JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

B.TECH. INFORMATION TECHNOLOGY (2017-18)

Subject: DIGITAL LOGIC DESIGN

(1) ***OBJECTIVES AND RELEVANCE***

This course introduces the basic concepts logic gates which is the most useful for designing the IC Technology in engineering discipline. The emphasis of this course is laid on the basic analysis of digital circuits.

(2) ***SCOPE***

The scope of this subject is to provide an insight into designing the DTl, TTL ECL circuits which are useful in digital devices. This concept is more useful in further applications of minimizing the complexity of logic circuits.

(3) ***PREREQUISITES***

This subject recommends continuous practice of various simple arithmetic operations like addition, subtraction. It needs requisite knowledge about assuming and designing the logic circuits. How to reduce the size and complexity of the digital circuits.

(4.1) ***SYLLABUS - JNTU***

B.Tech. II Year I Sem.

Course Objectives:

- To understand basic number systems, codes and logical gates.
- To understand the concepts of Boolean algebra.
- To understand the use of minimization logic to solve the Boolean logic expressions.
- To understand the design of combinational and sequential circuits.
- To understand the state reduction methods for Sequential circuits.
- To understand the basics of various types of memories.

Course Outcomes:

- Able to understand number systems and codes.
- Able to solve Boolean expressions using Minimization methods. Able to design the sequential and combinational circuits.
- Able to apply state reduction methods to solve sequential circuits.

UNIT - I

Digital Systems, Binary Numbers, Number base conversions, Octal, Hexadecimal and other base numbers, complements, signed binary numbers, Floating point number representation, binary codes, Error detection and correction, binary storage and registers, binary logic, Boolean algebra and logic gates , Basic theorems and properties of Boolean Algebra, Boolean functions, canonical and standard forms, Digital Logic Gates.

UNIT - II

Gate–Level Minimization, The K-Map Method, Three-Variable Map, Four-Variable Map, Five-Variable Map , sum of products , product of sums simplification, Don’t care conditions , NAND and NOR implementation and other two level implementations, Exclusive-OR function.

UNIT - III

Combinational Circuits (CC), Analysis procedure, Design Procedure, Combinational circuit for different code converters and other problems, Binary Adder-Subtractor, Decimal Adder, Binary Multiplier, Magnitude Comparator, Decoders, Encoders, Multiplexers, De-multiplexers.

UNIT - IV

Synchronous Sequential Circuits, Latches, Flip-flops, analysis of clocked sequential circuits, Registers, Shift registers, Ripple counters, Synchronous counters, other counters. Asynchronous Sequential Circuits -Introduction, Analysis procedure, Circuits with latches, Design procedure, Reduction of state and follow tables, Race- free state assignment, Hazards.

UNIT - V

Memory: Introduction, Random-Access memory, Memory decoding, ROM, Programmable Logic Array, Programmable Array Logic, Sequential programmable devices.
Register Transfer and Microoperations - Register Transfer Language, Register Transfer, Bus and Memory Transfers,
Arithmetic Microoperations, Logic Microoperations, Shift Microoperations, Arithmetic Logic Shift Unit.

(4.2) ***SYLLABUS - GATE***

UNIT I
Logic gates; Boolean algebra,

UNIT II
Minimization of Boolean functions;

UNIT III
Combinatorial circuits: arithmetic circuits, code converters, multiplexers, decoders

UNIT IV
Sequential circuits: latches and flip-flops, counters and shift-registers.

UNIT V

(4.3) ***SYLLABUS - IES***

UNIT I

Transistor as a switching element, Boolean algebra, simplification of Boolean functions

UNIT II
K-map and applications

UNIT III
Combinational logic Circuits; Half adder, Full adder; Digital comparator, Multiplexer Demultiplexer

UNIT IV

Flip flops. R-S, J-K, D and T flip flops; Different types of counters and registers

UNIT V
---

(5) ***SUGGESTED BOOKS***

TEXT BOOKS:

1. Digital Design, M. Morris Mano, M.D.Ciletti, 5th edition, Pearson.(Units I, II, III, IV, Part of Unit V)
2. Computer System Architecture, M.Morris Mano, 3rd edition, Pearson.(Part of Unit V)

REFERENCE BOOKS:

1. Switching and Finite Automata Theory, Z. Kohavi, Tata McGraw Hill.
2. Fundamentals of Logic Design, C. H. Roth, L. L. Kinney, 7th edition, Cengage Learning.
3. Fundamentals of Digital Logic & Micro Computer Design, 5TH Edition, M. Rafiquzzaman, John Wiley.

(6) *WEBSITES*

1. www.ieee.org
2. www.2dix.com
3. www.xillinx.com
4. www.cdac.com
5. www.vlsi.edu
6. www.vlsi.iitkgp.ernet.in
7. www.educypedia.be/electronic/digital.com
8. www.iitb.ac.in
9. www.iitm.ac.in
10. www.iitr.ac.in
11. www.iitg.ernet.in
12. www.bits-pilani.ac.in
13. www.metorgraphics.com
14. www.vlsi-reasearch.com
15. www.iisc.ernet.in
16. www.samsung.com
17. www.vedaiit.com

(7) *EXPERT DETAILS*

The Expert Details which have been mentioned below are only a few of the eminent ones known Internationally, Nationally and Locally. There are a few others known as well.

INTERNATIONAL

1. **Prof. K Subbarangaiah** is Director of VEDA IIT, Hyderabad

NATIONAL

1. Dr.K. LAL KISHORE, Ph.D., MIEEE, FIETE, MISTE, MISHM, JNTU, Hyderabad
2. Mr .Sundaram, AGM, CAD R&D, ECIL, Hyderabad..
3. Mr. Rajendra naik, Asst Prof, Dept of ECE, Osmania University, Hyderabad.

REGIONAL

1. Dr. N.S.Murthy, Professor and Head Dept. of ECE, REC,Warangal - 506004 (India) email: nsm@recw.ernet.in
2. ***S.G Vinayaka Prasad, Sr. App. Engineer, Silicon Micro Systems***
3. DR. M. Madhavi Latha, JNTU, Hyderabad
4. Dr. Sarat Chandra Babu, Centre Head C-DAC, Hyderabad email: Sarat_chandra@hotmail.com

(8) *JOURNALS*

INTERNATIONAL

1. IEEE transactions on DIGITAL DESIGNS
2. IEEE proceedings circuits, devices and systems
3. International journal of circuit designs and applications
4. IEEE transactions on IC design
5. VSI vision

NATIONAL

1. DIGITAL DESGIN MAGAZINE
2. JOURNAL FOR DIGITAL SYSTEMS
3. IBM system magazine

UNIT – 1

Number Systems and Codes

philosophy of Number Systems

- Number System is a basis for counting various items
  We are familiar with decimal number system with its 10 digits : 0, 1, 2, 3, 4, ... , 9.
- But modern computers communicate and operate with binary numbers which use only 0 and 1.

  Ex : 18 — 10010

  In decimal it is represented by two digits where as in binary 5 digits.

  ∴ If decimal quantities are represented in binary form they take more digits.
- for large decimal numbers people have to deal with very large binary strings and therefore, they do not like working with binary numbers. This fact give rise to there new number Systems.

  octal, Hexadecimal and Binary Coded decimal
- These number Systems represent binary number in a Compressed form.

  ∴ these number Systems are now widely used to compress long strings of binary numbers.

1.1 Decimal number Systems

— In this we can express any number in units, hundred, thousand and so on.

Ex: 5678.9

: 5000+600+70+8+0.9 = 5678.9

Can also be written as 5678.9_{10} where 10 subscript indicates the radix or base.

— In power of 10 we can write

$$5\times10^{3} + 6\times10^{2} + 7\times10^{1} + 8\times10^{0} + 9\times10^{-1}$$

— The left most bit which has the greatest weight is called most significant bit (MSB)

— The Right most bit which has the least weight is called least significant bit (LSB)

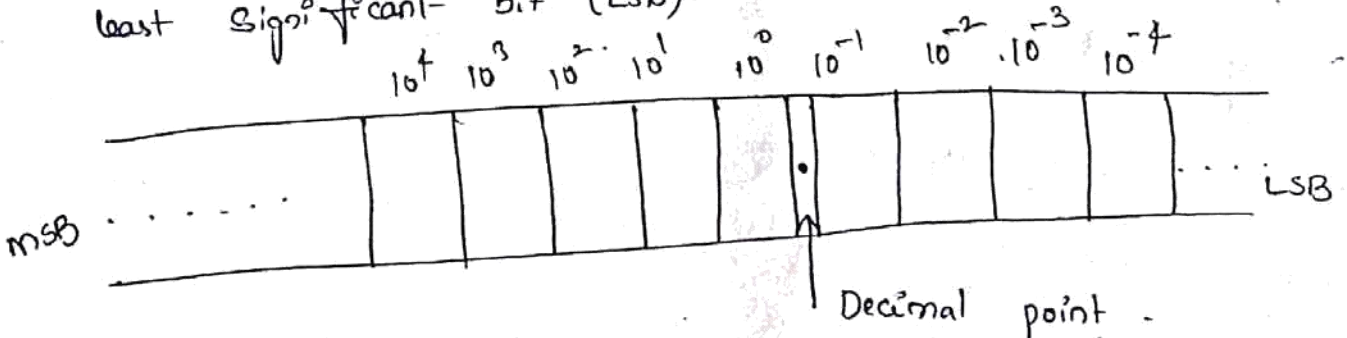


fig:- Decimal position values as powers of 10

Ex:- Represent decimal number 98.72 in power of 10

$$9\times10^{1} + 8\times10^{0} + 7\times10^{-1} + 2\times10^{-2}$$

for 9 power of 10 is 1

for 8 power of 10 is 0

for 7 power of 1/10 is -1

for 2 power of 1/100 is -2

Binary Number System :-

Binary system with its two digits is a base-two system. The two binary digits are 1 and 0. Each binary digits are known as bit.

- In binary system weight is expressed as power of 2.

Ex: Represent binary number 1101.101 in power of 2 and find its decimal Equivalent

$$N = 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0 + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3}$$

$$= 8 + 4 + 0 + 1 + \tfrac{1}{2} + 0 + \tfrac{1}{8}$$

$$= (13.625)_{10}$$

1.3 Octal Number System :-

It uses first eight digits of decimal number system 0, 1, 2, 3, 4, 5, 6, and 7. As it uses 8 digits, its base is 8

Ex: Represent octal number 567 in power of 8 and find its decimal Equivalent

$$5\times 8^2 + 6\times 8^1 + 7\times 8^0 = 5\times 64 + 6\times 8 + 7\times 1 = (375)_{10}$$

1.4 Hexadecimal Number System :-

- Its base is 16, it is having 16 digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Since its base is a power of 2, (2^4) it is easy to convert hexadecimal to binary numbers and vice versa.

- It is useful for human communications with a computer
- Each hexadecimal digit represents a group of four binary digits called nibbles.

Relation b/n decimal, binary, Hexadecimal, octal:-

Decimal	Binary	Hexadecimal	octal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17

Represent
%s

Represent hexadecimal number 3FD in power of 16 and find its decimal equivalent-

$$3\times16^2 + F\times16^1 + D\times16^0$$

3 F D

$$3\times256 + 15\times16 + 13\times1$$

$$= 768 + 240 + 13 = (1021)_{10}.$$

1.5. Counting in Radix (Base) :—

— In general, a number represented in radix r, has r characters in its set and r can be any value. They are 0, 1, ... r-1.

Radix	characters in set
2	0, 1
3	0, 1, 2
4	0, 1, 2, 3
⋮	
7	0, 1, 2, 3, 4, 5, 6,
8 (octal)	0, 1, 2, 3, 4, 5, 6, 7
⋮	⋮
(10) (Decimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
⋮	
(16) (Hexadecimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Ex: find decimal Equivalent of $(231.23)_4$

$= 2\times 4^2 + 3\times 4^1 + 1\times 4^0 + 2\times 4^{-1} + 3\times 4^{-2}$

$= 32 + 12 + 1 + 0.5 + 0.1875$

$= (45.6875)_{10}$

Ex: Count from 0 to 9 in Radix 5

Sol Radix 5 has 5 characters.

0, 1, 2, 3, 4, 10, 11, 12, 13, 14

5 in radix $(10)_5$ —— $(5)_{10}$

$1\times 5^1 + 0\times 5^0 = 5+0 = 5_{10}$

$(11)_5$ —— $(6)_{10}$

$(11)_5 = 1\times 5^1 + 1\times 5^0 = 5+1 = 6_{10}$

$(7)_{10}$ — $(12)_5 \Rightarrow 1\times 5^1 + 2\times 5^0 = 7_{10}$

$(8)_{10}$ —— $(13)_5 \Rightarrow 1\times 5^1 + 3\times 5^0 = 8_{10}$

$(9)_{10}$ —— $(14)_5 \Rightarrow 1\times 5^1 + 4\times 5^0 = 9_{10}$

Ex: Represent $(13)_{10}$ in octal

$(13)_{10}$ —— $(?)_8$

$8 \mid 13$

$1 - 5$ ↑ $= (15)_8$

check $(15)_8$ —— $(?)_{10}$

$1\times 8^1 + 5\times 8^0$

$= 8+5 = (13)_{10}$

Number System Conversion

1 Binary to octal :—

— for octal Numbers base is 8 and the base for binary is 2.

i.e the base for octal number is the third power of base for binary numbers

∴ By grouping digits of 3 digits of binary numbers and then converting Each group digits to its octal Equivalent

Ex: octal to Binary Convert $(111\ 101\ 100)_2$ to octal

— $(7\ 5\ 4)_8$.

2.2 Octal to Binary

— Each digit of the octal number is individually converted to its binary Equivalent to get octal to binary conversion of the number

Ex: $(634)_8$ — $(?)_2$

$= (110\ 011\ 100)_2$

Ex: $(725.63)_8$ — $(?)_2$

$(111010101.110011)_2$

2.3 Binary to Hexadecimal

— Base for hexadecimal is 16 and the base for binary is 2.

The base for hexadecimal number is the fourth power of the base for binary numbers

∴ By grouping 4 digits of binary number and converting each group digit to its hexadecimal Equivalent, we can convert binary number to its hexadecimal Equivalent.

Ex: Convert $(1101\ 1000\ 1001\ 1011)_2$ to hexadecimal Equivalent

$(D\ 8\ 9\ B)_H$

2.4 Hexadecimal to Binary Conversion:

— Each digit of the hexadecimal number is individually converted to its binary Equivalent to get hexadecimal to binary conversion of the number.

Ex: Convert $(5A9.B4)_H$ to binary

$(0101\ 1010\ 1001.\ 1011\ 0100)_2$

Ex: Convert $(3FD)_H$ to binary

$(0011\ 1111\ 1101)_2$.

2.5 octal to hexadecimal:

1. Convert octal to binary number
2. Convert binary number to its hexadecimal Equivalent.

Ex: $(615)_8$ —— $(\ )_{16}$

(i) $(\underline{110}\ \underline{001}\ \underline{101})_2 \rightarrow (\overline{11})\ (\overline{01}) = (18D)_H$

Hexadecimal to octal Conversion

(i) Convert hexadecimal number to its binary Equivalent

(ii) Convert binary number to its octal Equivalent.

Ex: $(25B)_{16}$ ——— $(?)_8$

$(0010\ 0101\ 1011)_2$

001 001 011 011

1 1 3 3 = $(1133)_8$

Converting any Radix to Decimal

→ In general numbers can be represented as

$$N = A_{n-1}R^{n-1} + A_{n-2}R^{n-2} + \dots + A_1R^1 + A_0R^0 + A_{-1}R^{-1} + A_{-2}R^{-2} + \dots + A_{-m}R^{-m}$$

N = Number in Decimal

A = Digit

R = Radix

n = The no. of digits in the integer portion of the number

m = The no of digits in the fractional portion of number

Ex: $(1101.1)_2$ ——— (?)

$= 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1}$

$= 8 + 4 + 0 + 1 + 0.5$

$= (13.5)_{10}$

Ex: $(475.25)_8$ ——— $(?)_{10}$

$$N = 4\times8^2 + 7\times8^1 + 5\times8^0 + 2\times8^{-1} + 2\times8^{-2}$$

$$= 256 + 56 + 5 + 0.25 + 0.078125$$

$$= (317.32813)_{10}$$

Ex: $(3102.12)_4$ ——— $(?)_{10}$

$$N = 3\times4^3 + 1\times4^2 + 0\times4^1 + 2\times4^0 + 1\times4^{-1} + 2\times4^{-2}$$

$$= 192 + 16 + 0 + 2 + 0.25 + 0.125$$

$$= (210.375)_{10}$$

Ex: $(614.15)_7$ ——— $(?)_{10}$

$$= 6\times7^2 + 1\times7^1 + 4\times7^0 + 1\times7^{-1} + 5\times7^{-2}$$

$$= 294 + 7 + 4 + 0.142857 + 0.102$$

$$= (305.24486)_{10}$$

Conversion of Decimal Numbers to any Radix Number

Two Steps:

Step 1: Convert integer part. — This is done by Successive division method.

Step 2: Convert fractional part — This is done by Successive multiplication method.

(i) Successive Division for integer part:—

Ex: $(37)_{10}$ ——— $(?)_2$

2 | 37
2 | 18 – 1
2 | 9 – 0
2 | 4 – 1
2 | 2 – 0
1 – 0 ↑ $= (100101)_2$ $32+4+1=37$

Ex: $(3509)_{10}$ ——— $(\ )_{16} - (\ )_H$

16 | 3509
16 | 219 – 5
13 – 11 ↑ $= (DB5)_{16}$

Ex: $(54)_{10}$ — $(?)_4$

4 | 54
4 | 13 – 2
3 – 1 ↑ $= (312)_4$

(ii) Successive multiplication for fractional part:

— The number to be converted is multiplied by radix of new number, producing a product that an integer part and a fractional part.

— The integer part of the product becomes a numberal in the new radix

— The fractional part is again multiplied by the radix
the process is repeated until fractional part reaches 0
the new radix number is carried out to sufficient digits
The integer part of each product is read downward to repr
the new radix number.

Ex: $(0.8125)_{10}$ —— $(?)_2$

Fraction	Radix	Result	Recorded carry.	
0.8125×2	$=$	$1.625 = 0.625$	1	MSB
0.625×2	$=$	$1.25 = 0.25$	1	
0.25×2	$=$	$0.5 = 0.5$	0	
0.5×2	$=$	$1.0 = 0.0$	1	LSB

$(0.1101)_2$

$(0.1101)_2$ — $(0.8125)_{10}$

Ex: $(0.95)_{10}$ —— $(?)_2$

$0.95 \times 2 = 1.9 = 0.9$	1	— 0.8 is repeated and if we multiply further, we will get repeated sequence. If we stop here, we get 7 binary digits (0.1111001)
$0.9 \times 2 = 1.8 = 0.8$	1	
$0.8 \times 2 = 1.6 = 0.6$	1	
$0.6 \times 2 = 1.2 = 0.2$	1	
$0.2 \times 2 = 0.4 = 0.4$	0	
$0.4 \times 2 = 0.8 = 0.8$	0	
$0.8 \times 2 = 1.6 = 0.6$	1	

$(0.1289062)_{10}$ —— $(?)_{16}$

0.1289062×16 — 2.0625 2

0.0625×16 — 1.0 1

$(0.21)_{16}$

Ex: Convert $(24.6)_{10}$ — $(?)_2$

Step1: Separate out integer and fraction part.

Integer part : 24, fraction part : 0.6

Step2: find Equivalent binary number for integer part.

2 | 24
2 | 12 — 0
2 | 6 — 0
2 | 3 — 0
1 — 1

$(11000)_2$

Step3: find Equivalent binary number for fractional

$0.6 \times 2 = 1.2$ 0.2 1

$0.2 \times 2 = 0.4$ 0.4 0

$0.4 \times 2 = 0.8$ 0.8 0

$0.8 \times 2 = 1.6$ 0.6 1

$0.6 \times 2 = 1.2$ 0.2 1

$(0.10011)_2$.

$\therefore (11000.10011)_2$

Questions

1) Convert $(725.25)_8$ to dec, bin, hex dec.

Sol (i) $= 7\times8^2 + 2\times8^1 + 5\times8^0 + 2\times8^{-1} + 5\times8^{-2}$

$= 448 + 16\times5 + \frac{1}{4} + \frac{5}{64}$

$= 469 + 0.25 + 0.078125$

$= (469.328125)_{10}$

(ii) $(725.25)_{10}$ — ( )$_2$

$(111010101.010101)_2$

(iii) 0001 1010 1010 1 . 0101 0100

$(1\ D\ 5\ .\ 5\ 4)_{16}$

Weighted Codes:

In this Each digit position of the number represents a specific weight.

Ex: If number is 567 then wt of 5 is 100
wt of 6 is 10
wt of 7 is 1

Ex: Binary, BCD

- BCD is an abbreviation for binary coded Decimal. BCD is a numeric code in which each digit of a decimal number is represented by a separate group of bits.

plement Representation of negative numbers.

In digital Computers, to simply the Subtraction operation and for logical manipulation Complements are used.

There are two types of Complements for Each radix System:

The radix complement and diminished radix complement.

The first is referred to as the r's complement and Second as the (r-1)'s complement.

for Example, in binary System we Substitute base value 2 in place of r to refer Complements as 2's complement and 1's complement. In decimal number system, we substitute base value 10 in place of r to refer complements as 10's complement and 9's complement.

1's complement Representation:

The In 1's complement Representation, the number changes all 1's to zero's and all zero's to 1's.

find 1's complement of $(1101)_2$

1101 ← number.
0010 ← 1's complement.

find 1's Complement of 10111001.

10111001 ← number
01000110 ← 1's complement.

2's complement Representation.

The 2's complement is the binary number that results add 1 to the 1's complement. It is given as

2's complement = 1's complement + 1

The 2's complement form is used to represent negative numbers.

find 2's complement of $(1001)_2$

```
  1 0 0 1   ← number
  0 1 1 0   ← 1's complement.
+       1
-----------
  0 1 1 1   - 2's complement.
```

find 2's complement of $(1010001)_2$

```
  1 0 1 0 0 0 1   ← number
  0 1 0 1 1 1 0   — 1's complement
+           1
-------------------
  0 1 0 1 1 1 1
-------------------
```

1's complement Subtraction

Subtraction of binary numbers can be accomplished by the direct method by using the 1's complement method, which allows to perform subtraction using only addition.

For subtraction of two numbers we have two cases.

- Subtraction of smaller number from larger number and
- Subtraction of larger number from smaller number.

Subtraction of Smaller number from larger number.

method:

1. Determine the 1's complement of the Smaller number.
2. Add the 1's Complement to the larger number.
3. Remove the Carry and add it to the result.

1. Subtract $(101011)_2$ from $(111001)_2$ using the 1's complement.

```
  111001
  010100   — 1's Complement.        57
 --------                           43
 (1)001101                         ----
  └──────→+1                        14
 ---------
   001110  → final answer.
```

Subtraction of larger number from Smaller number.

method:

1. Determine 1's complement of larger number.
2. Add the 1's complement to the Smaller number.
3. Answer is in 1's Complement form. To get the answer in true form take the 1's Complement and assign negative sign to the answer.

Subtract $(111001)_2$ from $(101011)_2$ using the 1's complement method.

```
  101011
  000110   — 1's complement of 111001
 --------
  110001   — Answer in 1's complement form.
 --------
 -001110   — Answer in true form.
```

2's Complement Subtraction

In 2's complement subtraction, the subtraction is accomplished by addition. Let us see the methods for 2's complement subtraction.

→ Subtraction of smaller number from larger number.

method:

1. Determine the 2's complement of a smaller number.
2. Add the 2's complement to the larger number.
3. Discard the carry.

Subtract $(101011)_2$ from $(111001)_2$ using the 2's complement method.

```
  1 1 1 0 0 1                    0 1 0 1 0 0
2's comp→ 0 1 0 1 0 1                    +1
          ---------              ---------
  (1) 0 0 1 1 1 0                0 1 0 1 0 1
  ↓                              ---------
Discard
```

0 0 1 1 1 0 — final answer.

Subtraction of larger number from smaller number:

method:

1. Determine the 2's complement of larger number.
2. Add the 2's complement to the smaller number.
3. Answer is in the 2's complement form. To get the answer in the true form take the 2's complement and assign negative sign to the answer.

Subtract $(111001)_2$ from $(101011)_2$ using 2's complement method.

```
  1 0 1 0 1 1
  0 0 0 1 1 1
  -----------
  1 1 0 0 1 0
```

```
  0 0 0 1 1 0
      + 1
  -----------
  0 0 0 1 1 1   — 2's complement.
```

↳ Answer in 2's complement form

```
  0 0 1 1 0 1
      + 1
  -----------
  - 0 0 1 1 1 0
  -----------
```

Decimal	Signed magnitude	1's complement	2's complement
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	1000	1111	0000
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001

Signed Binary Numbers

Signed Binary Numbers mainly used for represent Sign of the number.

However, because of hardware limitations, in computers both positive and negative numbers are represented with only binary digits.

The left most bit (sign bit) in the number represents sign of the number.

The Sign bit is 0 positive numbers

The Sign bit is 1 negative numbers.

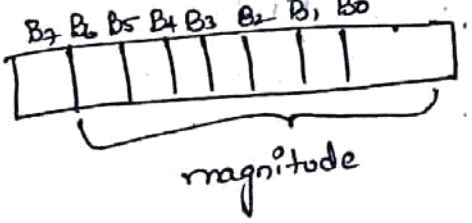
These numbers are represented by the Signed magnitude format.

Below fig shows the Sign magnitude format for 8-bit Signed number.

Here, the most Significant bit (msB) represents Sign of the number.

if msB is 1, number is negative and if msB is 0, number is +ve, The Remaining bits represent magnitude of the numbes.

Here Some Examples for Sign-magnitude numbers.



+6

2 | 6
2 | 3 - 0
1 - 1

00000110

-14

2 | 14
2 | 7 - 0
2 | 3 - 1
1 - 1

10001110

+24 0001 1000

-64 1100 0000

In case of unsigned 8-bit binary numbers the decimal is 0 to 255..

For Signed magnitude 8-bit binary numbers the largest magnitude is reduced from 255 to 127.

because we need to represent both positive and negative numbers.

maximum +ve number 0111 1111 = +127

maximum -ve number 1111 1111 = -128

We have seen +ve and -ve number representation in the Signed magnitude format.

This is the only way to represent +ve numbers!

however there are two more ways represent negative numbers:

Signed -1's Complement representation.

Signed 2's complement representation

Let us see how -6 represents in three formats:

Signed - magnitude representation 10000110

Signed 1's Complement representation 11111001

Signed 2's complement representation 11111010

Binary codes:-

The digital data is transmitted, stored and represented as groups of binary digits (bits). The group of bits, also known as binary code,

Classification of various binary codes.

Alphanumeric codes : The code which consists of both alphabets and numbers.

ASCII Code:- It is a 7-bit code which includes 0-9 -bit code upper case alphabets, lower case alphabets, and other symbols. it is preceded by 011 for decimal no.

0 → 011 0000
9 → 011 1001

A — 65 → 100 0001
Z — 90 → 10 11010

a — 97 → 110 0001
z — 122 — 111 1010.

2	90
2	45 — 0
2	22 — 1
2	11 — 0
2	5 — 1
2	2 — 1
	1 — 0

EBCDIC:— It is a 8-bit code. It is preceded by 1111 for decimal digits

a — 129
z — 154.

Decimal Digit	BCD Code 8 4 2 1
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 0 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

— In multidigit coding, each decimal digit is individual coded with 8421 BCD code

Ex: $(58)_{10} \xrightarrow{8421}$ 0101 1000.

— If we want to represent same 58 in binary the equivalent is $(111010)_2$ i.e we require only 6 digits that means 8421 BCD is less efficient than pure binary number.

— Advantage: It is Easy to convert between it and deci...
— Disadvantage: low efficiency, the arithmetic operations are more complex than they are in pure binary.

2421 code:

Decimal Digit	2421
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1110
9	1111

2: Non weighted Codes: These are not assigned with any weight to Each digit position. Ex:

Excess -3 Code:

It is a modified form of a BCD number. It can be derived from the natural BCD code by adding 3 to Each coded number

Ex: 12 in BCD as 0001 0010

Now adding 3 to Each digit we get Excess-3 code as

$$\begin{array}{cc} 0001 & 0010 \\ 11 & 11 \\ \hline 0100 & 0101 \end{array}$$ — Excess-3 code for $(12)_{10}$

BCD Addition:

The addition of two BCD numbers can be best understood by considering the three cases that occur when two BCD digits are added.

Sum Equal to 9 or less with Carry 0.

let us Consider additions of 3 and 6 in BCD.

```
  6      0110  ← BCD for 6
 +3      0011  ← BCD for 3
 ---    ------
  9      1001  ← BCD for 9.
```

```
  9
 +3
 --
 12
```

Sum greater than 9 with Carry 0.

```
  6      0110
  8      1000
 --     ------
 14      1110  ← invalid BCD (1110) > 9.
```

Whenever invalid BCD no. occurs, the Sum has to be Corrected by the addition of six (0110) in the invalid BCD no. as shown below.

```
  6      0110
  8      1000
 --     ------
 14      1110  — invalid BCD no.
         0110
        ------
        10100  ← BCD for 14.
        ------
         1  4
```

Sum Equals 9 or less with carry 1.

let us consider addition of 8 and 9 in BCD.

26/53

```
8        1000
9        1001
--      ------
17   0001 0001   — incorrect BCD result.
     ---------

     0001 0001   — add 6 for correction
          0110
     ---------
     0001 0111
     ---------
      1    7    — BCD for 17.
```

```
24      0010      0100
18      0001      1000
        ----      ----
        0011      1100   invalid BCD no.
                         1100>9.
                  0110   add 6 for correction.
                 ----
                1 1 1
        0100      0010   — propagate carry to
        ----      ----     next higher digit.
```

```
589     0101   1000   1001
199     0001   1001   1001
---     ---------------------
788     0111   0010   0010
                0110   0110
                1 1    1 1
        ---------------------
        0111   1000   1000   — corrected Sum.
        ---------------------
```

99
26
53

0111 1001 99

0111 0011 = 73 - 9's 26

1 1 1 1 1 Complement 73

1110 1100 for BCD 26.

0110 1100 > 9 add 6 to result.

1 1

1111 0010 1111 > 9.

0110

1 1

10101 0010

→ 1

01 01 0011 BCD for 53.

89
- 54

99 1000 1001

54 0100 0101

45

1100 1110 1110 > 9 So add 6.

0110

1 1 1

1101 0100 1101 > 9 add 6.

0110

1

10011 0100

→ 1

BCD for 35.

0011 0101

175	0001	0111	0101	
326	0011	0010	0110	
501				
	0100	1001	1011	1011 > 9
			0110	add 6 to 1011 for correction.
		11	11	
	0100	1010	0001	
		0110		1010 > 9. add 6.
	0101	0000	0001	propagate carry to next higher digit.

BCD Subtraction :

Addition of Signed BCD numbers can be performed by using 9's or and 10's Complement methods.

A negative BCD number can be Expressed by taking the 9's or 10's complement.

8	9	8
-2.	2	+7
	7.	15
		↳1
		6

7-9 - 26. perform the BCD subtraction using 9's complement method.

Subtraction using 10's complement.

Regular Subtraction

```
  8
- 2
---
  6
```

10's complement Subtraction.

```
  9
  8
---
  7
 +1
---
  8
```

```
  8
  8
---
 16.
```

```
  8        1000
- 8        1000
---      ------
 16       10000
           0111
          ------
           0110   -6.
          ------
```

- find the 10's complement of negative numbers.
- Add two numbers using BCD addition
- if carry is not generated find the 10's complement of the result otherwise find the 9's complement of the result.

```
→ 79        99        79        0111  1001
- 86        86       +72        0111  0100
---        ---       ---       ---------
  53        13                  1101  1101     1101 > 9.
            +1                        0110
                                    ---------
                                    1111  0011  1111 > 9
                                          0110
                                    ---------
                                1 01 01  0011
                                   5     3
```

discard

Excess-3 :- It is a modified form of BCD. & add in code add 3 to BCD of each digit.

	BCD		Excess-3	
12 —	0001	0010	0100	0101

It is a Self Complementing code.

Excess-3 addition:

Ex:- $(592)_{10}$ — 1000 1100 0101

9's comp(592) 0111 0011 1010

Ex:

	BCD	
98	1001	000
Excess	1100	10
	0011	0100

99
98
01

Excess-3 addition

To perform Excess-3 addition we have to

- Add two Excess-3 numbers.
- if carry = 1 → add 3 to the Sum of two digits
  = 0 → Subtract 3

```
  8     1011
+ 6     1001
 14   ---------
    00010100
     00110011
   ------------
    0100 0111
   ------------
      1   4
```

```
1   0100 — Excess-3
+2  0101 — Excess-3
 3  1001
```

Sub3 - 0011
0110 — Excess-3 for 3.

Decimal Digit	Excess-3 code
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

— In Excess-3 code, we get 9's complement of a number by just complementing each bit. Due to this Excess-3 code is called self-complementing code.

Ex: 592 in Excess-3 1000 1100 0101 9's complement of 592 is 999

592

407

407 in Excess-3 0111 0011 1010

i.e nothing but just complementing the 592 in Excess-3 you will get 9's complement of 592.

Ex: 403 in Excess-3 0111 0011 0110

9's complement of 403 is 1000 1100 1001

$$\begin{array}{r} 999 \\ -\ 403 \\ \hline 596 \end{array}$$

596 in Excess-3 is 1000 1100 1001

Gray Code :

It is a special case of unit-distance code. In this bit patterns for two consecutive numbers differ in only one bit position. It is also called as <u>cyclic code</u>.

Decimal Code	Gray code	Decimal	Gray
0	0000	13	1011
1	0001	14	1001
2	0011	15	1000
3	0010		
4	0110		
5	0111		
6	0101		
7	0100		
8	1100		
9	1101		
10	1111		
11	1110		
12	1010		

- gray code any two adjacent code groups differ only in one bit position. It is also called as reflected code, because two least Significant bits for 4 through 7 are the mirror images of those for 0 to 3.
- Similarly, the three LSB for 8 to 15 are the mirror images of are from 0 to 7.

Binary to Gray conversion.

Let us represent binary numbers as $B_1, B_2, B_3, B_4, \ldots B_n$ and its Equivalent gray code as $G_1, G_2, G_3 \ldots G_n$

$$G_1 = B_1$$
$$G_2 = B_1 \oplus B_2$$
$$G_3 = B_2 \oplus B_3$$
$$G_4 = B_3 \oplus B_4$$
$$\vdots$$
$$G_n = B_{n-1} \oplus B_n$$

Ex: 1) $(10110110)_2 \xrightarrow{\text{Gray}} (11101101)$

2) $(1010101011 0)_2 \xrightarrow{\text{Gray}} (1111111101)$

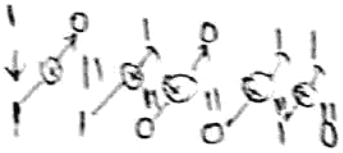
3) $(110010)_2 \xrightarrow{\text{Gray}} (101011)$

Gray to Binary Conversion

Steps

1. The MSB of the binary number is the same as the MSB of Gray code number.
2. To obtain the next binary digit, perform an Ex-OR between the bit just written down and the next gray code bit. Write down the result.
3. Repeat steps until all gray code bits have been Ex-ORed with binary digits. The sequence of bits that have been written down is the binary equivalent of gray code

Ex: 1) Gray code: 1 0 1 0 1 1



(Binary)

2) Gray code: 1 1 1 1 1 1 1 0 1

$(1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0)_2$

3) Gray code 1 1 1 0 1 1 0 1

$(1\ 0\ 1\ 1\ 0\ 1\ 1\ 0)_2$

⇒ Alphanumeric codes: It includes numbers, letters, and other symbols.

ASCII (American standard code for information interchange)

EBCDIC (Extended binary coded decimal Interchange code)

Detecting and Correcting Codes

- When the digital information in the binary form is transmitted from one circuit to another an Error may occur.
- To maintain data integrity between transmitter and receiver Extra bits are added in the data, these allow the detection and Sometimes correction of Errors.

parity Bit:

- It is used for the purpose of detecting Errors during transmission.
- A parity bit is an Extra bit included with a binary message to make the number of 1s either odd or Even.
- The message, including the parity bit is transmitted and then checked at the receiving End for Errors.
- An Error is detected if the checked parity does not correspond with the one transmitted.
- The ckt that generates the parity bit in the Tx is called as parity Generator.
- The ckt that checks the parity in the Rx is called as parity checker.

Decoding the Received Codewords (Detecting the Error) :-

At the receiving End the receiver doesnot know the transmitted word. However it knows A matrix used for generation of code words. Its function is to check the message bits using check bits.

Steps:

1. form the matrix H as $H = [A^T : I_r]$

   Where A^T of Sub-matrix A

   I_r = Identity matrix of order of r (no of check bits)

   matrix H is called parity check matrix.

2. Now if $H \cdot R^T = 0$ — Received word is correct

   $H \cdot R^T \neq 0$ Error in received code.

Ex: For a (4,2) block code received code is 10 11

If $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$, find received code is correct or wrong.

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad ; \quad A^T = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

n – msg. bits
k – check bits

$$H = [A^T : I_r]$$

$$= \left[\begin{array}{cc:cc} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{array}\right]$$

WKT,

$$R = [1 \ 0 \ 1 \ 1]$$

$$R^T = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

$$H \cdot R^T = \left[\begin{array}{cc:cc} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{array}\right] \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1\cdot1 \oplus 0\cdot0 \oplus 1\cdot1 \oplus 0\cdot1 \\ 1\cdot1 \oplus 1\cdot0 \oplus 0\cdot1 \oplus 1\cdot1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

for even no. of 1's of Ex-OR = 0

odd no. of 1's of Ex-OR = 1

$\therefore H \cdot R^T = 0$, received code is correct.

Ex: ~~for~~ a (4,2) block code received code is 1010

If $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$, find received code is correct or wrong

$$H = [A^T : I_2] = \left[\begin{array}{cc:cc} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{array}\right]$$

$$R^T = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$$H \cdot R^T = \left[\begin{array}{cc:cc} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{array}\right] \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1\cdot1 \oplus 0\cdot1 \oplus 1\cdot1 \oplus 0\cdot0 \\ 1\cdot1 \oplus 1\cdot0 \oplus 0\cdot1 \oplus 1\cdot0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \therefore H \cdot R^T \neq 0$$, received code is wrong

Error Correction :

- In order to correct the codeword we multiply received codeword with transpose of parity check matrix to get Syndrome.
- Syndrome is compared with the row of transpose of parity check matrix. (H^T)
- matching row number is the number of bits in Error. Error bit is then inverted to get the correct code.

Steps :-

1. find $S = RH^T$

   $R \rightarrow$ Received code $H^T \rightarrow$ transpose of H.

   $S \rightarrow [S_1, S_2, S_3 \ldots]$ is called Syndrome.

2. Match the result, i.e S, with row of H^T. The no. of row where the match occur gives the no. of bit in Error. This bit is inverted to correct the Error.

Ex:- for a (6,3) block code received code is 1 1 0 1 0 0

If $A = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ find the received code is correct/wrong

If wrong Correct it

$$A^T = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

$$H = [A^T \mid I_r] = \left[\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right]$$

$$R = [1\ 1\ 0\ 1\ 0\ 0]$$

$$R^T = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$H \cdot R^T = \left[\begin{array}{ccc|ccc} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right] \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1\cdot1 \oplus 1\cdot1 \oplus 1\cdot0 \oplus 1\cdot1 \oplus 0\cdot0 \oplus 0\cdot0 \\ 0\cdot1 \oplus 0\cdot1 \oplus 1\cdot0 \oplus 0\cdot1 \oplus 1\cdot0 \oplus 0\cdot0 \\ 0\cdot1 \oplus 1\cdot1 \oplus 1\cdot0 \oplus 0\cdot1 \oplus 0\cdot0 \oplus 1\cdot0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

$\because$ $HR^T \neq 0$, received code is wrong.

let us find $S = R \cdot H^T$

$$S = [1\ 1\ 0\ 1\ 0\ 0]_{1\times6} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}_{6\times3}$$

$1\cdot1 \oplus 1\cdot1 \oplus 0\cdot1 \oplus 1\cdot1 \oplus 0\cdot0 \oplus 0\cdot0 \quad 1\cdot0 \oplus 1\cdot0 \oplus 0\cdot1 \oplus 1\cdot0 \oplus 0\cdot1 \oplus 0\cdot0$

$1\cdot1 \oplus 1\cdot1 \oplus 0\cdot1 \oplus 1\cdot0 \oplus 0\cdot0 \oplus 0\cdot1]$

$$= [1\ 0\ 1]$$

$S = [1\ 0\ 1]$ matches with second row of H^T

∴ Second bit of received code is in Error. By inverting Second

we get,

correct code = $R_c = [1\ 0\ 0\ 1\ 0\ 0]$

Hamming Code

- It provides the detection of a bit Error, also identifies which bit is in Error so that it can be corrected. So it is called as Error detecting and correcting code.

Number of parity bits

- No. of parity bits depend on the no. of information bits

If no. of parity bits is p, then the relation is

$$2^p \geq x + p + 1$$

Ex: If $x = 4$, then p is formed by trial

Let $p = 2$

$2^2 \geq 4 + 2 + 1$

$4 \geq 7$ ⟶ not Satisfied.

So, let $p = 3$

$2^3 \geq 4 + 3 + 1$

$8 \geq 8$

∴ Three parity bits are required to provide single Error.

correction for four information bits.

...ion of parity bits in the code :-

- There are located in the positions that are numbered corresponding to ascending powers of two (1, 2, 4, 8, ...).
- So, for 4-bit information total no. of bits are 7 as

$D_7, D_6, D_5, D_4, D_3, D_2, D_1$

Assigning Values to parity Bits

- In hamming code, Each parity bit provides a check on certain other bits in the total code.

Bit designation	D_7	D_6	D_5	D_4	D_3	P_2	P_1
Bit location number	7	6	5	4	3	2	1
Binary location	111	110	101	100	011	010	001

Assigning of P_1 : P_1 has a 1 for its right most digit. This checks all bit locations, including itself, that have 1's in the same location in the binary number location.

∴ P_1 checks 1, 3, 5, 7 and assigns P_1 according to even or odd parity.

- for even parity hamming code, it assigns P_1 such that bit locations 1, 3, 5 and 7 will have even parity.

Assigning of P_2 :- P_2 has 1 for its middle bit. It checks all bit locations, including itself that have 1s in the middle bit

∴ P_2 → 3, 6 and 7.

Assigning of P_4 :- P_4 has 1 for its left most digit

$\therefore P_4 \rightarrow 4, 5, 6$ and 7

Ex: Encode the binary word 1011 into Seven bit Even parity Hamming code.

Step 1: No. of parity bits is

$2^p = 2^3 = 8$

$x + p + 1 = 4 + 3 + 1 = 8$

Step 2: Construct a bit location table

Bit designation	D_7	D_6	D_5	D_4	D_3	P_2	P_1
Bit location	7	6	5	4	3	2	1
Bit location number	111	110	101	100	011	010	001
Information bits	1	0	1		1		
parity bits				0		0	1

$\therefore$ code 1 0 1 0 1 0 1.

Ex: Determine the Single Error Correcting code for the following information code 10 111 for odd parity.

