CS 3351 - DP & CO

UNIT - I

COMBINATIONAL LOGIC

Combinational circuits - Karnaugh Map - Analysis and Design Procedures - Binary Adder - Subtractor - Decimal Adder - Magnitude comparator - Decoder - Encoder - Multiplexers - Demultiplexers.

2.1 Combinational Logic

* Logic circuits for digital systems may be
  i) combinational or
  ii) sequential..

Combinational circuit:

* consists of logic gates whose outputs at any time are determined from the present combination of inputs.
* Performs an operation that can be specified logically by a set of Boolean functions

Sequential circuit:

* employ storage elements in addition to logic gates
* Their outputs are a function of the inputs and the state of the storage elements.
* The outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs.
* The circuit behavior must be specified by a time sequence of inputs and internal states.

Combinational circuit

* A combinational circuit consists of input variables, logic gates and output variables.
  - The logic gates accept signals from the inputs and generate signals to the outputs.
    → This process transforms binary information from the given input data to a required output data.

Block diagram of combinational circuit

* The n input binary variables come from an external source, the m output variables go to an external destination
* i/p & o/p → binary signal represents
  - logic 1
  - logic 0
* For n input variables, there are 2^n possible binary input combinations.
  - For each possible input combination, there is one possible output value.
* A combinational circuit can be specified with a truth table that lists the output values for each combination of input variables.
* A combinational circuit can be described by m Boolean functions, one for each output variable.
  - Each output function is expressed in terms of the n input variables.

1.1 SIMPLIFICATION OF BOOLEAN FUNCTIONS USING

1.2 KARNAUGH MAP

* The simplification of the functions using Boolean laws and theorems becomes complex with increase in the number of variables and terms.

K-map:

- Proposed by Veitch and slightly improved by Karnaugh.
- The map method provides a simple straightforward procedure for minimizing Boolean functions.
* Map method is also known as the Karnaugh map or K-map or Veitch diagram.
  - regarded as a pictorial form of a truth table
* The map is a diagram made up of squares, with each square representing one minterm of the function.
  - presents a visual diagram of all possible ways a function may be expressed in standard form.
* for n variables on a K-map, there are 2^n number of squares
* Karnaugh maps can be used for expressions with two, three, four and five variables.

[1-var: x | x' | x]

① Two-Variable Map

m_0	m_1
m_2	m_3

x \ y	0	1
0	x'y'	x'y
1	xy'	xy

② Three-Variable Map

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

x \ yz	00	01	11	10
0	x'y'z'	x'y'z	x'yz	x'yz'
1	xy'z'	xy'z	xyz	xyz'

③ Four-Variable Map

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

wx \ yz	00	01	11	10
00	w'x'y'z'	w'x'y'z	w'x'yz	w'x'yz'
01	w'xy'z'	w'xy'z	w'xyz	w'xyz'
11	wxy'z'	wxy'z	wxyz	wxyz'
10	wx'y'z'	wx'y'z	wx'yz	wx'yz'

(4) Five- Variable Map:

A = 0

BC \ DE	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

A = 1

BC \ DE	00	01	11	10
00	16	17	19	18
01	20	21	23	22
11	28	29	31	30
10	24	25	27	26

Procedure to Simplify Boolean Expressions:

1. Plot the K-map
2. Place 1's in those cells corresponding to the 1's in the sum of product expression. Place 0's in the other cells.
3. check the K-map for adjacent 1's
4. Encircle the 1's which are not adjacent to any other 1's. These are called isolated 1's
5. check for the 1's which are adjacent to only one other and encircle such pairs.
6. check for quads and octets of adjacent 1's even if it contains some 1's that have already been encircled.
   → Make sure that there are minimum number of groups.
7. Combine any pairs necessary to include any 1's that have not yet been grouped.
8. Form the simplified expression by summing product terms of all the groups.

① Two-Variable Map:

– 4 minterms – 2^2 → 2 variables

* Variable x - appears primed in row 0 and unprimed in row 1

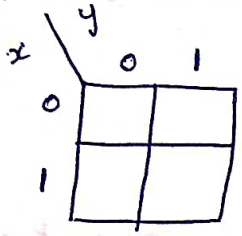
y - appears primed in column 0 and unprimed in column 1.

Representation in 2-variable K-map.

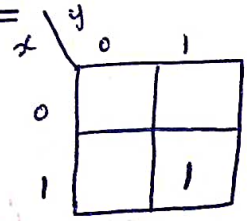
Ex:

1) $F = xy$

Step 1:

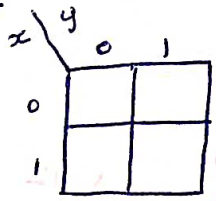


Step 2:

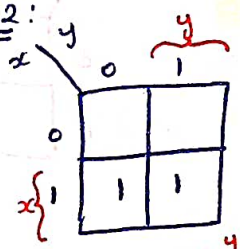


2) $F = x + y$

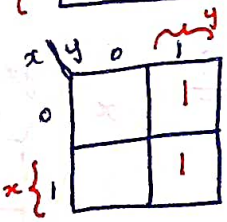
Step 1:



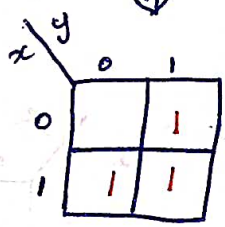
Step 2:



$= x$



$= y$



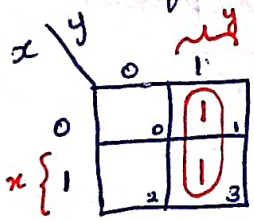
$= xy' + xy + x'y$

Simplified form is $x + y$

Example : 1 :

Find the simplified expression using K-map for the function $F = x'y + xy$

Soln:



01 → 1
11 → 3

Simplified form: $F = y$

② Three - Variable Map:

→ 3 - variables, $2^3 = 8$ minterms

↓

8 squares

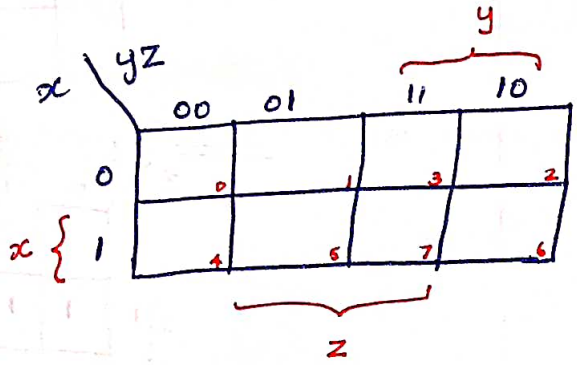
→ minterms are arranged in a sequence of Gray code
- only one bit changes in value from one adjacent column to the next.

Example 1: Simplify the Boolean function using map method.

$F(x, y, z) = \Sigma\,(2, 3, 4, 5)$

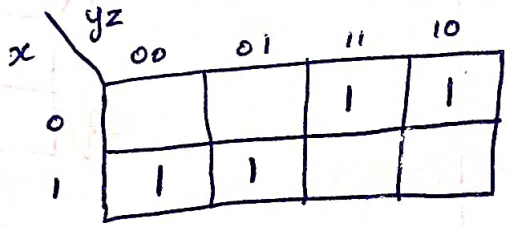
Soln:

Step 1:



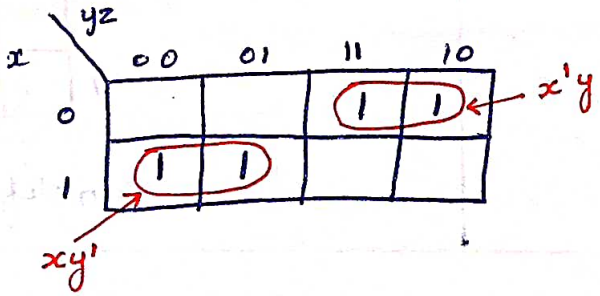
* Plot K-map

Step 2:



* Place 1's for the minterms

Step 3:



* Grouping adjacent 1's

Step 4:

* Logical sum of product terms
* Write the simplified expression

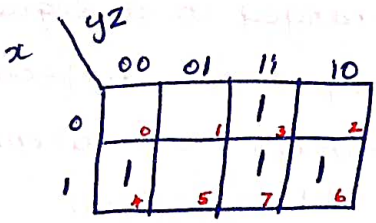
$$F = x'y + xy'$$

Example 2:

Simplify the Boolean function $F(x,y,z) = \Sigma(3,4,6,7)$

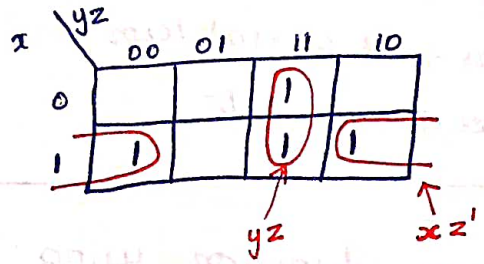
Soln:

Step 1:



* Plot & place 1's

Step 2:



* Grouping 1's

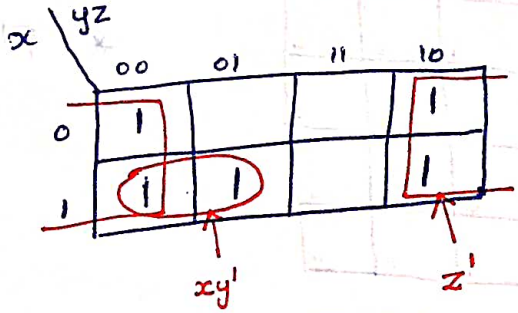
Step 3:

$$F = yz + xz'$$

Example 3:

Simplify the Boolean function $F(x,y,z) = \Sigma(0,2,4,5,6)$

Soln:



$$F = z' + xy'$$

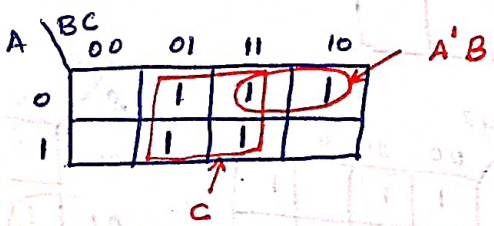
Example 4:

Given the Boolean function $F = A'C + A'B + AB'C + BC$

a) express it in sum of minterms

b) and find the minimal sum of products expression

Soln:



$$F = C + A'B$$

Ans:

a) Sum of minterms

$F(A,B,C) = \Sigma(1,2,3,5,$

b) $F = C + A'B$

③ Four - Variable Map:

* 4-variable → 2^4 = 16 squares = 16 minterms
* minterms are arranged in a sequence of Gray code.

→ Combination of adjacent squares - inspection of 4-var. map.

1 square → one minterm — 4 literals.
2 adjacent squares → 3 literals term
4 adjacent squares → 2 literals term
8 adjacent squares → 1 literal term
16 adjacent squares → equal to 1

Example 1:

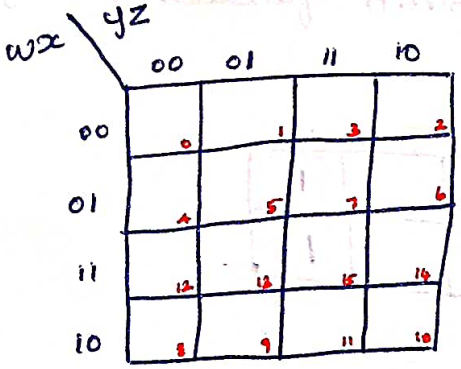
Simplify the Boolean function using k-map method.

$$F(w,x,y,z) = \sum (0,1,2,4,5,6,8,9,12,13,14)$$

Soln:

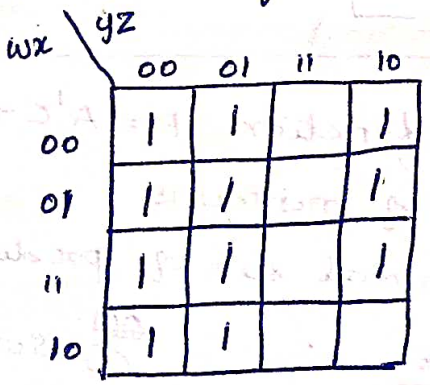
* 4-variable map.

Step 1:

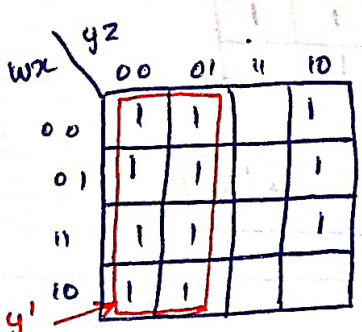


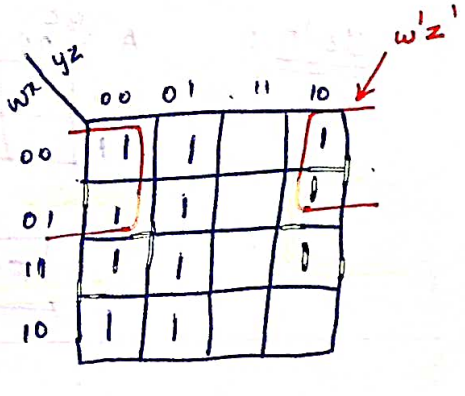
* Plot k-map.

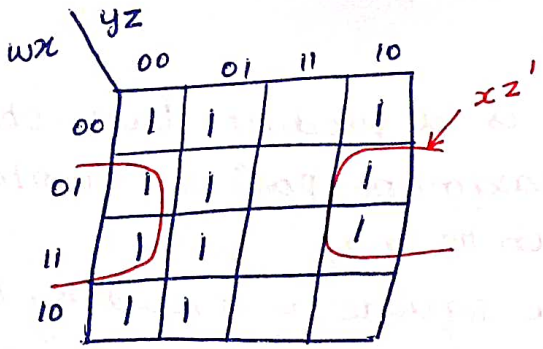
Step 2: Place 1's for the minterms



Step 3: Grouping







Step 4:

→ Sum of products

Simplified Expression is

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

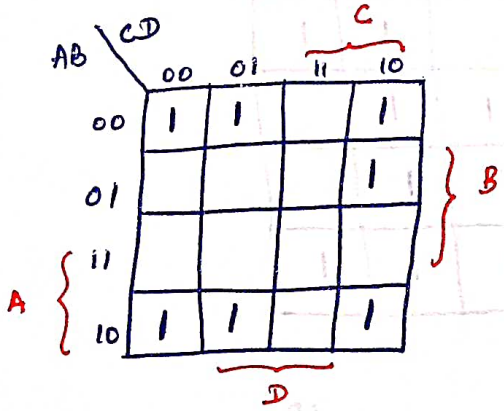
Example 2:

Simplify the Boolean function

$$F = A'B'C' + B'CD' + A'BCD' + AB'C'$$

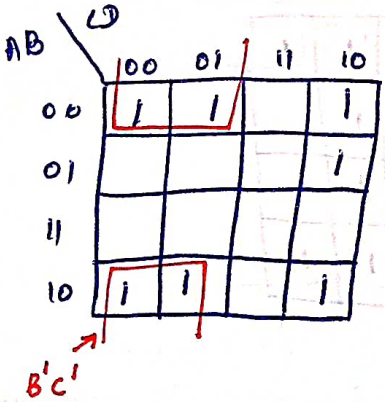
Soln:

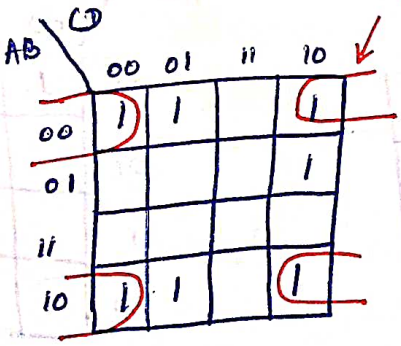
Step 1:

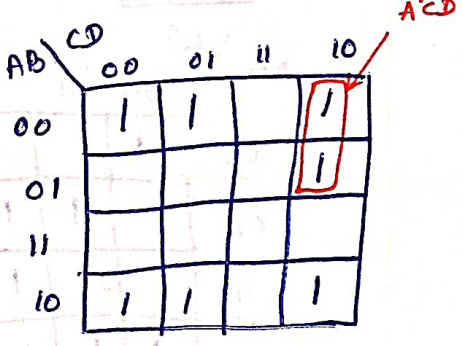


* Plot k-map
* Place 1's
* Grouping adjacent 1's
* Write expressions

Step 2: Grouping







Step 3:

Simplified function is

$$F = B'C' + B'D' + A'CD'$$

Prime Implicants:

* A prime implicant is a product term obtained by combining the maximum possible number of adjacent squares in the map.
- If a minterm in a square is covered by only one prime implicant, that prime implicant is said to be essential.

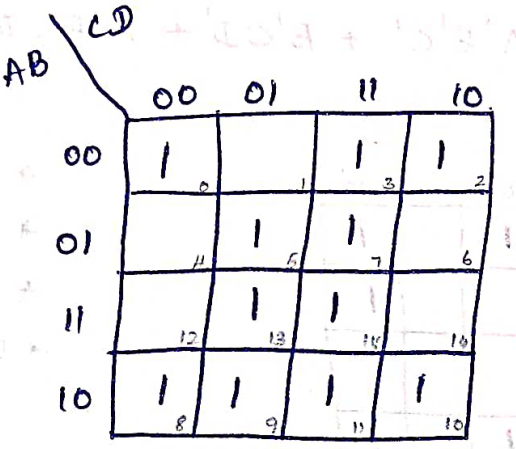
Example:

Simplify the four-variable Boolean function

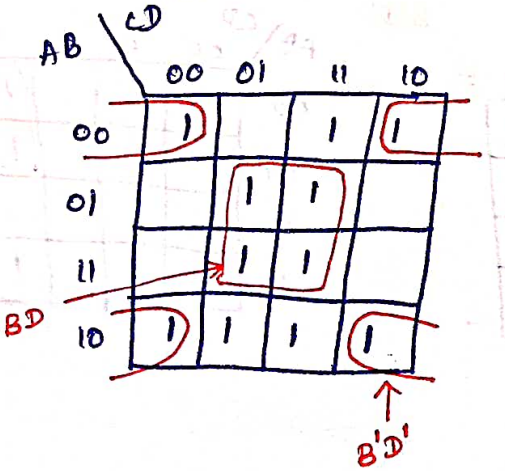
$$F(A,B,C,D) = \sum(0,2,3,5,7,8,9,10,11,13,15)$$

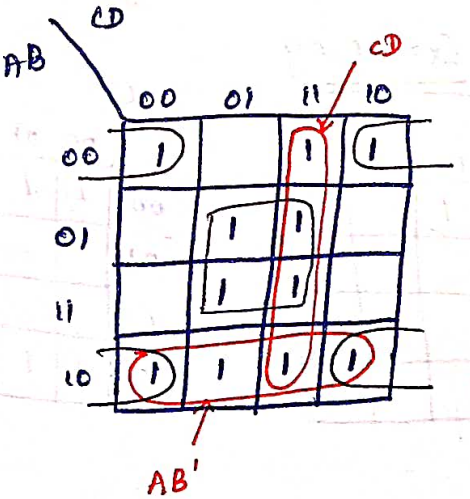
Soln:

Step 1:



Step 2:



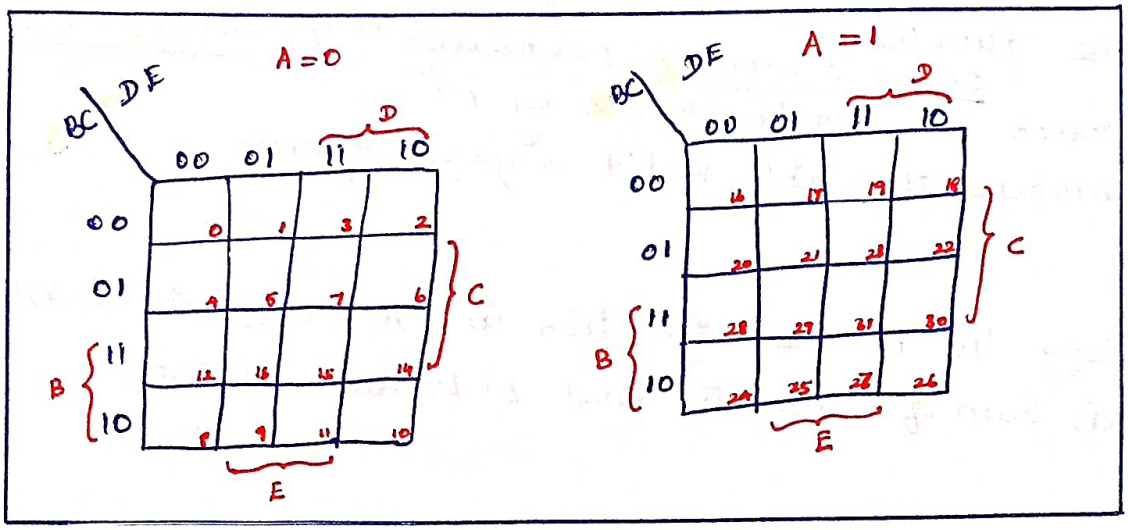


* Essential Prime implicants.

$$F = BD + B'D' + CD + AB'$$

Five - Variable Map:

* 5 - variables → $2^5 = 32$ squares need.
* It consists of 2 four - variable maps with variables A, B, C, D and E.

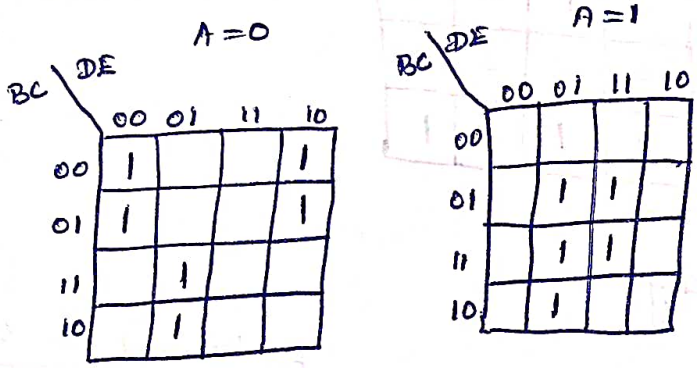


Example: Simplify the Boolean function:

$$F(A, B, C, D, E) = \sum (0, 2, 4, 6, 9, 13, 21, 23, 25, 29, 31)$$

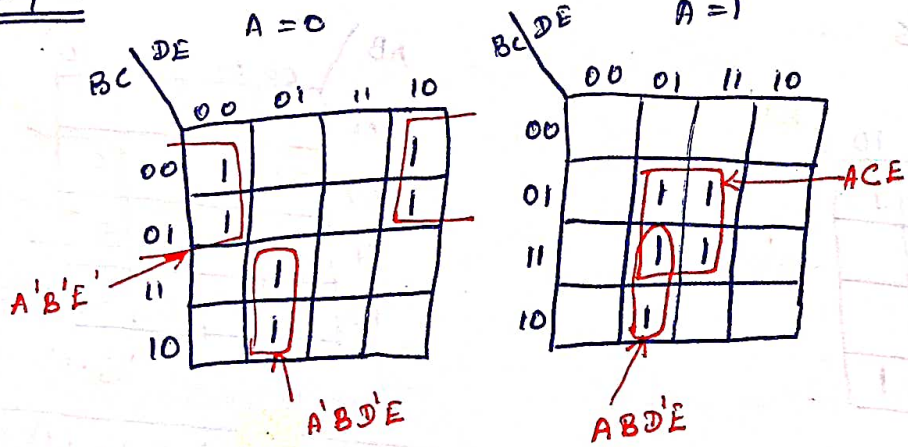
Soln:

Step 1:



* Plot K-map.

Step 2:



* Grouping all 1's
* simplified Boolean function.

Step 3:

$$F = A'B'E' + A'BD'E + ACE + ABD'E$$

$$= A'B'E' + ACE + BD'E(A' + A) \quad \ldots\ldots\ldots [x + x' = 1]$$

$$\boxed{F = A'B'E' + ACE + BD'E}$$

Product of Sums Simplification:

POS → complement of SOP [Using DeMorgan's theorem]

1's in the squares → minterms

* minterms not included in the function denote the complement of the function.

obtain a simplified expression of the complement of the functio

→ mark the empty squares by 0's

→ combine them into valid adjacent squares.

