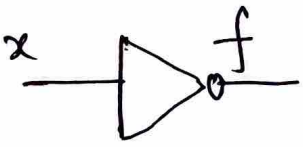
(2) OR




$F = x + y$ (Addition)

| Input | | output |
|---|---|---|
| x | y | F |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

(3) NOT

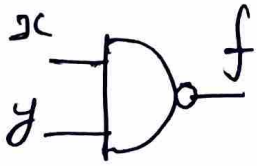



$F = x'$

| Input | output |
|---|---|
| x | f |
| 0 | 1 |
| 1 | 0 |

| Name | Symbol | function | Truth table |
|---|---|---|---|

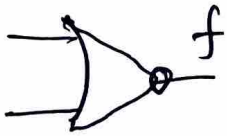
(4) NAND




$f = (xy)'$

| input | | output |
|---|---|---|
| x | y | f |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

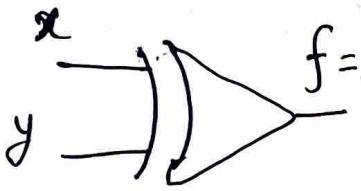
(5) NOR




$f = (x+y)'$

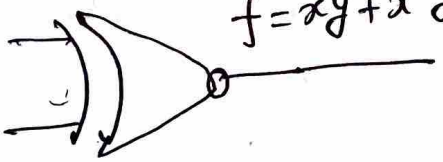
| input | | output |
|---|---|---|
| x | y | f |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

(6) X-OR




| x | y | f |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

(7) X-NOR

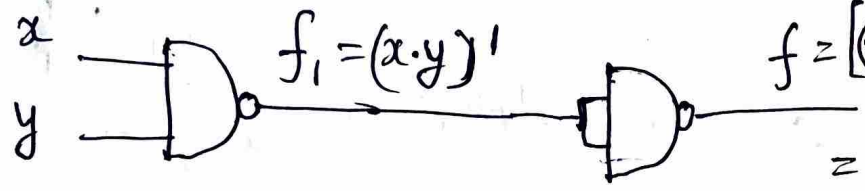



| input | | output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Que → Show that NAND and NOR Gate are called universal gate.

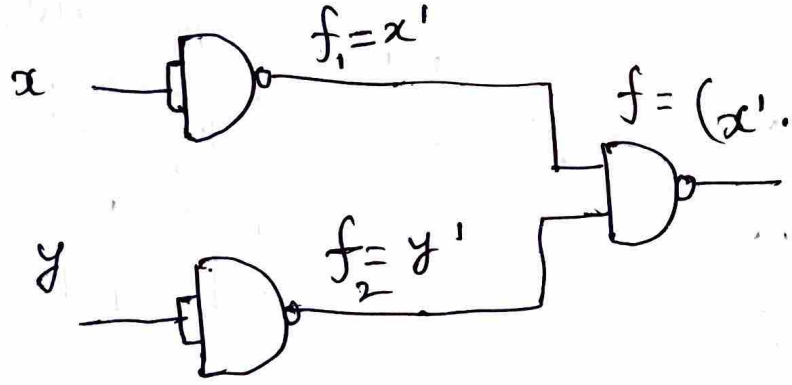
Ans → NAND and NOR gates are called universal gates because all the basics gates can be realized using NAND and NOR gates.

(i) (a) Realization of AND gate using NAND gate.




$= x \cdot y$ (AND)

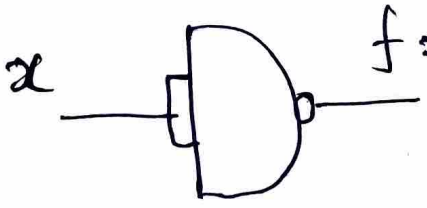
(ii) Realization of OR gate using NAND Gate.




$= x + y$

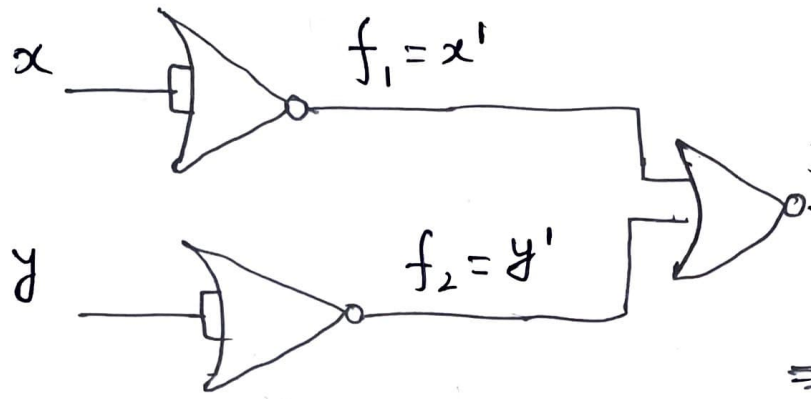
(OR Gate)

(iii) Realization of NOT Gate using NAND gate

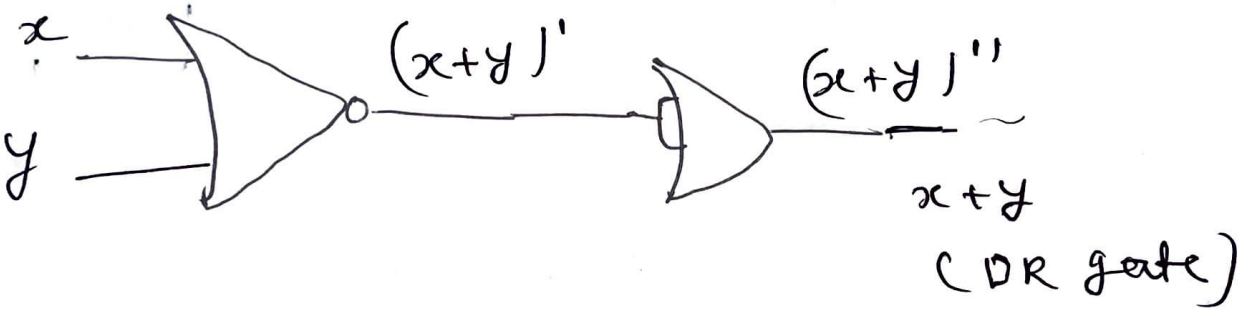



(NOT

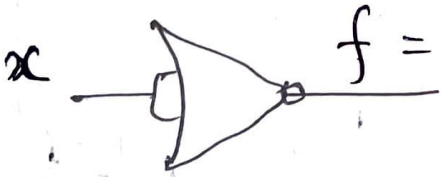
Que → (a) Realization of AND gate using NOR gate.



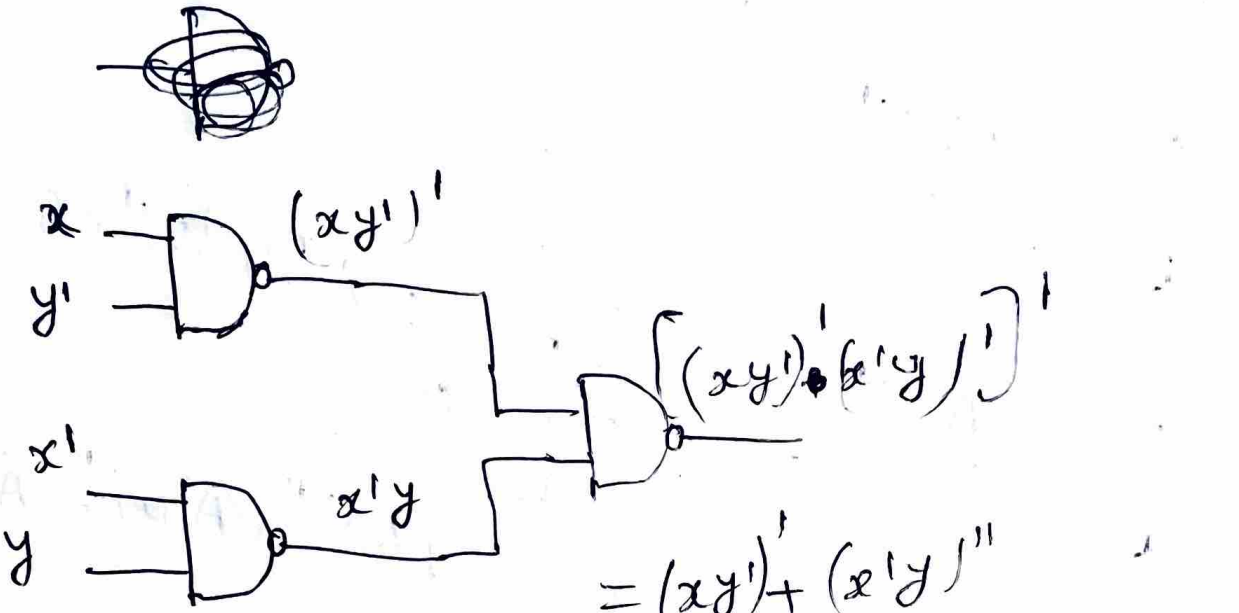

(b) Realization of OR gate using NOR gate




(C) Realization of NOT gate using NOR gate.



NAND to XOR → $f = xy' + x'y$

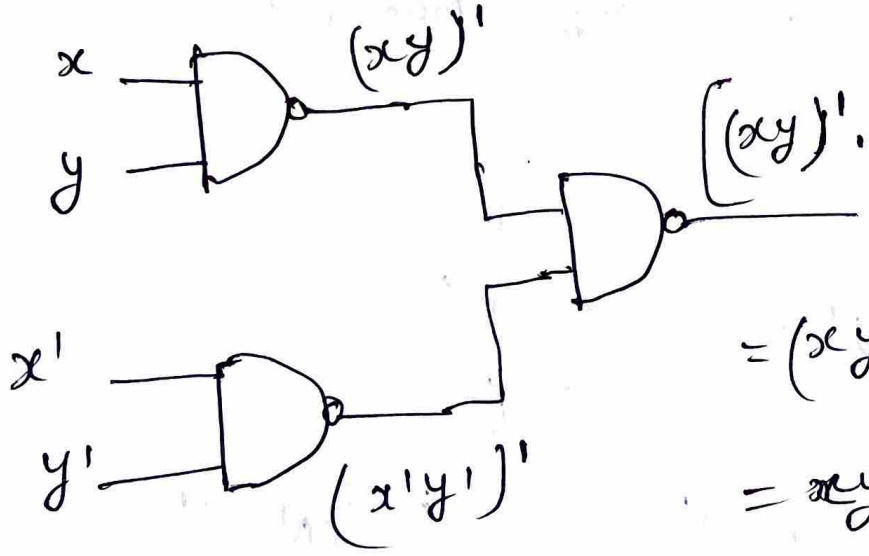



$= (xy')' + (x'y)''$

$= xy' + x'y$ (XOR)

(2) NAND to XNOR →

$f = xy + x'y'$



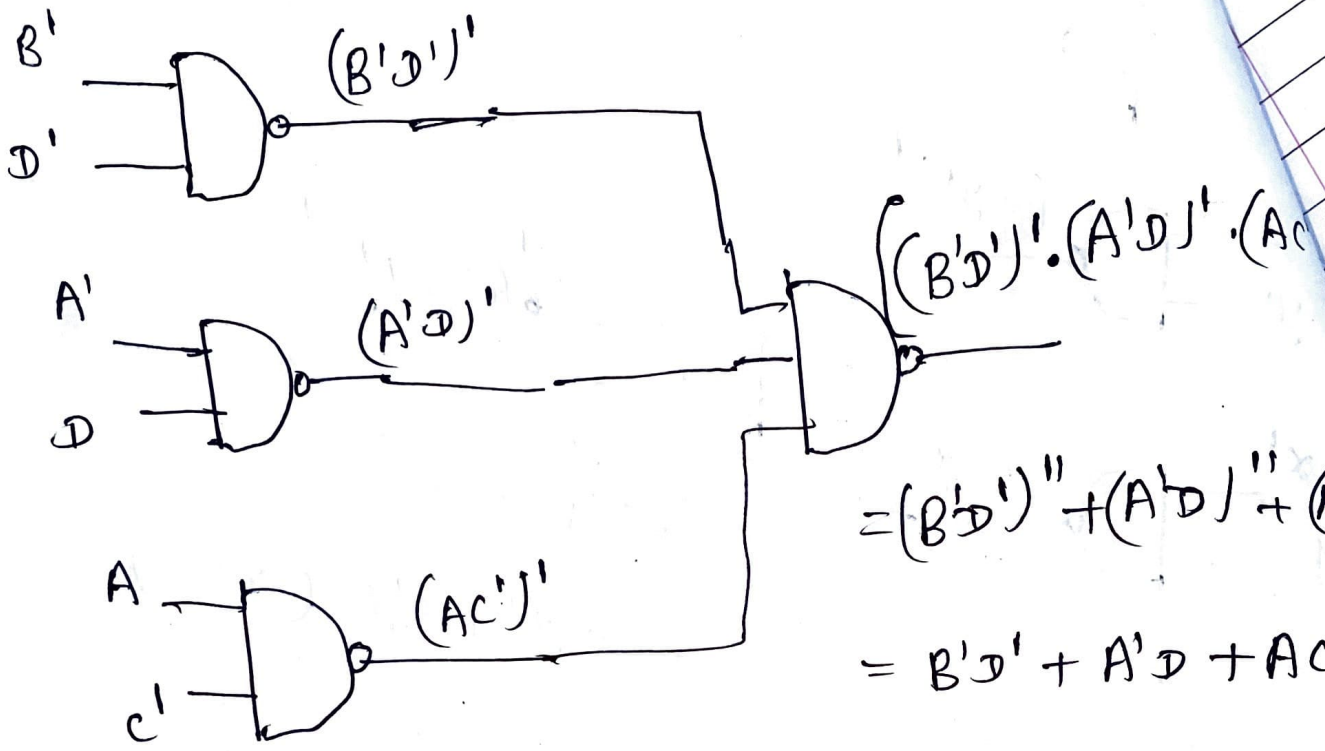

$= (xy)'' + (x'y')''$

$= xy + x'y'$

(XNOR)

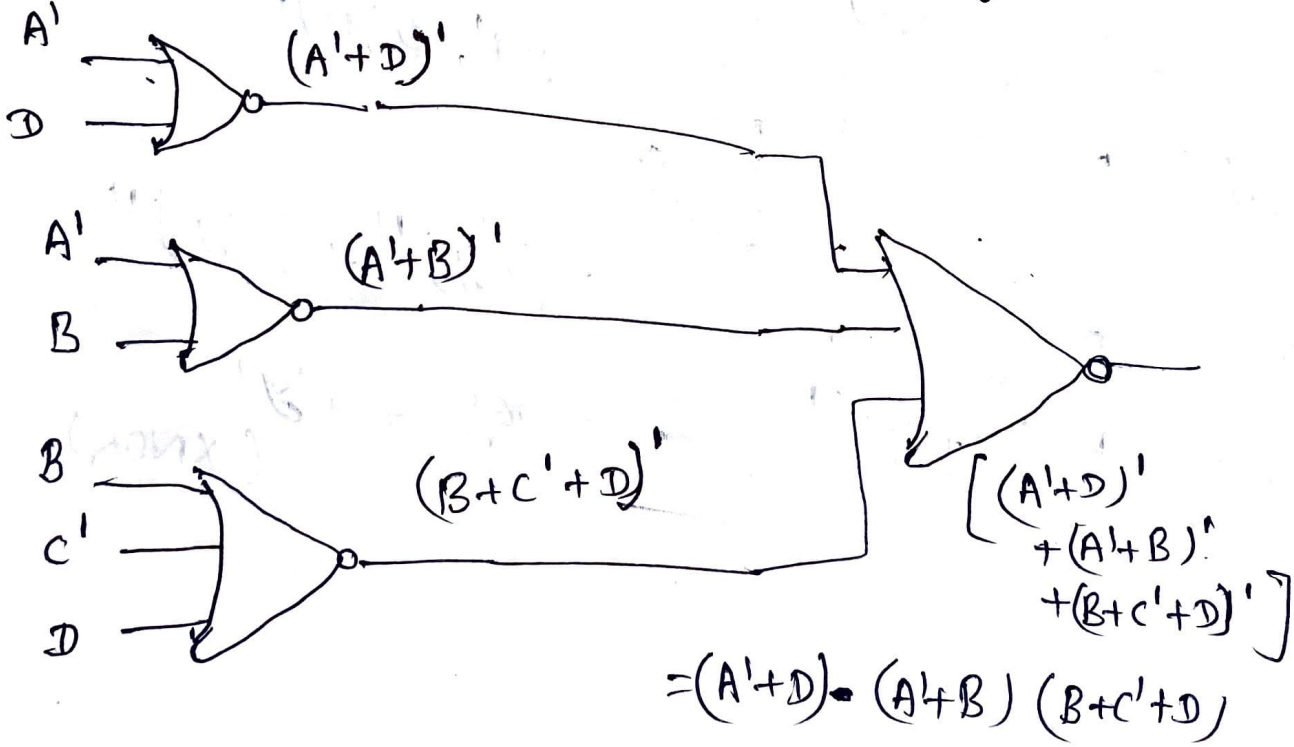
(3) $f = B'D' + A'D + AC'$

Implement it only NAND gate

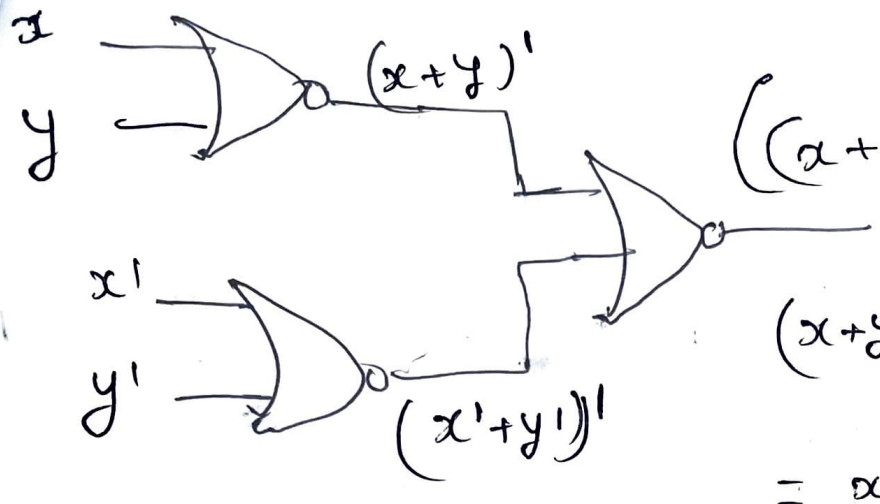



(4) $f = (A'+D)(A'+B)(B+C'+D)$

Implement it with only NOR gate



NOR to XOR → $f = x'y + xy'$




$[(x+y)' + (x'+y')]'$

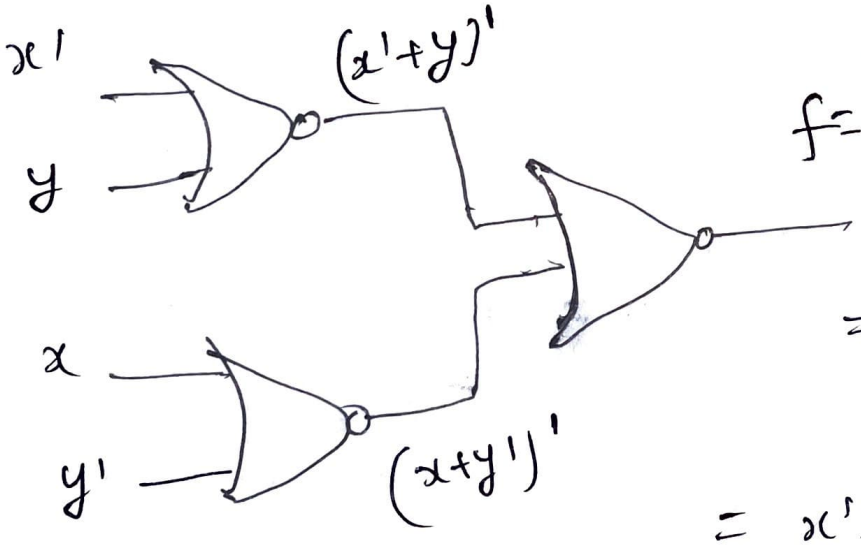
$(x+y)'' \cdot (x'+y')''$

$= xx' + x'y + xy' + yy'$

$= x'y + xy'$

(XOR)

(5) NOR to XNOR → $f = xy + x'y'$




$f = [(x'+y)' + (x+y')]'$

$= \cancel{x'x + }$

$(x'+y)'' \cdot (x+y')''$

$= x'x + xy + x'y' + yy'$

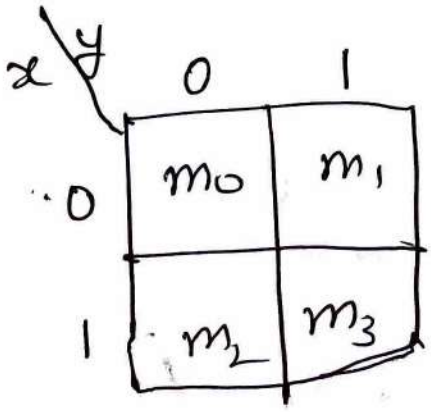
$= xy + x'y'$

(XNOR)