No. of parity bits is $2^p \geq x + p + 1$

$2^4 \geq 5 + 4 + 1$

$16 \geq 10$

Total bits $= 5 + 4 = 9$

Bit Designation	D_9	P_8	D_7	D_6	D_5	P_4	D_3	P_2	P_1
Bit location	9	8	7	6	5	4	3	2	1
Binary location number	1001	1000	0111	0110	0101	0100	0011	0010	0001

1001 1000 0111 0110 0101 0100 0011 0010

Information bits 1 0 1 1

Even parity bits 0 1 1 0

∴ Code word = 1 00 111 1 0

Detecting and Correcting an Error :

— In this Each parity bit, along with its corresponding group of bits must be checked for proper parity.

The Correct result of individual parity check is marked as '0' wrong as '1'. After all parity checks, binary word is formed taking resulting bit for P_1 as LSB. This word gives bits location where Error has occured.

If all bits are zero, then no Error.

Assume that the Even parity Hamming code is 0 11 0 0 11 is transmitted and that 0 1 0 0 0 11 is received. The receiver doesnot know what was transmitted. Determine bit location where Error has occured using received code.

Step1: Construct bit location table

Bit designation	D_7	D_6	D_5	P_4	D_3	P_2	P_1
Bit location	7	6	5	4	3	2	1
Binary location number	111	110	101	100	011	010	001
Received Code	0	1	0	0	0	1	1

Step2: Check for parity bits

for P_1: P_1 checks 1, 3, 5 and 7

There is one 1 in the group

∴ parity check for Even parity is Wrong — 1 (LSB)

for P_2 : P_2 checks 2, 3, 6 and 7

There are two 1s in the group.

∴ parity check is correct → 0

for P_4 : P_4 checks 4, 5, 6 and 7

There is one 1 in the group

∴ parity check is wrong → 1

The resultant word is 101. It says 5 location is in Error. It is 0 and should be 1. ∴ Correct code is 0110011

Ex: A Seven bit hamming code is received as 1111101 what is the correct code. (Assume as Even parity).

Step1

Bit designation	D_7	D_6	D_5	P_4	D_3	P_2	P_1
Bit Location	7	6	5	4	3	2	1
Bit Location number	111	110	101	100	011	010	001
Received code	1	1	1	1	1	0	1

Step2 : Check for parity

P_1 : 1, 3, 5, 7 → 1111 → (Even) → 0 LSB

P_2 : 2, 3, 6, 7 → 0111 → (odd) → 1

P_3 : 4, 5, 6, 7 → 1111 → (Even) → 0

010 → 2 bit position

∴ Correct code is 1111111

message below has been coded in the Even parity hamming. Decode message assuming that at most a single Error occured in each code word.

a) 1 0 0 1 0 0 1

b) 0 1 1 1 0 0 1

c) 1 1 1 0 1 1 0

d) 0 0 1 1 0 0 1

(i) 1 0 0 1 0 0 1

Bit designation	D_7	D_6	D_5	P_4	D_3	P_2	P_1	Error code
Bit location	7	6	5	4	3	2	1	
Hamming code message	1	0	0	1	0	0	1	

1, 3, 5 and 7 → P_1 — 1 0

2, 3, 6, 7 → P_2 — 0 1

4, 5, 6, 7 → P_4 — 1 0

0 1 0 → Error in bit position 2

∴ Correct code is 1 0 0 1 0 1 1 and message is 1000

(ii) 0 1 1 1 0 0 1

Bit designation	D_7	D_6	D_5	P_4	D_3	P_2	P_1	Error code
Bit location	7	6	5	4	3	2	1	
Hamming code message	0	1	1	1	0	0	1	0

1, 3, 5, 7 → P_1 — 1 1

2, 3, 6, 7 → P_2 — 0 1

4, 5, 6, 7 → P_3 — 1

110 → Error in 6 position

Correct code is 0011001, message is 0010.

(iii) 1110110

Bit designation	D_7	D_6	D_5	P_4	D_3	P_2	P_1	Error code
Bit location	7	6	5	4	3	2	1	
Hamming coded message	1	1	1	0	1	1	0	

1, 3, 5, 7 → P_1 — 0 1 LSB

2, 3, 6, 7 → P_2 — 1 0

4, 5, 6, 7 → P_4 — 0 1

∴ 101 → 5 bit position Error

∴ Correct code is 1100110

message bits 1101

(iv) 0011011

Bit designation	D_7	D_6	D_5	P_4	D_3	P_2	P_1	Error code
Bit location	7	6	5	4	3	2	1	
Hamming coded message	0	0	1	1	0	1	1	

1, 3, 5, 7 → P_1 — 1 0 (LSB)

2, 3, 6, 7 → P_2 — 1 1

4, 5, 6, 7 → P_4 — 1 0

∴ 010 → Error 2bit

∴ Correct Code is 0011001

∴ message is 0010

Boolean Algebra

→ Boolean algebra is a mathematical system that defines a series of logical operations (AND, OR, NOT) performed on sets of variables (a, b, c...)

→ fundamental postulates of Boolean Algebra.

1. Result of each operation is either 0 or 1 $\Rightarrow 1, 0 \in B$

2. Identity Elements

$A+0 = A$ $\quad$ $A \cdot 1 = A$.

$0+0 = A$ $\quad$ $0 \cdot 1 = 0 = A$

$1+0 = A$ $\quad$ $1 \cdot 1 = 1 = A$.

3. Commulative law.

a) $(A+B) = (B+A)$

b) $(A \cdot B) = (B \cdot A)$

a) $A = 0$, $B = 0$

$0+0 = 0+0$

b) $A = 0$ (A), $B = 1$

$0 \cdot 1 = 1 \cdot 0$

4. a) $A \cdot (B+C) = (A \cdot B) + (A \cdot C)$

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

} distributed law.

$A = 0$ $\quad$ $B = 1$ $\quad$ $C = 0$.

$0 \cdot (1+0) = (0 \cdot 1) + (0 \cdot 0)$

$0 = 0$

b) $A=0 \quad B=0 \quad C=0$.

$$0+(0\cdot 0) = (0+0)(0+0)$$

$$0=0 \quad .$$

5) a) $A+\bar{A}=1$ Since $0+\bar{0} = 0+1 = 1$

and $1+\bar{1} = 1+0 = 1$

b) $A\cdot\bar{A}=0$ Since $0\cdot\bar{0} = 0\cdot 1 = 0$.

and $1\cdot\bar{1} = 1\cdot 0 = 0$

} Complement .

Basic theorems and properties

Duality :- The principle of duality theorem say that, starting with a Boolean relation, you can derive another boolean relation by.

1. changing each OR sign to an AND sign.
2. changing each AND sign to an OR sign and.
3. complementing any 0 to 1 appearing in the expression.

Example: Dual of relation $A+\bar{A}=1$ is $A\cdot\bar{A}=0$.

Duality is a very important property of Boolean algebra.

Basic Theorems :-

We can define the following theorems using fundamental posulates of Boolean algebra.

Theorem 1 (a) $A+A=A$.

$$\begin{matrix}0\\1\end{matrix} + \begin{matrix}0\\1\end{matrix} = \begin{matrix}0\\1\end{matrix} \Rightarrow A+A=A$$

proof:- $A+A = (A+A)\cdot 1$

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

$= A+A\bar{A}$

$= A+0 = A$.

Theorem 1(b) $A\cdot A = A$

$$\begin{matrix}0\\1\end{matrix} + \begin{matrix}0\\1\end{matrix} = \begin{matrix}0\\1\end{matrix} \Rightarrow A+A=A$$

proof:- $A \cdot A = A \cdot A + 0$

$= AA + AA'$

$= A(A + A')$

$= A \cdot 1 = A$.

Theorem 2(a) : $A + 1 = 1$

$1 + \boxed{0} = 1$

$1 + \boxed{1} = 1$ $\Rightarrow$ $1 + A = 1$ or $A + 1 = 1$

proof: $A + 1 = 1 \cdot (A + 1)$

$= (A + A')(A + 1) = AA + A + 0 + A' = A + A' \cdot 1$

$= A + A' \cdot 1 = A + A' = 1$

Theorem 2(b) : $A \cdot 0 = 0$

$0 \cdot \boxed{0} = 0$

$0 \cdot \boxed{1} = 0$ $\Rightarrow$ $0 \cdot A = 0$ or $A \cdot 0 = 0$.

proof: $A \cdot 0 = 0$ by duality theorem 2(a)

Theorem (3): $\overline{\overline{A}} = A$

$\boxed{\overline{\overline{0}}} = \boxed{0}$

$\overline{\overline{1}} = 1$ $\qquad$ $\overline{\overline{A}} = A$

proof: Complement of $\overline{A}$ is A and also $\overline{\overline{A}}$

Theorem 4(a): $A + AB = A$

proof: $A + AB = A \cdot 1 + AB$

$= A(1 + B) = A \cdot 1 = A$.

em 4(b) : $A(A+B) = A$

proof: $A(A+B) = A \cdot A + AB$ By postulate : 4(b)

$= A + AB$ By theorem : 1(b)

$= A(1+B) = A$

Theorem 5(a) : $A + A'B = A+B$

proof: $A + AB + A'B$

$= A + B \cdot (A + A') = A + B \cdot 1 = A+B$

Theorem 5(b): $A \cdot (\bar{A} + B) = AB$

proof: $A \cdot (\bar{A} + B)$

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

$= AA' + AB + ABB$

$= AB + ABB$

$= AB + AB$

$= AB$

Demorgan's Theorems

Demorgan's Suggested two theorems that form an important part of Boolean algebra. In the Equation form, they are.

1) $\overline{AB} = \overline{A} + \overline{B}$

The complement of a product is Equal to the Sum of Complements. This illustrated by below table.

Truth Table

A	B	$\overline{AB}$	$\overline{A} + \overline{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

2) $\overline{A+B} = \overline{A}\,\overline{B}$

The complement of a Sum is Equal to the product of the Complements. The below truth table illustrate this law.

Truth Table

A	B	$\overline{A+B}$	$\overline{A} \cdot \overline{B}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Consensus Theorem :

In Simplication of Boolean Expression, an Expression of the form $AB+A'C+BC$ the term BC is Redundant and Can be Eliminated to form the Equivalent Expression $AB+A'C$. The theorem used for this Simplification is known as consensus theorem and it is stated as

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

— The Key to recognize the Consensus terms is to first find a pair of terms, one of which Contains a Variable and the other Contains its Complement.

— Now we have to find the third term which Should Contain the remaining Variables from pair of terms eliminating Selected Variable and its Complement.

Proof:
$$AB+A'C+BC = AB+A'C+(A+A')BC$$
$$= AB+A'C+ABC+A'BC$$
$$= AB+A'C \pm ABC+A'BC$$
$$= AB(1+C)+A'C(1+B)$$
$$= AB+A'C$$

Example 2.1: Solve the given Expression using Consensus theorem

$$A'B'+AC+BC'+B'C+AB$$

Solution:
$$A'B'+AC+BC'+B'C+AB = A'B'+AC+BC'+AB$$

$$A'B'+AC+BC'+AB = A'B'+AC+BC'$$

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

Note: The brackets indicate how the Consensus terms are identified.

Dual of Consensus Theorem

- The dual form of consensus theorem is stated as

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

Proof: $(AA' + AC + A'B + BC)(B+C) = AA' + AC + A'B + BC$

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

$$ABC + A'BB + BCC + ACC + A'BC + BCC = AC + A'B + BC$$

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

$$(1)BC + A'B + BC + AC = AC + A'B + BC$$

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

$$BC + A'B + AC = AC + A'B + BC .$$

Example 2.2: Solved the following Boolean Expression using dual of Consensus theorem.

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

Solution: $(A+B)(A'+C)(B+C)(A'+B)(B+D)$

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

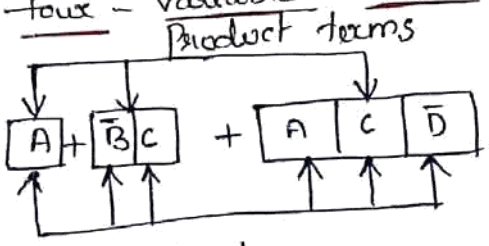
$$= (A+B)(A'+C)(A'+D)$$

Switching function / Boolean function

- Boolean Expressions are constructed by connecting Boolean constants and Variables with the Boolean operations.
- These Boolean Expressions are also known as Boolean formulas.
- Boolean Expressions are used to describe the functions f.
- for Example, if the Boolean Expression $(A+B')C$ is used to describe the function f, then Boolean function is written as

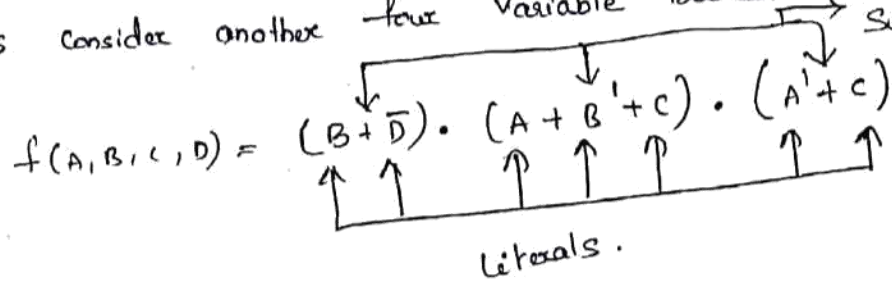
$$f(A,B,C) = (A+B')C \quad \text{or} \quad f = (A+B')C$$

- Based on the Structure of Boolean Expression, It can be categorized in different formulas.
- let us consider the four-Variable Boolean function



- In this Boolean function the Variables are appeared either in a Complemented or an uncomplemented form.
- Each occurance of a variable in either a complemented or an uncomplemented form is called a literal.
- The above Boolean function consists of 6 Variables or literals.
- The product term is defined as either a literal or a product of literals.
- The above function contains three product terms, A, $\bar{B}C$ and $AC\bar{D}'$.

Let us consider another four variable Boolean function.



— The above Boolean function consists of seven literals.

— A Sum term is defined as either a literal or a sum of literals.

— The above function contains three Sum terms, namely

$(B+D')$, $(A+B'+C)$ and $(A'+C)$.

— There literals and terms are arranged in one of the two forms.

- Sum of product form (SOP) and
- product of sum form (POS)

Sum of product form

— The words Sum and product are derived from the Symbolic representations of the OR and AND functions by + and . (addition and multiplication), respectively.

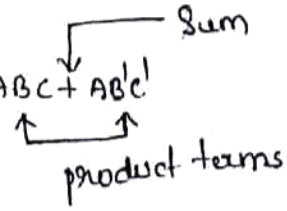
— A product term is any group of literals that are ANDed together.

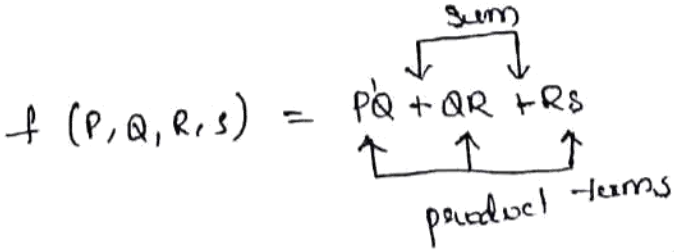
— for example; ABC, zy and so on

— Sum term is any group of literals are ORed together. Such as A+B+C, x+y and so on

— A Sum of products (SOP) is a group of product terms ORed together

— for Example





- The Sum of products Expressions Consists of two or more product terms (AND) that are ORed together.
- Each product term Consists of one or more literals appearing in either Complemented or uncomplemented form.
- for Example, in this Expression $ABC + AB'c'$, the first product term Contains literals A, B and C, in their uncomplemented form.
- The Second product term Contains B and C in their Complemented form.
- The Sum of product form is also known as normal form.

product of Sum form

A product of Sums is any groups of Sum terms ANDed together.

Some Examples

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

product

Sum terms.

$f(P,Q,R,S) = (P+Q) \cdot (R+\bar{S}) \cdot (P+S)$

product

Sum terms

- There product of Sums Expressions Consists of two or more Sum terms (OR) that are ANDed together.
- Each Sum term consists of one or more literals appearing in either complemented or uncomplemented form.
- The product of Sum form is also known as conjective normal form or conjunctive normal formula.

Canonical form (Standard SOP and POS forms)

— The Canonical forms are the Special cases of SOP and POS forms. There are also known as standard SOP and POS forms.

Standard SOP form or minterm Canonical form

— for Example, in Expression $AB + ABC'$ the first product term do not Contain literal C.

— If Each term in SOP form Contains all the literals then the SOP form is known as standard or Canonical SOP form.

— Each individual term in the standard SOP form is known as minterm.

— Therefore Canonical SOP form is also known as minterm Canonical form.

— See this Expression $ABC' + ABC + A'BC + AB'C$ all the literals are present in Each product term.

— one standard Sum of products Expression is as shown in fig

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

Each product term Consists of all literals in either Complemented form or uncomplemented form.

Standard POS form or maxterm Canonical form

- If each term in POS form contains all the literals then the POS form is known as standard or canonical POS form.
- Each individual term in the standard POS form is called maxterm.
- ∴ Canonical POS form is also known as maxterm Canonical form.
- one standard product of sums Expression is as shown in fig.

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

Each Sum term Consists of all literals in either Complemented form or uncomplemented form.

fig: Standard POS form.

Converting Expressions in standard SOP or POS form

- Sum of products form Can be converted to standard Sum of products by ANDing the terms in the Expression with terms formed by ORing literals and its Complement which are not present in the form.
- for example, a three literal Expression with literals A, B and C
- if there is a term AB, where C is missing, then we form term $(C+\bar{C})$ it with AB.
- ∴ we get $AB(C+\bar{C}) = ABC + AB\bar{C}$

Steps to convert SOP to standard SOP form.

Step1 :- find the missing literal in each product term.

Step2 :- AND each product term having missing literal/s with terms form by ORing the literal and its complement form.

Step3 :- Expand terms by applying distributive law and recorder the literals in the product terms.

Step4 :- Reduce the expression by omitting repeated product terms if any. Because $A+A = A$.

Example 2.9 : Convert the given expression in standard SOP form

$f(A,B,C) = AC + AB + BC$

Solution: **Step1:** find the missing literal/s in each product term

$f(A,B,C)$ [AC] + [AB] + [BC]

- Literal A is missing
- Literal C is missing
- Literal B is missing

Step2 : AND product term with (missing literal + its complement) original product terms

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

missing literals and their complements

Step 3: Expand the terms and reorder literals

Expand: $f(A,B,C) = ACB + ACB' + ABC + ABC' + BCA + BCA'$

Reorder: $f(A,B,C) = ABC + AB'C + ABC + ABC' + ABC + A'BC$

Note: After having sufficient practice student should expand product term and reorder literals in it in a single step.

Step 4: omit repeated product terms

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

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

Example 2.10: Convert the given expression in standard SOP form

Solution: Step 1: find the missing literal/s in each product term

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

literals B and C are missing.

Step 2: AND product term with (missing literal + its complement)

$f(A,B,C) = A \cdot (B+B') \cdot (C+C') + ABC$

$= (AB + AB') \cdot (C+C') + ABC$

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

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

Steps to convert POS to standard POS form

Step1: find the missing literals in each Sum term if any.

Step2: OR each Sum term having missing literal/s with form by ANDing the literal and its complement.

Step3: Expand the terms by applying distributive law and reorder the literals in the Sum terms.

Step4: Reduce the expression by omitting repeated Sum terms if any. Because $A+A=A$.

Example: Convert the given expression in standard POS form.

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

Solution: Step1: find the missing literal/s in each Sum term

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

$A+C$ — literal B is missing

$B+C$ — literal A is missing

$A+B$ — literal C is missing

Step2: OR Sum term with (missing literal + its complement.

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

(original products)

(missing literals and their complements)

Step3: Expand the terms and reorder literals

Expand: Since $A+BC = (A+B)\,(A+C)$ we have

$$f(A,B,C) = (A+B+C)\,(A+B+C')\,(B+C+A)\,(B+C+A')$$
$$(A+C+B)\,(A+C+B')$$

reorder: $f(A,B,C) = (A+B+C)\,(A+B+C')\,(A+B+C)\,(A'+B+C)$
$$(A+B+C)\,(A+B'+C)$$

Step 4: omit Repeated Sum terms

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

Repeated Sum terms

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

Example : Convert the given Expression in Standard POS fo

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

Solution: Step1 : find the missing literal/s in Each Sum term

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

Literals B and C are missing.

Step2: OR Sum term with (missing literal · its complement.

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

Step3: Expand the terms and reorder literals

Since $A+BC = (A+B)(A+C)$ we have,

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

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

Step4: Omit repeated Sum terms Repeated Sum-term.

$$f(A,B,C) = (A+B+C)\ (A+B'+C')\ (A+B+C')\ (A+B'+C')\ (A+B+C)$$

$$f(A,B,C) = (A+B+C)\ (A+B'+C)\ (A+B+C')\ (A+B'+C')$$

M-Notations : Minterms and maxterms

— Each individual term in standard SOP form is called minterm and Each individual term in standard POS form is called maxterm.

— The concept of minterms and maxterms allow us to introduce a very convenient shorthand notations to Express logical functions.

— Below table gives the minterms and maxterms for a three literal/ Variable logical function

— where the number of minterms as well as maxterms is $2^3 = 8$.

— for an n-variable logical function there are 2^n minterms and an Equal number.

ple 12: $A+A'B+AB' = A+B$

L.H.S $= A+A'B+AB'$

$= A(1+B') + A'B$

$= A(1+B') + A'B = A+AB'+A'B$

$= (A+B) + AB'$

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

$= A(1+B')+B \quad A+A+B$

$= A+B$

Example 13: $\overline{A\bar{B} + \bar{A} + AB}$

$= \overline{\bar{A}+\bar{B}+\bar{A}+AB}$ Demorgan's theorem : $[\overline{AB} = \bar{A}+\bar{B}]$

$= \overline{\bar{A}+\bar{B}+AB} = \overline{A'+B+B'}$ $\boxed{A+A'=1}$

$= \overline{A'+1}$ $\boxed{A+1} = 1$

$= \bar{1} \cdot C$

$= 0$

$\bar{A}^T \cdot 1'$

$A \cdot 1$

$A = 0 = 0$

Example 14: $AB + \overline{\bar{A}C} + AB'C\,(AB+C)$

$= AB + \overline{\bar{A}C} + AA\bar{B}BC + A\bar{B}CC$

$= AB + \overline{\bar{A}C} + A\bar{B}CC$

$= AB + \overline{\bar{A}C} + AB'C$

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

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

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

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

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

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

$= A' + C' + 1 = 1$

Example: Simplify the Expression $Z = AB + AB' + (\overline{\bar{A}\,\bar{C}})$

Solution:-

Step1:- Apply the Demorgan's theorem and multiply out all to get Expression in Sum of products form.

$$Z = AB + AB' \cdot (\overline{\bar{A} \cdot \bar{C}})$$

$$= AB + AB' \cdot \bar{\bar{A}} + \bar{\bar{C}}$$

$$= AB + AB' \cdot (A + C)$$

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

$$= AAB + ABC + AAB' + AB'C$$

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

$$= AB(1 + C) + AB'(1 + C)$$

$$= AB + AB' = A(1 + B') = A.$$

Example: Simply the following three Variable Expression using Boolean algebra. $Y = \sum m(1, 3, 5, 7)$

Solution: Step1: from the minterms we can write Expression in Sum of products form as follows

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

Step2: Search for Common terms for factorization and apply boolean rules.

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

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

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

$$= A'C + AC$$

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

: Convert the Expression given in the previous Example into (11) minterms using Complementary property and Simply the Expression.

Solution: $Y = \pi M(3,5,7)$

$Y = \Sigma m(0,1,2,4,6)$

Step1: from the minterms write the Expression in SOP form

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

$$= A'B'C' + A'B'C + A'BC' + ABC' + A'B'C \quad \text{Rearranging terms}$$

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

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

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

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

$$= \bar{C} + \bar{A}\bar{B}\bar{C} \qquad \boxed{A + A'B = A + B}$$

Digital Logic Gates

Logic operators: To represent and solve arithmetic Expressions we use arithmetic operators such as +, −, × and ÷,

- Similarly, we can use logical operators to represent and solve logical Expressions.
- There are basic logical operators: NOT / INVERT, AND and OR.
- Logical operator NOT / INVERT

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

Table: minterms and maxterms for three variables

— As shown in table (2.5) Each minterm is represented by m_i and Each maxterm is represented by M_i, where subscript i is the decimal number Equivalent of the natural binary number.

— With these shorthand notations logical function can be represented as follows

1. $f(A,B,C) = A'B'C' + A'B'C + A'BC + ABC'$
$= m_0 + m_1 + m_3 + m_6 = \Sigma m\,(0,1,3,6)$

2. $f(A,B,C) = (A+B+C')\,(A+B'+C')\,(A'+B'+C)$
$= M_1 + M_3 + M_6 = \pi M\,(1,3,6)$

Where Σ denotes Sum of product while π denotes product of Sum.

ebraic Simplification.

Example 1 $A \cdot A'C = 0 \cdot C$

$= 0$

Example 2. $ABCD + ABD$

$ABD(1+C)$

$ABD(1) = ABD.$

Example 3: $ABCD + AB'CD = (ACD)(B+B')$

$= ACD \cdot 1 = ACD.$

Example 4: $A(A+B) = AA + AB$

$= A + AB$

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

Example 5: $AB + ABC + AB(D+E)$

$= AB + ABC + ABD + ABE$

$= AB(1+C+D+E)$ $\quad \therefore A+1 = 1$

$= AB.$

Example 6: $xy + xyz + xyz' + x'yz$

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

$= xy + xyz' + x'yz$

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

$= xy + x'yz$

$= y(x+x'z)$

$= y(x+z)$

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

$(A+A')(A+B)$

$1 \cdot (A+B) = A+B$

Example 7: $A'B'C' + A'BC' + A'BC$

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

$= A'C' + A'BC$

$= A' (C' + BC)$ $\quad \because A + A' = 1$

$\therefore A + A'B = A + B$

$= A' (C' + B)$

Example 8: $ABC + AB'C + ABC' = AC + B)$

$L.H.S = AC (B + B') + ABC'$

$= AC + ABC'$

$= A (C + BC')$ $\quad \therefore A + A'B = A + B.$

$= A (C + B)$

Example 9: $A'B CD' + BCD' + BC'D' + BC'D$

$= BCD' (A' + 1) + BC'D' + BC'D$ $\quad \boxed{(A+1) = 1}$

$= BCD' + BC'D' + BC'D$

$= BD' (C + C') + BC'D$

$= BD' + BC'D$ $\quad A + A'B = (A + B)$

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

$= B (D' + C')$

Example 10: $AC + C (A + A'B)$

$= AC + AC + A'BC$

$= AC + A'BC$

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

Example 11: $A'BC'D + A'BCD + ABD$

$= A'BD (C' + C) + ABD$

$= A'BD + ABD$

$= BD (A + A') = BD.$

c Gates

Logic gates are the basic elements that makeup a digital system.

The Electronic gate is a circuit that is able to operate on a number of binary i/p's in order to perform a particular logical function.

The types of logical gates are.

NOT, AND, OR, NAND, NOR, Exclusive-OR, and Exclusive-NOR

The gate is a digital circuit with one or more input voltages but only one o/p voltage.

By connecting the different gates in different ways, we can build circuits that perform arithmetic and other functions associated with the human brain because they simulate mental processes.

The operation of a logic gate can be easily understood with the help of truth table.

The truth table is a table that shows all the input-output possibilities of a logic circuit; i.e the truth table indicates the o/p's for different possibilities of the inputs.

INVERTER - NOT Gate

The inverter (NOT circuit) performs a basic logic function called 'inversion' or "complementation".

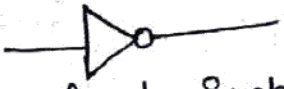


fig: Inverter Symbol

- The inverter changes one logic level to its opposite level.
- In terms of bits, it changes a logic 1 to a logic 0 and a logic 0 to a logic 1. above fig shows the symbol for the inverter.

Inverter operation :

When a high level is applied to an inverter input, a low will appear on its output.

When a low level is applied to an inverter input, a high will appear on its output.

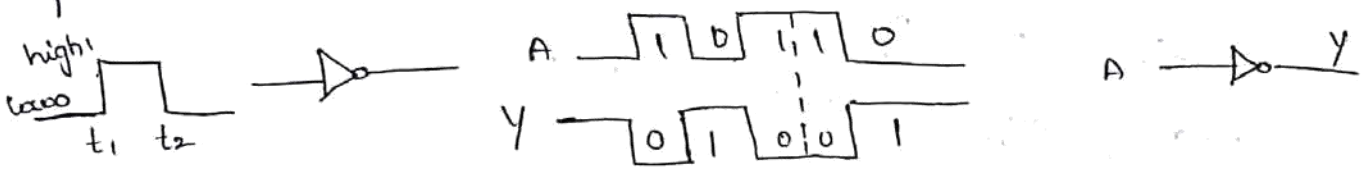
Inverter Truth Table

Input	output	Input	output
Low	high	0	1
High	Low	1	0

The below fig shows the o/p of an inverter for a pulse input. Here t_1 and t_2 indicate the corresponding points on the i/p and o/p pulse waveforms.

Note: When the input is low, the o/p is high, and when the i/p is high, the o/p is low.

The inverter produces an inverted o/p pulse for a given input pulse



o/p of an inverter for pulse i/p.

& gate

The AND gate performs logical multiplication, The AND gate have two or more inputs and a single output, as indicated by the standard logic symbol. shown in the below fig.

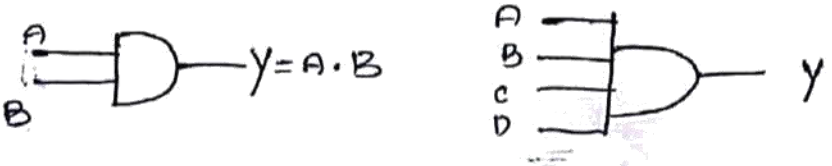


fig (a) Two inputs to AND gate fig (b) four inputs to AND gate.

The operation of the AND gate is such that the o/p is high only when all of the inputs are high.

When any of the inputs are low, the output is low.

Below fig illustrates a two-input AND gate with all four possibilities of input combinations, and resulting output for each.

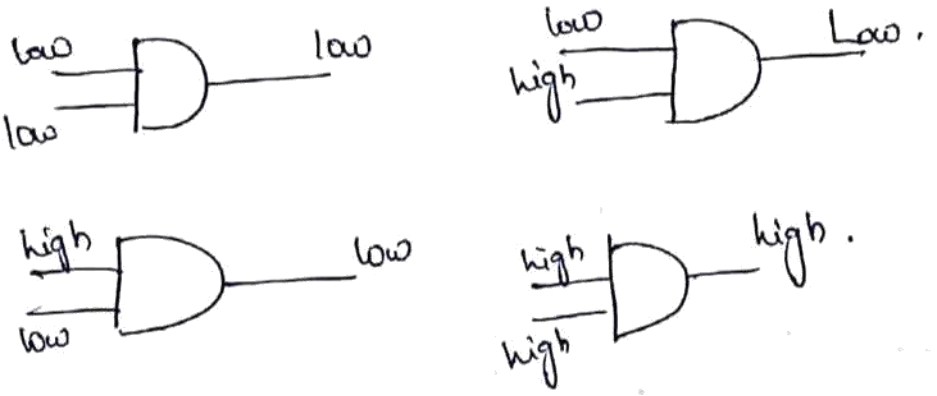


fig: four possible inputs for two input AND gate and resulting outputs.

The truth table for a two-input AND gate is shown in table.

This table can be Expanded for any number of inputs.

Inputs		output
A	B	y
0	0	0
0	1	0
1	0	0
1	1	1

Table: Truth table for 2 input AND gate.

- Below fig shows one way to build a 2-input AND gate.
- The inputs are labeled A and B, while the o/p is y.
- Let us assume a supply voltage Vcc of +5V.
- Next assume the input voltages are either 0V (low) or 5V (high). With 2 inputs, there are four possible input cases and we will now observe the o/p for all four input cases.

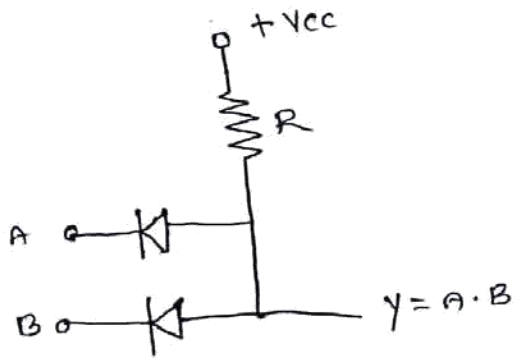


fig: 2-input AND gate.

Case 1: A is low and B is low: When both input voltages are low, the cathode of each diode is grounded. Therefore, the positive supply forward-biases both diodes in parallel. Because of this, the o/p voltage is ideally zero (practically 0.7V for Si). This means y is low.

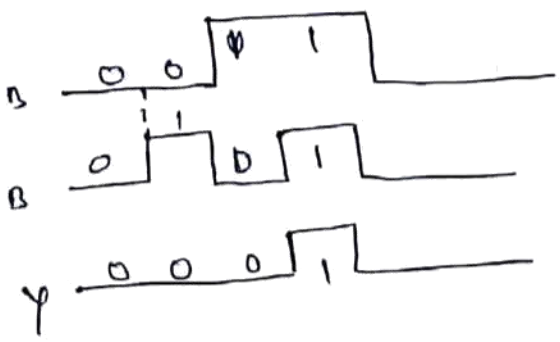
Case 2: A is low and B is high: When A is low, the upper diode is forward-biased (ON), and it pulls the o/p down to a low voltage, i.e $y=0$. With the B input high, the lower diode goes into reverse bias (OFF).

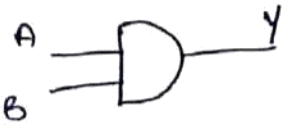
Case 3: A is high and B is low: Because of the symmetry of the circuit, the circuit operation is similar to case 2. But in this case, upper diode is reverse biased (OFF), lower diode is forward biased (ON) and y is low.

Case 4: A is high and B is high: When both inputs are at +5V, both diodes are reverse biased and there is no current through diodes and resistor R. This pulls up the o/p y to the supply voltage. Therefore y is high.

pulsed operation:

In majority of applications, the inputs to a gate are not constant levels but are voltages that change with time between two logic levels and that can be classified as pulse waveforms.

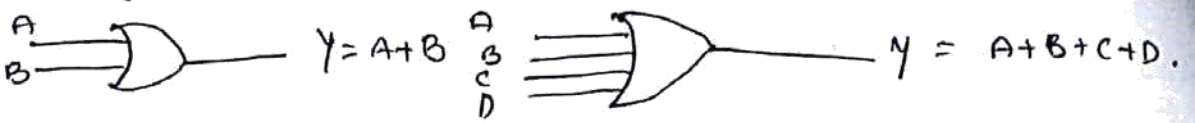




The OR Gate:

The OR gate performs logical addition.

Symbol

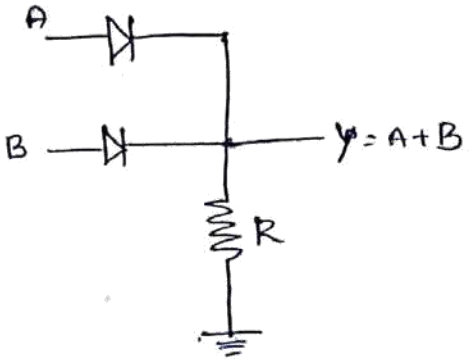


Truth table

A	B	output
0	0	0
0	1	1
1	0	1
1	1	1

Two i/p OR- Gate:

a) When both are low, anodes of both the diodes are grounded with results in RB. So o/p is low.

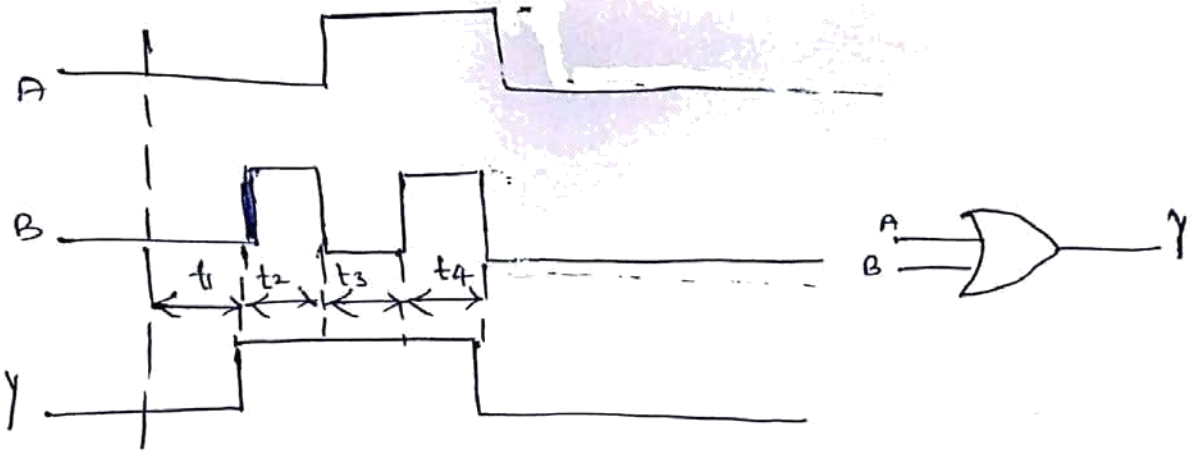


b) When A is low and B is high the diode A is RB and diode B is FB .. which produces o/p voltage of 5V. So the o/p is high

Same as Case b.

1) when both the i/p's are at +5V, both diodes are F.B. Since the i/p voltages, are in parallel, the o/p is ideally +5V, o/p is high.

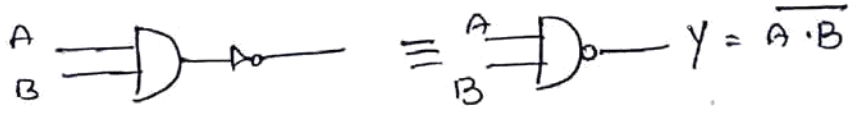
pulsed operation



NAND Gate: (Universal)

A NAND gate is a logic gate with two or more inputs are only one output. Its o/p is only when Each any one & all of its input is a 1.

It is AND gate followed by a inverter.

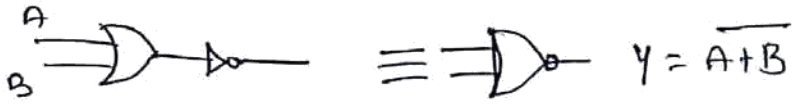


Truth -lable

A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate:

A NOR gate is a logic gate with two or more inputs and only one o/p. its o/p is HIGH only when all inputs are LOW.



Truth table

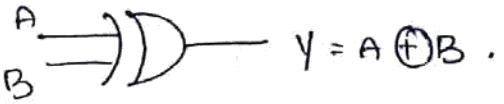
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Ex-OR: (anti coincidence gate or inequality detector)

It produces an o/p 1 only when the inputs are not Equal i.e if they are odd no. of 1's then o/p is 1 otherwise 0.

Truth table

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0



Boolean Expression $Y = A \oplus B$

$= AB' + A'B$.

Properties of XOR Gates

property 1: $A \oplus A = 0$: output is logic Zero when inputs are same.

property 2: $A \oplus \overline{A} = 1$: output is logic 1 when inputs are different.

property 3: $A \oplus 1 = \overline{A}$: Ex-OR as inverter.

When one input of Ex-OR gate is connected to logic one we get the complement of the other input at the o/p of Ex-OR gate.

$0 \oplus 0 = 0$

$0 \oplus 1 = 1$

input tied to logical 1 $\begin{bmatrix} 1 \\ 1 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ output is complement form of other input.

other input

property 4: $A \oplus 0 = A$ Ex-OR as Non-inverter.

When one input of Ex-OR gate is connected to logic 0 we get the un complement of the other input at the o/p of Ex-OR gate.

input is Grounded $\begin{bmatrix} 0 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ Output is drawn complement of other input.

$1 \oplus 0 = 1$

$1 \oplus 1 = 0$

property 5: Ex-OR as modulo 2 Adder.

The Exclusive-OR gate can be used as a modulo 2 adder its truth table is same as the truth table of modulo 2 adder

$0 + 0 = 0$	$0 \oplus 0 = 0$
$0 + 1 = 1$	$0 \oplus 1 = 1$
$1 + 0 = 1$	$1 \oplus 0 = 1$
$1 + 1 = 0$	$1 \oplus 1 = 0$

property 6: $AB \oplus AC = A(B \oplus C)$

A	B	C	$AB \oplus AC$	$A(B \oplus C)$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

property 7: $A \oplus B = C$, then $A \oplus C = B$, $B \oplus C = A$ and $A \oplus B \oplus C = 0$

A	B	$A \oplus B = C$	$A \oplus C = B$	$B \oplus C = A$	$A \oplus B \oplus C = 0$
0	0	0	0	0	0
0	1	1	1	0	0
1	0	1	0	1	0
1	1	0	1	1	0

Note: 1. for three input ex-OR, output is logic 1 only for odd number of logic 1 i/p's

2. No similar terms for Ex-NOR gate.

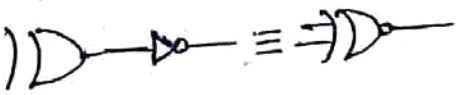
$A \odot B = C$ then $A \odot C = B$ $B \odot C = A$ and $A \odot B \odot C = 1$

- NOR : (Coincidence or Equality detector)

Combination of an X-OR and a NOT gate. It produces an o/p 1 when all the i/p's are '0' or when all i/p's are 1

Truth table

A	B	A⊙B
0	0	1
0	1	0
1	0	0
1	1	1



Boolean Expression $Y = A \odot B$

$= AB + A'B'$

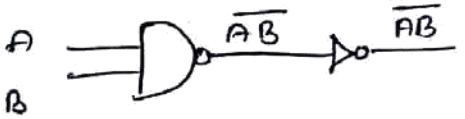
Implementation of NOT, AND, OR and NOR Gates using NAND gate.

NOT function : Considering all the inputs as a single input i.e common input as inverter can be obtained using NAND:

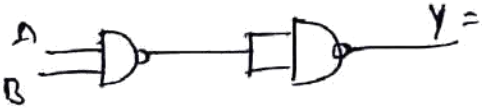
$Y = \overline{X \cdot X} = \overline{X} + \overline{X}$

$= \overline{X}$

AND function : By connecting an inverter to the NAND we get the AND function.