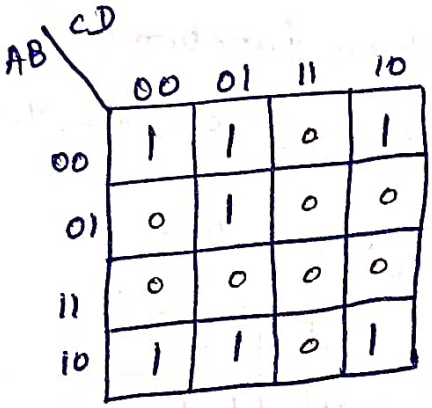
Example:

Simplify the Boolean function $F(A,B,C,D) = \Sigma(0,1,2,5,8,9,1$

in a) Sum of Products and b) Product of Sums

Soln:

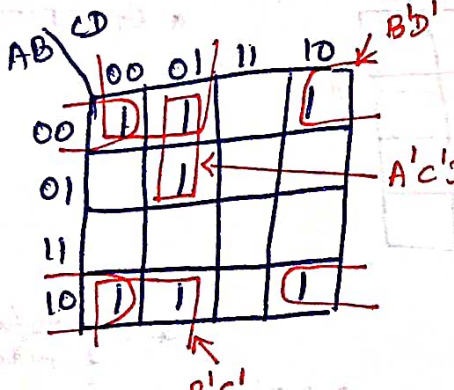
Step 1: * Plot K-map.



Step 2: * Grouping

a) Sum of Products

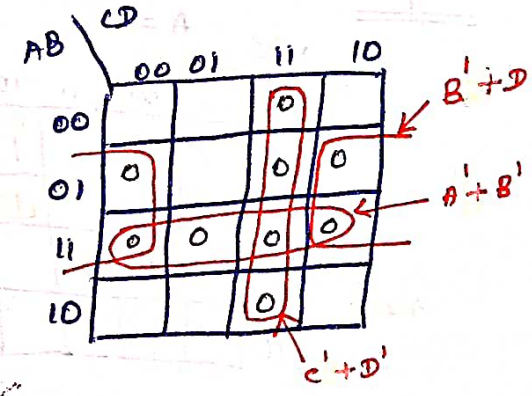
* Group all 1's



Ans: $F = B'D' + B'C' + A'C'D$

b) Product of Sums

* Group all 0's



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

Sum of Products

(grouping 1's)

$$F = B'D' + B'C' + A'C'D$$

[grouping 0's]

$$F' = AB + CD + BD'$$

Product of sums

⇓ DeMorgan's Theorem.

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

Don't Care Conditions:

* In some applications, the function is not specified for certain combinations of the variable.
* Functions that have unspecified outputs for some input combinations are called incompletely specified functions.
  - simply don't care - unspecified minterms → logical value is not specified
* Don't care conditions can be used on a map to provide further simplification of the Boolean expression.
  - X is used. → don't care whether the value of 0 or 1 is assigned to F.
* When simplifying the function,
  - choose don't care to include each don't care minterm with either the 1's or the 0's, depending on which combination gives the simplest expression.

Example:

Simplify the Boolean function $F(w,x,y,z) = \Sigma(1,3,7,11,15)$ which has the don't care conditions $d(w,x,y,z) = \Sigma(0,2,5)$

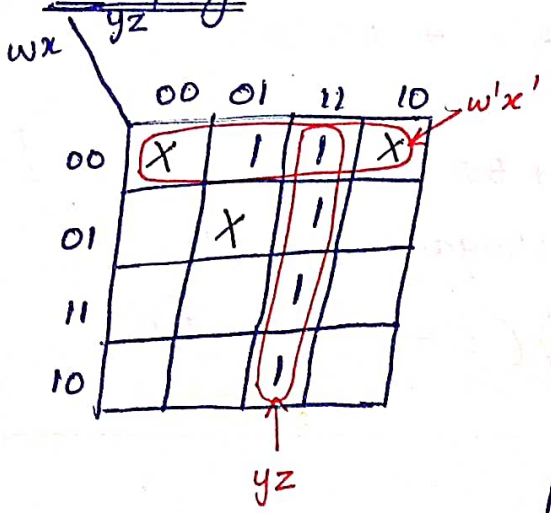
Soln:

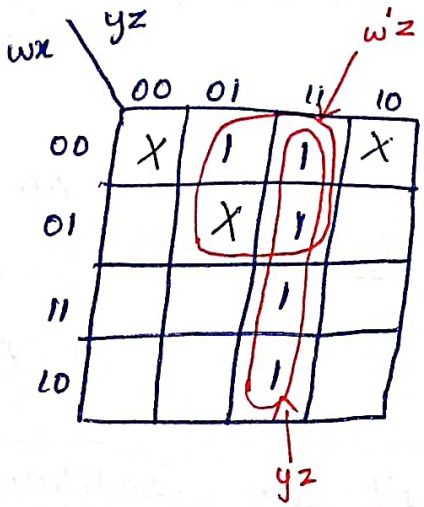
Step 1:

* Plot K-map

wx \ yz	00	01	11	10
00	X	1	1	X
01		X	1	
11			1	
10			1	

Step 2: Grouping





Step 3:

$F = yz + w'x'$

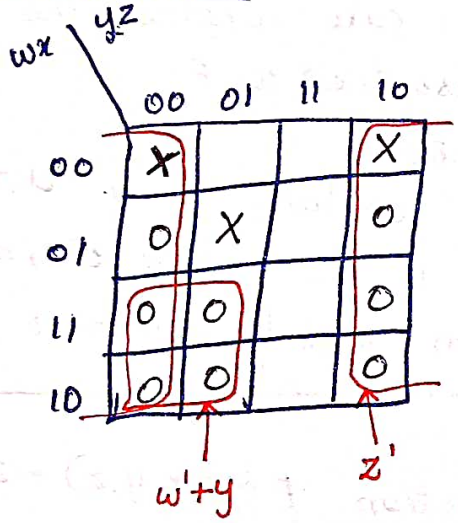
$F = yz + w'z$

Ans: $F = yz + w'x'$

(or)

$F = yz + w'z$

Product of Sum: (Above Problem)



$F = z'(w'+y)$

Problems:

1) Given the Boolean function $F = AB'C + A'B'C + A'BC + AB'C' + A'B'C'$
   a) Express it in sum of minterms
   b) Find the minimal sum of products expression.
2) Simplify the Boolean expression using K-map.
   i) $F = A'BC'D' + A'BC'D + ABC'D' + ABC'D + AB'C'D + A'B'CD'$
   ii) $Y = ABCD + AB'C'D' + AB'C + AB$
   iii) $Y(A,B,C,D) = \sum m(7,9,10,11,12,13,14,15)$

Problems:

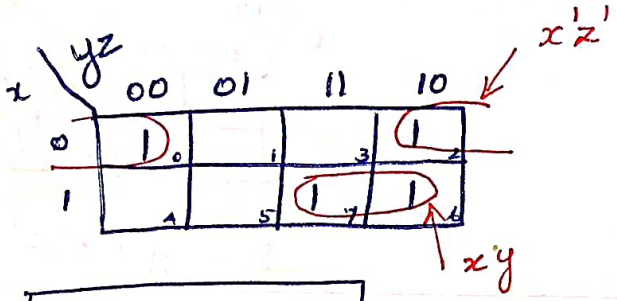
Karnaugh Map

Three-Variable Maps.

1. Simplify the following Boolean functions, using three-variable maps:

a) $F(x,y,z) = \Sigma(0,2,6,7)$

Soln:

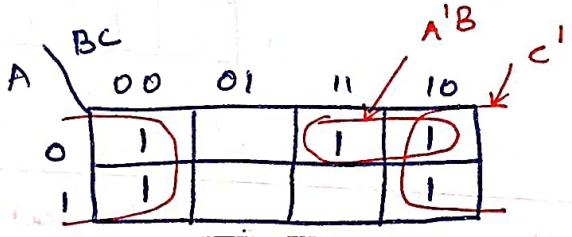


$$F = x'z' + xy$$

---

b) $F(A,B,C) = \Sigma(0,2,3,4,6)$

Soln:

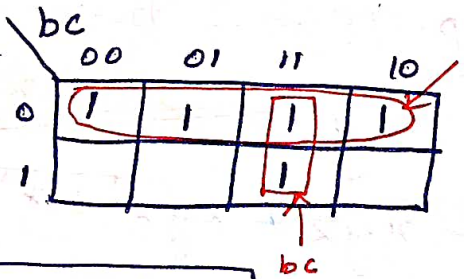


$$F = c' + A'B$$

---

c) $F(a,b,c) = \Sigma(0,1,2,3,7)$

Soln:

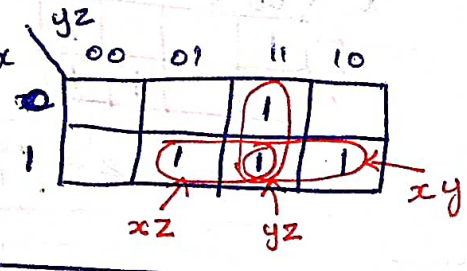


$$F = a' + bc$$

---

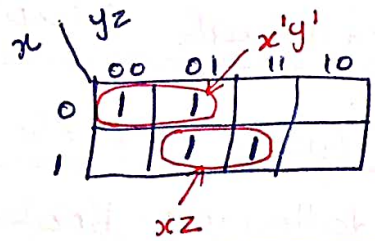
d) $F(x,y,z) = \Sigma s(3,5,6,7)$

Soln.



$$F = xz + yz + xy$$

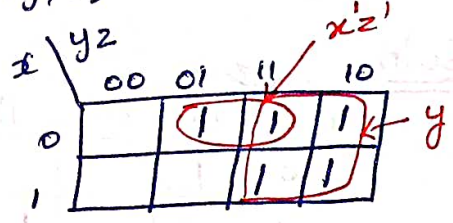
d) $F(x,y,z) = \Sigma(0,1,5,7)$



$F = x'y' + xz$

---

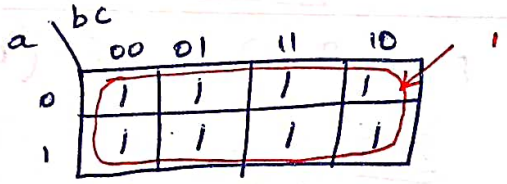
e) $F(x,y,z) = \Sigma(1,2,3,6,7)$



$F = y + x'z'$

---

f) $F(a,b,c) = \Sigma(0,1,2,3,4,5,6,7)$



$F = 1$

---

2) Simplify the following Boolean expressions, using three-variable maps:

a) $xy + x'y'z' + x'yz'$

step1: SOP: (Sum of Products)

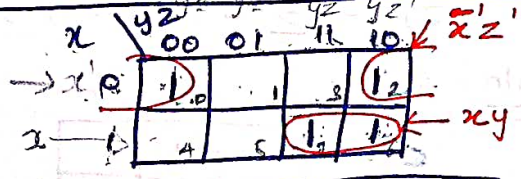
$= xy(z+z') + x'y'z' + x'yz'$

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

(111, 110, 000, 010)

$F(x,y,z) = \Sigma(0,2,6,7)$

step2:



00
01
10
11

$F = x'z' + xy$

b) $x'y' + yz + x'yz'$

Step 1: SoP.

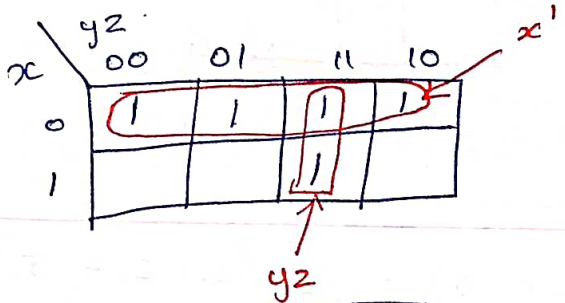
$= x'y'(z+z') + yz(x+x') + x'yz'$

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

$= \underset{001}{x'y'z} + \underset{000}{x'y'z'} + \underset{111}{xyz} + \underset{011}{x'yz} + \underset{010}{x'yz'}$

$F(x,y,z) = \Sigma(0, 1, 2, 3, 7)$ sum of minterms

Step 2: Boolean Expression using K-map.



$F = x' + yz$

---

c) $A'B + Bc' + B'c'$

Step 1: SoP.

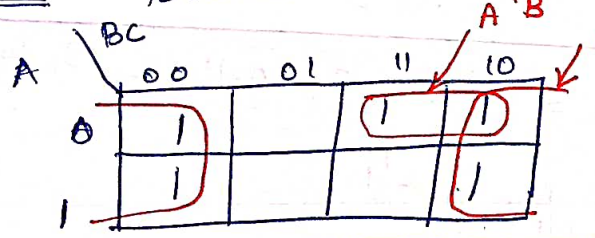
$= A'B(c+c') + Bc'(A+A') + B'c'(A+A')$

$= A'Bc + A'Bc' + ABc' + A'Bc' + AB'c' + A'B'c'$

$= \underset{011}{A'Bc} + \underset{010}{A'Bc'} + \underset{110}{ABc'} + \underset{010}{A'Bc'} + \underset{100}{AB'c'} + \underset{000}{A'B'c'}$

$F(A,B,C) = \Sigma(0, 2, 3, 4, 6)$

Step 2: Boolean Expression using 3-var. K-map.



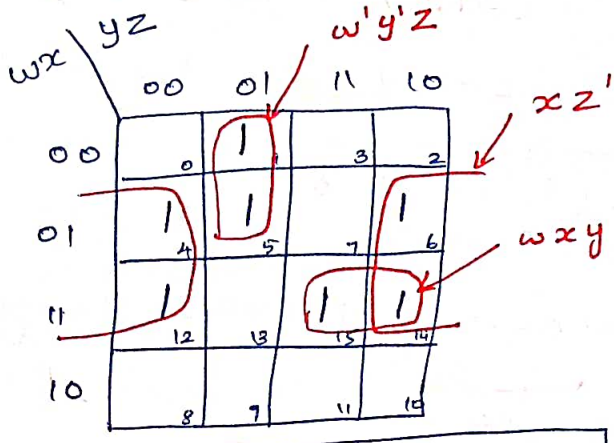
$F = c' + A'B$

---

3) Simplify the following Boolean functions using four-variable maps:

a) $F(w,x,y,z) = \sum(1,4,5,6,12,14,15)$

Soln:

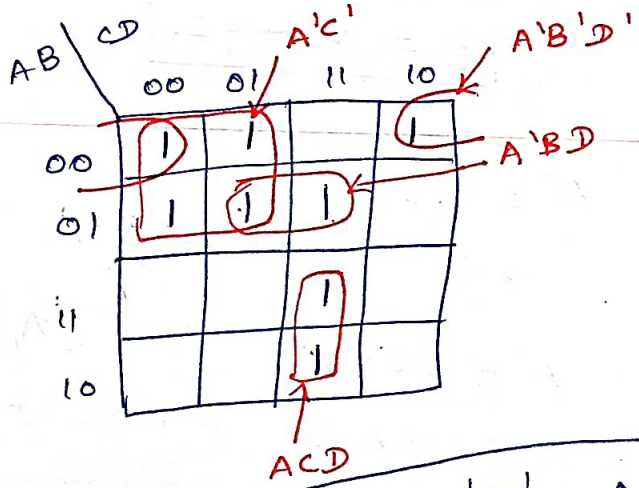


$$F = xz' + w'y'z + wxy$$

---

b) $F(A,B,C,D) = \sum(0,1,2,4,5,7,11,15)$

Soln:



$$F = A'C' + ACD + A'B'D' + A'BD$$

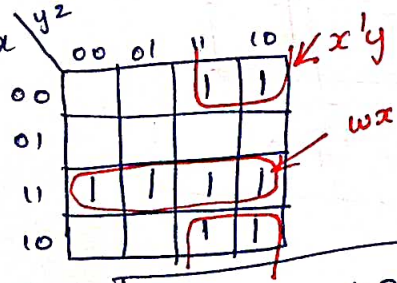
$$F = A'C' + ACD + A'(B'D' + BD)$$

$$F = A'C' + ACD + A'(B \oplus D)'$$

---

c) $F(w,x,y,z) = \sum(2,3,10,11,12,13,14,15)$

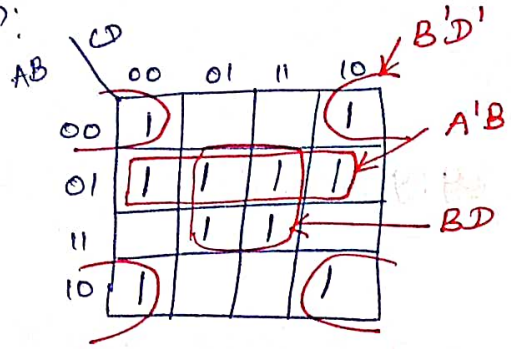
Soln:



$$F = wx + x'y$$

d) $F(A, B, C, D) = \Sigma(0, 2, 4, 5, 6, 7, 8, 10, 13, 15)$

Soln:



$F = B'D' + A'B + BD$

---

4) Simplify the following Boolean expressions using four-variable maps:

a) $A'B'C'D' + AC'D' + B'CD' + A'BCD + BC'D$

Soln:

Step1: SOP.

$= A'B'C'D' + AC'D'(B+B') + B'CD'(A+A') + A'BCD + BC'D(A+A')$

$= A'B'C'D' + ABC'D' + AB'C'D' + AB'CD' + A'B'CD' + A'BCD + ABC'D + A'BC'D + A'BC'D$

$A'B'C'D'$: 0 0 0 0 → 0

$ABC'D'$: 1 1 0 0 → 12

$AB'C'D'$: 1 0 0 0 → 8

$AB'CD'$: 1 0 1 0 → 10

$A'B'CD'$: 0 0 1 0 → 2

$A'BCD$: 0 1 1 1 → 7

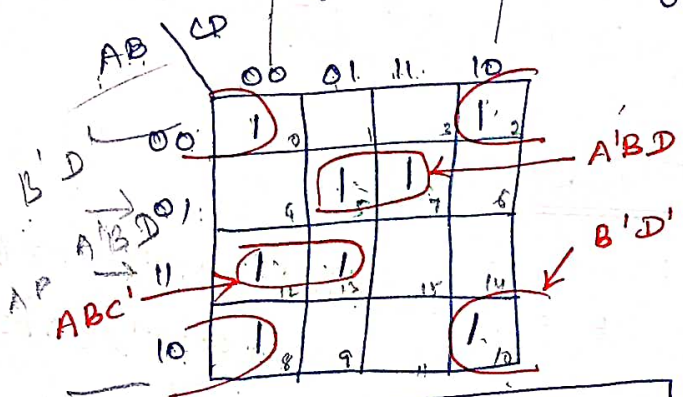
$ABC'D$: 1 1 0 1 → 13

$A'BC'D$: 0 1 0 1 → 5

Sum of minterms

$F(A, B, C, D) = \Sigma(0, 2, 5, 7, 8, 10, 12, 13)$

Step2: Boolean functions using K-map.



$F = B'D' + A'BD + ABC'$

b) $x'z + w'xy' + w(x'y + xy')$

Soln:

Step 1: SOP

$F(w,x,y,z) = x'z(w+w')(y+y') + w'xy'(z+z') + wx'y(z+z') + wxy'(z+z')$

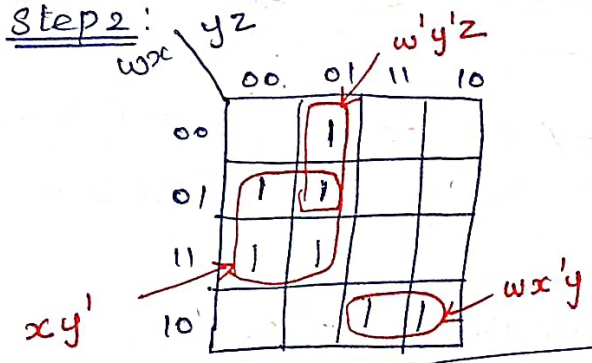
$= (wx'z + w'x'z)(y+y') + w'xy'z + w'xy'z' + wx'yz + wx'yz' + wxy'z + wxy'z'$

$= wx'yz + w'x'y'z + w'xy'z + w'xy'z' + wx'yz + wx'yz' + wxy'z' + wxy'z'$

1011 (11) 0001 (1) 0101 (5) 1011 (11) 1010 (10) 1101 (13) 1100 (12)

$F(w,x,y,z) = \Sigma(1, 4, 5, 10, 11, 12, 13)$ sum of minterms

Step 2:



$F = xy' + w'y'z + wx'y$

---

c) $w'z + xz + x'y + wx'z$

Soln:

Step 1: SOP

$F(w,x,y,z)$

$F = w'z(x+x')(y+y') + xz(w+w')(y+y') + x'y(w+w')(z+z') + wxz(y+y')$

$= (w'xz + w'x'z)(y+y') + (wxz + w'xz)(y+y') + (wx'y + w'x'y)(z+z') + wx'yz + wx'y'z$

b) $x'z + w'xy' + w(x'y + xy')$

Soln:

Step 1: SOP

$F(w,x,y,z) = x'z(w+w')(y+y') + w'xy'(z+z') + wx'y(z+z') + wxy'(z+z')$

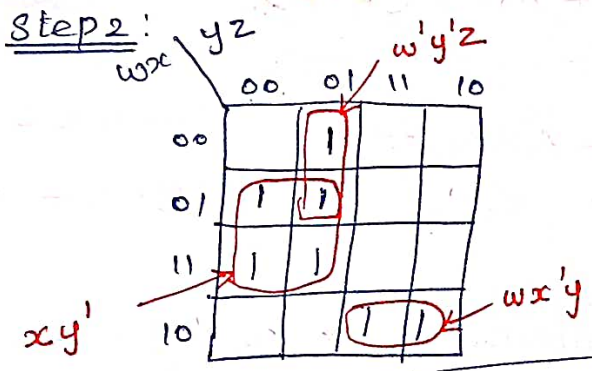
$= (wx'z + w'x'z)(y+y') + w'xy'z + w'xy'z' + wx'yz + wx'yz' + wxy'z + wxy'z$

$= wx'yz + w'x'y'z + w'xy'z + w'xy'z' + wx'yz + wx'yz' + wxy'z' + wxy'z'$

1011 (11), 0001 (1), 0101 (5), 0100 (4), 1011 (11), 1010 (10), 1101 (13), 1100 (12)

$F(w,x,y,z) = \Sigma(1, 4, 5, 10, 11, 12, 13)$ sum of minterms

Step 2:



$F = xy' + w'y'z + wx'y$

---

c) $w'z + xz + x'y + wx'z$

Soln:

Step 1: SOP

$F(w,x,y,z)$

$F = w'z(x+x')(y+y') + xz(w+w')(y+y') + x'y(w+w')(z+z') + wx'z(y+y')$

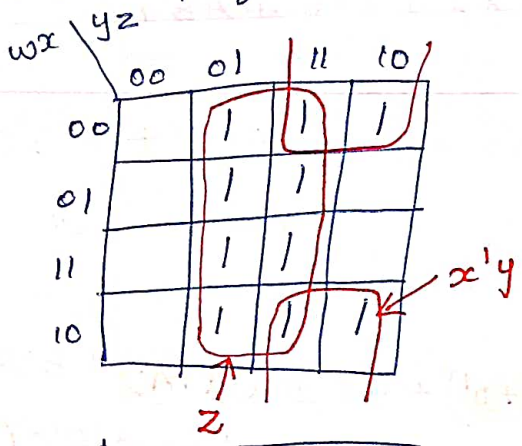
$= (w'xz + w'x'z)(y+y') + (wxz + w'xz)(y+y') + (wx'y + w'x'y)(z+z') + wx'yz + wx'y'z$

$= w'xyz + w'xy'z + w'x'yz + w'x'y'z + wxyz + wxy'z$
$+ w'xyz + w'xy'z + wx'yz + wx'yz' +$
$w'x'yz + w'x'yz' + wx'yz + wx'y'z$

$F(w,x,y,z) = \Sigma(7,5,3,1,15,13,11,10,2,9)$

$F(w,x,y,z) = \Sigma(1,2,3,5,7,9,10,11,13,15)$

Step 2: Find simplified Boolean functions using K-map



$F = z + x'y$

---

d) $B'D + A'BC' + AB'C + ABC'$

Soln:

Step 1: SOP

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

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

$= AB'CD + AB'C'D + A'B'CD + A'B'C'D + A'BC'D +$
$A'BC'D' + AB'CD + AB'CD' + ABC'D + ABC'D'$

$F(A,B,C,D) = \Sigma(11,9,3,1,5,4,10,13,12)$

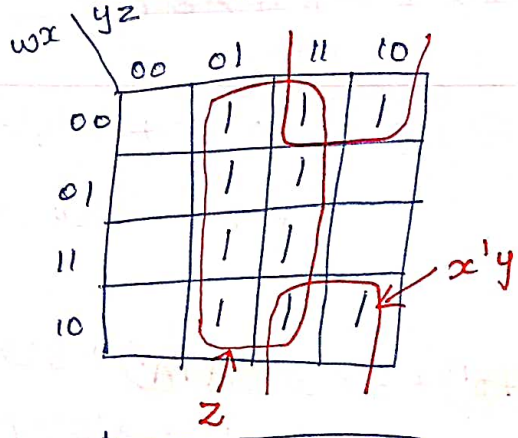
$F(A,B,C,D) = \Sigma(1,3,4,5,9,10,11,12,13)$

$= w'xyz + w'xy'z + w'x'yz + w'x'y'z + wxyz + wxy'z + w'xyz + w'xy'z + wx'yz + wx'yz' + w'x'yz + w'x'yz' + wx'yz + wx'y'z$

$F(w,x,y,z) = \Sigma(7,5,3,1,15,13,11,10,2,9)$

$F(w,x,y,z) = \Sigma(1,2,3,5,7,9,10,11,13,15)$

Step 2: Find simplified Boolean functions using K-map



$F = z + x'y$

---

d) $B'D + A'BC' + AB'C + ABC'$

Soln:

Step 1: SOP

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

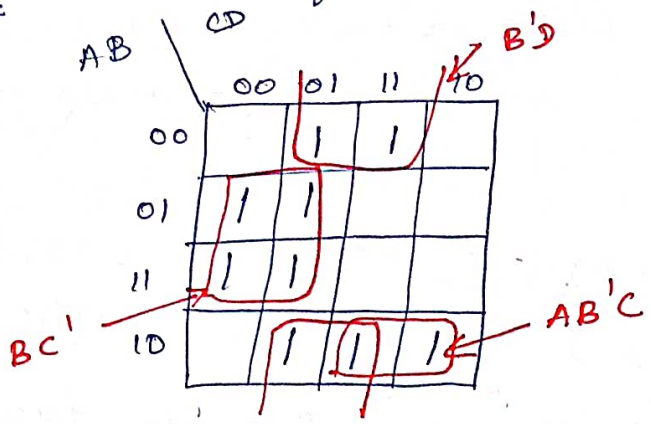
$= (AB'D + A'B'D)(C+C') + A'BC'D + A'BC'D' + AB'CD + AB'CD' + ABC'D + ABC'D'$

$= AB'CD + AB'C'D + A'B'CD + A'B'C'D + A'BC'D + A'BC'D' + AB'CD + AB'CD' + ABC'D + ABC'D'$

$F(A,B,C,D) = \Sigma(11,9,3,1,5,4,10,13,12)$

$F(A,B,C,D) = \Sigma(1,3,4,5,9,10,11,12,13)$

Step 2: Boolean functions using K-map.



$$F = BC' + B'D + AB'C$$

---

e) $AB'C + B'C'D' + BCD + ACD' + A'B'C + A'BC'D$

Soln:

Step 1: SOP.

$$f = AB'C(D+D') + B'C'D'(A+A') + BCD(A+A') + ACD'(B+B') + A'B'C(D+D') + A'BC'D$$

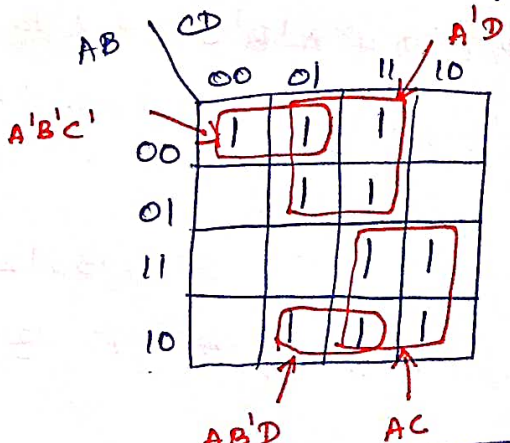
$$= \underset{1011}{AB'CD} + \underset{1010}{AB'CD'} + \underset{1000}{AB'C'D'} + \underset{0000}{A'B'C'D'} + \underset{1111}{ABCD} + \underset{0111}{A'BCD}$$

$$+ \underset{1110}{ABCD'} + \underset{1010}{AB'CD'} + \underset{0011}{A'B'CD} + \underset{0010}{A'B'CD'} + \underset{0101}{A'BC'D}$$

$$F(A,B,C,D) = \Sigma(11,10,9,0,15,7,14,3,1,5)$$

$$F(A,B,C,D) = \Sigma(0,1,3,5,7,9,10,11,14,15)$$

Step 2: Find Boolean functions using K-map.

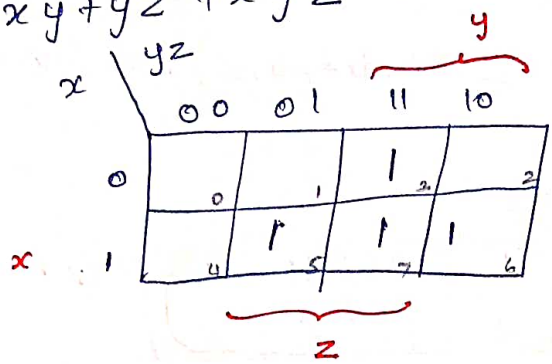


$$F = AC + A'D + A'B'C' + AB'D$$

5) Find the minterms of the following Boolean expressions by first plotting each function in a map.

3 variables.

a) $xy + yz + xy'z$ →



$xy \rightarrow 6, 7$

$yz \rightarrow 3, 7$

$xy'z \rightarrow 5$

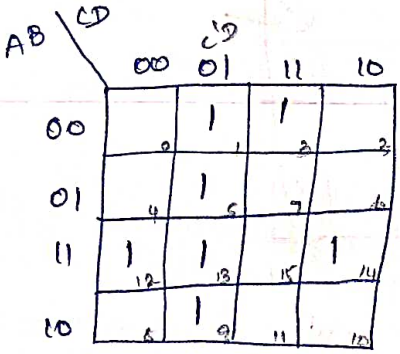
Sum of minterms

$F(x,y,z) = \Sigma\ (3, 5, 6, 7)$

---

b) $C'D + ABC' + ABD' + A'B'D$

4 variables. — A, B, C, D



$C'D \rightarrow 1, 5, 9, 13$

$ABC' \rightarrow 12, 13$

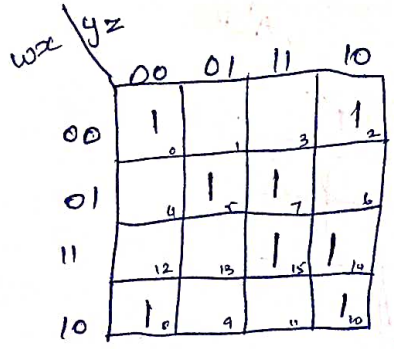
$ABD' \rightarrow 12, 14$

$A'B'D \rightarrow 1, 3$

$F(A,B,C,D) = \Sigma\ (1, 3, 5, 9, 12, 13, 14)$

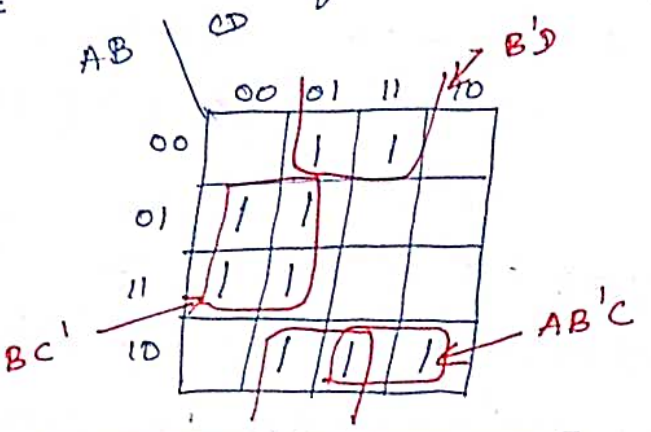
---

c) $wxy + x'z' + w'xz$



$F(w,x,y,z) = \Sigma\ (0, 2, 5, 7, 8, 10, 14, 15)$

Step 2: Boolean functions using K-map.



$$F = BC' + B'D + AB'C$$

---

e) $AB'C + B'C'D' + BCD + ACD' + A'B'C + A'BC'D$

Soln:

Step 1: SOP.

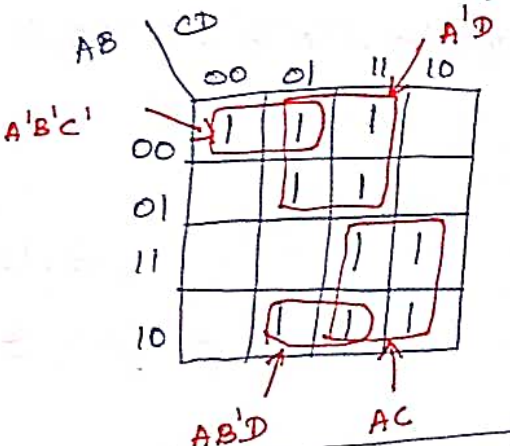
$$f = AB'C(D+D') + B'C'D'(A+A') + BCD(A+A') + ACD'(B+B') + A'B'C(D+D') + A'BC'D$$

$$= \underset{11}{AB'CD} + \underset{10}{AB'CD'} + \underset{9}{AB'C'D'} + \underset{0}{A'B'C'D'} + \underset{15}{ABCD} + \underset{7}{A'BCD} + \underset{14}{ABCD'} + \underset{10}{AB'CD'} + \underset{3}{A'B'CD} + \underset{1}{A'B'CD'} + \underset{5}{A'BC'D}$$

$$F(A,B,C,D) = \Sigma(11,10,9,0,15,7,14,3,1,5)$$

$$F(A,B,C,D) = \Sigma(0,1,3,5,7,9,10,11,14,15)$$

Step 2: Find Boolean functions using K-map.

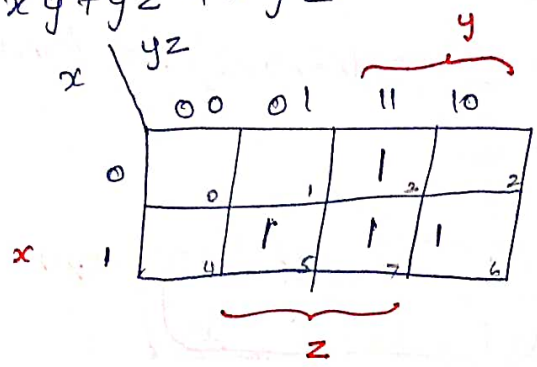


$$F = AC + A'D + A'B'C' + AB'D$$

---

5) Find the minterms of the following Boolean expressions by first plotting each function in a map.

a) $xy + yz + xy'z$ → 3 variables.



$xy \rightarrow 6, 7$
$yz \rightarrow 3, 7$
$xy'z \rightarrow 5$

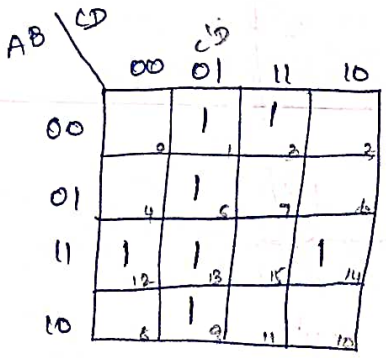
Sum of minterms

$F(x,y,z) = \Sigma\ (3, 5, 6, 7)$

---

b) $C'D + ABC' + ABD' + A'B'D$

4 variables. — A, B, C, D

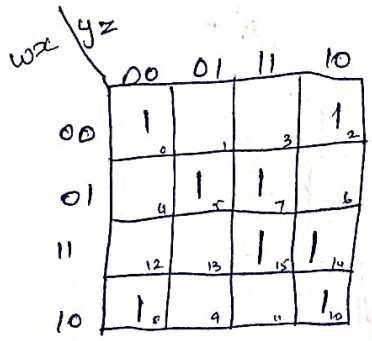


$C'D \rightarrow 1, 5, 9, 13$
$ABC' \rightarrow 12, 13$
$ABD' \rightarrow 12, 14$
$A'B'D \rightarrow 1, 3$

$F(A,B,C,D) = \Sigma\ (1, 3, 5, 9, 12, 13, 14)$

---

c) $wxy + x'z' + w'xz$



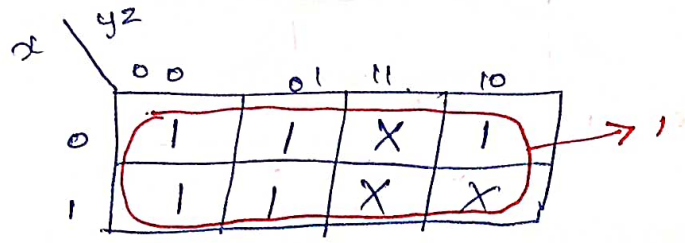
$F(w,x,y,z) = \Sigma\ (0, 2, 5, 7, 8, 10, 14, 15)$

---

6) Simplify the following Boolean function F, together with don't care conditions d and then express the simplified function in sum of minterms.

a) $F(x,y,z) = \Sigma(0,1,2,4,5)$, $d(x,y,z) = \Sigma(3,6,7)$

Soln:



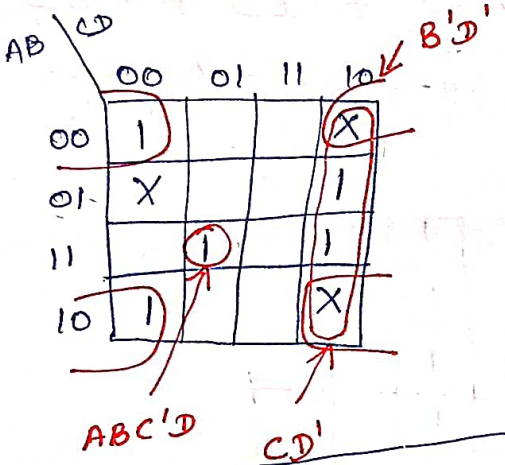
$F = 1$

---

b) $F(A,B,C,D) = \Sigma(0,6,8,13,14)$

$d(A,B,C,D) = \Sigma(2,4,10)$

Soln:

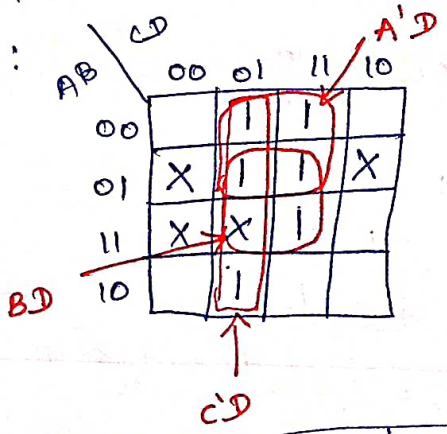


$F = B'D' + CD' + ABC'D$

---

c) $F(A,B,C,D) = \Sigma(1,3,5,7,9,15)$, $d(A,B,C,D) = \Sigma(4,6,12,13)$

Soln:



$F = C'D + A'D + BD$

Analysis and Design Procedures

→ Analysis:

* Determine the function that the circuit implements.
  - Starts with a logic diagram
  - Culminates with a set of Boolean functions, a truth table or a possible explanation of the circuit operation.

Step 1:

* The first step in the analysis is to make sure that the given circuit is combinational and not sequential
  - logic gates with no feedback paths or memory elements

→ feedback path:
  - a connection from the output of one gate to the input of a second gate that forms part of the input to the first gate.
  - feedback paths define a sequential circuit.

Step 2:

* Proceed to obtain the output Boolean functions or the truth table.

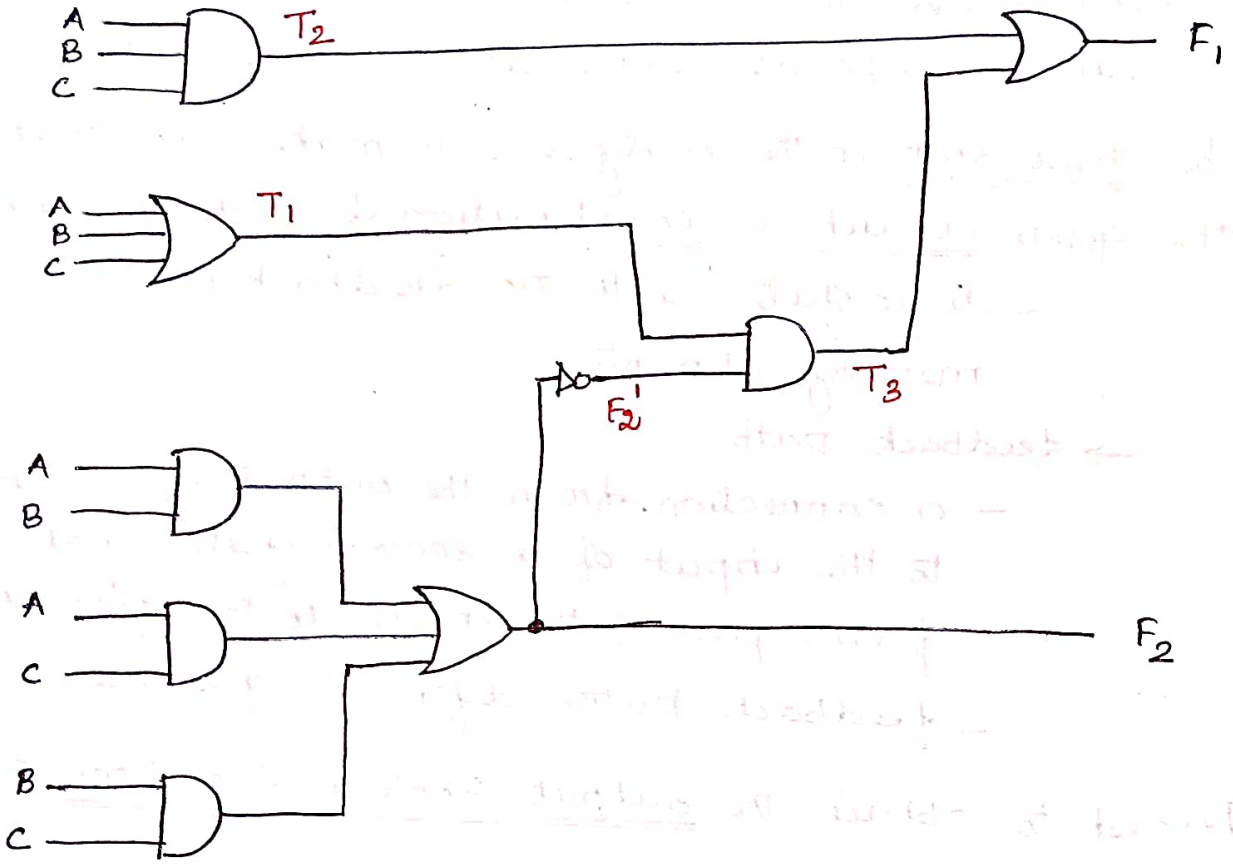
a) Obtain the output Boolean functions from a logic diagram

Procedure:

1. Label all gate outputs
   - that are a function of input variables with symbo
   * Determine the Boolean functions for each gate output
2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols.
   * Find the Boolean functions for these gates.
3. Repeat the process in step 2 until the outputs of the circuit are obtained.
4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

Example:

Logic diagram for Analysis Example:



Analysis: a) Obtain Boolean functions:

Step 1:

- Label the gates o/ps, T_1, T_2

Step 2:

- Label the gates - i/p & previously labeled gates

T_3, F_2'

Boolean functions on initial step: (from step 1)

$$F_2 = AB + AC + BC$$
$$T_1 = A + B + C$$
$$T_2 = ABC$$

Next step: from already defined symbols (from step2)

$$T_3 = F_2' T_1$$
$$F_1 = T_3 + T_2$$

Step 3:

- Obtain F_1 as a function of A, B, C

* Substitution.

$$F_1 = T_3 + T_2$$

$$= F_2' T_1 + ABC$$

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

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

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

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

$$\boxed{F_1 = A'BC' + A'B'C + AB'C' + ABC}$$

b) Obtain the Truth Table:

Procedure:

1. Determine the number of input variables in the circuit.
   - For n inputs, form the 2^n possible input combinations and list the binary numbers from 0 to $2^n - 1$ in a table.
2. Label the outputs of selected gates with arbitrary symbols.
3. Obtain the truth table for the outputs of those gates that are a function of the input variables only.
4. Proceed to obtain the truth table for the outputs of those gates that are a function of previously defined values until the columns for all outputs are determined.

—

b) Truth Table

A	B	C	F_2	F_2'	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

Design Procedure:

* The design of combinational circuits starts from the specification of the problem and culminates in a logic circuit diagram or a set of Boolean functions.

Procedure:

1. From the specification of the circuit,
   - determine the required number of inputs and outputs
   - assign a symbol to each.
2. Derive the truth table
   - defines the relationship between inputs and outputs
3. Obtain the simplified Boolean functions
   - for each output as a function of the input variables
4. Draw the logic diagram and verify the correctness of the design.

Guidelines:

* Verbal specifications be interpreted correctly in the truth table.
* Simplified Boolean functions by using any method. (algebraic manipulation, map method ...)
* Minimum number of inputs, gates, interconnections
* Limitations of the driving capability of each gate should be

Binary Adder - Subtractor

* A binary adder-subtractor is a combinational circuit that performs the arithmetic operations of addition and subtraction with binary numbers.

a) Half adder → addition of 2 bits
b) Full adder → addition of 3 bits
c) Binary adder → connecting n full adders in cascade for 2 n-bit numbers
d) Binary subtractor → complementing circuit.

a) Half adder :

* A combinational circuit that performs the addition of two bits is called a half adder.

Design :

Step 1 : Number of inputs & outputs:

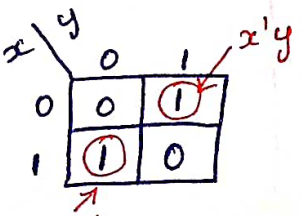
Number of inputs - 2 → x & y
Number of outputs - 2 → S & C

Step 2 : Derive the truth Table

x	y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

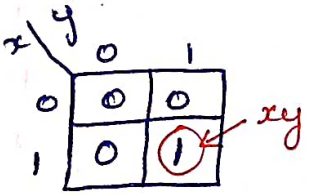
Step 3 : Obtain Boolean functions

Sum:



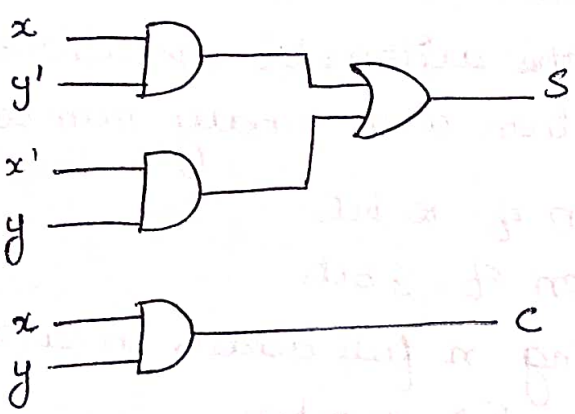
$S = x'y + xy'$

Carry:



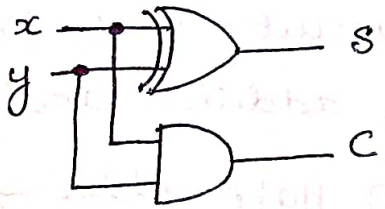
$C = xy$

Step 1: Draw the logic diagram:



$S = xy' + x'y$

$C = xy$



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

$S = x \oplus y$

$C = xy$

b) Full Adder:

* A full adder is a combinational circuit that forms the arithmetic sum of three bits.

