Chapter – 6

* MSI and LSI Components in Combinational Logic Design

We use Boolean function simplification method to obtain low cost ckt and the method is known as classical procedure. This classical method can be tedious and cumbersome for large number of input variable and the design may not be efficient.

Nowadays, Integrated circuit (IC) are prefered in design of modern digital system which performs the same function of combination ckt. In other word we can say that these ckts are available in integrated ckts and are classified according to the number of gates.

1. **Small Scale Integration (SSI) :→** The number of gates in a single package is less than 10 are called small scale Integration (SSI) Ic.
2. **Medium scale Integration (MSI) :→** It has 10 to 200 gates in a single package. They perform specific digital function such as decoder, encoder, multiplexer, Demultiplexer, adder, substractors and comparators.
3. **Large Scale Integration (LSI) :→** It has 200 and a few thousand gates in single package. They include digital system such as processors, memory chips and Programmable modules.
4. **Very Large Scale Integration (VLSI) :→** VLSI devices contains thousand of gates. eg :→ Large memory arrays.

and complex microcomputer chips. VLSI revolutionized the computer system design technology introducing Digital Integrated Circuits.

* Binary parallel Adder :→

A Binary parallel Adder is a digital circuit that produces the arithmetic sum of two binary number in parallel. It consist of full Adder Connected in cascade, with the output carry from one full adder connected to the input carry of the next full adder.

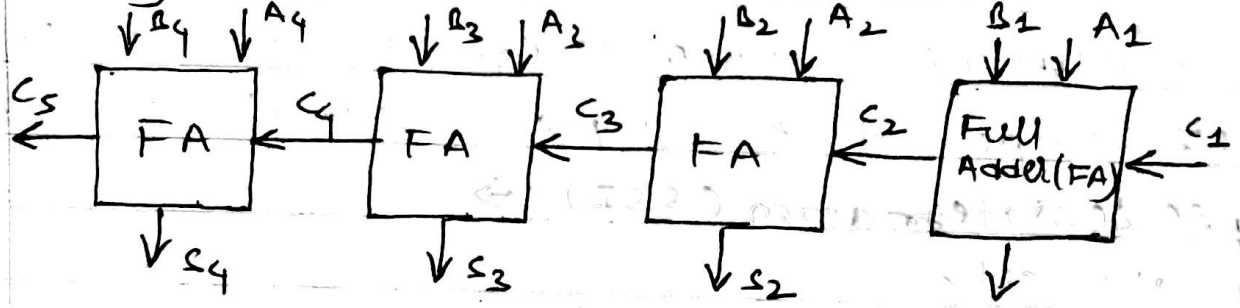


fig:→ Interconnection of 4 bit binary parallel Adder

The above figure shows four bit binary parallel Adder. It requires 4 full adders. The Augend bits of A and the Addent bits of B are designated by subscript numbers (1, 2, 3, 4) from right to left. The carries are connected in a chain through the full adder. The input carry to the adder is C_1 and output carry is C_5 and generates the required sum bit 'S'.

* Binary substractor (Binary Adder substractor)

[As Already been discussed to perform substraction i.e A-B can be accomplished by taking the 2's complement of B and adding it to A.]

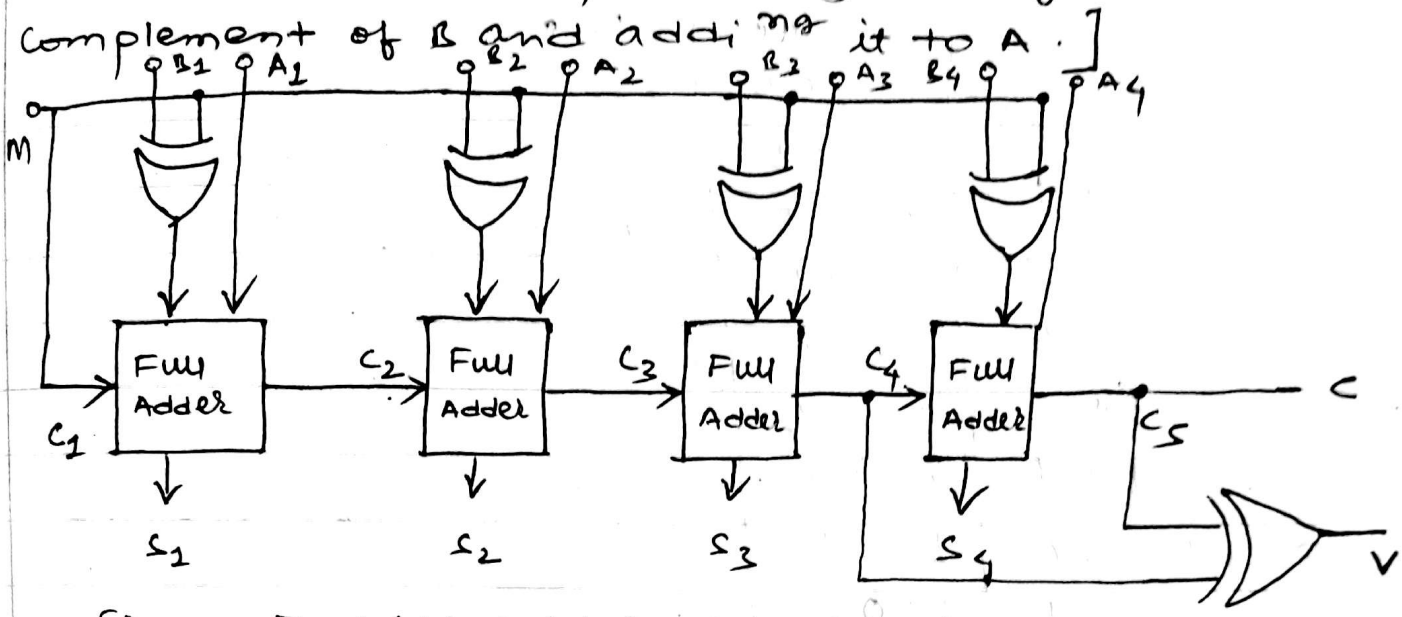


fig :→ Four bit Adder - substractor

Above figure shows the four bit Adder - substractor ckt that can perform both the operation of Adder and substractor which can be controlled by the mode input 'M'. It has two values

1) when M=0, it becomes $B_1 \oplus 0 = B_1$, the full Adder receives B_1, as the input carry $C_1 = 0$. Therefore the full Adder performs $A_1 + B_1$ and so operates the other Full Adder.

2. Similarly when M=1, it becomes $B_1 \oplus 1 = \bar{B}_1$, the full adder receives $\bar{B}_1$ and carry $C_1 = 1$. The input $\bar{B}_1$ is then added 1 via input carry and result's 2's complement of B_1. This is then further added with other input A_1 resulting substraction $A_1 - B_1$. The same procedure is followed by other Full adder ckt that performs substraction.

The exclusive-OR with output V is for detecting an overflow.

Example :⇒ Design a BCD to Excess-3 code converter by 4-bit full adder MSI ckt.

	A_4 A_3 A_2 A_1 BCD 8 4 2 1	Excess-3 S_4 S_3 S_2 S_1
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0

From the truth table the excess-3 equivalent code is obtained by addition of binary 0011.

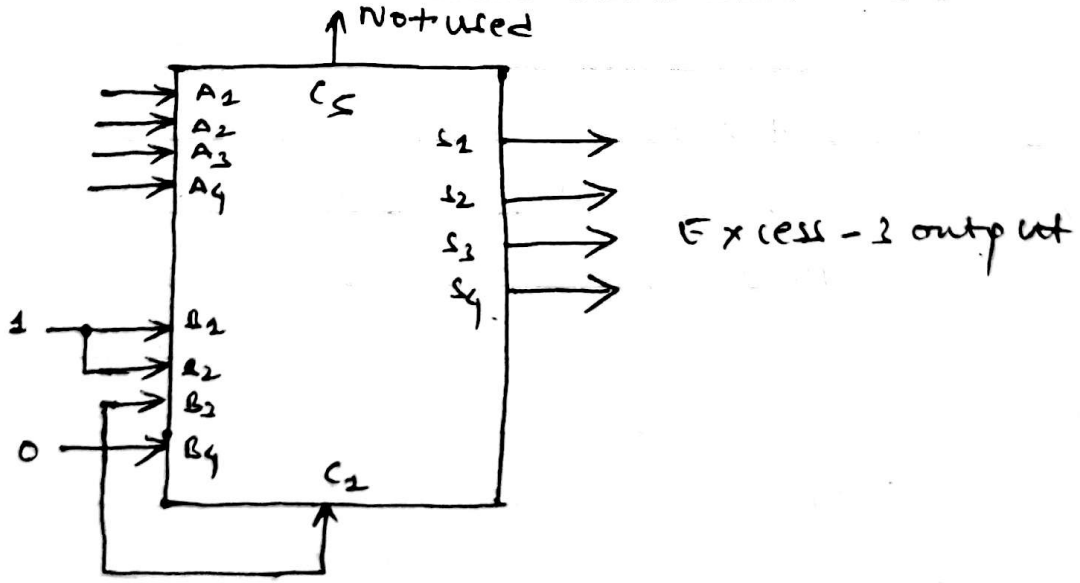


fig :⇒ 4 bit - Full Adder MSI ckt (BCD to Excess-3 Code converter)

From the figure BCD digit is applied to input A. Input B are set to constant $B_4B_3B_2B_1$ 0011. Therefore, S outputs from the ckt gives the Excess-3 code of the input BCD digit.

eg :→ when two number with n digits each are added and the sum is a number occupying n+1 digit, we say overflow has occured.

* Decimal Adder :→

Computers or calculators performs airthmetic operation of decimal number system by Adder in Binary coded decimal (BCD) form and present results in the same code. Airthmetic operation is done by Adder Similarly Decimal Adder requires minimum of nine inputs and five output, since four bits are required to code each decimal digit and the circuit must have an input carry and output carry.

Therefore their is wide variety of decimal adder circuits depending upon the code used to represent the decimal digit.

Eg :→ Decimal adder for the BCD code

* BCD Adder :→

Consider the airthmetic addition of two decimal digits in BCD together with an input carry from a decimal digit.

Each input digit doesnot exceed 9, the output sum cannot be greater than 9+9+1=19 (1←input carry).

Suppose we apply two BCD digit to four bit binary adder. The Adder will form the sum in binary and produce result 0 to 19.