$\Rightarrow Y = AB$



$Y = A \cdot B$

Truth table

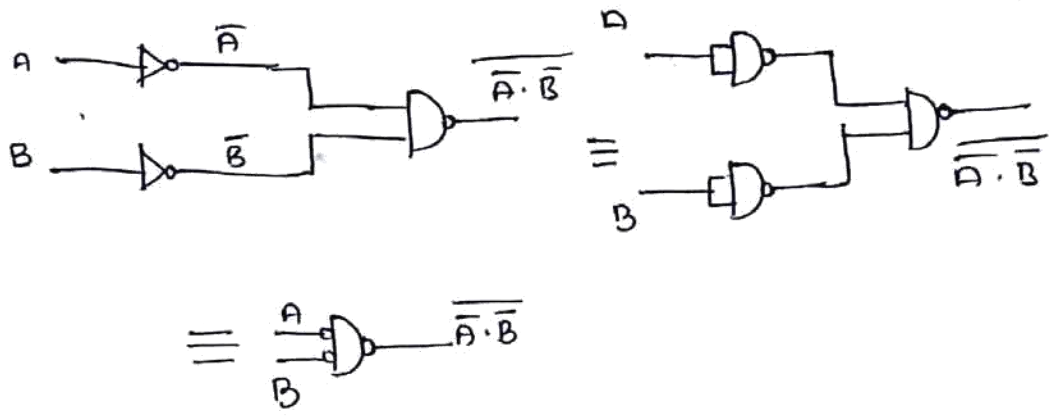
A	B	AB	$\overline{AB}$	$\overline{\overline{AB}}$
0	0	0	1	0
0	1	0	1	0
1	0	0	1	0
1	1	1	0	1

OR function: By connecting an inverters to input of th

Boolean Expression $Y = A+B$ or $\overline{\overline{A}+\overline{B}}$

$$Y = \overline{\overline{A+B}}$$

$$= \overline{\overline{A}\cdot\overline{B}}$$

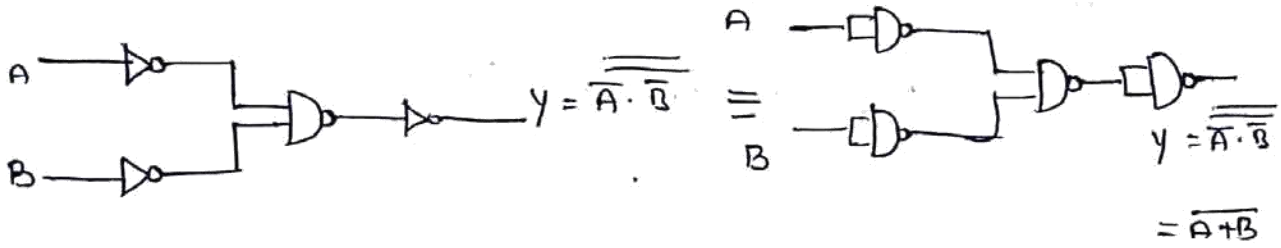


NOR function

Boolean Expression

$$Y = \overline{A+B}$$

$$= \overline{\overline{\overline{A}\cdot\overline{B}}}$$



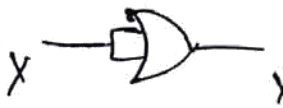
Truth table

A	B	$\overline{A}\cdot\overline{B}$	$\overline{\overline{A}\cdot\overline{B}}$	$\overline{\overline{\overline{A}\cdot\overline{B}}}$	$\overline{A+B}$
0	0	1	0	1	1
0	1	0	1	0	0
1	0	0	1	0	0
1	1	0	1	0	0

NOR gate:

NOT Gate:

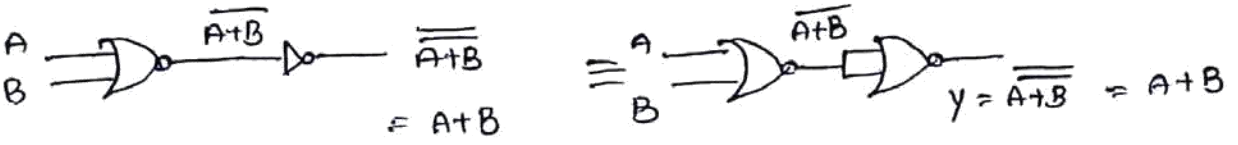
By connecting all inputs to a Single Common input then NOT gate is made.



$\overline{X+X} = \overline{X} \cdot \overline{X} = \overline{X}$

OR function:

obtained by inverting the o/p of NOR gate.



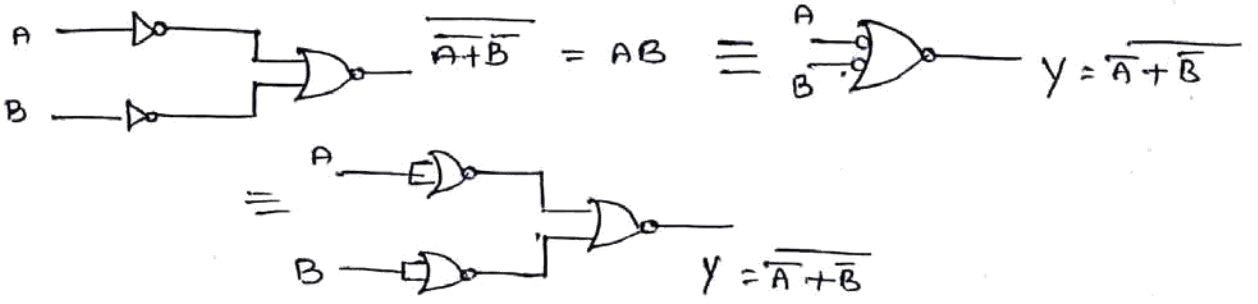
Truth table

A	B	$\overline{A+B}$	$\overline{\overline{A+B}}$	A+B
0	0	1	0	0
0	1	0	1	1
1	0	0	1	1
1	1	0	1	1

AND function:

$Y = A \cdot B = \overline{\overline{A \cdot B}}$

$= \overline{\overline{A} + \overline{B}}$

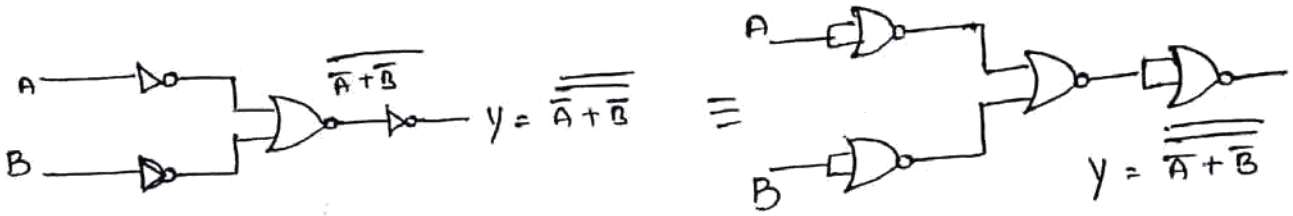


Truth table

A	B	$\overline{A}+\overline{B}$	$\overline{\overline{A}+\overline{B}}$	AB
0	0	1	0	0
0	1	1	0	0
1	0	1	0	0
1	1	0	1	1

NAND function

$$y = \overline{A \cdot B} = \overline{\overline{A}+\overline{B}}$$



Truth table

A	B	$\overline{A}+\overline{B}$	$\overline{\overline{A}+\overline{B}}$	$\overline{\overline{\overline{A}+\overline{B}}}$
0	0	1	0	1
0	1	1	0	1
1	0	1	0	1
1	1	0	1	0

Conversion of AND / OR / NOT logic to NAND / NOR logic using Graphical procedure

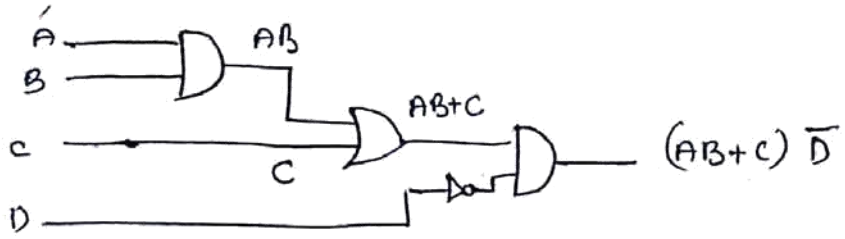
1) Draw AND / OR logic
2) If NAND h/w has been choosen, add bubbles on the o/p of Each AND gate and on input side to all OR gates.
3) If NOR gate h/w has been choosen, add bubbles on the o/p of Each OR gate and bubbles on input side to all AND gates.
4) Add or Subtract an inverter on Each line that Rxed a bubble

Replace bubbled OR by NAND and bubbled AND by NOR

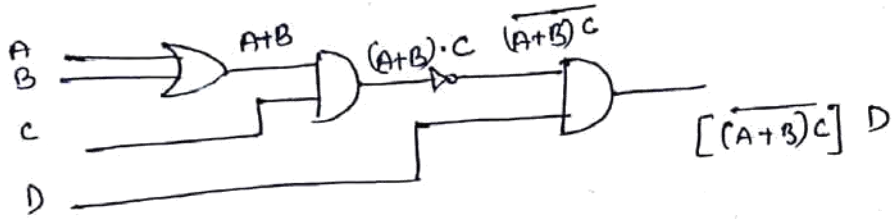
Eliminate double inversions.

→ Draw the AOI logic for the Expression $(AB+C)\overline{D}$

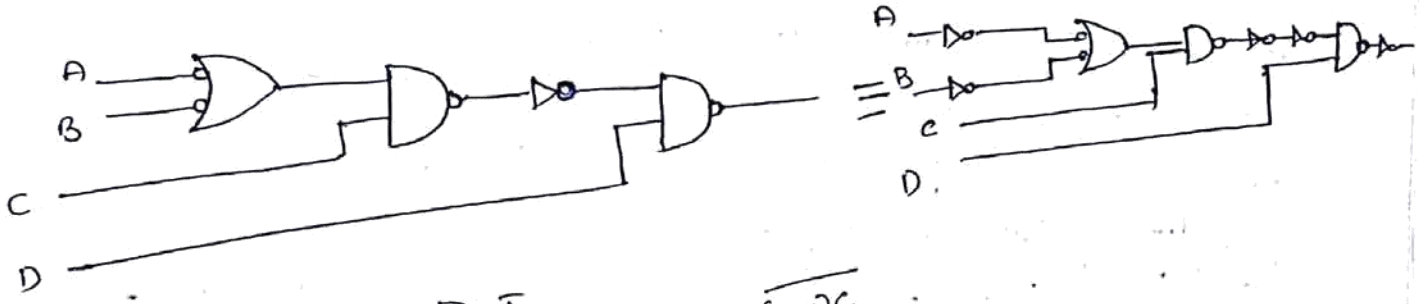
Sol



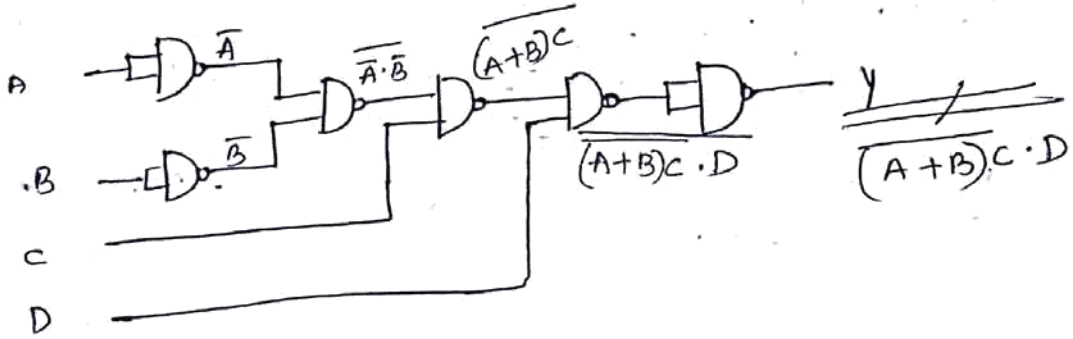
→ Given boolean Expression $y = [\overline{(A+B)C}]\,D$



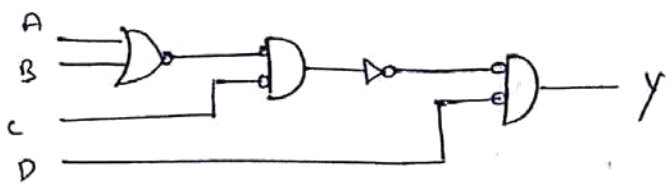
Using NAND

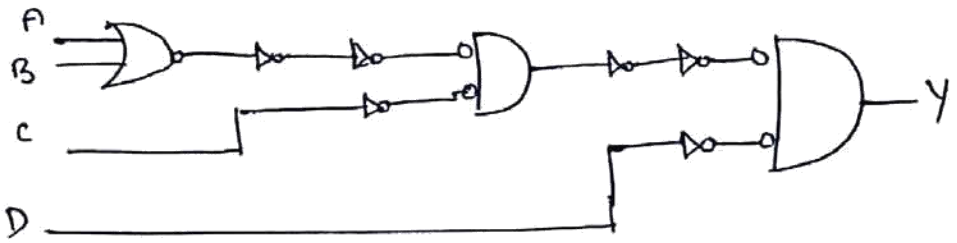


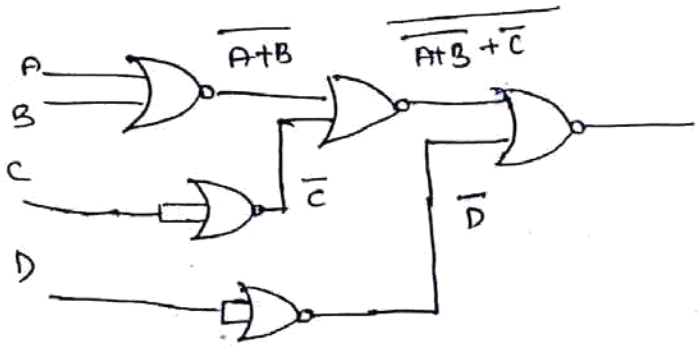
≡



NOR circuit







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

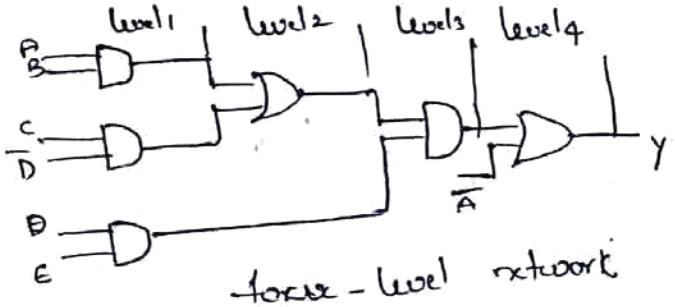
$$= \overline{(A+B)C} + \overline{D}$$

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

→ multilevel Gate Implementation.

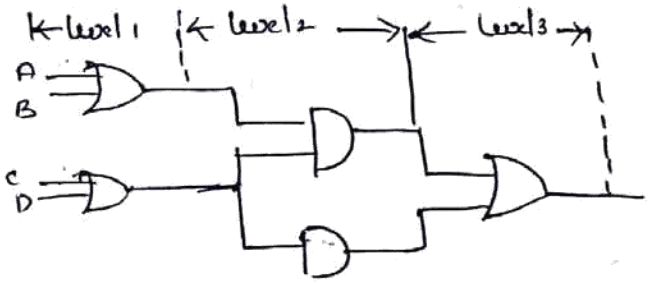
Logic gates are cascaded to get the desired output. The maximum number of gates cascaded in series between a network input and the o/p is referred to as number of levels of gates.

Thus the boolean functions written in sum-of-products form or in product of sum form are the two-level gate networks.



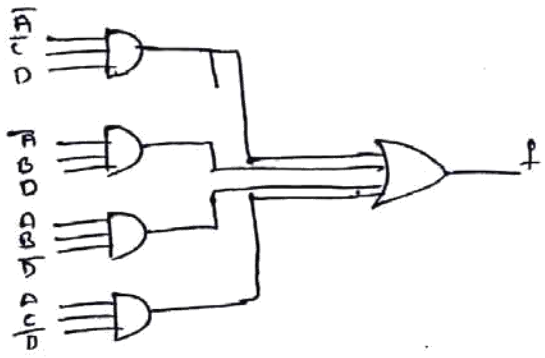
two-level network

letus consider Boolean functio



Three-level network.

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



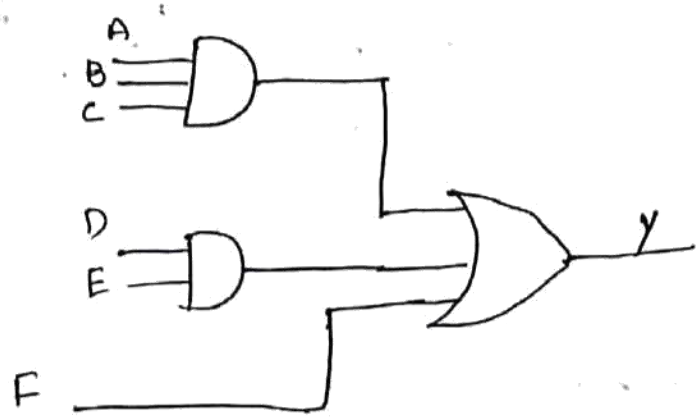
Level 2

Gates 5

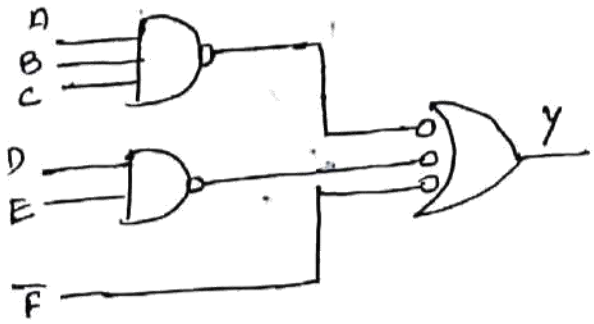
Gate inputs 16

ement the following using NAND - NAND

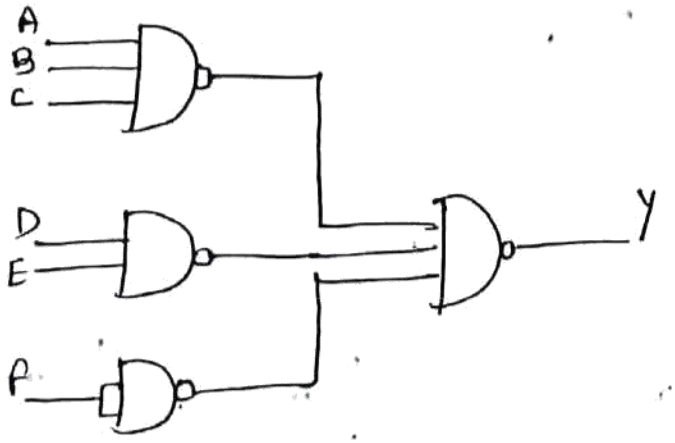
a) $Y = ABC + DE + F$



AND - OR



NAND - Bubbled OR



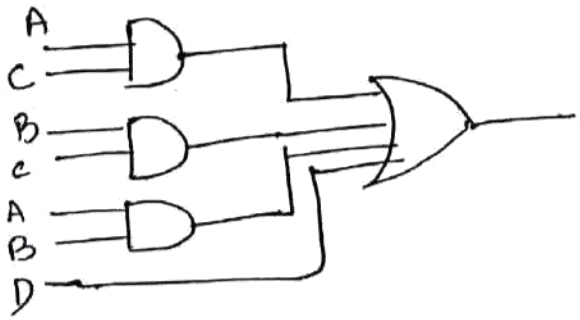
NAND - NAND.

b) $Y = AC + ABC + A'BC + AB + D$

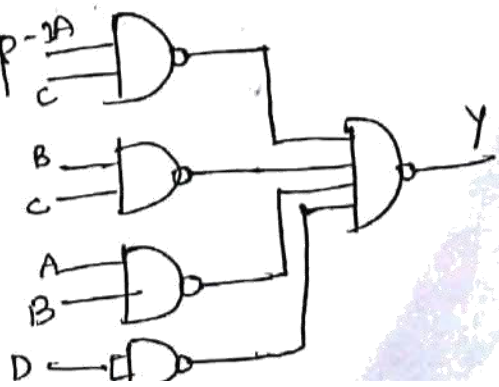
Step-1 $AC(1+B) + A'BC + AB + D$ OR $AC + BC(A+A') + AB + D$

$AC + A'BC + AB + D$. OR $AC + BC + AB + D$.

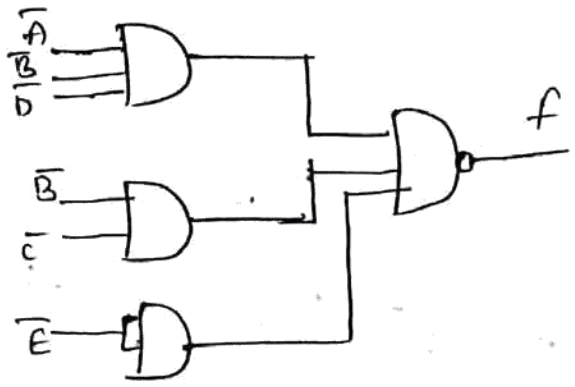
Step-2



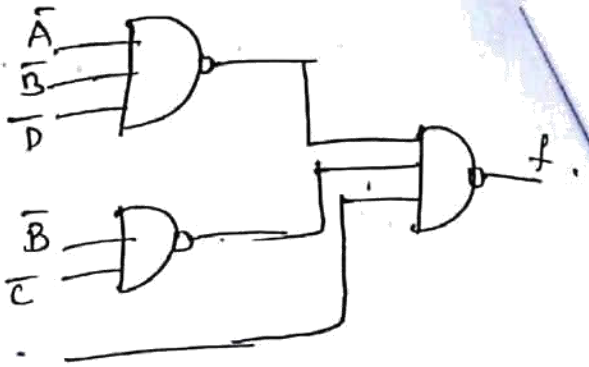
Step-2A



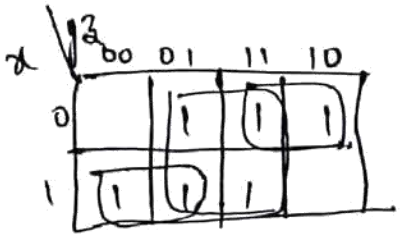
d) $f = A'B'D' + B'C' + E'$



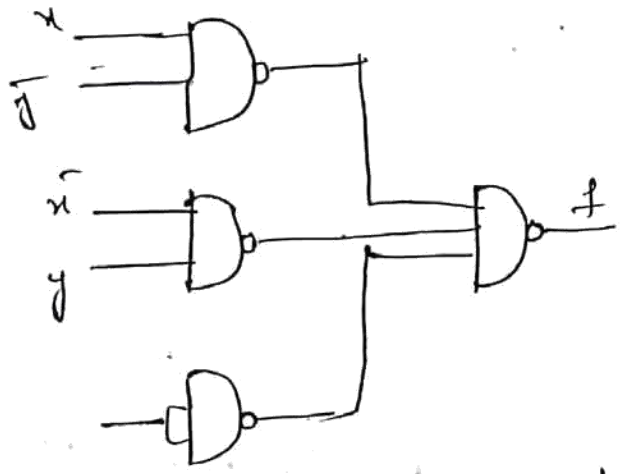
OR



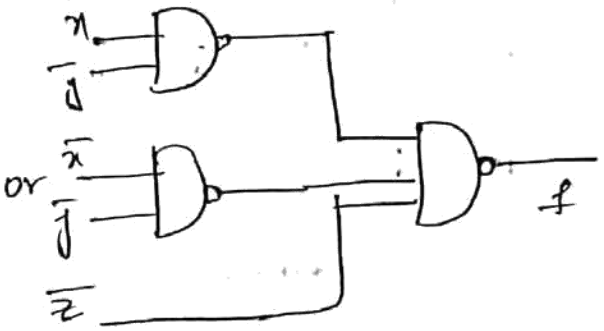
e) $f(x, y, z) = \sum 1, 2, 3, 4, 5, 7$



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



or



f) Find the Reduced pos form of the following Equation

$f(A, B, C, D) = \sum m(1, 3, 7, 11, 15) + \sum d(0, 2, 5)$ implement using NAND logic

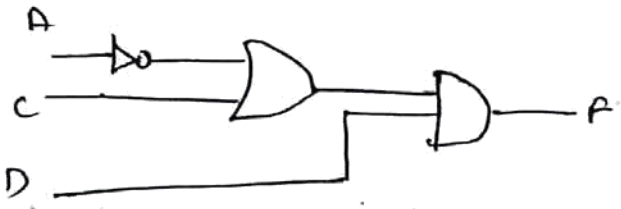
AB \ CD	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

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

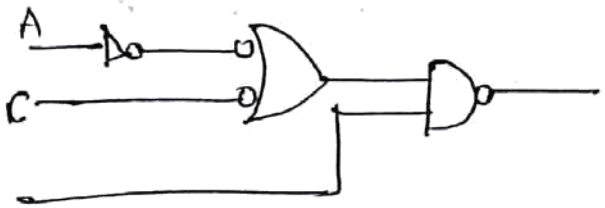
$F = \overline{\bar{D} + A\bar{C}}$

$= \bar{\bar{D}} \cdot \overline{(A\bar{C})}$

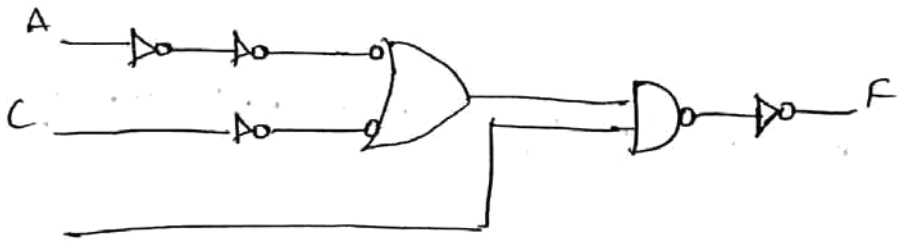
$= D(\bar{A} + C)$



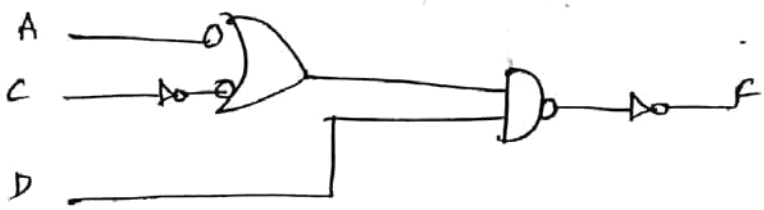
Step1 put bubbles at i/p of OR & o/p of AND.



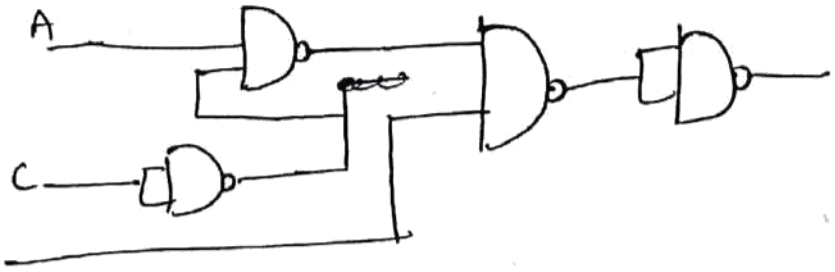
Step2 insert inverter for each bubble.



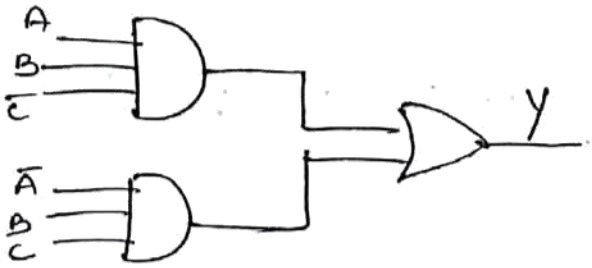
Step3 Eliminate double conversion.



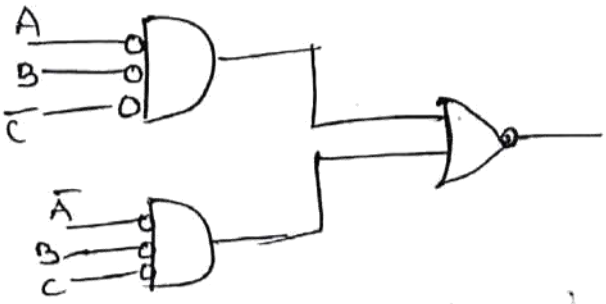
Step4 Replace bubbled OR, and inverter by NAND place



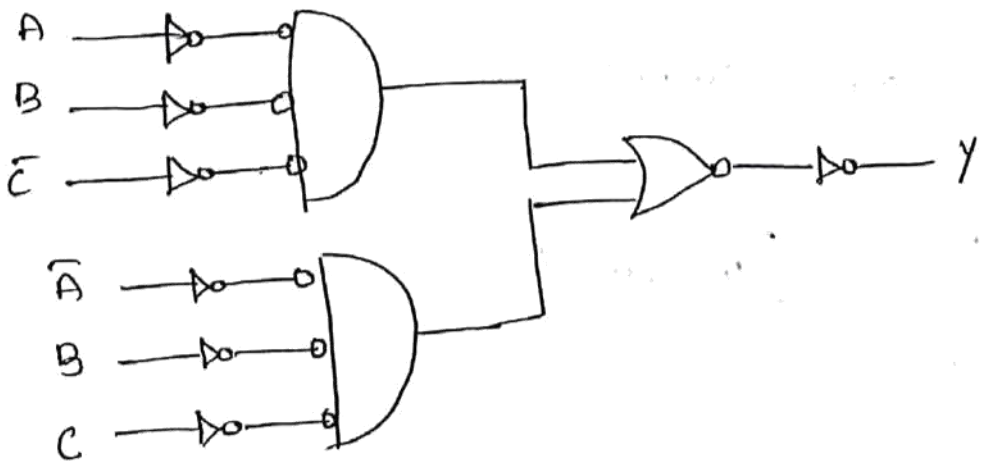
1) $AB\bar{C} + \bar{A}BC$ Using a). NOR
b) NAND



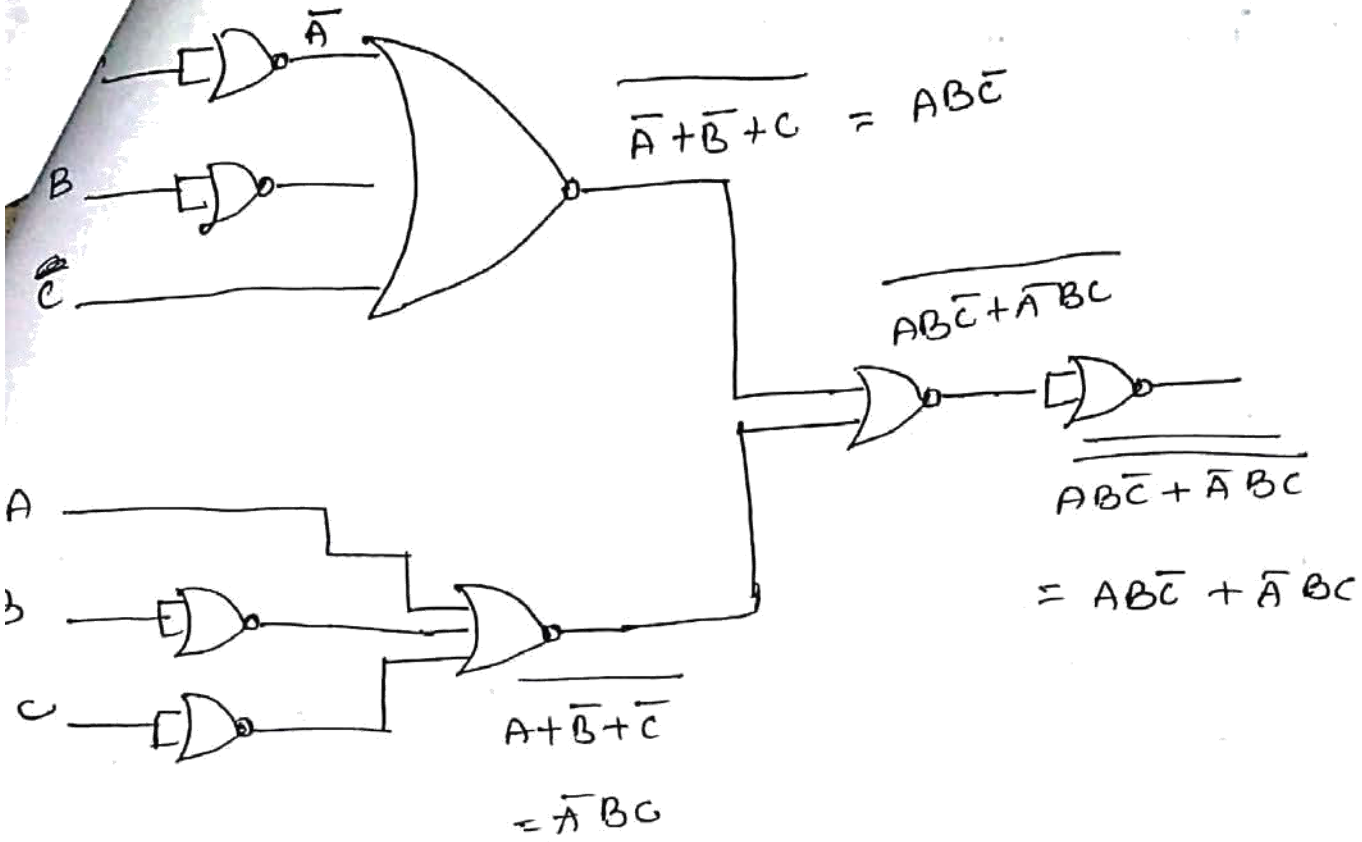
(i) Replace AND with bubbled AND.
OR with NOR.



ii) Add inverters at the i/p of bubbled AND, o/p of NOR

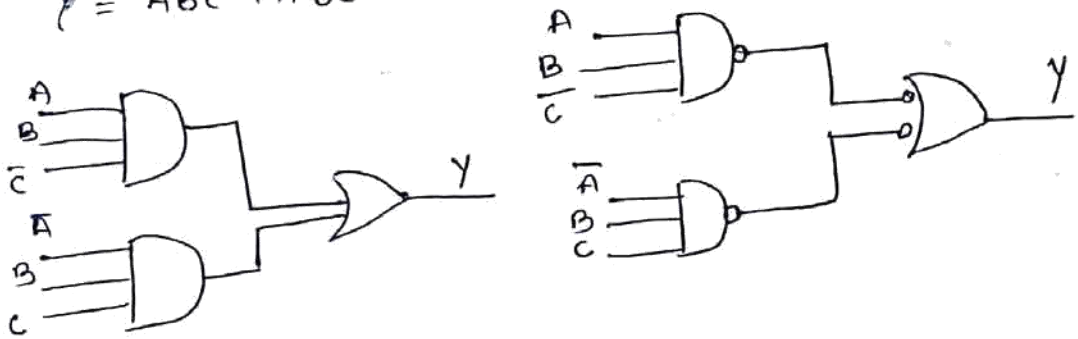


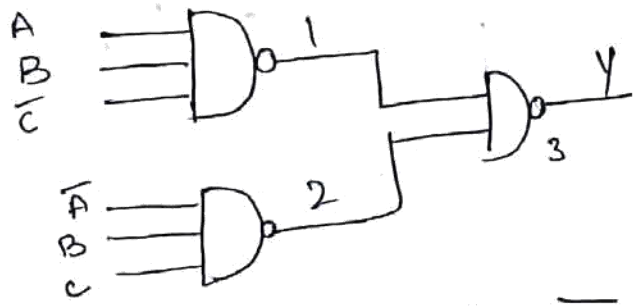
bubbled and, and inverter by NOR.



Using NAND

$Y = ABC' + A'BC$





① $\overline{AB\bar{C}} = \bar{A}+\bar{B}+C$ ② $\overline{\bar{A}BC} = A+\bar{B}+\bar{C}$

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

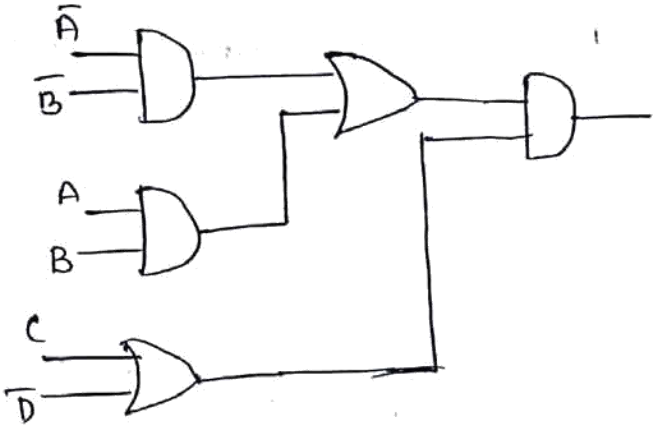
$= \overline{(\bar{A}+\bar{B}+C)} + \overline{(A+\bar{B}+\bar{C})}$

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

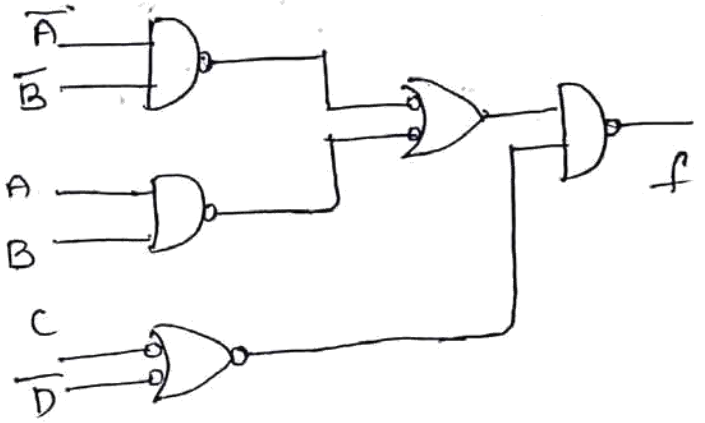
$= AB\bar{C} + \bar{A}BC$

NAND : $(A'B'+AB)\ (C+D')$

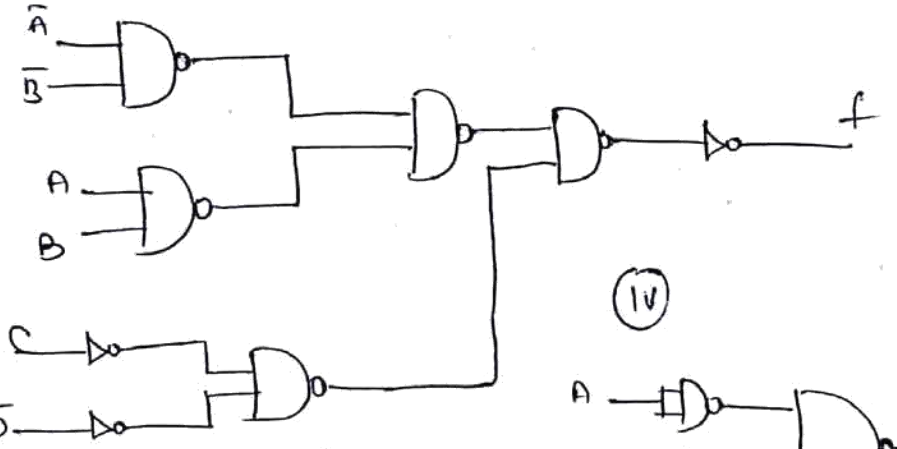
(i)



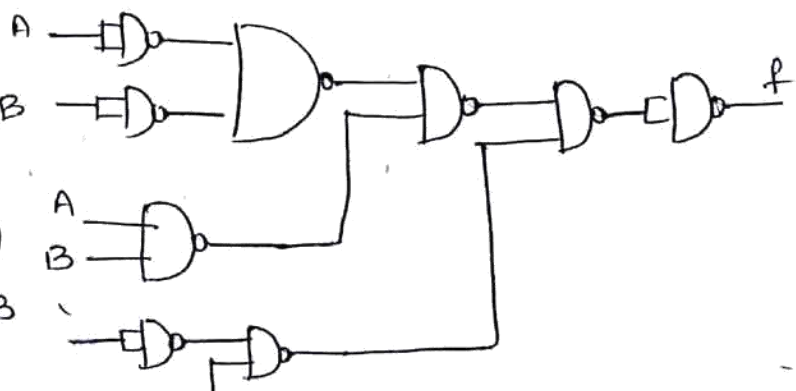
(ii)



(iii)



(iv)



① $\overline{\bar{A} \cdot \bar{B}} = A+B$ ② $\overline{AB} = \bar{A}+\bar{B}$

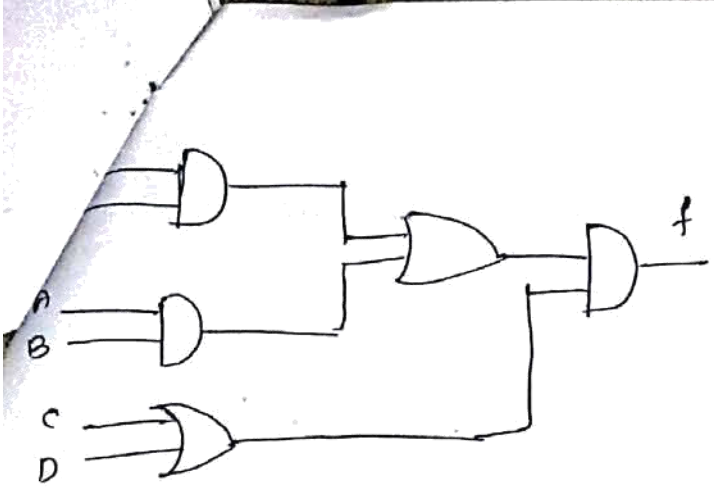
③ $\overline{\bar{C} \cdot D} = C+\bar{D}$

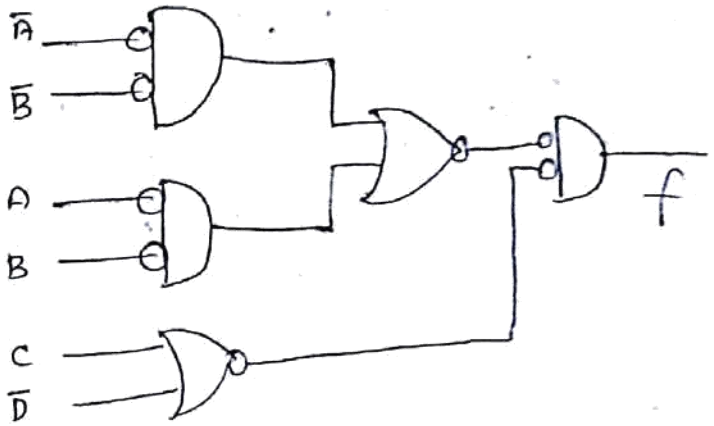
④ $\overline{(A+B)\ (\bar{A}+\bar{B})} = \overline{(A+B)} + \overline{(\bar{A}+\bar{B})}$

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

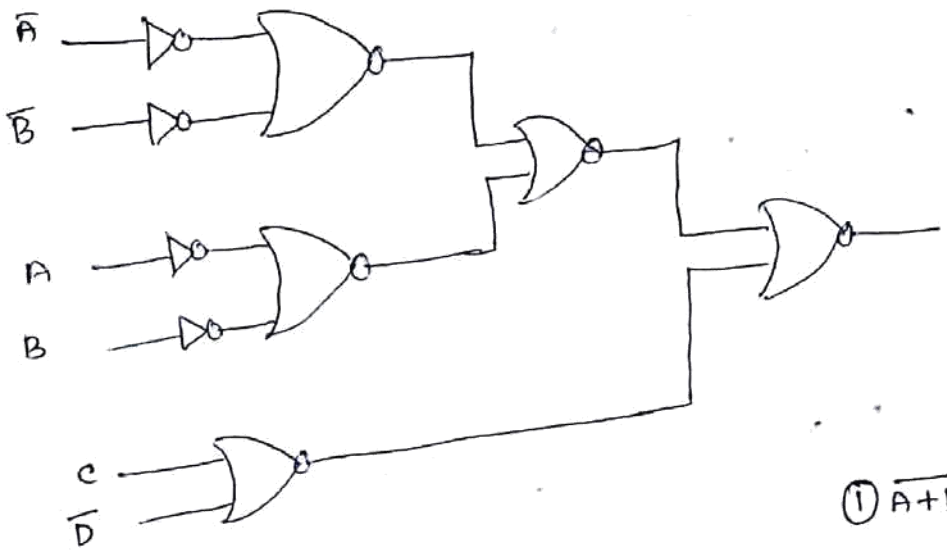
⑤ $(\bar{A} \cdot \bar{B} + A \cdot B)\ (C+\bar{D})$

⑥ $\overline{(\bar{A} \cdot \bar{B} + A \cdot B)\ (C+\bar{D})} = \overline{(\bar{A} \cdot \bar{B} + AB)}\ \overline{(C+\bar{D})}$

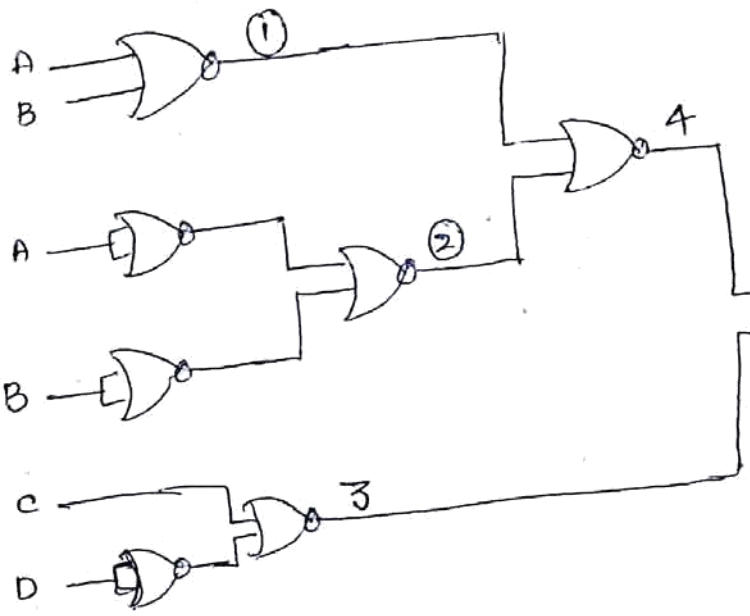




(iii)



(iv)



① $\overline{A+B} = \bar{A} \cdot \bar{B}$

② $\overline{\bar{A}+\bar{B}} = AB$

③ $\overline{C+\bar{D}} = \bar{C} \cdot D$

④ $\overline{\bar{A} \cdot \bar{B} + AB} = (\overline{\bar{A} \cdot \bar{B}})(\overline{AB})$

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

$\Rightarrow (A+B)(\bar{A}+\bar{B})$ ~~$(\overline{C+D})$~~

⑤ $\overline{\bar{A}B + A\bar{B} + \bar{C} \cdot D}$

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

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

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

UNIT NO -II

UNIT - III (MINIMIZATION OF SWITCHING FUNCTIONS)

Map Method:

For Simplification of boolean Expressions by boolean algebra, so we need better understanding of boolean laws, rules and theorems. During the process of Simplification we have to predict Each Successive step.