Design a full adder circuit

Step 1: Number of inputs & outputs

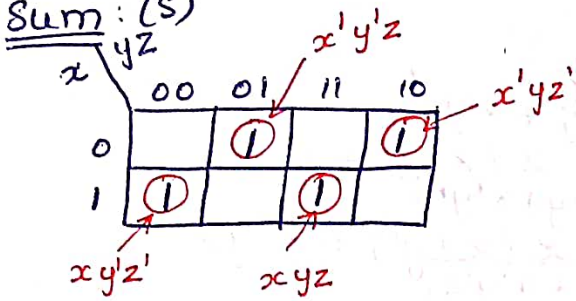
* Number of inputs - 3 → x, y, z
* Number of outputs - 2 → S, C

Step 2: Derive the truth table:

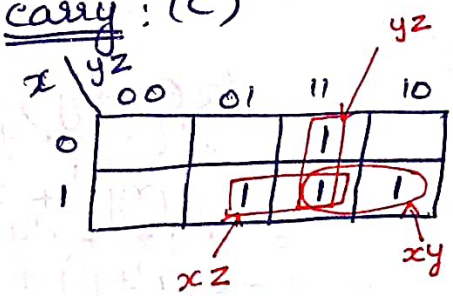
x	y	z	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Step 3: Obtain the Boolean functions:

Sum : (S)



carry : (C)



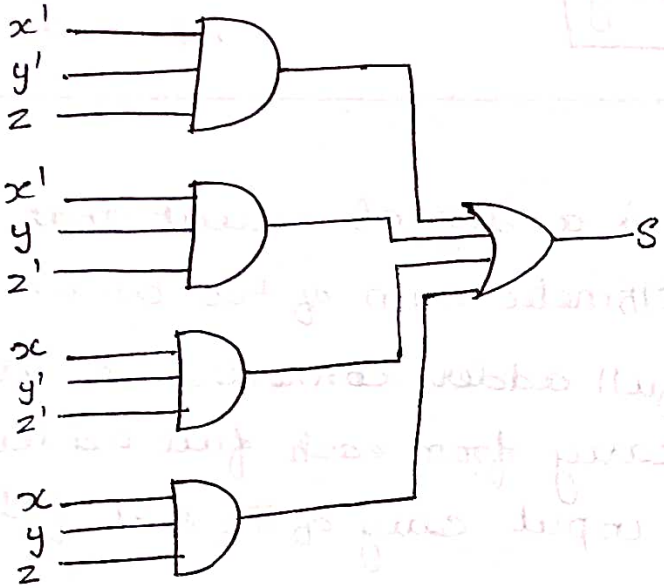
Simplified Expressions:

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

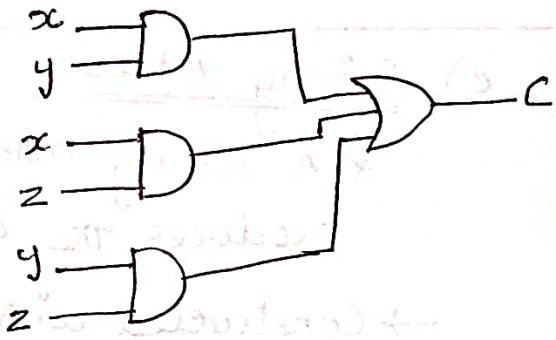
$$C = xy + xz + yz$$

Step 4: Draw the logic diagram:

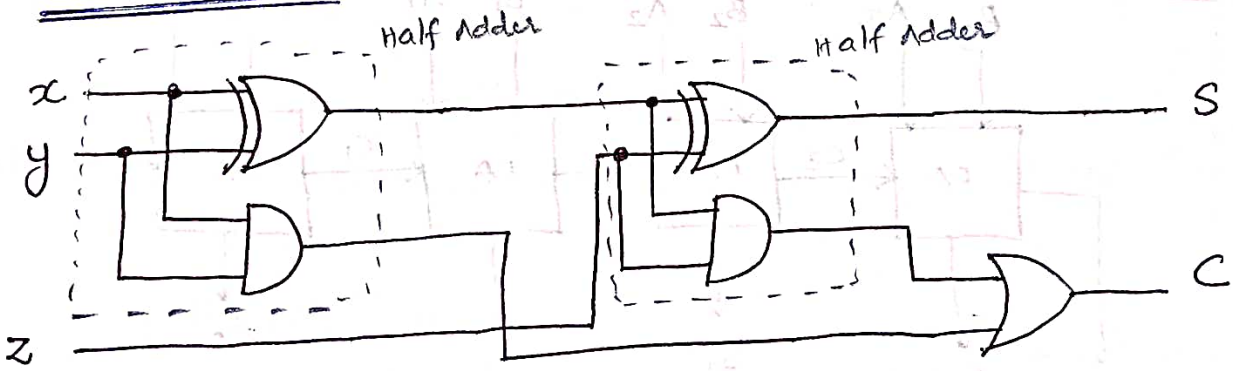
Sum



carry



Implementation of Full Adder with Two Half Adders and an OR Gate:



Sum output:

$$S = z \oplus x \oplus y$$

Carry output:

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

$$S = z \oplus (x \oplus y)$$

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

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

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

$$\boxed{S = xy'z' + x'yz' + xyz + x'y'z}$$

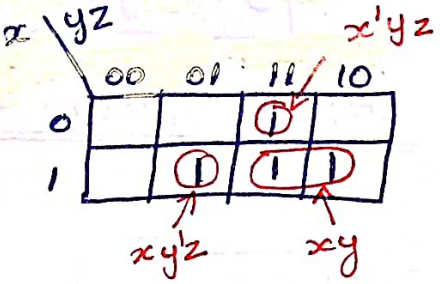
$x \oplus y = xy' + x'y$

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

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

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

$$\boxed{C = xy'z + x'yz + xy}$$

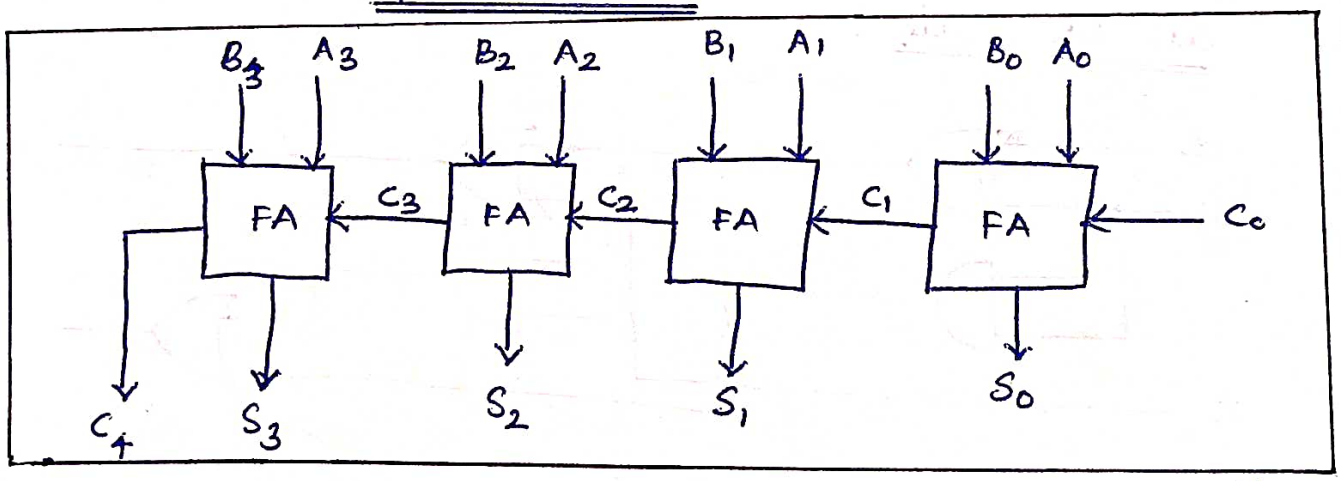


c) **Binary Adder:**

* A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.

→ 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.

4-Bit Adder



* The interconnection of four full Adder (FA) circuits to provide a 4-bit binary ripple carry adder.

* The augend bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting LSB.
- The carries are connected in a chain through the full adders
- The input carry to the adder is C_0 and it ripples through the full adders to the output carry C_4.
- The S outputs generate the required sum bits.

* An n-bit adder requires a full adders with each output carry connected to the input carry of the next higher-order full adder.

Example:

A = 1011 , B = 0011

Subscript i:	3	2	1	0
Input carry	0	1	1	0
Augend	1	0	1	1
Addend	0	0	1	1
Sum	1	1	1	0
output carry	0	0	1	1

* Sum S = 1110 is formed with the four-bit adder.

→ The bits are added with full adders, starting from the LSB, to form the sum bit and carry bit.
- The input carry C_0 in the LSB must be 0
- The value of C_{i+1} is the output carry of the full adder.
  * This value is transferred into the input carry of the full adder
    - adds the bits one higher significant position to the left.
- The sum bits are then generated starting from the rightmost position and are available as soon as the corresponding

previous carry bit is generated.
- All the carries must be generated for the correct sum bits to appear at the outputs.

* The 4-bit adder can be used in many applications involving arithmetic operations.

Carry Propagation:

* The addition of two binary numbers in parallel implies that all the bits of the augend and addend are available for computation at the same time.
* As in any combinational circuit, the signal must propagate through the gates before the correct output sum is available in the output terminals.

→ **Total Propagation Time:**
- is equal to the propagation delay of a typical gate times the number of gate levels in the circuit.

→ **Longest Propagation delay time in an adder**
- is the time it takes the carry to propagate through the full adders.

* Each bit of the sum output depends on the value of the input carry, the value of S_i in any given stage in the adder will be in its steady state final value only after the input carry to that stage has been propagated.

- Inputs A_3 and B_3 are available as soon as input signals are applied to the adder.
- 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 correct value.

Carry Lookahead Generator

* Define 2 binary variables

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

* output sum and carry can be expressed as

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

G_i → carry generate
- it produces a carry of 1 when both A_i and B_i are 1, regardless of the input carry C_i.

P_i → carry propagate
- propagation of the carry from C_i to C_{i+1}

Boolean Functions:

- carry outputs of each stage & substitute for each C_i from previous equations

C_0 = input carry

$$C_1 = G_0 + P_0 C_0$$

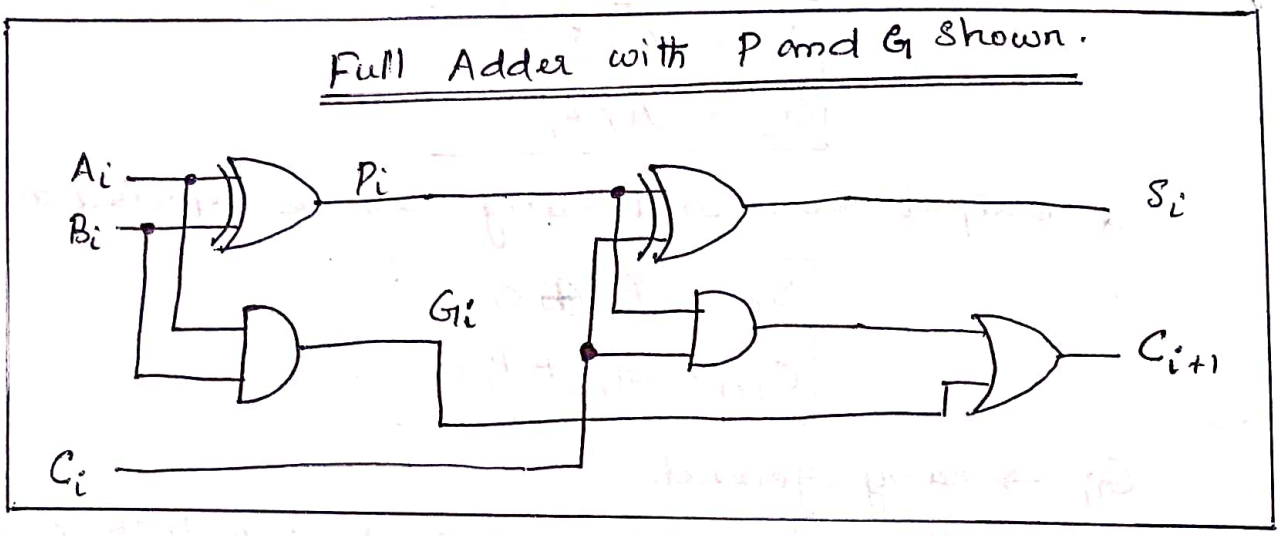
$$C_2 = G_1 + P_1 C_1$$
$$= G_1 + P_1 (G_0 + P_0 C_0)$$
$$= G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$C_3 = G_2 + P_2 C_2$$
$$= G_2 + P_2 (G_1 + P_1 G_0 + P_1 P_0 C_0)$$
$$= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

* Each output carry is expressed in sum of products
* Each function can be implemented with one level of AND gates followed by an OR gate. (2 level NAND)

- Three Boolean functions for C_1, C_2 & C_3 are implemented in the carry lookahead generator.

* The number of gate levels for the carry propagation can be found from the circuit of the full adder.



$i \rightarrow$ stage in the adder.

* The signals at P_i and G_i settle to their steady state values after they propagate through their respective gates.
  - Two signals are common to all full adders
  - depend only on the input augend & addend bits.
* The signal from the input carry C_i to the output carry C_{i+1}, propagates through an AND gate and OR gate.
  - 2 gate levels.
* For n-bit adder, there are 2n gate levels for the carry to propagate from input to output.

Reducing carry Propagation Time:

* The outputs will not be correct unless the signals are given enough time to propagate through the gates connected from the inputs to the outputs.

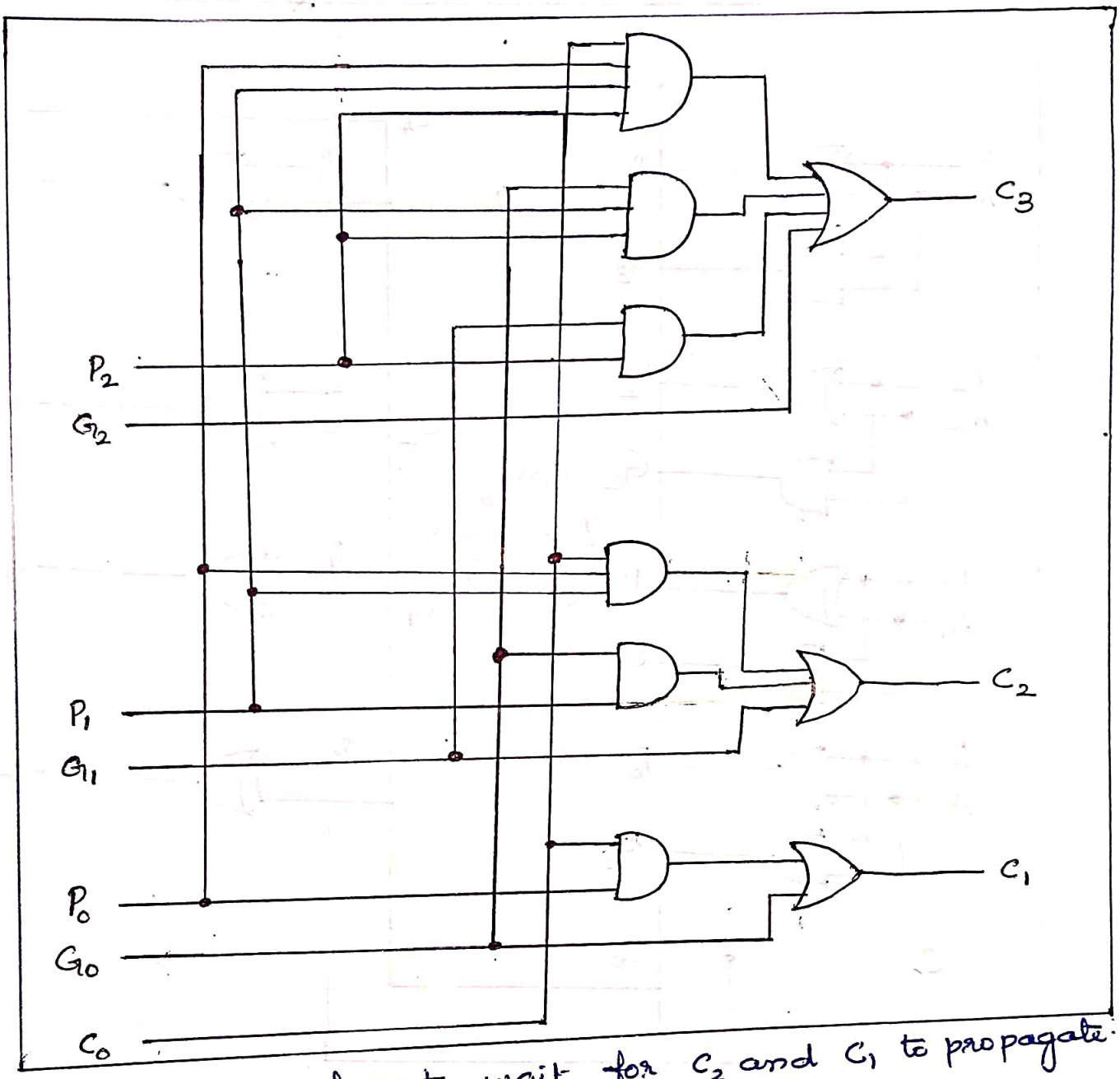
Solution: - for reducing carry Propagation time.

i) To employ faster gates with reduced delays.

ii) To increase the equipment complexity in such a way that the carry delay time is reduced.

iii) most widely used technique employs the principle of carry lookahead.

Logic Diagram of Carry Lookahead Generator

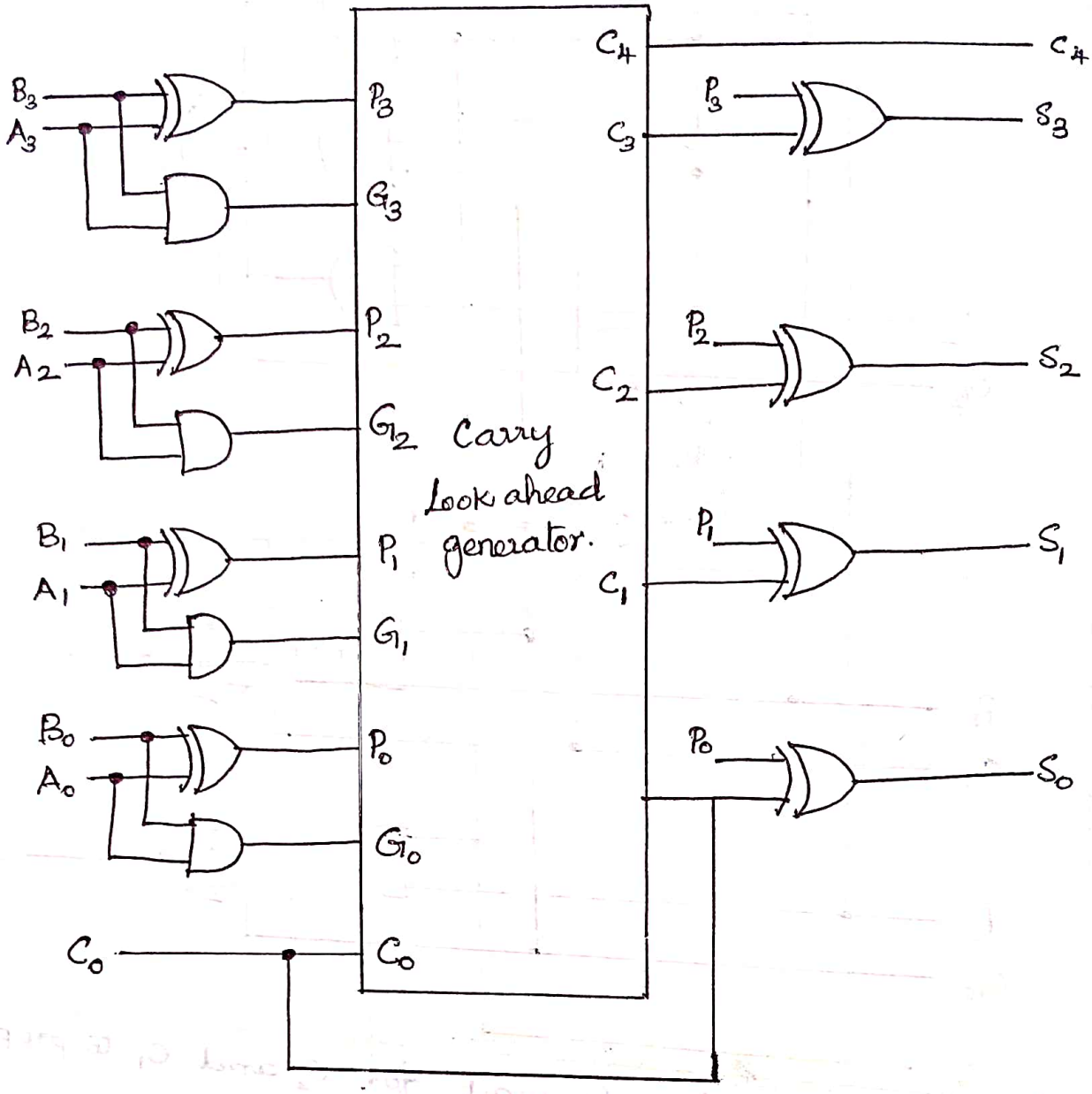


* C_3 does not have to wait for C_2 and C_1 to propagate.
* C_3 is propagated at the same time as C_1 and C_2

Construction of a 4-bit adder with a carry lookahead scheme:

* Each sum output requires two exclusive-OR gates.
  - The output of the first exclusive-OR gate generates the P_i variable
  - The AND gate generates the G_i variable.
* The carries are propagated through the carry lookahead generator. and applied as inputs to the second exclusive-OR gate.

4-bit Adder with Carry Lookahead



* All output carries are generated after a delay through two levels of gates.

- Outputs S_1 through S_3 have equal propagation delay times

1) Half Subtractor:

* A combinational circuit that performs subtraction of two bits is called a half subtractor.

Design:

Step 1: i/ps & o/ps.

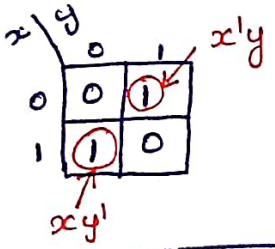
Number of inputs – 2 → x & y.
Number of outputs – 2 → Borrow (B), Difference (D)

Step 2: Derive the Truth Table:

x	y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

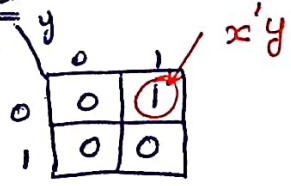
Step 3: Obtain Boolean functions

Difference (D):



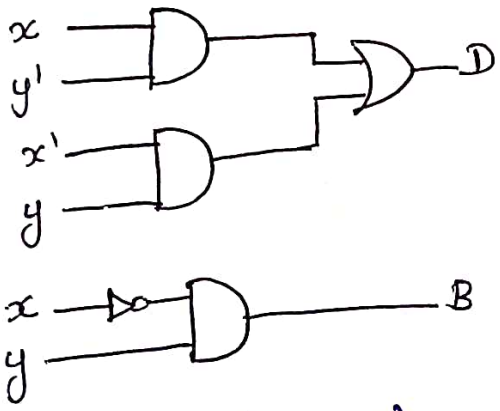
$$D = x'y + xy' = x \oplus y$$

Borrow (B):

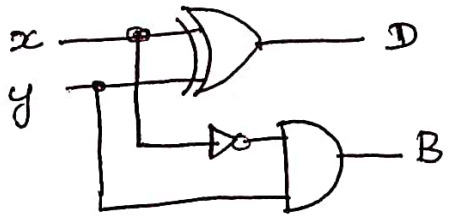


$$B = x'y$$

Step 4: Draw the logic diagram:



$$D = x'y + xy'$$
$$B = x'y$$



$$D = x \oplus y$$
$$B = x'y.$$

e) Full Subtractor:

* A combinational circuit that performs subtraction involving three bits.

Design:

Step 1: Inputs & outputs.

