MODULE III

COMBINATIONAL LOGIC AND CIRCUITS

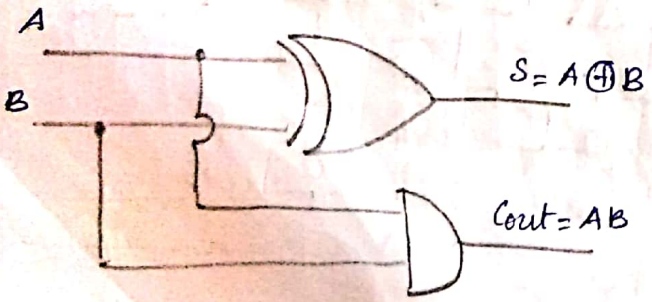
1. Half adder

A	B	S	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

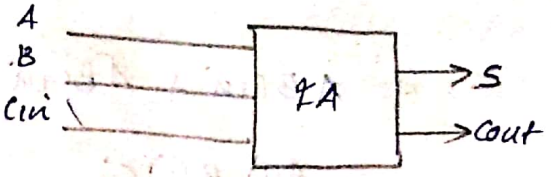
Circuit

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

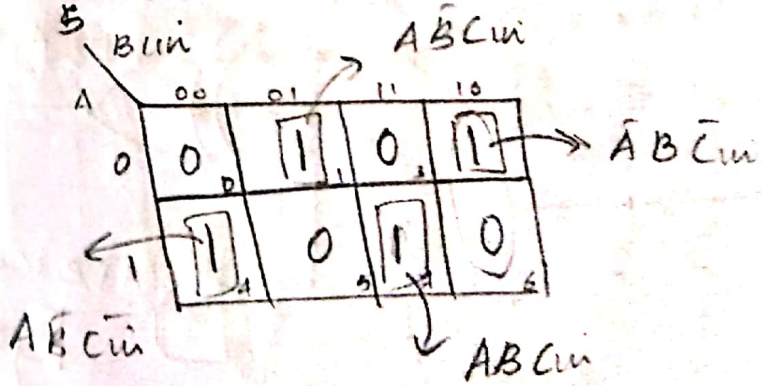
$Cout = AB$



2. Full adder



A	B	Cin	S	Cout
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



$S = A\bar{B}C_{in} + \bar{A}BC_{in} + ABC_{in} + A\bar{B}\bar{C}_{in}$

$= A \oplus B \oplus C_{in}$

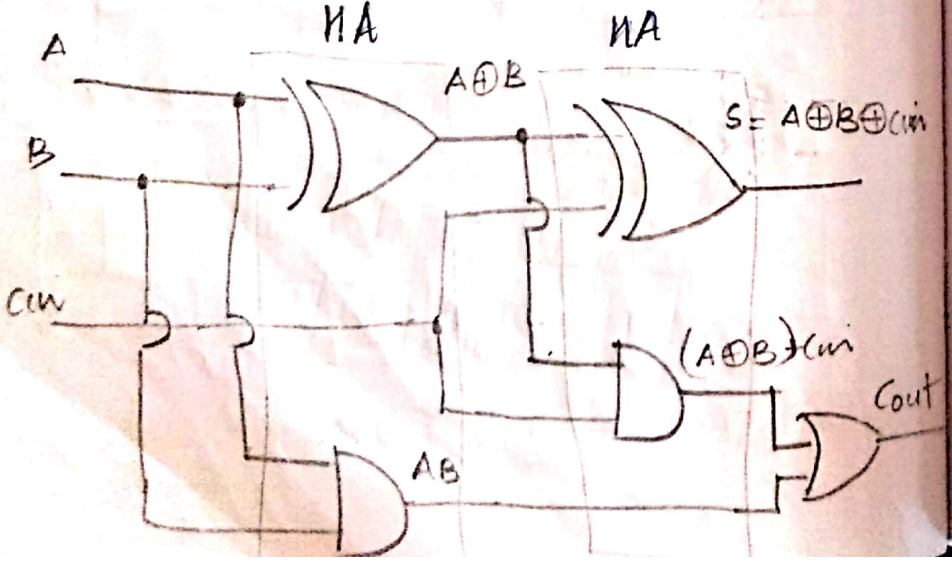
$C = \bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in}$

$= C_{in}(\bar{A}B + A\bar{B}) + AB(\bar{C}_{in} + C_{in})$

$= C_{in}(\bar{A}B + A\bar{B}) + AB$

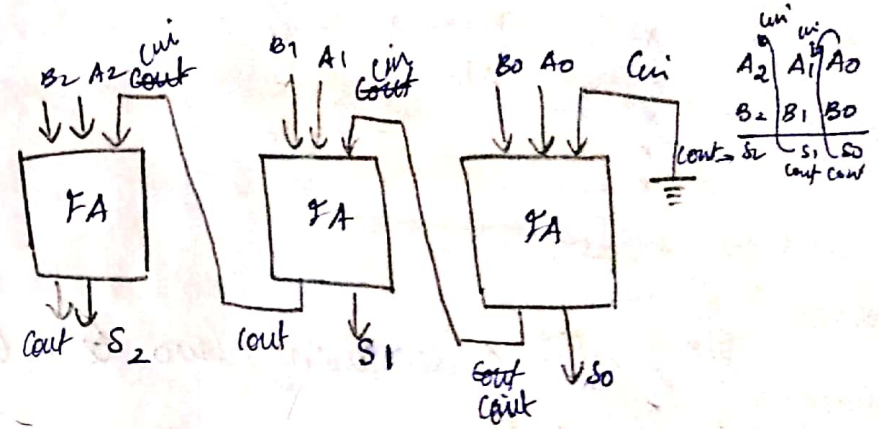
$= C_{in}(A \oplus B) + AB$

Circuit

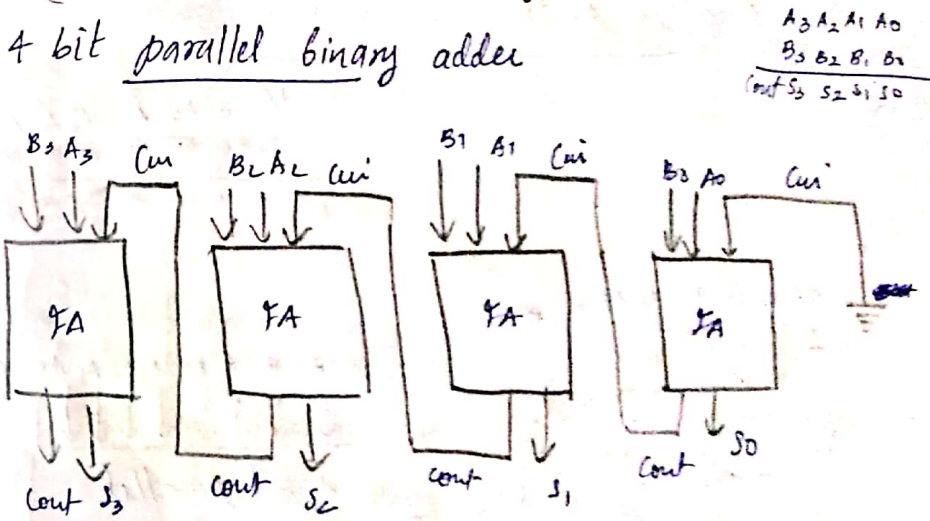


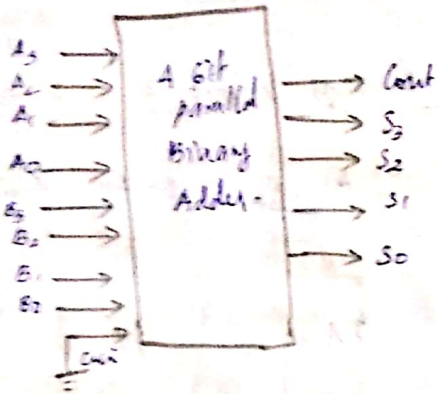
3. Parallel Binary Adder / Parallel Full Adder / Triple carry adder

3 bit parallel binary adder

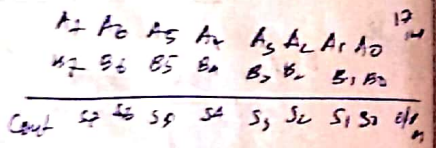


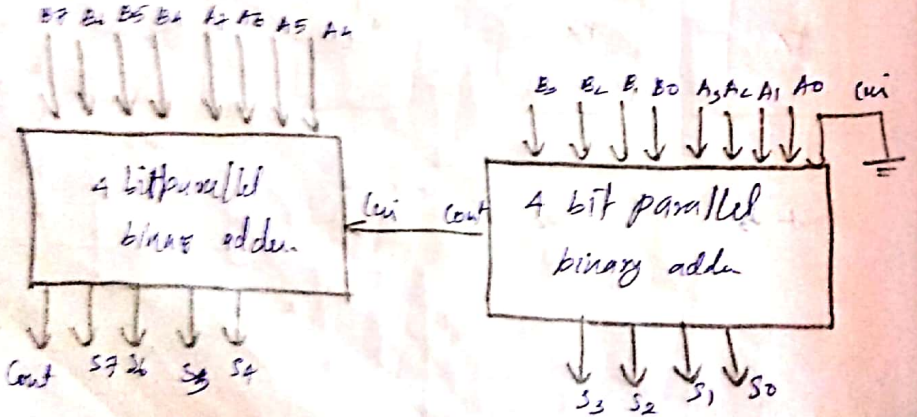
4 bit parallel binary adder





8 bit parallel Adder using two 4 bit parallel Adder





Ripple Carry Adder

A parallel adder in which the carry out of each full adder is the carry in to the next most significant full adder is called a ripple carry adder