For these Reasons we can never Simplified expressions by boolean algebra alone.

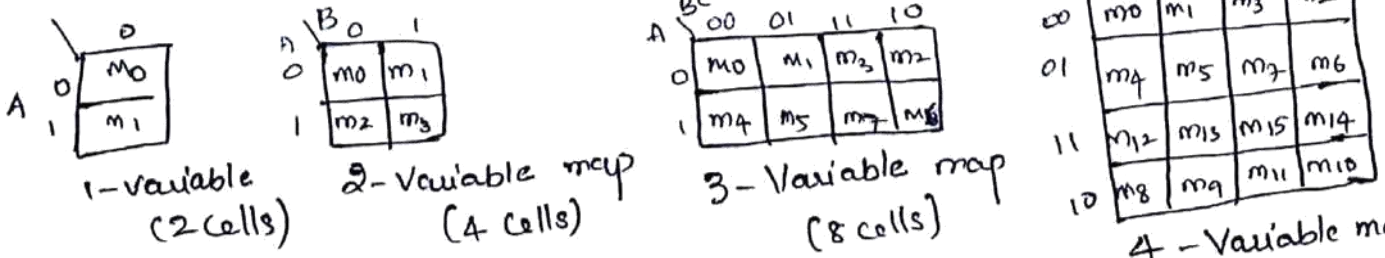
Map method gives us Systematic approach for Simplifying a boolean Expression.

The map method, first proposed by Veitch and modified by Karnaugh. hence it is known as the Veitch diagram or the Karnaugh map.

One Variable, two - Variable, Three - Variable, and four - Variable maps.

The basis of this method is a graphical chart known as Karnaugh map (K-map). it Contains boxes called cells. Each of the cell represents one of the 2^n possible products that can be formed from n Variables.

A 2-Variable map Contains $2^2 = 4$ Cells, a 3-Variable map Contains $2^3 = 8$ cells, below fig shows the 1, 2, 3, and 4 Variable maps.



1-Variable (2 Cells) | 2-Variable map (4 Cells) | 3-Variable map (8 cells) | 4-Variable map

plotting a K-map.

Logic function can be represented in various forms truth table, SOP boolean Expression and POS boolean Expression.

Representation of truth table on Karnaugh map.

K-map plotted from truth tables with 2, 3 and 4 variables. the terms which are having output 1, have the corresponding cells marked with 1's. The other cells are marked with zeros.

Ex:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

A \ B	0	1
0	0 → 1	
1	1	0

OR

A \ B	$\bar{B}$	B
$\bar{A}$	0	1
A	1	0

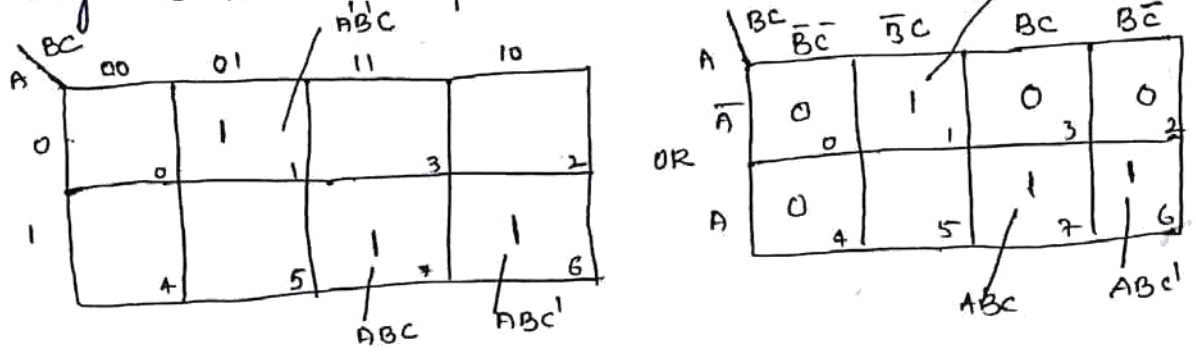
Representation of 2-variable truth table on K-map.

Sum terms Representation on Karnaugh map.

A Boolean Expression with Sum terms (Sum of product form) can be plotted with the K-map by placing a 1 in each cell corresponding to a term (minterm) in the SOP Expression. Remaing cells are filled with zeros. This is illustrated in the following Examples.

Example: plot Boolean Expression $Y = AB\bar{C} + ABC + \bar{A}\bar{B}C$ on the K-map.

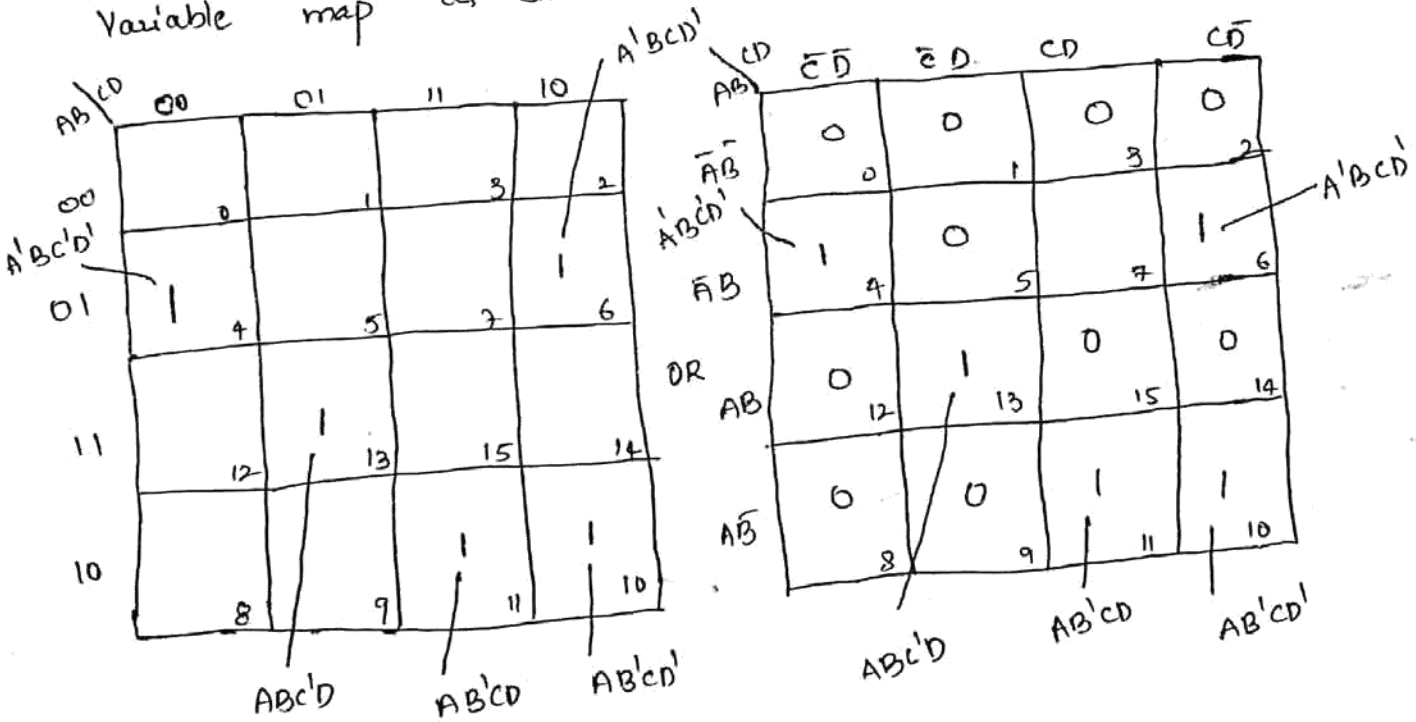
Solution: The Expression has 3 Variables and hence it can be plotted using 3-Variable map as shown below.



Example: plot Boolean Expression.

$Y = A'B'C'D' + AB'C'D' + A'BCD' + AB'CD + ABCD'$ on the Karnaugh map.

The Expression has 4-Variables and hence it can be plotted using Variable map as shown below.

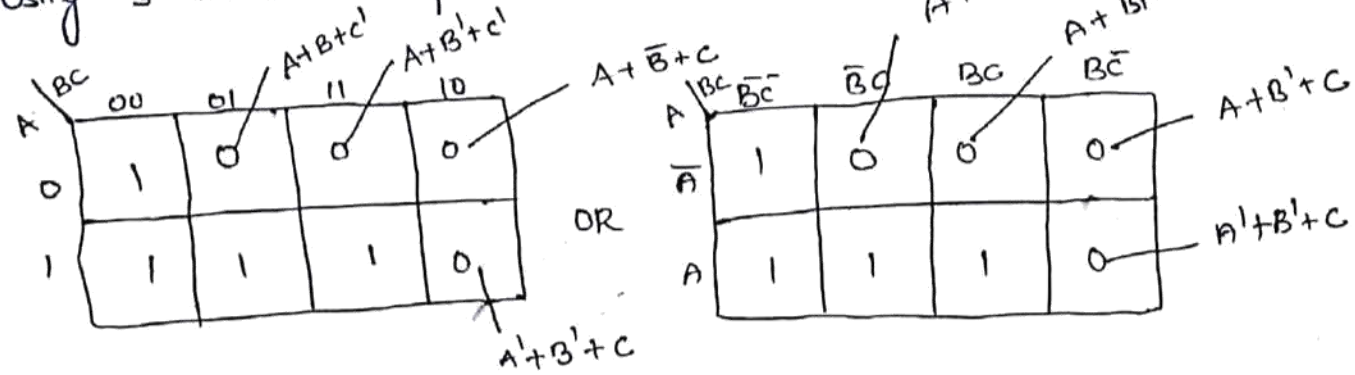


product terms Representation on Karnaugh maps.

A Boolean Expression with product terms (the product of Sum can be plotted on the Karnaugh map by placing a 0 in each corresponding to a term (maxterm) in Expression. Remaining cells are filled ones. This is illustrated in the following Examples.

Example: plot Boolean Expression $Y = (A+B+\bar{C})(A+\bar{B}+\bar{C})(\bar{A}+\bar{B}+C)(A+B$ on the Karnaugh map.

Solution: The Expression has 3 Variables and hence it can be plotted using 3-Variable map as shown below.



$m_2 = A+\bar{B}+C$ $m_3 = A+\bar{B}+\bar{C}$, $m_6 = \bar{A}+\bar{B}+C$ $m_1 = A+B+\bar{C}$

Using K-maps to obtain minimal Expressions for complete Boolean function

Once the boolean function is plotted on the Karnaugh map we have to use grouping technique to Simplify the boolean function.

Grouping nothing but a Combining terms in adjacent cells. Two cells are said to be adjacent if they conform Single change Rule.

When adjacent 1's are grouped then we get result in the SOP form otherwise we get result in the POS form.

us See Various grouping rules.

Grouping two Adjacent ones (pair).

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

A \ BC	00	01	11	10
0		1	1	
1				

The K-map contains a pair of 1's that are horizontally adjacent to Each other, the first Represents $\bar{A}\bar{B}C$ and Second $\bar{A}BC$. In this two terms only the B Variable appears in both normal and complemented form. So, by Combining these two we can Eliminate the B term and the result is $\bar{A}C$.

Ex: $A'BC + ABC = BC$

A \ BC	$\bar{B}\bar{C}$ 00	$\bar{B}C$ 01	BC 11	$B\bar{C}$ 10
$\bar{A}$ 0	0	0	1	0
A 1	0	0	1	0

- In K-map we can Combine two Vertically adjacent 1's
- In a K-map leftmost column and right most column are considered to be adjacent.

Thus, two 1's in these columns with a common row can be combined to eliminate one Variable.

- Ex: $Y = AB'C' + ABC'$

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

$= AC'$

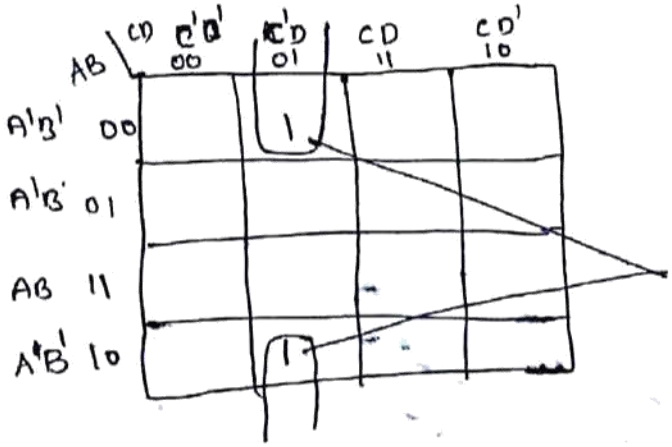
In a K-map the leftmost column and rightmost column are Considered to be adjacent. Thus, the two 1's in these columns with a common row can be combined to eliminate one variable.

Ex: $Y = A'B'C'D + AB'C'D$.

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

$= B'C'D$.

Let us see this Example, pairing is from top row and bottom of some column are combined. Eliminate Variable A, since in a map the top row and bottom row are considered to be adjacent.



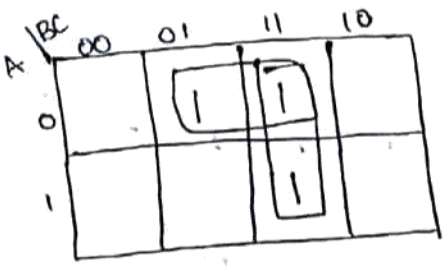
Ex: $A'B'C + A'BC + ABC$

$= A'B'C + A'BC + A'BC + ABC$ Rule $[A+A=A]$

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

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

$= A'C + BC$.



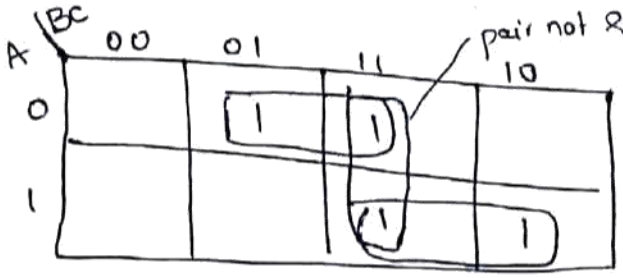
This shows a k-map that has two overlapping pair of that 1's. This shows that we can share one term between two pairs

→ Another Example

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

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

$= A'C + AB$ $[A'+A=1]$.

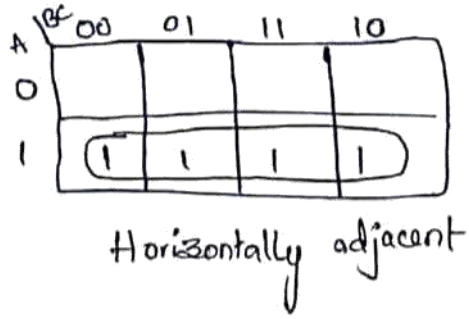


This fig shows a k-map where three group of pairs can be formed. But only two pairs are Enough to include all is present in the k-map.

Grouping of 4 adjacent ones (Quad)

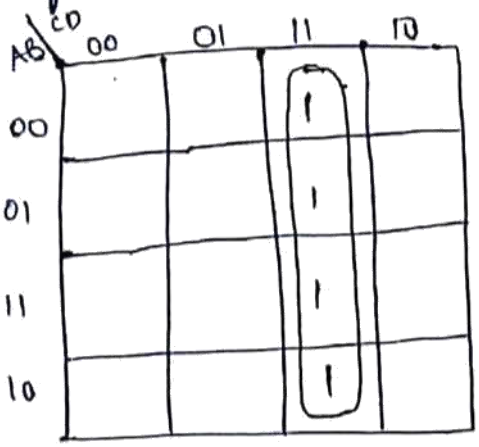
- In a K-map we can group four adjacent 1's. The resultant group is called Quad.

- four 1's may be horizontally adjacent or vertically adjacent.



Horizontally adjacent

$Y = A$



Vertically adjacent.

$Y = CD$

If a K-map contains four 1's in a square. and they are considered as adjacent to each other. [fig ①]

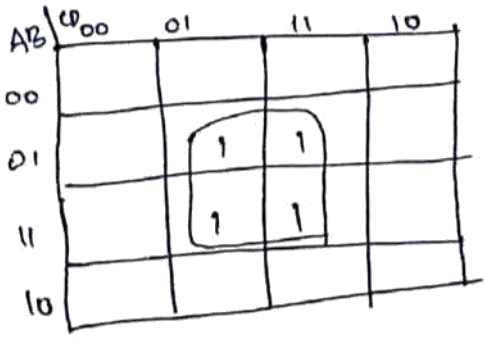


fig ① $Y = BD$

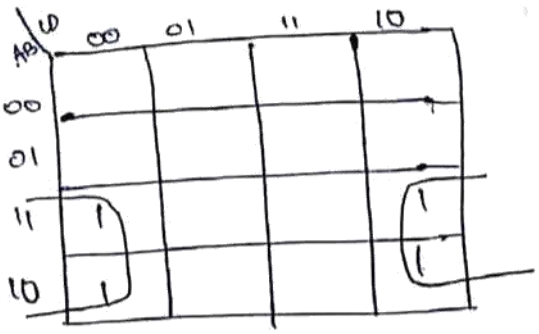


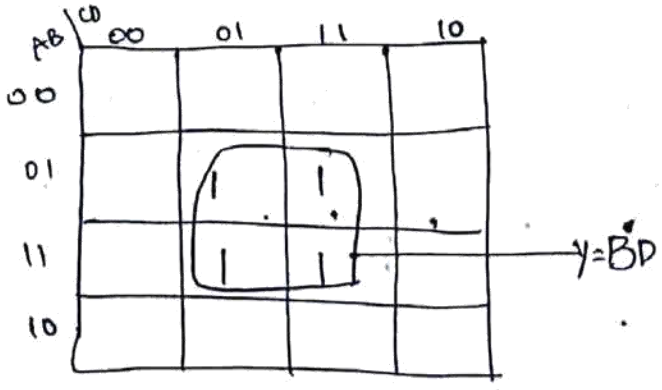
fig ②

$Y = AD$.

- The four 1's in fig ② are also adjacent, because the top and bottom rows are considered to adjacent to each other and the leftmost and rightmost columns are also adjacent to each other.

Ex: $Y = \bar{A}B\bar{C}D + \bar{A}BCD + AB\bar{C}D + ABCD$.

Solution: As there are 4-Variable total no. of cells $= 2^4 = 16$.



3. Grouping Eight adjacent- ones (octet):

— Grouping of eight adjacent 1's is called as octet.

— either you can group horizontally, vertically.

— When an octet is combined in a four variable map, three of four variables are eliminated because only one variable remains unchanged.

Simplification of SOP Expression:

- The combination of pairs, quads, and octets on a Karnaugh map can be used to obtain a simplified expression.

— A pair of 1's eliminates one variable.

— A quad of 1's eliminates two variables

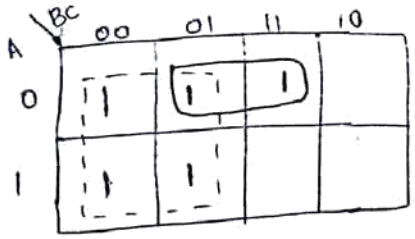
— An octet of 1's eliminates three variables.

— In general, when a variable appears in both complemented and uncomplemented form with in a group. that variable eliminated from a resultant expression.

Generalised procedure to Simplify boolean Expression

1. plot the K-map and place 1's in those cells corresponding to the 1s in the truth table or Sum of product Expression. place as in other cells.
2. check the K-map for adjacent 1s and then circle those 1's which are not adjacent to any other 1s. These are called isolated 1s.
3. check for those 1s which are adjacent to only one other 1 and Encircle such pairs.
4. Check for quards and octets of adjacent 1s even if it contains some 1s that have already been Encircled, while doing this make sure that there are minimum number of groups.
5. Combine any pairs necessary to include any 1's that have not yet been grouped.
6. from the Simplified Expression by Summing product terms of all the groups.

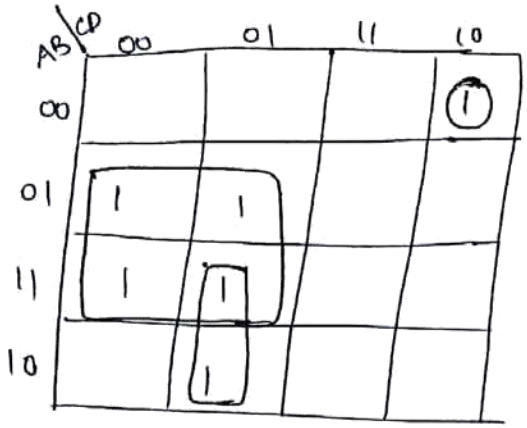
Example: Minimize the Expression $Y = A\bar{B}C + \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C}$



1. first see for isolated ones. In this there are no isolated ones.
2. See for adjacent pair of 1s — 1
3. See for Quad — 1

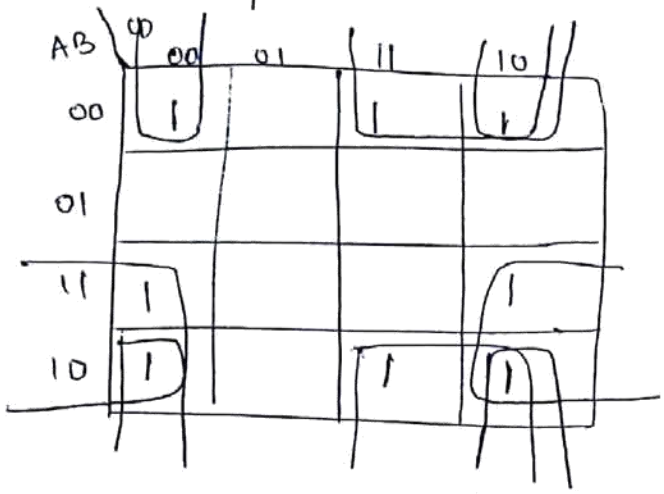
$$Y = \bar{B} + \bar{A}C$$

2) minimize the Exp $Y = \bar{A}B\bar{C}\bar{D} + \bar{A}BC\bar{D} + AB\bar{C}\bar{D} + AB\bar{C}D + A\bar{B}CD + \bar{A}\bar{B}C\bar{D}$



$Y = BC + AB'C + A'B'CD'$

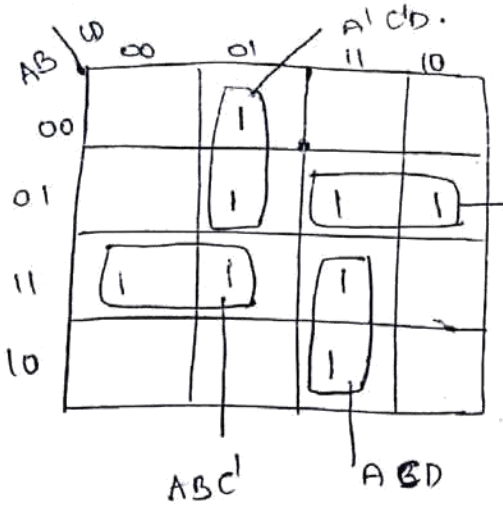
3) minimize the Exp $Y = A'B'CD' + ABCD' + AB'CD' + AB'CD + ABCD + A'B'CD + A'B'C'D'$



$Y = B'D' + AB'C + AD'$

4) Reduce the following to its minimum SOP form.

$Y = A'B'CD + A'BCD + A'BCD + A'BCD' + AB'CD' + ABCD + ABCD + AB'CD$



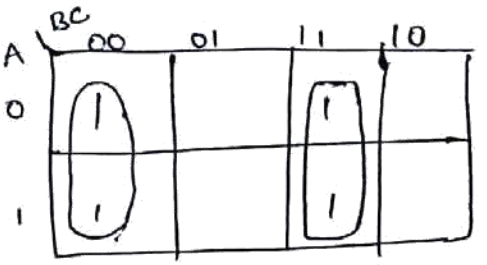
— There are no isolated 1's

— There is no octet, but there is a quad. However, all 1's in the quad have already been grouped. Therefore this quad is ignored.

$Y = A'C'D + A'BC + ABC' + ACD.$

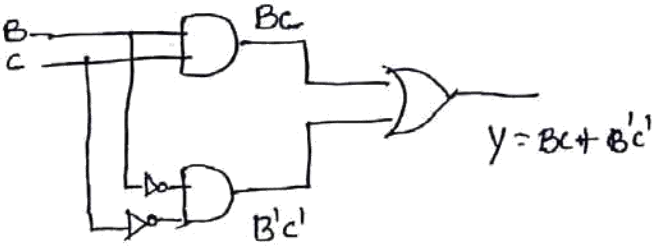
Implement the logic function specified by the truth table (Below) using the Karnaugh map method. y is the output variable and A, B and C are the input variables.

A	B	C	y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



$B'C' + BC$.

The boolean Expression can be implemented as follows



Simplification of POS Expression:-

Once the Expression is plotted on the K-map instead of making the group of ones, we have to make groups of zero.

Steps

1. plot the K-map and place 0's in those cells corresponding to the 0s in the truth table or maxterms in the products of sum Expressio
2. check the K-map for adjacent 0's and Encircle those 0's which are not adjacent to any other 0's. These are called isolated as
3. check for those 0's which are adjacent to only one other 0 and Encircle Such pairs.

4. Check for quads and octets of adjacent 0's even if it contains that have already been encircled. While doing this make sure there are minimum number of groups.

5. Combine any pairs necessary to include any 0's that have not been grouped.

6. From the Simplified SOP Expression for $\bar{F}$ by summing product terms of all the groups.

Note: The Simplified SOP necessary to include any 0s Expression is in the complemented form because we have grouped 0s to Simplify the Expression.

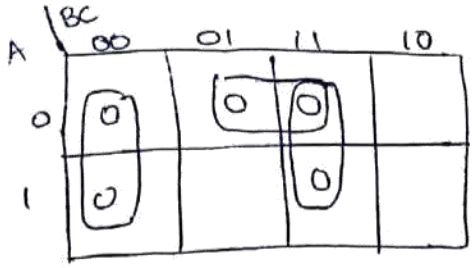
7. Use Demorgan's theorem on $\bar{F}$ to produce the Simplified Expression in POS form.

Ex: minimize the Expression

$$Y = (A+B+\bar{C})\ (A+\bar{B}+\bar{C})\ (\bar{A}+\bar{B}+\bar{C})\ (\bar{A}+B+C)\ (A+B+C) \cdot \text{(POS) (maxterms}$$

Solution: $(A+B+\bar{C}) = m_1$, $(A+\bar{B}+\bar{C}) = m_3$, $(\bar{A}+\bar{B}+\bar{C}) = m_7$

$(\bar{A}+B+C) = m_4$, $(A+B+C) = m_0$.



$$\bar{y} = B'C' + BC + A'C$$

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

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

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

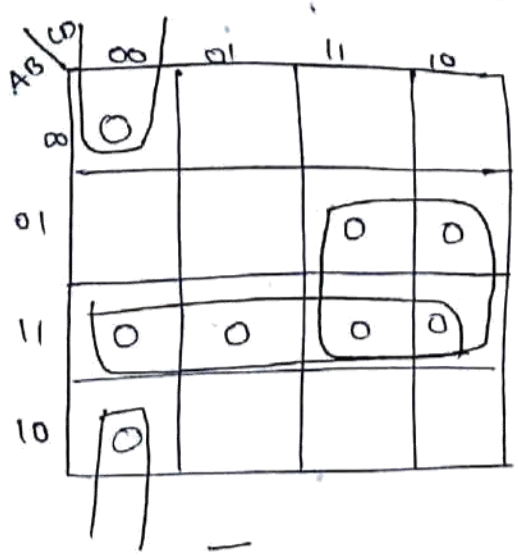
Minimize the following Eqn in POS form.

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

Solution: $(\bar{A}+\bar{B}+C+D) = M_{12}$ $(\bar{A}+\bar{B}+\bar{C}+D) = M_{14}$ $(\bar{A}+\bar{B}+\bar{C}+\bar{D}) = M_{15}$

$(\bar{A}+B+C+D) = M_8$, $(A+\bar{B}+\bar{C}+D) = M_6$, $(A+\bar{B}+\bar{C}+\bar{D}) = M_7$

$(A+B+C+D) = M_0$ and $(\bar{A}+\bar{B}+C+\bar{D}) = M_{13}$



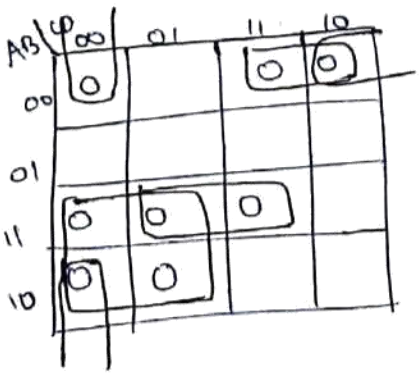
There is no isolated 0's.

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

$$Y = \bar{\bar{Y}} = \overline{BC + AB + B'C'D'}$$
$$= (\bar{B}+\bar{C})(\bar{A}+\bar{B})(\bar{B}+C+D)$$
$$= (\bar{B}+\bar{C})(\bar{A}+\bar{B})(B+C+D)$$

Ex: Reduce the following function using K-map technique.

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



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

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

Don't Care Conditions

In some logic ckts, certain input conditions never occur. ∴ the corresponding o/p never appears. In such cases the o/p is not designed, defined, it can be either High or low. These output levels are indicated by 'x' or 'd' in the truth table are called don't care o/p's.

- A ckt designer is free to make the o/p for any "don't care" condition either a '0' or '1' in order to produce the simplest o/p Expression.
- It is important to decide which don't cares to change to and which to 1 to produce the best K-map grouping.

Ex: find the reduced SOP form of the following function

$f(A,B,C,D) = \Sigma m(1,3,7,11,15) + \Sigma d(0,2,4)$

AB \ CD	00	01	11	10
00	X	1	1	X
01	X	0	1	0
11	0	0	1	0
10	0	0	1	0

To form a quad of cells 0, 1, 2, and 4 the don't care conditions 0 and 2 are replaced by 0 since it is not required to form any group.

AB \ CD	00	01	11	10
00	1	1	1	1
01	0	0	1	0
11	0	0	1	0
10	0	0	1	0

$= A'B' + CD.$

Ex: Reduce the following fn using K-map.

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

AB \ CD	00	01	11	10
00	0	0	0	0
01	X	1	X	X
11	1	1	X	X
10	0	X	0	0

To form a octet of cells 4, 5, 6, 7, 12, 13, 14 and 15 the don't care conditions 4, 14, and 15 are replaced by 1s. The remaining don't care condition 9 is replaced by 0 to get simplified fn.

AB \ CD	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	0	0	0	0

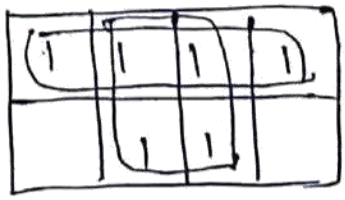
$f(A,B,C,D) = B$

Ex:- Implement the following using logic gates.

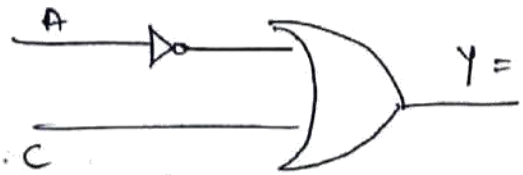
$$f(A,B,C) = \sum (0,1,3,7) + \sum d(2,5)$$

A \ BC	00	01	11	10
0	1	1	1	X
1	0	X	1	0

To form two quads both don't care conditions are replaced by 1s and we get.



$= \bar{A} + C$



$Y = \bar{A} + C$

Five Variable K-map:-

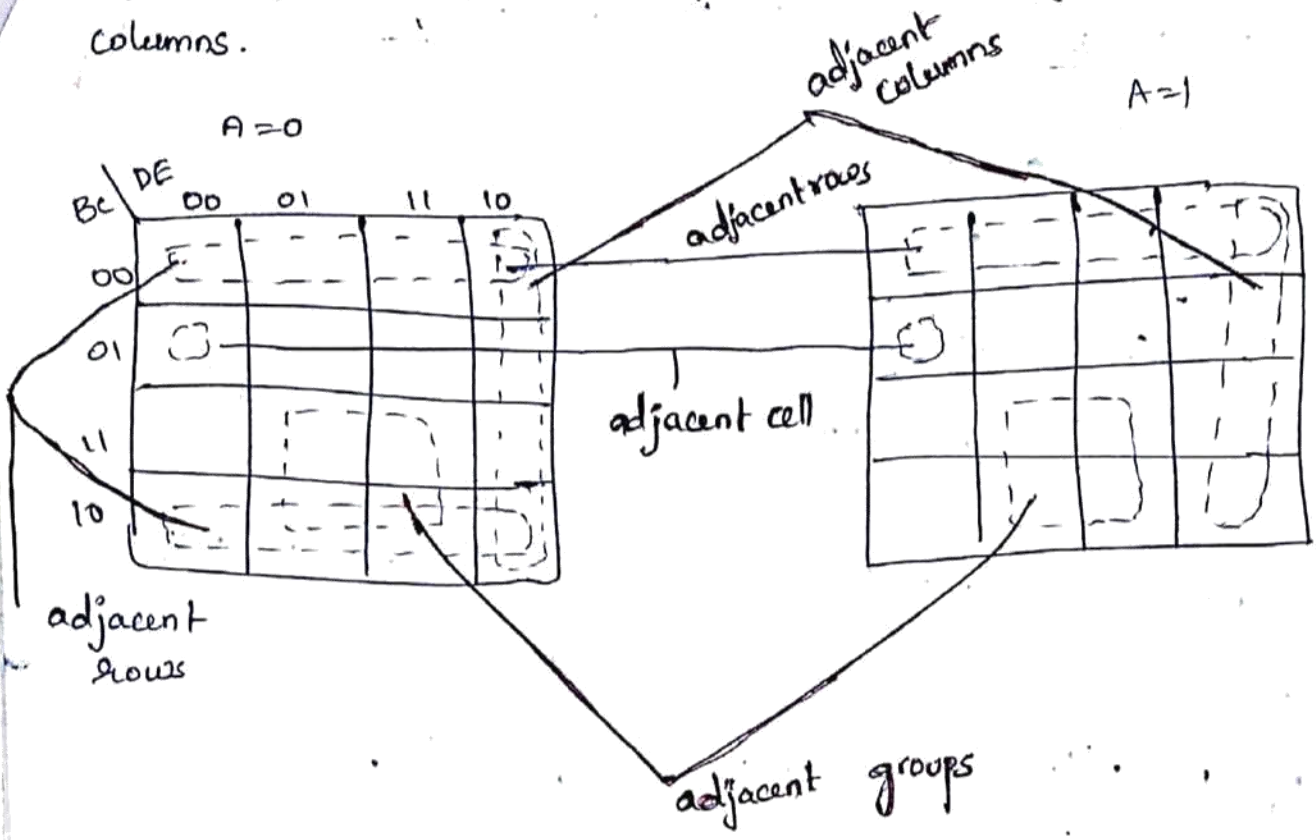
- A 5-Variable K-map require $2^5 = 32$ cells, but adjacent are difficult to identify on a single 32-cell map.

∴ Two 16-bit K-maps are used. If the Variables A, B, C, D and E two identical 16-cell maps containing B, C, D, E can be constructed. One map is used for A and other for $\bar{A}$.

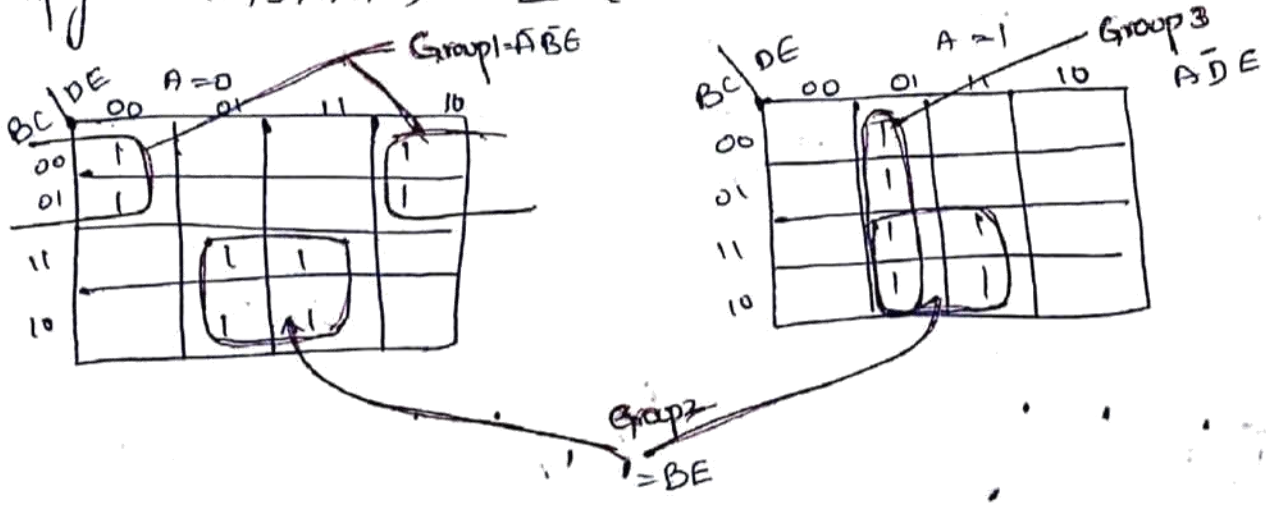
- In order to identify the adjacent grouping in the five Variable we must imagine the two maps superimposed on one another.
- Every cell in one map is adjacent to the corresponding cell in the other map, because only one Variable changes between such corresponding cells.
- Every row on one map is adjacent to the corresponding row
- The rightmost and leftmost columns within each 16-cell map are

jacent, just as they are in any 16-cell map, as are the top and bottom rows.

However, the rightmost column of one map is not adjacent to the leftmost column of other map. Since those are not corresponding columns.