Number of inputs – 3 → x, y, B_{in}

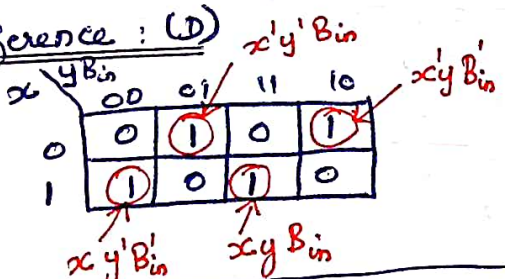
Number of outputs – 2 → Difference (D), Borrow out (B_{out})

Step 2: Derive the truth table:

x	y	B_{in}	D	B_{out}
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

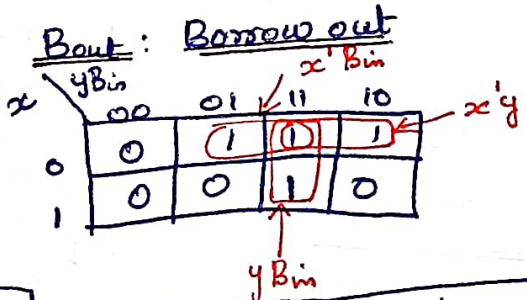
Step 3: Obtain the Boolean functions:

Difference: (D)



$D = x'y'B_{in} + x'yB'_{in} + xy'B'_{in} + xyB_{in}$

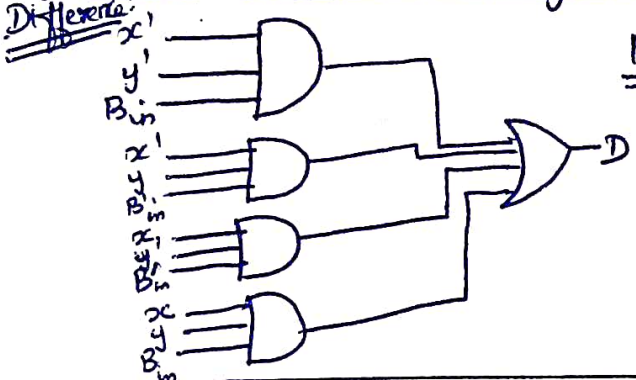
B_{out}: Borrow out



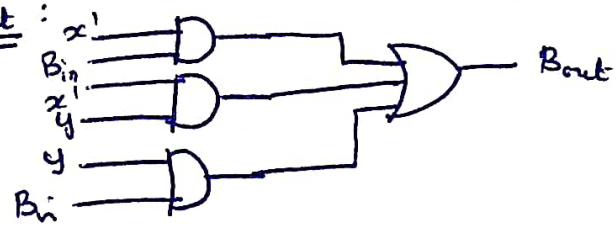
$B_{out} = x'B_{in} + x'y + yB_{in}$

Step 4: Draw the logic diagram:

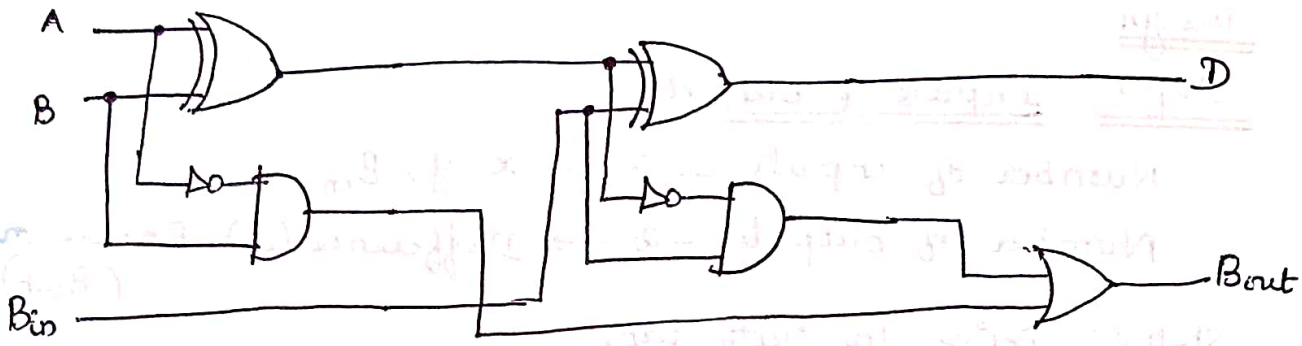
Difference:



B_{out}:



Implementation of Full Subtractor with 2 Half-Subtractors and an OR Gate



→ **Difference (D):**

$$B_{in} \oplus (x \oplus y)$$

→ **Borrow out (Bout):**

$$B_{out} = B_{in}(x \oplus y) + x'y.$$

Difference output:

$$D = B_{in} \oplus (x \oplus y)$$

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

$$= B_{in} \oplus (xy' + x'y)$$

$$= B_{in}(xy' + x'y)' + B'_{in}(xy' + x'y) \qquad (x \oplus y)' = xy + x'y'$$

$$= B_{in}(xy + x'y') + B'_{in}(xy' + x'y)$$

$$= B_{in}xy + B_{in}x'y' + B'_{in}xy' + B'_{in}x'y$$

$$\boxed{D = xyB_{in} + x'y'B_{in} + xy'B'_{in} + x'yB'_{in}}$$

Borrow output:

$$B_{out} = B_{in}(x \oplus y) + x'y$$

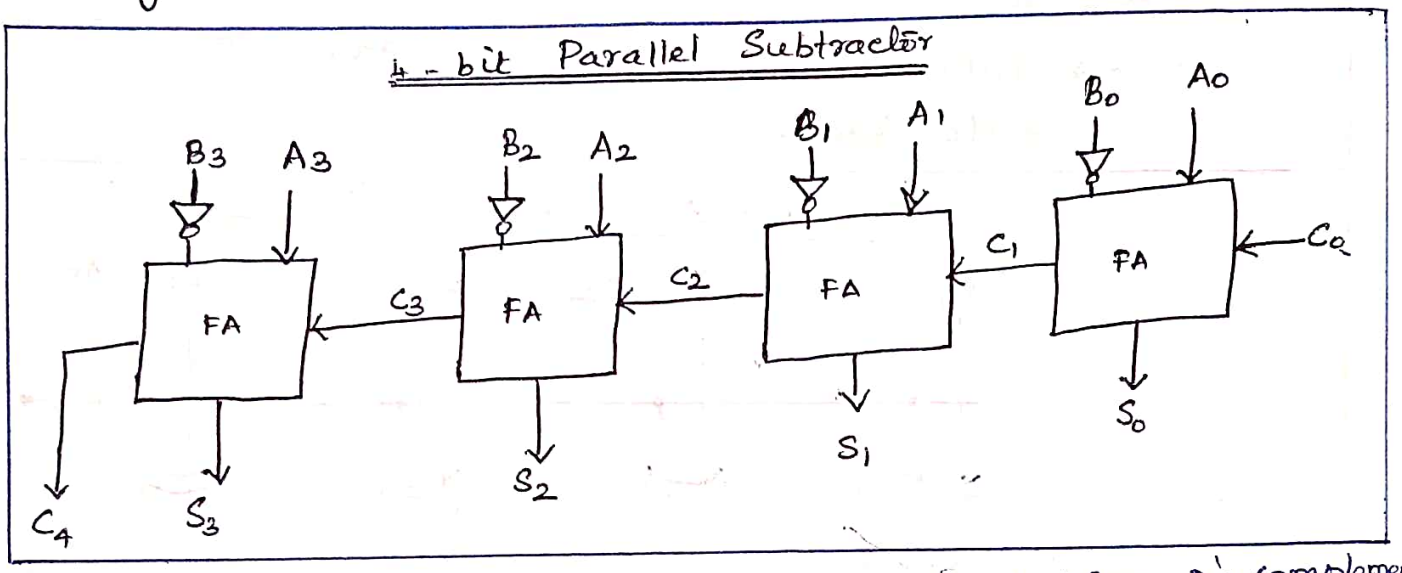
$$= B_{in}(xy' + x'y) + x'y$$

$$\boxed{B_{out} = xy'B_{in} + x'yB_{in} + x'y}$$

e) Binary Subtractor:

* The subtraction of unsigned binary numbers can be done by means of complements.

4 - bit Parallel Subtractor



* The subtraction A - B can be done by taking 2's complement of B and adding it to A.

→ 2's complement:

i) Taking 1's complement

ii) Adding 1 to the LSB.

* 1's complement → inverter → implementation
  - one can be added to the sum through the input carry.

* The circuit for subtracting A-B consists of an adder with inverters placed between each data input B and the corresponding input of the full adder.

* The input carry Co must be equal to 1 when performing subtraction.

ie) $C_0 = 1$

A + 2's complement of B.

* For unsigned numbers,

A - B if A ≥ B

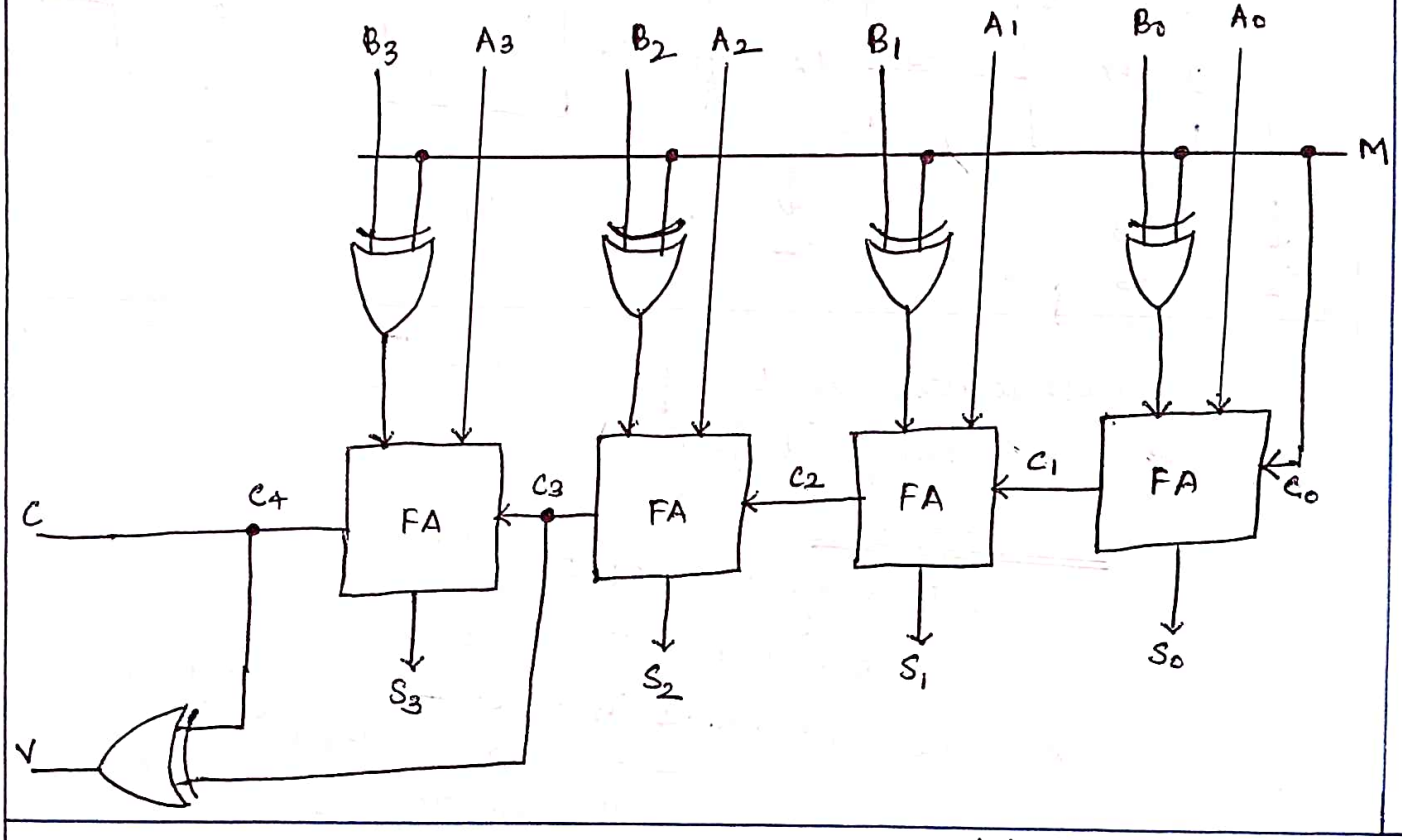
2's comp. of (B-A) if A < B

* For signed numbers, the result is A-B.
  - there is no overflow.

* 4-bit Adder Subtractor :

* The addition and subtraction operations can be combined into one circuit with one common binary adder.
  → Include an exclusive-OR gate with each full adder.

4-bit Adder-Subtractor Circuit.



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

→ Each exclusive-OR gate receives input M and one of the inputs of B.
  ⇒ M=0 ; $B \oplus 0 = B$
  - full adders receive the value of B, the input carry is 0, the circuit performs A plus B.
  ⇒ M=1 ; $B \oplus 1 = B'$, $C_0 = 1$
  - The B inputs are all complemented
  - 1 is added through the input carry.

* The circuit performs the operation A + 2's comp.(B)

* output V – for detecting an overflow.

* **Overflow:**
  * When two numbers of n digits each are added
    - the sum occupies n+1 digits
      ⇒ an overflow occurs.
  * overflow is a problem in digital computers.
    - no of bits that hold the number is finite.
    - flip-flop is set that can be checked by the user.

⇒ **Unsigned numbers:**

* When two unsigned numbers are added.
  - the overflow is detected from the end carry

⇒ **Signed numbers:**

* When two signed numbers are added.
  - LSB - sign bit
  - end carry does not indicate an overflow.

⇒ one number is +ve & the other is negative,
  - overflow cannot occur.

⇒ both +ve or both -ve
  - overflow may occur.

Examples:

carries:	0 1	carries:	1 0
+70	0 1000110	-70	1 0111010
+80	0 1010000	-80	1 0110000
+150	1 0010110	-150	0 1101010

* An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position.
  - If two carries are not equal, an overflow has occurred

* If the two carries are applied to an exclusive-OR gate, an overflow is detected when the output of the gate is equal to 1.

* The binary adder-subtractor circuit with outputs C and V.

Unsigned numbers:

* If the two binary numbers are unsigned,
  - then the C bit detects a carry after addition or a borrow after subtraction.

Signed numbers:

* If the numbers are signed,
  - then the V bit detects an overflow.

→ V=0, after an addition/subtraction:
  - it indicates that no overflow occurred
  - n-bit result is correct.

→ V=1,
  - the result of the operation contains n+1 bits
  - ∴ an overflow has occurred.

* The (n+1)th bit is the actual sign and has been shifted out of position.

DECIMAL ADDER / BCD Adder

* Adds two BCD digits and produces a sum digit in BCD.
* Computers or calculators that perform arithmetic operations directly in the decimal number system
  - decimal numbers in binary coded form.

Step 1:

* A decimal adder requires a minimum of nine inputs and five outputs
  - 4 bits for each decimal digit
  - the circuit must have an input and output carry.

9 - 1001 (4), carry out → 1 (5)

A = 6 0110 } 8 } → 9.
B = 3 0011 }
carry in → 1

BCD Adder:

* Arithmetic addition of 2 decimal digits in BCD.
* Each input digit does not exceed 9
  - output sum cannot be greater than 9+9+1 = 19
    1 → input carry.

Derivation of BCD Adder.

Step 2:

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

contd....

Binary sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
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 → input carry.

subscript → weights (8, 4, 2, 1)

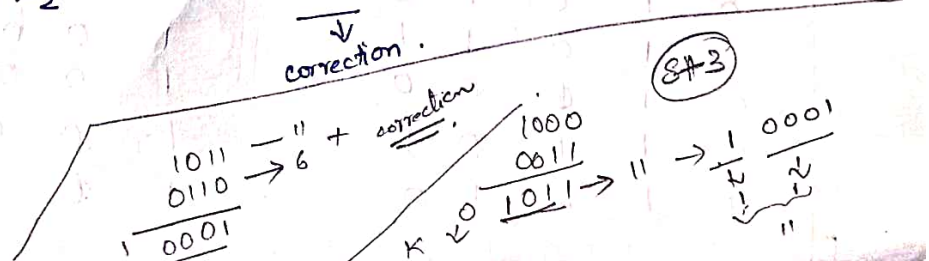
* The columns under the binary sum list the binary value that appears in the outputs of the 4-bit binary adder.
* The output sum of two decimal digits must be represented in BCD.

→ To find a rule by which the binary sum is to be converted to the correct BCD digit representation of the number in the BCD sum.

* binary sum ≤ 1001 → no conversion is needed
  → corresponding BCD number is identical.

* binary sum > 1001 → obtain a non-valid BCD representation.
  * The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation
    – also produces an output carry as required.

* 1010 to 1111, ① $K = 1$ → correction needed, ② $Z_8 = 1$
* 1000, 1001 → Z_4 / Z_2 must have a 1. ie) ③ $Z_4, Z_2 \rightarrow 0$.
  check. ↓ correction.



Step 3:

Boolean function:

* The condition for a correction and an output carry can be expressed by the Boolean function

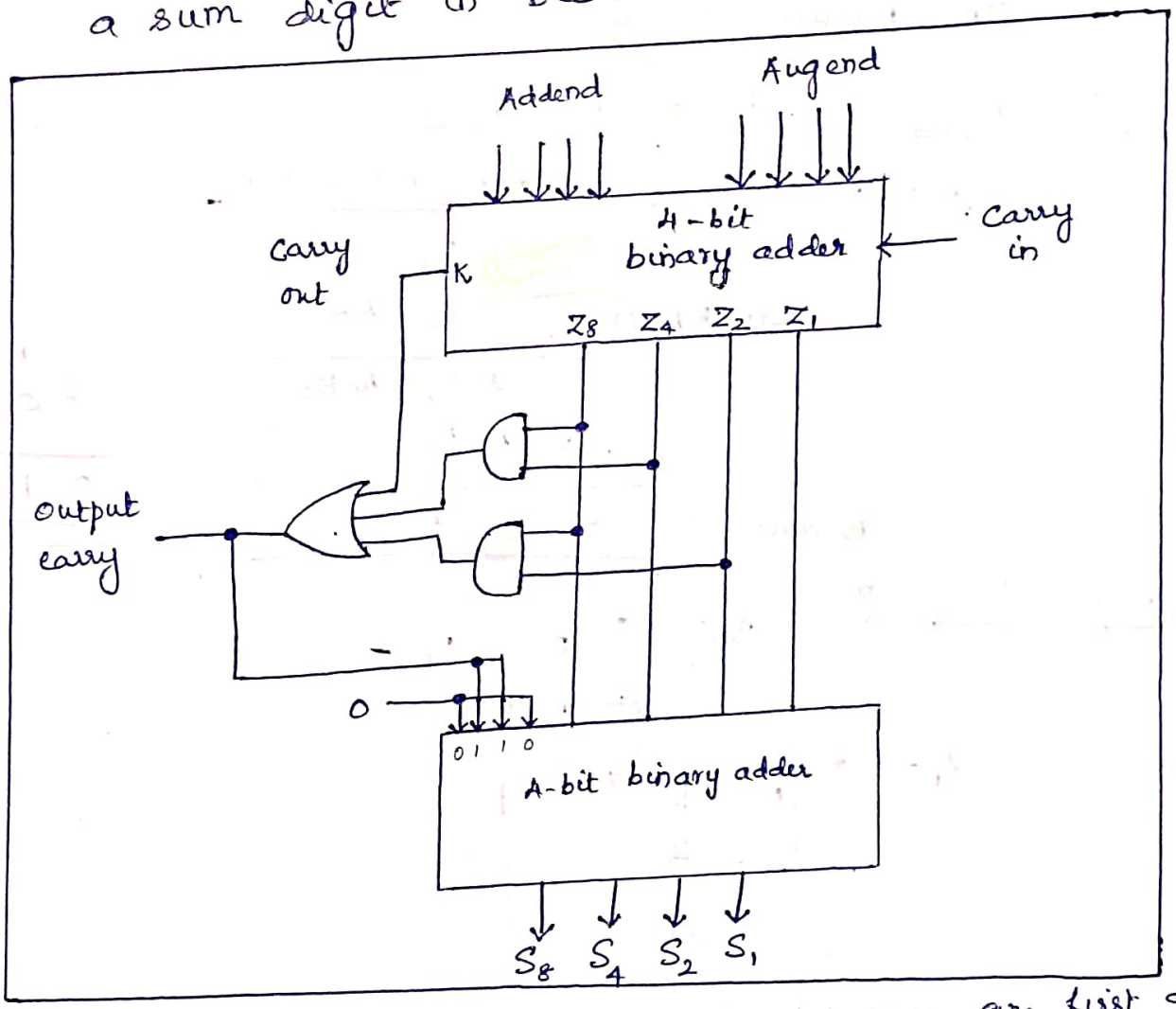
$$C = K + Z_8 Z_4 + Z_8 Z_2$$

$C = 1$,

* Add 0110 to the binary sum and provide an output carry for the next stage.

Step 4: Block diagram.

* A BCD adder that adds two BCD digits and produces a sum digit in BCD.



* 2 decimal digits, together with the input carry, are first added in the top 4-bit adder to produce the binary sum.
  - output carry = 0, nothing is added to the binary sum.
  - output carry = 1, binary 0110 is added to the binary sum through the bottom 4-bit adder.
* A decimal parallel adder - adds n decimal digits needs n BCD adder stages
* output carry from one stage must be connected to the input carry of the next higher-order stage.

Magnitude comparator

* The comparison of two numbers is an operation that determines if one number is i) greater than,
ii) less than, or
iii) equal to

the other number.

* A magnitude comparator is a combinational circuit that compares two numbers, A and B and determines their relative magnitude.

– The outcome of the comparison is specified by three binary variables that indicates whether

$A > B$
$A = B$
$A < B$.

Step 1:

* The circuit for comparing two n-bit numbers.

Op: * 3 binary variables. – $A>B$, $A=B$, $A<B$.

step 2:

* 2^{2n} entries in the truth table.

→ designed by means of an algorithmic procedure.

* An algorithm is a procedure that specifies a finite set of steps, if followed, give the solution to the problem.

Step 3: Algorithm for the design of a 4-bit Magnitude comparator. & Boolean functions.

→ Compare the relative magnitudes of two numbers.

Two numbers – A & B with four digits each.

Write the coefficients of the numbers:

$$A = A_3\ A_2\ A_1\ A_0$$

$$B = B_3\ B_2\ B_1\ B_0$$

* Each subscripted letter represents one of the digits in the number.

(i) $A = B$:

* The two numbers are equal if all pairs of significant digits are equal:

$$A_3 = B_3,\ A_2 = B_2,\ A_1 = B_1,\ A_0 = B_0$$

* Equality relation of each pair of bits

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

$x_i = 1$ only if the pair of bits in position i are equal.

* equality is displayed in a combinational circuit by an output binary variable, designate by the symbol $(A = B)$.
  - The binary variable is equal to 1 if the input numbers are equal.
  - it is equal to 0, otherwise.
* For equality, all x_i variables must be equal to 1.
  - AND operation of all variables:

$$(A = B) = x_3 x_2 x_1 x_0$$

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

(ii) $(A > B)$ or $(A < B)$:

* To determine $A < B$ or $A > B$, inspect the relative magnitudes of pairs of significant digits starting from the MSB. (most significant bit)

* If the two digits are equal, compare the next lower significant pair of digits.
* This comparison continues until a pair of unequal digits is reached.

→ $A = 1, B = 0 \implies A > B$.

$A = 0, B = 1 \implies A < B$.