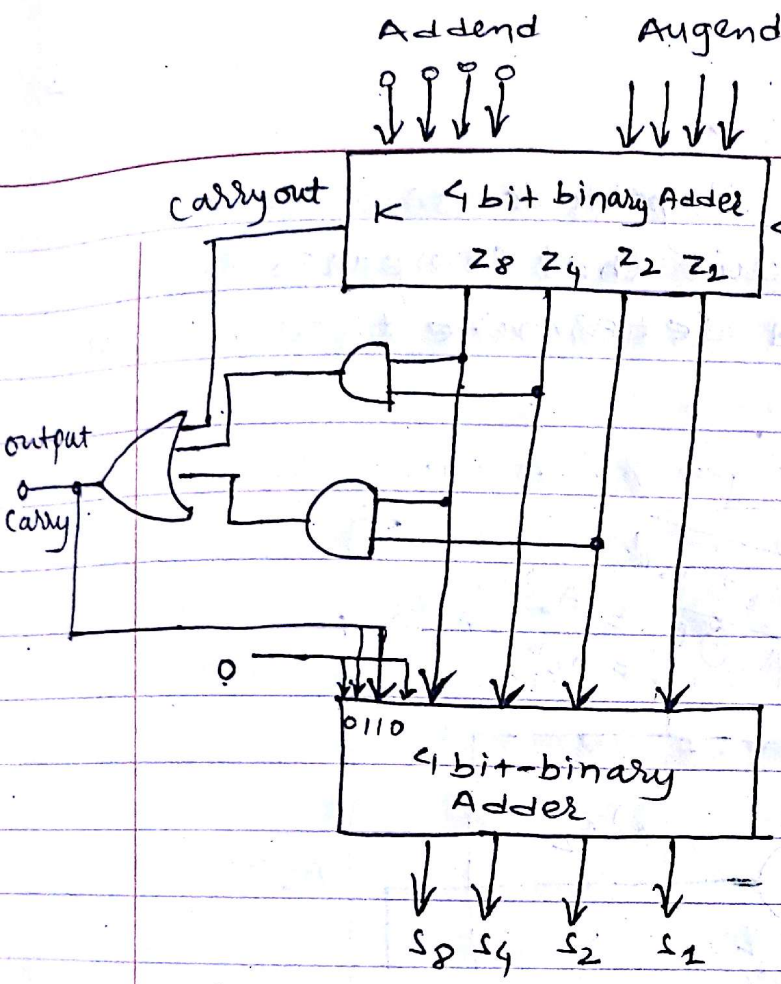


fig :→ Block Diagram of BCD Adder

BCD Adder that adds two BCD digits and produces a sum digit is shown in above figure. The two decimal digits, together with the input carry are first added to produce Binary sum.

When the output carry is 0, nothing is added to binary sum. But when output carry is equal to 1, (0110) binary is added to binary sum through the bottom four-bit Adder. The output carry is given by

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

* Magnitude Comparator :→

Magnitude Comparator is a combinational circuit that compares two number A and B and determine their relative magnitude.

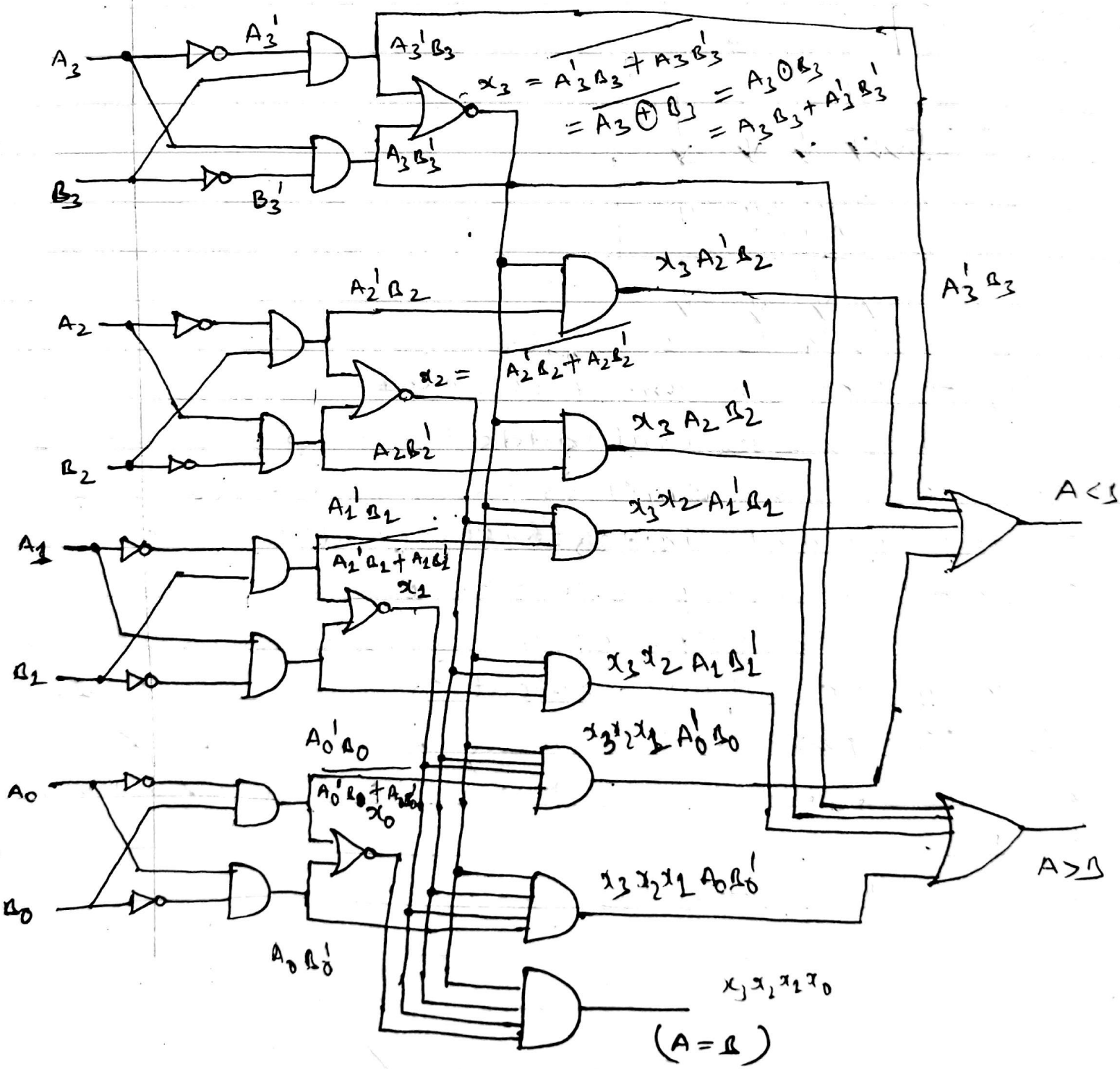


fig :→ Four bit magnitude Comparator

The outcome of the comparision is specified by 3 binary variables that indicate whether $A > B$, $A = B$ and $A < B$.

Consider two 4 digit number A and B.

Here, $A = A_3 A_2 A_1 A_0$ $B = B_3 B_2 B_1 B_0$. The two numbers are equal if all pair of significant digits are equal i.e $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$ and $A_0 = B_0$.

Therefore it can be expressed logically as

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

For $A = B$ the output of the combinational ckt is given by AND operation for all variable, Therefore $A = B = x_3 x_2 x_1 x_0$

If $A = 1$ and $B = 0$

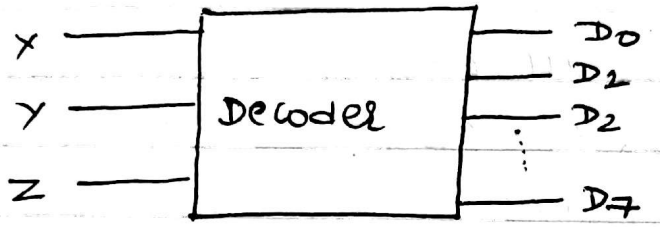
$$A > B = A_3 B_3' + x_3 A_2 B_2' + x_3 x_2 A_1 B_1' + x_3 x_2 x_1 A_0 B_0'.$$

IF $A = 0$ and $B = 1$

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

* Decoder

A decoder is a combinational circuit that converts binary information from n input lines to maximum of 2^n unique output lines. But if the decoder has unused or don't care combination the output will be less than 2^n combination.



Here, input $n = 3$
output $= 2^n = 2^3 = 8$

fig:→ Block Diagram of 3 input Decoder

* Design 3 to 8 line Decoder

If x y z are three input to the Decoder, it has $2^3 = 8$ output $D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7$.

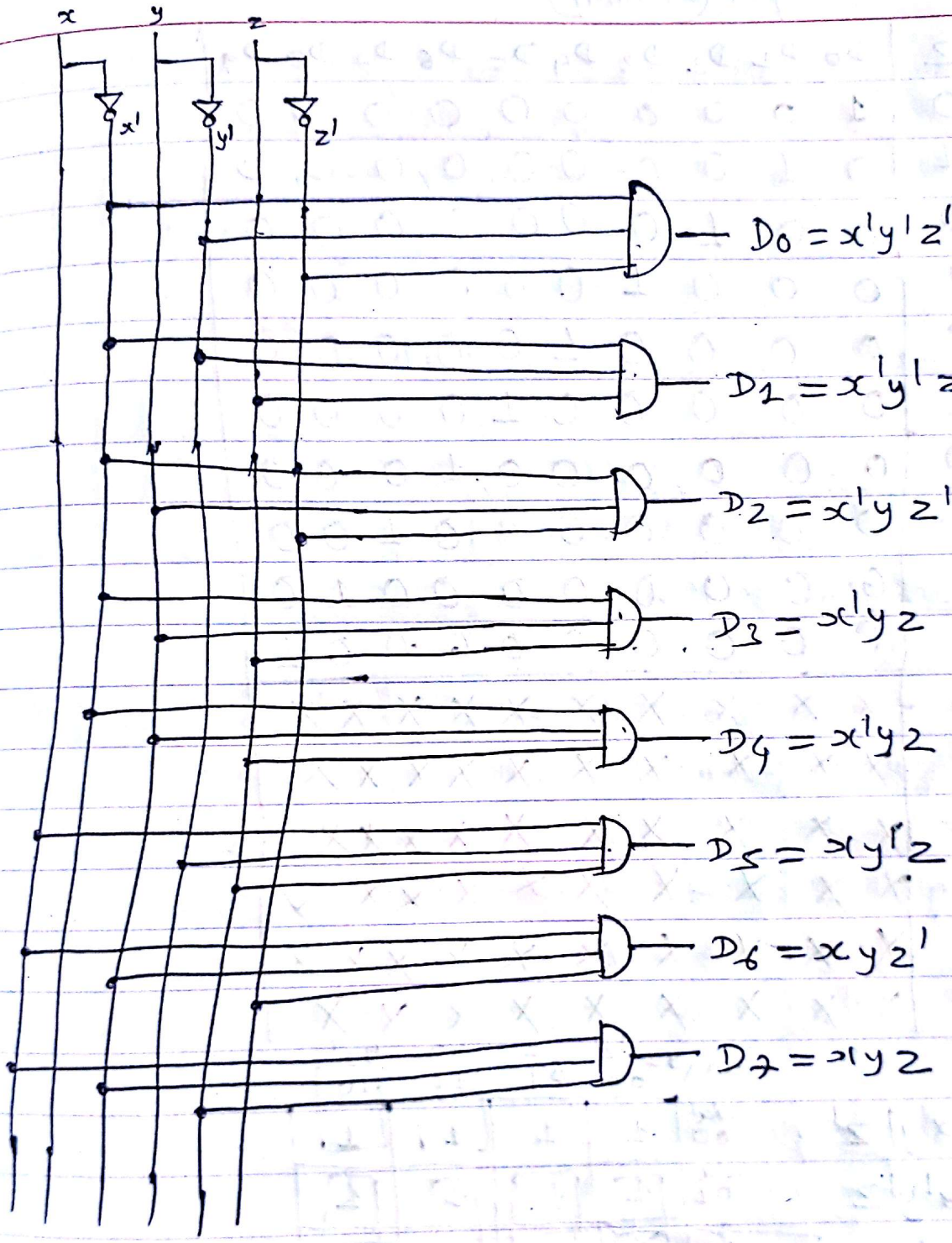
The truth table for 3 to line Decoder is

	x y z	D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7
0	0 0 0	1 0 0 0 0 0 0 0
1	0 0 1	0 1 0 0 0 0 0 0
2	0 1 0	0 0 1 0 0 0 0 0
3	0 1 1	0 0 0 1 0 0 0 0
4	1 0 0	0 0 0 0 1 0 0 0
5	1 0 1	0 0 0 0 0 1 0 0
6	1 1 0	0 0 0 0 0 0 1 0
7	1 1 1	0 0 0 0 0 0 0 1

The operation of decoder has seven output equal to '0' and one output equal to '1'.