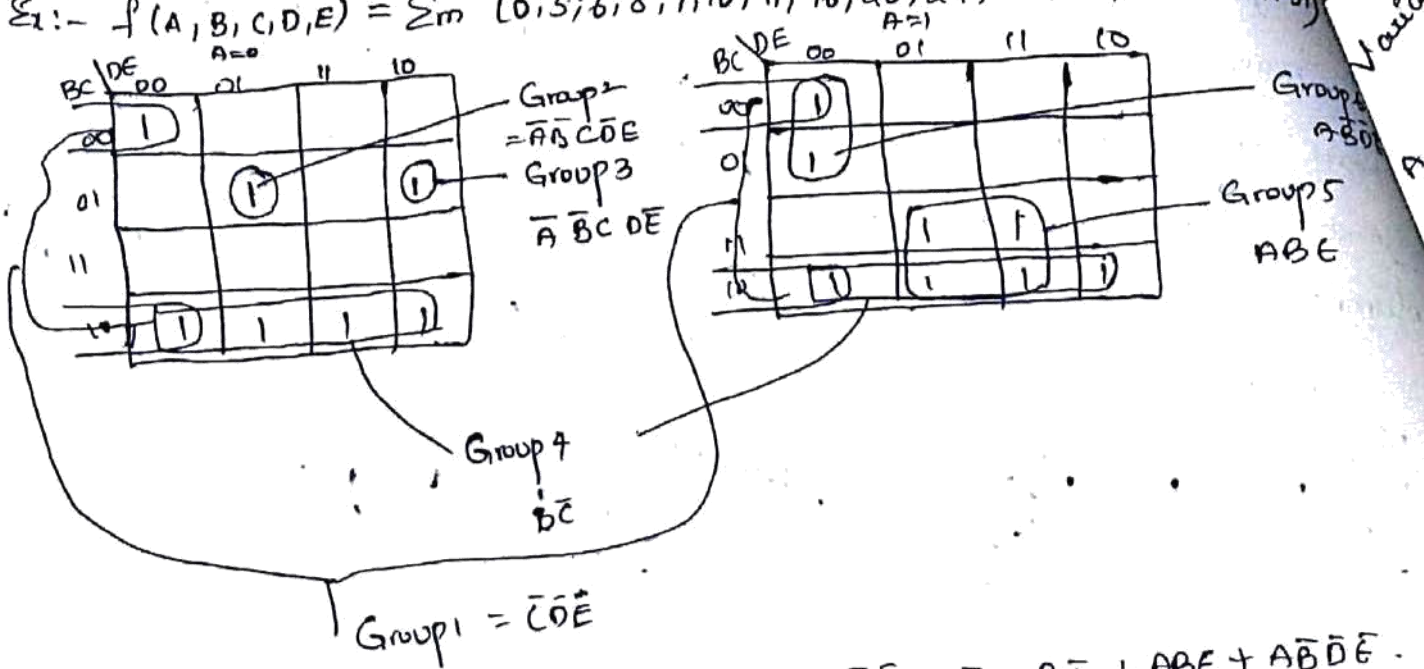


Simply $f(A,B,C,D,E) = \sum m\,(0,2,4,6,9,11,13,15,17,21,25,27,29,31)$



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

Ex:- $f(A,B,C,D,E) = \Sigma m\ (0,5,6,8,9,10,11,16,20,24,25,26,27,29,31)$



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

Ex: $f(A,B,C,D,E) = \Sigma m\ (1,4,8,10,11,20,22,24,25,26) + d\ (0,12,16,17)$

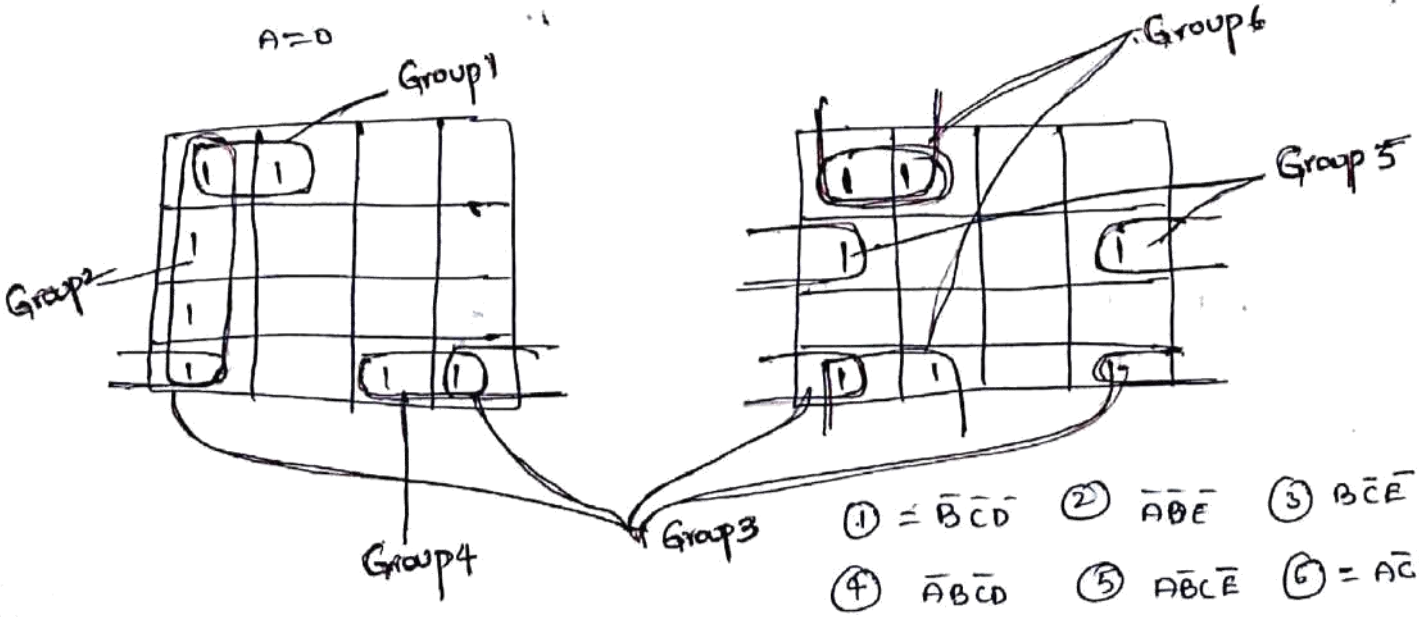
A=0

BC \ DE	00	01	11	10
00	X	1		
01	1			
11	X			
10	1		1	1

A=1

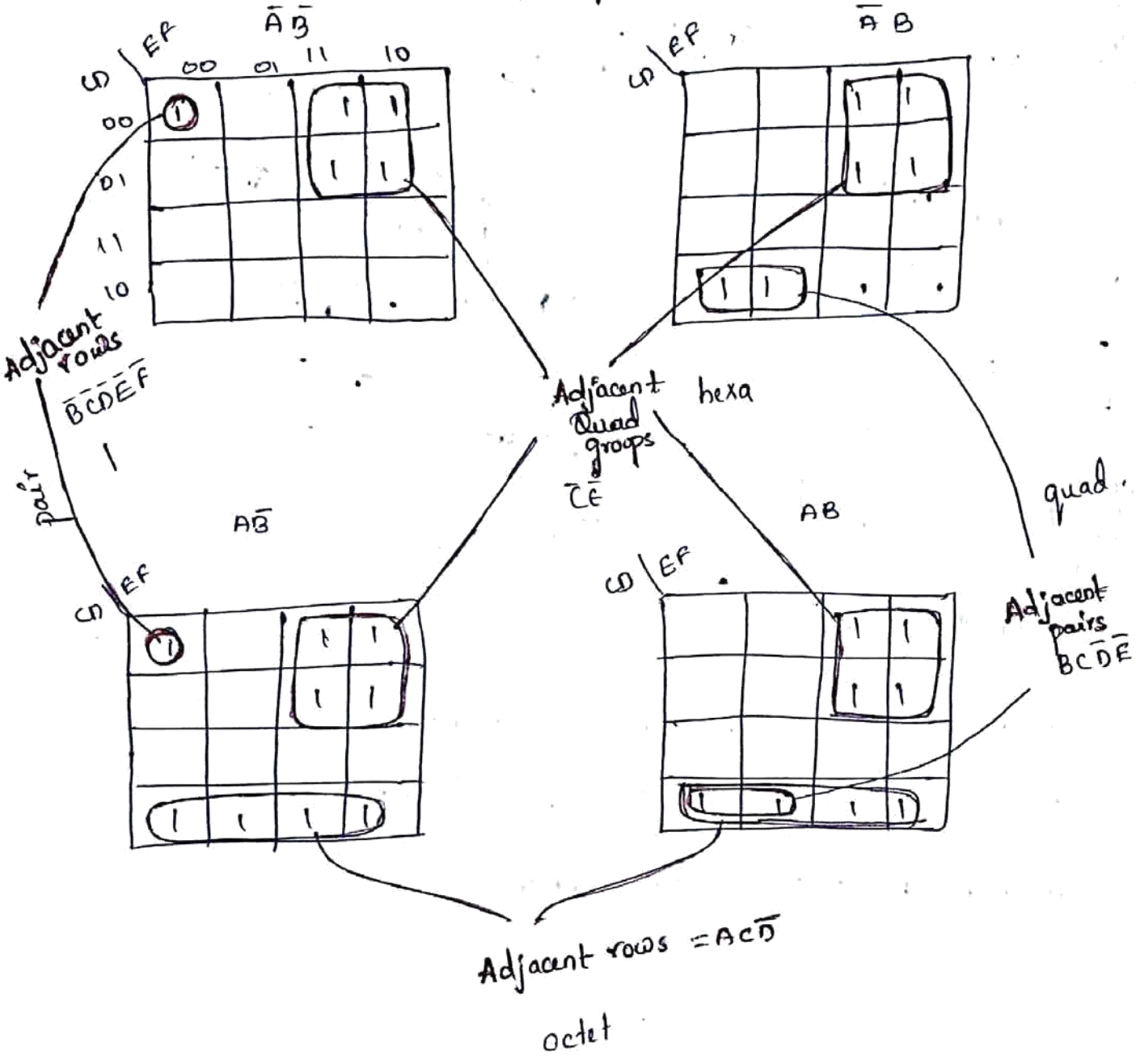
BC \ DE	00	01	11	10
00	X	X		
01	1			1
11				
10	1	1		1

In order to minimize the given Exp all don't Care. has to be taken as 1's.



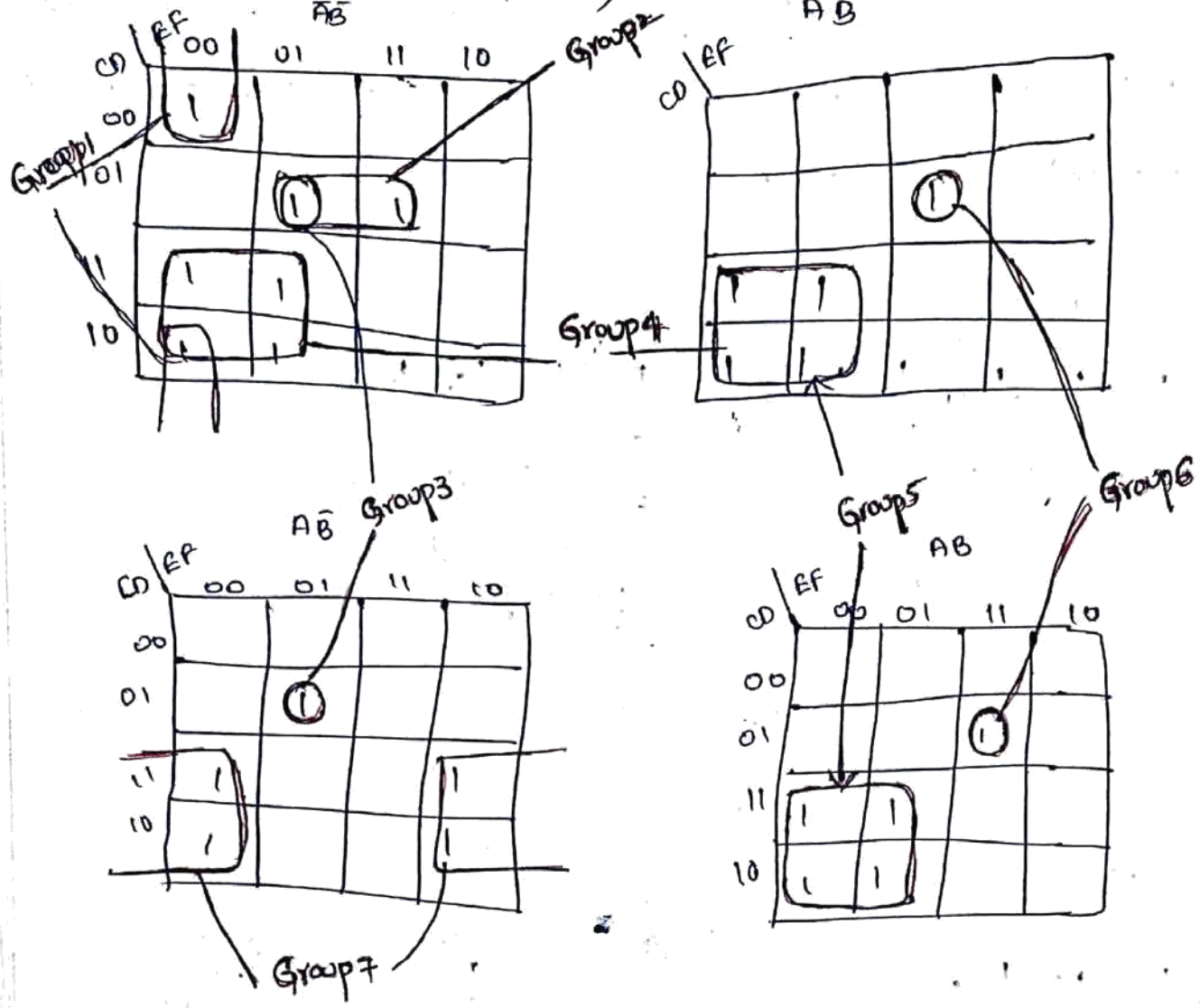
Variable K-map :-

A 6-Variable K-map requires $2^6 = 64$ cells. These cells are divided into four identical 16-cell maps. If the variables are A, B, C, D, E and F 16-cell maps contain C, D, E and F and each 16-cell map represents four combinations of A and B.



Ex: Simplify the Boolean fn

$f(A,B,C,D,E,F) = \sum m(0,5,7,8,9,12,13,23,24,25,28,29,37,40,46,55,56,57,60,61)$



① $= \bar{A}\bar{B}\bar{D}\bar{E}\bar{F}$ ② $\bar{A}\bar{B}\bar{C}DF$ ③ $\bar{B}CD\bar{E}F$

④ $\bar{A}C\bar{E}$ ⑤ $BC\bar{E}$ ⑥ $B\bar{C}DEF$ ⑦ $A\bar{B}CF$

$f(A,B,C,D,E,F) = \bar{A}\bar{B}\bar{D}\bar{E}\bar{F} + \bar{A}\bar{B}\bar{C}DF + BCDEF$

Quine - Mc Cluskey Method :-

The minimization of Boolean Expressions using K-maps is usually to a maximum of Six variables.

The Quine - mc cluskey method, also known as the tabular method. It is a more systematic method of minimizing expressions of even larger number of variables.

principle :- The minterms whose binary Equivalents differ only in one place can be combined to reduce the minterms.

Example: obtain the set of prime implicants for the boolean Expression $f = \sum m (0, 1, 6, 7, 8, 9, 13, 14, 15)$ using the tabular method.

Step 1: list all minterms in binary form

minterm	Binary Representation
m_0	0000
m_1	0001
m_6	0110
m_7	0111
m_8	1000
m_9	1001
m_{13}	1101
m_{14}	1110
m_{15}	1111

Step2: Arrange minterms according to no. of 1's.

minterm	Binary Rep.
m_0	0000 ✓
m_1	0001 ✓
m_8	1000 ✓
m_6	0110 ✓
m_9	1001 ✓
m_7	0111 ✓
m_{13}	1101 ✓
m_{14}	1110 ✓
m_{15}	1111 ✓

Step3: Compare Each term in Each group with next higher group terms. If they differ in single variable, mark that position with '-'

minterm	Binary Rep.
0, 1	000 - ✓
0, 8	- 000 ✓
1, 9	- 001 ✓
8, 9	100 - ✓
6, 7	011 - ✓
6, 14	- 110 ✓
9, 13	1 - 01
7, 15	- 111 ✓
13, 15	11 - 1 ✓
14, 15	111 - ✓

Step4: Repeat step 3

minterm	Binary Rep.
0, 1, 8, 9	- 00 -
0, 8, 1, 9	- 00 -
6, 7, 14, 15	- 11 -
6, 14, 7, 15	- 11 -

Step5: List P.I

P.I	Binary Rep.
9, 13 (1-01)	1-01
13, 15 (11-1)	11-1
0, 1, 8, 9 (-00-)	-00-
6, 7, 14, 15 (-11-)	-11-

Step6: EPI Chart (essential prime implicant)

EPI / minterms	m_0	m_1	m_6	m_7	m_8	m_9	m_{13}	m_{14}	m_{15}
✓ 9, 13 either or						⊙	⊙		
13, 15							⊙		⊙
✓ 0, 1, 8, 9	⊙	⊙			⊙	⊙			
✓ 6, 7, 14, 15			⊙	⊙				⊙	⊙

Check for Single dot in a column:- 0, 1, 8, 14, 6, ~~7~~ which are not cover all minterms, so check for double dot

double dot : 9, 13 (1-01)

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

(2) $f(A,B,C,D,E) = \Sigma m(0,2,4,5,6,7,8,10,14,17,18,21,29,31) + \Sigma d(11,20,22)$

* When a Switching function has don't cares, we set them Equal to 0 and find P.I.

* In determining E.P.I and minimal cover of the function we take all don't cares Equal to 0. that is the don't care decimal number donot appear in P.I table.

Sol: Step1: List all minterms with binary Equivalents.

Step2: Arrange the minterms according to no. of 1's.

minterm	Binary Rep.	
0	00000	✓
2	00010	✓
4	00100	✓
8	01000	✓
5	00101	✓
6	00110	✓
10	01010	✓
17	10001	
18	10010	✓
20	10100	✓
7	00111	✓
11	01011	✓
14	01110	✓
21	10101	✓
22	10110	✓
29	11101	✓
31	11111	✓

2) Compare each term in each group with next higher order term.

minterm	Binary representation	
0,2	0 0 0 - 0	✓
0,4	0 0 - 0 0	✓
0,8	0 - 0 0 0	✓
2,6	0 0 - 1 0	✓
2,10	0 - 0 1 0	✓
2,18	- 0 0 1 0	✓
4,5	0 0 1 0 -	✓
4,6	0 0 1 - 0	✓
4,20	- 0 1 0 0	✓
8,10	0 1 0 - 0	✓
5,7	0 0 1 - 1	✓
5,21	- 0 1 0 1	✓
6,7	0 0 1 1 -	✓
6,14	0 - 1 1 0	✓
6,22	- 0 1 1 0	✓
10,11	0 1 0 1 -	
10,14	0 1 - 1 0	✓
17,21	1 0 - 0 1	
18,22	1 0 - 1 0	✓
20,21	1 0 1 0 -	✓
20,22	1 0 1 - 0	✓
21,29	1 - 1 0 1	
29,31	1 1 1 - 1	

Step4: Repeat step3

minterms	Binary rep.
0, 2, 4, 6	00 – – 0
0, 2, 8, 10	0 – 0 – 0
2, 6, 10, 14	0 – – 1 0
2, 6, 18, 22	– 0 – 1 0
4, 5, 6, 7	0 0 1 – –
4, 5, 20, 21	– 0 1 0 –
4, 6, 20, 22	– 0 1 – 0

Step5: List all P.I.

P.I	Binary rep.
10, 11	0 1 0 1 –
17, 21	1 0 – 0 1
21, 29	1 – 1 0 1
29, 31	1 1 1 – 1
0, 2, 4, 6	0 0 – – 0
0, 2, 8, 10	0 – 0 – 0
2, 6, 10, 14	0 – – 1 0
2, 6, 18, 22	– 0 – 1 0
4, 5, 6, 7	0 0 1 – –
4, 5, 20, 21	– 0 1 0 –
4, 5, 20, 22	– 0 1 – 0

Step6: EPI Chart

PI / minterms	m_0	m_2	m_4	m_5	m_6	m_7	m_8	m_{10}	m_{14}	m_{17}	m_{18}	m_{21}	m_{29}	m_{31}	d_{11}	d_{20}	d_{22}
10, 11								⊙							⊙		
✓ 17, 21										⊙		⊙					
21, 29												⊙	⊙				
✓ 29, 31													⊙	⊙			
0, 2, 4, 6	⊙	⊙	⊙		⊙												
✓ 0, 2, 8, 10	⊙	⊙					⊙	⊙									
✓ 2, 6, 10, 14		⊙			⊙			⊙	⊙								
✓ 2, 6, 18, 22		⊙			⊙						⊙						⊙
✓ 4, 5, 6, 7			⊙	⊙	⊙	⊙											
4, 5, 20, 21			⊙	⊙								⊙			⊙		
4, 6, 20, 22			⊙		⊙									✓	⊙		⊙
						✓	✓		✓	✓							

$Y = A + B + C$

$Y = 17,21 + 29,31 + 0,2,8,10 + 2,6,10,14 +$
$2,6,18,22 + 4,5,6,7$

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

Problems on K-map

3-var:-

1. $F(x,y,z) = \Sigma m(2,3,4,5) = x \oplus y$
2. " $= \Sigma m(3,4,6,7) = yz + xz'$
3. " $= \Sigma m(0,2,4,5,6) = z' + xy'$
4. " $= \Sigma m(1,2,3,5,7) = c + A'B$
5. " $= \Sigma m(0,2,6,7) = xy + \bar{x}\bar{z}$
6. " $= \Sigma m(0,2,3,4,6)$
7. " $= \Sigma m(0,1,2,3,7) = \bar{a} + bc$
8. " $= \Sigma m(3,5,6,7) = xy + xz + yz$
9. $xy + x'y'z' + x'yz' = \Sigma m(0,2,6,7) = xy + \bar{x}\bar{z}$
10. $x'y' + yz + xyz' = \Sigma m(0,1,2,3,7) = \bar{x} + yz$

4-var:-

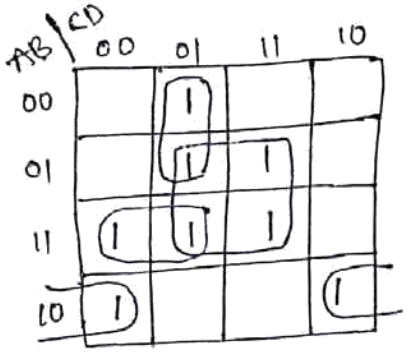
1. $F(w,x,y,z) = \Sigma m(0,1,2,4,5,6,8,9,12,13,14)$
   $= y' + w'z' + xz$
2. " $= \Sigma m(0,1,2,6,8,9,10) = \bar{B}\bar{D} + \bar{B}C + \bar{A}C\bar{D}$

Prime Implicant:

It is a smallest possible product term removing of any literal from which is not possible. The product must be an implicant of a given fn for it to be a prime implicant.

→ The bunch of 1s on the K-map which form a 2-square, 4-square, etc. is called PI or Subcube.

Ex: $f(A,B,C,D) = \Sigma m(1,5,7,8,10,12,13,15)$



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

$h_1 = A\bar{B}\bar{D}$

if A is removed $\bar{B}\bar{D} \rightarrow (0,1,8,10)$ is not an implicant

$\bar{B}$ $A\bar{D} \rightarrow (8,10,12,14)$ " " " .

if $\bar{D}$ is removed $A\bar{B} \rightarrow (8,9,10,13)$ " " " .

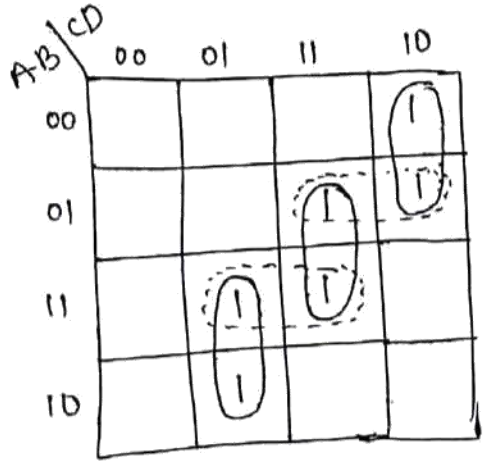
$\therefore h_1$ is a prime implicant

Essential prime Implicant:

It is the prime Implicant, It must cover at least one minterm which is not covered by any other P.I.

$EPI = \{\bar{A}\bar{C}D, BD, A\bar{B}\bar{D}, AB\bar{C}\}$

Ex: $f(A,B,C,D) = \Sigma m(2,6,7,9,13,15)$



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

No. of PI = 5 → $\bar{A}C\bar{D}$ (2,6), BCD (7,15), $A\bar{C}D$ (9,13), ABD (13,15), $\bar{A}BC$ (6,7)

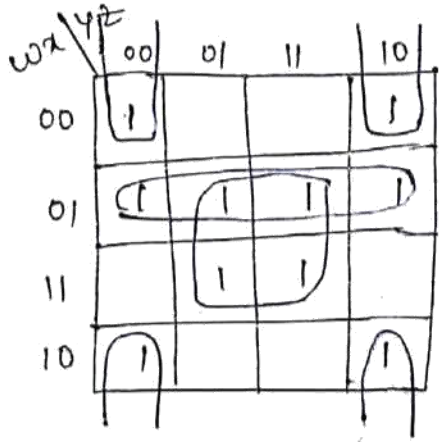
No. of EPI = 2 ⇒ (2,6), (9,13)

ABD
BD

0101 – 5
0111 – 7
1101 – 13
1111 – 15

find the prime Implicant for the following and determine which are essential.

1) $f(w,x,y,z) = \Sigma m(0,2,4,5,6,7,8,10,13,15)$

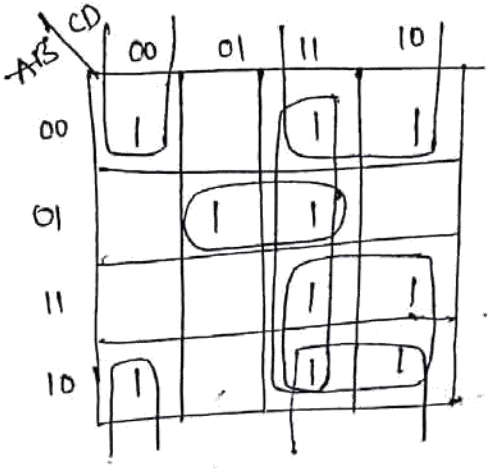


$= \bar{w}x + xz + \bar{x}\bar{z} + \bar{w}\bar{z}$

↳ prime Implicants

$\bar{x}\bar{z}$ and xz are not covered by any other cells, so those are called as the EPI.

$f(A,B,C,D) = \Sigma m(0,2,3,5,7,8,10,11,14,15)$

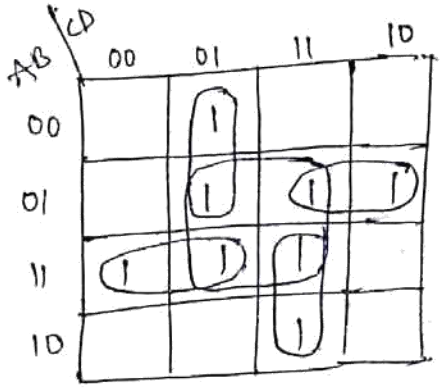


$= CD + \bar{B}\bar{D} + AC + \bar{A}BD + \bar{B}C$

↳ P.I.

EPI $= \bar{B}\bar{D} + \bar{A}BD + AC$

3] $f(A,B,C,D) = \Sigma m(1,5,6,7,11,12,13,15)$



$BD + \bar{A}\bar{C}D + AB\bar{C} + \bar{A}BC + ACD$

No. of PI = 5

$(1,5) \rightarrow \bar{A}\bar{C}D$

$(11,15) \rightarrow ACD$

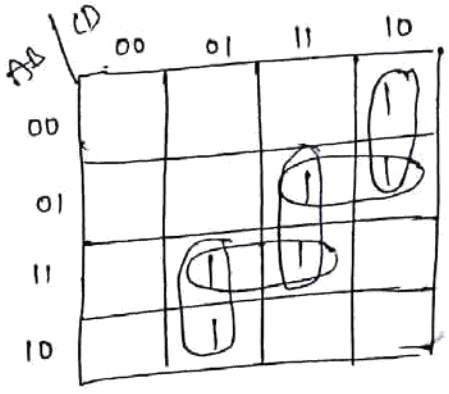
$(6,7) \rightarrow \bar{A}BC$

$(12,13) \rightarrow AB\bar{C}$

$(5,7,13,15) \rightarrow BD$

EPI $= \bar{A}\bar{C}D + ACD + \bar{A}BC + AB\bar{C}$ $\{=4\}$.

4]. $f(A,B,C,D) = \Sigma m(2,6,7,9,13,15)$



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

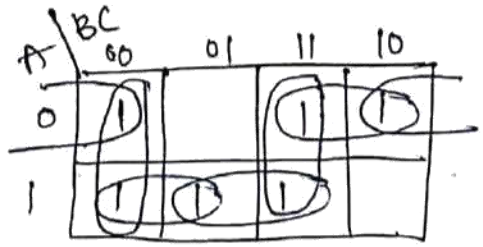
no. of PI = 5 $\Rightarrow$ $\overset{\bar{A}C\bar{D}}{(2,6)}$ $\overset{BCD}{(7,15)}$ $\overset{A\bar{C}D}{(9,13)}$

$\overset{ABD}{(13,15)}$ $\overset{\bar{A}BC}{[6,7)}$

no. of EPI = 2 $\Rightarrow$ (2,6), (9,13)

$\bar{A}C\bar{D} + A\bar{C}D$.

5] $f(A,B,C) = \Sigma m(0,2,3,4,5,7)$



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

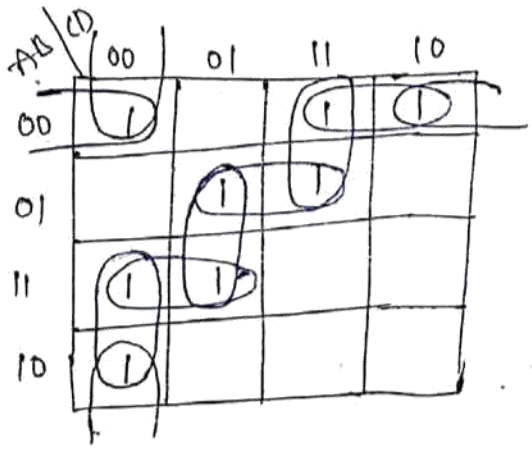
↳ P.I

no. of PI = 6 ⇒ (0,2), (2,3), (3,7), (5,7), (4,5), (0,4)

$f_3 = \bar{A}\bar{C} + A\bar{B} + BC$ (or) $\bar{B}\bar{C} + AC + \bar{A}B$

no. of EPI = 0

6]. $f(A,B,C,D) = \Sigma m(0,2,3,5,7,8,12,13)$



$f_4 = \bar{A}\bar{B}\bar{D} + \bar{A}CD + B\bar{C}D + A\bar{C}\bar{D}$

no. of PI = 8

No. of EPI = 0.

UNIT NO III

COMBINATIONAL CIRCUITS

Combinational ckt :—

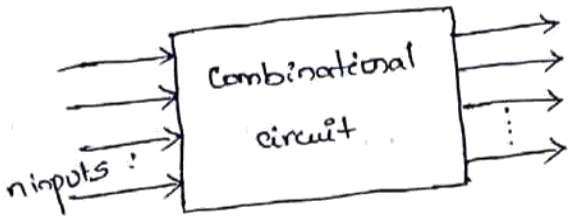
logic ckts for digital systems may be combinational or Sequential.

— A combinational ckt consists of logic gates whose o/p's at any time are determined from the present combination of inputs.

It performs an operation that can be specified logically by a set of Boolean functions.

— It consists of input variables, logic gates and o/p variables.

The logic gate accept signals from the inputs and generate signals to the outputs.



— The n I/p binary variables come from an External source; The m output variables go to an External destination.

— for n input variables, the are 2^n possible o/p values for each combinations. for each possible input combination, there is one possible output value. Thus a Combinational ckt can be specified with a truth table that lists the o/p values for each combination of input variables.

Analysis procedure :—

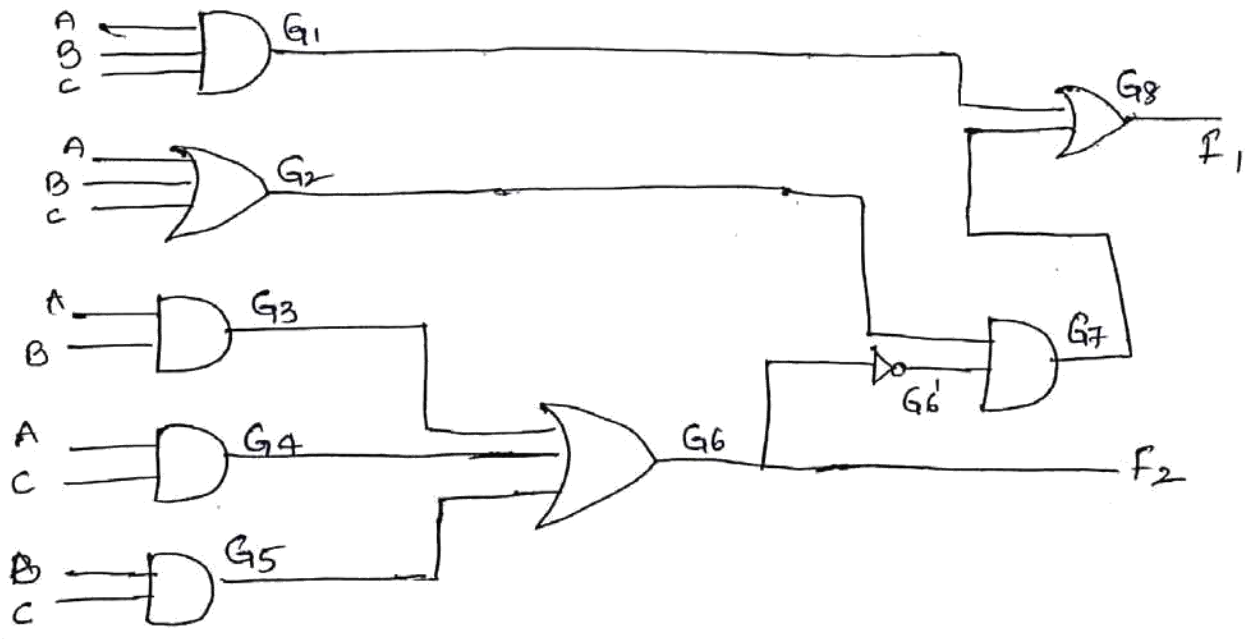
— The first step in the analysis is to make sure that the is combinational and not a Sequential.

— The diagram of a combinational ckt has logic gates with paths or memory Elements. A feedback path is a connec from the o/p of one gate to the input of a second ga forms part of the input to the first stage.

— Once the logic diagram is verified as a combinational ckt one can proceed to obtain o/p Boolean function or truth table.

— To obtain the o/p boolean function from a logic diagram, proceed as follows.

1. Label all gate o/ps that are a function of i/p variables with arbitary symbols. Determine the boolean functions for Each gate o/p.
2. label the gates that are a fn of input variables and previously labeled gates with other arbitary symbols.
3. Repeat the process of step2 until the o/ps of the ckt are obtained.
4. By repeated substitution of previously defined functions, obtain the o/p boolean functions in terms of input variables.



$G_1 \rightarrow ABC \quad G_2 \rightarrow A+B+C \quad G_3 = AB \quad G_4 \rightarrow AC \quad G_5 \rightarrow BC$

$G_6 \rightarrow AB+AC+BC = F_2$

$G(6)' \rightarrow (AB+AC+BC)' = \overline{AB} \cdot \overline{AC} \cdot \overline{BC}$

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

$G_7 \rightarrow (A+B+C)(\bar{A}+\bar{B})(\bar{A}+\bar{C})(\bar{B}+\bar{C})$

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

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

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

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

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

$G_8 \rightarrow \bar{A}B\bar{C} + \bar{A}\bar{B}C + A\bar{B}\bar{C} + ABC$

— To obtain the truth table directly from logic diagram through the derivation of boolean functions.

1. Determine the no. of i/p variables in ckt. for n-inputs, there are 2^n possible i/p combination and list the binary numbers from 0 to $2^n - 1$.
2. Label the o/ps of selected gates with the arbitary symbols.
3. obtain the truth table for o/ps of those gates that are a function of i/p variables only.
4. proceed to obtain the T.T for o/ps of those gates that are fn of previously defined values until columns for all o/ps are determined.

A	B	C	G_1	G_2	G_3	G_4	G_5	G_6	G_6	G_7	G_8
0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	0	1	0	0	0	0	1	1	1
0	1	0	0	1	0	0	0	0	1	1	1
0	1	1	0	1	0	0	1	1	0	0	0
1	0	0	0	1	0	0	0	0	1	1	1
1	0	1	0	1	0	0	0	1	0	0	0
1	1	0	0	1	1	0	0	1	0	0	0
1	1	1	1	1	1	1	1	1	0	0	1

Design procedure:

The design procedure starts from the specification of the problem and ends up with a logic diagram or a set of boolean functions from which the logic diagram can be obtained.

1. From the specifications of the ckt, determine the required no. of inputs and o/p's and assign a symbol to each.
2. Derive the truth table that defines the required relationship b/w inputs and outputs.
3. obtain the simplified boolean functions for each o/p as a fn of i/p variables.
4. Draw the logic diagram and verify the correctness of the design.

- A truth table for a combinational ckt consists of input columns and o/p columns.
- Input columns are obtained from the 2^n binary numbers for the n input variables. The binary values for the o/p's are

termined from the Stated Specifications.

- o/p functions Specified in the truth table give the Exact definition of Combinational Ckt.
- The o/p binary functions listed in the truth table are simplified by any available method Such as algebric manipulation, the map method.
- The o/p binary functions listed in the truth table are simplified by any available method Such as algebric manipulation, the map method.

Code Conversion Example

- code converter is a ckt that makes the two systems Even though Each uses a different binary code.

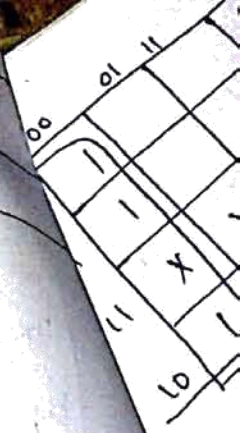
Ex: BCD to Excess-3 code

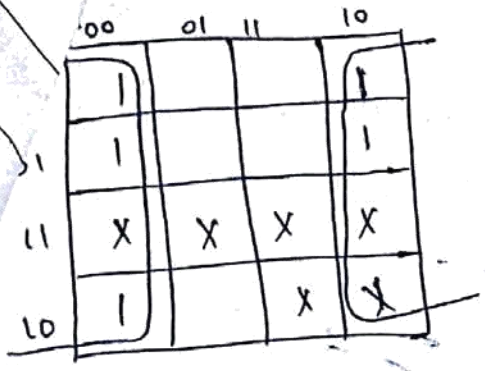
Input is BCD and the o/p is Excess-3 code. The i/p BCD has numbers Starting from 0 to 9 which uses 4-bits to represent. The Excess-3 code for 0 is 3, 1 is 4, ... 9 is 12 which is also represented by 4 bits. So, therefore, we need 4 input as well as 4 output Variables.

- Designate four i/p binary Variables by A, B, C, D and four o/p by w, x, y & z.

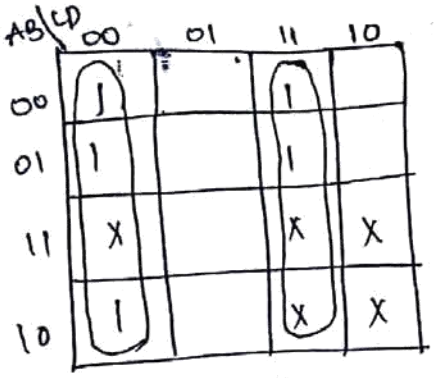
Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0				
1	0	1	1				
1	1	0	0				
1	1	0	1				
1	1	1	0				
1	1	1	1				

These are don't Care Conditions for BCD.

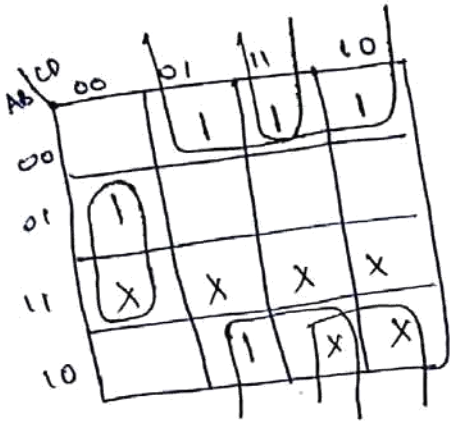




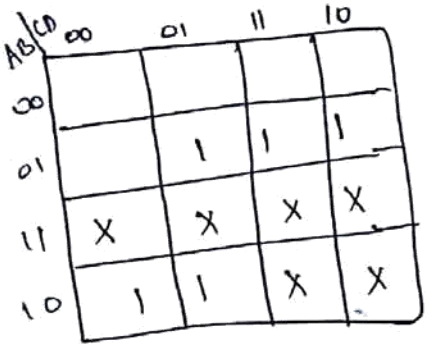
$Z = \bar{D}$



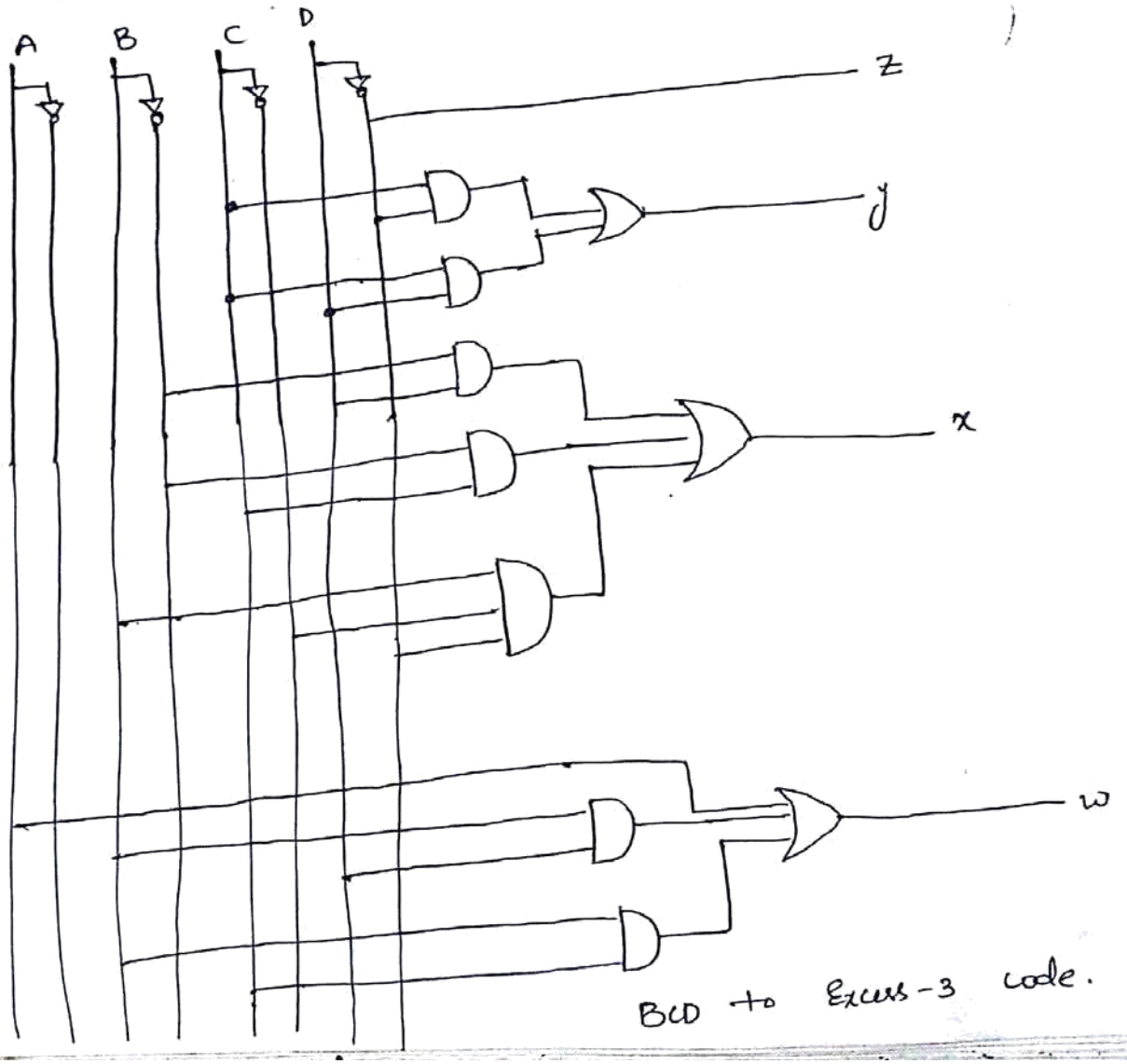
$y = \bar{C}\bar{D} + CD$



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



$W = A + BD + BC$



BCD to Excess-3 code.

BINARY ADDER - SUBTRACTOR

— Digital Computers perform a variety of information processing. The most basic arithmetic operation is the addition of two binary digits.