Sum and output carry of any stage cannot be produced until the input carry occurs. This causes a time delay in addition process. The greater the no. of bits that a ripple carry adder must add, the greater the time required for it perform a valid addition

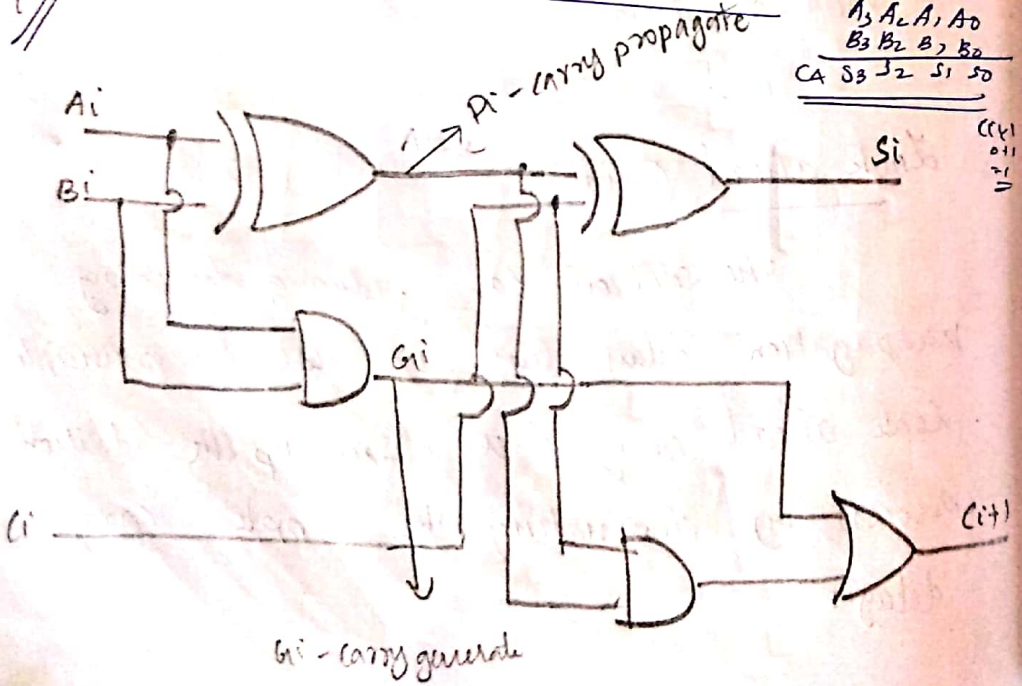
Look ahead Principle

The solution for reducing the carry propagation delay time is to use the principle Look ahead carry adder to speed up the addition process by eliminating this ripple carry delay.

It examines all the i/p bits simultaneously and also generates the carry-in bits for all the stages simultaneously. The method of speeding up the addition process is based on two additional functions of the full adder called carry generate (G_i) and carry propagate (P_i) functions.

4MP

Four Bit Look ahead carry Adder / Generator



$P_i = A_i \oplus B_i$ $\quad S_i = P_i \oplus C_i$

$G_i = A_i B_i$ $\quad C_{i+1} = P_i C_i + G_i$
$= G_i + P_i C_i$

$C_1 = G_0 + P_0 C_0$

$C_2 = G_1 + P_1 C_1$

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

$= G_1 + P_1 G_0 + P_1 P_0 C_0$

$C_3 = G_2 + P_2 C_2$

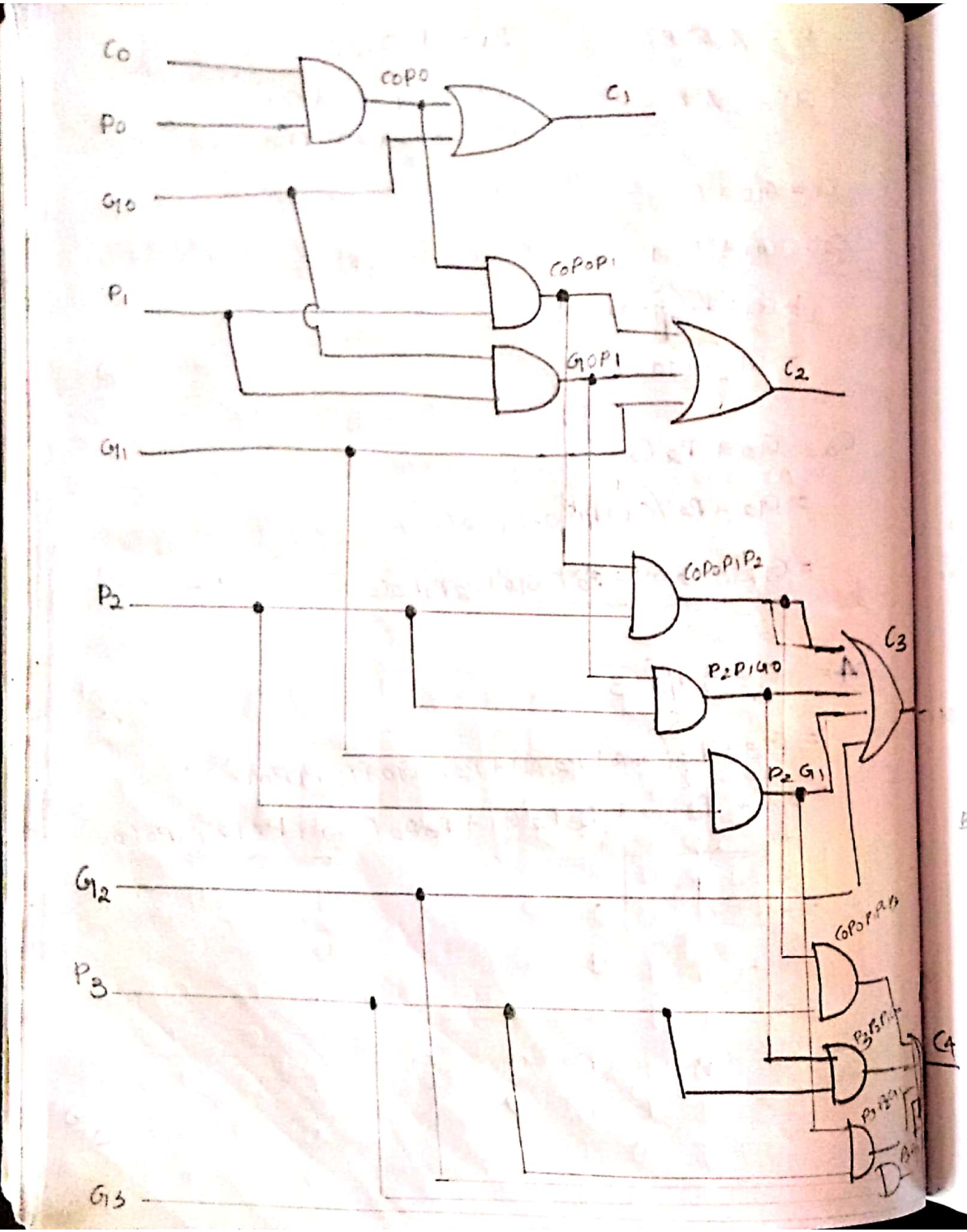
$= G_2 + P_2 (G_1 + P_1 G_0 + P_1 P_0 C_0)$