$D_0 = x'y'z'$, $D_1 = x'y'z$, $D_2 = x'yz'$, $D_3 = x'yz$, $D_4 = xy'z$, $D_5 = xy'z$

$D_6 = xyz'$, $D_7 = xyz$



* Design a BCD to decimal decoder:

to represent BCD the decoder consist four inputs and ten outputs to represent decimal.

	input BCD	output (Decimal)									
	$w\,x\,y\,z$	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9
0	0 0 0 0	1	0	0	0	0	0	0	0	0	0
1	0 0 0 1	0	1	0	0	0	0	0	0	0	0
2	0 0 1 0	0	0	1	0	0	0	0	0	0	0
3	0 0 1 1	0	0	0	1	0	0	0	0	0	0
4	0 1 0 0	0	0	0	0	1	0	0	0	0	0
5	0 1 0 1	0	0	0	0	0	1	0	0	0	0
6	0 1 1 0	0	0	0	0	0	0	1	0	0	0
7	0 1 1 1	0	0	0	0	0	0	0	1	0	0
8	1 0 0 0	0	0	0	0	0	0	0	0	1	0
9	1 0 0 1	0	0	0	0	0	0	0	0	0	1
10	1 0 1 0	X	X	X	X	X	X	X	X	X	X
11	1 0 1 1	X	X	X	X	X	X	X	X	X	X
12	1 1 0 0	X	X	X	X	X	X	X	X	X	X
13	1 1 0 1	X	X	X	X	X	X	X	X	X	X
14	1 1 1 0	X	X	X	X	X	X	X	X	X	X
15	1 1 1 1	X	X	X	X	X	X	X	X	X	X

$D_0 = w'x'y'z'$

$D_1 = w'x'y'z$

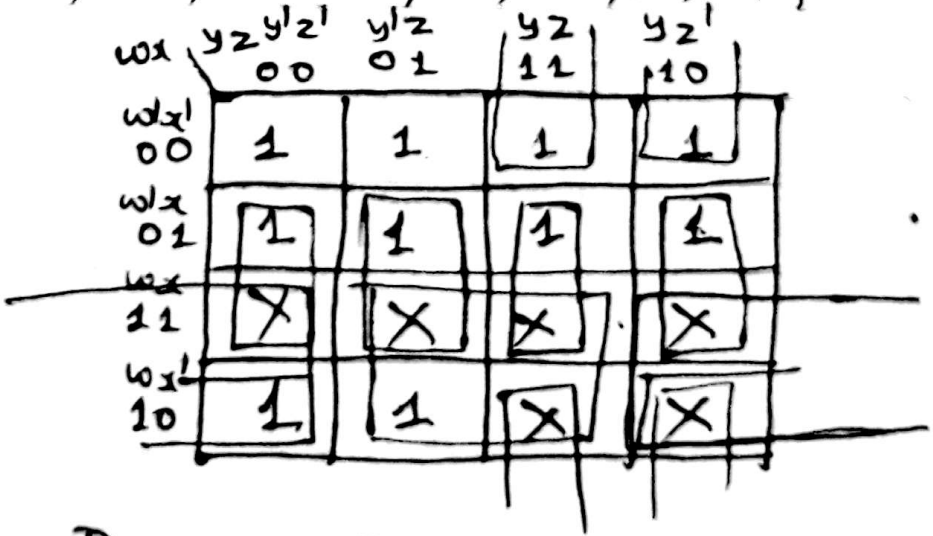
$D_2 = x'yz'$

$D_3 = x'yz$

$D_4 = xy'z'$

$D_5 = xy'z$

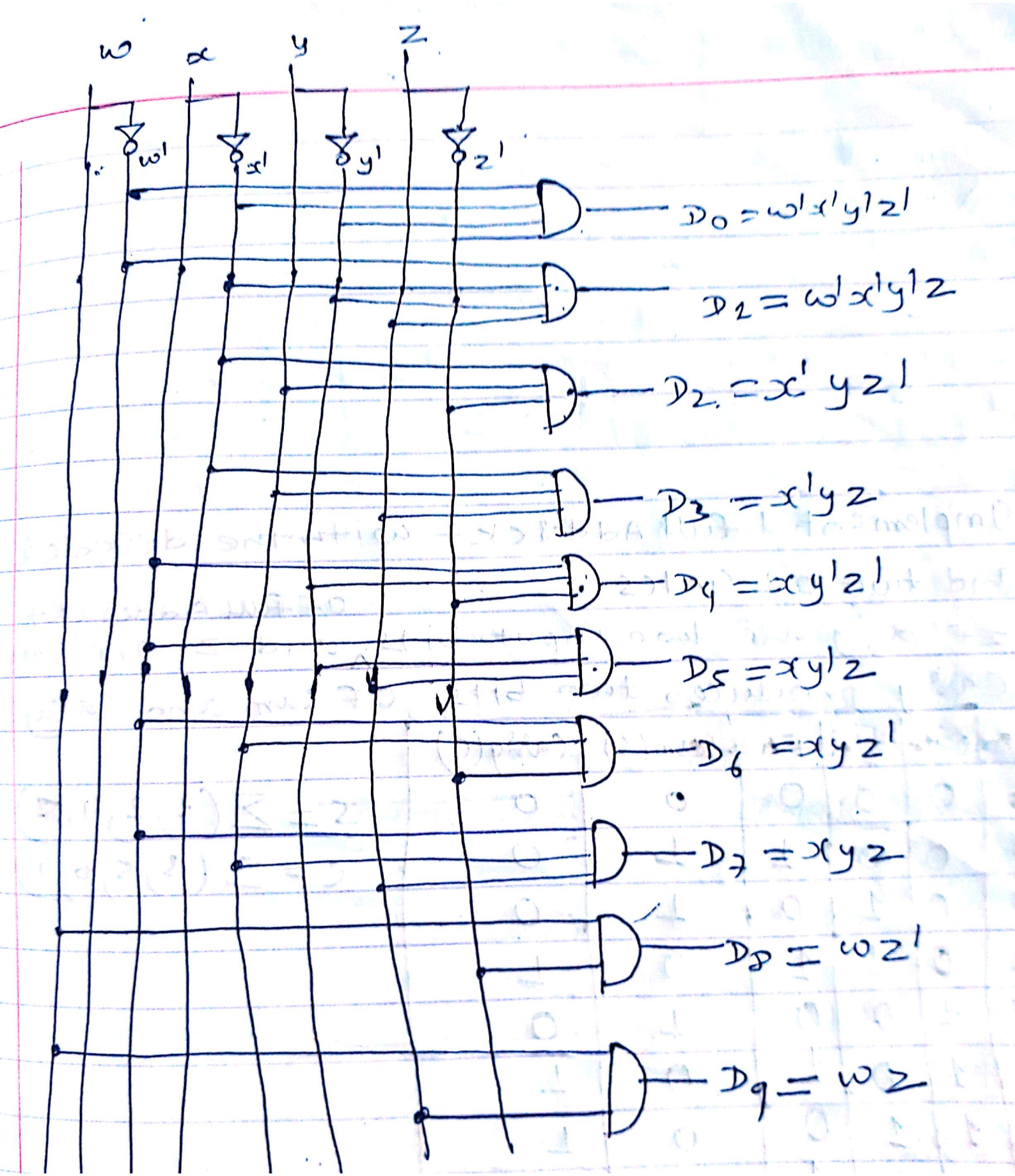
$D_6 = xyz'$



$D_7 = xyz$

$D_8 = w\,z'$

$D_9 = wz$



15 23 27 29 30 0

01111 10111 11011 11101 11110 00000

$\bar{v}wxyz + v\bar{w}xyz + vw\bar{x}yz + vwx\bar{y}z + vwxyz + \bar{v}\bar{w}x\bar{y}\bar{z}$

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

00010 00100 01000

2 4 8

vw \ xyz	$x'y'z'$	$x'y'z$	$x'yz$	xyz'	xyz	$xy'z$	$xy'z'$
$\bar{v}w'$	1			1			1
$\bar{v}w$	1				1		
vw			1		1	1	
vw'					1		

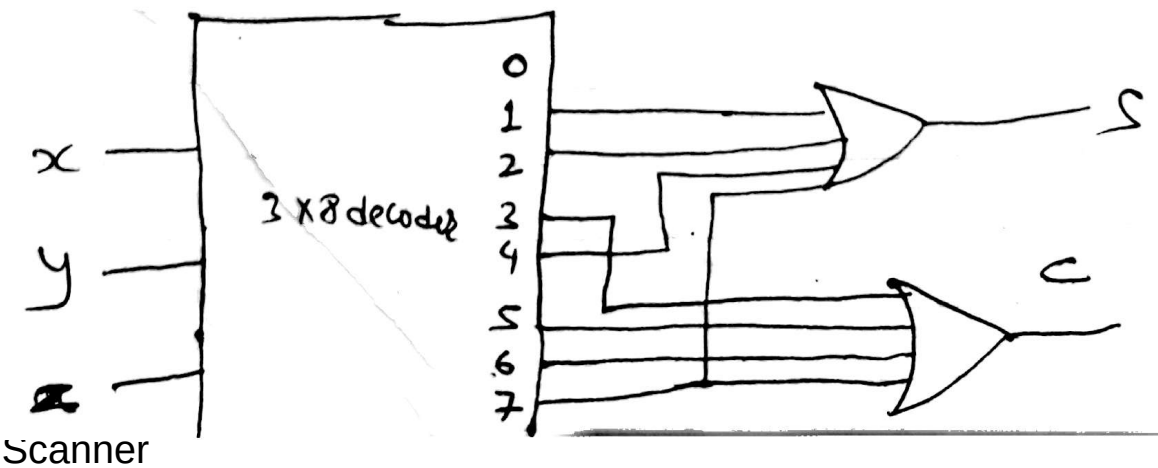
* Implement a full Adder ckt with the decoder and two OR Gates.