— When both augend & addend bits are Equal to 1, the binary Sum Consists of two digits. The higher Significant bit of this result is called a Carry.

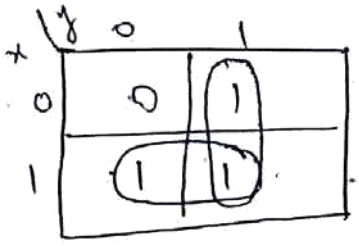
— A Combinational ckt that performs the addition of two bits is called a half adder.

One performs the addition of three bits is a full adder.

Half adder

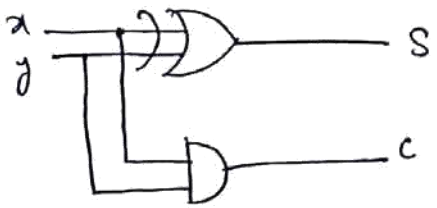
The ckts needs two binary i/p's & two binary o/p. Let the i/p's are x & y and o/p are S & C.

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



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

$C = xy.$

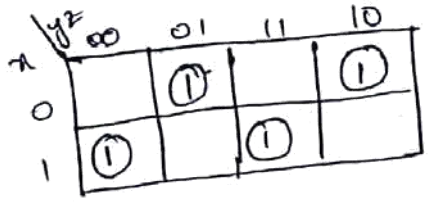


Full adder:

Full adder is a combinational ckt performs the sum of three bits. It consists of three inputs and two o/p's. Two of i/p's are x and y, represents two significant bits to be added. The third i/p, z represents the carry from previous LSB. The two o/p's are S & C.

- The S o/p is Equal to 1 when only one i/p is Equal to 1 or When all three i/p's are Equal to 1. The C o/p has carry to 1 if two or three i/p's are Equal to 1.

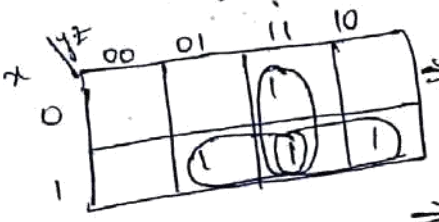
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



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

$= x \oplus y \oplus z$.

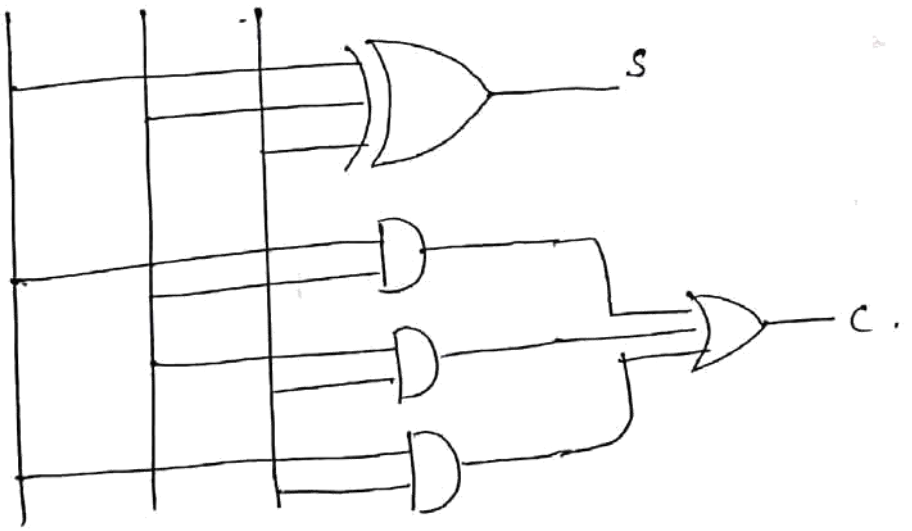
$x'(yz + yz')$
$+x(y'z' + yz)$



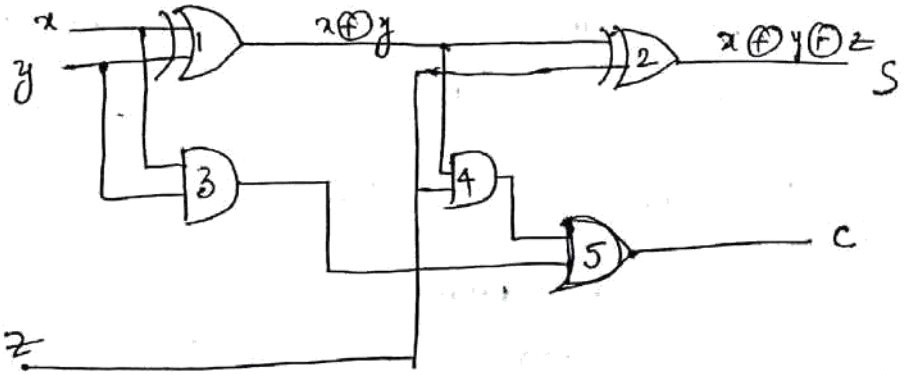
$\Rightarrow x'(y \oplus z)$
$+x(y \odot z)$

$C = yz + xy + xz$ $\Rightarrow x'(y \oplus z) + x(\overline{y \oplus z})$

$= x \oplus y \oplus z$



Implementation of F.A with two H.A and an OR Gate:



1) $x \oplus y$ 2) $x \oplus y \oplus z$ 3) xy 4) $(x \oplus y)(z) = (x'y + xy')z$

$= x'yz + xy'z$

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

$x'yz + xy'z + xy$

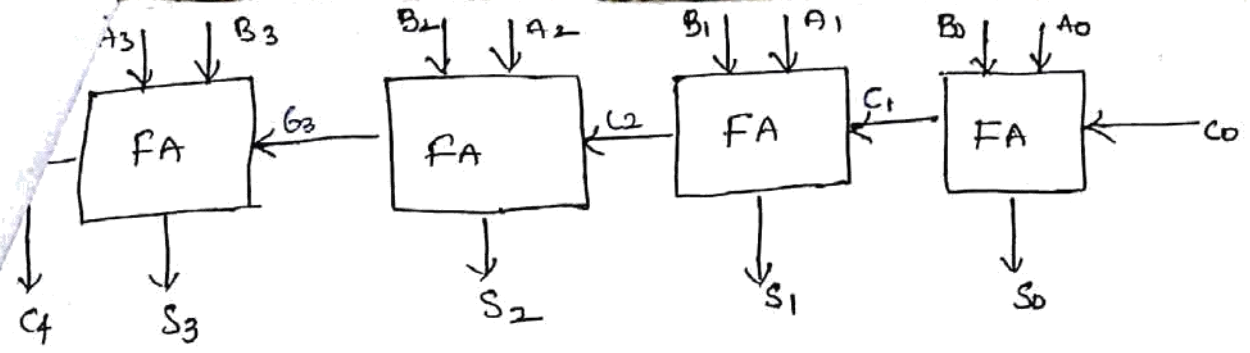
$\sum m(3, 5, 7, 6)$

Binary Adder:- (parallel Binary Adder)

A binary adder is a digital ckt that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascade, with the o/p carry from each F.A connected to the i/p carry to the next FA.

Let two binary numbers A = 1011₂ B = 0011

Subscript i:	3	2	1	0	
I/p carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
O/p Carry	0	0	1	1	C_{i+1}



The bits are added with F.A, Starting from least Significant-position, to form the sum & carry. The i/p carry Co in the LSB must be 0. The value of C_{i+1} in a given Significant-position is the o/p carry of f.A. This is transferred into the i/p carry of f.A that adds the bits one higher Significant position to left.

Extra

Carry propagation

- The add of two binary numbers in parallel implies that all the bits of the augend and addend are available for computation at the same time. In any comb. ckt, the s/g must propagate through the gates before the correct o/p sum is available in o/p terminals.

- The total propagation time is Equal to the propagation delay of typical gate times and the no. of gate levels in the ckt.

- Inputs A_3 & B_3 are available as soon as i/p s/gs are applied. However, C_3 doesnot settle to its final value until C_2 is available from the previous stage.

The no. of gate levels for the carry propagation can be found from the ckt of the F.A. The i/p & o/p variables use the Subscript i to denote a typical Stage in the adder.

The Signals at p_i & G_i Settle to their steady state after they propagate through their respective gates.

— A comb. ckt will always have some value at its o/p terminals o/p's will not be correct unless the signals are given enough time to propagate through the gates connected from i/p's to o/p's.

As, the arithmetic operations are implemented by successive additions time consumed during the addition process is very critical.

for reducing the carry propagation delay time, employ faster gates with reduced delays.

Consider the F.A designed with two H.A Define two new binary Variables.

$P_i = A_i \oplus B_i$ & $G_i = A_i B_i$ G_i - Carry generate

$S_i = P_i \oplus C_i$, $C_{i+1} = G_i + P_i C_i$ P_i - Carry propagate.

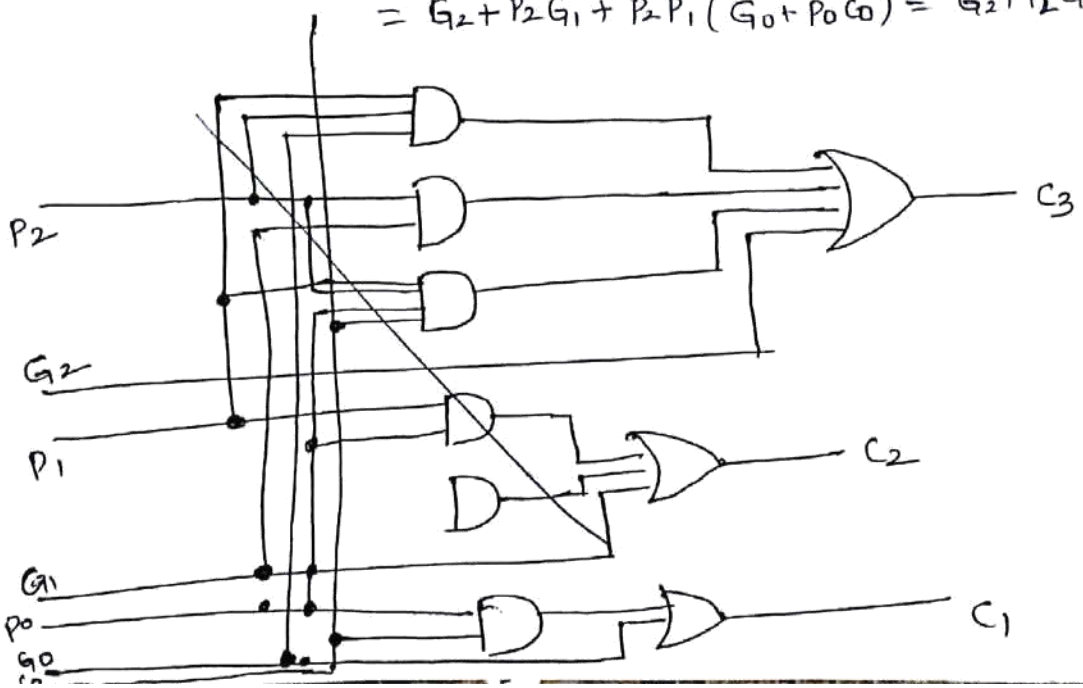
C_0 = i/p Carry.

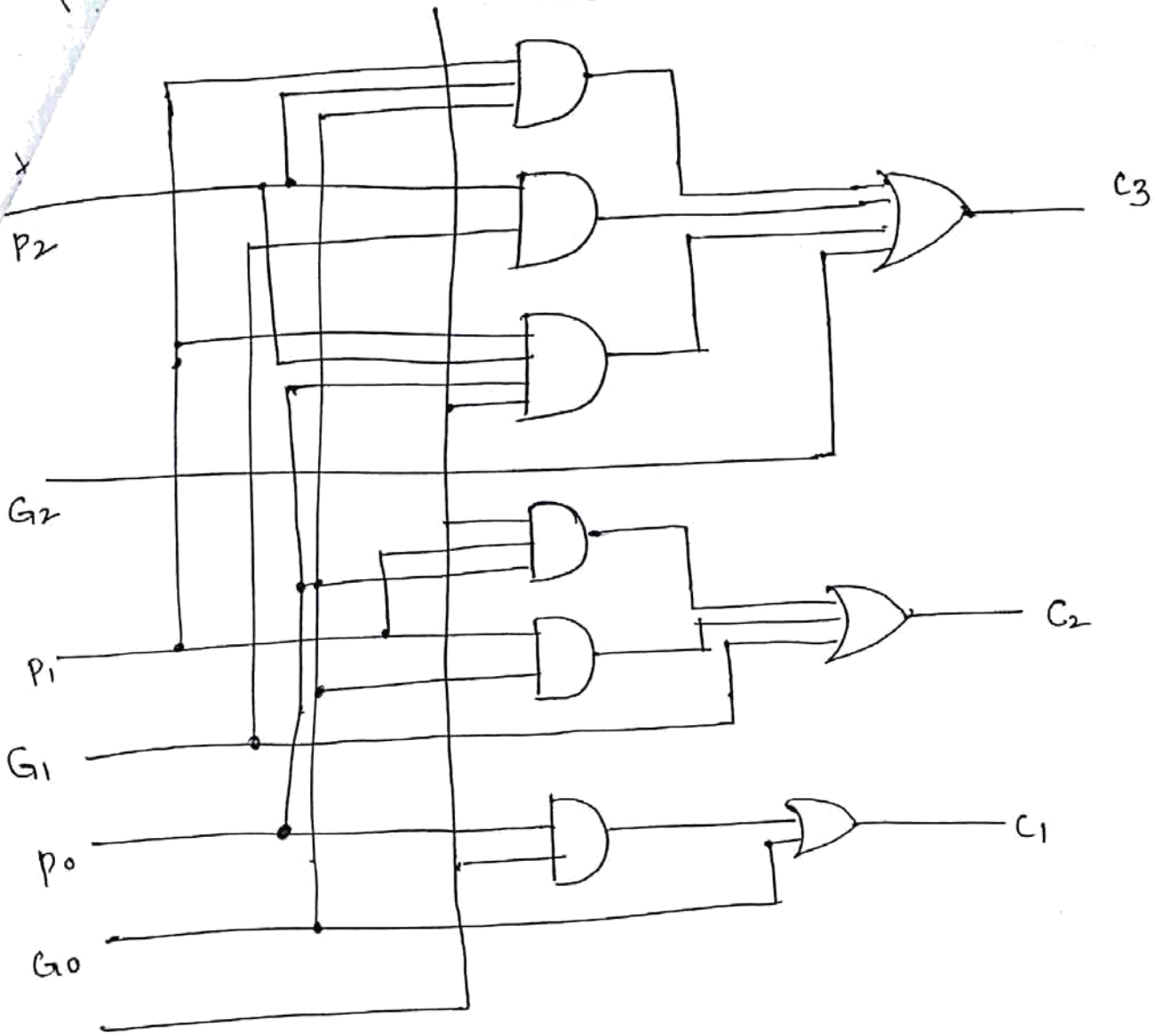
$C_1 = G_0 + P_0 C_0$

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

$C_3 = G_2 + P_2 C_2 = G_2 + P_2 (G_1 + P_1 C_1) = G_2 + P_2 G_1 + P_2 P_1 C_1$

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

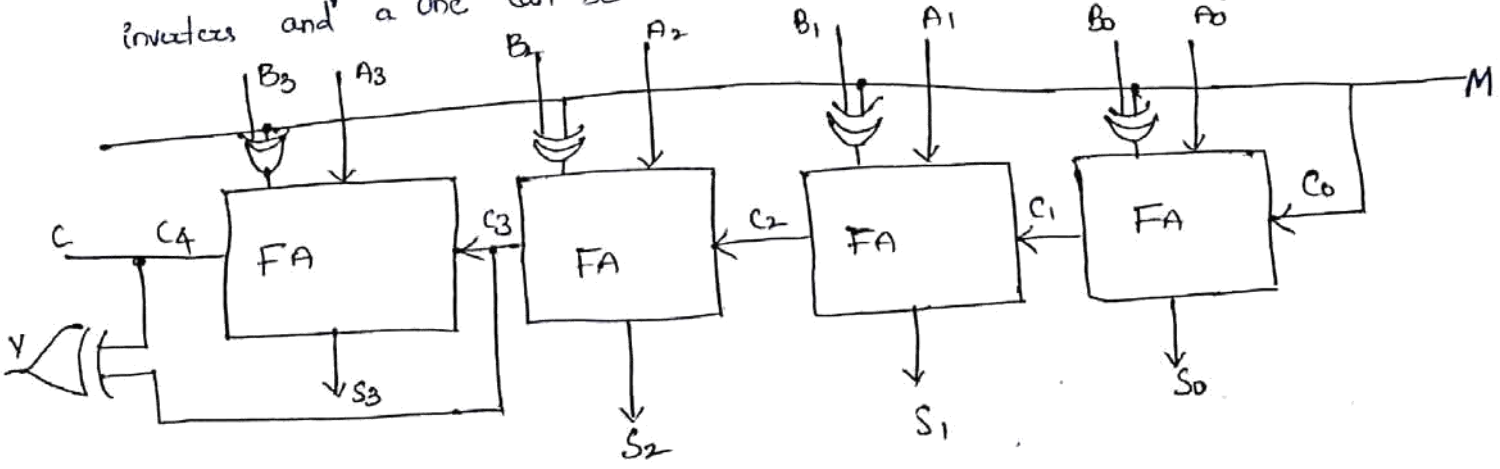




Extra Adder/

Binary Subtractor: The Subtraction of unsigned binary numbers can be done by means of Complements.

A-B Can be done by taking 2's complement of B and adding it to A. The 2's Complement Can be obtained by taking the 1's comp and adding one to LSB. The 1's Complement Can be obtained with inverters and a one Can be added to the Sum through the i/p Carry.



— The mode i/p m controls the operation.

When m=0, the ckt is an adder, because when m=0, B⊕

The full adder receive the value of B, the i/p Carry is 0,

the ckt performs A plus B.

When m=1, B⊕1 = B' and Co=1. The B inputs are all compl

nted and a 1 is added through the i/p Carry. The ckt performs the

operation A plus the 2's Complement of B.

Decimal Adder:

Computers or Calculators that perform arithmatic operation in decimal number System represent decimal numbers in binary coded form

— A dec. adder requires a min of nine i/p's and five since four bits are required to code Each decimal digit and the ckt must have an i/p & o/p Carry.

BCD Adder:

Consider the arithmatic addition of two dec digits in BCD, togeth with an i/p Carry from a previous stage.

If we apply two BCD digits to a 4-bit binary adder. It will form Sum in binary and produce a result that ranges from 0 to 19.

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	1	6
0	0	1	1	1	0	1	0	0	0	7

Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	Decimal
1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	9
0	1	0	1	0	0	1	0	1	10
0	1	0	1	1	0	1	1	0	11
0	1	1	0	0	0	1	1	1	12
0	1	1	0	1	0	1	1	1	13
0	1	1	1	0	0	1	1	1	14
0	1	1	1	1	0	1	1	1	15
1	0	0	0	0	1	0	1	0	16
1	0	0	0	1	1	0	1	0	17
1	0	0	1	0	1	1	0	0	18
1	0	0	1	1	1	1	0	0	19

- The columns under the binary Sum list the binary value that appears in the o/p's of the 4-bit binary adder. The o/p of two decimal digits must be represents in BCD and should appear in the form listed in the columns under BCD Sum.

From the table, it is apparent that when the binary Sum is Equal to or less than 1001, the corresponding BCD number is identical, and therefore no Conversion is needed.

When the binary Sum is >1001, we obtain a non-valid BCD. The addition of 6 (0110) to the binary Sum converts it to the Correct BCD representation and produces an o/p Carry.

The Correction is needed when binary Sum has an o/p Carrey

K = 1

The other Six combinations from 1010 through 1111 that need have a 1 in position Z_8. To distinguish them from binary 10 which also have a 1 in Z_8, we Specify further either Z_4 or must have a 1. The condition for a correction & an o/p Carry can Expressed by the Boolean fn.

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

When C=1, it is neccessary to add 0110 to the binary Sum & provide an o/p Carry for the next stage.

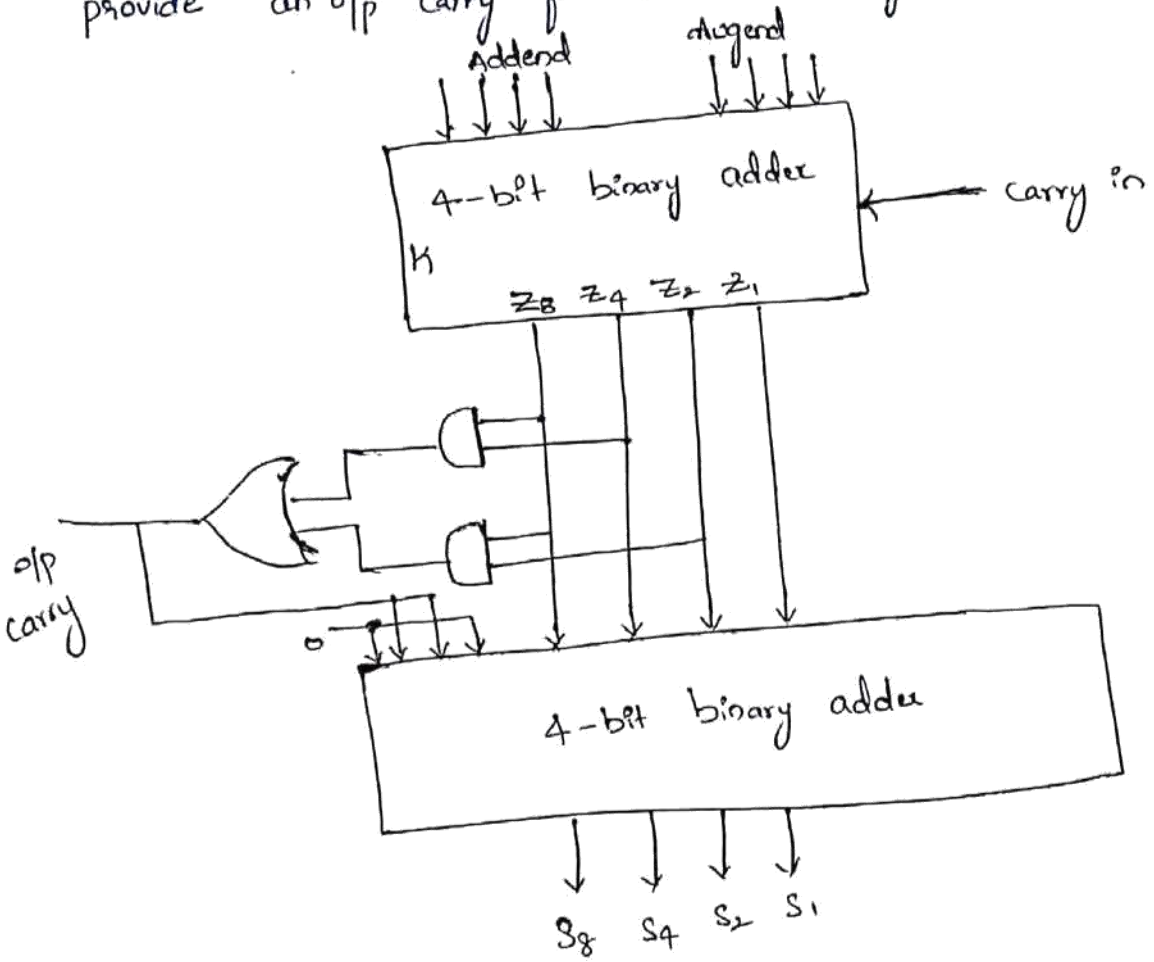


Fig: BCD adder.

Extra

multiplier

The multiplicand is multiplied by each bit of the multiplier starting from LSB. Each such multiplication forms a partial product. Successive partial products are shifted one position to the left. The final product is obtained from the sum of the partial products

Consider the multiplication of two 2-bit numbers. The multiplicand bits are B_1 and B_0, multiplier bits are A_1 and A_0 and the product is

$C_3\ C_2\ C_1\ C_0$

$B_1\ B_0$

$A_1\ A_0$

$A_0B_1\ \ A_0B_0$

$A_1B_1\ \ A_1B_0$

$C_3\ \ C_2\ \ C_1\ \ C_0$

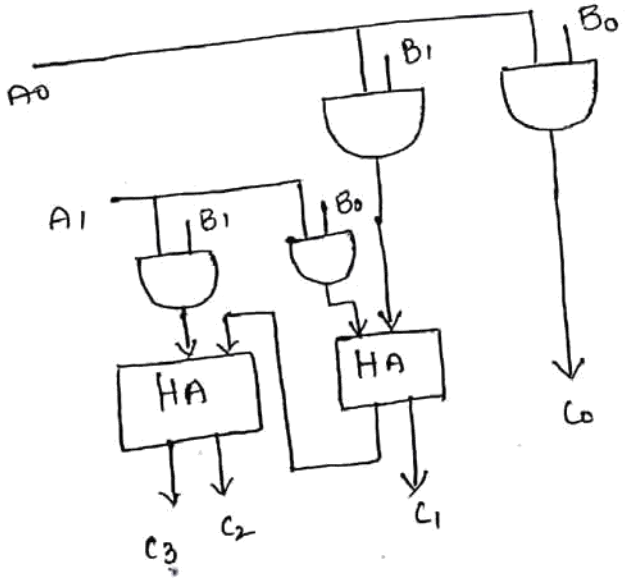


fig: 2-bit Binary multiplier.

better understanding of boolean laws, rules and theorems.

Magnitude Comparator :

A magnitude Comparator is a comb ckt that compares two A and determines their relative magnitudes. The outcome of the comparision specified by three binary variables that indicate whether $A > B$, $A = 0$ or

let the two numbers, A and B, with four digits Each.

$$A = A_3 A_2 A_1 A_0$$
$$B = B_3 B_2 B_1 B_0$$

The two numbers are Equal if all pairs of significant digits are Equal. $A_3 = B_3$ and $A_2 = B_2$ and $A_1 = B_1$ and $A_0 = B_0$. When the numbers are binary, the digits are either 1 or 0, and the Equality relation of Each pair of bits Can be Expressed logically with an Ex-NOR fn as

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

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

The Equality of two numbers, A and B is displayed in a comb ckt by an o/p binary variable that we designate by Symbol $(A=B)$. for the Equality condition to Exist, all x_i variables must be=1 This dictates an AND operation of all variables..

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

- To determine if $A >$ or $< B$, the relative mag of pairs of Significant digits starting from MSB. If two digits are Equal Compare the next lower Significant pair of digits. This comparision Continues until a pair of unEqual digits is Reached.

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

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

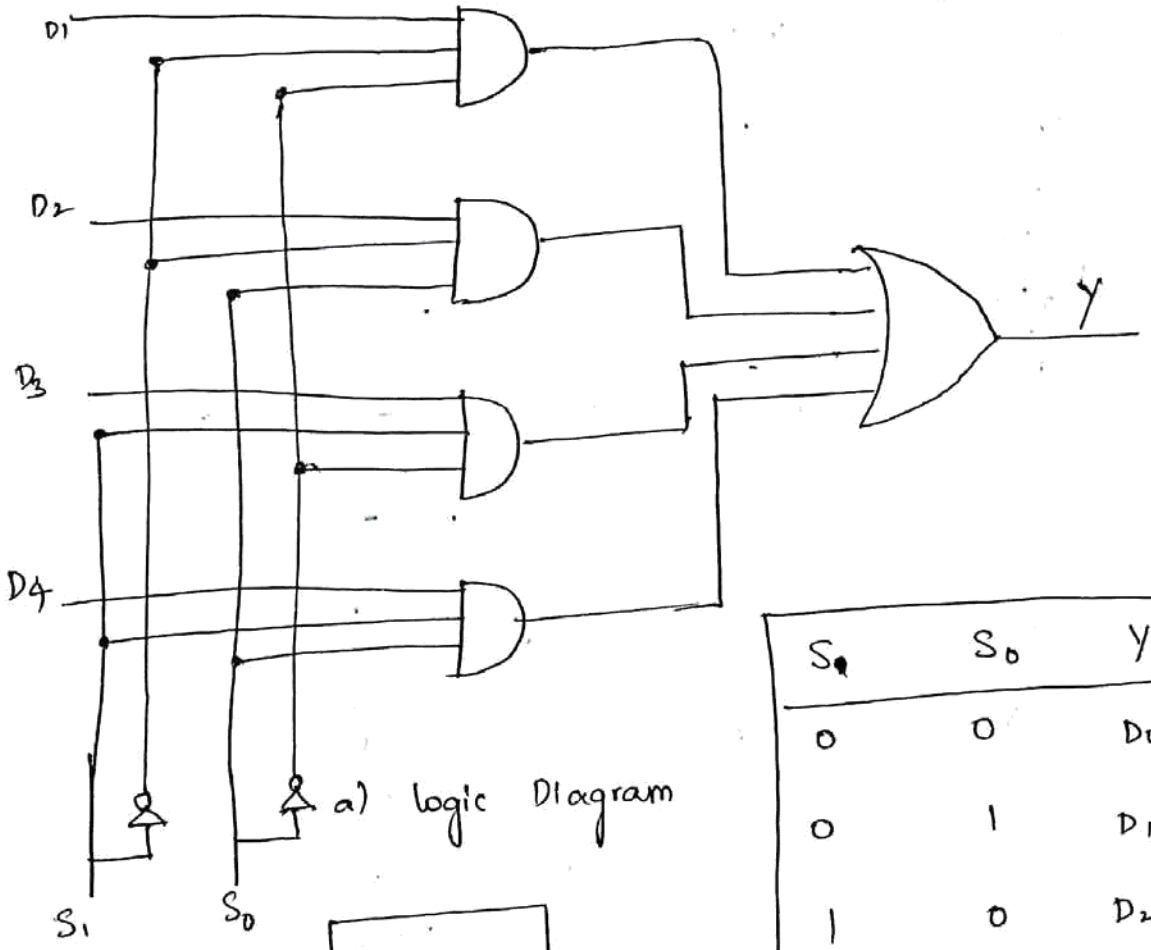
Multiplexers:

It is a digital switch, which allows digital information from several sources to be routed onto a single output line.

- It consists of several data-input lines and a single o/p line. The selection of a particular input line is controlled by a set of selection lines.
- Normally, there are 2^n input lines and n selection lines whose bits combinations determine which input is selected.

4 to 1 line multiplexer:

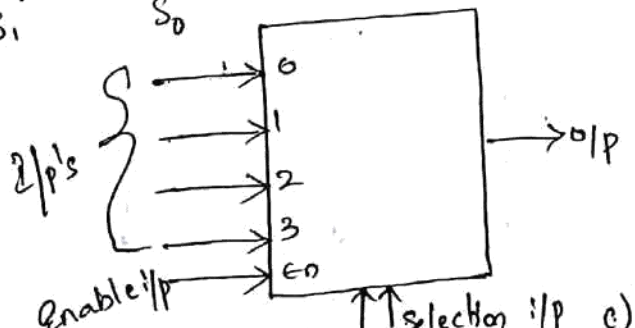
- It consists of four lines, D_1 to D_3, is applied to one input of an AND gate. Selection lines are decoded to select a particular AND gate.



a) logic Diagram

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

b) functional table

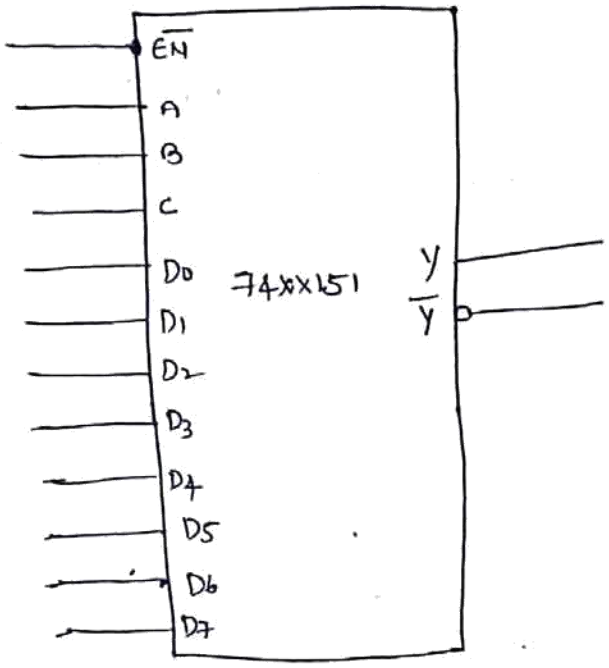


c) logic Symbol

74X151 8 to 1 multiplexer

— It is a 8 to 1 multiplexer. It has eight inputs. It provides two o/p's, one is active high, the other is active low. There are Select lines C, B, A which Selects one of the eight inputs.

In this o/p is not specified in 1s and 0s. Because, multiplexer is a data switch, it does not generate only data of its own, but it simply passes External data, input to the o/p.

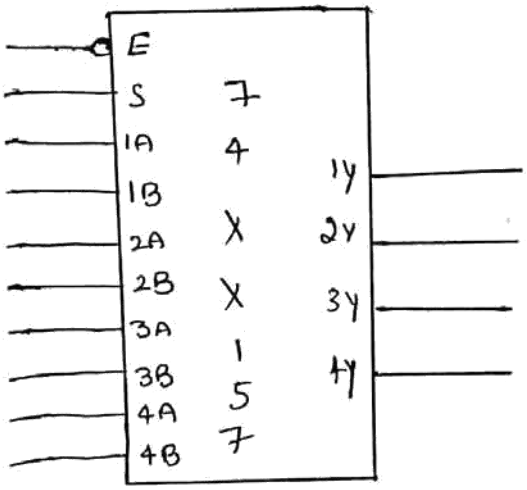


Inputs				outputs	
Select			Enable	y	ȳ
C	B	A	$\overline{EN}$	y	ȳ
X	X	X	1	0	1
0	0	0	0	D_0	$\overline{D_0}$
0	0	1	0	D_1	$\overline{D_1}$
0	1	0	0	D_2	$\overline{D_2}$
0	1	1	0	D_3	$\overline{D_3}$
1	0	0	0	D_4	$\overline{D_4}$
1	0	1	0	D_5	$\overline{D_5}$
1	1	0	0	D_6	$\overline{D_6}$
1	1	1	0	D_7	$\overline{D_7}$

74XX157 Quad 2-input multiplexer:-

It Selects four bits of data from two Sources under the control a Common Select input (S)

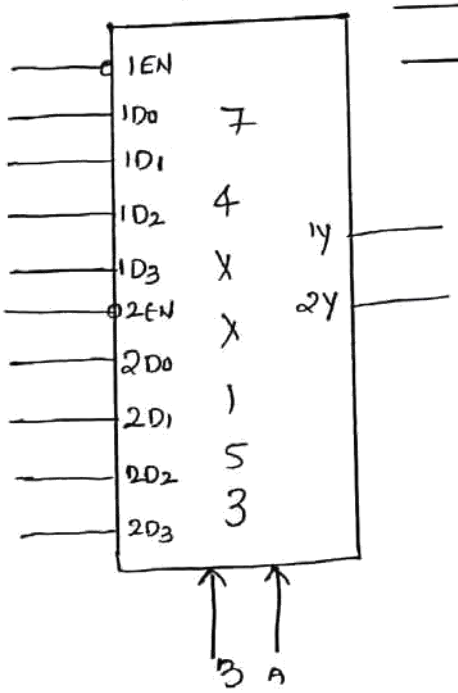
— The Enable input ($\bar{E}$) is active low. when $\bar{E}$ is high, all of the o/p's (Y) are forced low Regardless of all other input conditions.



Inputs		Outputs			
$\bar{E}$	S	1y	2y	3y	4y
1	X	0	0	0	0
0	0	1A	2A	3A	4A
0	1	1B	2B	3B	4B

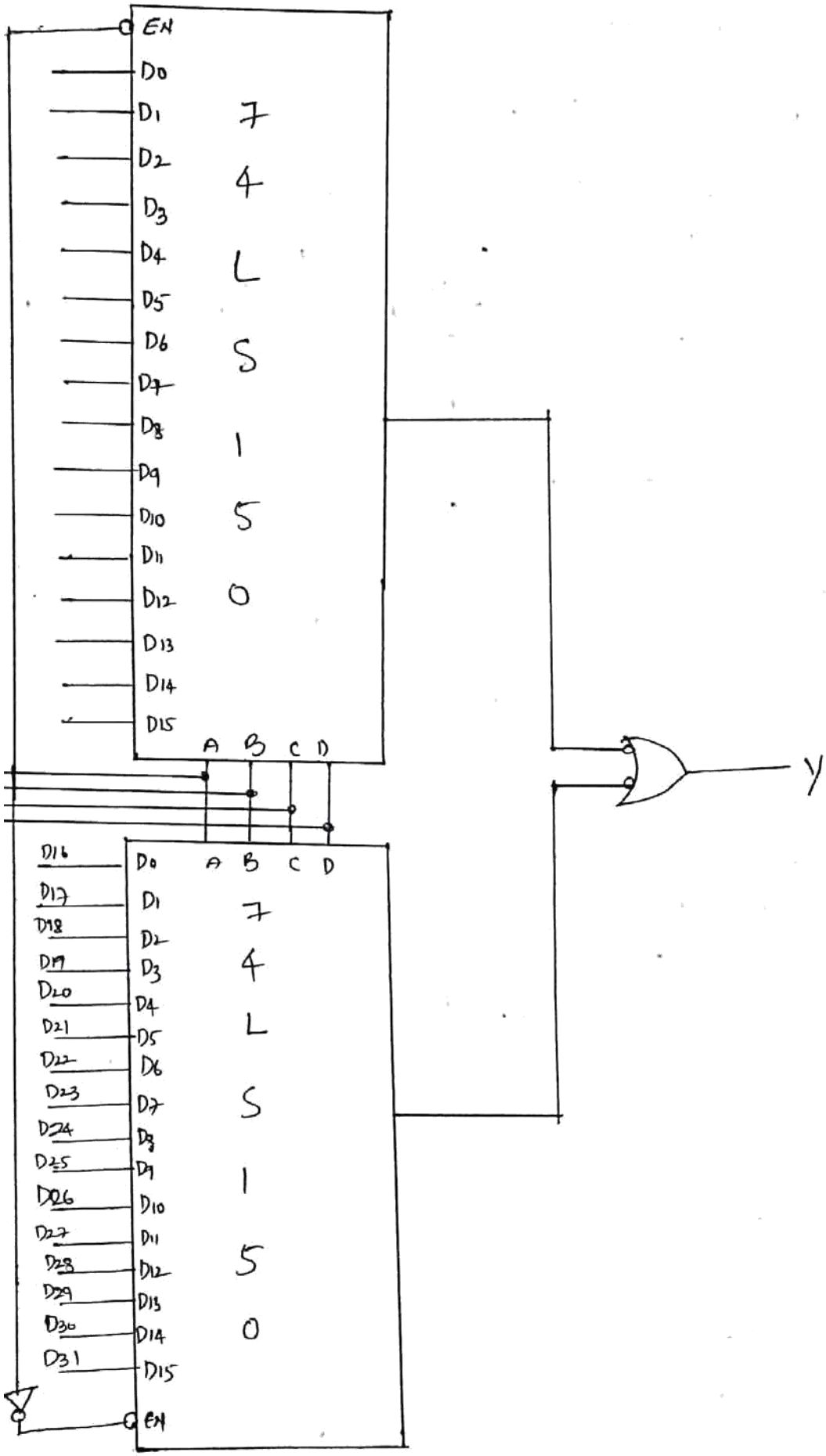
74XX153 Dual 4 to 1 multiplexer:—

— It contains two identical and identical independent 4 to 1 multiplexers. Each multiplexer has separate Enable input.