**K-Map** → The k-map method provides a simple and straight forward procedure for minimizing Boolean functions.

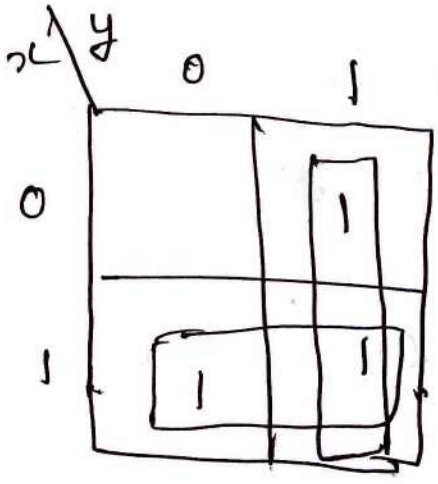
## Two Variable K-map →

| x | y | output | |
|---|---|---|---|
| 0 | 0 | $x'y'$ | $m_0$ |
| 0 | 1 | $x'y$ | $m_1$ |
| 1 | 0 | $xy'$ | $m_2$ |
| 1 | 1 | $xy$ | $m_3$ |



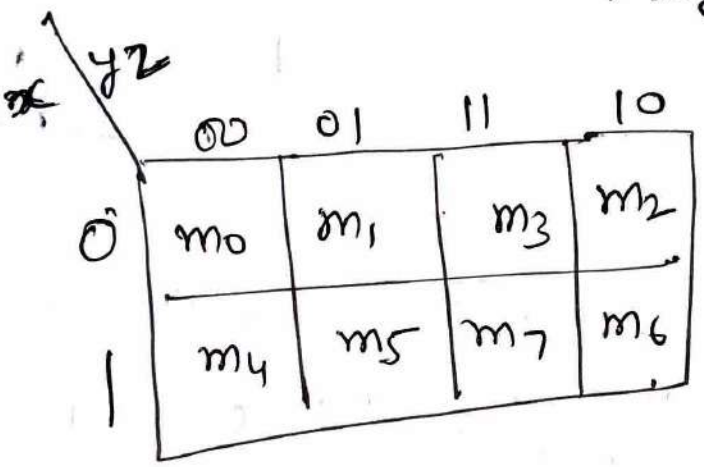

(1) $f(x,y) = xy' + xy + x'y$

$f = x + y$



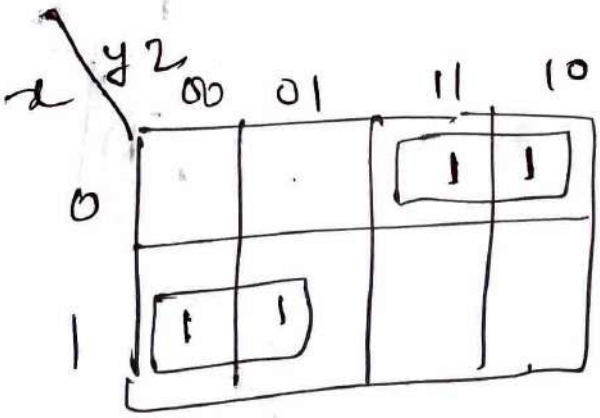
## Three Variable k-Map →

| x | y | z | f |
|---|---|---|---|
| 0 | 0 | 0 | $m_0 \rightarrow x'y'z'$ |
| 0 | 0 | 1 | $m_1 \rightarrow x'y'z$ |
| 0 | 1 | 0 | $m_2 \rightarrow x'yz'$ |
| 0 | 1 | 1 | $m_3 \rightarrow x'yz$ |
| 1 | 0 | 0 | $m_4 \rightarrow xy'z'$ |
| 1 | 0 | 1 | $m_5 \rightarrow xy'z$ |
| 1 | 1 | 0 | $m_6 \rightarrow xyz'$ |
| 1 | 1 | 1 | $m_7 \rightarrow xyz$ |




(2) Simplify the Boolean function using k-map. $f(x,y,z) = \Sigma(2,3,4,5)$

Sol:→



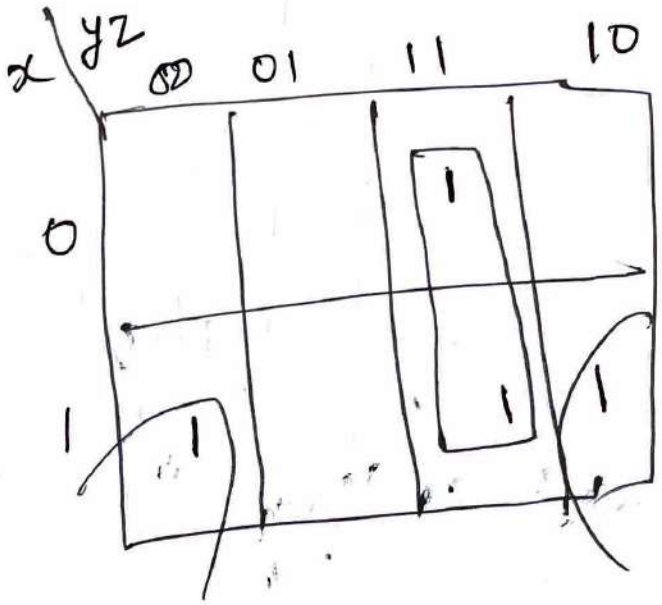

$$f = xy' + x'y$$

(3) Simplify the Boolean function using K-map.

$$f(x, y, z) = \Sigma(3, 4, 6, 7)$$

Sol:→

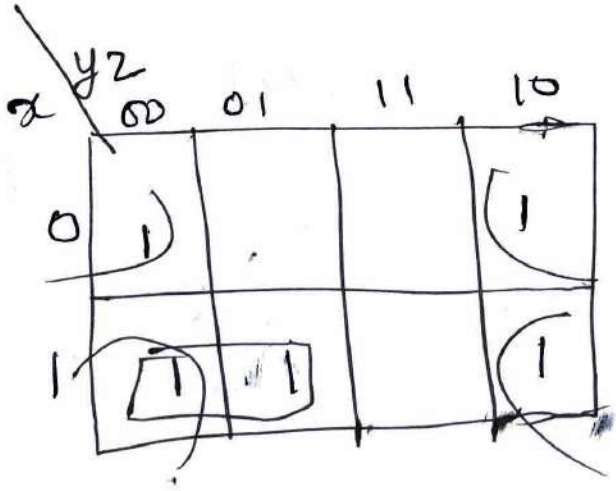
$$f = yz + xz'$$




(4) Simplify the Boolean function using K-map. $f(x, y, z) = \Sigma(0, 2, 4, 5, 6)$

Sol:→

$$f = z' + xy'$$



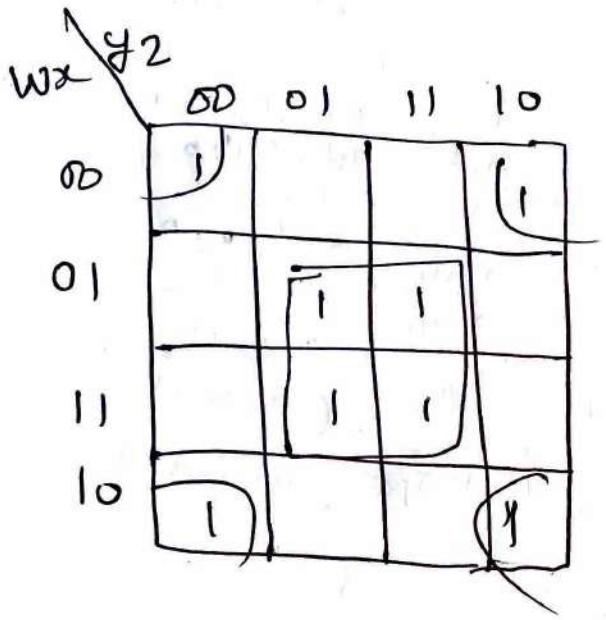
## Four Variable k-map →

| w | x | y | z | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | → $m_0$ | $(w'x'y'z')$ |
| 0 | 0 | 0 | 1 | → $m_1$ | $(w'x'y'z)$ |
| 0 | 0 | 1 | 0 | → $m_2$ | $(w'x'yz')$ |
| 0 | 0 | 1 | 1 | → $m_3$ | $(w'x'yz)$ |
| 0 | 1 | 0 | 0 | → $m_4$ | $(w'xy'z')$ |
| 0 | 1 | 0 | 1 | → $m_5$ | $(w'xy'z)$ |
| 0 | 1 | 1 | 0 | → $m_6$ | $(w'xyz')$ |
| 0 | 1 | 1 | 1 | → $m_7$ | $(w'xyz)$ |
| 1 | 0 | 0 | 0 | → $m_8$ | $wx'y'z'$ |
| 1 | 0 | 0 | 1 | → $m_9$ | $wx'y'z$ |
| 1 | 0 | 1 | 0 | → $m_{10}$ | $wx'yz'$ |
| 1 | 0 | 1 | 1 | → $m_{11}$ | $wx'yz$ |
| 1 | 1 | 0 | 0 | → $m_{12}$ | $wxy'z'$ |
| 1 | 1 | 0 | 1 | → $m_{13}$ | $wxy'z$ |
| 1 | 1 | 1 | 0 | → $m_{14}$ | $(wxyz')$ |
| 1 | 1 | 1 | 1 | → $m_{15}$ | $(wxyz)$ |

| wx \ yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | $m_0$ | $m_1$ | $m_3$ | $m_2$ |
| 01 | $m_4$ | $m_5$ | $m_7$ | $m_6$ |
| 11 | $m_{12}$ | $m_{13}$ | $m_{15}$ | $m_{14}$ |
| 10 | $m_8$ | $m_9$ | $m_{11}$ | $m_{10}$ |

(1) Simplify the Boolean function using K-map and Realize it through basic logic gates.

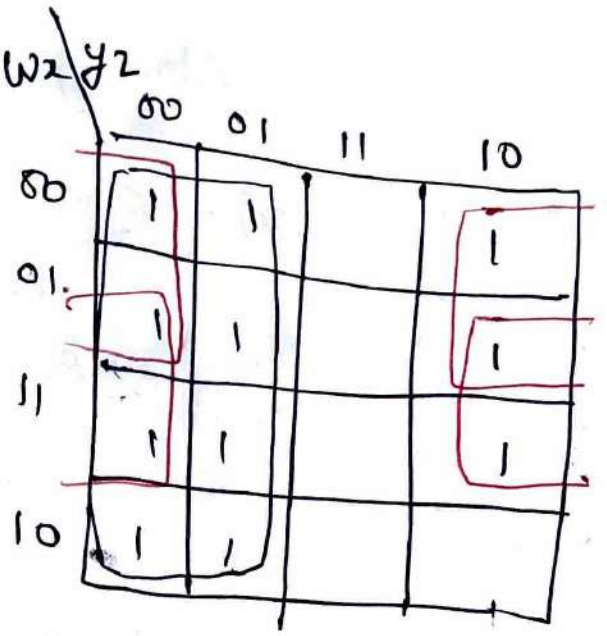
Sol:→ $f(w,x,y,z) = \Sigma(0,2,5,7,8,10,13,15)$



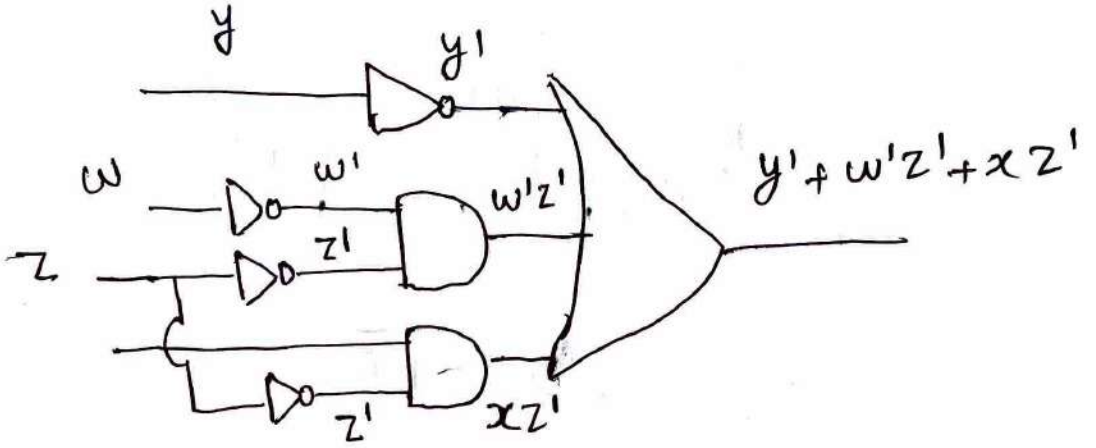

$f = xz + x'z'$

(2) Simplify the Boolean function using k-map $f(w,x,y,z) = \Sigma(0,1,2,4,5,6,8,9,12,13,14)$ and Realize it through basic logic gates.

Sol:→



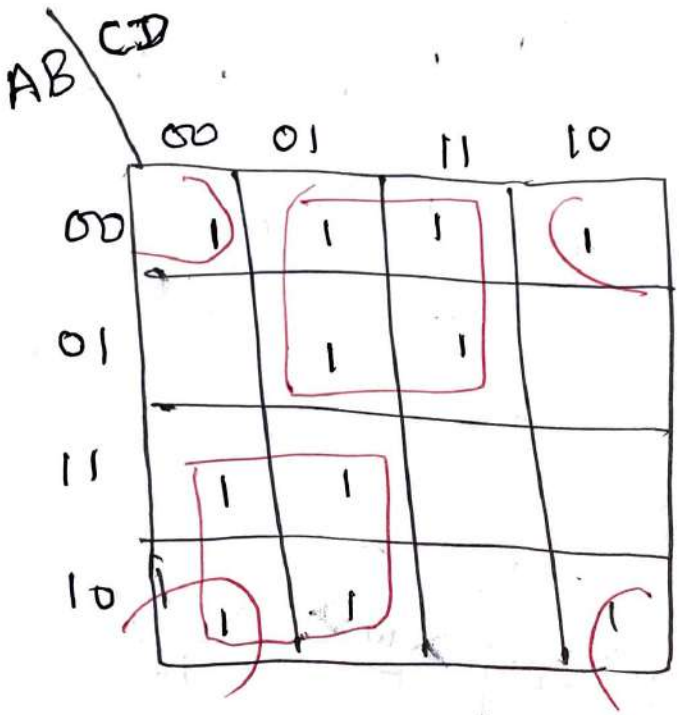

$$f = y' + w'z' + xz'$$



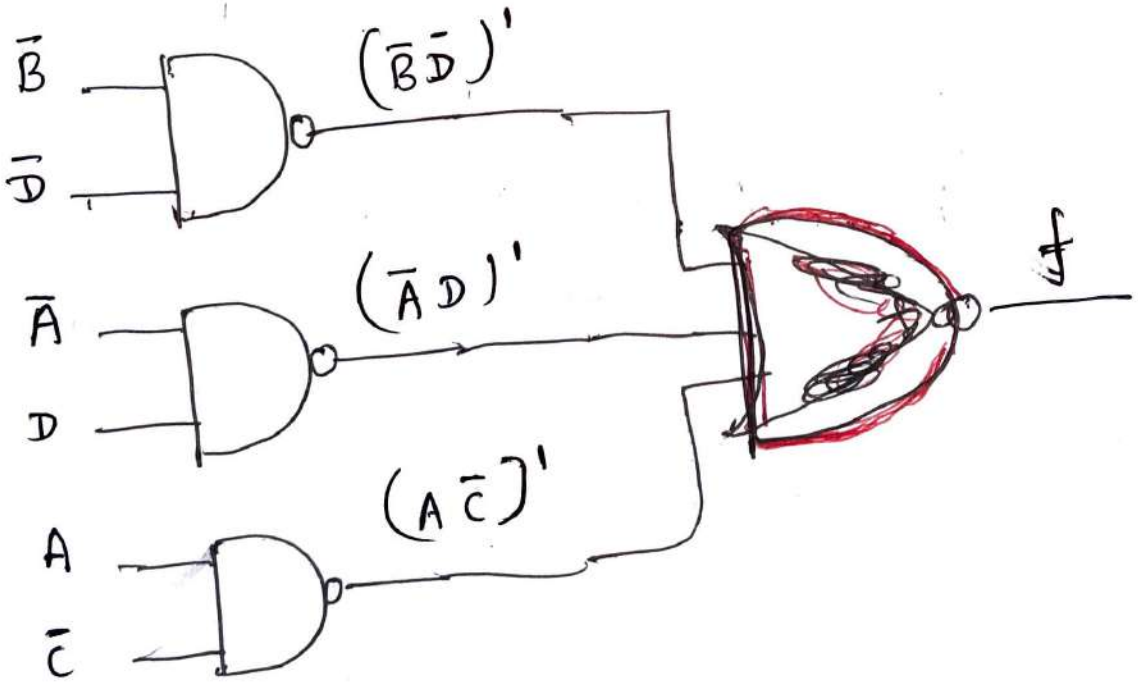
(3) Simplify the following function using K-map $f(A, B, C, D) = \sum m(0, 1, 2, 3, 5, 7, 8, 9, 10, 12, 13)$

And implement the output with only NAND gate.

Sol:→




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



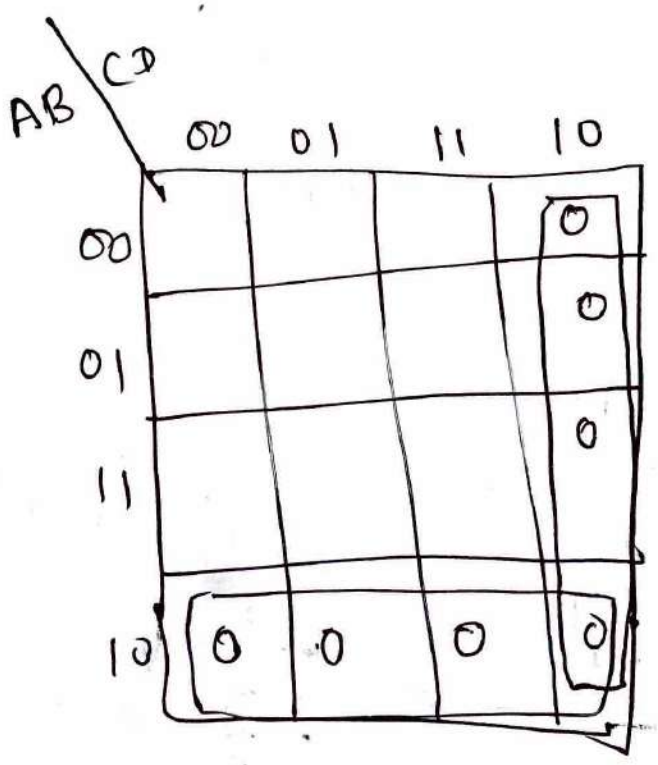
$$f = \left[ (\bar{B}\bar{D})' \cdot (\bar{A}D)' \cdot (A\bar{C})' \right]'$$

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

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

(4) $f(A,B,C,D) = \Pi M(2,6,8,9,10,11,14)$

Simplify the Boolean function using K-map
Also implement the Simplify function using logic gates.