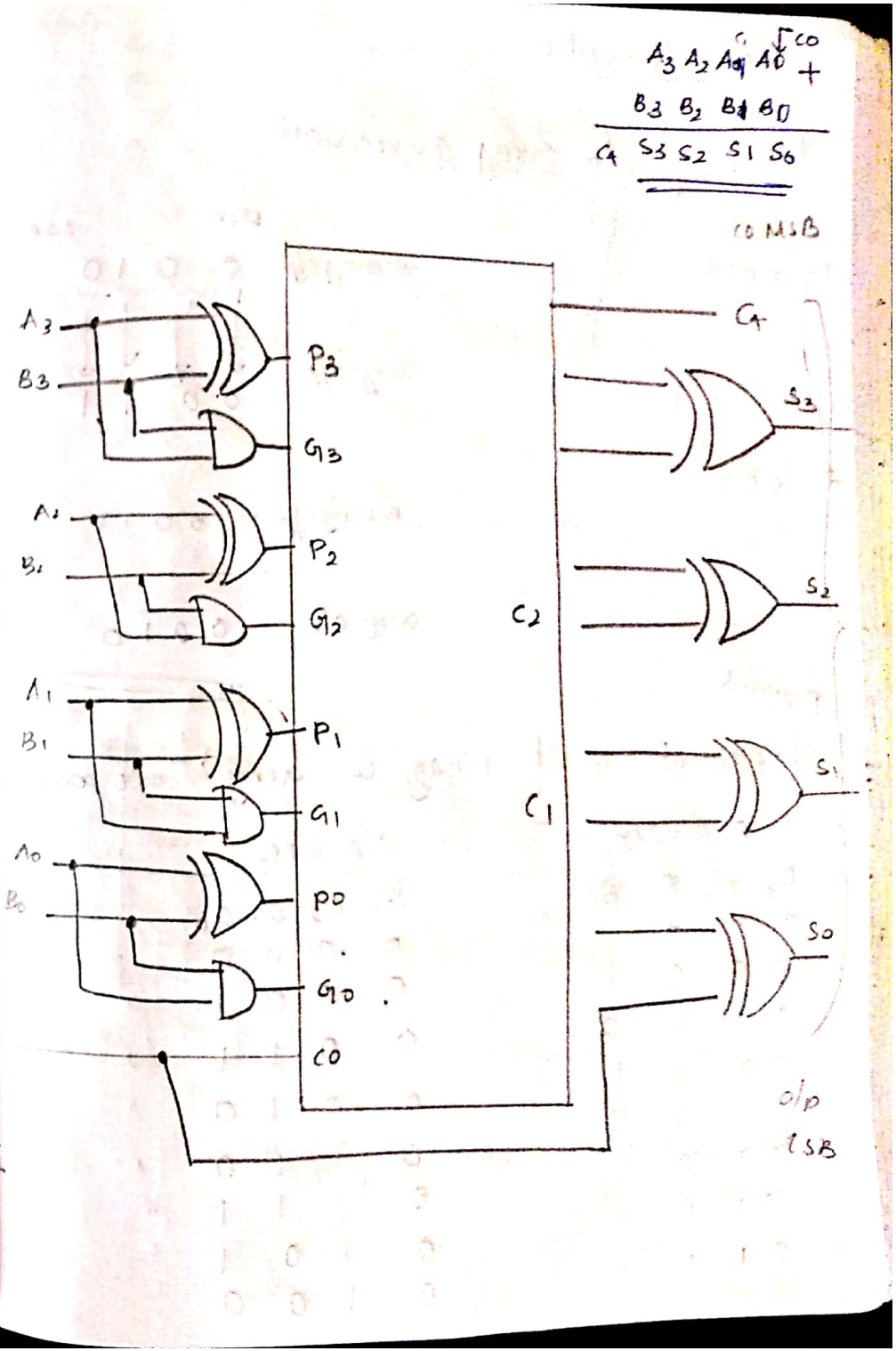
$= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

$C_4 = G_3 + P_3 C_3$

$= G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0)$

$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$

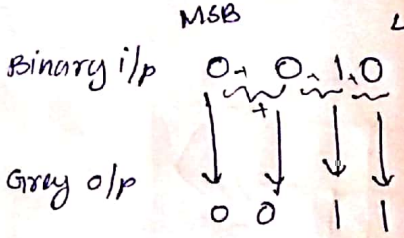




Code Converter

Binary to Grey conversion

1. 0010



2. 0011

Binary i/p 0 0 1 1

Grey o/p 0 0 1 0

Design

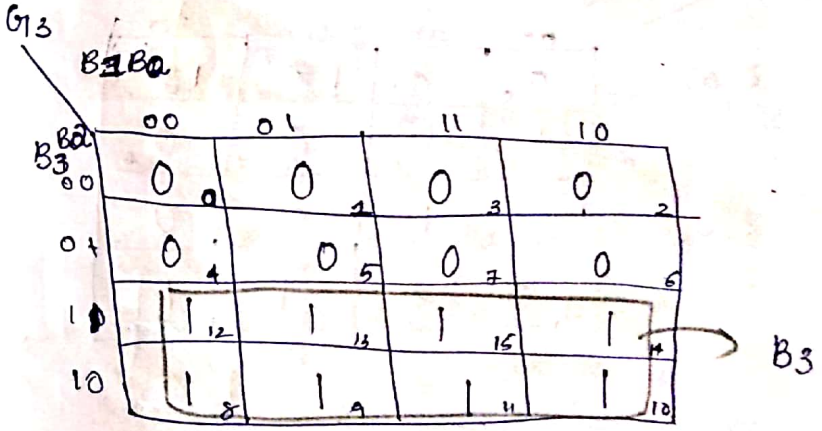
1. Implement 4 bit binary to grey code converter.

i/p Binary

B_3	B_2	B_1	B_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1

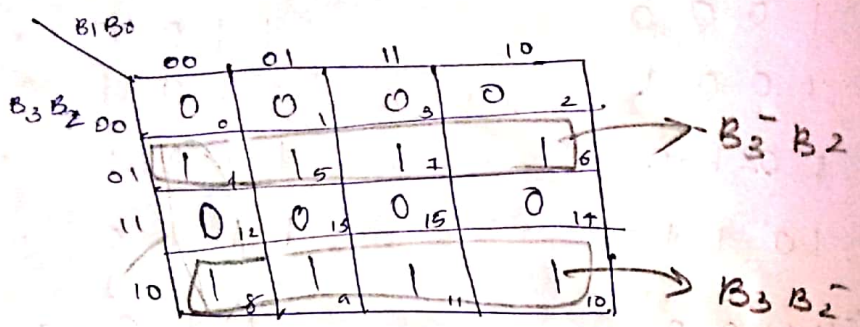
o/p Grey

G_3	G_2	G_1	G_0	
0	0	0	0	0
0	0	0	1	1
0	0	1	1	2
0	0	1	0	3
0	1	1	0	4
0	1	1	1	5
0	1	0	1	6
0	1	0	0	7
1	1	0	0	8
1	1	0	1	9
1	1	1	1	10
1	1	1	0	11
1	0	1	0	12
1	0	1	1	13
1	0	0	1	14
1	0	0	0	15



$G_3 = B_3$

G_2

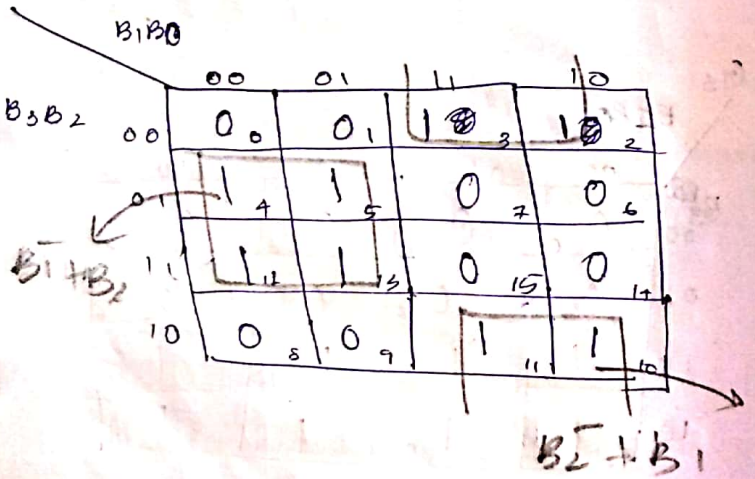


$G_2 = B_2 \oplus B_3$

$G_1 = B_1 \oplus B_2$

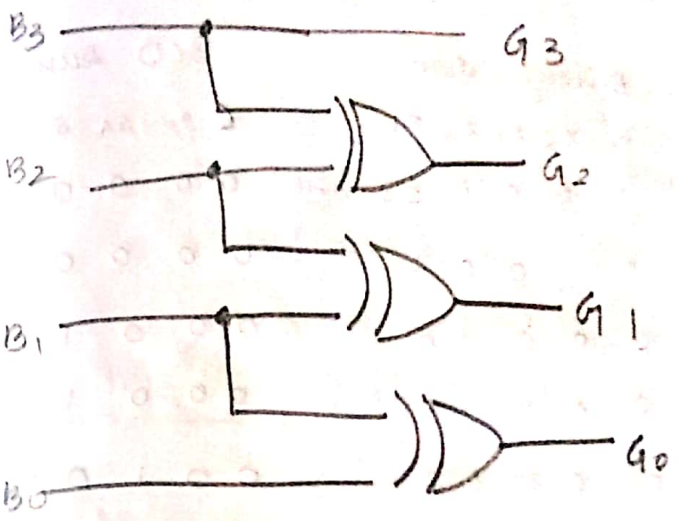
$G_0 = B_0 \oplus B_1$

G_1



~~$G_1 = \bar{B}_1 + B_2$~~

$G_1 = B_1 \oplus B_2$



HW

2. Design and implement 4 bit grey to binary code converter.

~~Grey to~~

Grey to Binary

1. 0 0 1 0.