Inputs				Outputs	
1EN	2EN	B	A	1y	2y
0	0	0	0	1D0	2D0
0	0	0	1	1D1	2D1
0	0	1	0	1D2	2D2
0	0	1	1	1D3	2D3
0	1	0	0	1D0	0
0	1	0	1	1D1	0
0	1	1	0	1D2	0
0	1	1	1	1D3	0
1	0	0	0	0	2D0
1	0	0	1	0	2D1
1	0	1	0	0	2D2
1	0	1	1	0	2D3

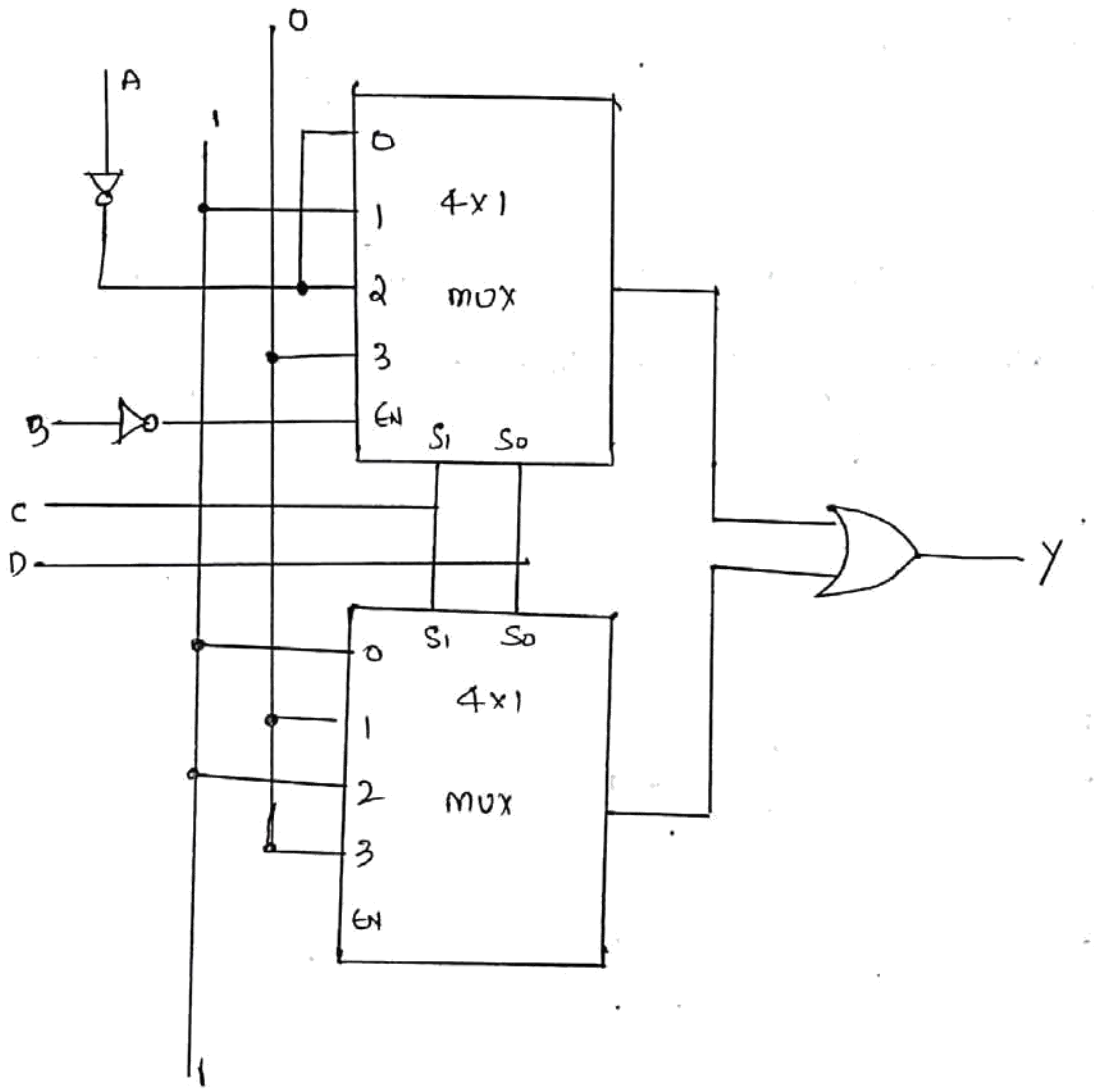
2 to 1 multiplexer using two 74LS150



$F(A, B, C, D) = \sum m\ (0, 1, 2, 4, 6, 9, 12, 14)$

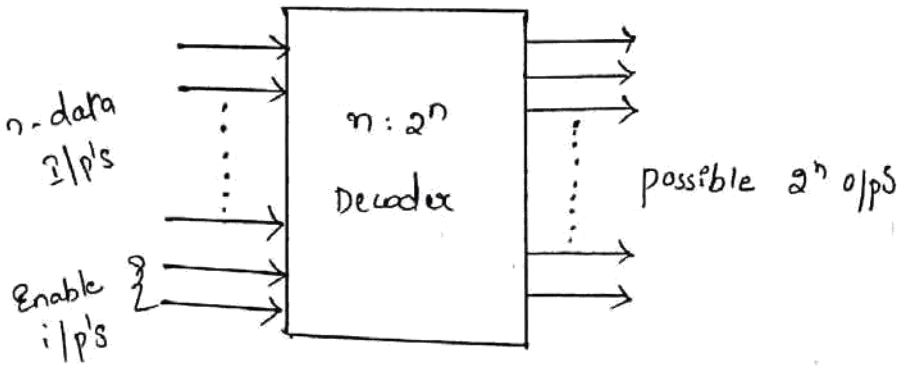
	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	(0)	(1)	(2)	3	(4)	5	(6)	7
A	8	(9)	10	11	(12)	13	(14)	15

$\bar{A}$ 1 $\bar{A}$ 0 1 0 1 0



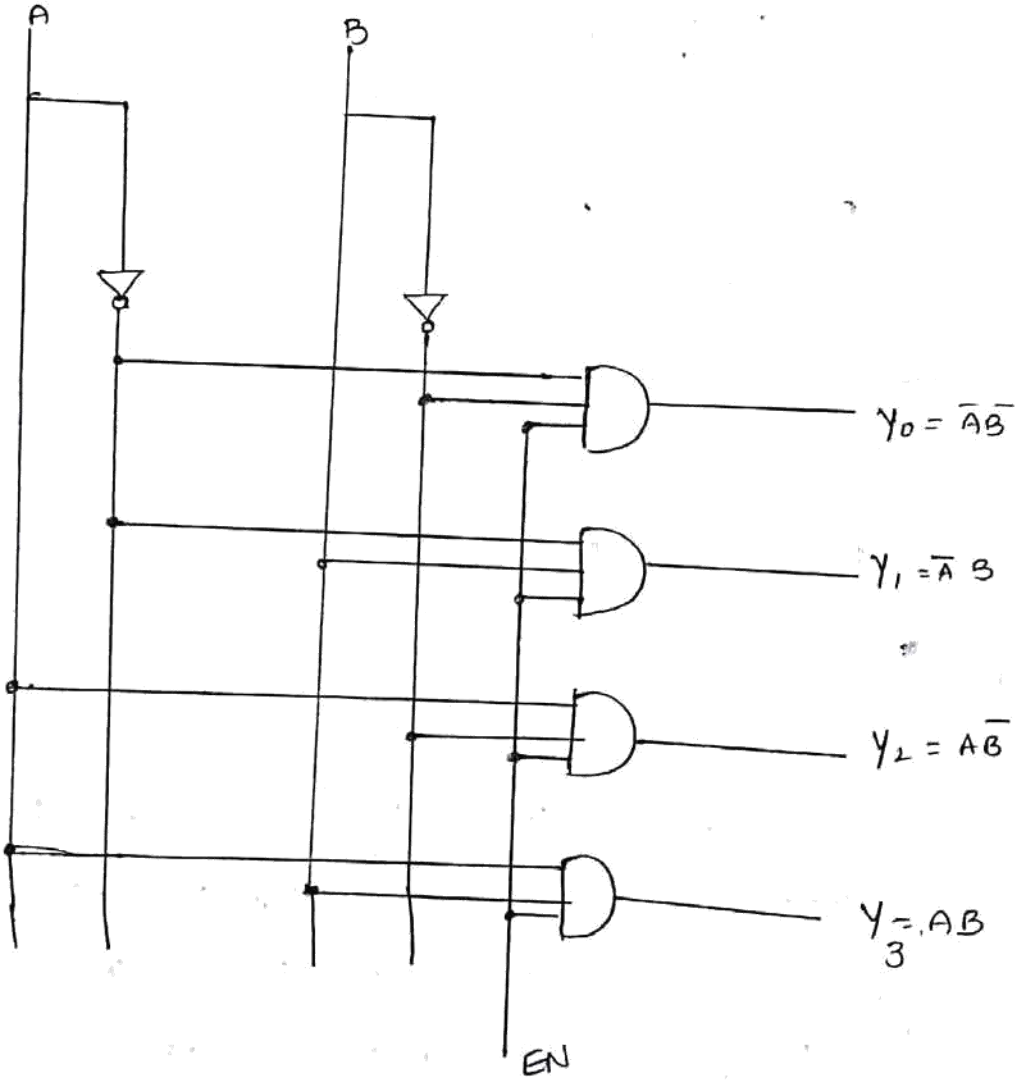
Decoder :-

- Decoder is a multiple-input, multiple o/p logic ckts which converts coded inputs into coded o/p's, where the input and o/p codes are different
- Input code generally has fewer bits than the o/p code.
- The Encoded information is presented as n input producing 2^n possible outputs. The 2^n o/p values are from 0 through $2^n - 1$



Binary Decoder:

— A decoder which has an n-bit binary input code and a one activated o/p out of 2^n o/p code is called binary Decoder.



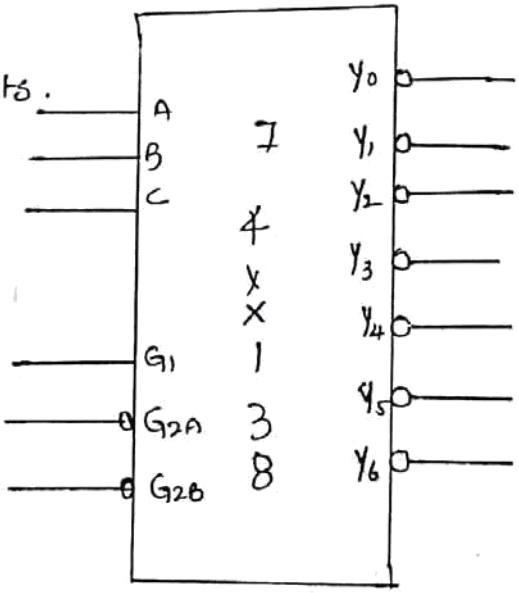
2-4, Decoder.

the two inputs are decoded into four o/p's, each o/p representing one of the minterms of the 2 input variables.

Inputs			outputs			
EN	A	B	Y_3	Y_2	Y_1	Y_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

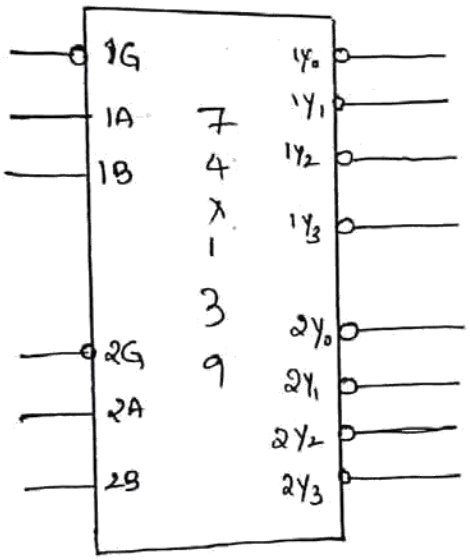
74X138 3 to 8 Decoder:—

- 74X138 is a commercially available 3 to 8 decoder.
- It accepts 3 binary inputs (C, B, A) and when Enabled provides eight individual active low o/p's (Y_0 - Y_7)
- The device has three Enable inputs. two active low ($\overline{G_{2A}}$, $\overline{G_{2B}}$) and one active high (G_1)



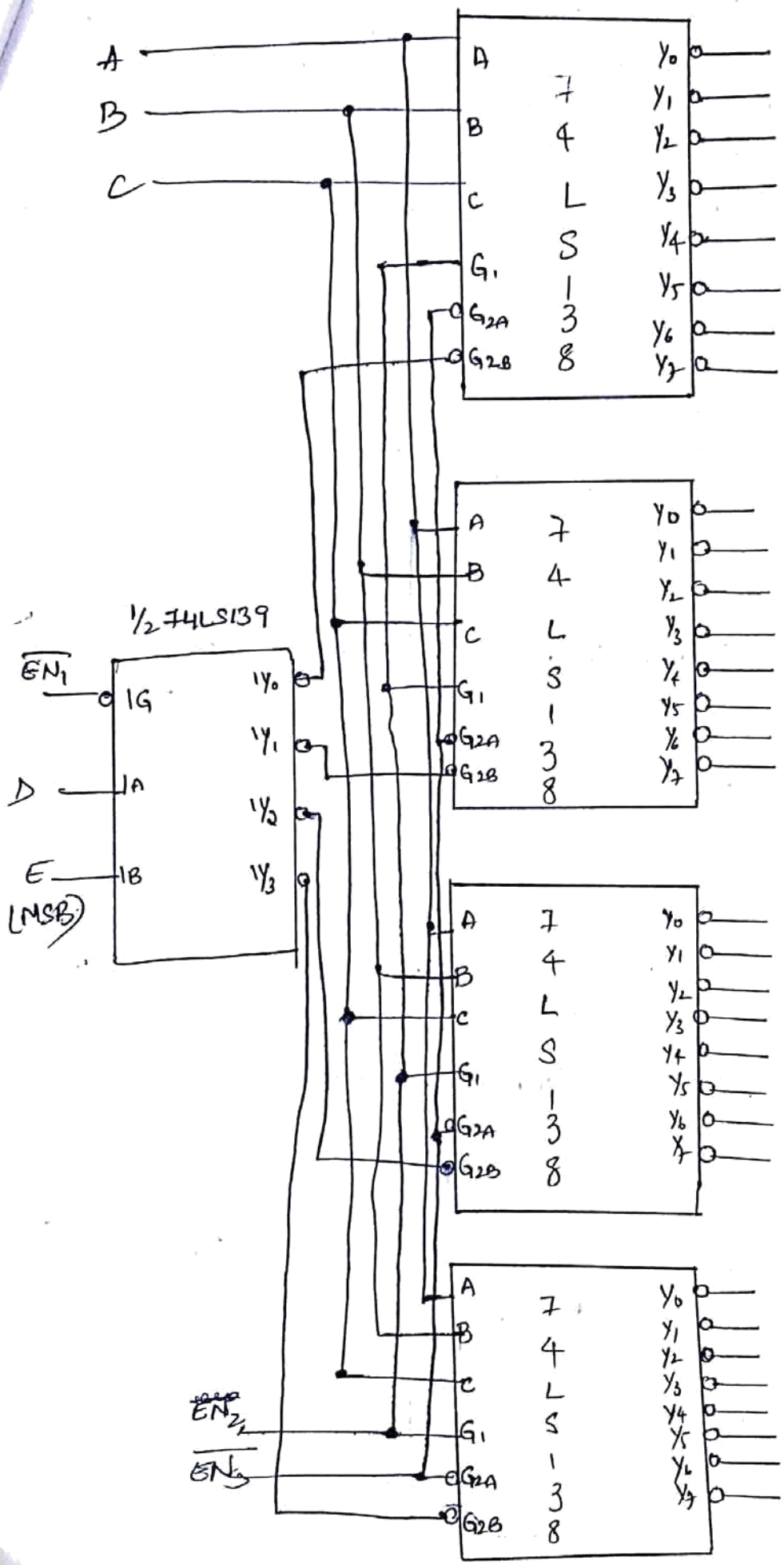
Inputs						Outputs							
G_{2B}	G_{2A}	G_1	C	B	A	$\bar{Y}_7$	$\bar{Y}_6$	$\bar{Y}_5$	$\bar{Y}_4$	$\bar{Y}_3$	$\bar{Y}_2$	$\bar{Y}_1$	$\bar{Y}_0$
1	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	0	X	X	X	1	1	1	1	1	1	1	1
0	0	1	0	0	0	1	1	1	1	1	1	1	0
0	0	1	0	0	1	1	1	1	1	1	1	0	1
0	0	1	0	1	0	1	1	1	1	1	0	1	1
0	0	1	0	1	1	1	1	1	1	0	1	1	1
0	0	1	1	0	0	1	1	1	0	1	1	1	1
0	0	1	1	0	1	1	1	0	1	1	1	1	1
0	0	1	1	1	0	1	0	1	1	1	1	1	1
0	0	1	1	1	1	0	1	1	1	1	1	1	1

74x139 Dual 2 to 4 Decoders —



Inputs			outputs			
$1\bar{G}$	1B	1A	$1\bar{Y}_3$	$1\bar{Y}_2$	$1\bar{Y}_1$	$1\bar{Y}_0$
1	X	X	1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

gn : 5 to 32 decoder using one 2 to 4 and four 3-8 deco-
der IC's.



→ 4th order to 32

$\overline{EN_1} = 0$

$EN_2 = 1$

$\overline{EN_3} = 0$

EDCBA.

If ED = 00 → $1Y_0 \Rightarrow Y_0 - Y_7$

01 → $1Y_1 \Rightarrow Y_8 - Y_{15}$

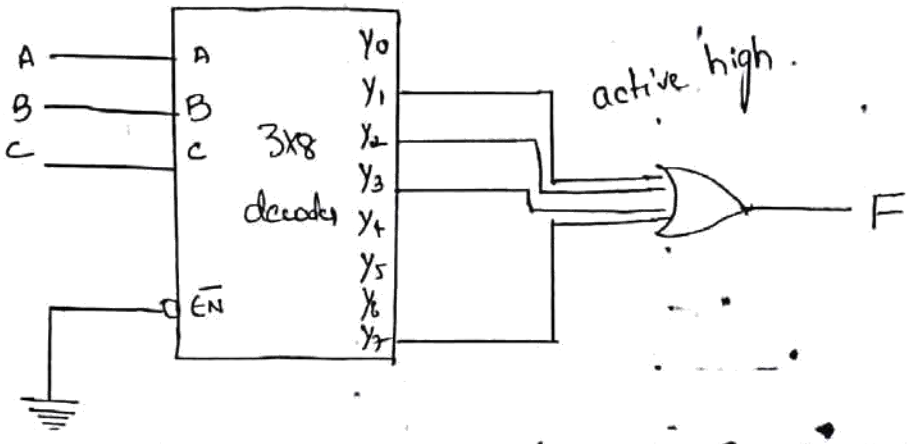
10 → $1Y_2 \Rightarrow Y_{16} - Y_{23}$

11 → $1Y_3 \Rightarrow Y_{24} - Y_{31}$

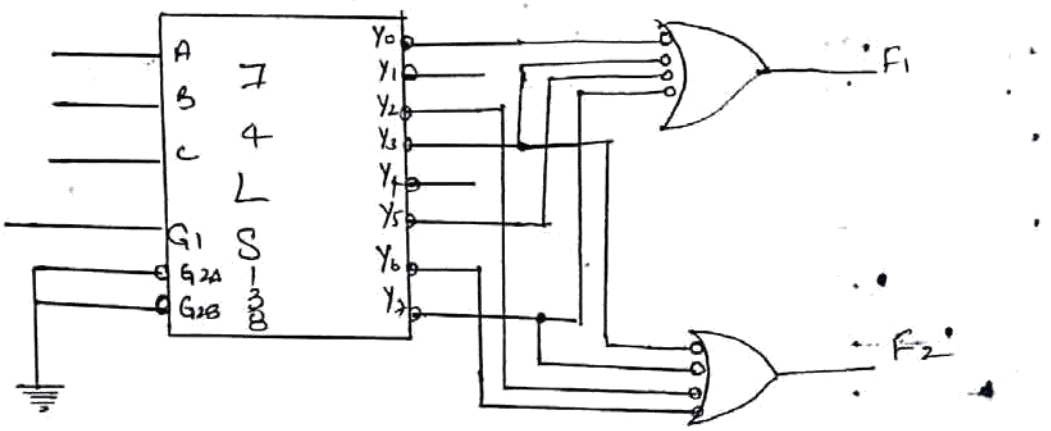
Realization of multiple o/p functions using Binary Decoder.

— The combination of decoder and External logic gates can used to implement Single or multiple o/p functions. The decoder generates minterms for input Variables. Thus by logically ORing Specified minterms we can implement the given function.

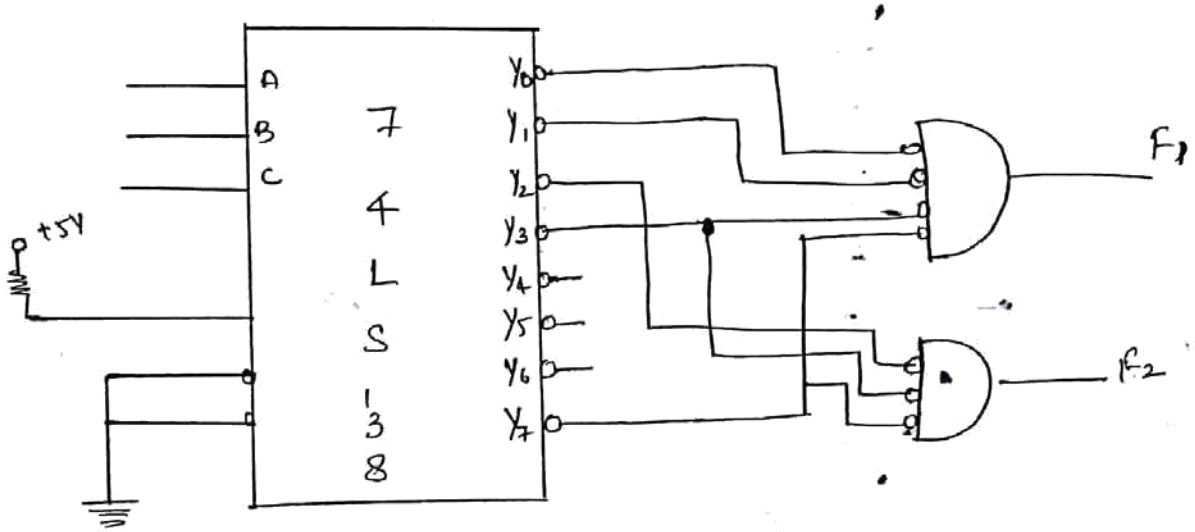
1) $f = \Sigma(1, 2, 3, 7)$ using 3 to 8 decoder.



2) $F(A,B,C) = \Sigma m(0, 3, 5, 7)$ and $f_2(A,B,C) = \Sigma m(2, 3, 6, 7)$ using 74LS138



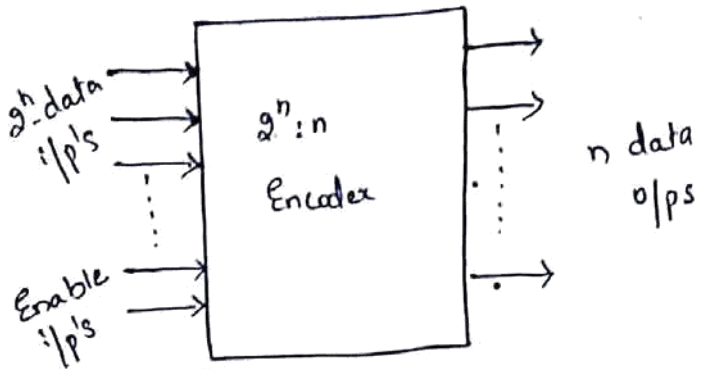
3) $f(A,B,C) = \pi(0, 1, 3, 7)$ and $f_2(A,B,C) = \pi(2, 3, 7)$



Implementation

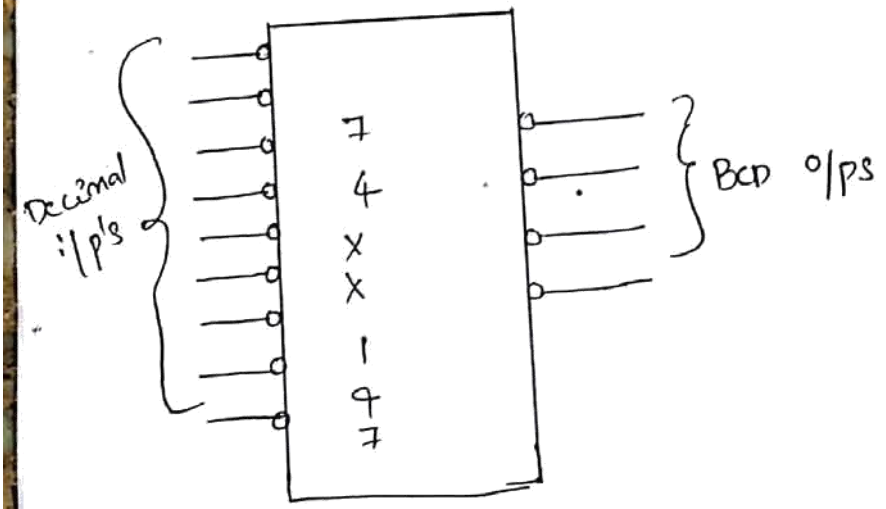
ODER :

An Encoder is a digital ckt that performs the inverse operation of decoder. An Encoder has 2^n input lines and n o/p lines. In Encoder the output lines generate the binary code corresponding to the input value.



Decimal to BCD Encoder :- (priority Encoder)

— Decimal to BCD Encoder, usually has ten input lines and four o/p lines. The decoded decimal data acts as an input for decoder and Encoded BCD o/p is available on the four o/p lines.

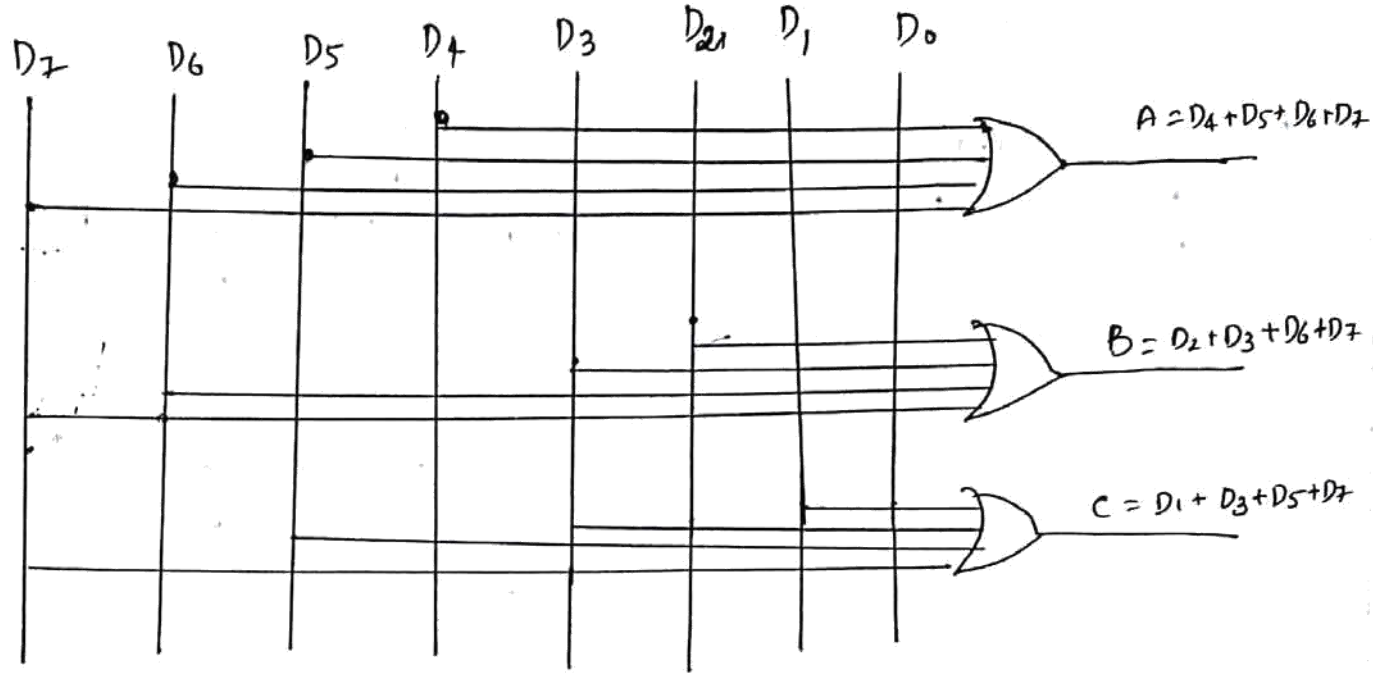


— It has nine input lines and four o/p lines. Both i/p and o/p lines are asserted low. It is important to note that there is no i/p line for decimal Zero. When this condition occurs all o/p lines are 1.

	Decimal Value	Inputs									Outputs			
		1	2	3	4	5	6	7	8	9	D	C	B	A
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1	0
2	2	X	0	1	1	1	1	1	1	1	1	1	0	1
3	3	X	X	0	1	1	1	1	1	1	1	1	0	0
4	4	X	X	X	0	1	1	1	1	1	1	0	1	1
5	5	X	X	X	X	0	1	1	1	1	1	0	1	0
6	6	X	X	X	X	X	0	1	1	1	1	0	0	1
7	7	X	X	X	X	X	X	0	1	1	1	0	0	0
8	8	X	X	X	X	X	X	X	0	1	0	0	1	1
9	9	X	X	X	X	X	X	X	X	0	0	1	1	0

octal to Binary Encoder

Inputs								outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A	B	C
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


D7
D6
D5
D4
D3
D2
D1
D0
A = D4 + D5 + D6 + D7
B = D2 + D3 + D6 + D7
C = D1 + D3 + D5 + D7

List of Code Converters.

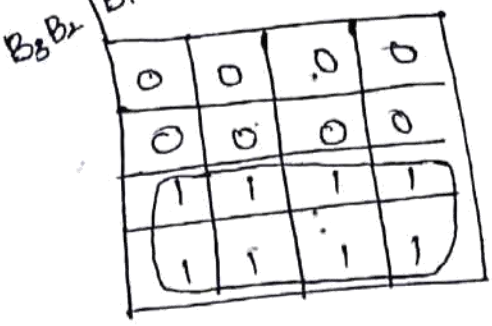
① Binary → Gray ② Gray → Binary ③ BCD → Gray

④ BCD → Ex-3 ⑤ Ex-3 → BCD ⑥ Binary - BCD :

of a 4-bit Binary-to-Gray code converter.

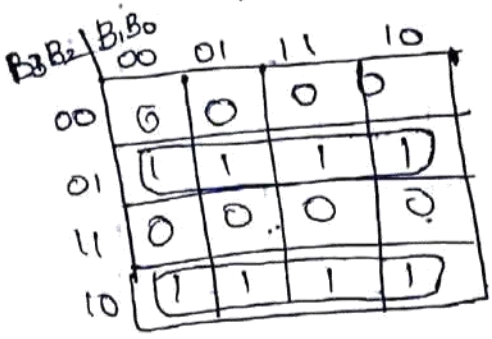
Binary				Gray			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0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

G_3



$G_3 = B_3$

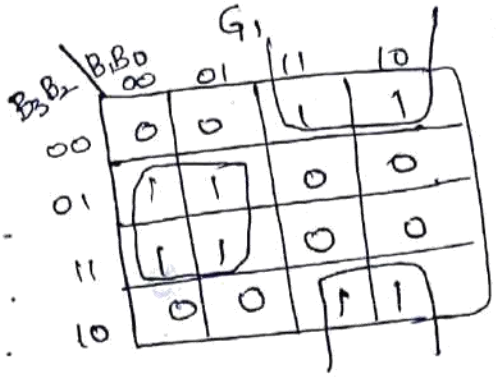
G_2



$G_2 = \bar{B}_3 B_2 + B_3 \bar{B}_2$

$B_2 \oplus B_3$

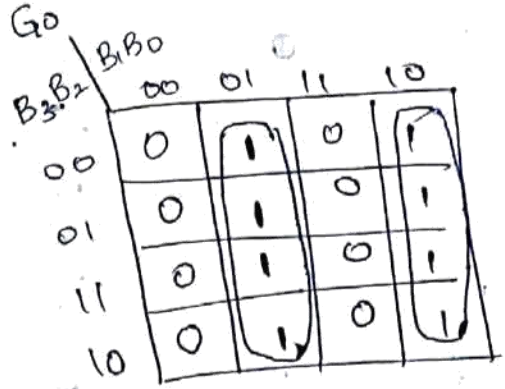
G_1

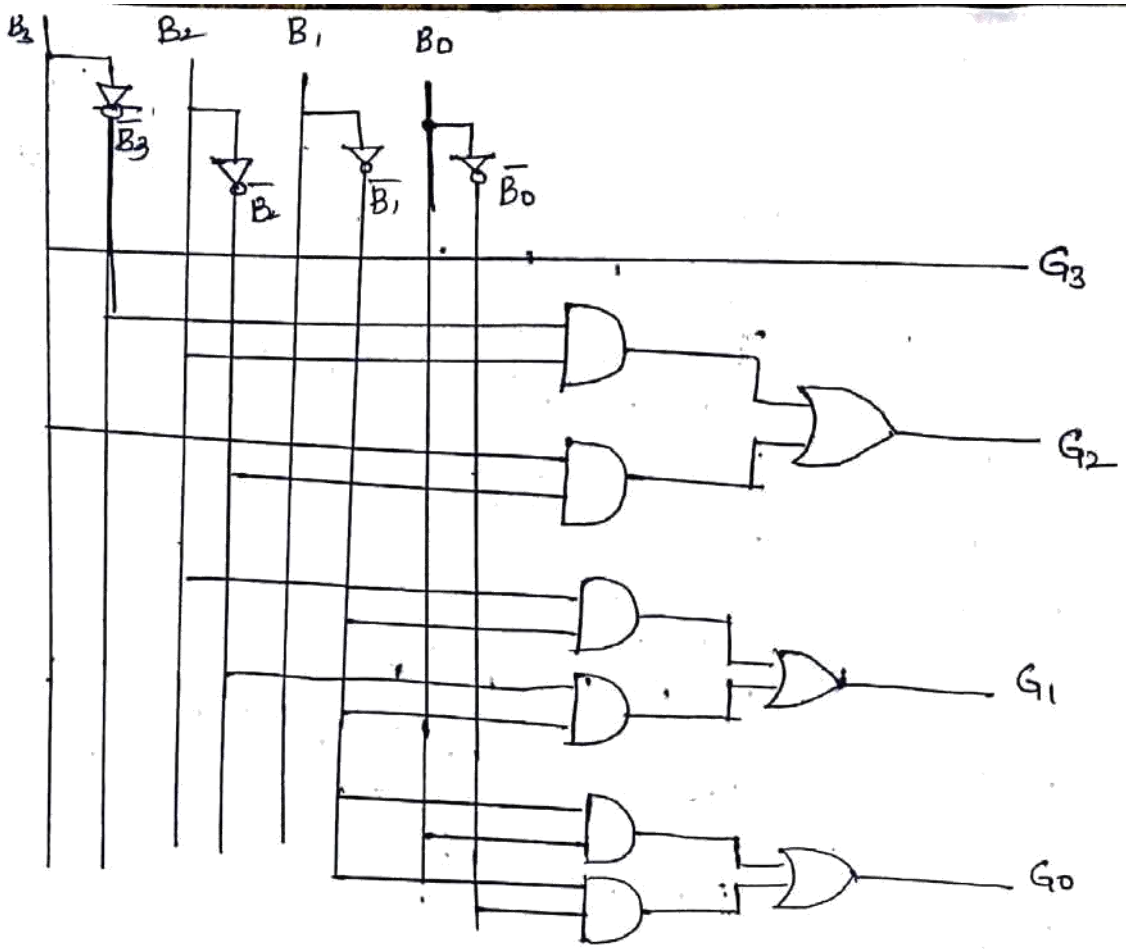


$G_1 = B_2 \bar{B}_1 + \bar{B}_2 B_1$

$B_1 \oplus B_2$

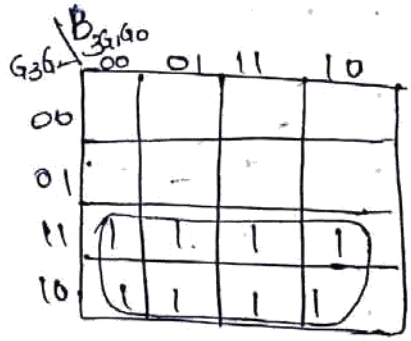
G_0



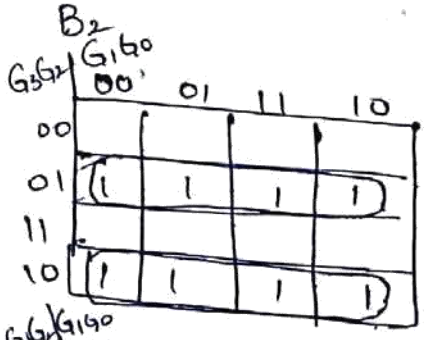


→ Design a 4-bit Gray to Binary code converter.

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

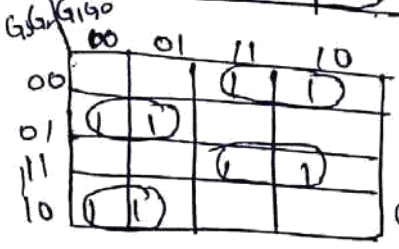


$B_3 = G_3$

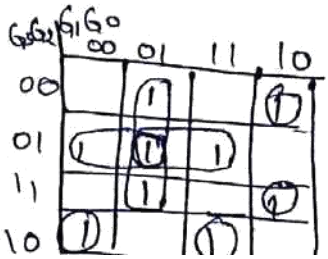


$B_2 = G_3\bar{G}_2 + \bar{G}_3G_2$

$= G_2 \oplus G_3$



$B_1 = G_3 \oplus G_2 \oplus G_1$

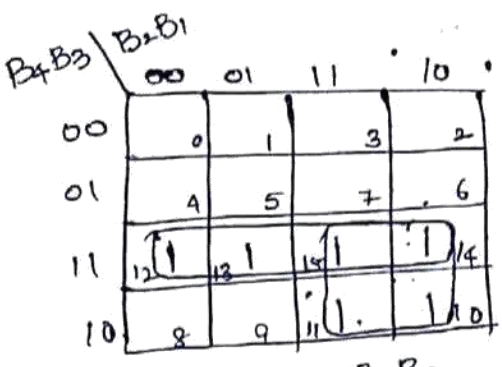


$B_0 = (G_3 \oplus G_2) \oplus (G_1 \oplus G_0)$

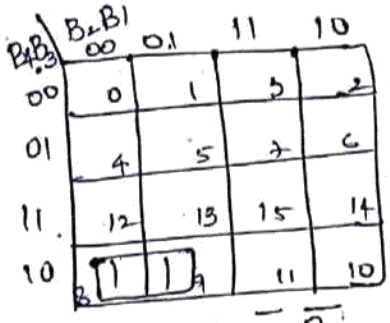
gn of a 4-bit Binary-to-BCD code converter.

Decimal	4-bit Binary $B_4\ B_3\ B_2\ B_1$	BCD output A B C D E
0	0 0 0 0	0 0 0 0 0
1	0 0 0 1	0 0 0 0 1
2	0 0 1 0	0 0 0 1 0
3	0 0 1 1	0 0 0 1 1
4	0 1 0 0	0 0 1 0 0
5	0 1 0 1	0 0 1 0 1
6	0 1 1 0	0 0 1 1 0
7	0 1 1 1	0 0 1 1 1
8	1 0 0 0	0 1 0 0 0
9	1 0 0 1	0 1 0 0 1
10	1 0 1 0	1 0 0 0 0
11	1 0 1 1	1 0 0 0 1
12	1 1 0 0	1 0 0 1 0
13	1 1 0 1	1 0 0 1 1
14	1 1 1 0	1 0 0 1 0
15	1 1 1 1	1 0 1 0 1

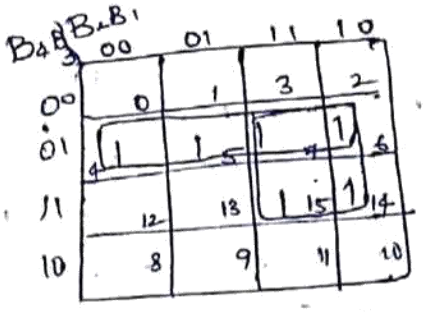
Block diagram



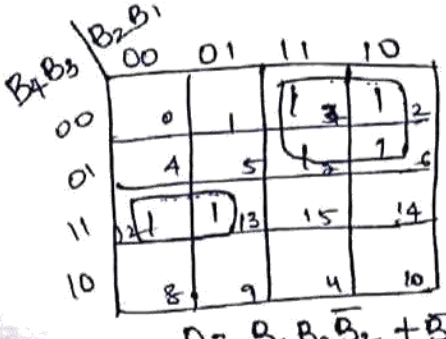
$A = B_4B_3 + B_4B_2$



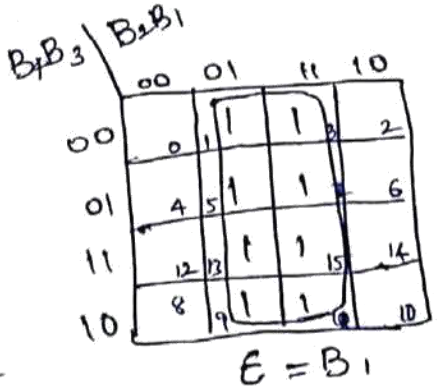
$B = B_4\bar{B}_3\bar{B}_2$



$C = \bar{B}_4B_3 + B_3B_2$



$D = B_4B_3\bar{B}_2 + \bar{B}_4B_2$



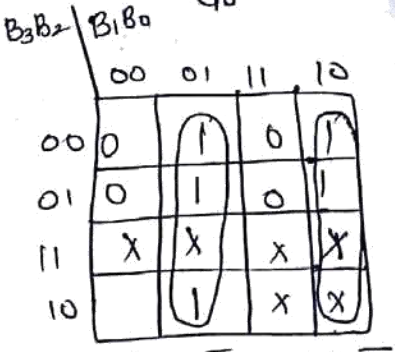
$E = B_1$

Design of a BCD-to-Gray code converters K-map Simplification

B_3	B_2	B_1	B_0	D	C	B	A
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	1				
1	1	0	0				
1	1	0	1				
1	1	1	0				
1	1	1	1				

} dont cares.

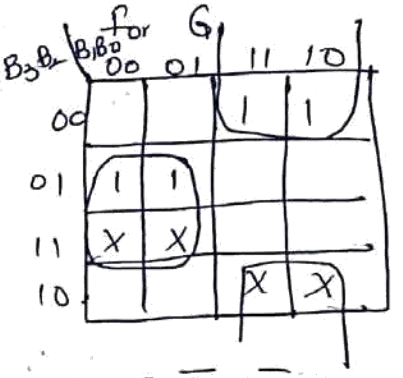
for G_0



$G_0 = \overline{B_1}B_0 + B_1\overline{B_0}$

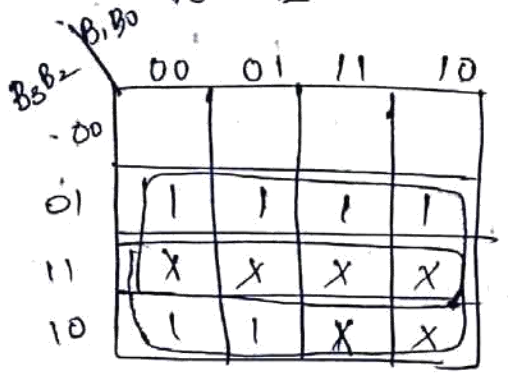
$B_1 \oplus B_0$

for G_1



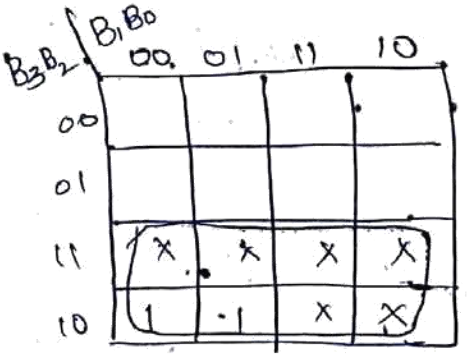
$G_1 = B_2\overline{B_1} + \overline{B_2}B_1$

for G_2



$G_2 = B_2 + B_3$

for G_3



$G_3 = B_3$

Design of full-Subtractor:-

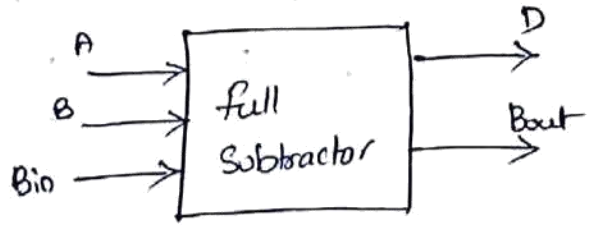
1. F.S Subbracts two bits and considering borrow from preceeding column produces difference D, Borrow (Bout)

2. A, B, Bin → i/p's

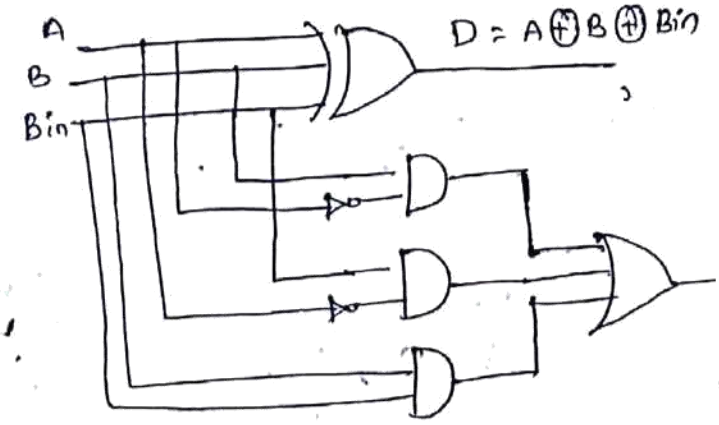
   D, Bout → o/p's

A	B	Bin	D	Bout
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

3. Block diagram

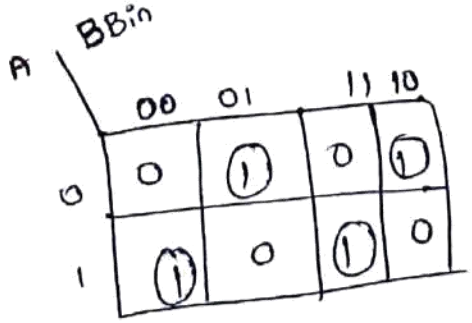


5. logic diagram



4. Simplify o/p function variables.

D

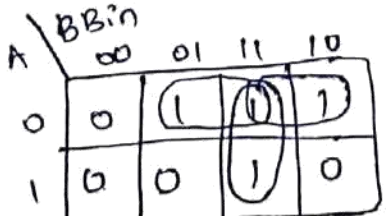


$D = \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}\bar{B}_{in} + ABB_{in}$

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

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

Bout



$B_{out} = \bar{A}B_{in} + \bar{A}B + BB_{in}$

Subtractors: Half - Subtractor → Subtracts one bit from other bit

full - Subtractor → Subtrahend is subtracted from m and considering borrow from preceding Colu

Design of Half - Subtractor: (LSB Subtraction)

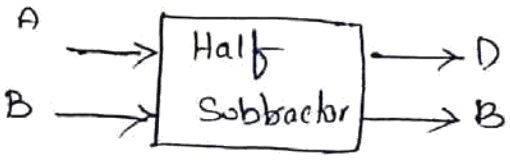
1. H.S is Combinational ckt that subtracts one bit from other produces the difference D, borrow B.