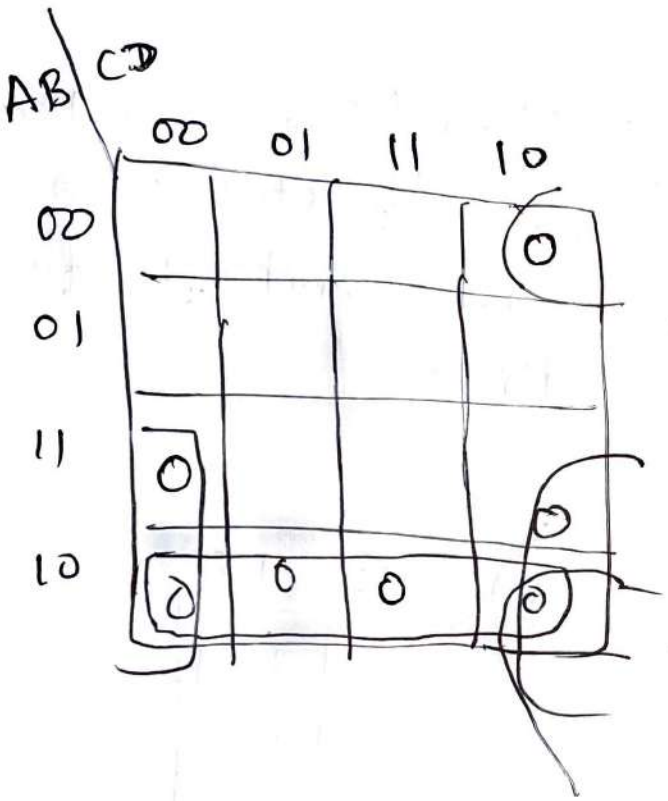



$$f = (A' + B)(C' + D)$$

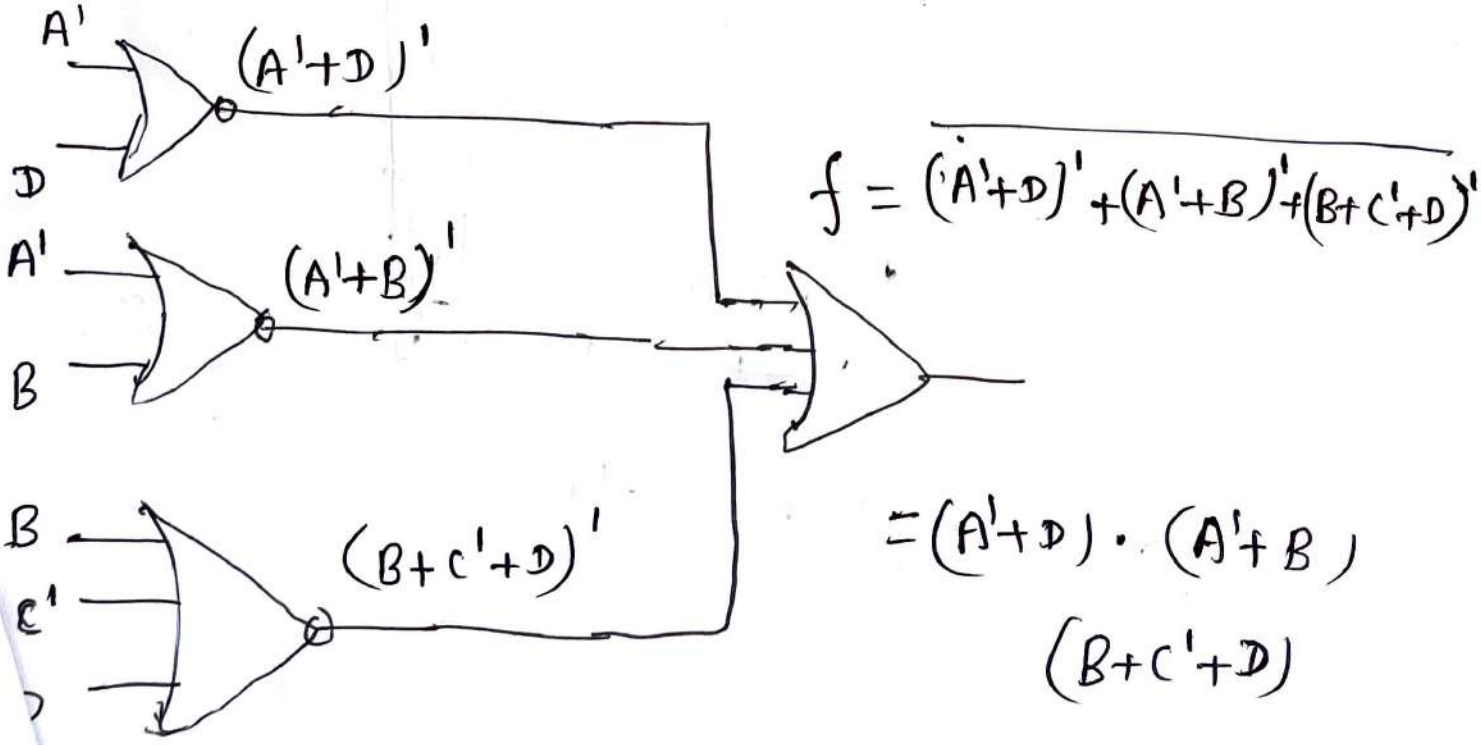
(5) Simplify the following function using K-map and minimized the output through the NOR Gate only.

Sol:→

$$f(A,B,C,D) = \pi M(2,8,9,10,11,12,14)$$




$$f = (A'+D)(A'+B)(B+C'+D)$$




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

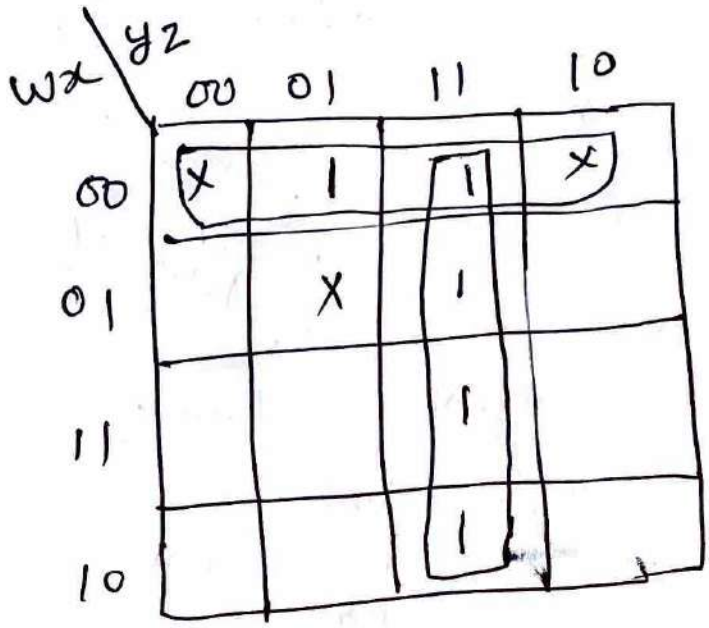
$$= (A'+D)\cdot(A'+B)(B+C'+D)$$

(6) Simplify the Boolean function

$f(w,x,y,z) = \Sigma(1,3,7,11,15)$

$d(w,x,y,z) = \Sigma(0,2,5)$

Sol:→




$f = yz + w'x'$

or

$yz + w'z$

(7) Simplify the following

$f(A,B,C,D) = \Sigma(3,4,5,7,9,13,14,15)$

Simplify the following function using Quin-McCluskey Method

(1) $f(A, B, C, D) = \Sigma m(0, 1, 3, 7, 8, 9, 11, 15)$

Sol:→

| | A | B | C | D |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 |
| 11 | 1 | 0 | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 |

Step 1 →

| Group | Minterm | Binary Rep. |
|---|---|---|
| 0 | $m_0$ | 0000 |
| 1 | $m_1$ | 0001 |
| | $m_8$ | 1000 |
| 2 | $m_3$ | 0011 |
| | $m_9$ | 1001 |
| 3 | $m_7$ | 0111 |
| | $m_{11}$ | 1011 |
| 4 | $m_{15}$ | 1111 |

Step 2 →

| Group | Matched pair | Binary Rep. A B C D |
|---|---|---|
| 0 | $m_0 - m_1$ | 0 0 0 - |
| | $m_0 - m_8$ | - 0 0 0 |
| 1 | $m_1 - m_3$ | 0 0 - 1 |
| | $m_1 - m_9$ | - 0 0 1 |
| | $m_8 - m_9$ | 1 0 0 - |
| 2 | $m_3 - m_7$ | 0 - 1 1 |
| | $m_3 - m_{11}$ | - 0 1 1 |
| | $m_9 - m_{11}$ | 1 0 - 1 |
| 3 | $m_7 - m_{15}$ | - 1 1 1 |
| | $m_{11} - m_{15}$ | 1 - 1 1 |

Step 3 →

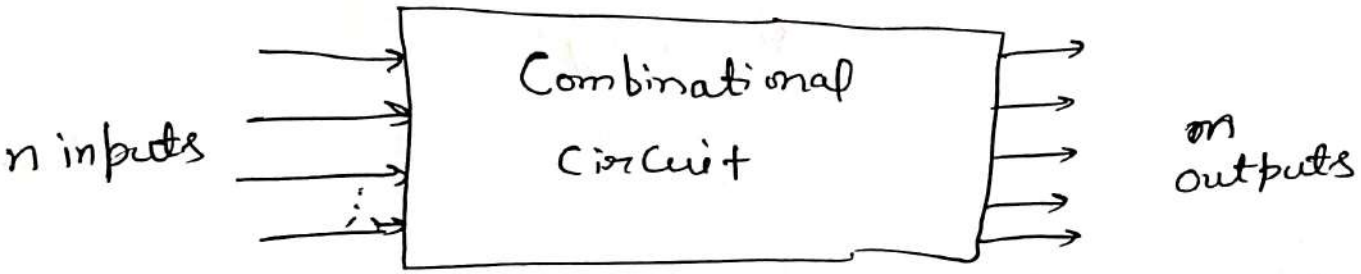
| Group | Matched pair | Binary Rep. A | B | C | D | |
|---|---|---|---|---|---|---|
| 0 | $m_0 - m_1 - m_8 - m_9$ | – | 0 | 0 | – | $\bar{B}\bar{C}$ |
| | $m_0 - m_8 - m_1 - m_9$ | – | 0 | 0 | – | |
| 1 | $m_1 - m_3 - m_9 - m_{11}$ | – | 0 | – | 1 | $\bar{B}D$ |
| | $m_1 - m_9 - m_3 - m_{11}$ | – | 0 | – | 1 | |
| 2 | $m_3 - m_7 - m_{11} - m_{15}$ | – | – | 1 | 1 | $CD$ |
| | $m_3 - m_{11} - m_7 - m_{15}$ | – | – | 1 | 1 | |

Step 4 →

| | Minterm | 0 | 1 | 3 | 7 | 8 | 9 | 11 | 15 |
|---|---|---|---|---|---|---|---|---|---|
| $\bar{B}\bar{C}$ | 0, 1, 8, 9 | (X) | X | | | (X) | X | | |
| $\bar{B}D$ | 1, 3, 9, 11 | | X | X | | | X | X | |
| $CD$ | 3, 7, 11, 15 | | | X | (X) | | | X | (X) |

$$f = \bar{B}\bar{C} + CD$$

## Combinational Circuit →




The logic diagram of a Combinational Circuit has logic gates with no feedback paths or memory elements. Logic circuits for digital systems may be Combinational or Sequential. A logic Circuit is Combinational if outputs at any time are a function of only the present inputs.

**Binary adder - Subtractor** → A Combinational Circuit that performs the addition of two bits is called a half adder. One that performs the addition of three bits (two significant bits and a previous Carry) is a full adder.

Ex/ and

**Half adder** → This circuit needs two binary inputs and two binary outputs.

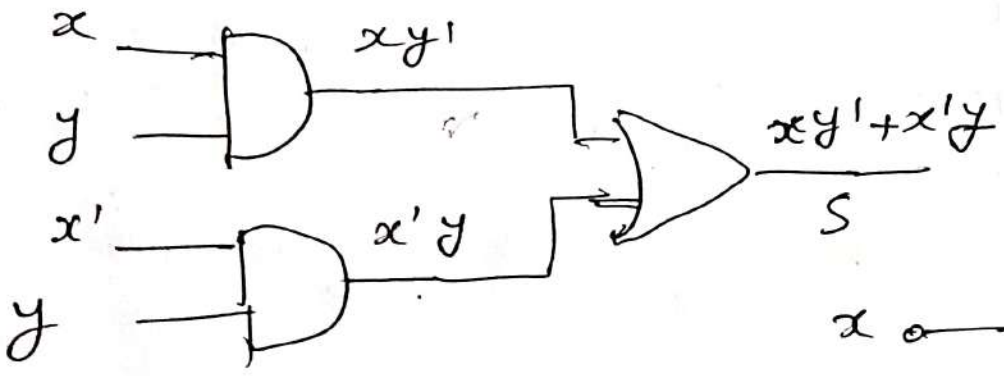
$$S = x'y + xy'$$

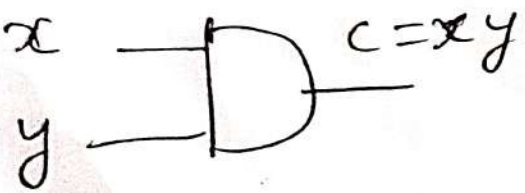
$$C = xy$$

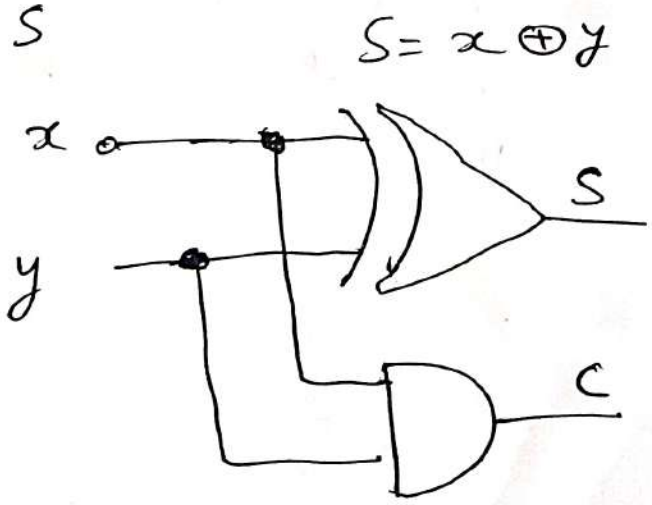
| x | y | Sum | Carry |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