Grey i/p 0 0 1 0

Binary o/p 0 0 1 1

2. 1 1 1 1

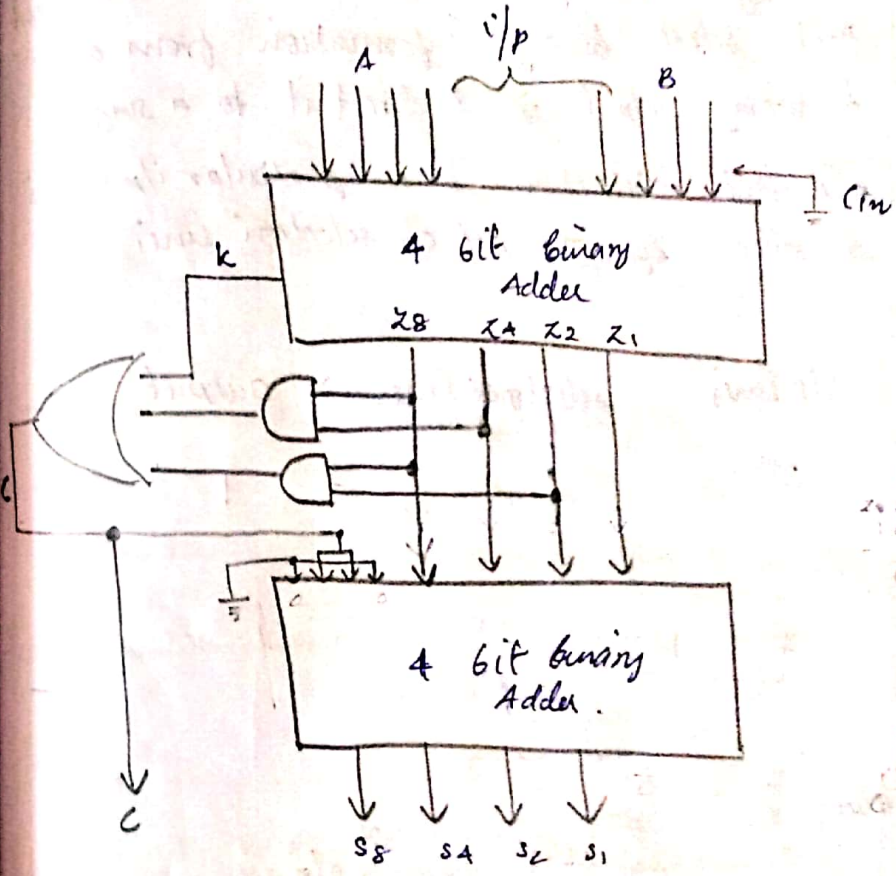
Grey i/p 1 1 1 1

Binary o/p 1 0 1 0

BCD Adder

Decimal	Binary Sum					BCD Sum				
#	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
1	0	0	0	0	1	0	0	0	0	1
2	0	0	0	1	0	0	0	0	1	0
3	0	0	0	1	1	0	0	0	1	1
4	0	0	1	0	0	0	0	1	0	0
5	0	0	1	0	1	0	0	1	0	1
6	0	0	1	1	0	0	0	1	1	0
7	0	0	1	1	1	0	0	1	1	1
8	0	1	0	0	0	0	1	0	0	0
9	0	1	0	0	1	0	1	0	0	1
10	0	1	0	1	0	1	0	0	0	0
11	0	1	0	1	1	1	0	0	0	1
12	0	1	1	0	0	1	0	0	1	0
13	0	1	1	0	1	1	0	0	1	1
14	0	1	1	1	0	1	0	1	0	0
15	0	1	1	1	1	1	0	1	0	1
16	1	0	0	0	0	1	0	1	1	0
17	1	0	0	0	1	1	0	1	1	1
18	1	0	0	1	0	1	1	0	0	0
19	1	0	0	1	1	1	1	0	0	1

$C = k + z_8 z_4 + z_8 z_2$



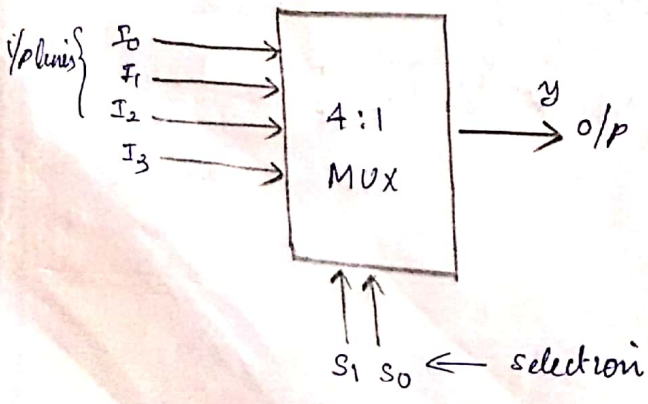
Multiplexer (Data Selector)

A digital multiplexer is a combinational circuit that select binary information from one of many i/p lines & direct it to a single o/p line. selection of a particular i/p line is controlled by a set of selection lines

I/p lines	selection lines	output
2^n	n	1

4x1

4 : 1 MUX

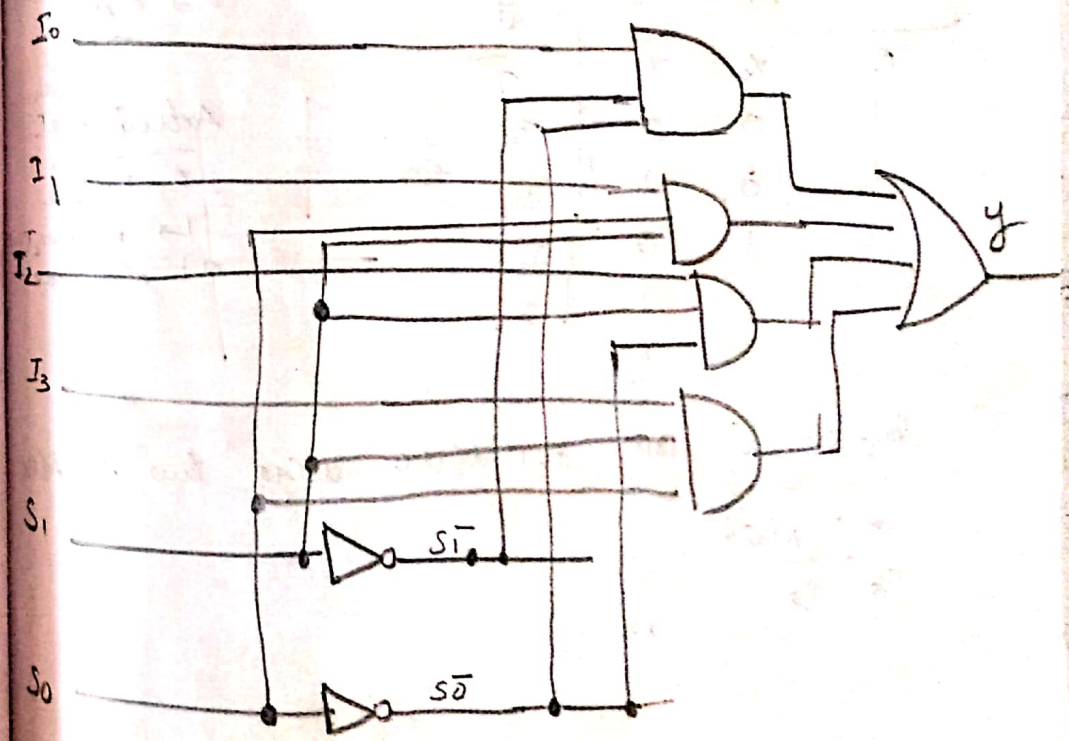


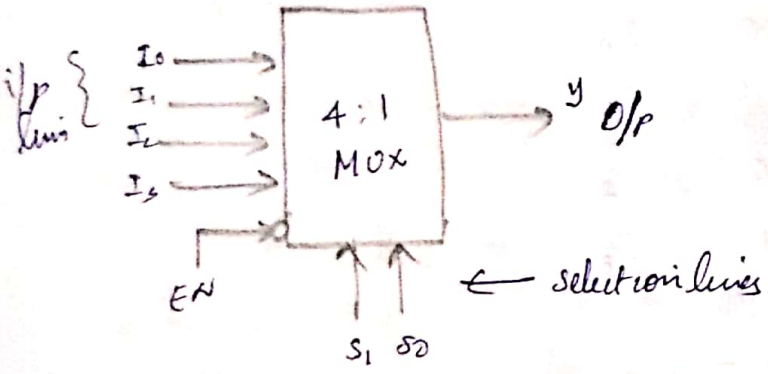
S_1	S_0	y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

0 1 I_1
$\bar{S_1}$ S_0 I_1
1.0 . I_1
= I_1

0 0 I_0
S_1 S_2 I_0
1 . 1 . I_0
= I_0

$$y = \bar{S_1}\bar{S_0} I_0 + \bar{S_1} S_0 I_1 + S_1 \bar{S_0} I_2 + S_1 S_0 I_3$$





→ Enable active high

E	S_1	S_0	y
0	X	X	0
1	0	0	I_0
1	0	1	I_1
1	1	0	I_2
1	1	1	I_3

Enable lines

Active high

1 → o/p will work

0 → o/p o and don't work.