If x, y are two input bits, and z previous carry produces two bits of sum and carry of full Adder ckt

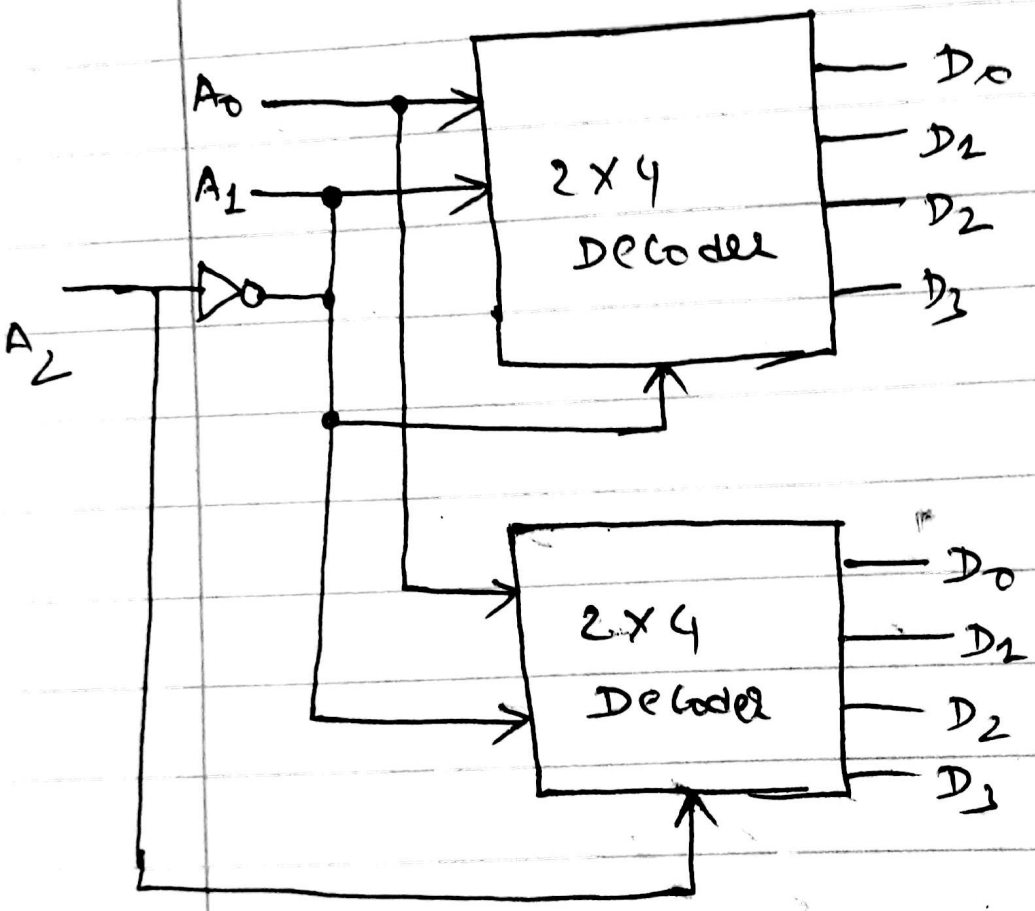
	x	y	z	Sum(s)	carry(c)
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

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

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



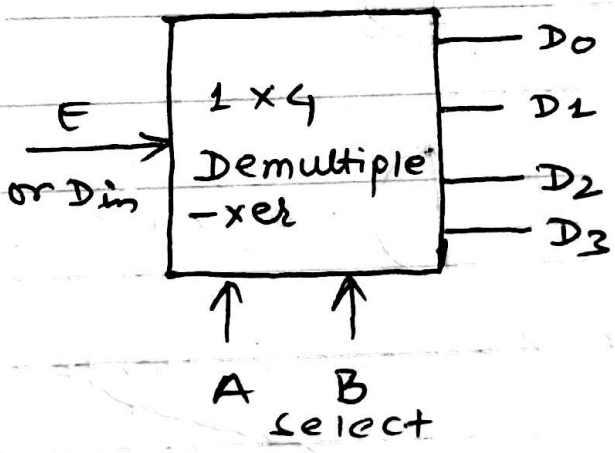
Design 3x8 Decoder using Two 2x4 Decoder



3x8 Decoder using two 2x4 decoder

* Demultiplexer :→

It is a Combinational logic ckt with one input and many output along with some control signals. It takes digital information from single line and distributes them to given number of output lines. It is also known as data distributor or one into many which means transmitting single information over a large number of channel or lines.



which is enable at E = 1

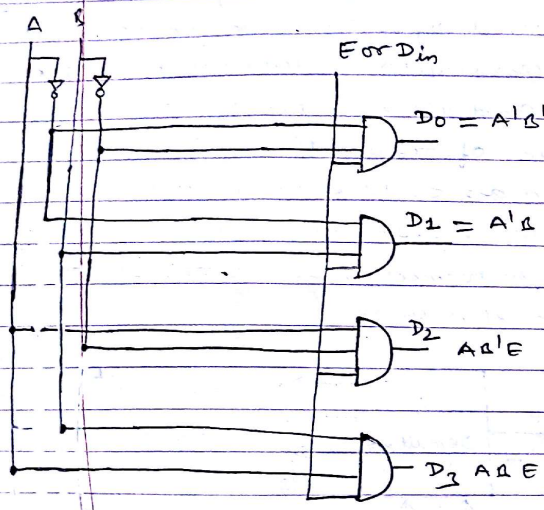
E ⇒ Data input line

A, B ⇒ selection

fig:→ Block Diagram of 1 to 4 Demultiplexer

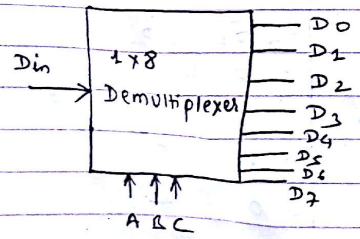
Input			output			
E or Din	A	B	D_0	D_1	D_2	D_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

$D_0 = A'B'E$ $D_1 = A'BE$ $D_2 = AB'E$, $D_3 = ABE$



Logic ckt of 1x4 Demultiplexer

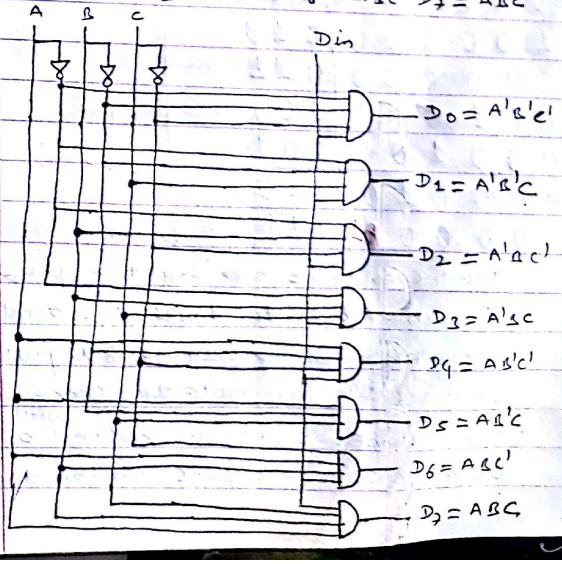
1) Design 1 to 8 Demultiplexer



Din	A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	
1	0	0	0	1	0	0	0	0	0	0	0	D_0
1	0	0	1	0	1	0	0	0	0	0	0	D_1
1	0	1	0	0	0	1	0	0	0	0	0	D_2
1	0	1	1	0	0	0	1	0	0	0	0	D_3
1	1	0	0	0	0	0	0	1	0	0	0	D_4
1	1	0	1	0	0	0	0	0	1	0	0	D_5
1	1	1	0	0	0	0	0	0	0	1	0	D_6
1	1	1	1	0	0	0	0	0	0	0	1	D_7

$D_0 = A'B'C'$; $D_1 = A'B'C$ $D_2 = A'BC'$ $D_3 = A'BC$

$D_4 = AB'C'$ $D_5 = AB'C$ $D_6 = ABC'$ $D_7 = ABC$



*Encoder :→

An Encoder is a combinational logic circuit that performs reverse operation of a Decoder. It has 2^n input lines and converts to 'n' output lines.

* Design octal to binary encoder

Input lines = 8 output lines = 3

Input								output			
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
0	1	0	0	0	0	0	0	0	0	1	1
0	0	1	0	0	0	0	0	0	1	0	2
0	0	0	1	0	0	0	0	0	1	1	3
0	0	0	0	1	0	0	0	1	0	0	4
0	0	0	0	0	1	0	0	1	0	1	5
0	0	0	0	0	0	1	0	1	1	0	6
0	0	0	0	0	0	0	1	1	1	1	7

The octal to binary encoder consist of 8 inputs and 3 outputs that generate the corresponding binary number. It is constructed with the OR gates whose inputs can be determined from the truth table. Here the output

$x = 1$ for 4, 5, 6, 7 octal digit

$y = 1$ for 2, 3, 6, 7 octal digit

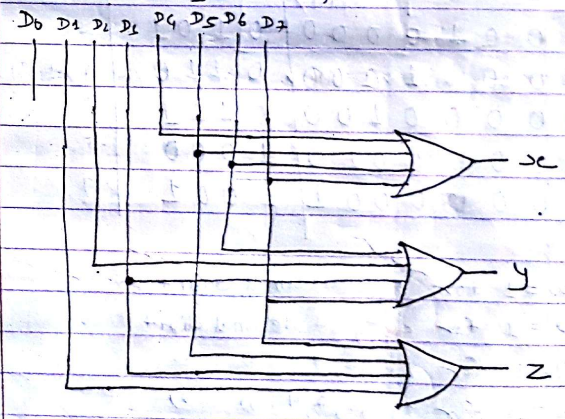
$z = 1$ for 1, 3, 5, 7 octal digit

It can be expressed as

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

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

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



* Design Decimal to BCD Encoder

This types of Encoder has ten inputs and four outputs corresponding to BCD code.