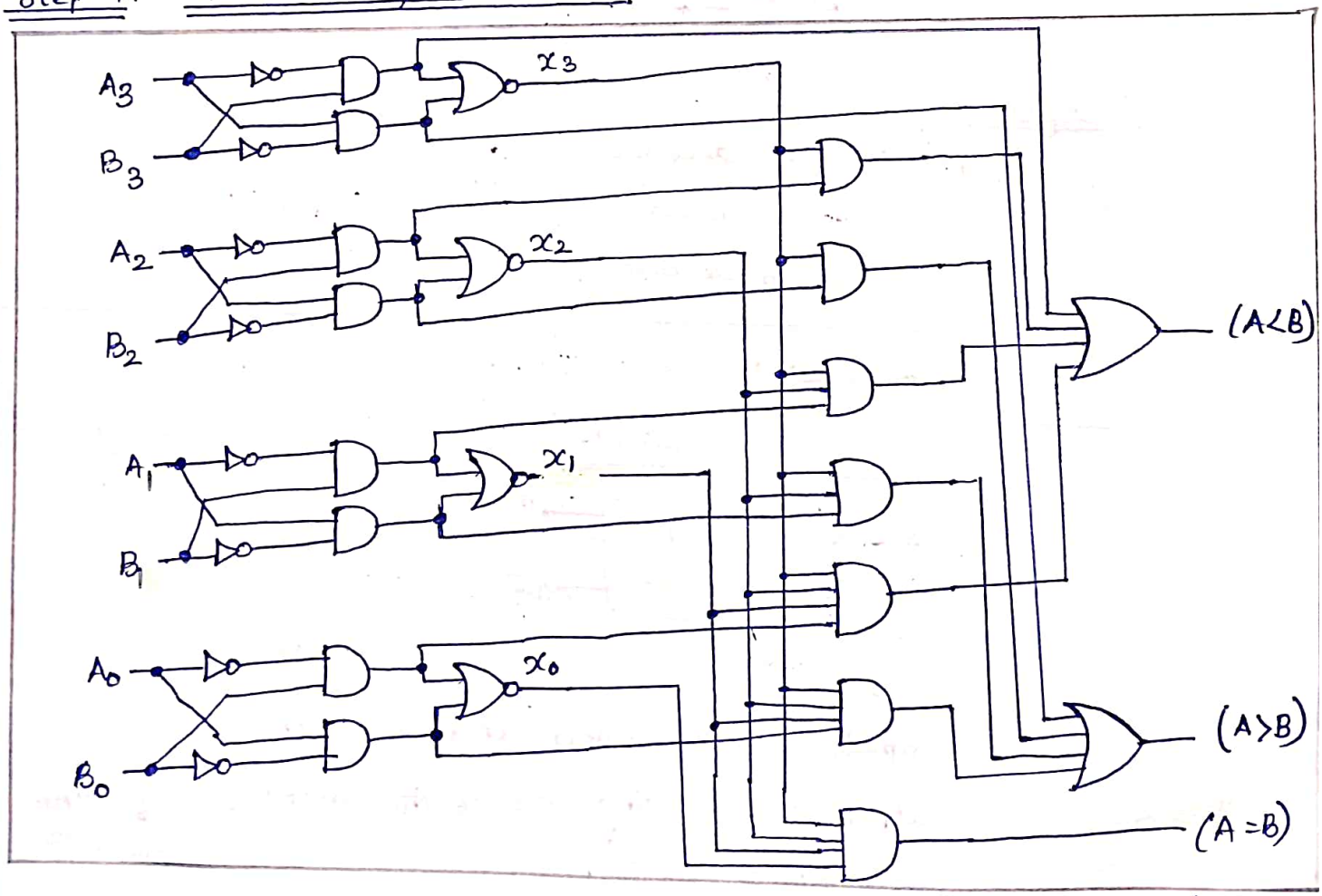
Boolean functions:

$$(A > B) = A_3 B_3' + x_3 A_2 B_2' + x_3 x_2 A_1 B_1' + x_3 x_2 x_1 A_0 B_0'$$

$$(A < B) = A_3' B_3 + x_3 A_2' B_2 + x_3 x_2 A_1' B_1 + x_3 x_2 x_1 A_0' B_0$$

* The symbols $(A > B)$ & $(A < B)$ are binary output variables that are equal to 1 when $A > B$ or $A < B$, respectively.

Step 4: Gate Implementation:



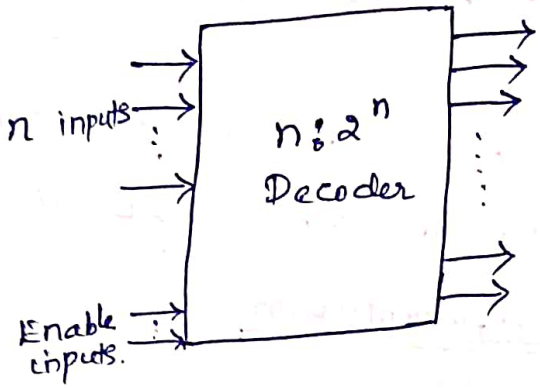
* x outputs are generated with exclusive-NOR circuits and applied to an AND gate for $(A = B)$ output.

2.7 Decoders

* A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines.
* Also called n-to-m line decoders, $m \leq 2^n$.
* To generate 2^n (or fewer) minterms of n input variables.

→ used in conjunction with other code converters such as BCD-to-seven-segment decoder.

General Structure



Types:

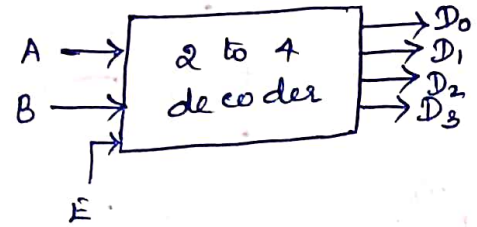
i) 2 to 4 decoder
ii) 3 to 8 decoder
iii) 4 to 16 decoder.

---

① 2 to 4 Decoder:

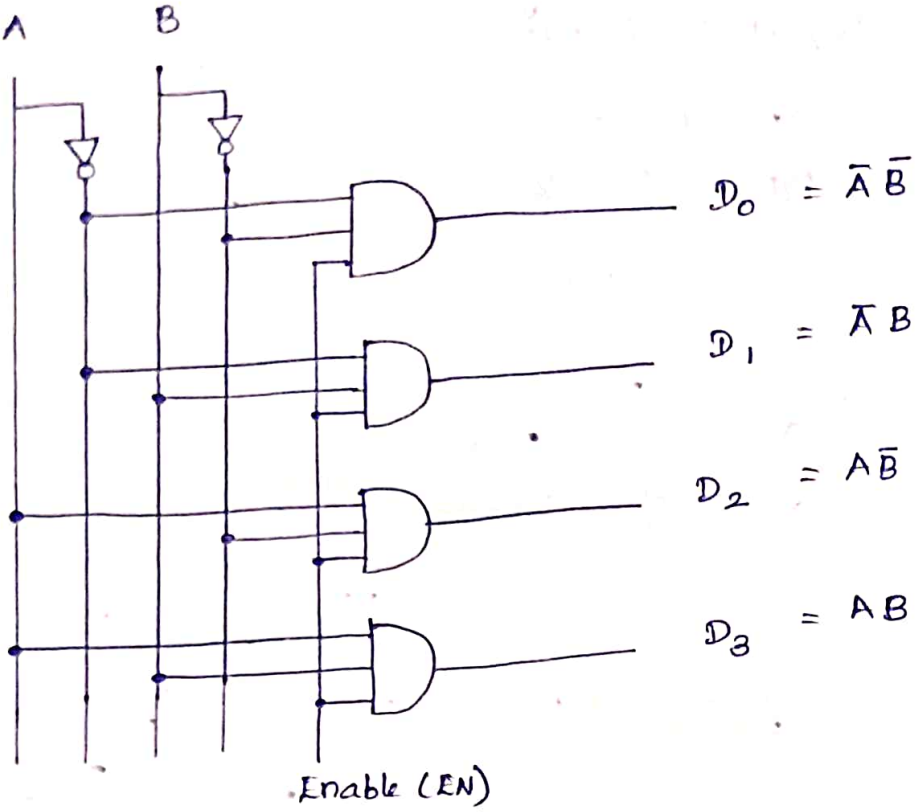
$n = 2 \rightarrow$ i/p
$2^n = 2^2 = 4 \rightarrow$ o/p

Block diagram.



* 2 inputs are decoded into 4 outputs
  - output representing one of the minterms of the two input variables.

2 to 4 line decoder circuit



* n bit binary input and a one activated output out of 2^n outputs.

Truth Table:

Inputs			outputs			
Enable	A	B	D_3	D_2	D_1	D_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

* Enable input = 1, only one of the outputs is active for a given input

* D_0 is active, when i/ps A = 0, B = 0
* D_1 is active, when A = 0, B = 1
* D_2 is active, when A = 1, B = 0
* D_3 is active, when A = 1, B = 1.

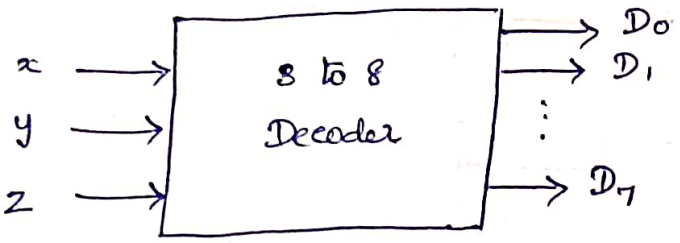
② 3 to 8 Line Decoder:

* The three inputs are decoded into eight outputs, each representing one of the minterms of the three input variables.

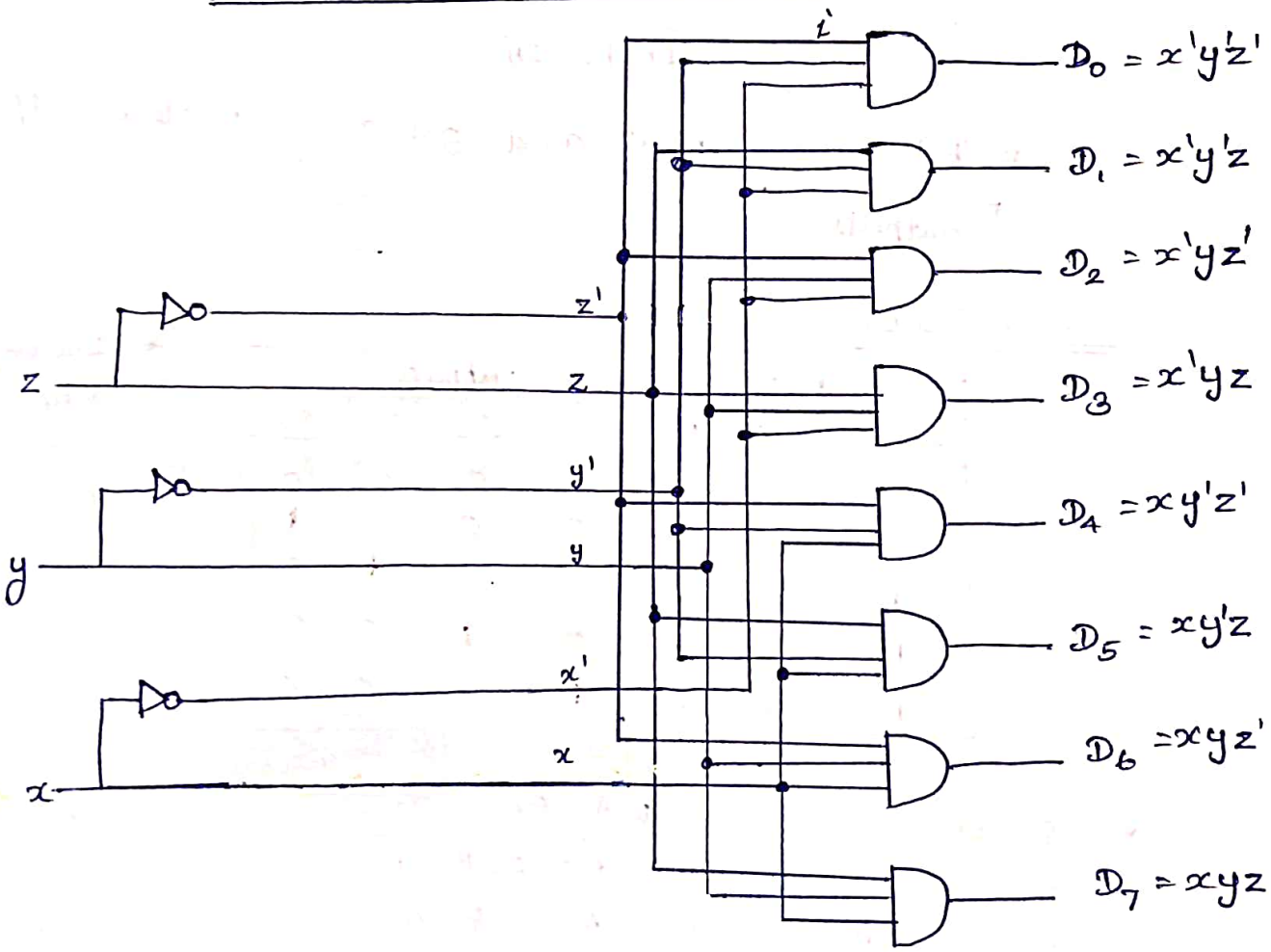
i/p → $n = 3$

o/p → $2^n = 2^3 = 8$

Block Diagram



3-to-8 line decoder circuit



* 3 inverters provide the complement of the inputs.
* Each one of the eight AND gates generates one of the minterms

Application:

* Binary - to - octal conversion

Truth Table:

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

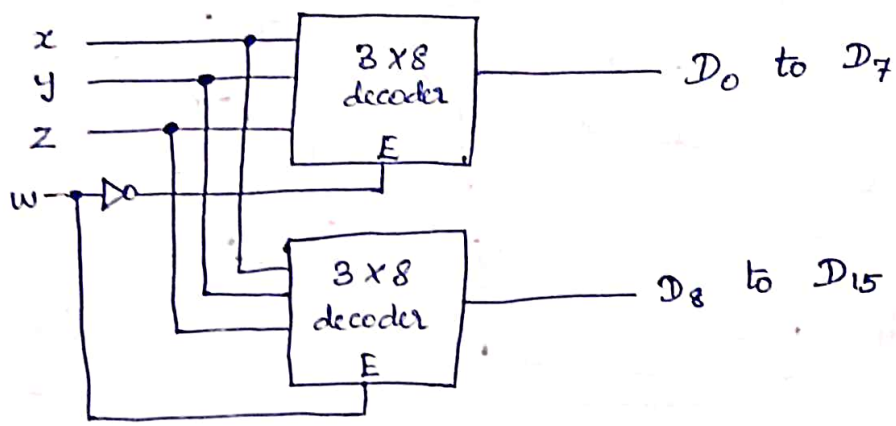
* For each possible input combination, there are seven outputs that are equal to 0 and only one that is equal to 1.

- The output whose value is equal to 1, represents the minterm equivalent of the binary number presently available in the input lines.

---

③ 4x16 Decoder constructed with Two 3x8 Decoders:

* Decoders with enable inputs can be connected together to form a larger decoder circuit.



$w = 0$, → top decoder is enabled and the other is disabled.

→ The bottom decoder outputs are all 0's

→ top eight outputs generate minterms 0000 to 1111

$w = 1$, → the enable conditions are reversed.

→ the bottom decoder outputs generate minterms 1000 to 1111.

→ the outputs of the top decoders are all 0's

Combinational Logic Implementation:

* Implement a combinational circuit by means of a
  * decoder
  * OR gates.
  - Boolean function for the circuit is expressed in sum of minterms.
* A decoder generates all the minterms of the input variables.
* The inputs to each OR gate are selected from the decoder outputs according to the list of minterm of each function.

Example: Implement a full adder circuit.

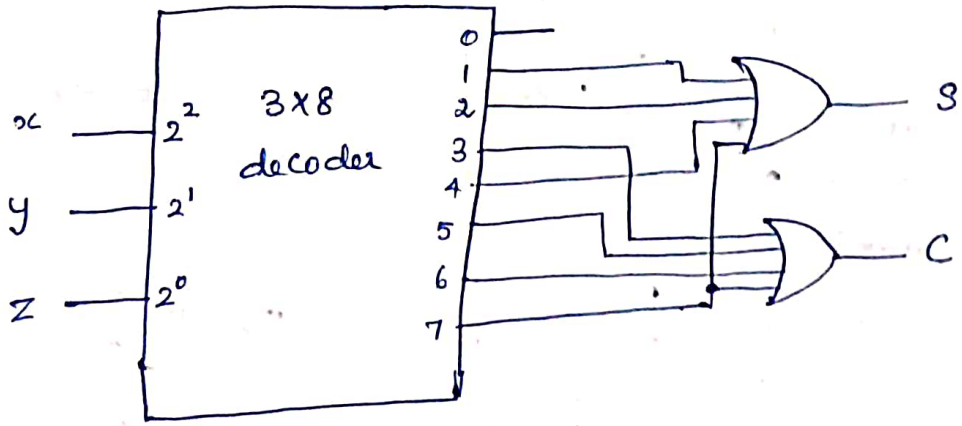
Soln: **Step 1:** Obtain the functions for the combinational circuit in sum of minterms.

$$S(x,y,z) = \Sigma(1,2,4,7)$$

$$C(x,y,z) = \Sigma(3,5,6,7)$$

Step 2: 3 i/ps, 8 minterms, need a 3-to-8 line decoder

Implementation of a full adder with a Decoder.



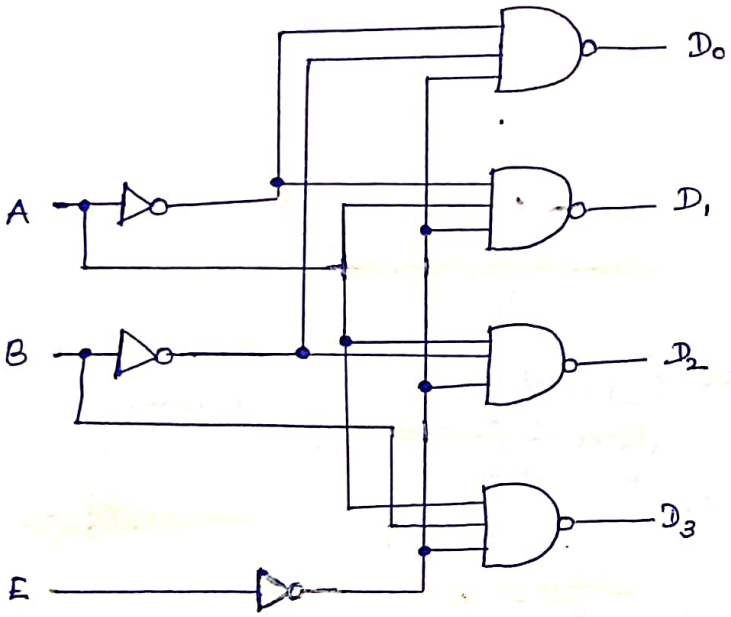
Using NAND gates:

* Some decoders are constructed with NAND gates.
- a NAND gate produces the AND operation with an inverted output.
  - it becomes more economical to generate the decoder minterms in their complemented form.
* decoders include one or more enable inputs to control the circuit operation.

Example: 2-to-4 line Decoder with Enable input.

- constructed with NAND gates.

Logic Diagram



* The circuit operates with complemented outputs and complement Enable input.

Truth Table

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

* The decoder is enabled when E = 0.
- only one output can be equal to 0 at any given time, all other outputs are equal to 1.
* 0 → minterm selected by A & B

* The circuit is disabled when E = 1, none of the o/ps = 0, none of the minterms are selected.

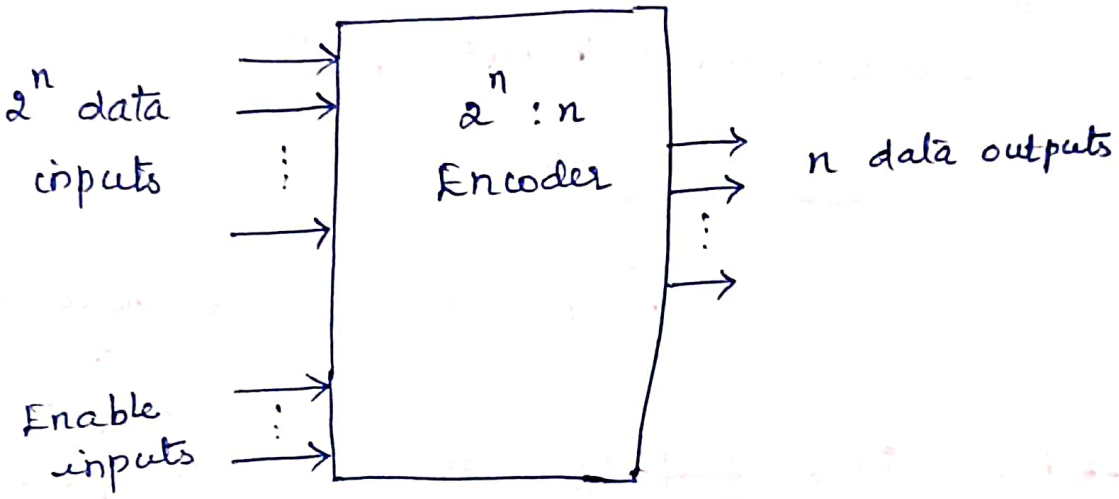
2.8 Encoders

* An encoder is a digital circuit that performs the inverse operation of a decoder.

→ An encoder has 2^n input lines and n output lines.

- The output lines generate the binary code corresponding to the input value.

General Structure



Types:

i) 4 to 2 Encoder
ii) 8 to 3 Encoder (octal-to-binary encoder)
iii) 16 to 4 Encoder

Example:

Octal-to-binary encoder / 8-to-3 Encoder

Step 1:

* Eight inputs and three outputs that generate the corresponding binary number.
- only one input has a value of 1 at any given time.

Step 2:

* Derive the truth table.

Truth Table of 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

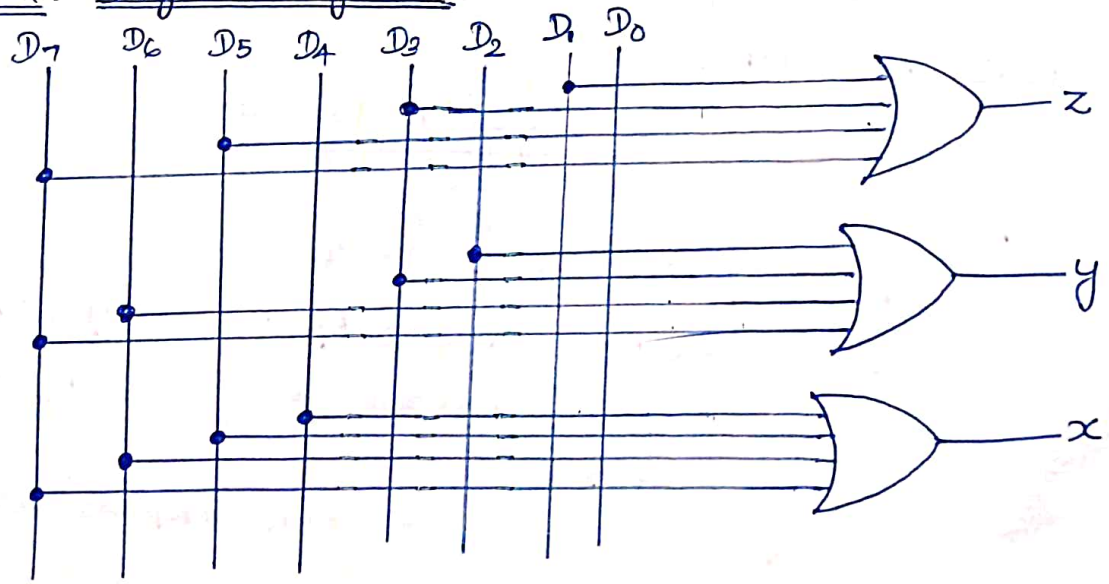
Step 3: Boolean functions

output z = 1, when input octal digit is 1, 3, 5, 7
y = 1, when input octal digit is 2, 3, 6, 7
x = 1, for digits 4, 5, 6, 7

*The conditions can be expressed by the output Boolean functions.

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

Step 4: Logic diagram:



Limitations & solution.

i) only one input can be active at any given time

* If two inputs are active simultaneously, the output produces undefined combination.

Ex: $D_3, D_6 \rightarrow 1$

$O/P \rightarrow 111$ ✗ does not represent 3 or 6

Soln:

→ Encoder circuits must establish an input priority to ensure that only one input is encoded.

ii) an output with all 0's is generated when all the inputs are 0.

– this O/p is same as when $D_0 = 1$

Soln:

→ Provide one more output to indicate that at least one input is equal to 1.

Priority Encoder:

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

4 - Input Priority Encoder

Step 1:

* 4 inputs D_0, D_1, D_2, D_3
* In addition to 2 outputs x & y, the circuit has a third output designated by V.

V → valid bit

V = 1, when one or more inputs are equal to 1.

* All inputs are 0, no valid bit, V = 0

Step 2: Truth Table of a Priority Encoder

Inputs				outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

X → don't care conditions = 0/1

Ex: x100 → 0100, 1100

* Input D_3 has the highest priority.

$D_3 = 1$, output xy = 11

- higher the subscript number, higher the priority of the input.

* D_2 has the next priority level.