Active low

0 → unit will

↳ network not with 1 & 0

⇒ Implement an 8:1 MUX using two 4:1 MUX

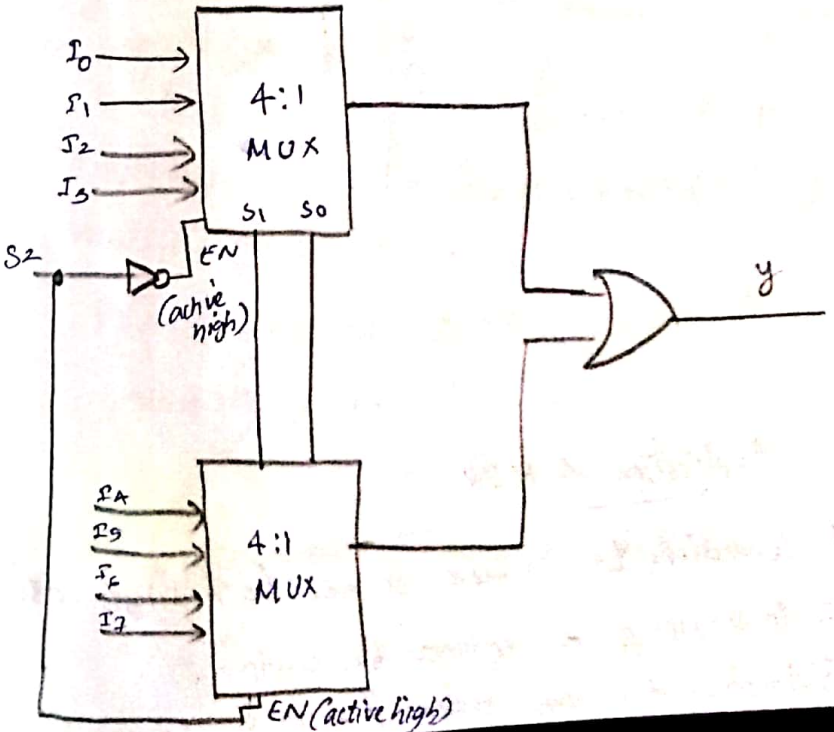
8:1 MUX

S_2 S_1 S_0

S_2

8:1 MUX

S_2	S_1	S_0	y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7



Application of MUX

1. Connecting two or more sources to a single destination
2. Constructing a common bus system.
3. Implementing any Boolean functions

Implementing Boolean function using MUX

- If we have a Boolean expression of n+1 variables, we take 'n' of these variables and connect them to the select lines of a multiplexer.
- The remaining single variable of the function is used for the i/ps of the multiplexer.
- If 'A' is this single variable, then the i/ps of the multiplexer are choosen to be either A' or $\bar{A}$ or 0 or 1.
- Any function of n+1 variable can be implemented with $2^n : 1$ MUX

⇒ Implement $F(A,B,C) = \Sigma(1,3,5,6)$ using multiplexer.

$F(A,B,C) = \Sigma(1,3,5,6)$

Number of variable = 3 = n+1

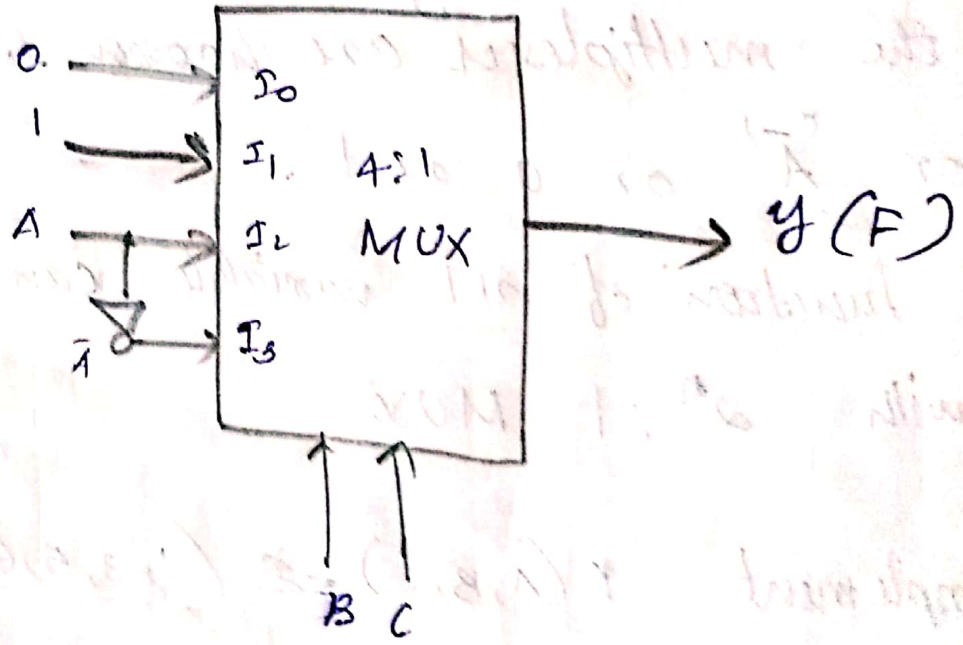
$\therefore n = 2$ (No. of selection lines)

∴ we have to use 4:1 MUX

Select line - B,C

	I_0	I_1	I_2	I_3
$\bar{A}$	0	(1)	2	(3)
A	4	(5)	(6)	7
	0	1	A	$\bar{A}$

0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
		1



4B

⇒ Implement full adder using multiplexer

$S(A, B, C_{in}) = \Sigma(1, 2, 4, 7)$

$C_{out}(A, B, C_{in}) = \Sigma(3, 5, 6, 7)$

No: of variable = 3 = n+1

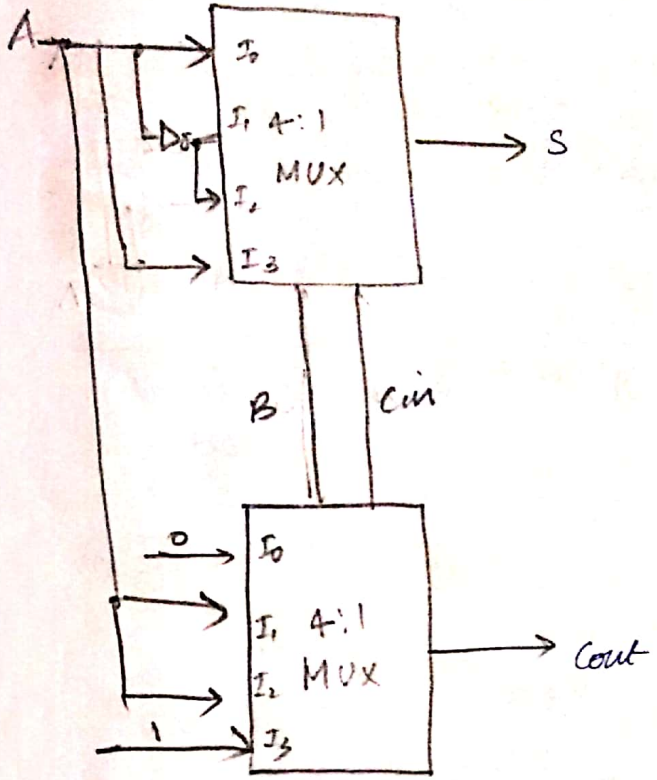
No: of select lines = 2

∴ we have to use 4:1 MUX

Select lines are B, C_{in}

S	I_0	I_1	I_2	I_3
$\bar{A}$	0	(1)	(2)	3
A	(4)	5	6	(7)
	A	$\bar{A}$	$\bar{A}$	A

C_{out}	I_0	I_1	I_2	I_3
$\bar{A}$	0	1	2	(3)
A	4	(5)	(6)	(7)
	0	A	A	1



HW

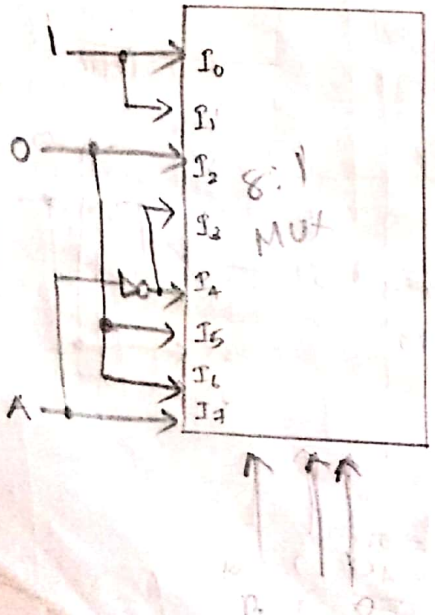
⇒ Implement $F(A, B, C, D) = \Sigma(0, 1, 3, 4, 8, 9, 15)$ using Multiplexer