Input (Decimal)										Output (BCD)				
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	w	x	y	z	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0	2
0	0	0	1	0	0	0	0	0	0	0	0	1	1	3
0	0	0	0	1	0	0	0	0	0	0	1	0	0	4
0	0	0	0	0	1	0	0	0	0	0	1	0	1	5
0	0	0	0	0	0	1	0	0	0	0	1	1	0	6
0	0	0	0	0	0	0	1	0	0	0	1	1	1	7
0	0	0	0	0	0	0	0	1	0	1	0	0	0	8
0	0	0	0	0	0	0	0	0	1	1	0	0	1	9

w = 1 for 8, 9 decimal digit

x = 1 for 4, 5, 6, 7 decimal digit

y = 1 for 2, 3, 6, 7 " "

z = 1 for 1, 3, 5, 7, 9 " "

9+ can be expressed as

$w = D_8 + D_9$

$x = D_4 + D_5 + D_6 + D_7$

$y = D_2 + D_3 + D_6 + D_7$

$z = D_1 + D_3 + D_5 + D_7 + D_9$

Design 8×1 MUX using two 4×1 MUX

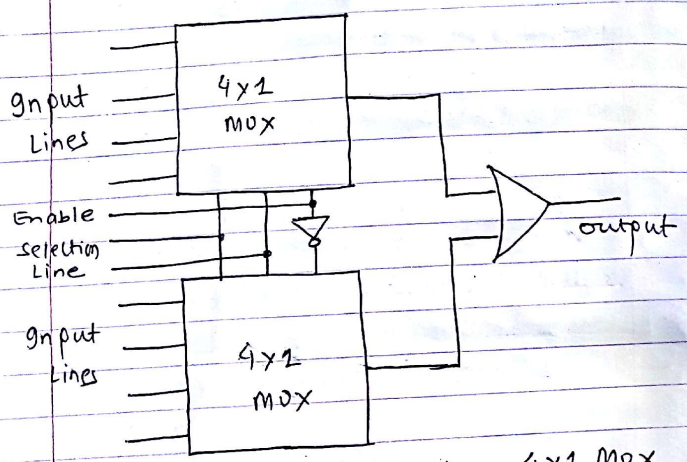


fig:→ 8×1 MUX using two 4×1 MUX

(Decoder with OR Gate)

* Multiplexer

Multiplexer is a combinational ckt that allows digital information from several sources to be routed onto single line for transmission and Data selectors. It is also known as many to one. Multiplexer selects binary information from one of many input lines and direct it to a single output line. The control and selection is done by selection line. Normally there are 2^n input lines and 'n' selection lines whose bit combination determines which input is to be selected

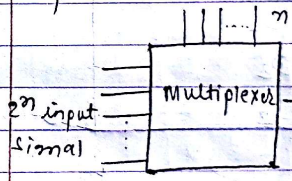


fig:→ Multiplexer (Block Diagram)

Q. Design and explain (4 to 1) 4 input multiplexer.

4 input multiplexer has four data input lines designated by D_0, D_1, D_2, D_3 and two data select bits S_1, S_0.

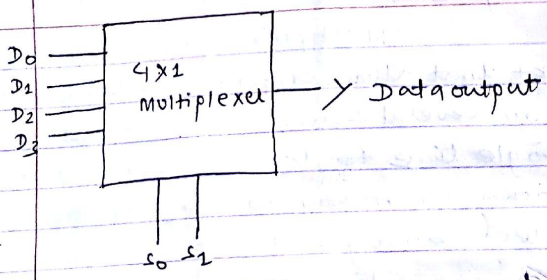


fig:→ Block Diagram of 4 x 1 Multiplexer

Data select inputs		Input selected
S_1	S_0	
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Truth table

Now, we can derive output terms in the data input and the select inputs.

$Y = D_0 S_0' S_1'$ for $S_1 = 0$ $S_0 = 0$

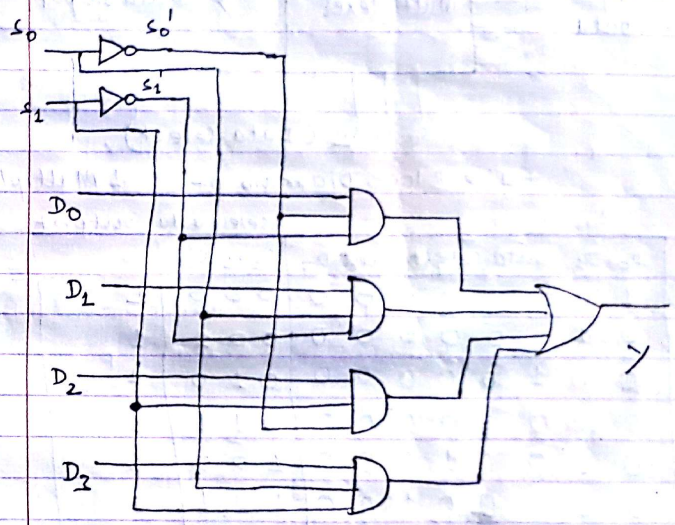
$Y = D_1 S_0 S_1'$ for $S_1 = 0$ $S_0 = 1$

$Y = D_2 S_0' S_1$ for $S_1 = 1$ $S_0 = 0$

$Y = D_3 S_0 S_1$ for $S_1 = 1$ $S_0 = 1$

Therefore the logic expression is

$$Y = D_0 S_0' S_1' + D_1 S_0 S_1' + D_2 S_0' S_1 + D_3 S_0 S_1$$



Q Design and explain (8 to 1) 8 input multiplexer.

8 input multiplexer has eight data input lines designated by D_0, D_1, D_2, D_3 D_4, D_5, D_6, D_7 and three data select bits S_0, S_1, S_2

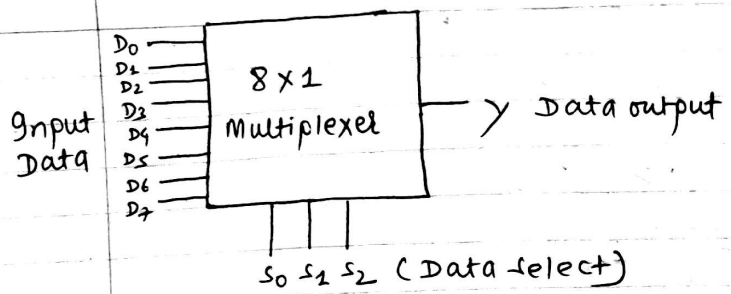
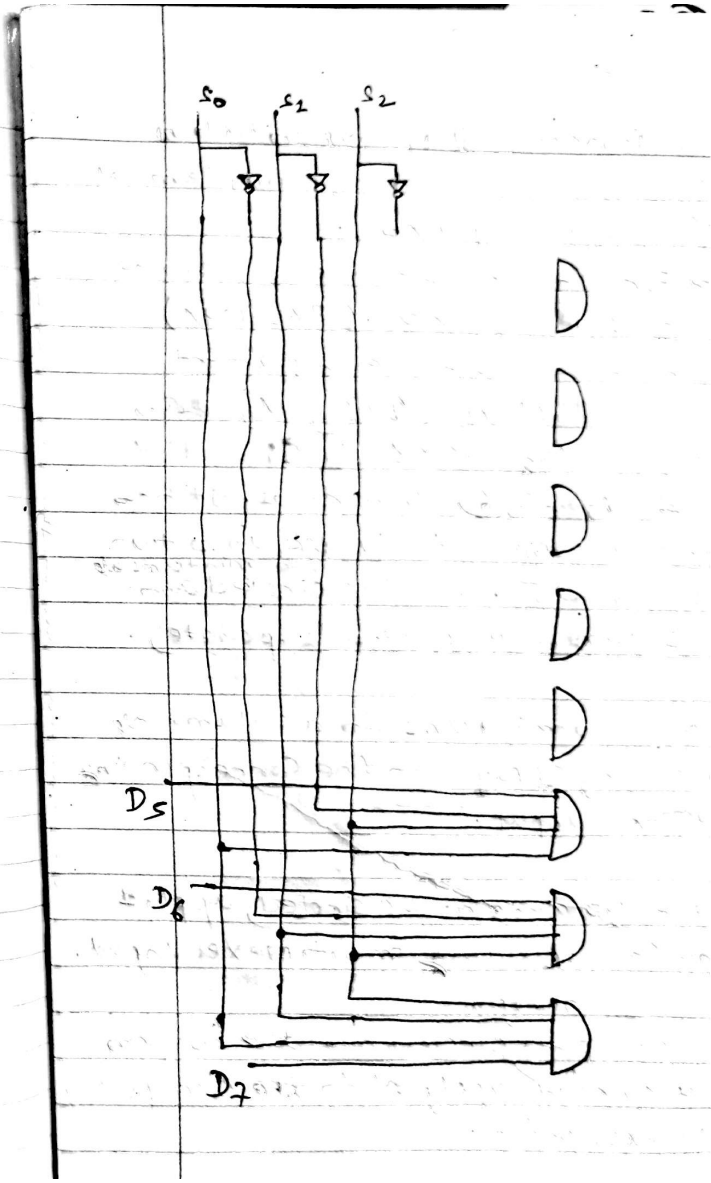


fig:⇒ Block Diagram of 8×1 Multiplexer

Input data								select data	output
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	$S_2 S_1 S_0$	Y
1	0	0	0	0	0	0	0	0 0 0	1
0	1	0	0	0	0	0	0	0 0 1	1
0	0	1	0	0	0	0	0	0 1 0	1
0	0	0	1	0	0	0	0	0 1 1	1
0	0	0	0	1	0	0	0	1 0 0	1
0	0	0	0	0	1	0	0	1 0 1	1
0	0	0	0	0	0	1	0	1 1 0	1
0	0	0	0	0	0	0	1	1 1 1	1

Therefore logic expression is,

$$Y = D_0 S_2' S_1' S_0' + D_1 S_2' S_1' S_0 + D_2 S_2' S_1 S_0' + D_3 S_2' S_1 S_0 + D_4 S_2 S_1' S_0' + D_5 S_2 S_1' S_0 + D_6 S_2 S_1 S_0' + D_7 S_2 S_1 S_0$$



* B and C for selection lines S_1 and S_0 and variable A for the input of multiplexer

Boolean function implementation

For n - variable we implement 2^{n-1} to 1 multiplexer.

eg:⇒ For 4-variable we implement 8 to 1 multiplexer. (8×1 MUX)

For 3-variable we implement 4 to 1 Multiplexer (4×1 MUX). If A, B, C are three variables, we list all the minterms in two rows. It has eight minterms and divided into two rows A and $\bar{A}$ and the circled and each column is inspected separately.

1. If the two minterms in a column are not circle, apply 0 to the corresponding multiplexer input.
2. If the two minterms are circled, apply 1 to the corresponding multiplexer input.
3. If the top is circled and the bottom is not circled, apply A' to the corresponding multiplexer input.
4. If the top minterm is not circled and bottom is circled, apply A to the corresponding multiplexer input.