$$\text{Sum}(S) = xy' + x'y = x \oplus y$$

$$C = xy$$

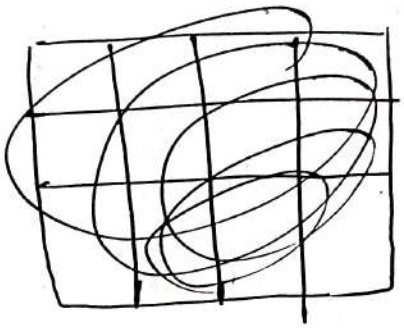


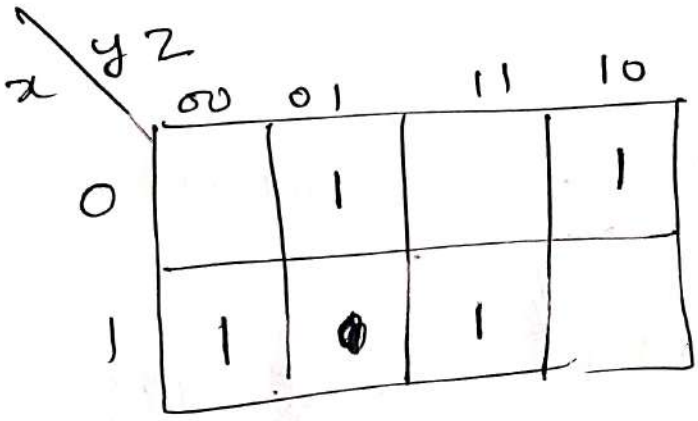



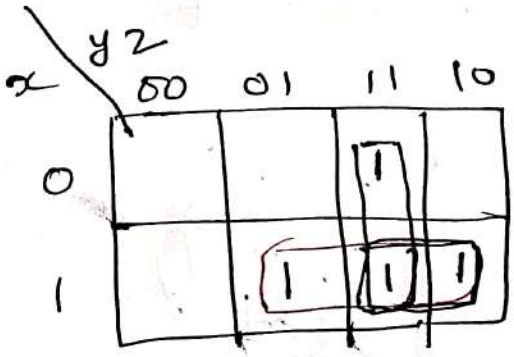



Full adder → It Consists three inputs and two outputs

| x | y | z | S | C |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |






$f = x'y'z + x'yz' + xy'z' + xyz$

$= z'(x'y + xy') + z(x'y' + xy)$

$= z'A + zA'$

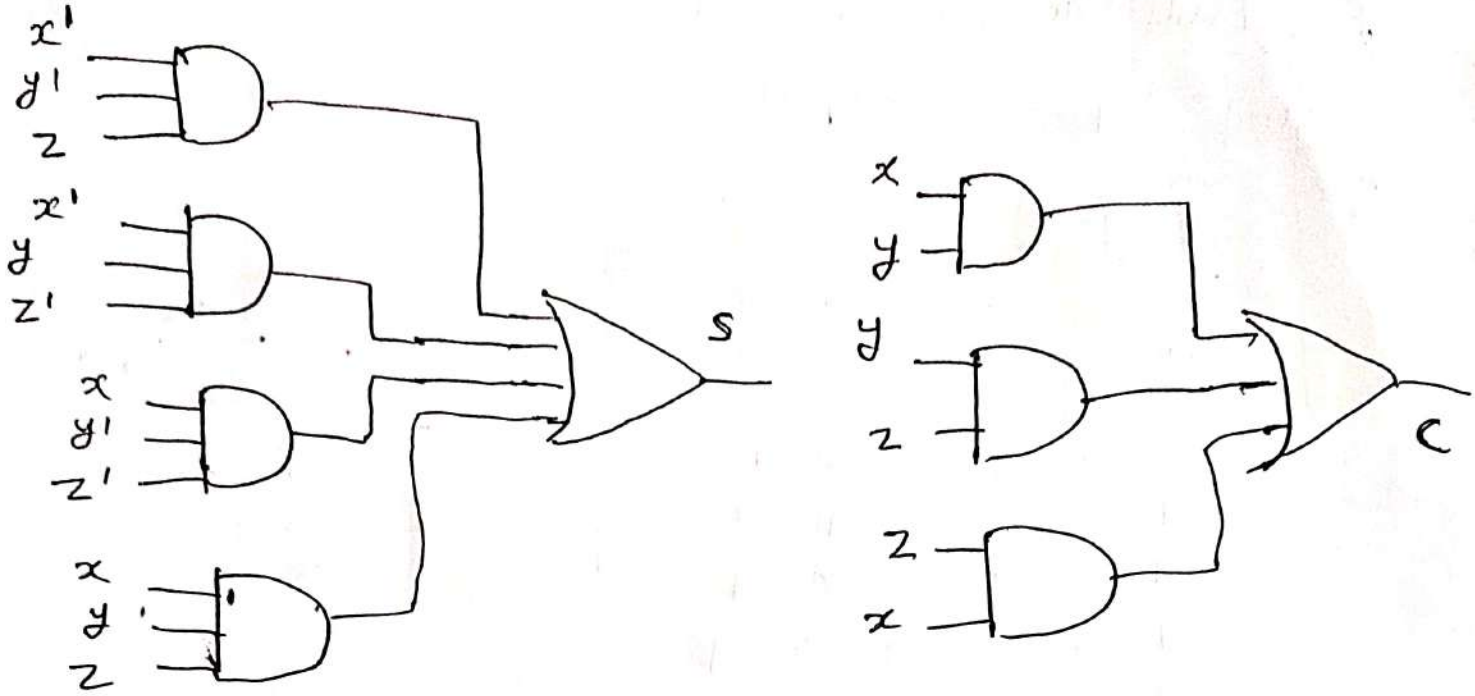
$= x \oplus y \oplus z$

$C = yz + xy + xz$

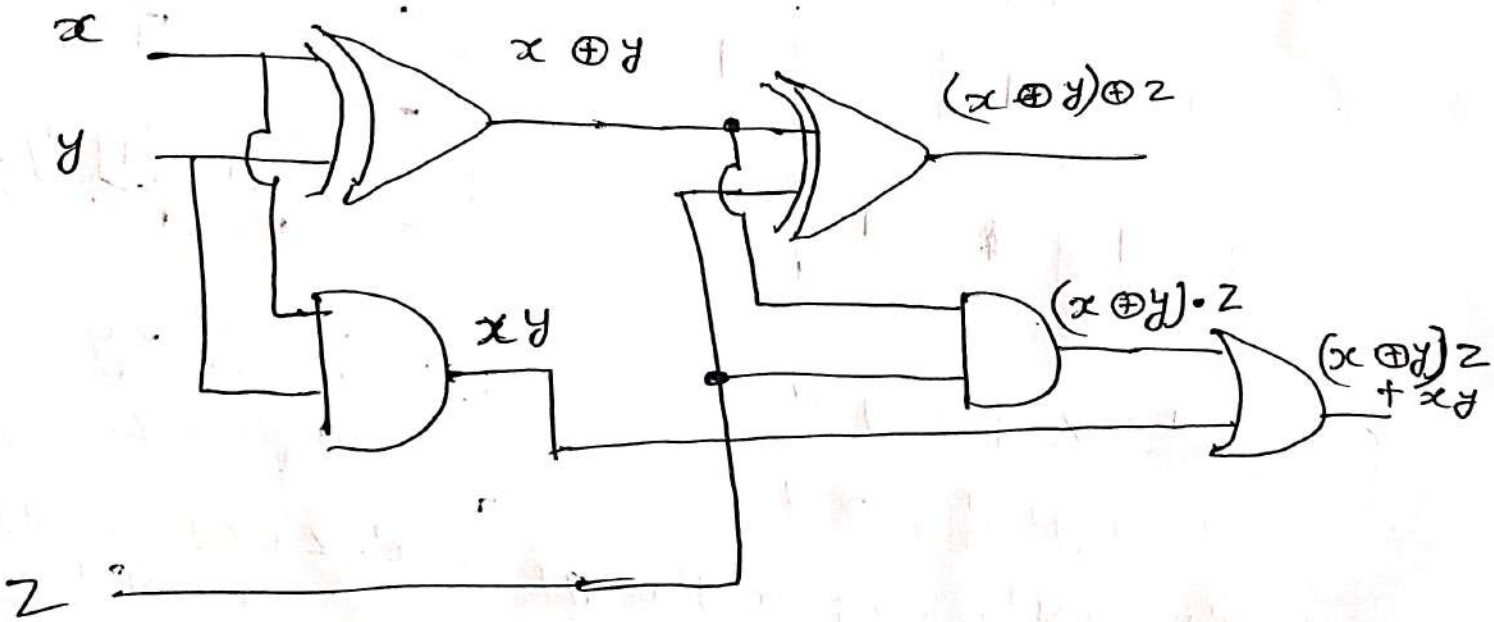
$= x'yz + xy'z + xyz + xyz'$

$= (x'y + xy')z + xy(z + z')$

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



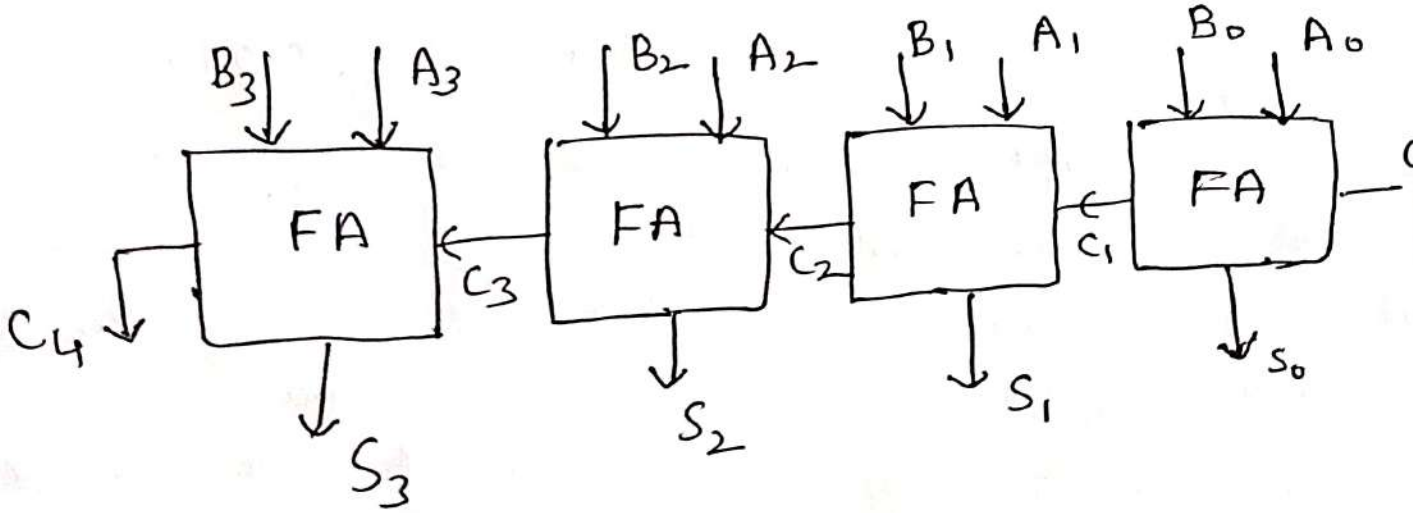

Implementation of full adder in sum of products form →




Implementation of full adder with two half adder and an OR Gate

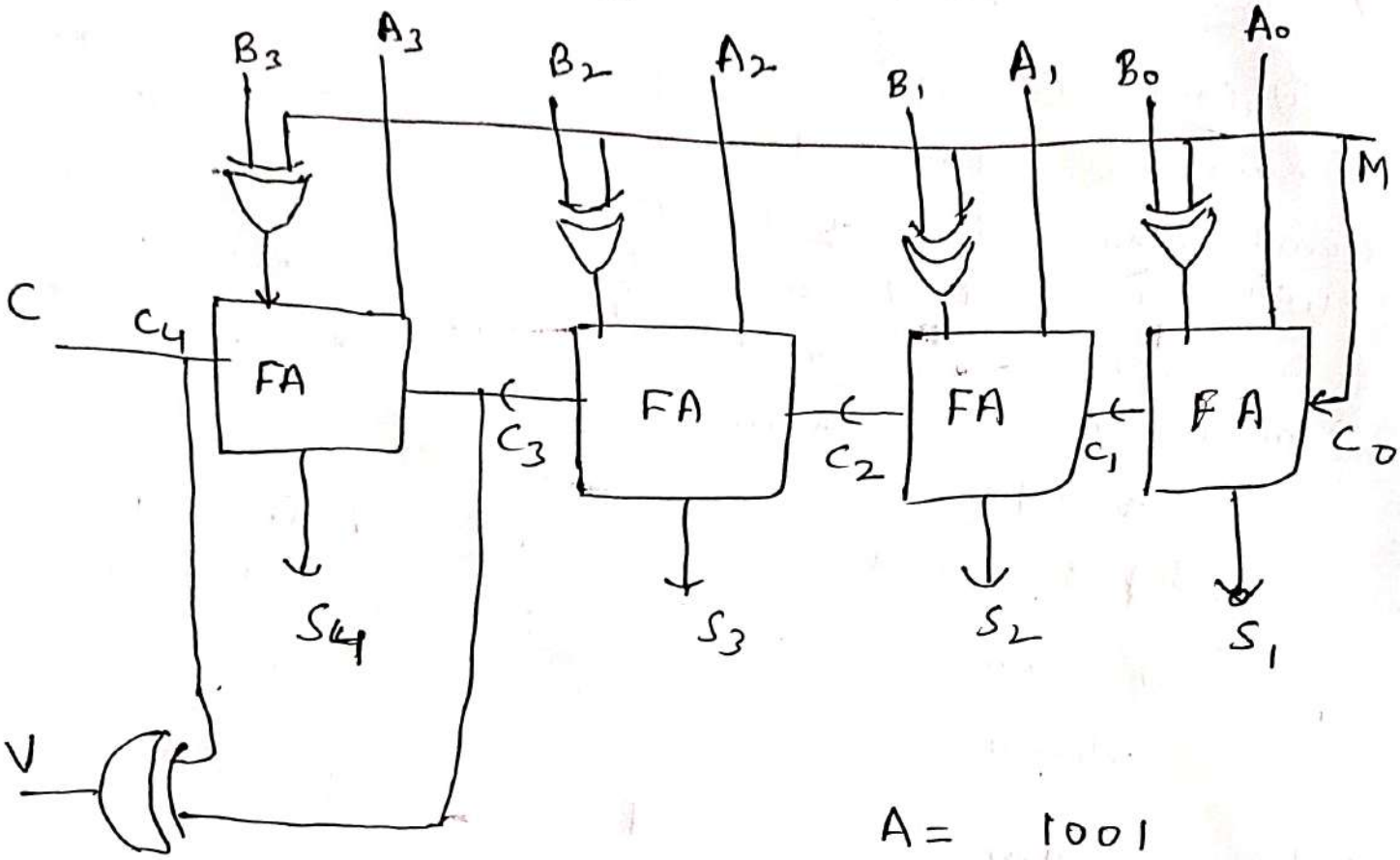
## Binary Adder → A binary adder is a digital circuit that produces the ~~sum~~ arithmetic sum of two binary numbers. It can be Constructed with full adders Connected in cascade, with the output Carry from each full adder Connected to the input Carry of the next full adder in the chain.

| Subscript | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|
| Input Carry | 0 | 1 | 1 | 0 | $C_i$ |
| Augend | 1 | 0 | 1 | 1 | $A_i$ |
| Addend | 0 | 0 | 1 | 1 | $B_i$ |
| Sum | 1 | 1 | 1 | 0 | $S_i$ |
| Output Carry | 0 | 0 | 1 | 1 | $C_{i+1}$ |




Four bit adder

## Binary Subtraction →




A = 1001

B = 0110

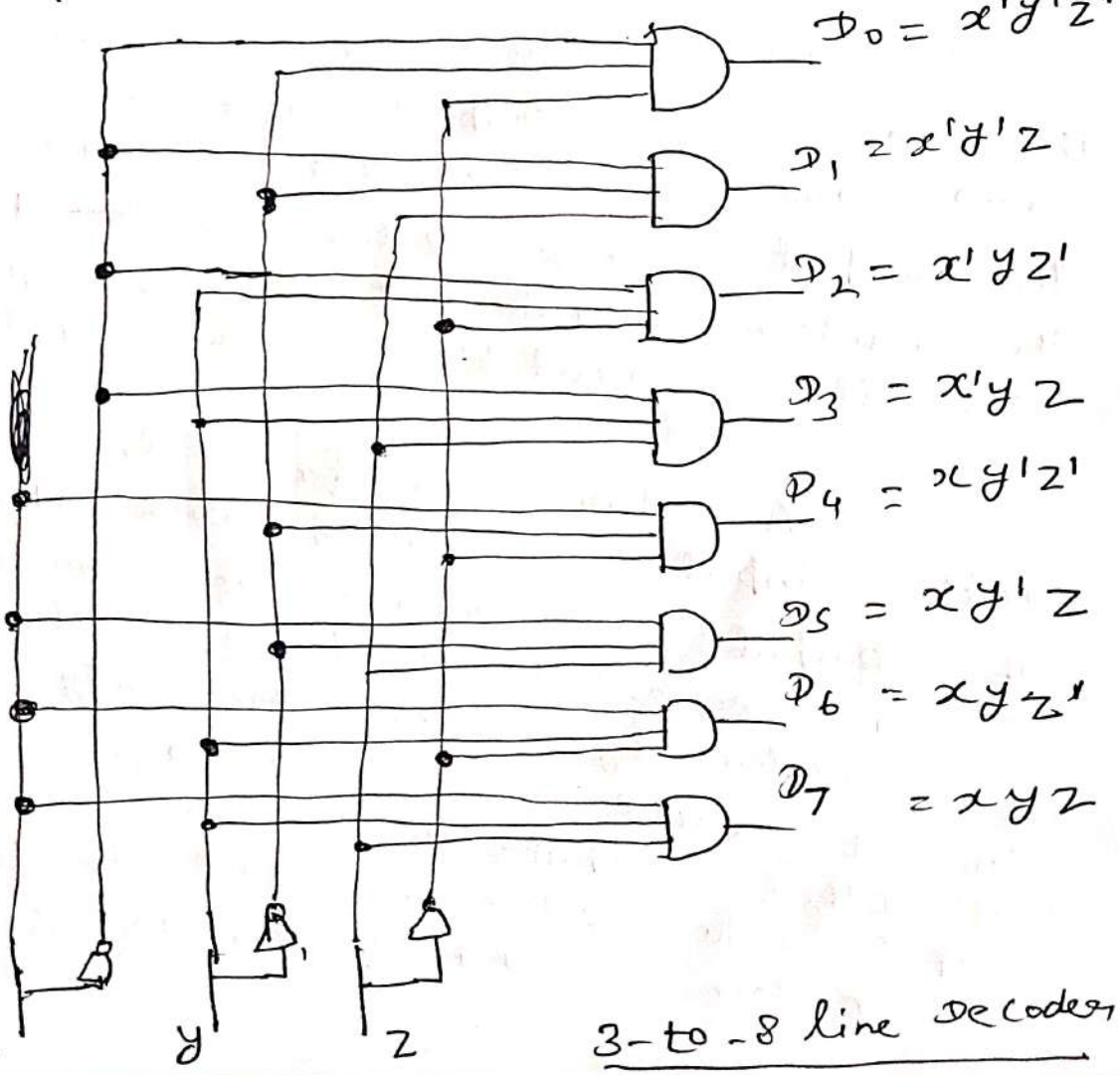
A − B = 10011

The addition and subtraction operations can be combined into one circuit with one common binary adder by including an XOR gate with each full adder.

The mode input M controls the operation, when M = 0, the circuit is an adder and when M = 1 the circuit becomes an subtractor.

- **Decoders** → A decoder is a combinational circuit that converts binary information from $n$ input lines to a maximum of $2^n$ unique output lines.

The decoders presented here are called $n$ to $m$ line decoders, where $m < 2^n$.
The three inputs are decoded into eight outputs, each representing one of the minterms of three input variables. In 3 to 8 line decoder. The three inputs are decoded into eight outputs, each representing one of the minterms of the three input variables.




3-to-8 line decoder

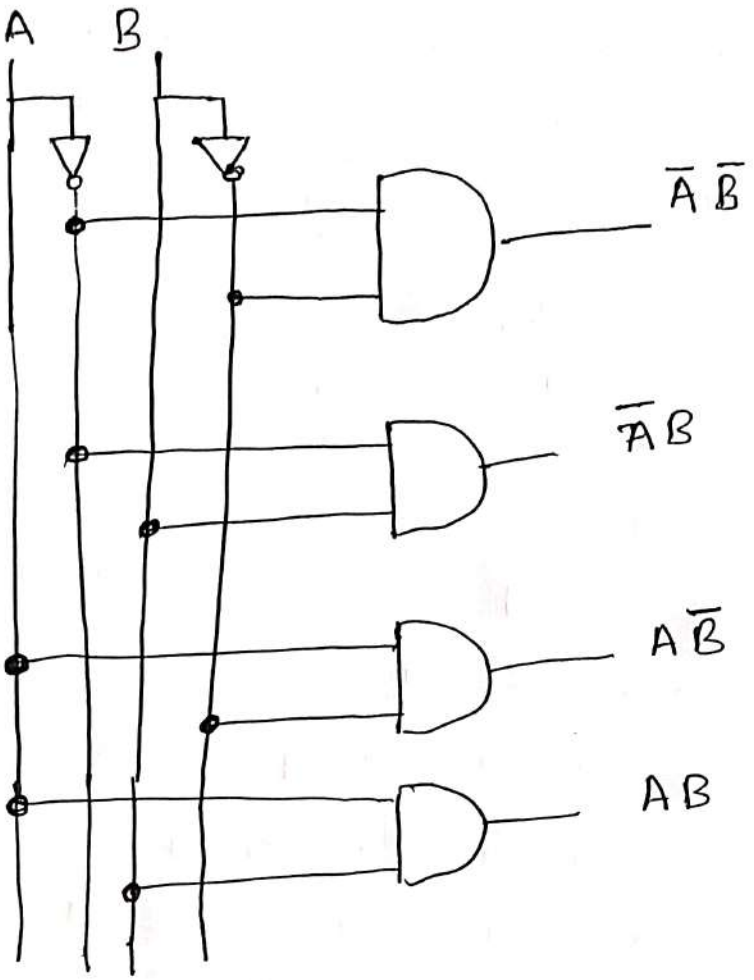
## Truth table of 3-8 3 to 8 line decoder

| Inputs | | | Outputs | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| x | y | z | $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

For each possible input combination, there are seven outputs that are equal to zero and only one that is equal to 1. The output whose value is 1 represents the minterms equivalent to binary number currently available in the input lines.

A two to four line decoder with enable input constructed with NAND gates. The outputs of decoder are enabled when E is equal to 0 (i.e. active low enable). As indicated by truth table, only one output can be equal to 0 at any given time, all other outputs are equal to 1. The output whose value is equal to 0 represents the minterm selected by inputs A and B.

## 2 to 4 line Decoder Constructed with AND gate :→




The 2-to-4 line binary decoder constructed with AND gates. The 2 binary inputs labelled A and B are decoded into one of 4 outputs. Each output represents one of the minterm of the 2 input variable.

| A | B | $O_0$ | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

Truth table

**Encoder** → An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has $2^n$ (or fewer) input lines and $n$ output lines.

**Octal to binary Encoder** →

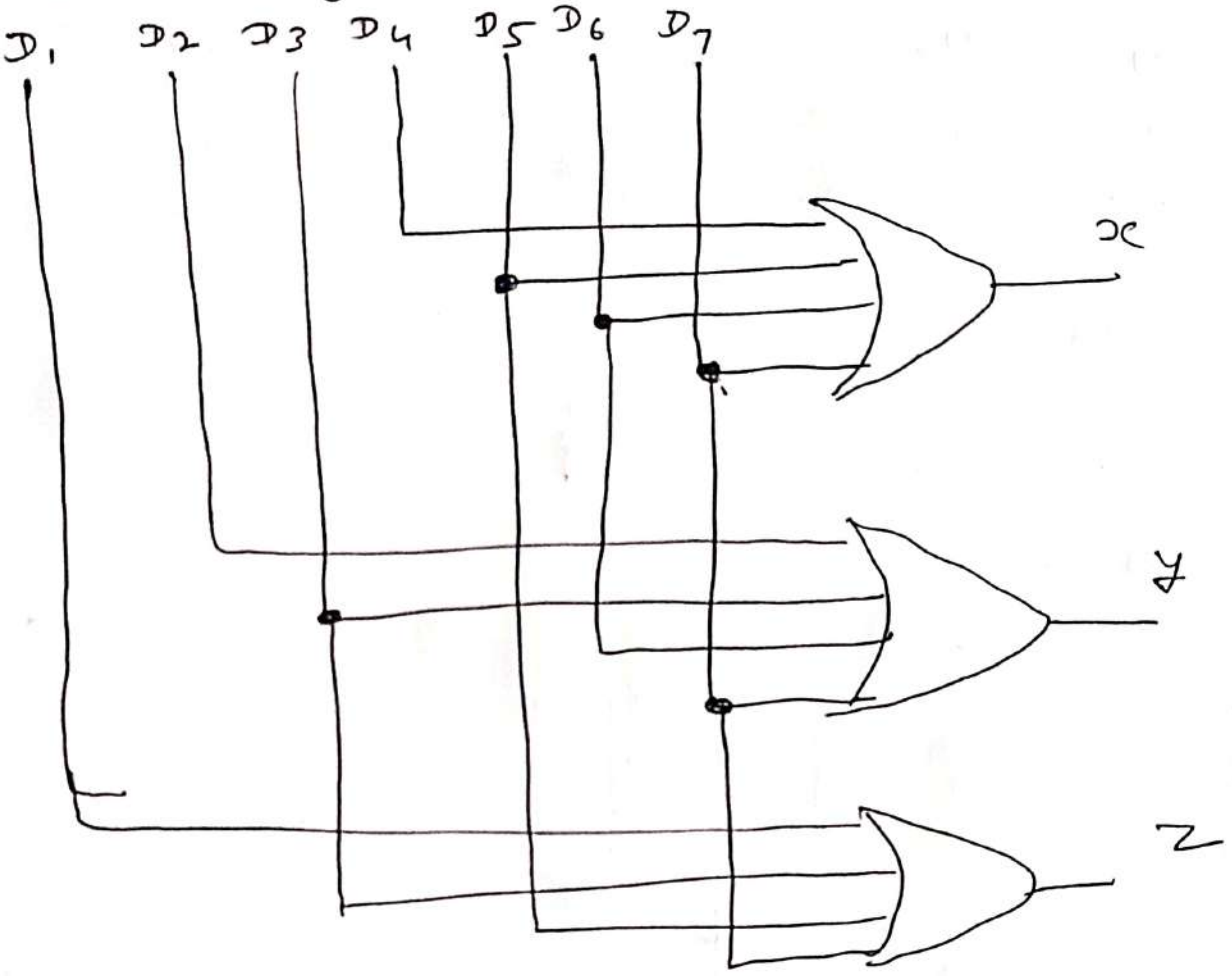
| Inputs | | | | | | | | outputs | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ | x | y | z |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

$$z = D_1 + D_3 + D_5 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

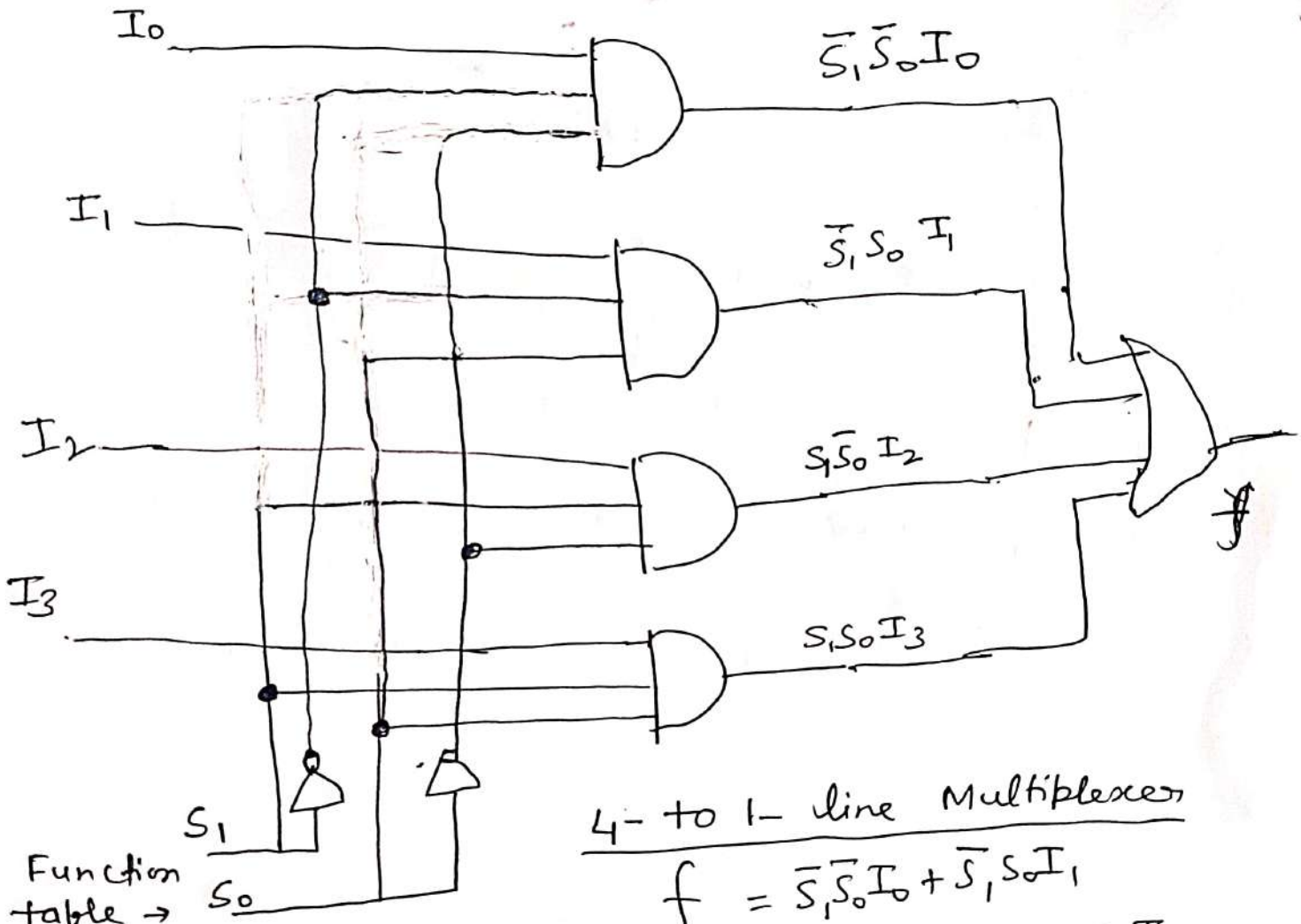
$$x = D_4 + D_5 + D_6 + D_7$$

An octal to binary encoder (8-line to 3 line encoder) accepts 8 input lines and produces a 3 bit output code corresponding to the activated input.