No: of variables = n+1 = 4

No: of select lines = 3

∴ 2^3:1 MUX (8:1 MUX)

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
	1	1	0	$\bar{A}$	$\bar{A}$	0	0	A



$A \oplus B \oplus C$

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

$B\bar{C}$

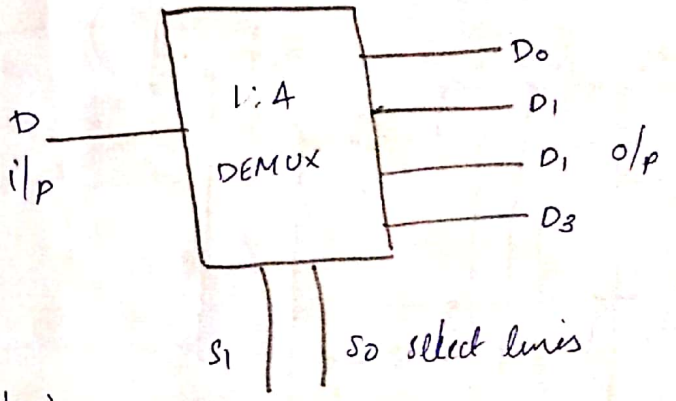
$BC + \bar{B}\bar{C}$

De-Multiplexer (Data distributor)

i/p line	select line	o/p line
1	n	2^n

If $n = 2$

1 : 4 DEMUX



Enable line active high o/p = 1

S_1	S_0	D_0	D_1	D_2	D_3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

$D_0 = S_1 \bar{S_0} D$

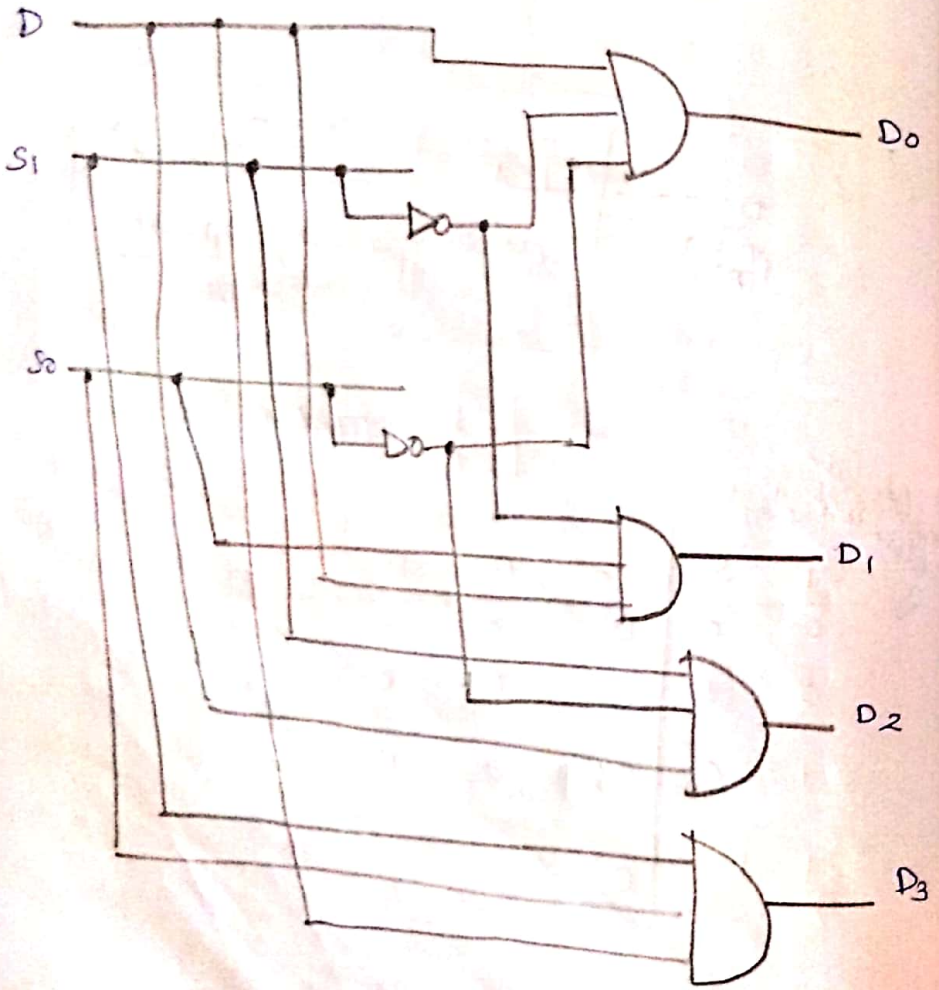
$D_1 = \bar{S_1} S_0 D$

$D_2 = S_1 \bar{S_0} D$

$D_3 = S_1 S_0 D$

$S_1 \bar{S_0} D$

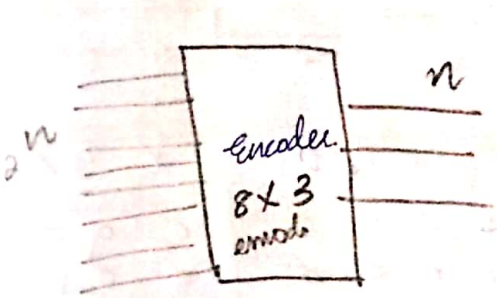
$D_0 = 1 \cdot 1 \cdot D = D$

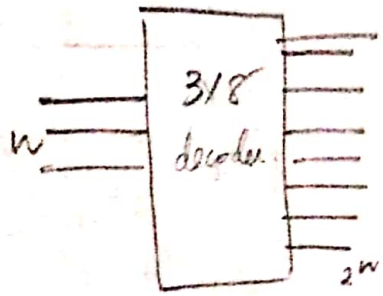


Encoder and Decoder Circuits

An encoder has 2^n or less input lines and n output lines. The o/p line generates the binary code for 2^n input variables

Encoder





Octal to Binary encoder

8 x 3 encoder

3 x 8 decoder.

Hexadecimal to Binary

16 x 4 encoder.

4 x 16 decoder.

Decimal to BCD encoder

10 x 4 encoder.

4 x 10 decoder.

$2^n \times n$ - encoder

2^n to n

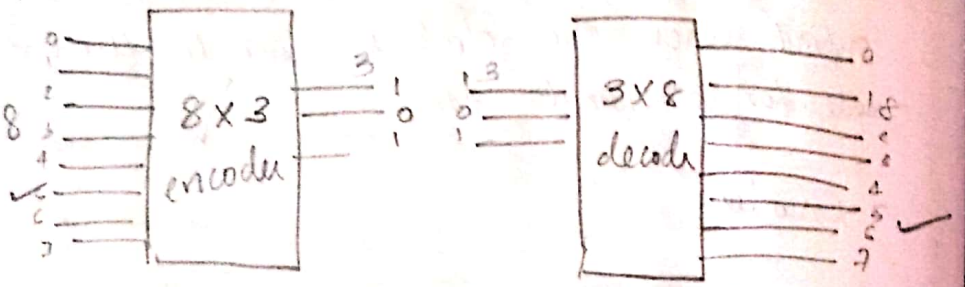
$n \times 2^n$ decoder

n to 2^n /

n to m line decoder.