$D_2 = 1$, output = 10

K-map table

0000 → 0
1000 → 8

x100
↓
0100 → 4
1100 → 12

xx10
↓
0010 → 2
0110 → 6
1010 → 10
1110 → 14

xxx1
↓
0001 → 1
0011 → 3
0101 → 5
0111 → 7
1001 → 9
1011 → 11
1101 → 13
1111 → 15

y:

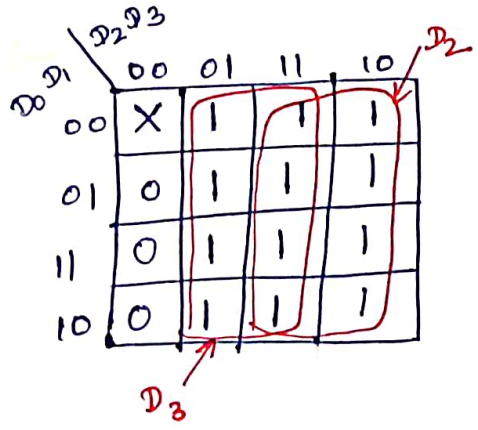
0000 — x
1000 — 0

x100 → 0100 – 4 → 1
1100 – 12 → 1

Step 3: Boolean functions.

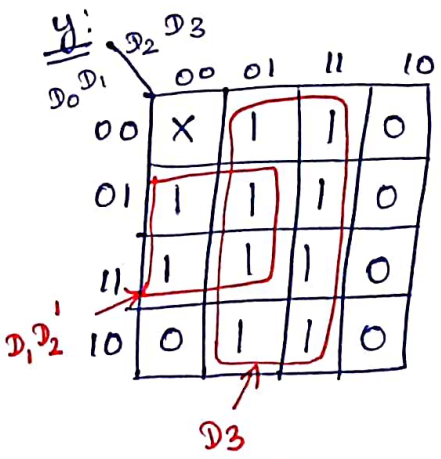
Maps for a Priority Encoder

x:



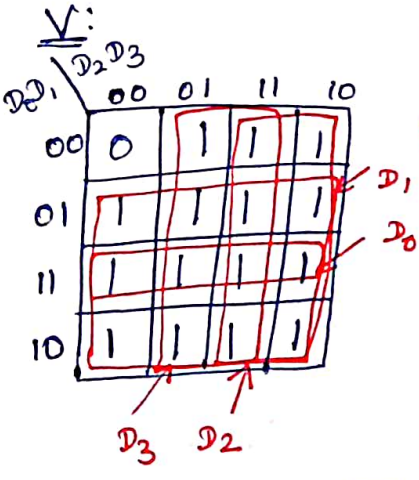
$x = D_2 + D_3$

y:



$y = D_3 + D_1 D_2'$

V:



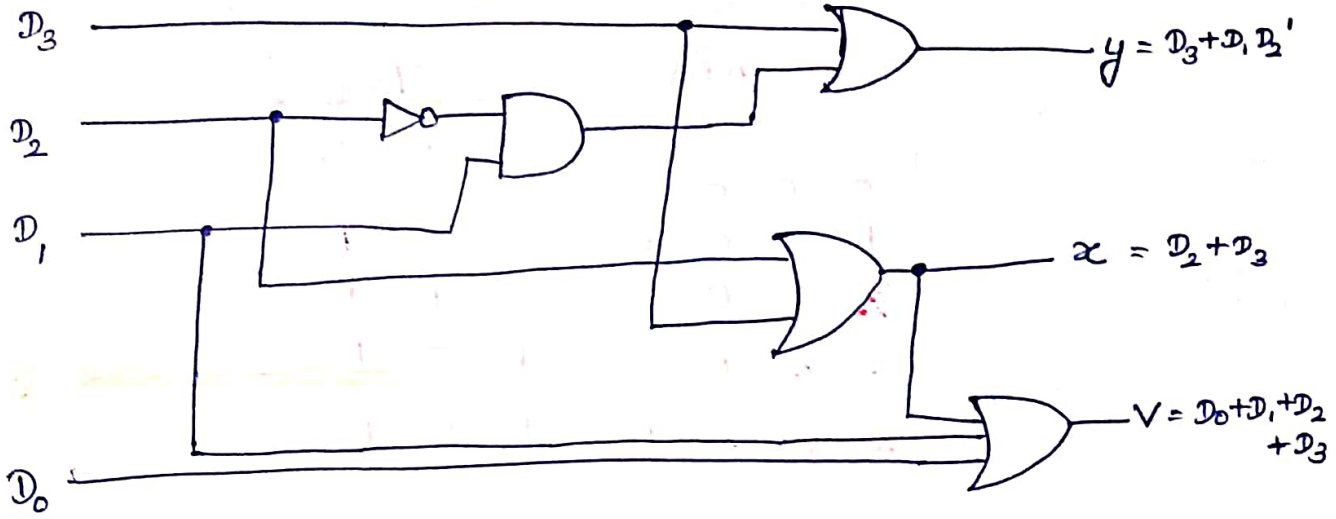
$V = D_0 + D_1 + D_2 + D_3$

(or)

from truth table $D_0 = 1, D_1 = 1, D_2 = 1, D_3 = 1$

So $V = D_0 + D_1 + D_2 + D_3$

Step 4: Logic Diagram.



* Condition for output V is an OR function of all the input variables.

Multiplexers (2^n to 1)

* A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.

- The selection of a particular input line is controlled by a set of selection lines.

* 2^n input lines and n selection lines whose bit combinations determine which input is selected.

Types:

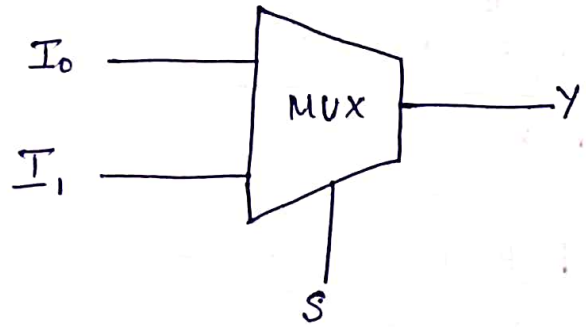
i) 2 to 1 MUX
ii) 4 to 1 MUX
iii) 8 to 1 MUX
iv) 16 to 1 MUX
v) 32 to 1 MUX. etc.

* Multiplexers - also called as Data selector / Many-to-one converter / MUX.

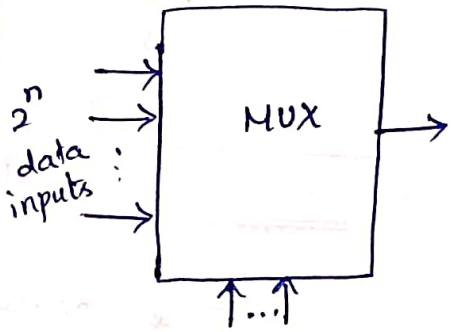
- selects one of many inputs and
- steers the binary information to the output line

Block diagram

2-to-1 MUX.



General Structure



① 2-to-1 line Multiplexer:

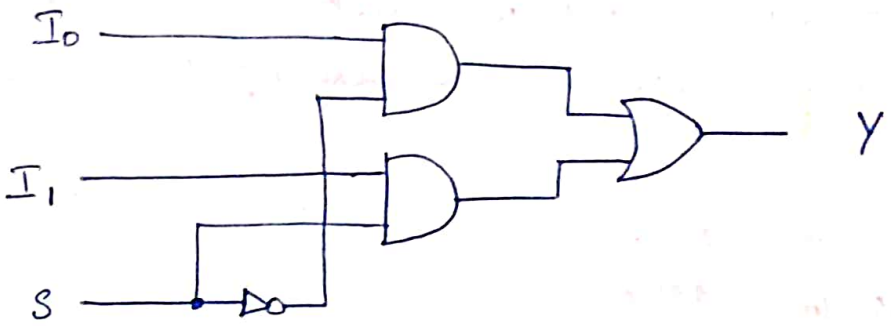
Step 1:

* The circuit has two data input lines
  - one output line
  - one selection line

Step 2 : Function Table

S	Y
0	I_0
1	I_1

Step 3 : Logic diagram



* When $S=0$, the upper AND gate is enabled and I_0 has a path to the output.

* When $S=1$, the lower AND gate is enabled and I_1 has a path to the output.

→ The multiplexer acts like an electronic switch that selects one of two sources

– depicted using a wedge-shaped symbol.

↳ selected one of multiple data sources is directed into a single destination.

---

② 4-to-1 line Multiplexer :

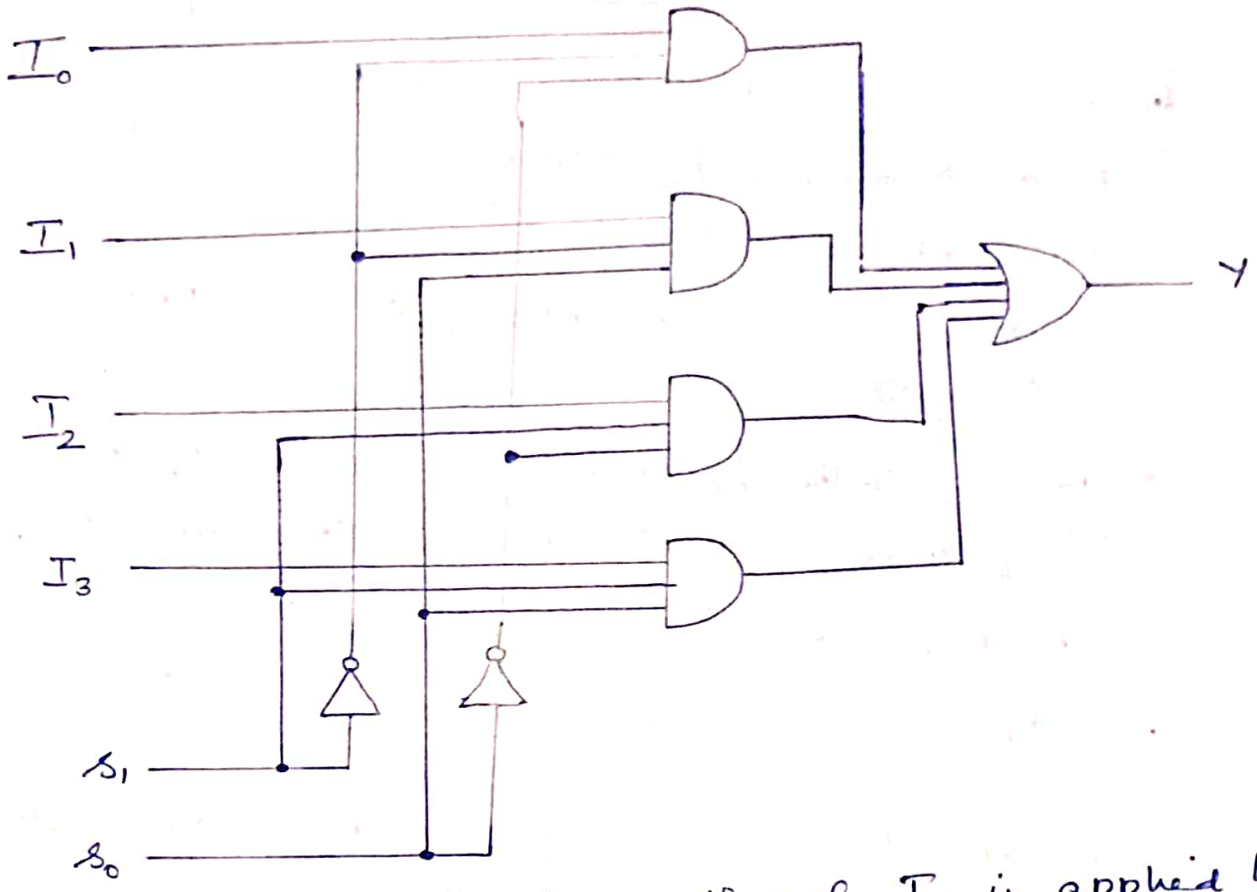
Step 1 :

* four inputs I_0, I_1, I_2, I_3
* Selection lines S_1 & S_0
* one output line

Step 2 : Function Table :

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Step 3 : Logic Diagram



* Each of the four inputs I_0 through I_3 is applied to one input of an AND gate
- Selection lines S_1 and S_0 are decoded to select a particular AND gate.
- The outputs of the AND gates are applied to a single OR gate that provides the 1-line output.

$S_1S_0 = 10$

* The AND gate associated with input I_2 has two of its inputs equal to 1 and the third input connected to I_2.
- The other three AND gates have at least one input equal to 0. , outputs equal to 0.
- The OR gate output is equal $= I_2$, providing a path from the selected input to the output.

Multiplexer from decoder:

* AND gates, inverters } resemble a decoder circuit
  - decode the selection input lines.

* 2^n-to-1 line multiplexer is constructed from an n to 2^n decoder by adding to it 2^n input lines, one to each AND gate.
  - The outputs of the AND gates are applied to a single OR gate.

* The size of the multiplexer is specified by the number 2^n of its data input lines and the single output line.
  → The n selection lines are implied from the 2^n data lines.

* Multiplexers may have an enable input to control the operation of the unit.
  → When the enable input is in inactive state,
    - the outputs are disabled
  → When it is in the active state,
    - the circuit functions as a normal multiplexer.

Quadruple 2-to-1 line Multiplexer

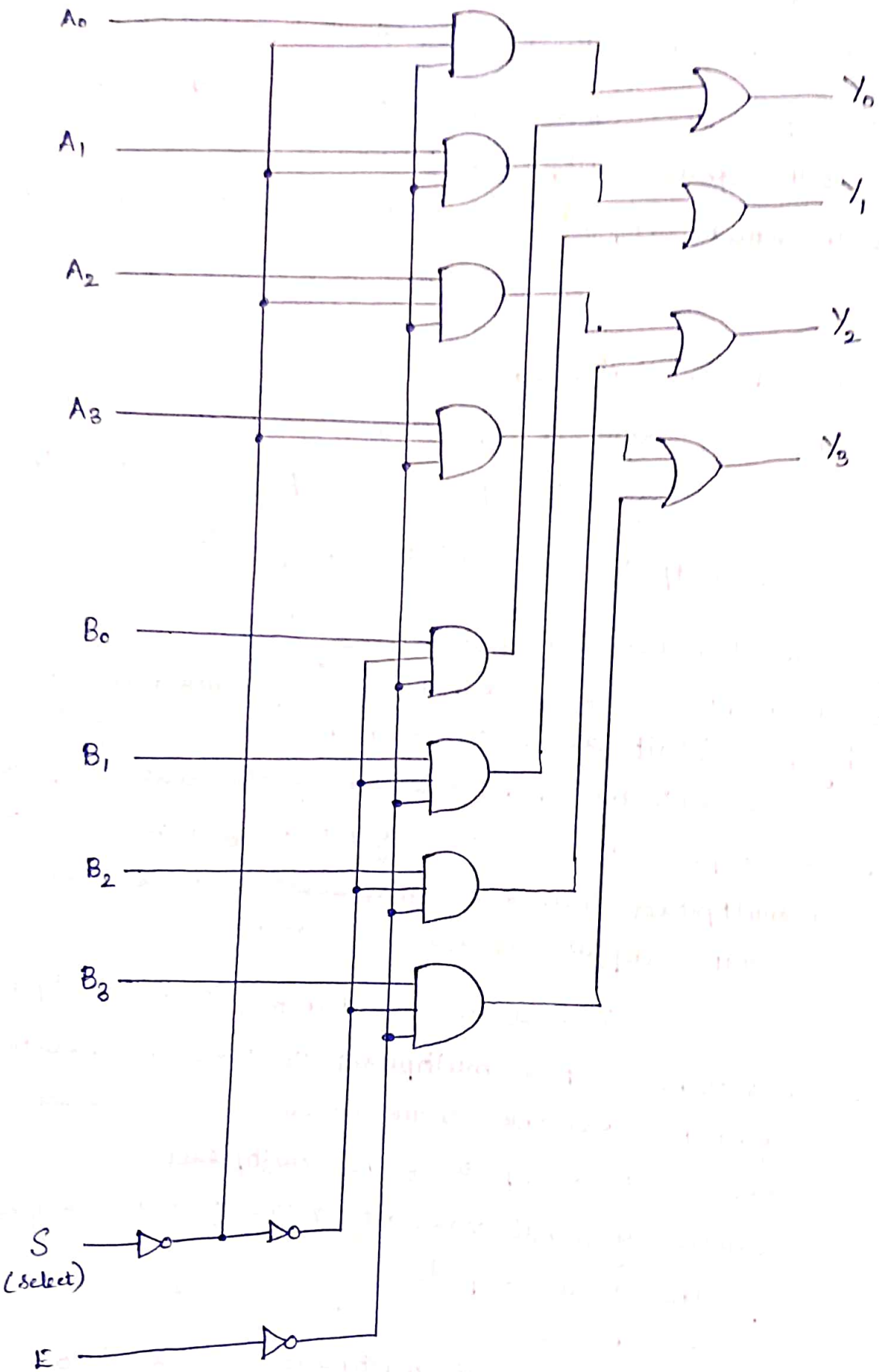
* Multiplexer circuits can be combined with common selection inputs to provide multiple-bit selection logic.

Step 1:
→ The circuit has 4 multiplexers
  - each capable of selecting one of two input lines.

Step 2: Function Table

E	S	output Y
1	×	all 0's
0	0	Select A
0	1	Select B

Step 3 : Logic diagram



* output Y_0 can be selected to come from either input A_0 or B_0.

* Y_1 may have the value of A_1 or B_1 and so on.

→ Input selection line S selects one of the lines in each of the four multiplexers.

→ The enable input E must be active for normal operation.

* The circuit selects one of two 4-bit sets of data lines

$E = 0$, the unit is enabled.

i) $S = 0$, four A inputs have a path to the four outputs.

ii) $S = 1$, four B inputs are applied to the outputs.

$E = 1$, the outputs have all 0's., regardless of the value of S.

Boolean Function Implementation:

* The minterms of a function are generated in a multiplexer by the circuit associated with the selection inputs.
  - minterms can be selected by the data inputs.

* Can implement a Boolean function of n variables with a multiplexer that has n selection inputs and 2^n data inputs, one for each minterm.

Implementing a Boolean function with a Multiplexer

→ n variables with a multiplexer that has n−1 selection inputs
- first n−1 variables of the function are connected to the selection inputs of the multiplexer.
- Remaining single variable of the function is used for the data inputs.
  * Variable-Z
  * data input of the multiplexer → Z, Z', 1, 0

Example : 1 :

Implement the Boolean function using multiplexer.

$$F(x, y, z) = \sum (1, 2, 6, 7)$$

Soln :

* The function can be implemented with a 4-to-1 line multiplexer.

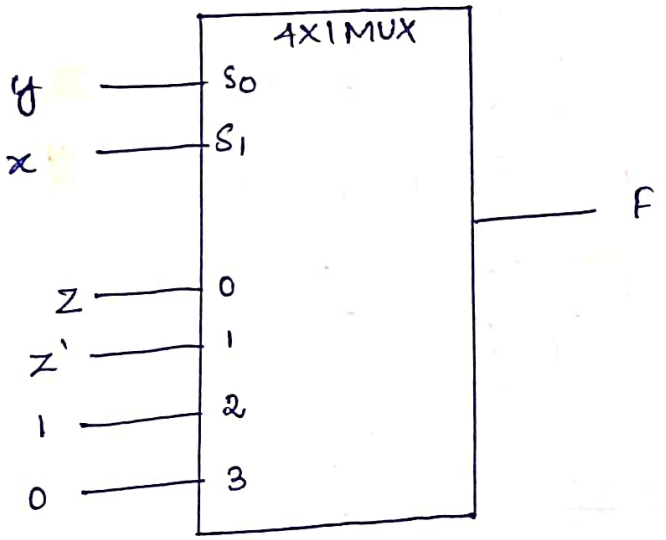
Step 1 : Truth Table

x	y	z	F	
0	0	0	0	F = z
0	0	1	1	
0	1	0	1	F = z'
0	1	1	0	
1	0	0	0	F = 0
1	0	1	0	
1	1	0	1	F = 1
1	1	1	1	

xy = 00, F = z
F = 0, z = 0
F = 1, z = 1

Step 2 : Multiplexer Implementation

x, y } selection lines



Procedure : for implementing any Boolean functions of n variables with a multiplexer with n-1 selection inputs and 2^{n-1} data inputs.

* Listed in a truth table
* first n-1 variables are applied to the selection inputs of the multiplexer.

* For each combination of the selection variables, evaluate the output as a function of the last variable
  - The function can be
    * 0
    * 1
    * Variable
    * complement of the variable
  - The values are applied to the data inputs in the proper order.

Example 2:

Implement the Boolean function with a MUX.

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

Soln:

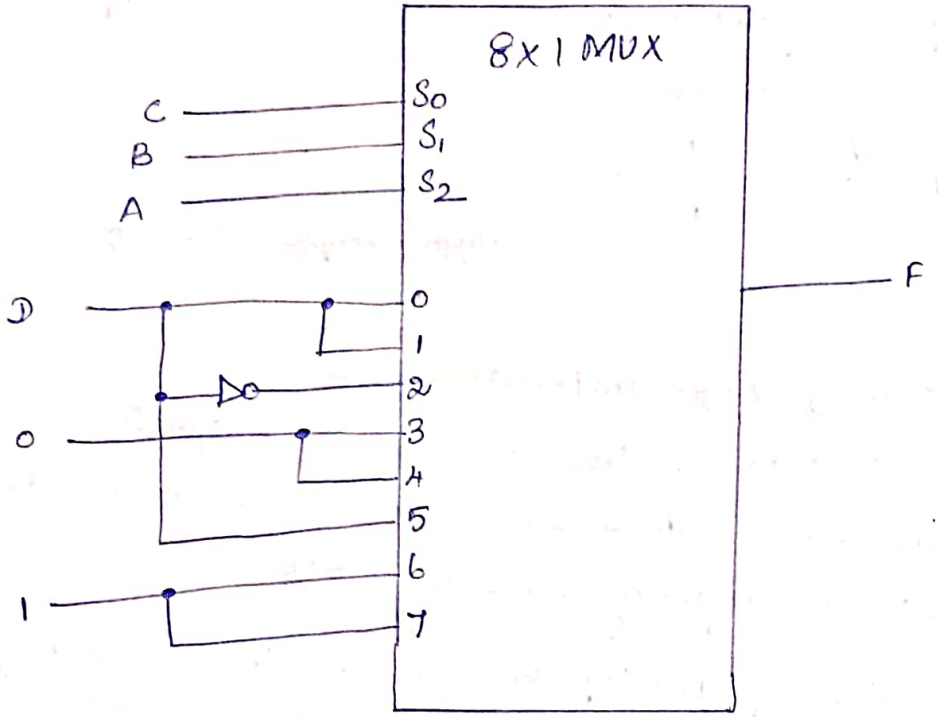
n-1 selection inputs → 3 selection inputs – A, B, C

Step 1: List the Truth Table

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	

Step 2

Implementing a 4-Input function with a Multiplexer.



* The values for the data inputs are determined from the truth table.
* The corresponding data line number is determined from the binary combination of ABC.

EX: ABC = 101

F = D, the input variable D is applied to data input 5

0, 1 → 2 fixed signal values.

In Integrated circuits,

logic 0 → signal ground.

logic 1 → power signal – (5 volts).

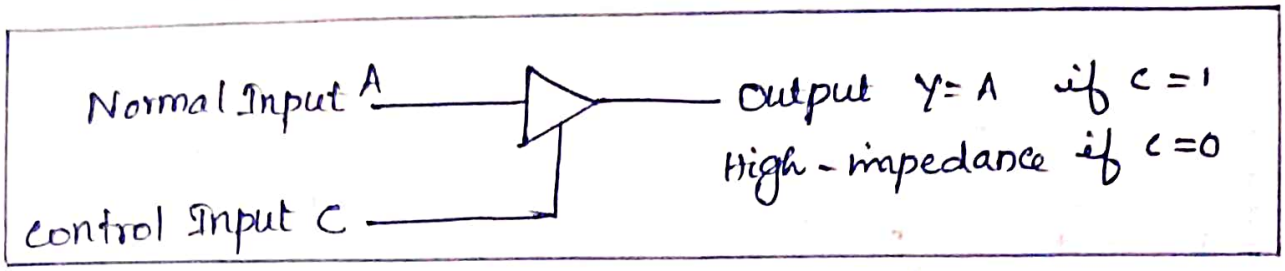
Three-State Gates:

* A multiplexer can be constructed with three-state gates.