Octal to binary encoder

**Multiplexer** → A multiplexer is a Combinational Circuit that selects binary information from one of many input line and direct it to a single output line. Normally, there are $2^n$ input lines and n selection lines whose bit Combination determine which input is selected.

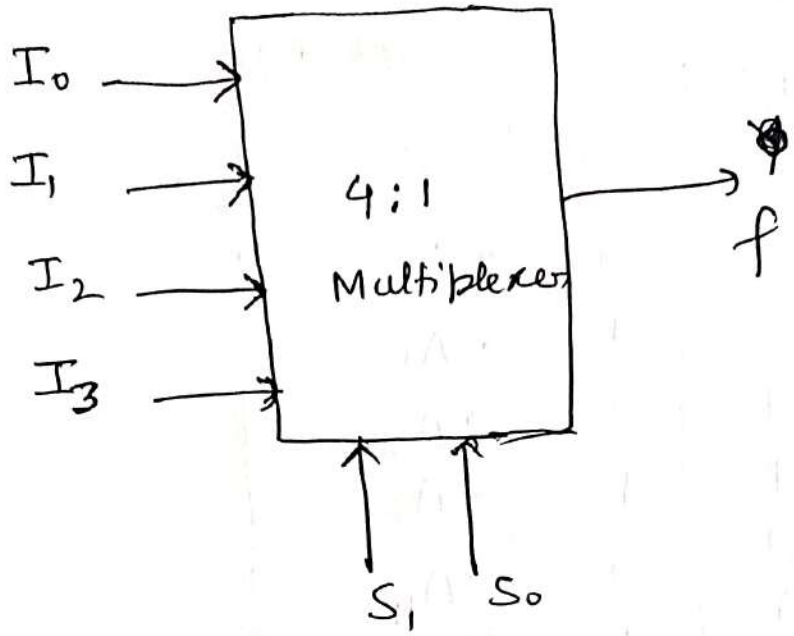



4- to 1- line Multiplexer

$$f = \bar{S}_1\bar{S}_0I_0 + \bar{S}_1S_0I_1 + S_1\bar{S}_0I_2 + S_1S_0I_3$$

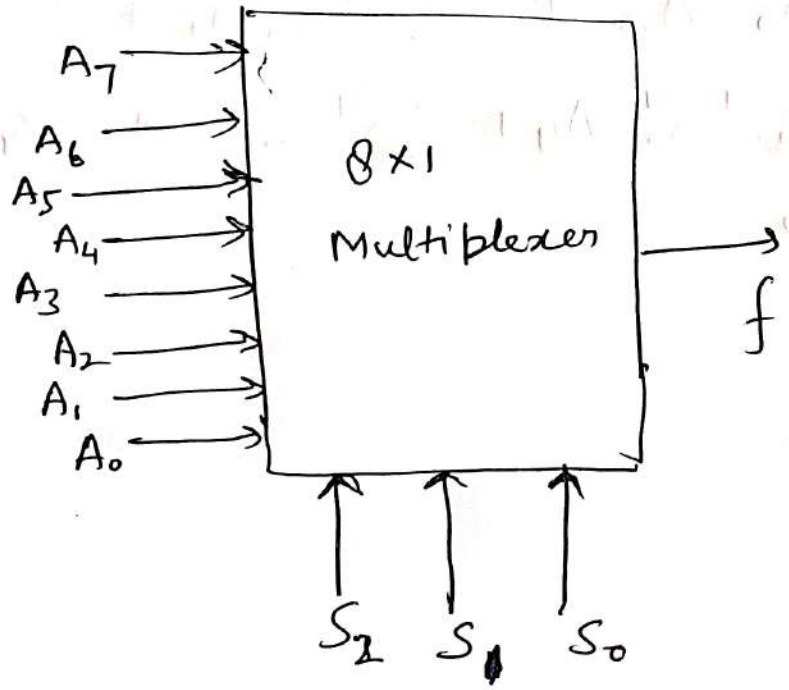
Function table →

| $S_1$ | $S_0$ | Y |
|---|---|---|
| 0 | 0 | $I_0$ |
| 0 | 1 | $I_1$ |
| 1 | 0 | $I_2$ |
| 1 | 1 | $I_3$ |




4:1 Multiplexer

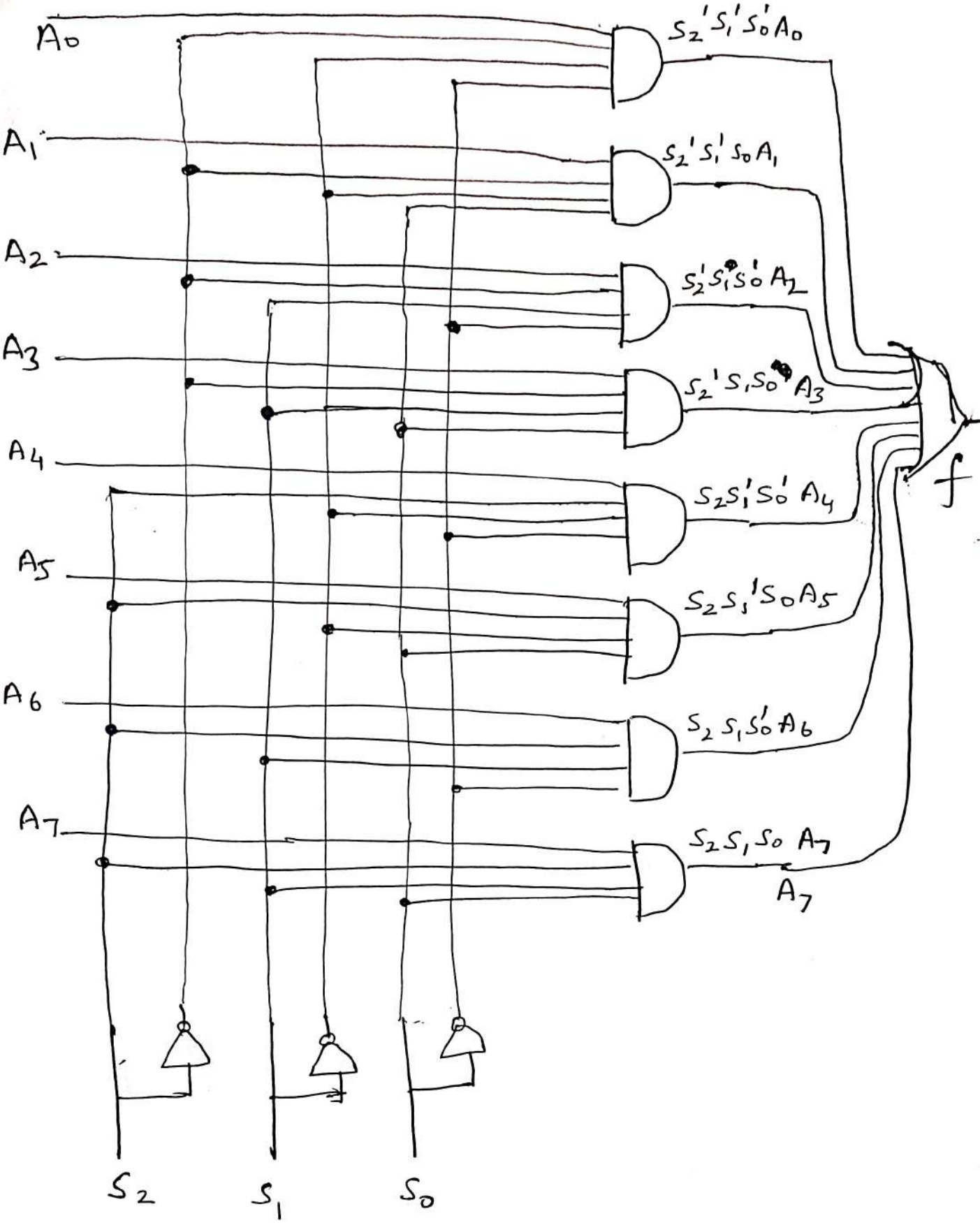
8 to 1 line Multiplexer



Truth table

| Inputs | | | Output |
|---|---|---|---|
| $S_2$ | $S_1$ | $S_0$ | $f$ |
| 0 | 0 | 0 | $A_0$ |
| 0 | 0 | 1 | $A_1$ |
| 0 | 1 | 0 | $A_2$ |
| 0 | 1 | 1 | $A_3$ |
| 1 | 0 | 0 | $A_4$ |
| 1 | 0 | 1 | $A_5$ |
| 1 | 1 | 0 | $A_6$ |
| 1 | 1 | 1 | $A_7$ |

$$f = S_2' S_1' S_0' A_0 + S_2' S_1' S_0 A_1 + S_2' S_1 S_0' A_2 + S_2' S_1 S_0 A_3 + S_2 S_1' S_0' A_4 + S_2 S_1' S_0 A_5 + S_2 S_1 S_0' A_6 + S_2 S_1 S_0 A_7$$


A0
A1
A2
A3
A4
A5
A6
A7
S2
S1
S0
S2'S1'S0'A0
S2'S1'S0A1
S2'S1S0'A2
S2'S1S0A3
S2S1'S0'A4
S2S1'S0A5
S2S1S0'A6
S2S1S0A7
f

(1) Implement the following function with a Mux.

$F(a, b, c) = \Sigma m(1, 3, 5, 6)$

Choose a and b as select inputs

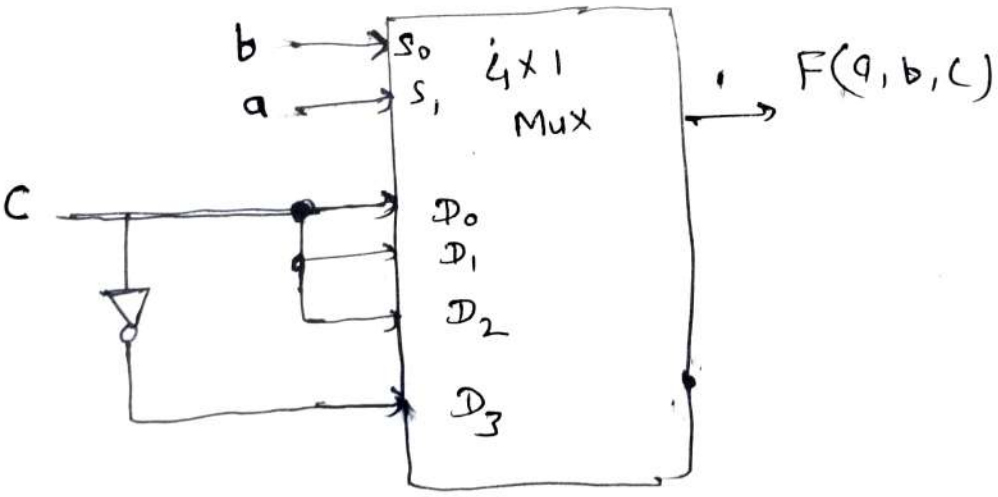
Sol:→

| $S_1$ | $S_0$ | | | |
|---|---|---|---|---|
| a | b | c | F | |
| 0 | 0 | 0 | 0 | F = c |
| 0 | 0 | 1 | 1 | |
| 0 | 1 | 0 | 0 | F = c |
| 0 | 1 | 1 | 1 | |
| 1 | 0 | 0 | 0 | F = c |
| 1 | 0 | 1 | 1 | |
| 1 | 1 | 0 | 1 | $F = \bar{c}$ |
| 1 | 1 | 1 | 0 | |

Truth table →

Case I →
For both value
ab = 00, ab = 01
ab = 10, F = c
So $D_0$, $D_1$ $D_2$ are connected to c.

Case II
For both values of
ab = 11, $F = \bar{c}$.
So, $D_3$ is connected to $\bar{c}$.




Multiplexer implementation

(2) Implement the function $f(a,b,c) = ab + \bar{b}c$ using 4:1 MUX.

$$f(a,b,c) = ab + \bar{b}c$$

$$= abc + ab\bar{c} + \bar{a}\bar{b}c + a\bar{b}c$$

$$= \Sigma m(1,5,6,7)$$ using a 4:1 MUX with two select input a and b

| $S_1$ | $S_0$ | | | |
|---|---|---|---|---|
| a | b | c | F | |
| 0 | 0 | 0 | 0 | F = c |
| 0 | 0 | 1 | 1 | |
| 0 | 1 | 0 | 0 | F = 0 |
| 0 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 0 | F = c |
| 1 | 0 | 1 | 1 | |
| 1 | 1 | 0 | 1 | F = 1 |
| 1 | 1 | 1 | 1 | |

Case I → For both value of ab = 00 and ab = 10, F = c

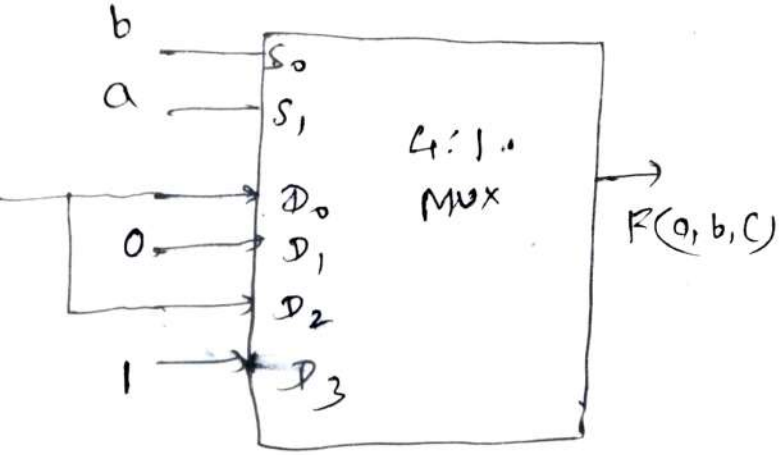
So, $D_0$ and $D_2$ are Connected to c

Case II → ab = 01, F = 0

So $D_1$ is Connected to 0

Case III → ab = 11, F = 1

So $D_3$ is Connected to 1.



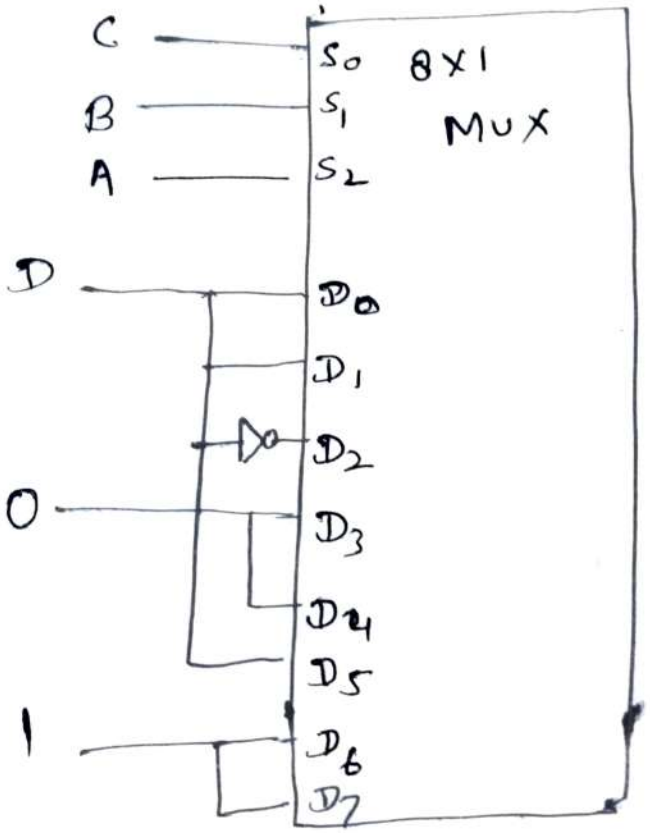
(3) Implement the following logic using 8×1 MUX.

$F(A, B, C, D) = \Sigma m(1, 3, 4, 11, 12, 13, 14, 15)$

Sol:→

$S_2$ $S_1$ $S_0$

| A | B | C | D | F | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | F = D |
| 0 | 0 | 0 | 1 | 1 | |
| 0 | 0 | 1 | 0 | 0 | F = D |
| 0 | 0 | 1 | 1 | 1 | |
| 0 | 1 | 0 | 0 | 1 | F = D' |
| 0 | 1 | 0 | 1 | 0 | |
| 0 | 1 | 1 | 0 | 0 | F = 0 |
| 0 | 1 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 0 | 0 | F = 0 |
| 1 | 0 | 0 | 1 | 0 | |
| 1 | 0 | 1 | 0 | 0 | F = D |
| 1 | 0 | 1 | 1 | 1 | |
| 1 | 1 | 0 | 0 | 1 | F = 1 |
| 1 | 1 | 0 | 1 | 1 | |
| 1 | 1 | 1 | 0 | 1 | F = 1 |
| 1 | 1 | 1 | 1 | 1 | |

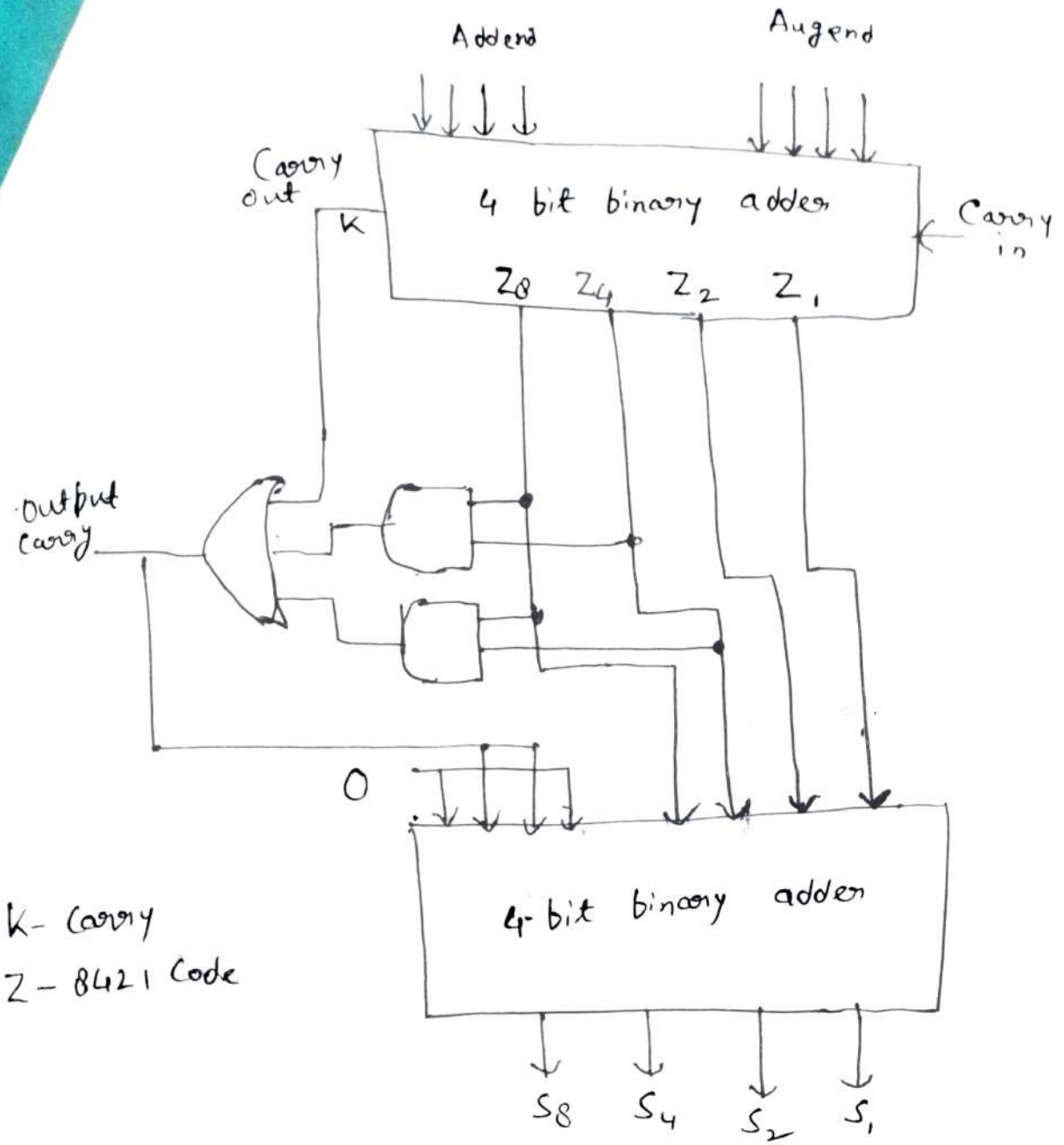

C
B
A
D
0
1
$S_0$
$S_1$
$S_2$
8 X 1
MUX
$D_0$
$D_1$
$D_2$
$D_3$
$D_4$
$D_5$
$D_6$
$D_7$

## Decimal Adder →

## BCD Adder →

## Derivation of BCD Adder →

| | Binary Sum | | | | BCD Sum | | | | | Decimal |
|---|---|---|---|---|---|---|---|---|---|---|
| K | $Z_8$ | $Z_4$ | $Z_2$ | $Z_1$ | C | $S_8$ | $S_4$ | $S_2$ | $S_1$ | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 3 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 4 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 5 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 6 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 7 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 8 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 9 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 10 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 11 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 12 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 13 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 14 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 15 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 16 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 17 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 18 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 19 |




K- Carry

Z- 8421 Code

BLock diagram of a BCD adder

Consider the arithmetic addition of two decimal digits in BCD together with an input carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than $9+9+1=19$.