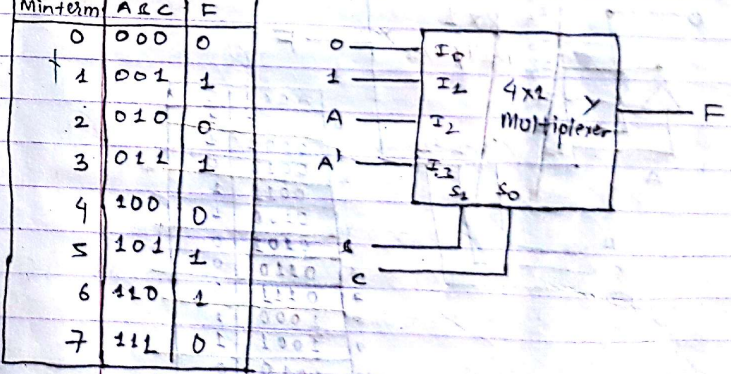
1. Implement the following Boolean function using 4×1 MUX: $F = \sum(1,3,5,6)$

we can draw implementation table for the above minterms $[2^{3-1} = 4]$

	I_0	I_1	I_2	I_3
A'	0	(1)	2	(3)
A	4	(5)	(6)	7
	0	1	A	A'

Here, we take three variable A, B, C. B and C are the selection lines (S_0, S_1) and 0, 1, A, A'. The I_0-I_3 all the input of multiplexer having value of minterm

Minterm	A B C	F
0	000	0
1	001	1
2	010	0
3	011	1
4	100	0
5	101	1
6	110	1
7	111	0



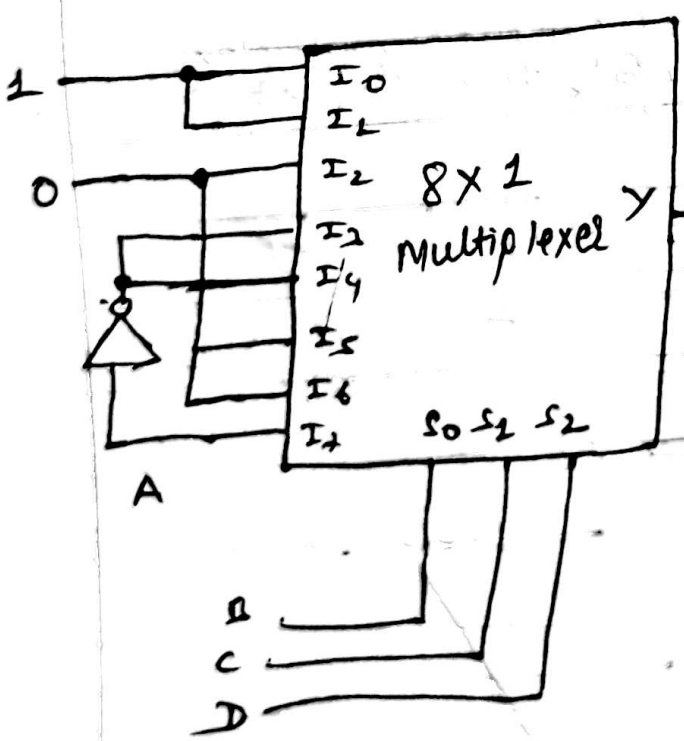
2. Implement the following Boolean function using 8 × 1 MUX : $F = \sum(0,1,3,4,8,9,15)$

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
A'	(0)	(1)	2	(3)	(4)	5	6	7
A	(8)	(9)	10	11	12	13	14	(15)
	1	1	0	A'	A'	0	0	A

Here, we take four variable A, B, C, D. B, C, D are the selection lines (S_0, S_1, S_2) which has the value of 0, 1, A and A'.

From above given value we can draw the implementation table of minterm.

$I_0 - I_7$ are the input of the multiplexer



(X)

	A	B	C	D	F
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	1

4. Implement the following Boolean function $F(w,x,y,z) = \Sigma(1,2,5,7,11,15)$ using 8 to 1 MUX.

5. Implement a full Adder ckt using two 4:1 MUX.

The addition of three binary bit is called Full Adder. If the three binary bit are A, B, C. The truth table for full adder ckt is

	A	B	C	sum(S)	carry(C)
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

$S = \Sigma(1,2,4,7)$ $C = \Sigma(3,5,6,7)$ Sum of

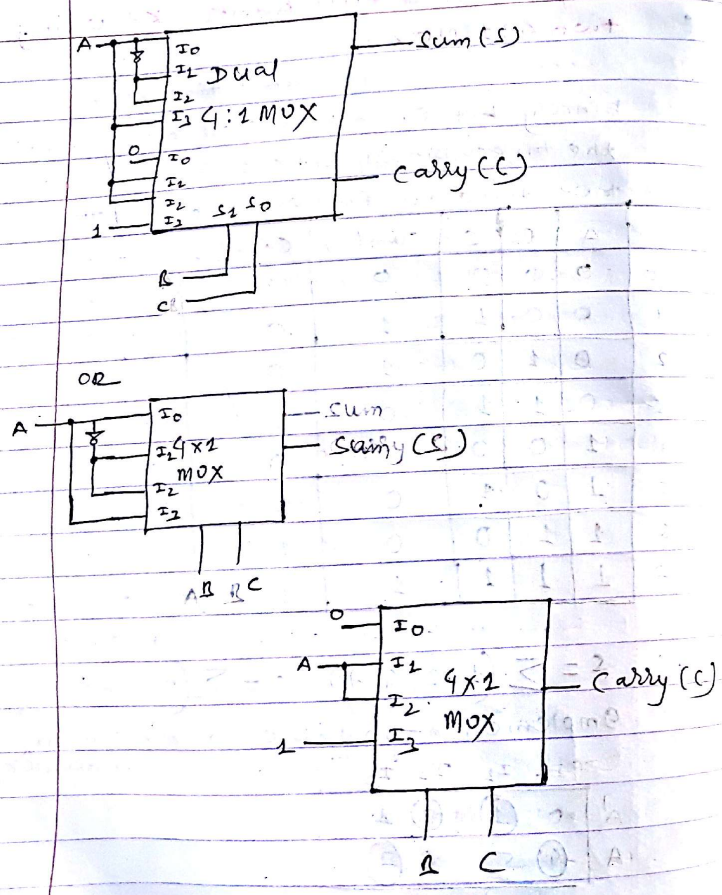
Implementation table of minterm for full Adder ckt

	I_0	I_1	I_2	I_3
A'	0	(1)	(2)	3
A	(4)	5	6	(7)
	A	A'	A'	A

full Adder ckt.

Implementation table of minterm for carry of

	I_0	I_1	I_2	I_3
A'	0	1	2	(3)
A	4	(5)	(6)	(7)
	0	A	A	1



OR

Application :→ Digital computer system.
Control units in digital system.

* Read-only Memory (ROM) :→

Read only Memory (ROM) is a memory device which stores data permanently i.e fixed set of binary information is stored. The binary information must first be specified by the user and is then embedded in the unit to form the required interconnection pattern which is done by programming. Programming specifies links that has to be fused or broken and provides required circuit path. Once a pattern is established for a ROM it remains fixed when power is turned off. ROM is used to implement a complex combinational ckt which will eliminate all the interconnecting wires.

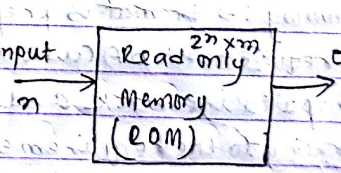


fig :→ Block diagram of ROM

Above figure shows the block diagram of Read only Memory (ROM). It consist of 'n' input lines and 'm' output

lines. The combination of input variable is called address and the combination that comes out from output lines is called a word.

* Number of input variable or number of distinct address is 2^n for 'n' variable. (Combination of selection of)
* The output will have 2^n word. $[m=2^n]$
* m will be the number of bits per word.

ROM is the combination of decoder and OR Gates i.e decoder outputs are connected with OR Gates.

* Types of ROM

There are two methods for manufacturing ROM

1) Mask programming :→ Format is prepared according to the requirement of customer. Manufacturer prepares the mask and are configured according to the requirement. For Mass production it is economical

2) It is hardware procedure which is programmed by electrically (producing) and is economical for small quantity

1. Programmable Read only memory (PROM) :→ This type of ROM is irreversible which is permanent type and cannot be changed.

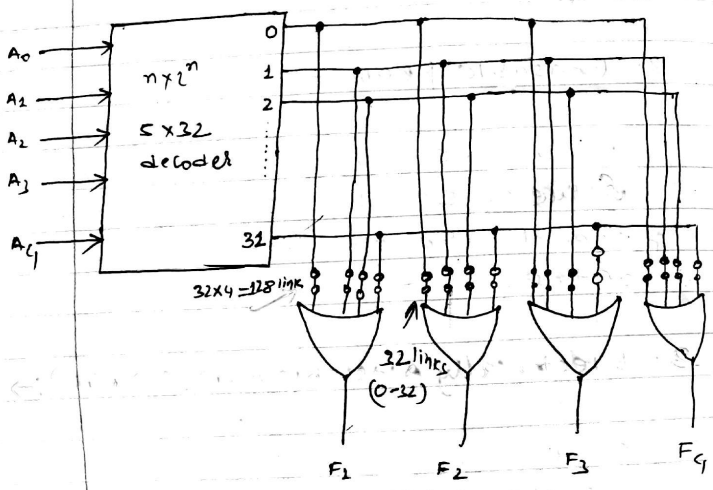
2. Erasable prom :→ Erasable PROM (EPROM) can be reconstructed to the intial state of ROM by placing under a special ultraviolet light for a given period of time.

3. Electrically Alterable ROM (EAROM) :→ ROM's that can be erased with electrical signals are called EAROM. Also known as (EEprom)

2048 $512 \times 4 = 2048$ $2^9 \times 4 = 2048$ output 4

*example construct 32×4 ROM

Here, $2^n = 32$, $n = 5$ | $m = 4$ number of output lines



2. Construct a ROM for

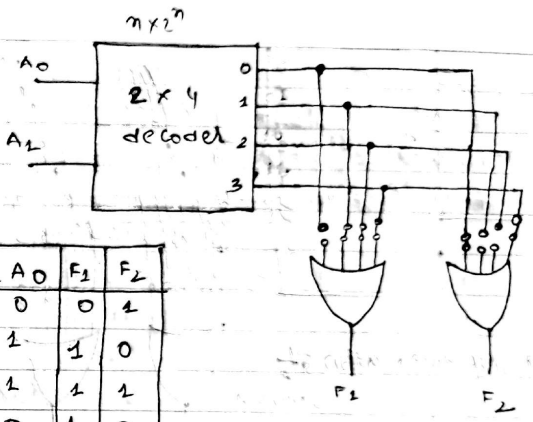
$F_1(A_1, A_0) = \sum(1, 2, 3)$

$F_2(A_1, A_0) = \sum(0, 2)$

$2^n = 4$ $n = 2$ | $m = 2^n = 4$

$2^n \times n$

1111
8421



A_1	A_0	F_1	F_2
0	0	0	1
0	1	1	0
1	1	1	1
1	0	1	0

fig: → Implementation of 4×2 ROM

3. Implement the following function using ROM.

$F_1 = \sum_m(0, 2, 5)$ $F_2 = \sum_m(3, 4, 7)$ $F_3 = \sum_m(0, 2, 5)$

$2^n = 8$ $n = 3$ $m = 8$

	A	B	C	F_1	F_2	F_3
0	0	0	0	1	0	1
1	0	0	1	0	0	0
2	0	1	0	1	0	1
3	0	1	1	0	1	0
4	1	0	0	0	1	0
5	1	0	1	1	0	1
6	1	1	0	0	0	0
7	1	1	1	0	1	0