→ A three-state gate is a digital circuit that exhibits three states: logic 0, logic 1, high-impedance state

High impedance state → an open circuit behaviour
- the output appears to be disconnected and the circuit has no logic significance

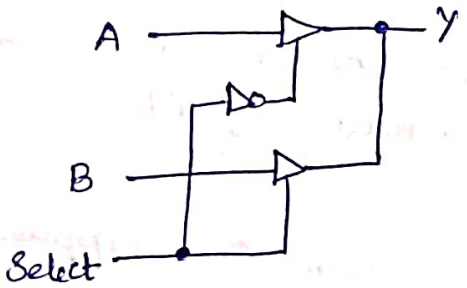
Graphic Symbol for a Three-State Buffer



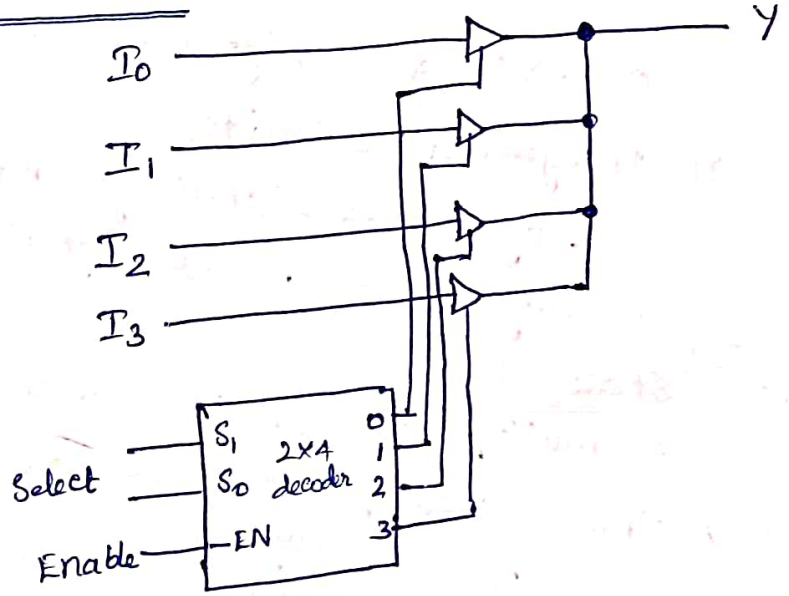
* Special feature of high impedance state.
→ a large number of three-state gate outputs can be connected with wires to form a common line without endangering loading effects.

construction of multiplexers with three-state buffers.

a) 2-to-1 line mux



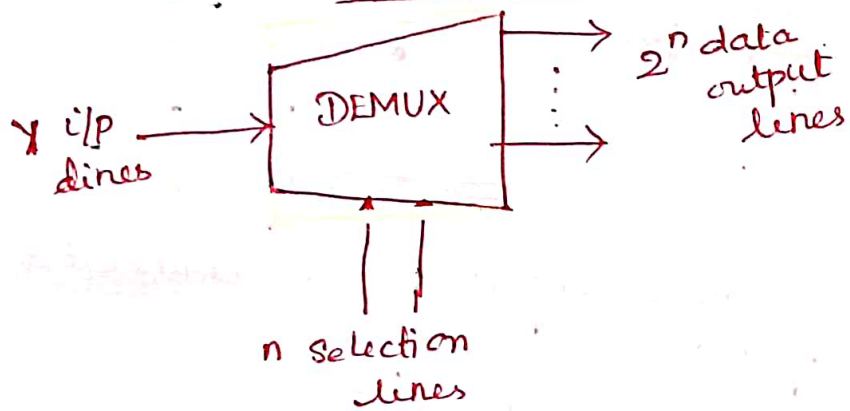
b) 4-to-1 line mux.



DEMULTIPLEXERS

* A demultiplexer is a circuit that receives information on a single line and transmits this information on one of 2^n possible output lines.
- The selection of a specific output line is controlled by the bit values of n selection lines.
- single-input, multiple-output circuit
* A decoder with an enable input can function as a demultiplexer.

Block Diagram

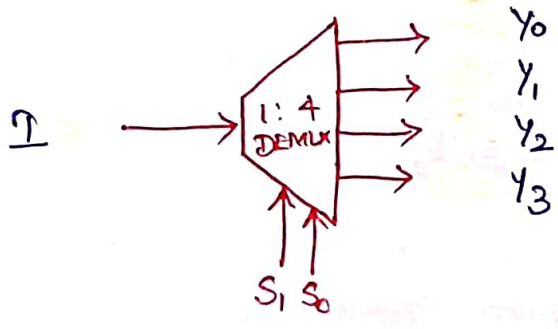


Types:

i) 1 to 2 demux
ii) 1 to 4 demux
iii) 1 to 8 demux. etc.

1 to 4 De-multiplexer.

Block Diagram

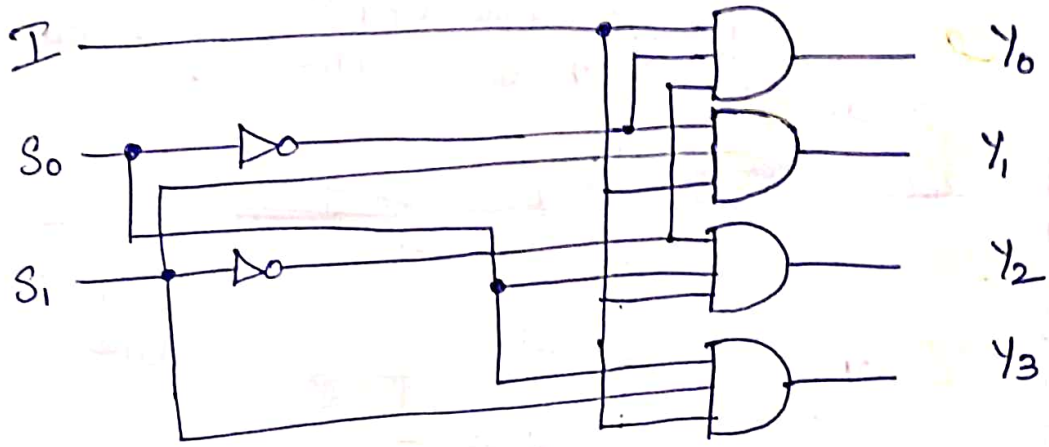


Truth Table

S_0	S_1	Y_0	Y_1	Y_2	Y_3
0	0	I	0	0	0
0	1	0	I	0	0
1	0	0	0	I	0
1	1	0	0	0	I

$Y_0 = \bar{S}_0 \bar{S}_1 I$
$Y_1 = \bar{S}_0 S_1 I$
$Y_2 = S_0 \bar{S}_1 I$
$Y_3 = S_0 S_1 I$

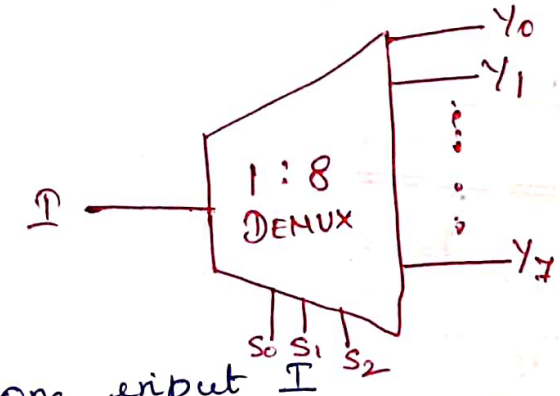
Logic diagram



* A single input I is connected to one of four outputs based on the values of selection lines
* Y_0 to Y_3
  S_0 & S_1.

1 to 8 Demultiplexer

Block Diagram

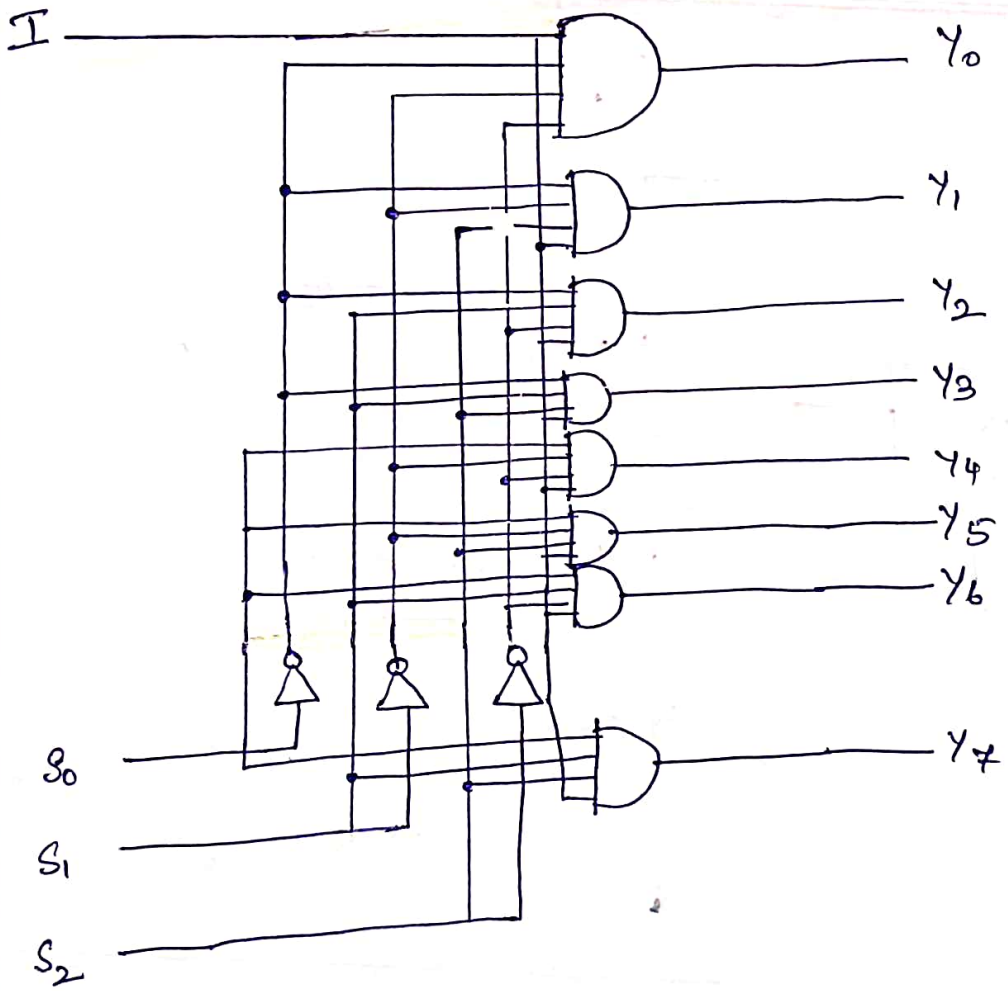


* One input I
* Three selection lines S_0, S_1, S_2
* Outputs from Y_0 to Y_7

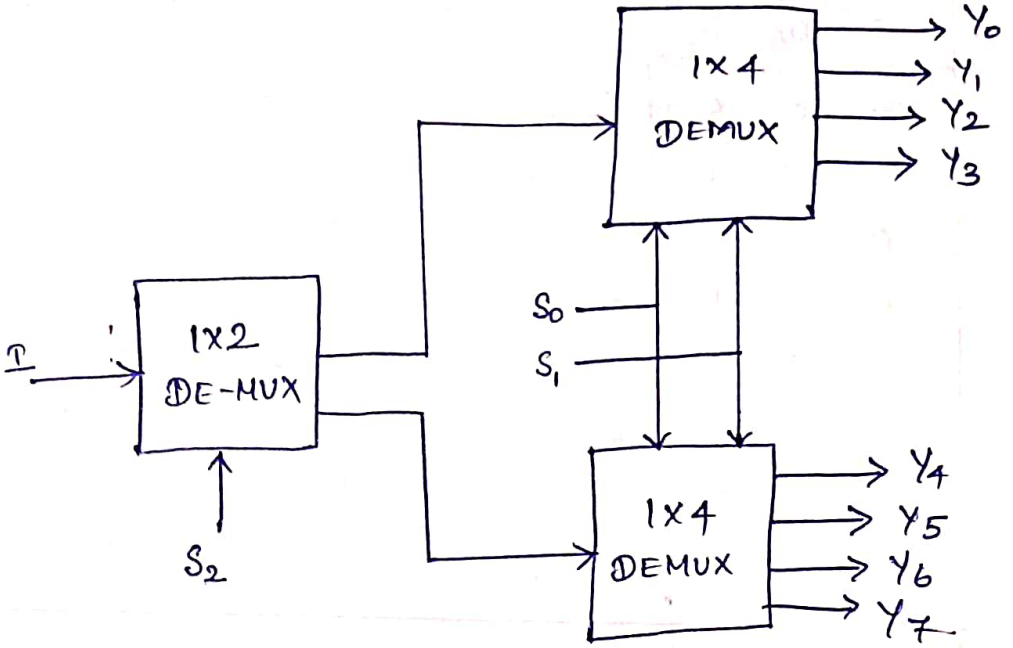
Truth Table:

S_0	S_1	S_2	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	I	0	0	0	0	0	0	0
0	0	1	0	I	0	0	0	0	0	0
0	1	0	0	0	I	0	0	0	0	0
0	1	1	0	0	0	I	0	0	0	0
1	0	0	0	0	0	0	I	0	0	0
1	0	1	0	0	0	0	0	I	0	0
1	1	0	0	0	0	0	0	0	I	0
1	1	1	0	0	0	0	0	0	0	I

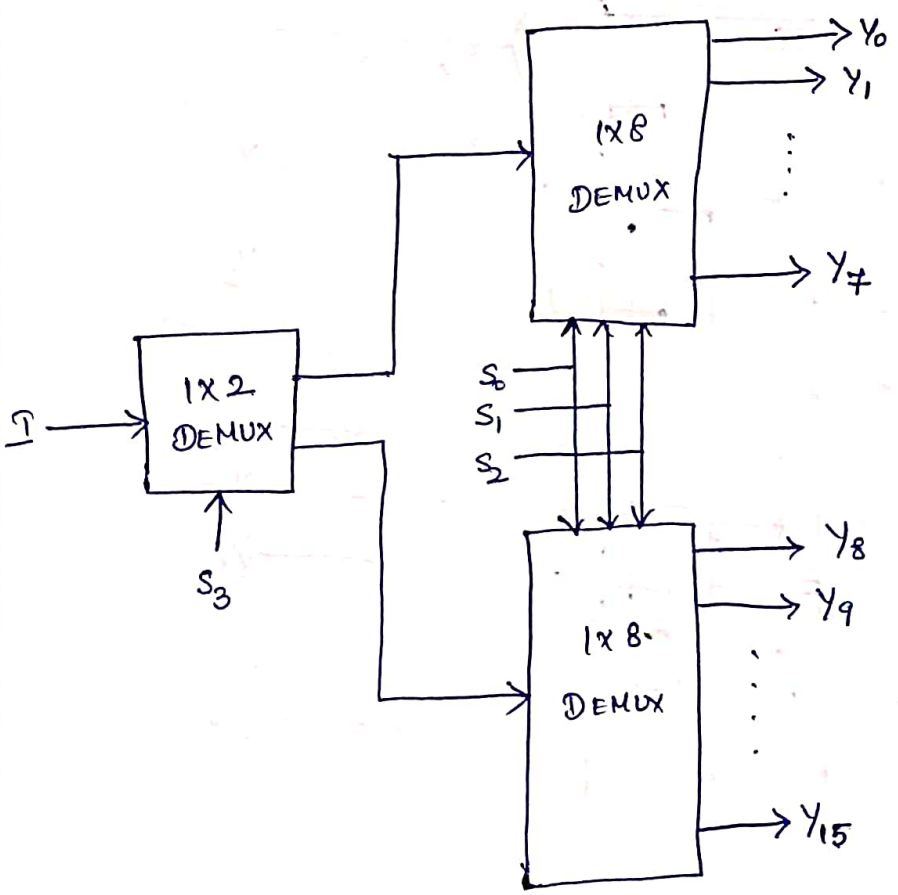
Logic Diagram.



1x8 DEMUX using one 1x2 DEMUX and Two 1x4 DEMUX

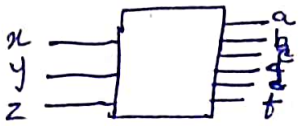


1x16 DEMUX using one 1x2 DEMUX and Two 1x8 DEMUX



Design a combinational circuit that accepts a 3-bit number and generates a 6-bit binary number output equal to the square of the input number. Write a high-level behavior VHDL description of the circuit.

Soln:



Step 1:

Number of inputs → 3 – x, y, z

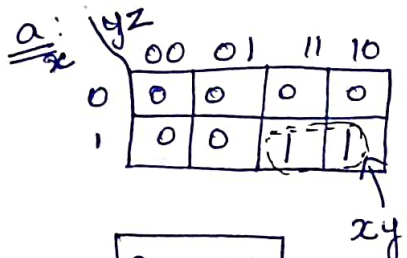
Number of outputs → 6 – a, b, c, d, e, f

Step 2: Truth Table

i/p			o/p					
x	y	z	a	b	c	d	e	f
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	0	0	1
1	0	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0	1
1	1	0	1	0	0	1	0	0
1	1	1	1	1	0	0	0	1

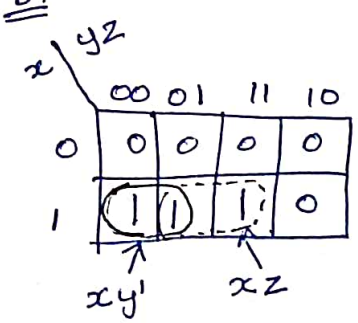
Step 3: Boolean expressions using K-map.

a:



$a = xy$

b:



$b = xy' + xz$

$b = x(y' + z)$

c:

x \ yz	00	01	11	10
0			1	
1		①		

$c = xy'z + x'yz$

$c = z(x \oplus y)$

d:

x \ yz	00	01	11	10
0	0	0	0	1
1	0	0	0	1

$d = yz'$

e:

x \ yz	00	01	11	10
0	0	0	0	0
1	0	0	0	0

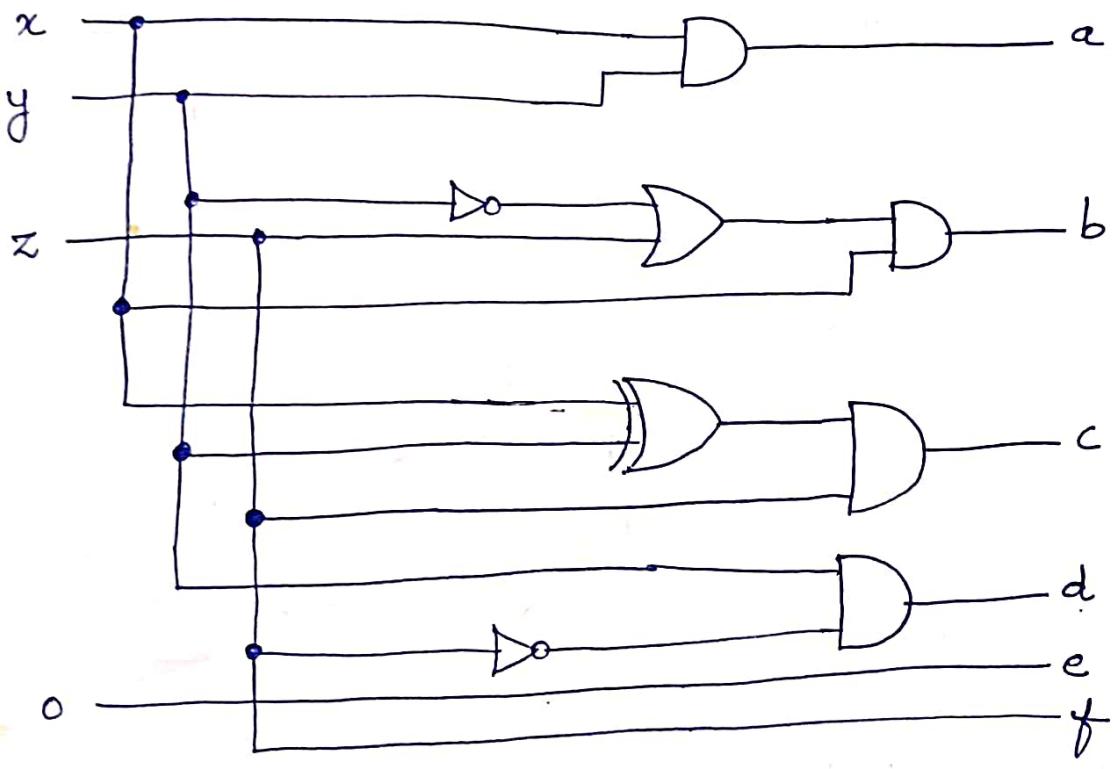
$e = 0$

f:

x \ yz	00	01	11	10
0		1	1	
1		1	1	

$f = z$

Step 4: Logic diagram:



Design a Binary to Gray code Converter.

Soln:

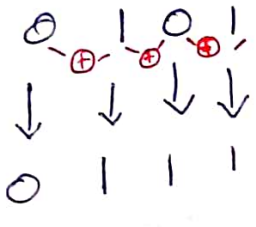
Step 1: Number of inputs → 4 – B_3, B_2, B_1, B_0

Number of outputs → 4 – G_3, G_2, G_1, G_0

Step 2: Derive the Truth Table

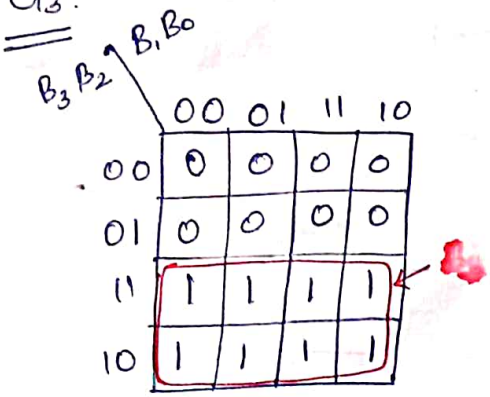
Binary				Gray			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Binary to Gray code



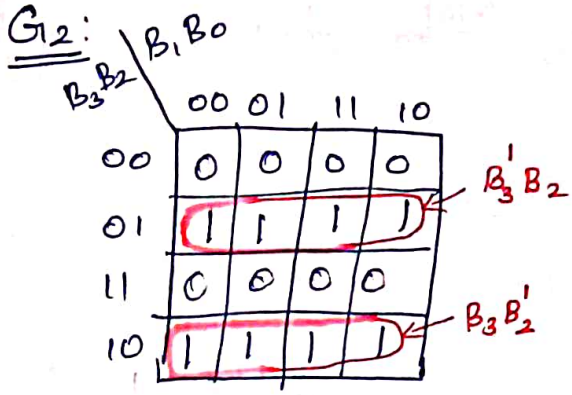
Step 3: Find the simplified boolean expressions using K-map.

G_3:



$G_3 = B_3$

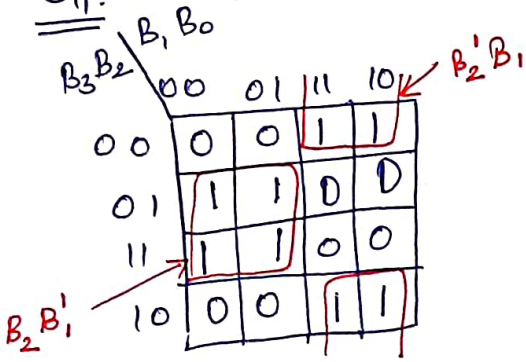
G_2:



$G_2 = B_3' B_2 + B_3 B_2'$

$G_2 = B_3 \oplus B_2$

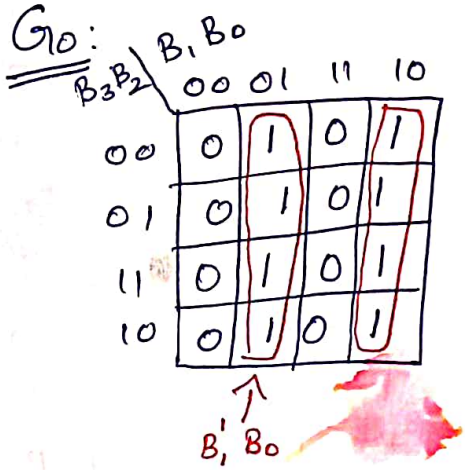
G_1:



$G_1 = B_2 B_1' + B_2' B_1$

$G_1 = B_2 \oplus B_1$

G_0:



$G_0 = B_1' B_0 + B_1 B_0'$

$G_0 = B_1 \oplus B_0$

Step 4: Logic diagram