When the binary sum is equal to or less than 1001, the corresponding BCD number is identical. and therefore no Conversion is needed.

When binary sum is greater than 1001, we obtain an invalid BCD representation. The addition of binary 6 (0110) to binary sum converts it to the correct BCD representation and also produces an output carry as required.

The correction is needed when binary sum has an output carry $K = 1$. The other six combination from 1010 through 1111 that need a correction have a 1 in position $Z_8$. To distinguish them from binary 1000 and 1001, which have a 1 in position $Z_8$. Then either $Z_4$ or $Z_2$ must have 1.

The condition for a correction and an output carry can be expressed as.

$$C = K + Z_8 Z_4 + Z_8 Z_2$$

when $C = 1$, it is necessary to add 0110 to binary sum and provide an output carry for next stage

design

## 4 Bit Binary to Gray Code Converter →

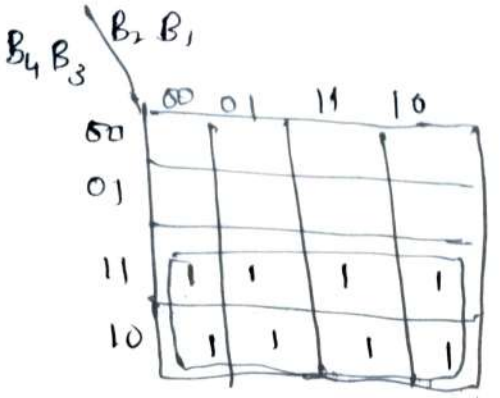
| 4 bit binary | | | | 4 bit Gray | | | |
|---|---|---|---|---|---|---|---|
| $B_4$ | $B_3$ | $B_2$ | $B_1$ | $G_4$ | $G_3$ | $G_2$ | $G_1$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 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 |

$G_4 = \Sigma m(8, 9, 10, 11, 12, 13, 14, 15)$

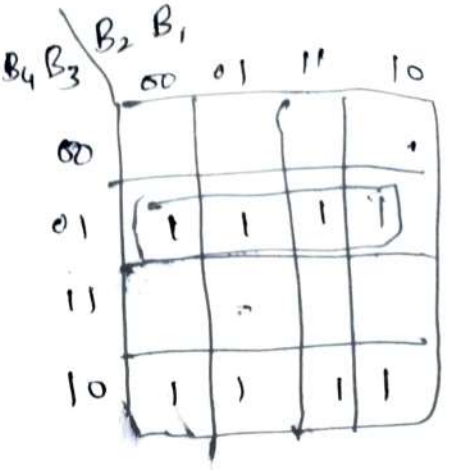
$G_3 = \Sigma m(4, 5, 6, 7, 8, 9, 10, 11)$

$G_2 = \Sigma m(2, 3, 4, 5, 10, 11, 12, 13)$

$G_1 = \Sigma m(1, 2, 5, 6, 9, 10, 13, 14)$

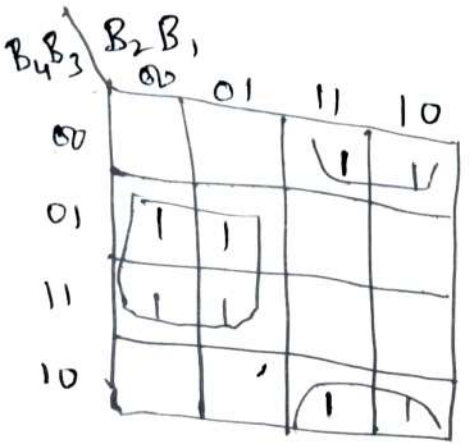



$G_4 = B_4$



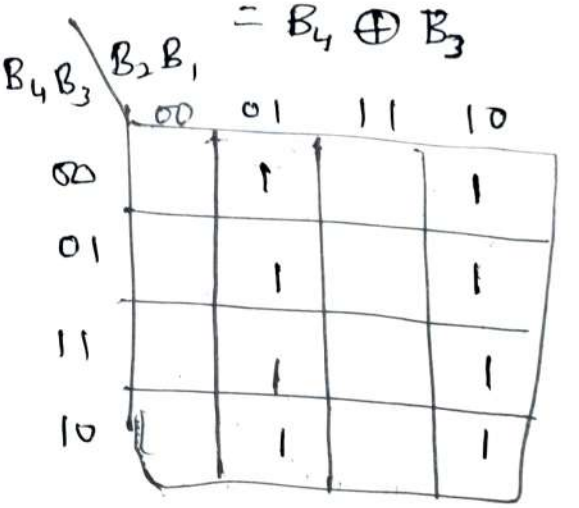

$G_3 = B_4' B_3 + B_4 B_3'$

$= B_4 \oplus B_3$



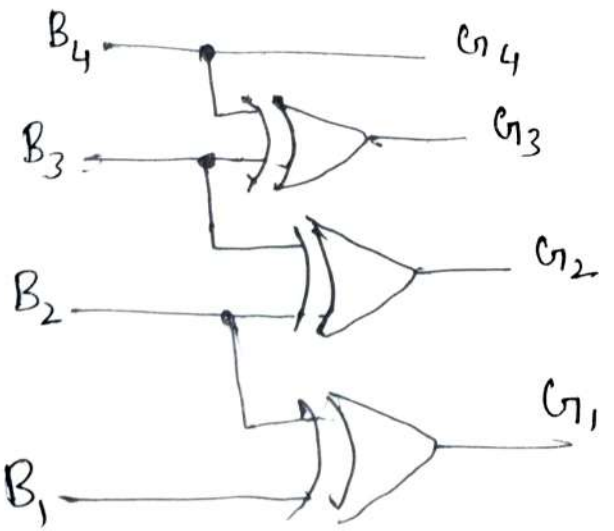

$G_2 = B_3 B_2' + B_3' B_2$

$G_2 = B_3 \oplus B_2$




$G_1 = B_2' B_1 + B_2 B_1'$

$G_1 = B_2 \oplus B_1$




4 bit binary to Gray Converter

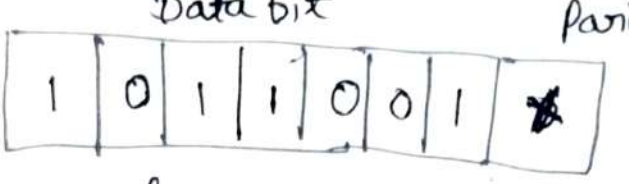
## Parity Generator and Parity Checker →

Parity is the additional bit which is used with the data bits for the error detection there are two types of Parity

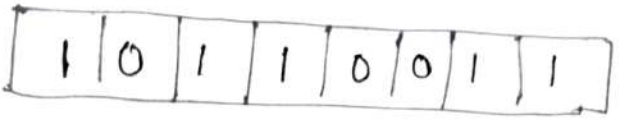
(I) Odd Parity (II) even Parity

**Odd Parity →** In odd parity, total number of 1's in the code, including the parity bit should be equal to odd

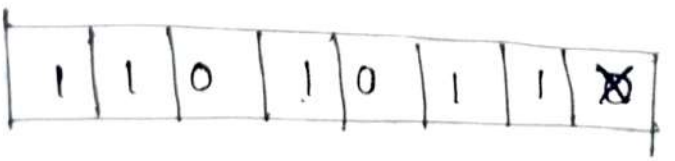
Case I

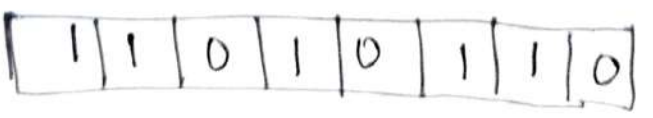



If we have 7 data bits. the total no. of 1's in the data bits is equal to four (even). Therefore for odd parity this parity bits should be equal to 1.

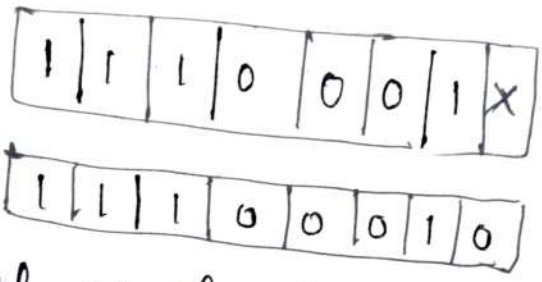



Case II If total no of 1's in data bit is odd then for odd parity, this parity bit will be equal to Zero.



Even Parity → Total no of 1's in the code, including the Parity bit should be even




total no of 1 in data bit is even. then this Parity bit is 0.

Even Parity Generator → (3 bit)

| | A | B | C | P |
|---|---|---|---|---|
| | 0 | 0 | 0 | 0 |
| odd | 0 | 0 | 1 | 1 |
| odd | 0 | 1 | 0 | 1 |
| | 0 | 1 | 1 | 0 |
| odd | 1 | 0 | 0 | 1 |
| | 1 | 0 | 1 | 0 |
| | 1 | 1 | 0 | 0 |
| odd | 1 | 1 | 1 | 1 |

The output P will be 1 when total number of 1 in the input is odd.

$P = A'B'C + A'BC' + AB'C' + ABC$

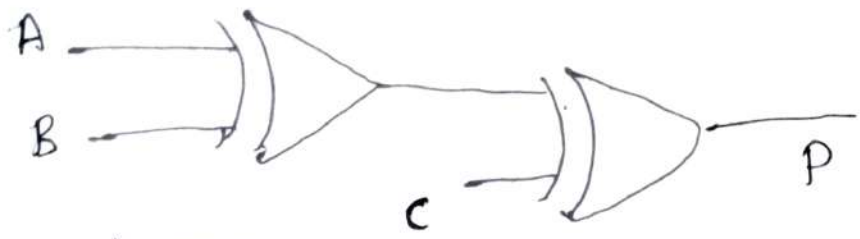
$= A'(B'C + BC') + A(B'C' + BC)$

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

($B \oplus C$ = X, $\overline{B \oplus C}$ = X')

$= A'X + AX'$

$= A' \oplus B \oplus C$




$P = A \oplus B \oplus C$

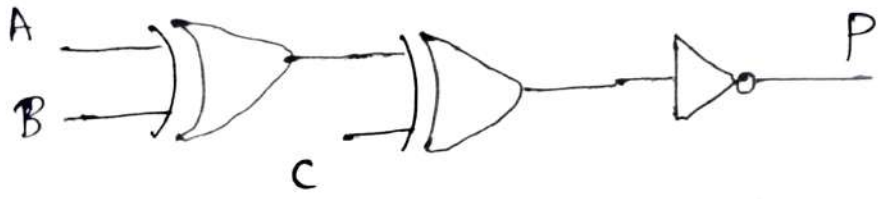
Odd Parity generator → (3 bit)

| A | B | C | P |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

~~$P = A'B'C'$~~

$\overline{P = A \oplus B \oplus C}$

If P output is equal to 1, when the total number of 1's in the data bits is equal to even.



# Even Parity Checker

| A | B | C | P |
|---|---|---|---|
| 1 | 0 | 1 | 0 |

⇒

| A | B | C | P | check |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 |

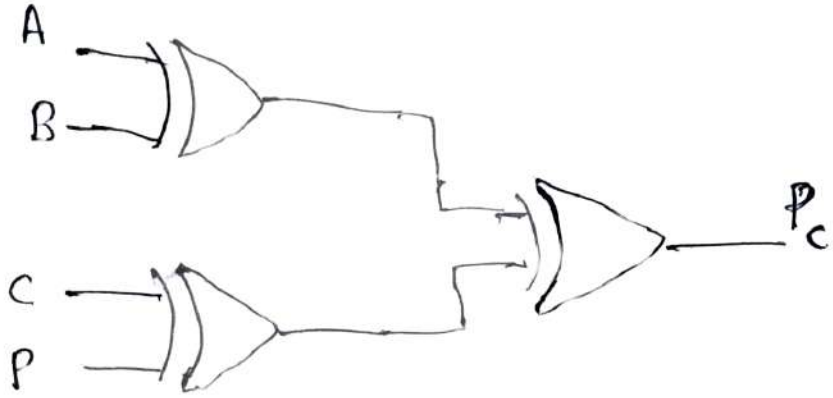
We have three data bits and one Parity bit. This is even Parity bit therefore, $P = 0$. Whenever, there is error output of Parity checker circuit will become 1. When, there is no error then output of Parity checker circuit will become 0.

4 bit received code we have total 16 input Combination

| A | B | C | P | Parity Check (Error) |
|---|---|---|---|---|
| 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 |

$$P_c = A \oplus B \oplus C \oplus P$$

It will generate error when total number of 1's in received code is equal to odd. It will generate error for 8 different input combinations.




Odd Parity Checker →

| A | B | C | P | | A | B | C | P | Check |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | ⇒ | 1 | 0 | 0 | 1 | 1 |

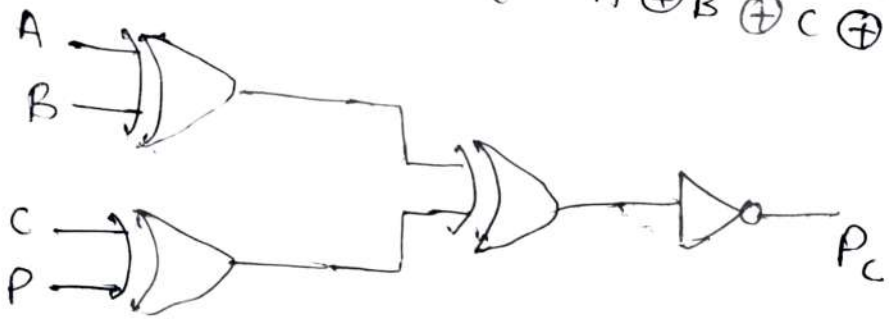
we have three data bit and one Parity bit. This is odd Parity bit then P = 1. In Case error, output of Parity checker circuit should become 1. when there is no error, output of Parity checker will become 0.

| A | B | C | P | Parity Check (Error) |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

It will generate error, when total number of 1's in received code is equal to even

If we compare odd Parity Checker and even Parity Checker In odd Parity Checker output Column is exactly the Complement of even Parity checker.

$$P_c = \overline{A \oplus B \oplus C \oplus P}$$



## 2 Bit Multiplier circuit :→

$A = A_1 \; A_0$

$B = B_1 \; B_0$

$A_1B_0 \quad A_0B_0$

$A_1B_1 \quad A_0B_1$

$P_3 \quad P_2 \quad P_1 \quad P_0$

$P_0 = A_0B_0$

$P_1 = \underbrace{A_1B_0 + A_0B_1}_{HA}$

$P_2 = \underbrace{A_1B_1 + C_1}_{HA}$

$P_3 = C_2$

$A = 1\;1$ (3)

$B \quad 1\;0$ (2)

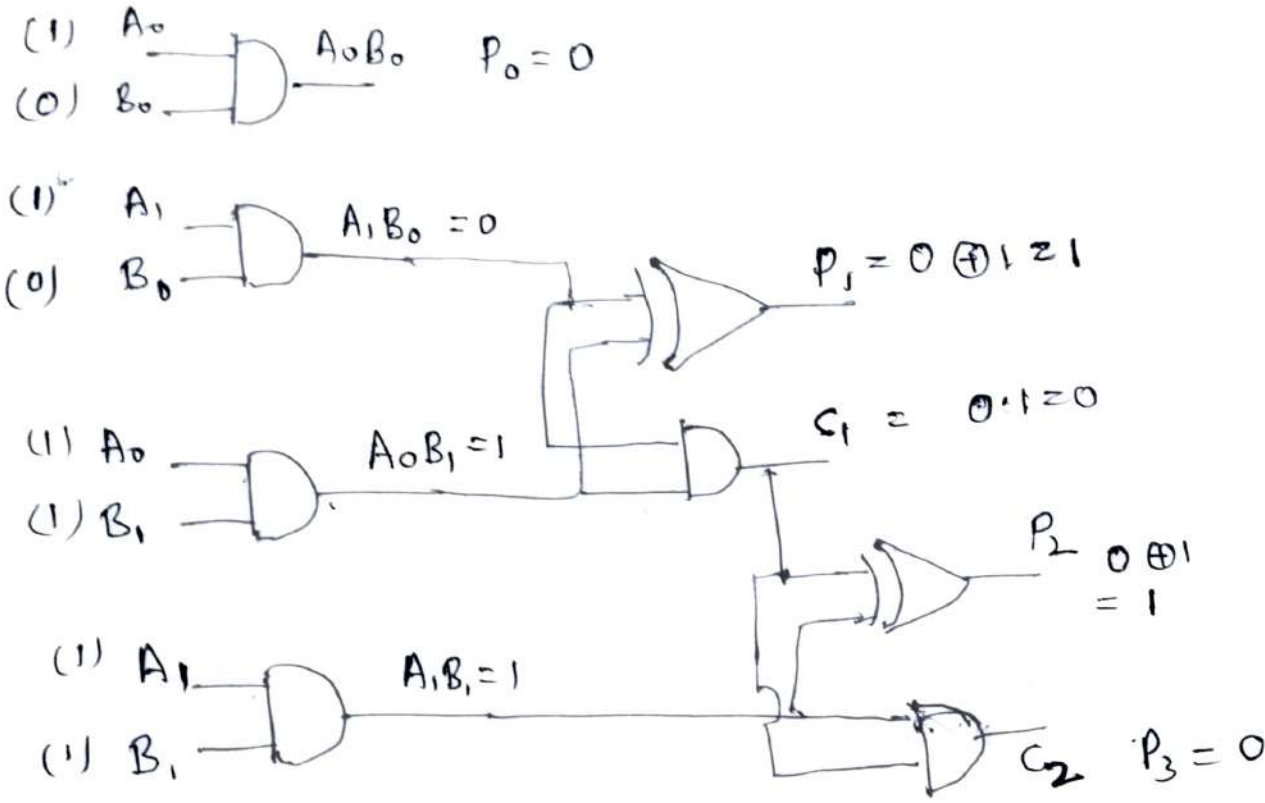
$0\;0$

$1\;1$

$0\;1\;1\;0$ (6)

$P_3 \; P_2 \; P_1 \; P_0$

Multiplication of Binary number is performed in the same way as multiplication of decimal numbers.




2 Bit Binary Multiplier

(1) Design Full adder using 4:1 multiplexer

Sol:-

| Inputs | | | Outputs | | |
|---|---|---|---|---|---|
| A | B | $C_{in}$ | Sum | Carry | |
| 0 | 0 | 0 | 0 | 0 | $D_0$ |
| 0 | 0 | 1 | 1 | 0 | |
| 0 | 1 | 0 | 1 | 0 | $D_1$ |
| 0 | 1 | 1 | 0 | 1 | |
| 1 | 0 | 0 | 1 | 0 | $D_2$ |
| 1 | 0 | 1 | 0 | 1 | |
| 1 | 1 | 0 | 0 | 1 | $D_3$ |
| 1 | 1 | 1 | 1 | 1 | |

Sum (S) = $\Sigma m$ (1, 2, 4, 7)

Carry (C) = $\Sigma m$ (3, 5, 6, 7)