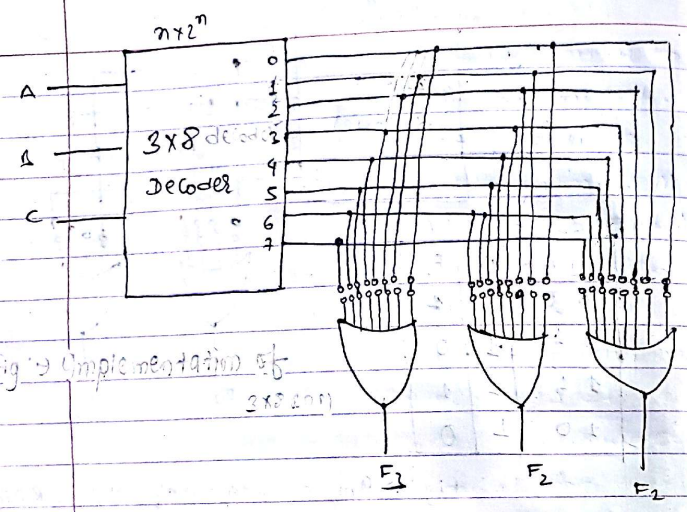


Fig :- Implementation of 3×8 ROM

4. Design a combinational ckt that accepts 3-bit number and generates an output binary number equal to the square of the input number using ROM.

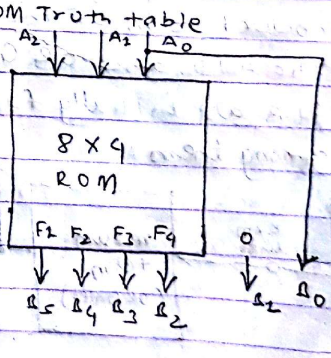
	Input			output						
				32	16	8	4	2	1	
	A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	
0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	1	1
2	0	1	0	0	0	0	1	0	0	4
3	0	1	1	0	0	1	0	0	1	9
4	1	0	0	0	1	0	0	0	0	16
5	1	0	1	0	1	1	0	0	1	25
6	1	1	0	1	0	0	1	0	0	36
7	1	1	1	1	1	0	0	0	1	49

for combinational ckt

By examining the above truth table, we have output B_0 is equal to input A_0, so their is no need to generate B_0. output B_1 is always '0', so this output is always known. Therefore only four output need to be generated with the ROM. By design observation ROM needs three input and four output.

A_2	A_1	A_0	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

ROM Truth table



* Programmable Logic Array (PLA)

PLA is similar to Read only Memory (ROM) in concept. However, PLA doesnot provide full decoding of variable as in ROM. Don't care condition has to be left in their original state but it becomes address input to ROM which will introduce waste of available equipment.

However, PLA doesnot provide full decoding of variable as in ROM, it is more economical for excessive don't care condition as it does not generate all the minterms as in the ROM.

In PLA the decoder is replaced by a group of AND gates, each of which can be programmed to generate product term of the input variables. The AND and OR Gates inside the PLA are initially fabricated with links among them

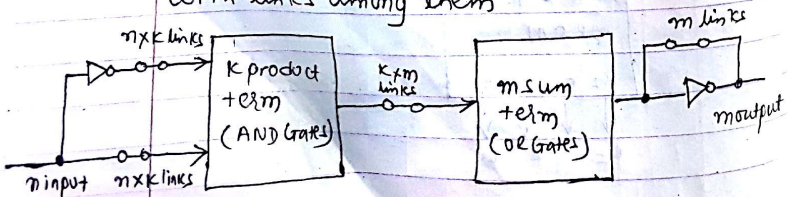


fig: → Block Diagram of PLA

The block diagram of PLA is shown above which consist of n-input, m-outputs, K-product terms and m-sum terms. The product term constitute a group of 'K' AND Gates and the sum terms constitute a group of 'm' OR gates. The size of PLA is specified by the no of inputs, the no of product terms and the no of outputs.

Therefore the number of programmed link is $2n \times K + K \times m + m$, whereas that of ROM is $2^n \times m$. The generated output function of PLA is in the AND-OR form and in the AND-OR invert form.

* Procedure to construct PLA program table

1. Construct three columns.
   a. First column lists the product term
   b. Second column specifies the path between AND Gates.
   c. Third column specifies the path between AND and the OR Gates.
2. For each output we write T (for true) for uncompleted output and C for

Complemented output by the inverter.

3. Each product term, the inputs are marked 1, 0 and –.

a. If the product term appears in its normal form, mark the corresponding input variables 1.

b. If the product term appears in its complemented form, mark the corresponding input variables 0.

c. If the variable is absent, mark with dash (–)

7. Both the true value and the complement of function should be simplified to see which one can be expressed with few product term and which one provides product term that are common to other function.

1. Consider From the given truth table derive a PLA program table and show the internal connection of the unit.

A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

From the truth table

$F_1 = AB'C' + AB'C + ABC$

A \ BC	00 B'C'	01 B'C	11 BC	10 BC'
0 A'				
1 A	1	1	1	

$F_1 = AB' + AC$

	B'C'	B'C	BC	BC'
A'	0	0	0	0
A				0

$F_1' = A' + BC'$

$F_2 = A'BC + AB'C + ABC$

A \ BC	B'C'	B'C	BC	BC'
A'			1	
A		1	1	

$F_2 = AC + BC$

$F_2' = C' + A'B'$

A \ BC				
	0	0		0
	0			0

$2n \times k + k \times m + m$
$2^2 \times 3 + 3 \times 2 + 2$
$12 + 6 + 2 = 26$ bits

n=3 (product terms)
k=3
m=2 (sum terms)

	product term	input A B C	outputs F_1 F_2
AB'	1	1 0 -	1 -
AC	2	1 - 1	1 1
BC	3	- 1 1	- 1
		T T	

we choose uncomplement function because one product term AC is in common.

PLA program Table

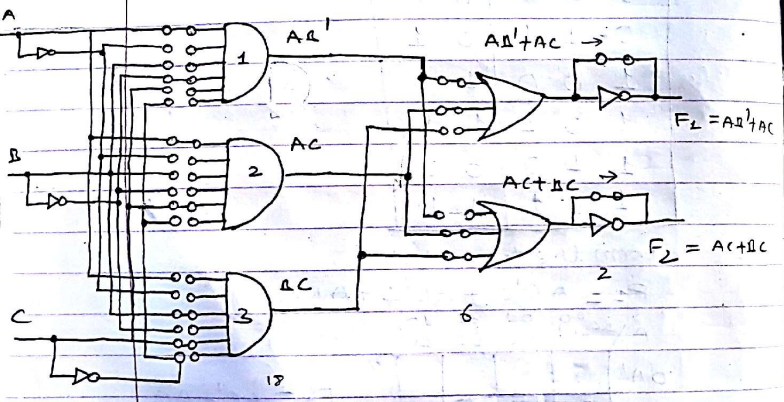


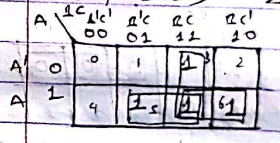
fig:⇒ Combinational ckt of internal connection of PLA.

2. A combinational circuit is defined by the functions:

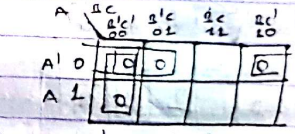
$F_1(A,B,C) = \Sigma(3,5,6,7)$
$F_2(A,B,C) = \Sigma(0,2,4,7)$

Implement the ckt with a PLA having three inputs, four product terms and two outputs.

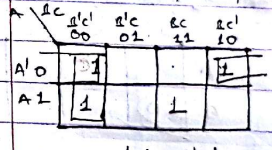
$F_1(A,B,C) = \Sigma(3,5,6,7)$
$F_2(A,B,C) = \Sigma(0,2,4,7)$



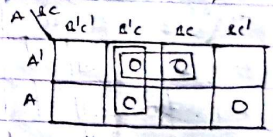
$F_2 = AC + AB + BC$



$F_1' = B'C' + A'B' + A'C'$



$F_2 = B'C' + A'C' + ABC$



$F_2'' = B'C + A'C + ABC'$

we choose complement function of F_1 and uncomplement function of F_2 because two product term are in common.

	product term	input A	B	C	output F_1	F_2
B'C	1	-	0	0	1	1
A'C'	2	0	-	0	1	1
A'B'	3	0	0	-	1	-
ABC	4	1	1	1	-	1
					C	T

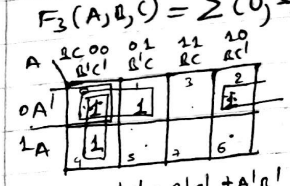
T/C

Implement the following three Boolean functions with a PLA.

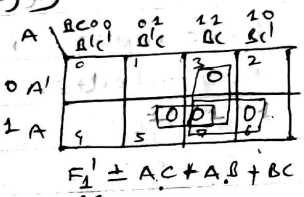
$F_1(A,B,C) = \Sigma(0,1,2,4)$

$F_2(A,B,C) = \Sigma(0,5,6,7)$

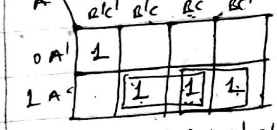
$F_3(A,B,C) = \Sigma(0,2,5,7)$



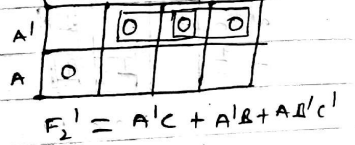
$F_1 = A'C' + B'C' + A'B'$



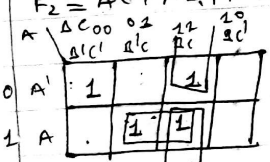
$F_1' = AC + AB + BC$



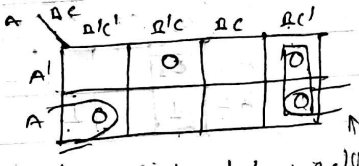
$F_2 = AC + AB + A'B'C'$



$F_2' = A'C + A'B + AB'C'$



$F_3 = AC + A'B'C' + BC$



$F_3' = AC' + A'B'C + BC'$

Product term	Input A B C	Output F_1	F_2	F_3
AB 1	1 1 -	1	1	—
AC 2	1 - 1	1	1	1
BC 3	- 1 1	1	-	1
$A'B'C'$ 4	0 0 0	-	1	1
		C	T	T