$m \leq 2^n$

Octal to Binary

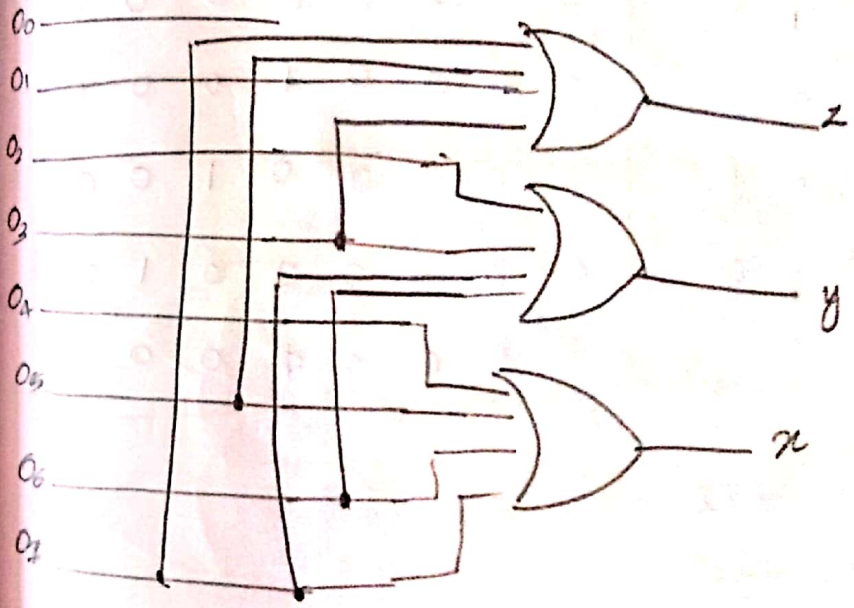


O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$x = O_4 + O_5 + O_6 + O_7$

$y = O_2 + O_3 + O_6 + O_7$

$z = O_1 + O_3 + O_5 + O_7$



Decoder

It is a combinational circuit that converts binary information from n i/p lines to a maximum of 2^n unique o/p lines. The purpose of a decoder is to generate 2^n or less minterms for n i/p variables.

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

$D_0 = \bar{x}\bar{y}\bar{z}$

$D_1 = \bar{x}\bar{y}z$

$D_2 = \bar{x}y\bar{z}$

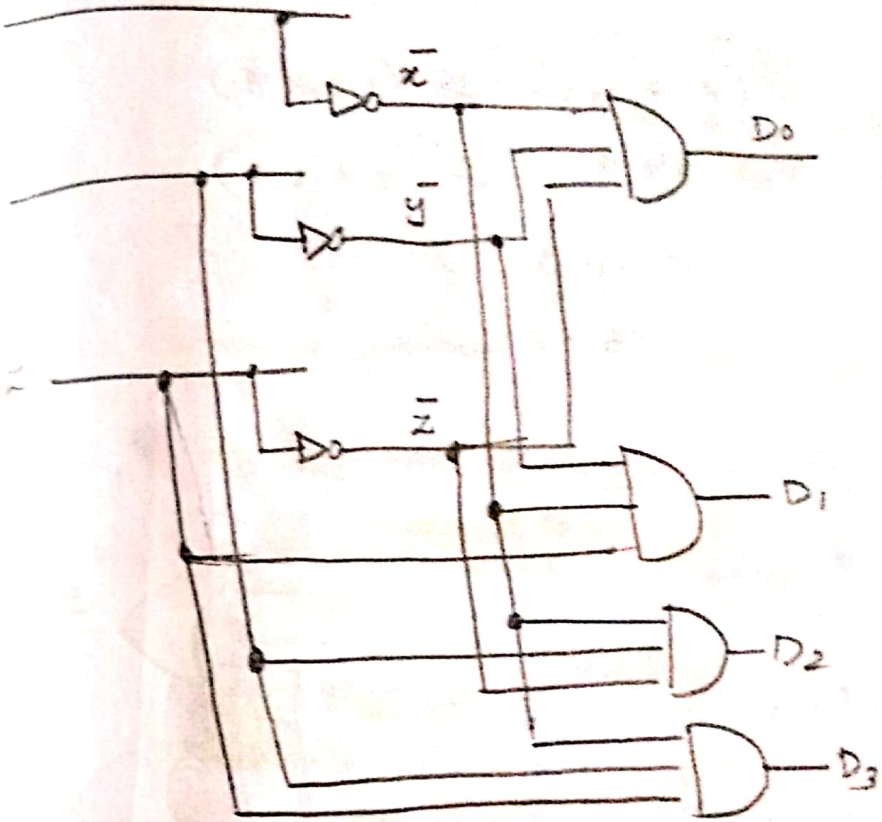
$D_3 = \bar{x}yz$

$D_4 = x\bar{y}\bar{z}$

$D_5 = x\bar{y}z$

$D_6 = xy\bar{z}$

$D_7 = xyz$



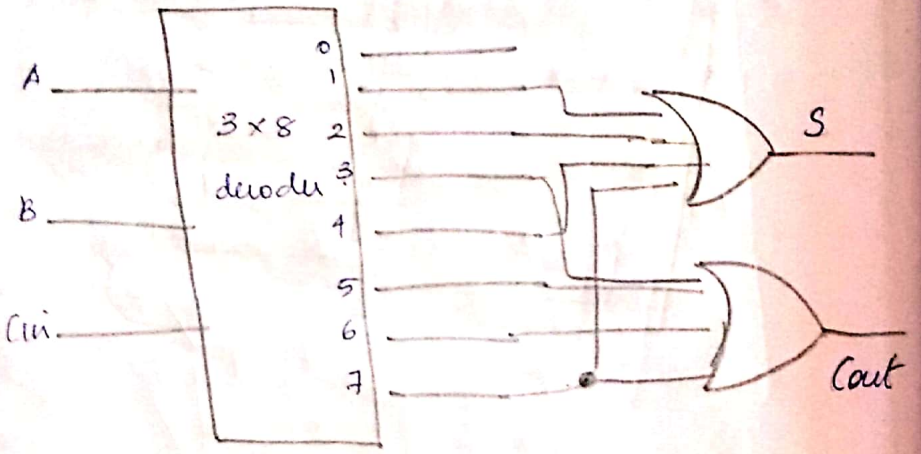
⇒ Implement full adder using decoder.

$S(A, B, C_{in}) = \Sigma(1,2,4,7)$

$Cout(A, B, C_{in}) = \Sigma(3,5,6,7)$

No: of variable = 3

∴ 3 × 8 decoder.



XOR and Equivalence function

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

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

An n variable XOR expression =

Boolean function with $\frac{2^n}{2}$ minterms whose equivalent binary numbers have an odd number of ones.

$x \oplus y = 01 + 10$

$= \bar{x}y + x\bar{y}$

00
01
10
11

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

An n variable equivalence (X-NOR) expression =

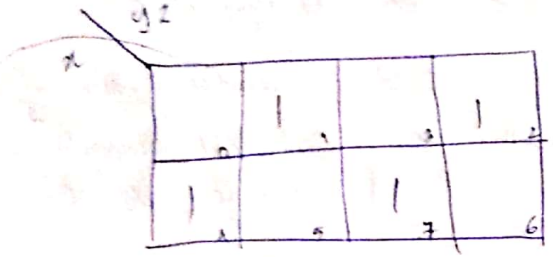
Boolean function with $\frac{2^n}{2}$ minterms whose equivalent binary number have an even number of zeros

When the number of variables in a function is odd the minterms with even number of zeroes are same as minterms with odd number of ones.

∴ XOR expression and equivalence expression are equal when both have same odd number of variables.

i.e,

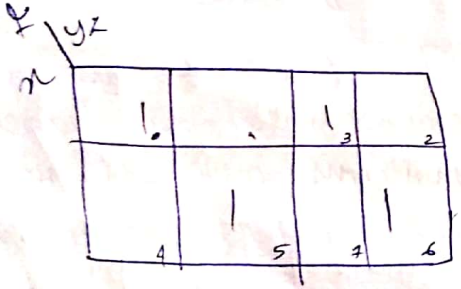
$x \oplus y \oplus z = x \odot y \odot z$



00
01
11
10

When the minterms of a function with an odd number of variables have an even number of ones or an odd number of zeroes then the function can be expressed as the complement of either XOR or equivalence expression.

$$Y=(x\oplus y\oplus z)' = (x\odot y\odot z)'$$



00
11
10

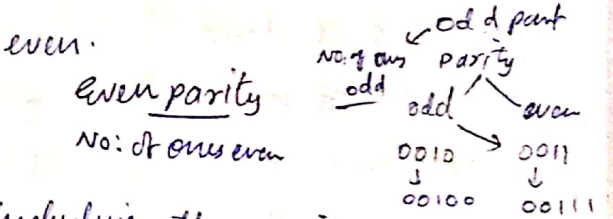
$$\Rightarrow (A\oplus B\oplus C)' = A\oplus B\odot C$$

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

Application of XOR and XNOR

1 Parity Generator and Parity Checker

1. A parity bit is a scheme for detecting errors during the transmission of binary information.
2. A parity bit is an extra bit included with a binary ~~mesg~~ message to make the number of ones either odd or even.