$Sum = D_0 = D_3$

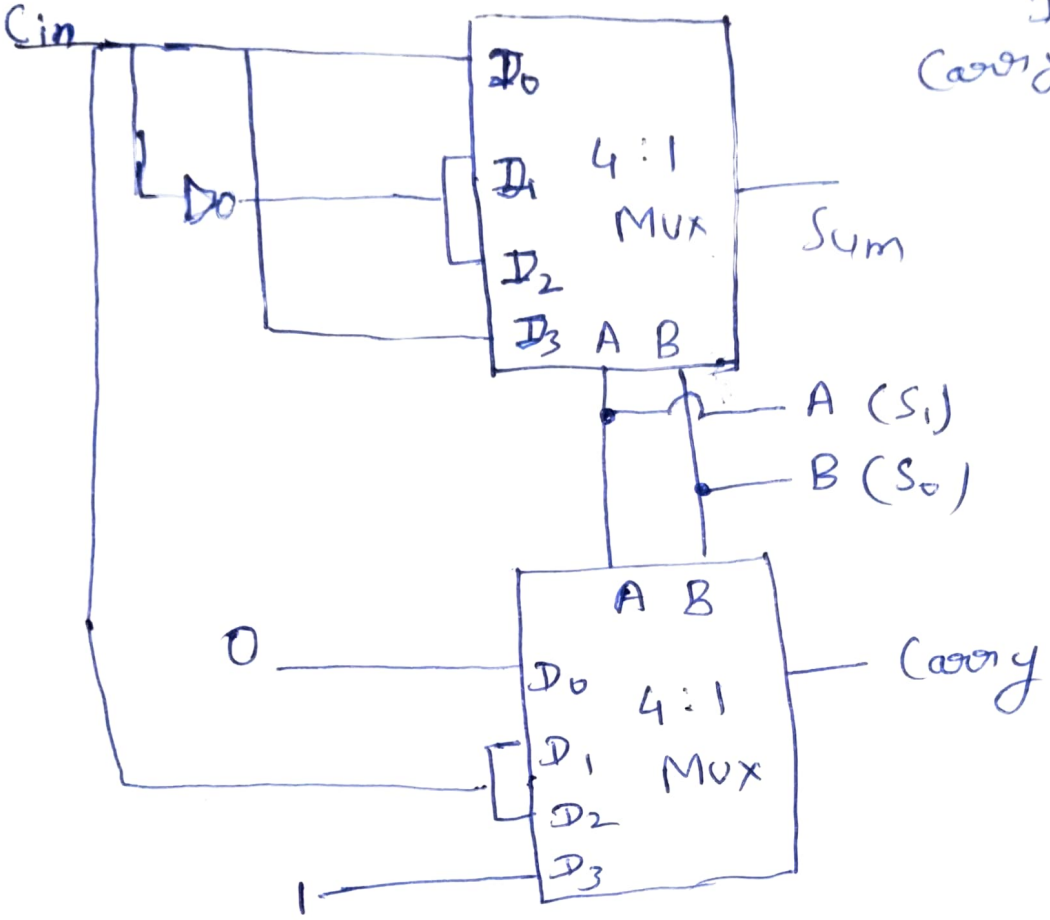
$= C_{in}$

$D_1 = D_2 = \overline{C_{in}}$

$Carry = D_0 = 0$

$D_3 = 1$

$D_1 = D_2 = C_{in}$



(2) Design Full adder using 8:1 MUX

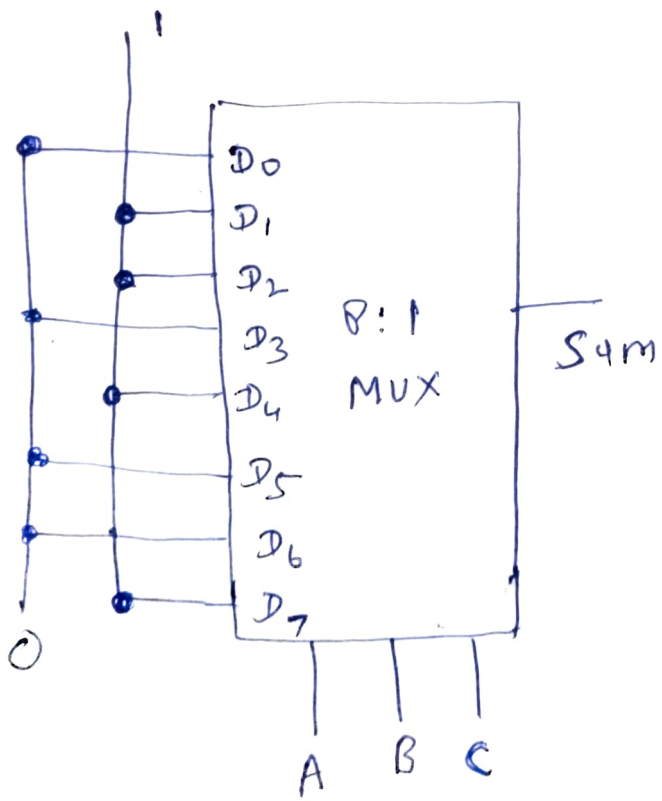
| Inputs | | | Outputs | |
|---|---|---|---|---|
| A | B | C | Sum | Carry |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

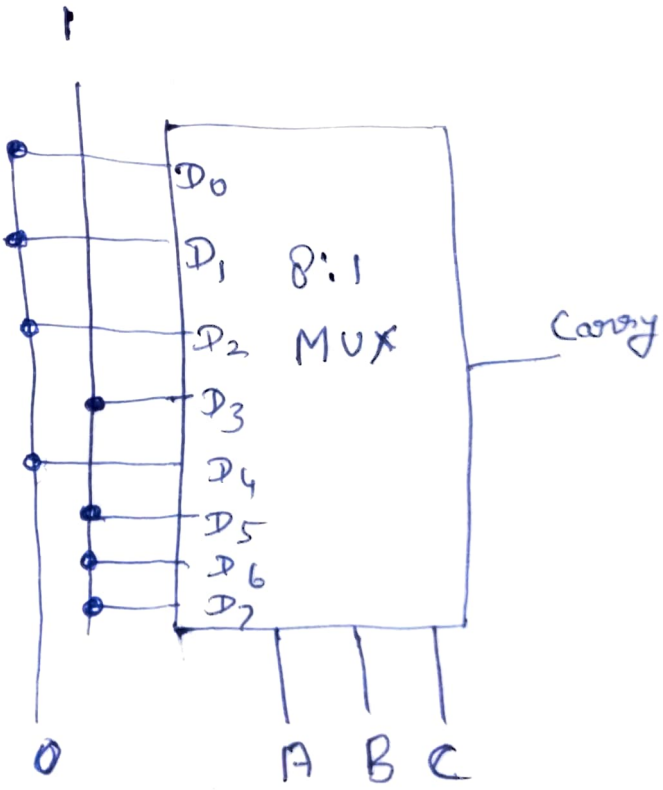
$S = \Sigma m(1,2,4,7)$

$C = \Sigma m(3,5,6,7)$

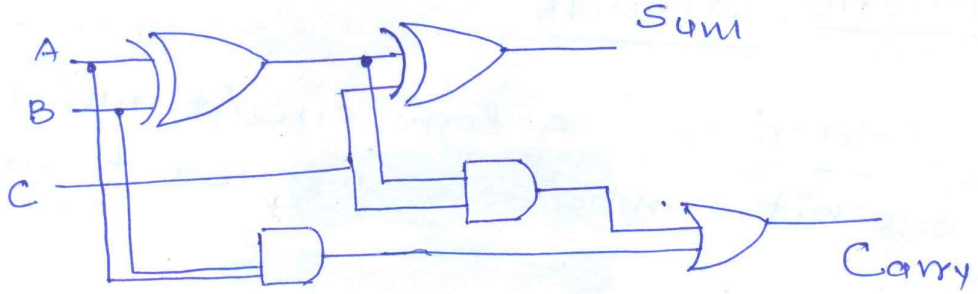
Sum $S = \Sigma m\ (1,2,4,7)$

Carry $C = \Sigma m\ (3,5,6,7)$



Logic circuit of Full Adder.




## Half Subtractor :-

A logic circuit which subtracts two one-bit numbers is referred to as a half subtra

Truth Table.

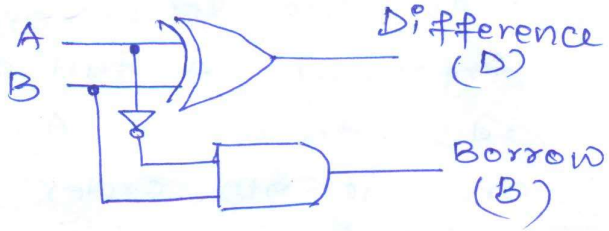
| Inputs | | outputs | |
|---|---|---|---|
| A | B | Diff. (D) | Borrow (B) |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

Logic expressions for Differen and borrow are derived fro the truth table.

$$D = A'B + AB'$$

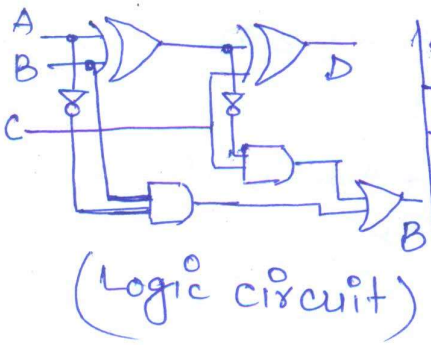
$$B = A'B$$

Logic circuits




## Full Subtractor :-

To perform multibit subtraction where in a borrow from the previous (higher position) bit position is required, full subtractor circuit is used.




(Logic circuit)

Truth Table:

| Inputs | | | outputs | |
|---|---|---|---|---|
| A | B | C | D | B |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Logic expressions

$$D = A'B'C + A'BC' + AB'C' + ABC$$
$$= A'(B \oplus C) + A(B \oplus C)'$$
$$= A \oplus B \oplus C$$

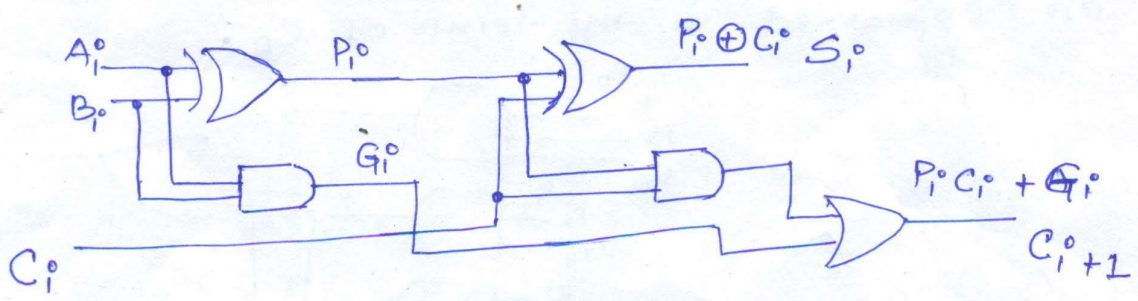
$$B = A'B'C + A'BC' + A'BC + ABC$$
$$= (A'B' + AB)C + A'B$$
$$= (A+B)'C + A'B$$

## Look ahead Carry generator :-

Addition of two or multibit binary numbers in parallel implies that all the bits of the augend and addened are available for computation at the same time. In any combinatio circuit, the signal must propagate through the gates before the correct output sum is available in the output terminals. The total propagation time is equal to the propagation delay of a typical gate, times the number of gate levels in the circuit. The logest propagation delay time in an adder is the time it takes the carry to propagate through the full adders.

For example, in a 4-bit parallel binary adder, input $A_3$ and $B_3$ are available as soon as input signals are applied to the adder, however input carry $C_3$ does not settle to its final value until $C_2$ is available from the previous stage. Similarly, $C_2$ has to wait for $C_1$ and so on down to $C_0$. Thus only after the carry propagates and ripples through all stages will the last output $S_3$ and Carry $C_4$ settle to their final value.

To reduce the carry propagation time delay, th most widely used technique employs the look ahead carry logic.




(Full adder with $P_i$ & $G_i$ Concept)

Let $P_i = A_i \oplus B_i$ & $G_i = A_i \cdot B_i$

The output Sum and carry can be expressed as

$$S_i = P_i \oplus C_i$$

& $$C_{i+1} = P_i C_i + G_i$$

$G_i$ is called Carry generate and it produces a carry of 1 when both $A_i$ and $B_i$ are 1, regardless of the input carry $C_i$.

$G_i$ indicales that the data into stage i generales a carry into stage $i+1$. $P_i$ is called Carry propagate, because it determines whether a carry into stage i will propagale into stage $i+1$.

Now write the boolean functions for the carry outputs of each stage and substitule the value of eac $C_i$ from the previous equations:

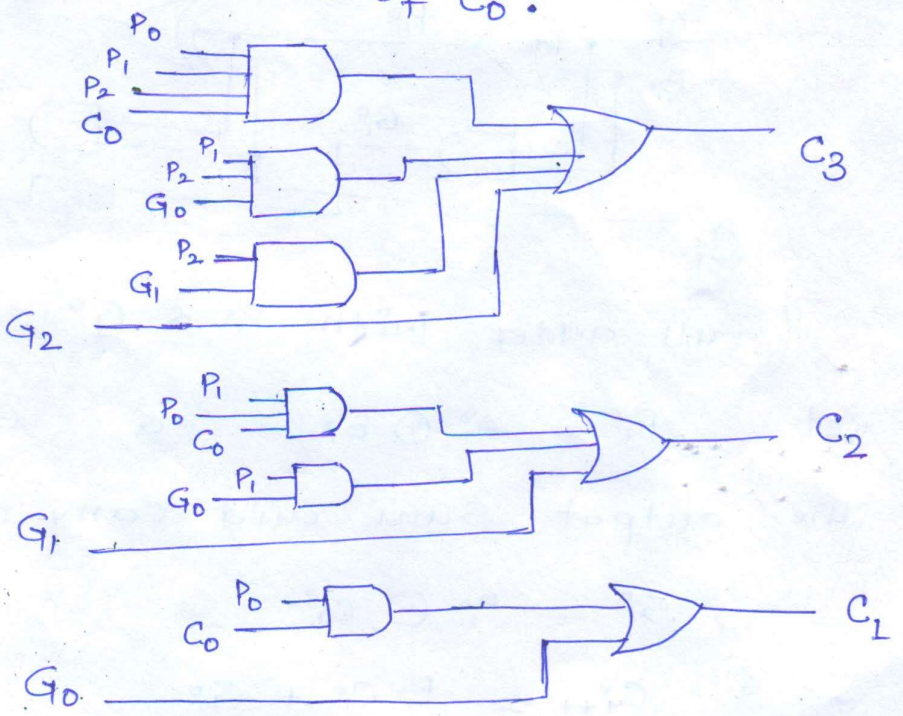
$C_0$ = input carry

$$C_1 = G_0 + P_0 C_0 \quad \text{——— (I)}$$

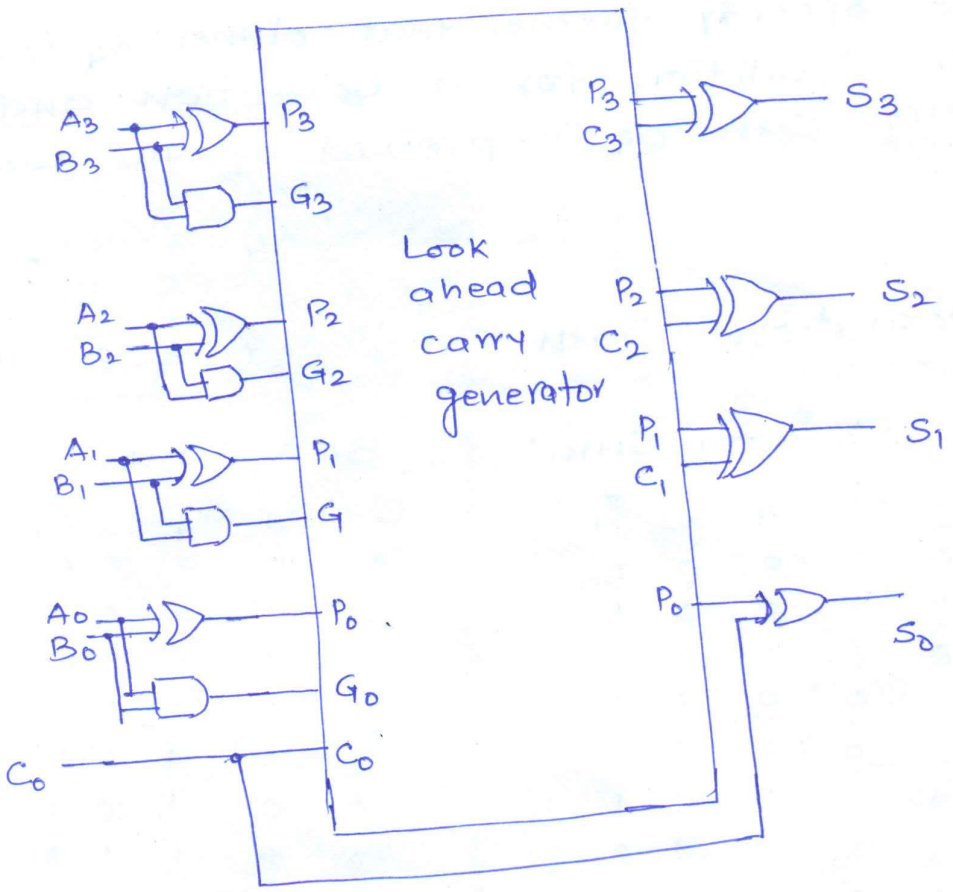
$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_0)$$
$$= G_1 + P_1 G_0 + P_1 P_0 C_0 \quad \text{——— (II)}$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 (G_1 + P_1 G_0 + P_1 P_0 C_0)$$
$$= G_2 + P_2 G_1 + P_1 P_2 G_0 + P_0 P_1 P_2 C_0 \quad \text{——— (III)}$$

The three boolean functions for $C_1$, $C_2$ and $C_3$ are implemented in the lookahead generator. This scheme will perform addition in less time because $C_1$, $C_2$ and $C_3$ are expressed in the terms of $C_0$.




(Diagram of lookahead carry generator)

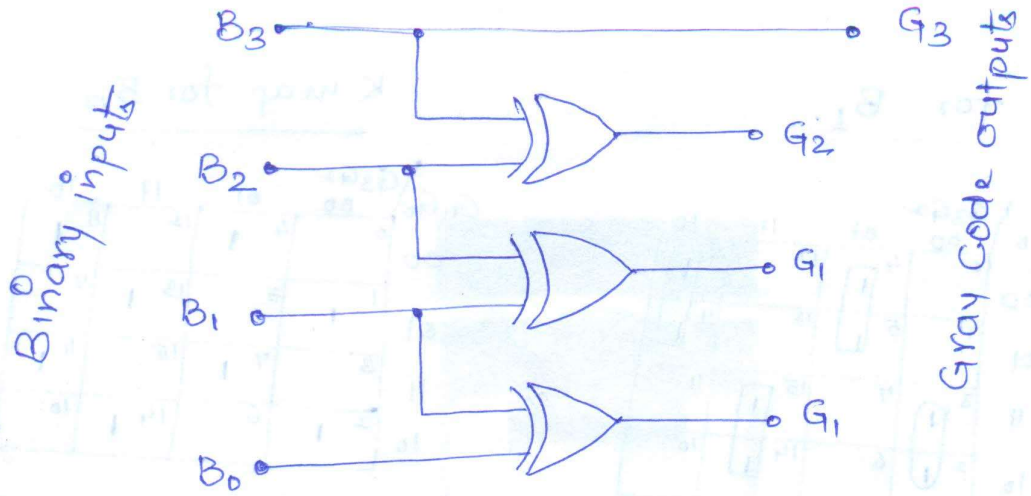



(Four-bit look ahead carry adder)

## BCD Adder:

Two BCD numbers can be added by parallel binary adder and sum is produc in binary. when the sum is less or equal to 9(1 it is in proper BCD form. But when the sum of two BCD numbers is greater than 9 (that is fro 10 to 19) a correction of 0110 should be added to that sum, to produce the proper BCD resu This will produce a carry to be added to the next higher position.

The logic circuit that detects the necessa correction can be derived from the truth table on It is obvious that a correction is needed when th binary sum has an output carry K = 1. The other si combinations from 1010 to 1111 that need a correctio have a 1 in position Zg. To distinguish them from bin




(Logic diagram of Binary to Gray code Converter)

## 2. Gray to Binary Code Converter :—

Truth Table :

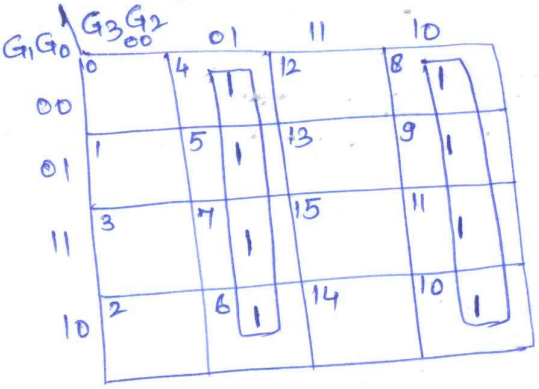
| Gray code input | | | | Binary output | | | |
|---|---|---|---|---|---|---|---|
| $G_3$ | $G_2$ | $G_1$ | $G_0$ | $B_3$ | $B_2$ | $B_1$ | $B_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 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 |

Simplify the output functions with K-map, we will get the simplified expressions.

K-map for $B_3$


$B_3 = G_3$

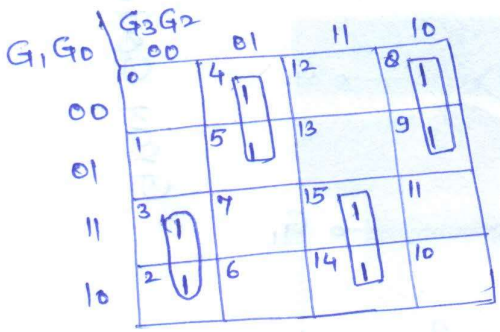
K-map for $B_2$




$B_2 = G_3'G_2 + G_3G_2'$

$= G_2 \oplus G_3$

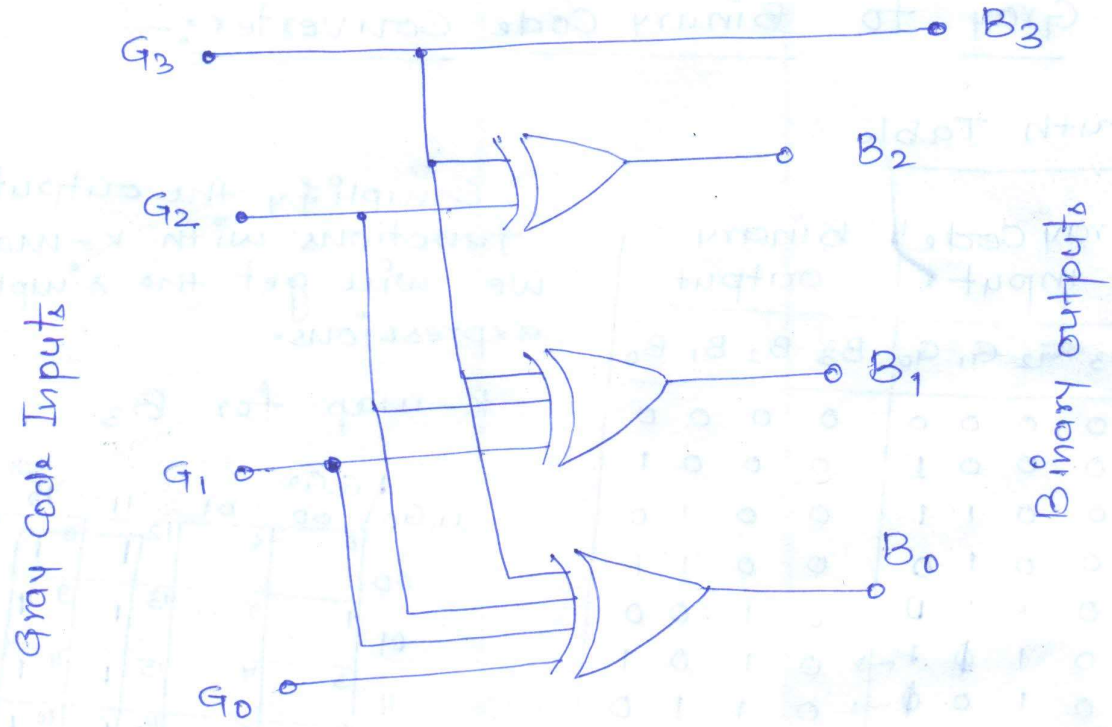
K-map for $B_1$.