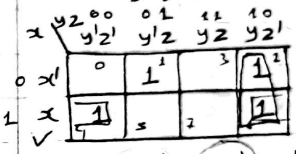
4. Implement the given four Boolean function using 8×4 PLA

$A(x,y,z) = \Sigma(1,2,4,6)$

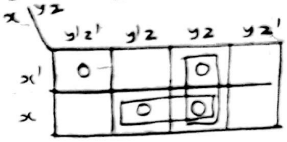
$B(x,y,z) = \Sigma(0,1,6,7)$

$C(x,y,z) = \Sigma(2,6)$

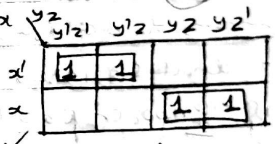
$D(x,y,z) = \Sigma(1,2,3,5,7)$



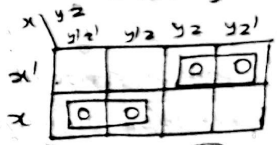
$A = xz' + yz' + x'y'z$



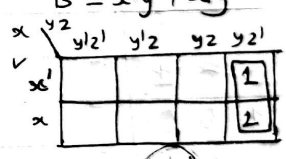
$A' = xz + yz + x'y'z'$



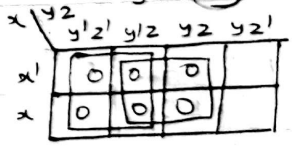
$B = x'y' + xy$



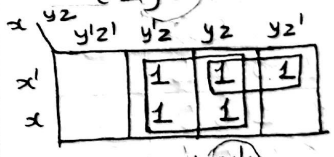
$B' = xy' + x'y$



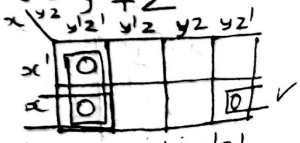
$C = yz'$



$C' = y' + z$



$D = z + x'y$



$D' = xz' + y'z'$

xz'

yz'

$x'y'z$

$x'y$

	Product term	Input x y z	Output A B C D
xz'	1	1 - 0	1 - - 1
yz'	2	- 1 0	1 - 1 -
$x'y'z$	3	0 0 1	1 - - -
$x'y'$	4	0 0 -	- 1 - -
xy	5	1 1 -	- 1 - -
$y'z'$	6	- 0 0	- - - 1
		T T T C	T/C

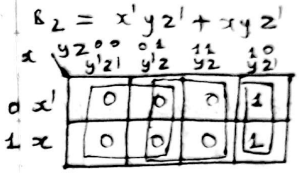
5) Derive a PLA program table for the Combinational ckt that squares 3 bit number minimize the number of product term

	A_2 A_1 A_0 x y z	B_5 B_4 B_3 B_2 B_1 B_0 (32 16 8 4 2 1)
0	0 0 0	0 0 0 0 0 0
1	0 0 1	0 0 0 0 0 1
2	0 1 0	0 0 0 1 0 0
3	0 1 1	0 0 1 0 0 1
4	1 0 0	0 1 0 0 0 0
5	1 0 1	0 1 1 0 0 1
6	1 1 0	1 0 0 1 0 0
7	1 1 1	1 1 0 0 0 1

$n = 3$
$k = 5$
$m = 9$

Here $B_0 = z$ and $B_1 = 0$

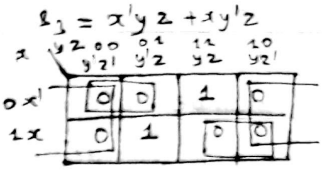
$B_2 = x'yz' + xyz'$



$B_2 = yz'$

$B_2' = y' + z$

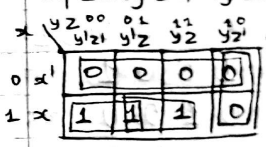
$B_3 = x'yz + xy'z$



$B_3 = x'yz + xy'z$

$B_3' = z' + xy + x'y'$

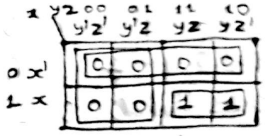
$B_4 = xy'z' + xy'z + xyz$



$B_4 = xy' + xz$

$B_4' = x' + yz'$

$B_5 = xyz' + xyz$



$B_5 = xy$

$B_5' = y' + x'$

Product term	Input x y z	Output B_5	B_4	B_3	B_2
yz' 1	- 1 0	-	1	-	1
$x'yz$ 2	0 1 1	-	-	1	-
$xy'z$ 3	1 0 1	-	-	1	-
xy 4	1 1 -	1	-	-	-
x' 5	0 - -	-	1	-	-
		T	C	T	T

T/C

6. Design a PLA programmable for the combinational ckt that squares 4 bit number minimize the number of product term.

	8 4 2 1 w x y z	128 64 32 16 8 4 2 1 S_7 S_6 S_5 S_4 S_3 S_2 S_1 S_0	
0	0 0 0 0	0 0 0 0 0 0 0 0	0
1	0 0 0 1	0 0 0 0 0 0 0 1	1
2	0 0 1 0	0 0 0 0 0 1 0 0	4
3	0 0 1 1	0 0 0 0 1 0 0 1	9
4	0 1 0 0	0 0 0 1 0 0 0 0	16
5	0 1 0 1	0 0 0 1 1 0 0 1	25
6	0 1 1 0	0 0 1 0 0 1 0 0	36
7	0 1 1 1	0 0 1 1 0 0 0 1	49
8	1 0 0 0	0 1 0 0 0 0 0 0	64
9	1 0 0 1	0 1 0 1 0 0 0 1	81
10	1 0 1 0	0 1 1 0 0 1 0 0	100
11	1 0 1 1	0 1 1 1 1 0 0 1	121
12	1 1 0 0	1 0 0 1 0 0 0 0	144
13	1 1 0 1	1 0 1 0 1 0 0 1	169
14	1 1 1 0	1 1 0 0 0 1 0 0	196
15	1 1 1 1	1 1 1 0 0 0 0 1	225

$S_0 = z$ & $S_1 = 0$

$S_2 = \Sigma(2, 6, 10, 14)$

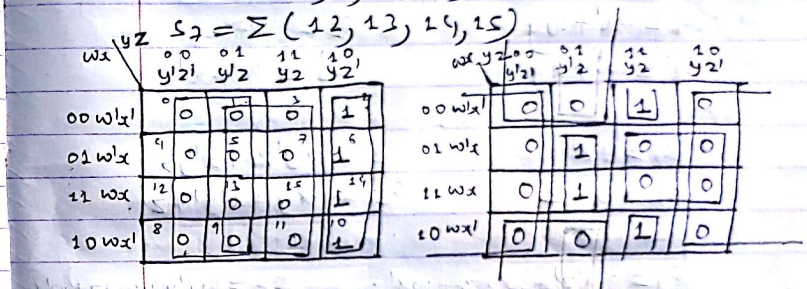
$S_3 = \Sigma(3, 8, 11, 13)$

$S_4 = \Sigma(4, 5, 7, 9, 11, 12)$

$S_5 = \Sigma(6, 7, 10, 11, 13, 15)$

$S_6 = \Sigma(8, 9, 10, 11, 14, 15)$

$S_7 = \Sigma(12, 13, 14, 15)$

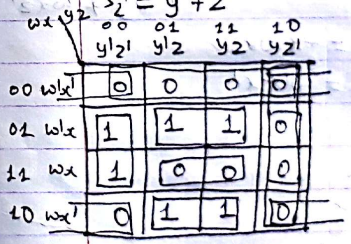


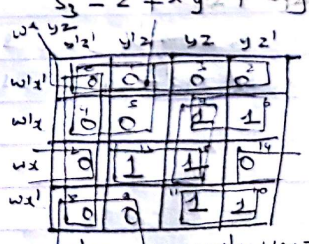
$S_2 = yz' + xy'$

$S_2' = y' + z$

$S_3 = xy'z + x'yz$

$S_3' = z' + x'y' + xy$



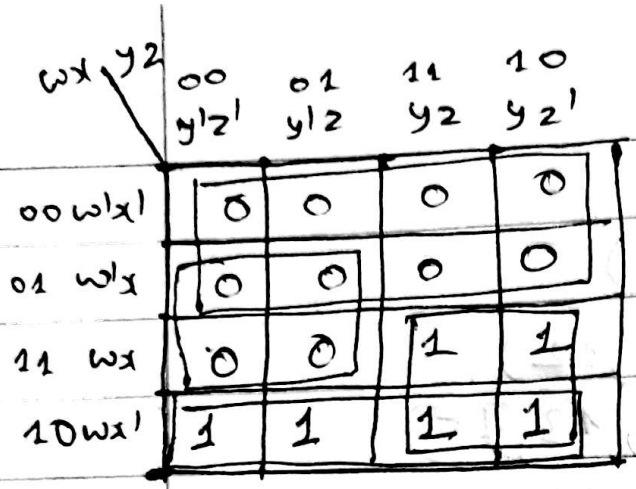


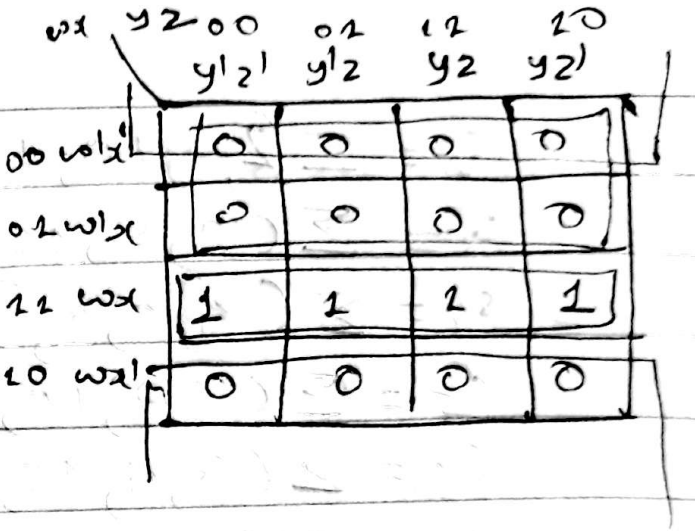
$S_4 = xy'z' + w'xz + wx'z$

$S_4' = x'z' + wxz + yz' + wx'$

$S_5 = wx'y + wxz + w'xy$

$S_5' = w'x' + y'z' + x'y' + w'y' + wxz'$





$S_6 = wx' + wy$

$S_6' = w' + xy'$

$S_7 = wx$

$S_7' = w' + x'$

$S_2 = yz'$

$S_2' = y' + z$

$S_3 = xy'z + x'yz$

$S_3' = xy + x'y' + z'$

$S_4 = xy'z' + w'xz + wx'z$

$S_4' = x'z' + wxz + yz' + w'x'$

$S_5 = wx'y + wxz + w'xy$

$S_5' = w'x' + y'z' + x'y' + w'y' + wxz'$

$S_6 = wx' + wy$

$S_6' = w' + xy'$

$S_7 = wx$

$S_7' = w' + x'$