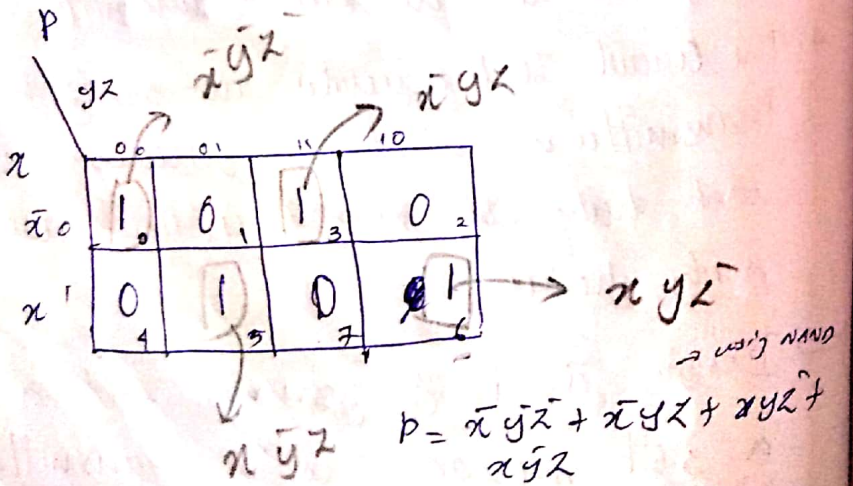
Even parity
No: of ones even

3. The message including the parity bit is transmitted and check at the receiving end for error. An error is detected if the checked parity doesn't corresponds to the one transmitted.
4. The circuit that generates the parity bit ~~is~~ in the transmitter is called parity generator and the circuit that checks the parity bit in the receiver is called parity checker.

⇒ Design a parity generator and a checker when a 3 bit message to be transmitted with an odd parity bit

Parity Generator

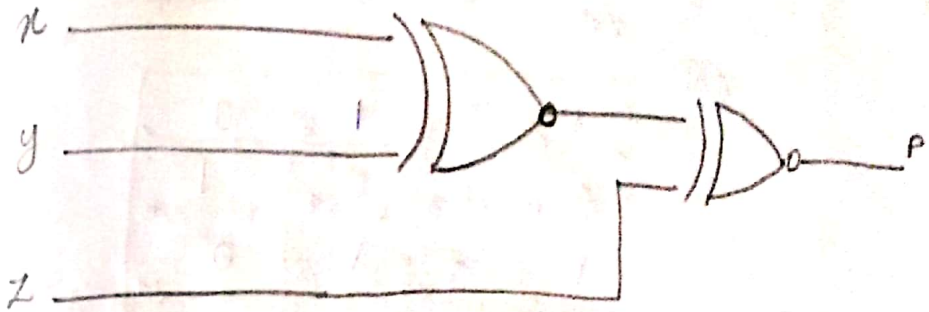
i/p 3bit message			o/p - parity bit (odd parity)
x	y	z	P
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



$P = \bar{x}\bar{y}\bar{z} + \bar{x}yz + xy\bar{z} + x\bar{y}z$

→ using NAND

$P = x \oplus y \odot z$

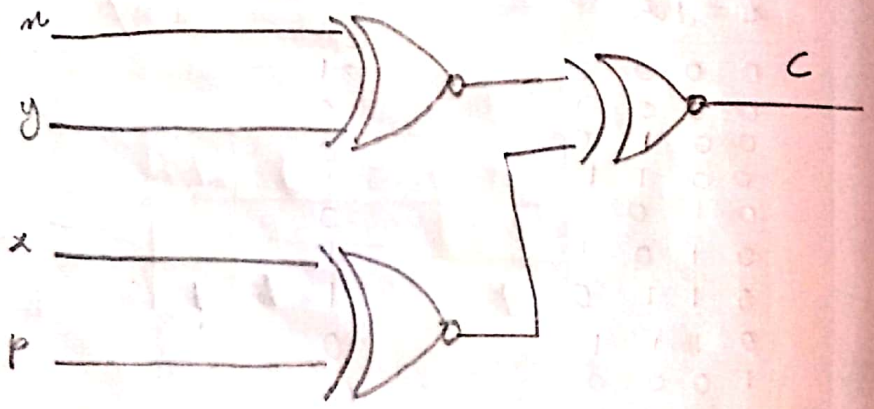


Four bit odd parity checker

i/p				o/p (if error = 1, no error = 0)
x	y	z	P	C
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	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

C \ xp / xy	00	01	11	10
00	1 (0)	0 (1)	1 (3)	0 (2)
01	0 (4)	1 (5)	0 (7)	1 (6)
11	1 (12)	0 (13)	1 (15)	0 (14)
10	0 (8)	1 (9)	0 (11)	1 (10)

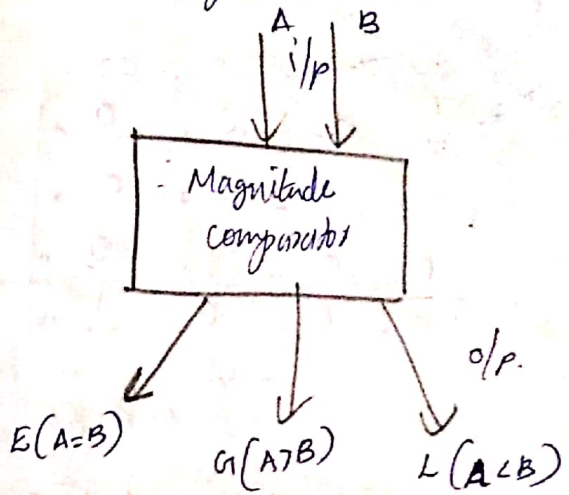
$$C = x \odot y \odot z \odot p$$



IMP ⇒ Implement the function $f = A \oplus B \oplus C \oplus D$ using multiplexer and using decoder.

$$f = \Sigma(1, 2, 4, 7, 8, 11, 13, 14)$$

Magnitude Comparator

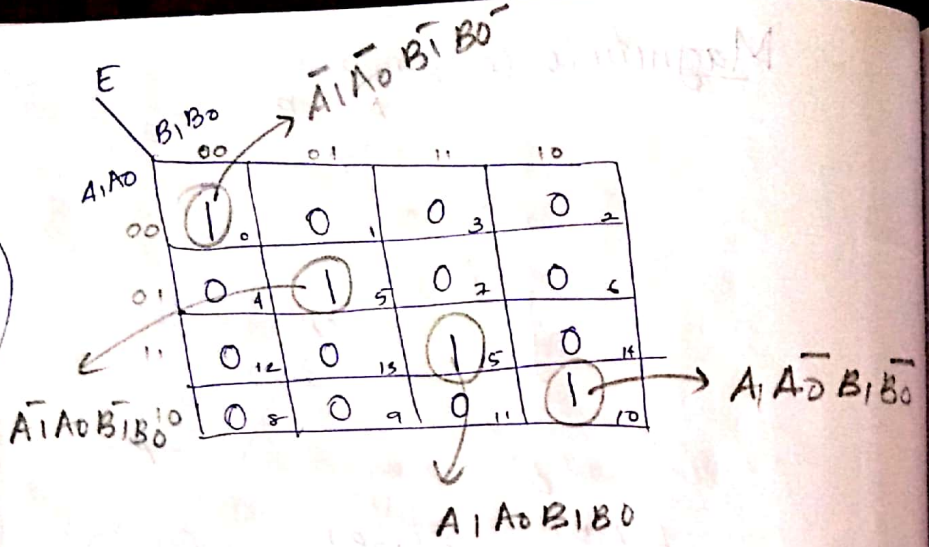


Two bit Magnitude Comparator

A - A_1A_0
B - B_1B_0

i/p

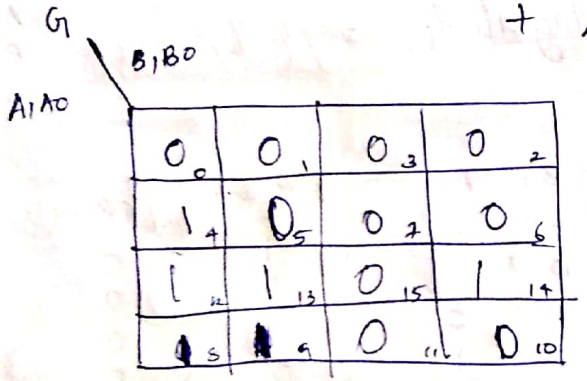
A_1	A_0	B_1	B_0	E(A=B)	G(A>B)	L(A<B)	
0	0	0	0	1	0	0	0
0	0	0	1	0	0	1	1
0	0	1	0	0	0	1	2
0	0	1	1	0	0	1	3
0	1	0	0	0	1	0	4
0	1	0	1	1	0	0	5
0	1	1	0	0	0	1	6
0	1	1	1	0	0	1	7
1	0	0	0	0	1	0	8
1	0	0	1	0	1	0	9
1	0	1	0	1	0	0	10
1	0	1	1	0	0	1	11
1	1	0	0	0	1	0	12
1	1	0	1	0	1	0	13
1	1	1	0	0	1	0	14
1	1	1	1	1	0	0	15



XNOR
same → 1
diff → 0

$$E = \bar{A}_1\bar{A}_0\bar{B}_1\bar{B}_0 + A_1\bar{A}_0B_1\bar{B}_0 + A_1A_0B_1B_0 + \bar{A}_1A_0\bar{B}_1B_0$$

REFER SUBTRACTOR



$$E(A=B) = (A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$

$A_2 A_1 A_0$
$B_2 B_1 B_0$