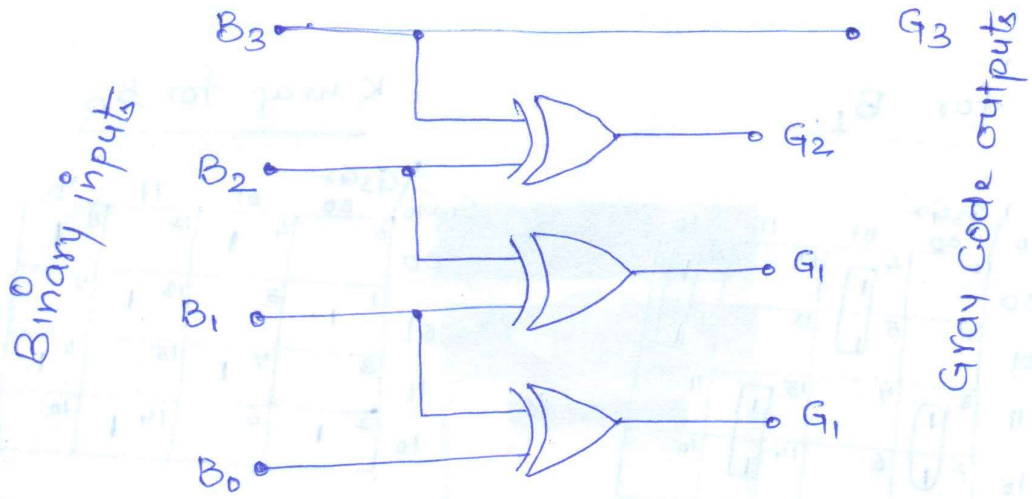

$B_1 = G_1 \oplus G_2 \oplus G_3$

K map for $B_0$


$B_0 = G_0 \oplus G_1 \oplus G_2 \oplus G_3$




(Logic diagram of Gray to binary Code Converter)




(Logic diagram of Binary to Gray code Converter)

## 2. Gray to Binary Code Converter :—

Truth Table :

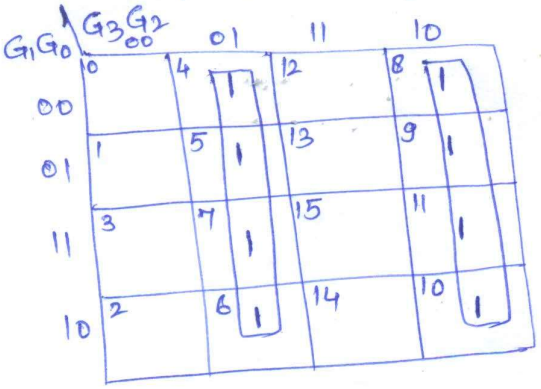
| Gray code input | | | | Binary output | | | |
|---|---|---|---|---|---|---|---|
| $G_3$ | $G_2$ | $G_1$ | $G_0$ | $B_3$ | $B_2$ | $B_1$ | $B_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 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 |

Simplify the output functions with K-map, we will get the simplified expressions.

K-map for $B_3$


$B_3 = G_3$

K-map for $B_2$




$B_2 = G_3'G_2 + G_3G_2'$

$= G_2 \oplus G_3$

K-map for $B_1$.

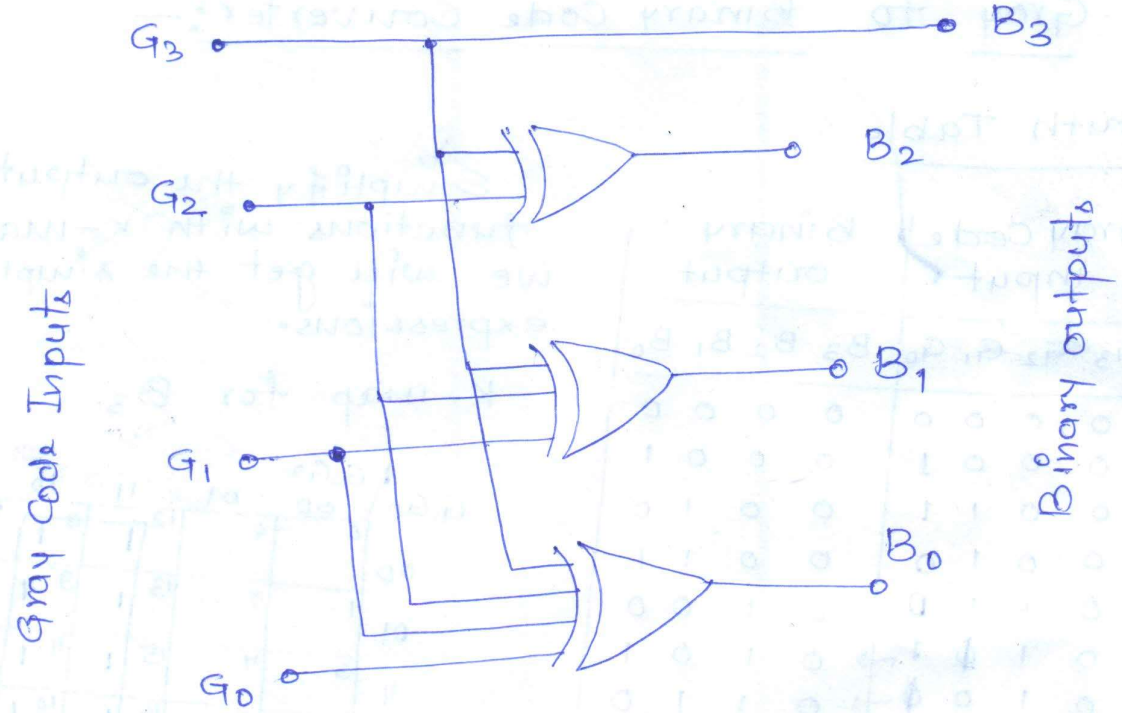
| $G_1G_0$ \ $G_3G_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 4 1 | 12 | 8 1 |
| 01 | 1 | 5 1 | 13 | 9 1 |
| 11 | 3 1 | 7 | 15 1 | 11 |
| 10 | 2 1 | 6 | 14 1 | 10 |

$B_1 = G_1 \oplus G_2 \oplus G_3$

K map for $B_0$

| $G_1G_0$ \ $G_3G_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 4 1 | 12 | 8 1 |
| 01 | 1 1 | 5 | 13 1 | 9 |
| 11 | 3 | 7 1 | 15 | 11 1 |
| 10 | 2 1 | 6 | 14 1 | 10 |

$B_0 = G_0 \oplus G_1 \oplus G_2 \oplus G_3$



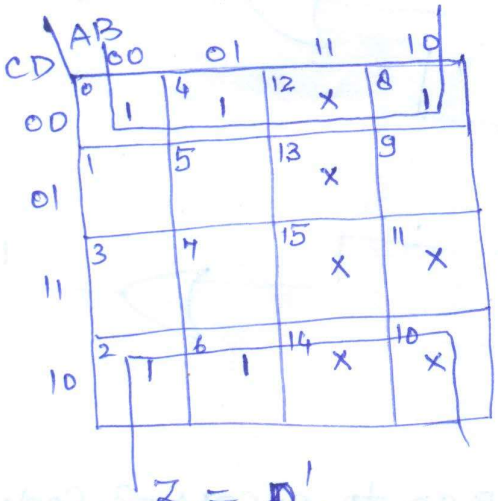

Logic diagram of Gray to binary Code Converter

## 3. BCD to excess-3 Code Converter:—

Code Converter is a circuit that makes the two systems Compa even though each uses a different binary code. BCD and excess-3 code uses four bits to represent a decimal digit, there must be four input variab and four output variables. Equivalent truth table is

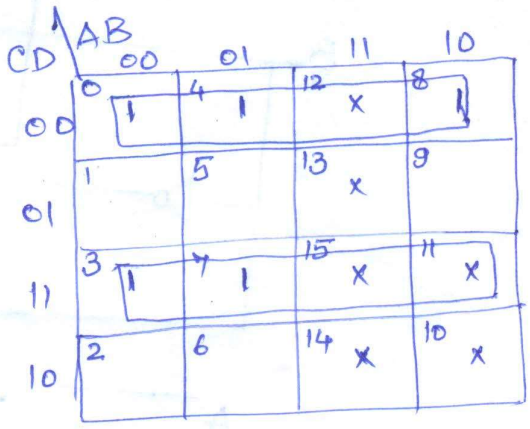
| Input BCD | | | | output Excess-3 code | | | |
|---|---|---|---|---|---|---|---|
| A | B | C | D | W | X | Y | Z |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 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 |

K map for Z



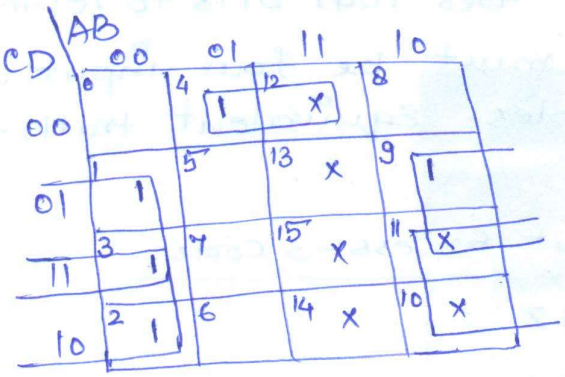

$Z = D'$

K map for Y



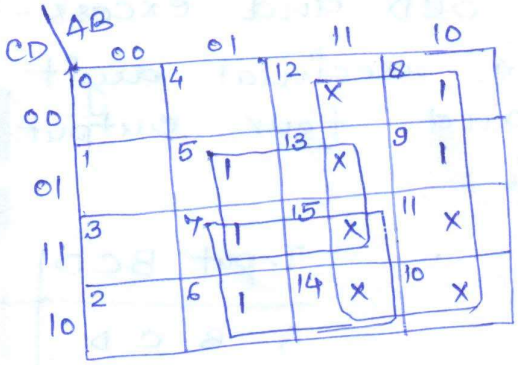

$Y = C'D' + CD$

## K – map for x




$X = BC'D' + B'D + B'C$

## K-map for W



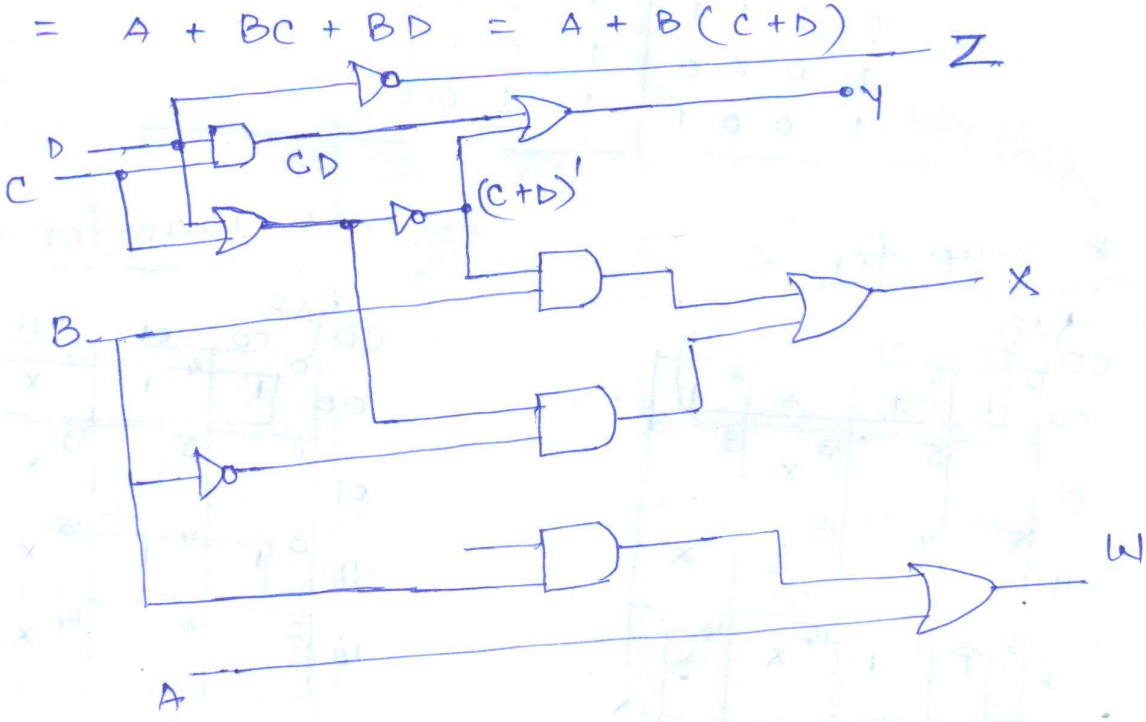

$W = A + BC + BD$

$Z = D'$

$Y = CD + C'D' = CD + (C+D)'$

$X = B'C + B'D + BC'D' = B'(C+D) + BC'D' = B'(C+D) + B(C+D$

$W = A + BC + BD = A + B(C+D)$



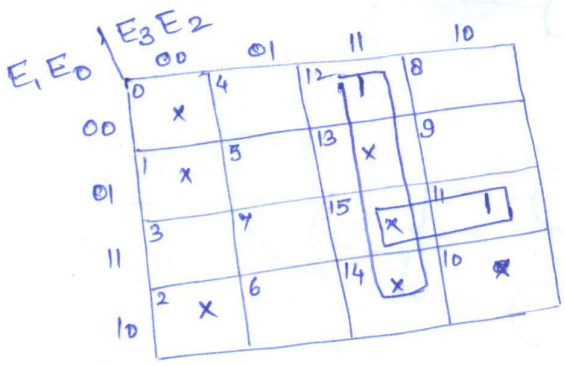

(Logic diagram of BCD to excess-3 Code Convert

## 4. Excess-3 to BCD Code Converter

### Truth Table

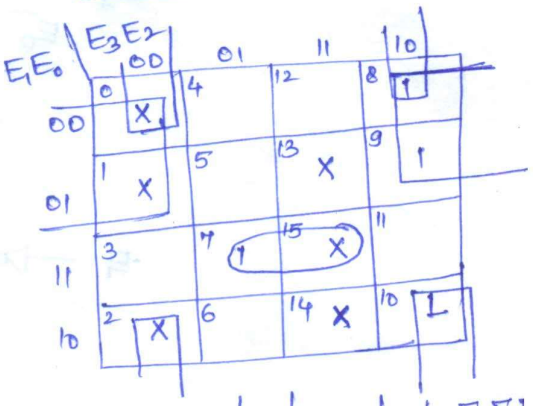
| Input XS-3 Code | | | | output BCD code | | | |
|---|---|---|---|---|---|---|---|
| $E_3$ | $E_2$ | $E_1$ | $E_0$ | A | B | C | D |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 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 |

### K-map for A




$A = E_3E_2 + E_3E_1E_0$

### K-map for B




$B = E_2'E_1' + E_2'E_0' + E_2E_1$

## 4 - bit magnitude Comparator :

Let A and B are two four bit numbers. Using algorithemic procedure, we can derive the boolean expressions showing the relation of magnitude between them.

Let $A = A_3 A_2 A_1 A_0$

& $B = B_3 B_2 B_1 B_0$

Two numbers (A & B) are equal in magnitude if all pairs of significant bits are equal that is $A_3 = B_3$ and $A_2 = B_2$ and $A_1 = B_1$ and $A_0 = B_0$, which can be logically expressed by the function.

$$x_i = A_i B_i + A_i' B_i' \quad \text{where } i = 0, 1, 2, 3$$

For equality condition to exist, all $x_i$ variables must be equal to 1, which is expressed by an AND operation of all variables

For $A = B = x_3 x_2 x_1 x_0$

$$= (A_3 \odot B_3) \cdot (A_2 \odot B_2) \cdot (A_1 \odot B_1) \cdot (A_0 \odot B_0)$$

The binary variable (A=B) is equal to 1 only if all pairs of digits of the two numbers are equal.

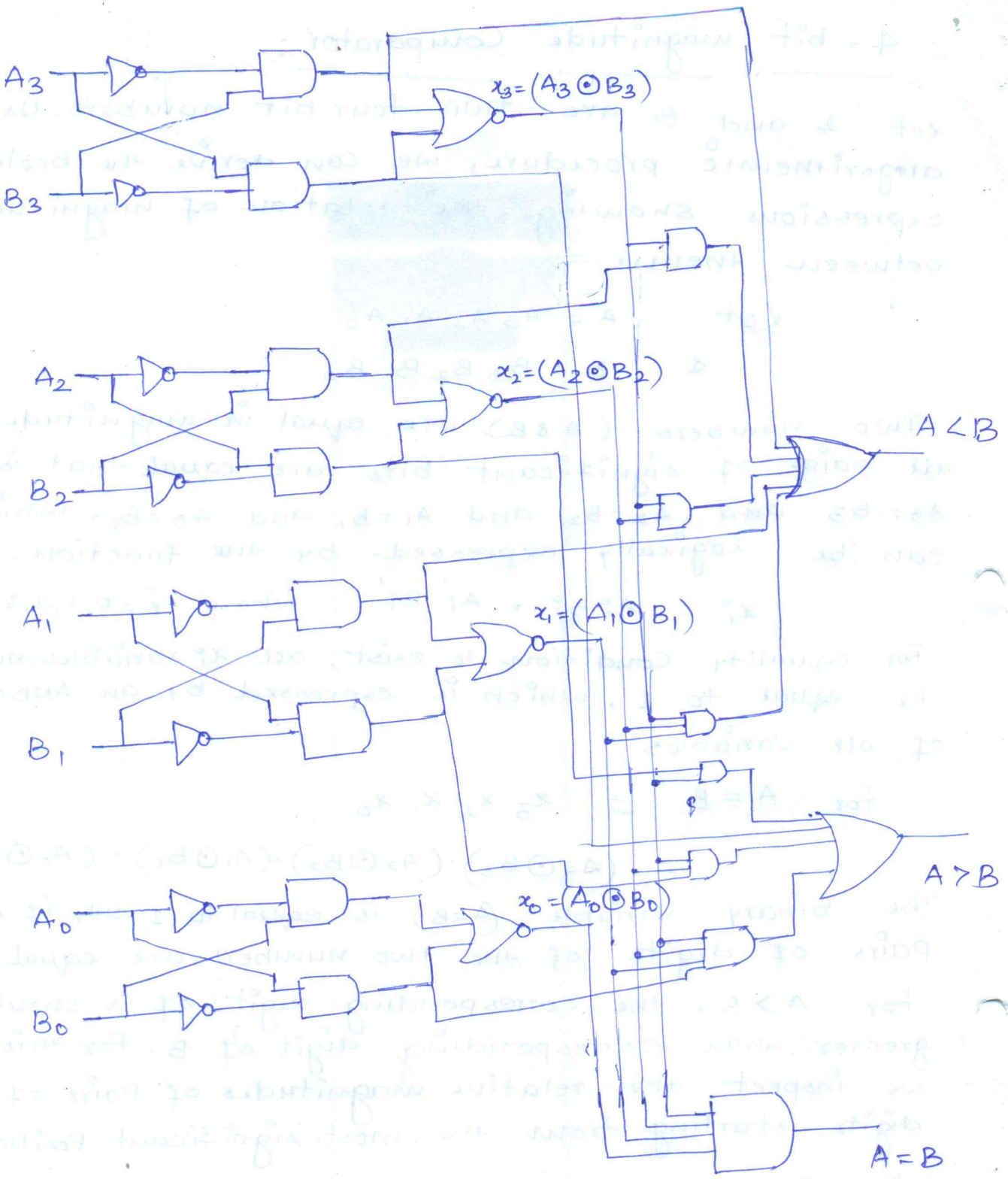
For A > B, the corresponding digit of A should be greater than corresponding digit of B. For this purpose we inspect the relative magnitudes of pairs of significant digits starting from the most significant Position.

For A > B, the boolean expression will be derived as

$$= A_3 B_3' + (A_3 \odot B_3) A_2 B_2' + (A_3 \odot B_3)(A_2 \odot B_2) A_1 B_1' + (A_3 \odot B_3) \cdot (A_2 \odot B_2) \cdot (A_1 \odot B_1) A_0 B_0'$$

For A < B, the corresponding digit of A should be smaller than corresponding digit of B, and boolean expression may be derived as:

$$A < B = A_3' B_3 + (A_3 \odot B_3) A_2' B_2 + (A_3 \odot B_3)(A_2 \odot B_2) A_1' B_1 + (A_3 \odot B_3)(A_2 \odot B_2) \cdot (A_1 \odot B_1) A_0' B_0$$




(Logic diagram of 4-bit magnitude comparator