2. A, B → i/p's ; D, B → o/p's.

3. T.T

I/p's		o/p's	
A	Bin	D	Bout
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Block diagram



4. Simplify o/p function Variables.

D

A \ B	0	1
0	0	(1)
1	(1)	0

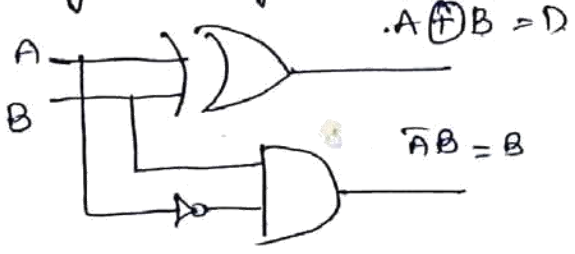
$D = \bar{A}B + A\bar{B} = A \oplus B$

B

A \ B	0	1
0	0	(1)
1	0	0

$B = \bar{A}B$

5. logic diagram



Hazards and Hazard - free Realizations:-

→ Hazards are unwanted switching transients that may appear at o/p of ckt because different paths exhibit different delays. Such a transient is also called a glitch or Spurious Spike; which is caused by hazards behaviour of logic ckt.

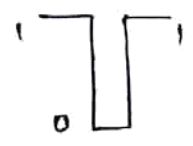
→ A hazard in a combinational ckt is a condition where a single variable change produces a momentary o/p change when no o/p change should occur.

→ Hazards are of two types
(i) Static Hazard (ii) Dynamic Hazard
↳ a) static 1- hazard
b) static 0- hazard.

Static - 1 hazard:

Suppose all the i/p's are assigned some level and only i/p say x changes from 0 to 1 or 1 to 0. If o/p is Expected to be at 1 regardless of changing variable, the Spurious 0 level for a short interval is called static 1 hazard.

Static 0- hazard: If o/p is Expected to be at '0' regardless of changing variable, the Spurious 1 Level for a short interval is called a static 0 hazard.

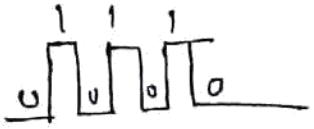


Static - 1 - hazard

static 0- hazard.

→ Static hazards can be eliminated by using redeundant gates.

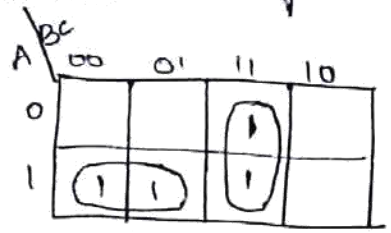
→ When o/p changes three or more times when it should change from 1 to 0 or 0 to 1 only once, it is called dynamic hazard.



⇒ When a ckt is implemented in SOP with AND-OR gates or with NAND gates, Removal of Static 1(0) hazard generates that no static 0(1) hazards or dynamic hazards will occur.

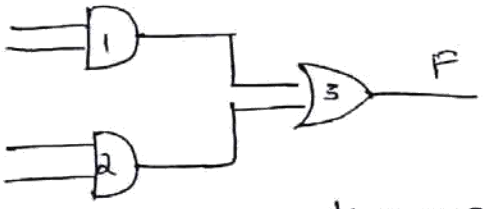
Static hazards :—

Consider the fn $F(A,B,C) = \sum m(3,4,5,7)$

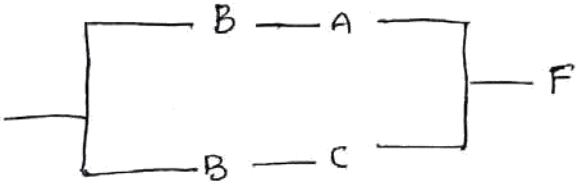


a) K-map.

$f = A\bar{B} + BC$



b) Logic diagram



c) Contact networks.

* let A=1, B=1, C=1, and only B is changing from 1 to 0. The o/p f has to remain at logic 1

(i) When B=1, o/p of gate 2 is 1, o/p of gate 1 is 0 and o/p F=1.

(ii) When B changes to 0, o/p of gate 2 is 0, o/p of gate 1 is 1 and o/p f remains at 1.

(iii) for the change in B from 1 to 0, if gate 1 responds faster than G2, f will be 1 as expected.

let G_1 = 10ns $B=0$, $\bar{B}=1$, $A=1$ $\Rightarrow G_1=1 \rightarrow F=1$

G_2 = 20ns

b) gate G2 is faster than G1, the o/p becomes 0 before the o/p of G1 changes to 1, and then at very short time the o/p of G2 will be '0' resulting in an o/p of 0.

Let G1 = done, G2 = turn, B = 0 G1 = 0, C = 0 till 0 = turn after done G1 = 1, G2 = 0 C = 1

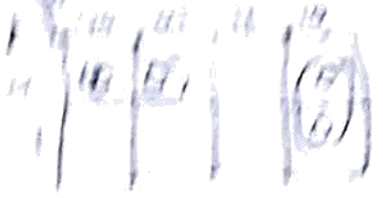
A little later of course the o/p gate to 1. This creates behaviour is known as static 1 hazard

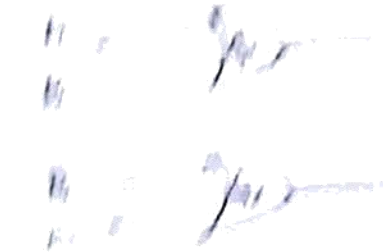
static 1 hazard

with contact when it is called the 1st hazard.

b) if we consider the pos evaluation of same fn, then

f = pos(A, 1, 2, 6)





let A = 0, B = 0, C = 0 and only B is changing from 0 to 1.

the o/p f has the domain as '0'

(ii) when B = 1, G1 = 0, G2 = 1 and for f = 0 when B changes to 1, o/p of G1 = 1, G2 = 0 and f remains at 0.

this the change in B from 0 to 1, if G2 responds faster than G1 f will be 0 as expected.

if G1 is faster than G2, the o/p becomes 1 before the o/p of G2 changes to 0 and for very short time o/p's of both

G_1 and G_2 will be 1 resulting in an o/p of 1.

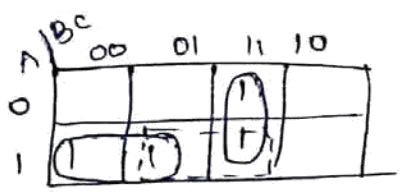
This Erratic behaviour is known as static-0 hazard. with free contact networks it is called cut set hazard.

Hazard free - Realization :—

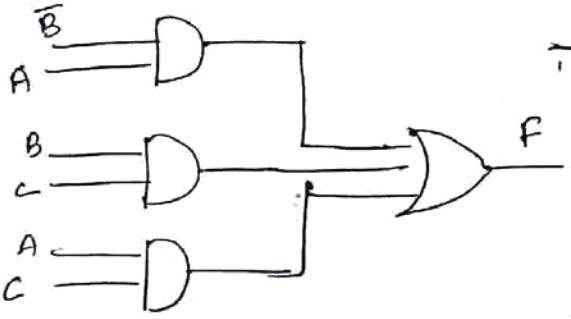
→ static -1 hazards arises because two adjacent 1's are covered by different subcubes.

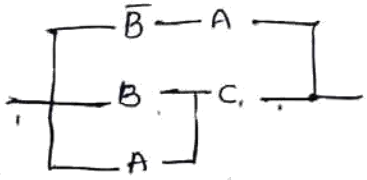
If we want to Ensure that this pair of adjacent 1's is covered by same subcube shown marked on map, we need to add one more AND gate.

The Corresponding contact n/w will have one more path. This realization will now have no static -1 hazards but the fn contains a redundant term BC



$f = A\bar{B} + BC + AC$

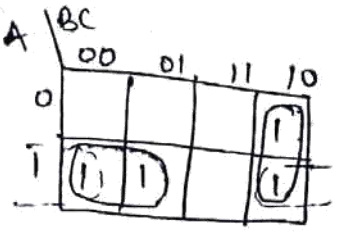




→ The removal of static 1 hazards in a fn by addition of subcubes does not guarantee the removal of static 0 hazards in fn.

with

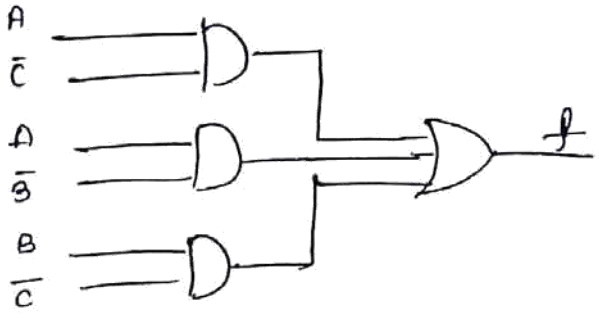
Realize the Switching fn $F(A,B,C) = \Sigma m\ (2,4,5,6)$ by a hazard free logic gate network.



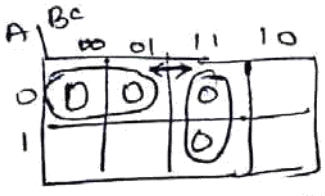
$f = A\bar{C} + A\bar{B} + B\bar{C}$

If we consider Sop form, $F = A\bar{C} + A\bar{B} + B\bar{C}$

This Sop form is hazard-free because every pair of adjacent 1 is covered by some Subcube, and in order to satisfy this constraint, a Redaundant Sube $A\bar{C}$ has tobe added.



→ Suppose we wish to Realize the same fn in Pos



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

There will be a hazard marked by arrow if we Realize it as $f = (A+B)(\bar{B}+\bar{C})$

→ By Expanding pos form & ignoring term $B \cdot \bar{B} = 0$ results in Same Sop form which is hazard free.

The hazard in Pos form must be attributed to ignoring terms like $B \cdot \bar{B} = 0$

→ we can conclude that hazard free fn has to be Realized in its original form without Resorting to Simplification or factoring.

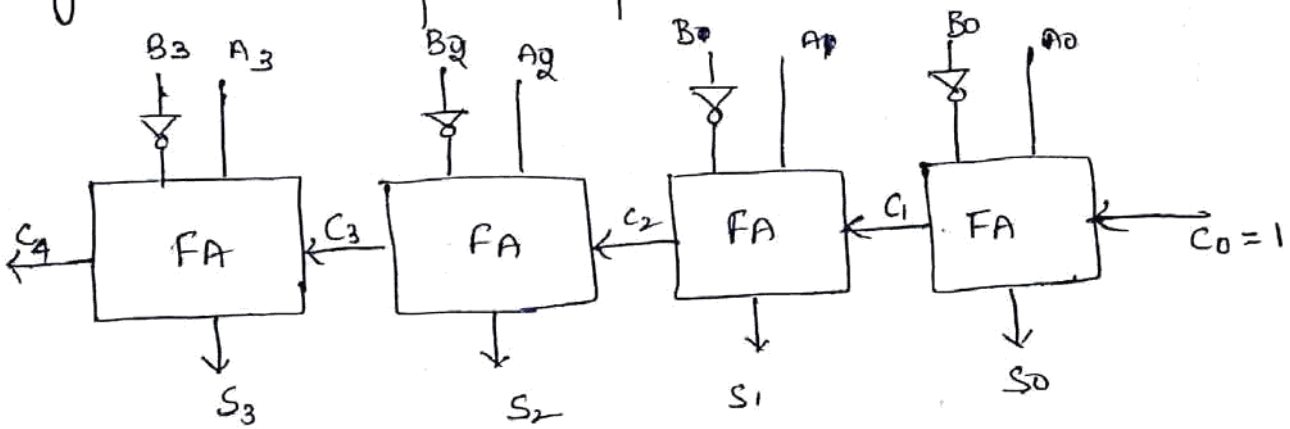
Essential Hazards:

- Essential hazard that occur in asynchronous Sequential ckts.
- It is caused due to unequal delays, along two or more paths that originate from same input.
- Excessive delay through an inverter ckt in comparision to delay associated with feedback path may cause such a hazard.
- Essential hazards cannot be corrected by adding redundant gates as in static hazard.

This can be corrected by adjusting the amount of delay in effected path.

→ To avoid essential hazards, each feedback loop must be handled with individual care to ensure that the delay in feedback loop is long enough compared to delays of other signals that originate from input terminals.

Binary Subtractor (parallel subtractor)

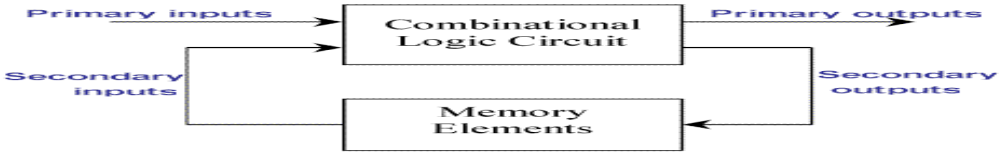


UNIT NO –IV
SEQUNTIAL CIRCUIT

Classification of sequential circuits: Sequential circuits may be classified as two types.

1. Synchronous sequential circuits
2. Asynchronous sequential circuits

Combinational logic refers to circuits whose output is strictly depended on the present value of the inputs. As soon as inputs are changed, the information about the previous inputs is lost, that is, combinational logics circuits have no memory. Although every digital system is likely to have combinational circuits, most systems encountered in practice also include memory elements, which require that the system be described in terms of sequential logic. Circuits whose output depends not only on the present input value but also the past input value are known as **sequential logic circuits**. The mathematical model of a sequential circuit is usually referred to as a **sequential machine**.



Comparison between combinational and sequential circuits

Combinational circuit	Sequential circuit
1. In combinational circuits, the output variables at any instant of time are dependent only on the present input variables	1. in sequential circuits the output variables at any instant of time are dependent not only on the present input variables, but also on the present state
2.memory unit is not requires in combinational circuit	2.memory unit is required to store the past history of the input variables
3. these circuits are faster because the delay between the i/p and o/p due to propagation delay of gates only	3. sequential circuits are slower than combinational circuits
4. easy to design	4. comparatively hard to design

Level mode and pulse mode asynchronous sequential circuits:

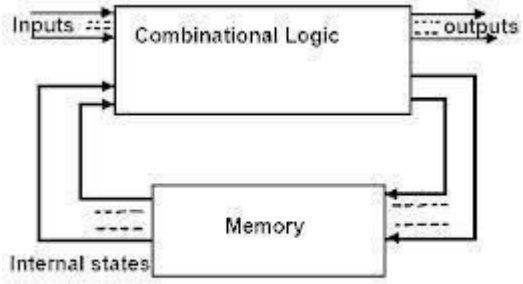


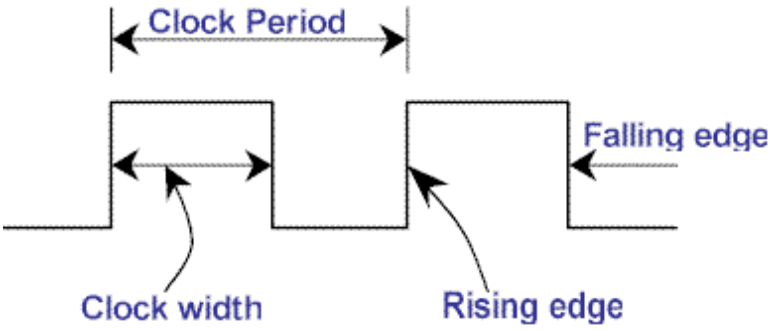
Figure 1: Asynchronous Sequential Circuit

Fig shows a block diagram of an asynchronous sequential circuit. It consists of a combinational circuit and delay elements connected to form the feedbackloops. The present state and next state variables in asynchronous sequential circuits called secondary variables and excitation variables respectively..

There are two types of asynchronous circuits: fundamental mode circuits and pulse mode circuits.

Synchronous and Asynchronous Operation:

Sequential circuits are divided into two main types: **synchronous** and **asynchronous**. Their classification depends on the timing of their signals.***Synchronous*** sequential circuits change their states and output values at discrete instants of time, which are specified by the rising and falling edge of a free-running **clock signal**. The clock signal is generally some form of square wave as shown in Figure below.



From the diagram you can see that the **clock period** is the time between successive transitions in the same direction, that is, between two rising or two falling edges. State transitions in synchronous sequential circuits are made to take place at times when the clock is making a transition from 0 to 1 (rising edge) or from 1 to 0 (falling edge). Between successive clock pulses there is no change in the information stored in memory.

The reciprocal of the clock period is referred to as the **clock frequency**. The **clock width** is defined as the time during which the value of the clock signal is equal to 1. The ratio of the clock width and clock period is referred to as the duty cycle. A clock signal is said to

be **active high** if the state changes occur at the clock's rising edge or during the clock width. Otherwise, the clock is said to be **active low**. Synchronous sequential circuits are also known as **clocked sequential circuits**.

The memory elements used in synchronous sequential circuits are usually flip-flops. These circuits are binary cells capable of storing one bit of information. A flip-flop circuit has two outputs, one for the normal value and one for the complement value of the bit stored in it. Binary information can enter a flip-flop in a variety of ways, a fact which give rise to the different types of flip-flops. For information on the different types of basic flip-flop circuits and their logical properties, see the previous tutorial on flip-flops.

In ***asynchronous*** sequential circuits, the transition from one state to another is initiated by the change in the primary inputs; there is no external synchronization. The memory commonly used in asynchronous sequential circuits are time-delayed devices, usually implemented by feedback among logic gates. Thus, asynchronous sequential circuits may be regarded as combinational circuits with feedback. Because of the feedback among logic gates, asynchronous sequential circuits may, at times, become unstable due to transient conditions. The instability problem imposes many difficulties on the designer. Hence, they are not as commonly used as synchronous systems.

Fundamental Mode Circuits assumes that:

1. The input variables change only when the circuit is stable
2. Only one input variable can change at a given time
3. Inputs are levels are not pulses

A pulse mode circuit assumes that:

1. The input variables are pulses instead of levels
2. The width of the pulses is long enough for the circuit to respond to the input
3. The pulse width must not be so long that is still present after the new state is reached.

Latches and flip-flops

Latches and flip-flops are the basic elements for storing information. One latch or flip-flop can store one bit of information. The main difference between latches and flip-flops is that for latches, their outputs are constantly affected by their inputs as long as the enable signal is asserted. In other words, when they are enabled, their content changes immediately when their inputs change. Flip-flops, on the other hand, have their content change only either at the rising or falling edge of the enable signal. This enable signal is usually the controlling clock signal. After the rising or falling edge of the clock, the flip-flop content remains constant even if the input changes.

There are basically four main types of latches and flip-flops: SR, D, JK, and T. The major differences in these flip-flop types are the number of inputs they have and how they change state. For each type, there are also different variations that enhance their operations. In this chapter, we

will look at the operations of the various latches and flip-flops.the flip-flops has two outputs, labeled Q and Q‘. the Q output is the normal output of the flip flop and Q‘ is the inverted output.

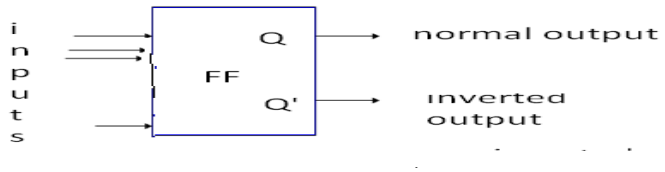


Figure: basic symbol of flipflop

A latch may be an active-high input latch or an active –LOW input latch.active –HIGH means that the SET and RESET inputs are normally resting in the low state and one of them will be pulsed high whenever we want to change latch outputs.

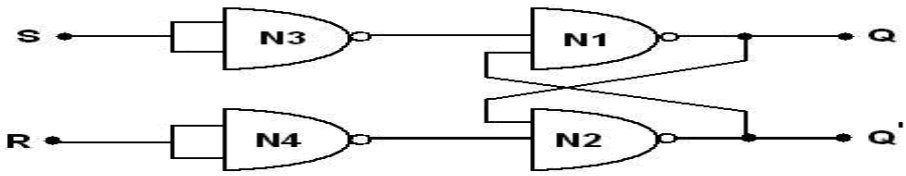
SR latch:

The latch has two outputs Q and Q‘. When the circuit is switched on the latch may enter into any state. If Q=1, then Q‘=0, which is called SET state. If Q=0, then Q‘=1, which is called RESET state. Whether the latch is in SET state or RESET state, it will continue to remain in the same state, as long as the power is not switched off. But the latch is not an useful circuit, since there is no way of entering the desired input. It is the fundamental building block in constructing flip-flops, as explained in the following sections

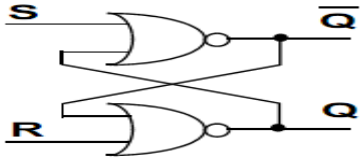
NAND latch

NAND latch is the fundamental building block in constructing a flip-flop. It has the property of holding on to any previous output, as long as it is not disturbed.

The opration of NAND latch is the reverse of the operation of NOR latch.if 0‘s are replaced by 1‘s and 1‘s are replaced by 0‘s we get the same truth table as that of the NOR latch shown



NOR latch



S	R	Q	$\overline{Q}$	Function
0	0	Q^+	$\overline{Q}^+$	Storage State
0	1	0	1	Reset
1	0	1	0	Set
1	1	0-?	0-?	Indeterminate State

The analysis of the operation of the active-HIGHNOR latch can be summarized as follows.

1. SET=0, RESET=0: this is normal resting state of the NOR latch and it has no effect on the output state. Q and Q‘ will remain in whatever stste they were prior to the occurrence of this input condition.
2. SET=1, RESET=0: this will always set Q=1, where it will remain even after SET returns to 0
3. SET=0, RESET=1: this will always reset Q=0, where it will remain even after RESET returns to 0
4. SET=1,RESET=1; this condition tries to SET and RESET the latch at the same time, and it produces Q=Q‘=0. If the inputs are returned to zero simultaneously, the resulting output stste is erratic and unpredictable. This input condition should not be used.

The SET and RESET inputs are normally in the LOW state and one of them will be pulsed HIGH. Whenever we want to change the latch outputs..

RS Flip-flop:

The basic flip-flop is a one bit memory cell that gives the fundamental idea of memory device. It constructed using two NAND gates. The two NAND gates N1 andN2 are connected such that, output of N1 is connected to input of N2 and output of N2 to input of N1. These form the feedback path the inputs are S and R, and outputs are Q and Q‘. The logic diagram and the block diagram of R-S flip-flop with clocked input

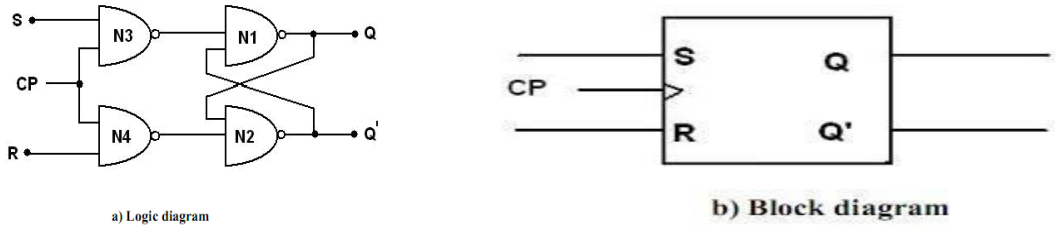


Figure: RS Flip-flop

The flip-flop can be made to respond only during the occurrence of clock pulse by adding two NAND gates to the input latch. So synchronization is achieved. i.e., flip-flops are allowed to change their states only at particular instant of time. The clock pulses are generated by a clock pulse generator. The flip-flops are affected only with the arrival of clock pulse**.**

Operation:

1. When CP=0 the output of N3 and N4 are 1 regardless of the value of S and R. This is given as input to N1 and N2. This makes the previous value of Q and Q‘unchanged.

2. When CP=1 the information at S and R inputs are allowed to reach the latch and change of state in flip-flop takes place.

3. CP=1, S=1, R=0 gives the SET state i.e., Q=1, Q‘=0.

4. CP=1, S=0, R=1 gives the RESET state i.e., Q=0, Q‘=1.

5. CP=1, S=0, R=0 does not affect the state of flip-flop.

6. CP=1, S=1, R=1 is not allowed, because it is not able to determine the next state. This condition is said to be a ‑race condition‖.

In the logic symbol CP input is marked with a triangle. It indicates the circuit responds to an input change from 0 to 1. The characteristic table gives the operation conditions of flip-flop. Q(t) is the present state maintained in the flip-flop at time ‗t‘. Q(t+1) is the state after the occurrence of clock pulse.

Truth table

S	R	$Q_{(t+1)}$	Comments
0	0	Q_t	No change
0	1	0	Reset / clear
1	0	1	Set
1	1	*	Not allowed

Edge triggered RS flip-flop:

Some flip-flops have an RC circuit at the input next to the clock pulse. By the design of the circuit the R-C time constant is much smaller than the width of the clock pulse. So the output changes will occur only at specific level of clock pulse. The capacitor gets fully charged when clock pulse goes from low to high. This change produces a narrow positive spike. Later at the trailing edge it produces narrow negative spike. This operation is called edge triggering, as the flip-flop responds only at the changing state of clock pulse. If output transition occurs at rising edge of clock pulse (0□1), it is called positively edge triggering. If it occurs at trailing edge 1□0) it is called negative edge triggering. Figure shows the logic and block diagram.

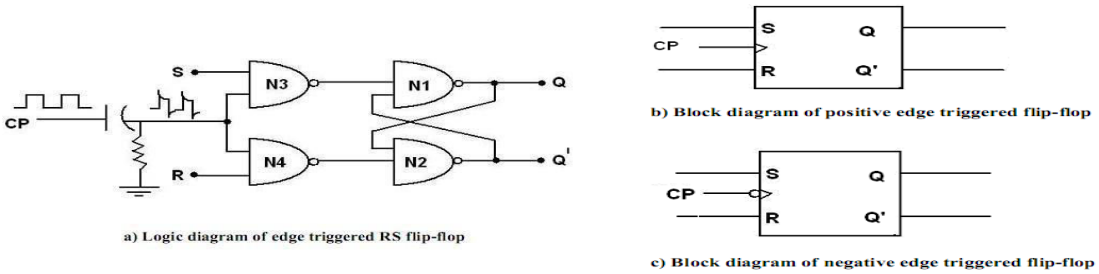
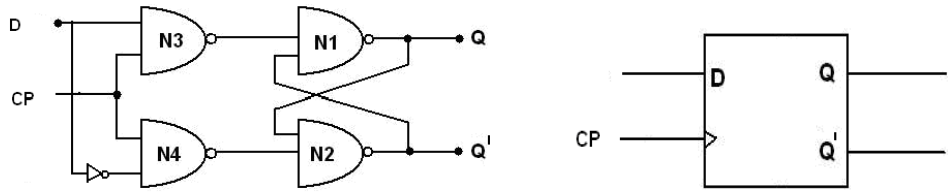


Figure: Edge triggered RS flip-flop

D flip-flop:

The D flip-flop is the modified form of R-S flip-flop. R-S flip-flop is converted to D flip-flop by adding an inverter between S and R and only one input D is taken instead of S and R. So one input is D and complement of D is given as another input. The logic diagram and the block diagram of D flip-flop with clocked input



a) Logic diagram

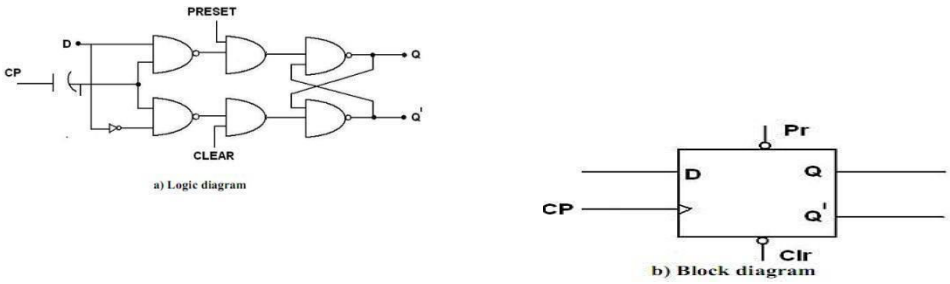
b) Block diagram

When the clock is low both the NAND gates (N1 and N2) are disabled and Q retains its last value. When clock is high both the gates are enabled and the input value at D is transferred to its output Q. D flip-flop is also called ―Data flip-flop‖.

Truth table

CP	D	Q
0	x	Previous state
1	0	0
1	1	1

Edge Triggered D Flip-flop:



a) Logic diagram

b) Block diagram

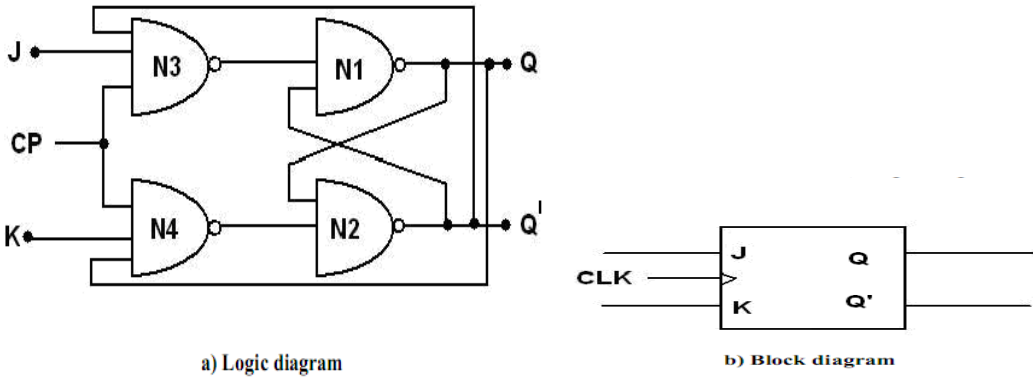
Truth table

PRESET	CLEAR	CP	D	Q
0	0	X	X	*(forbidden)
0	1	X	X	1
1	0	X	X	0
1	0	0	X	NC
1	1	1	X	NC
1	1	↓	X	NC
1	1	↑	0	0
1	1	↑	1	1

Figure: truth table, block diagram, logic diagram of edge triggered flip-flop

JK flip-flop (edge triggered JK flip-flop)

The race condition in RS flip-flop, when R=S=1 is eliminated in J-K flip-flop. There is a feedback from the output to the inputs. Figure 3.4 represents one way of building a JK flip-flop**.**



a) Logic diagram

b) Block diagram

Truth table

J	K	$Q_{(t+1)}$	Comments
0	0	Q_t	No change
0	1	0	Reset / clear
1	0	1	Set
1	1	Q'_t	Complement/ toggle.

Figure: JK flip-flop

The J and K are called control inputs, because they determine what the flip-flop does when a positive clock edge arrives.

Operation:

1. When J=0, K=0 then both N3 and N4 will produce high output and the previous value of Q and Q‘ retained as it is.

2. When J=0, K=1, N3 will get an output as 1 and output of N4 depends on the value of Q. The final output is Q=0, Q‘=1 i.e., reset state

3. When J=1, K=0 the output of N4 is 1 and N3 depends on the value of Q‘. The final output is Q=1 and Q‘=0 i.e., set state

4. When J=1, K=1 it is possible to set (or) reset the flip-flop depending on the current state of output. If Q=1, Q‘=0 then N4 passes ‘0‘to N2 which produces Q‘=1, Q=0 which is reset state. When J=1, K=1, Q changes to the complement of the last state. The flip-flop is said to be in the toggle state.

The characteristic equation of the JK flip-flop is:

$$Q_{next} = J\overline{Q} + \overline{K}Q$$

JK flip-flop operation[28]								
Characteristic table				Excitation table				
J	**K**	**Q_{next}**	**Comment**	**Q**	**Q_{next}**	**J**	**K**	**Comment**
0	0	Q	hold state	0	0	0	X	No change
0	1	0	reset	0	1	1	X	Set
1	0	1	set	1	0	X	1	Reset
1	1	Q	toggle	1	1	X	0	No change

T flip-flop:

If the T input is high, the T flip-flop changes state ("toggles") whenever the clock input is strobed. If the T input is low, the flip-flop holds the previous value. This behavior is described by the characteristic equation

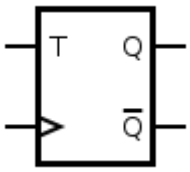


Figure : symbol for T flip flop

$$Q_{next} = T \oplus Q = T\overline{Q} + \overline{T}Q$$ (expanding the XOR operator

When T is held high, the toggle flip-flop divides the clock frequency by two; that is, if clock frequency is 4 MHz, the output frequency obtained from the flip-flop will be 2 MHz This "divide by" feature has application in various types of digital counters. A T flip-flop can also be built using a JK flip-flop (J & K pins are connected together and act as T) or D flip-flop (T input and $P_{revious}$ is connected to the D input through an XOR gate).

T flip-flop operation[28]							
Characteristic table				**Excitation table**			
T	Q	Q_{next}	**Comment**	Q	Q_{next}	T	**Comment**
0	0	0	hold state (no clk)	0	0	0	No change
0	1	1	hold state (no clk)	1	1	0	No change
1	0	1	toggle	0	1	1	Complement
1	1	0	toggle	1	0	1	Complement

Flip flop operating characteristics:

The operation characteristics specify the performance, operating requirements, and operating limitations of the circuits. The operation characteristics mentions here apply to all flip-flops regardless of the particular form of the circuit.

Propagation Delay Time: is the interval of time required after an input signal has been applied for the resulting output change to occur.

Set-up Time: is the minimum interval required for the logic levels to be maintained constantly on the inputs (J and K, or S and R, or D) prior to the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop.

Hold Time: is the minimum interval required for the logic levels to remain on the inputs after the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop.

Maximum Clock Frequency: is the highest rate that a flip-flop can be reliably triggered.

Power Dissipation: is the total power consumption of the device. It is equal to product of supply voltage (Vcc) and the current (Icc).

$P=V_{cc}.I_{cc}$

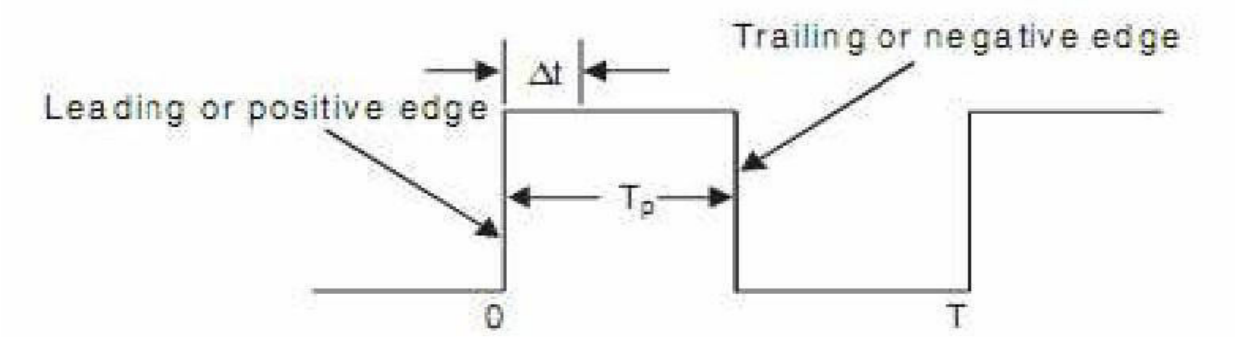
The power dissipation of a flip flop is usually in mW.

Pulse Widths: are the minimum pulse widths specified by the manufacturer for the Clock, SET and CLEAR inputs.

Clock transition times: for reliable triggering, the clock waveform transition times should be kept very short. If the clock signal takes too long to make the transitions from one level to other, the flip flop may either triggering erratically or not trigger at all.

Race around Condition

The inherent difficulty of an S-R flip-flop (i.e., S = R = 1) is eliminated by using the feedback connections from the outputs to the inputs of gate 1 and gate 2 as shown in Figure. Truth tables in figure were formed with the assumption that the inputs do not change during the clock pulse (CLK = 1). But the consideration is not true because of the feedback connections



- Consider, for example, that the inputs are J = K = 1 and Q = 1, and a pulse as shown in Figure is applied at the clock input.
- After a time interval t equal to the propagation delay through two NAND gates in series, the outputs will change to Q = 0. So now we have J = K = 1 and Q = 0.
- After another time interval of t the output will change back to Q = 1. Hence, we conclude that for the time duration of tP of the clock pulse, the output will oscillate between 0 and 1. Hence, at the end of the clock pulse, the value of the output is not certain. This situation is referred to as a race-around condition.
- Generally, the propagation delay of TTL gates is of the order of nanoseconds. So if the clock pulse is of the order of microseconds, then the output will change thousands of times within the clock pulse.
- This race-around condition can be avoided if tp< t < T. Due to the small propagation delay of the ICs it may be difficult to satisfy the above condition.
- A more practical way to avoid the problem is to use the master-slave (M-S) configuration as discussed below.

Applications of flip-flops:

Frequency Division: When a pulse waveform is applied to the clock input of a J-K flip-flop that is connected to toggle, the Q output is a square wave with half the frequency of the clock input. If more flip-flops are connected together as shown in the figure below, further division of the clock frequency can be achieved

. **Parallel data storage:** a group of flip-flops is called register. To store data of N bits, N flip-flops are required. Since the data is available in parallel form. When a clock pulse is applied to all flip-flops simultaneously, these bits will transfer will be transferred to the Q outputs of the flip flops.

Serial data storage: to store data of N bits available in serial form, N number of D-flip-flops is connected in cascade. The clock signal is connected to all the flip-flops. The serial data is applied to the D input terminal of the first flip-flop.

Transfer of data: data stored in flip-flops may be transferred out in a serial fashion, i.e., bit-by-bit from the output of one flip-flops or may be transferred out in parallel form.

Excitation Tables:

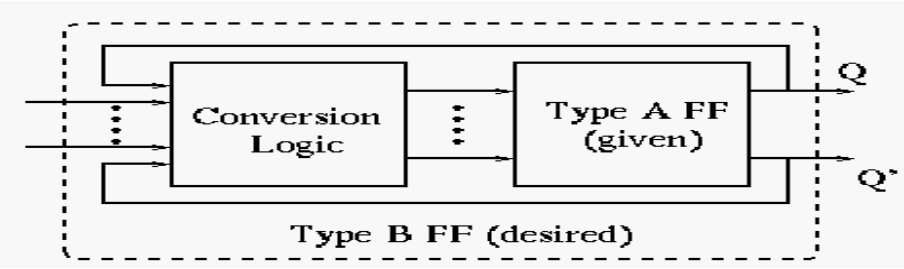
Previous State -> Present State	D
0 -> 0	0
0 -> 1	1
1 -> 0	0
1 -> 1	1

Previous State -> Present State	J	K
0 -> 0	0	X
0 -> 1	1	X
1 -> 0	X	1
1 -> 1	X	0

Previous State -> Present State	S	R
0 -> 0	0	X
0 -> 1	1	0
1 -> 0	0	1
1 -> 1	X	0

Previous State -> Present State	T
0 -> 0	0
0 -> 1	1
1 -> 0	1
1 -> 1	0

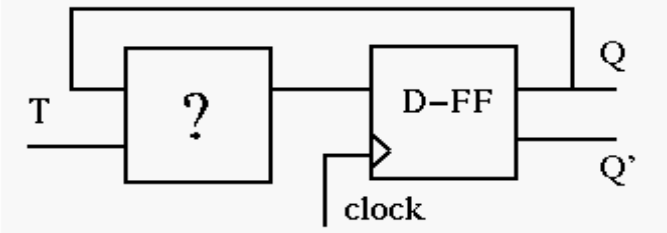
Conversions of flip-flops:



The key here is to use the excitation table, which shows the necessary triggering signal (S,R,J,K, D and T) for a desired flip-flop state transition :

Q_t	Q_{t+1}	S	R	J	K	D	T
0	0	0	x	0	x	0	0
0	1	1	0	1	x	1	1
1	0	0	1	x	1	0	1
1	1	x	0	x	0	1	0

Convert a D-FF to a T-FF:



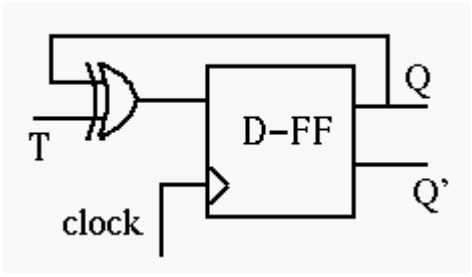
We need to design the circuit to generate the triggering signal D as a function of T and Q:
. Consider the excitation table:

$$D = f(T, Q).$$

Q_t	Q_{t+1}	T	D
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

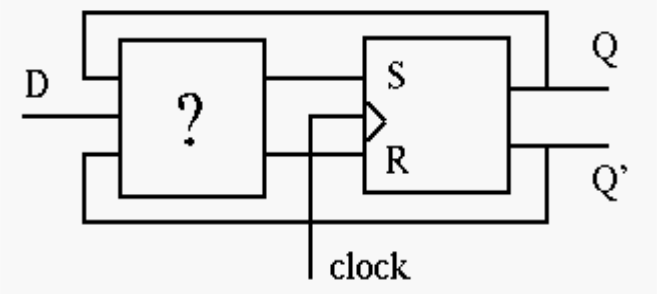
Treating as a function of and current FF state , we have

$$D = T'Q + TQ' = T \oplus Q$$



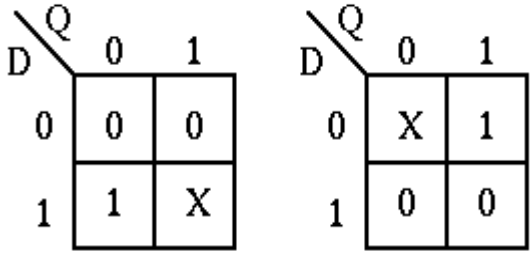
Convert a RS-FF to a D-FF:

We need to design the circuit to generate the triggering signals S and R as functions of and consider the excitation table:

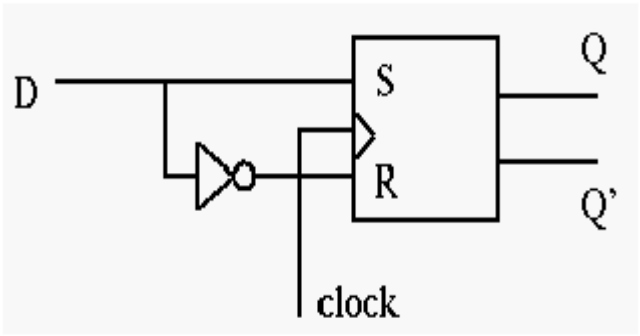


Q_t	Q_{t+1}	D	S	R
0	0	0	0	x
0	1	1	1	0
1	0	0	0	1
1	1	1	x	0

The desired signal and can be obtained as functions of and current FF state from the Karnaugh maps:



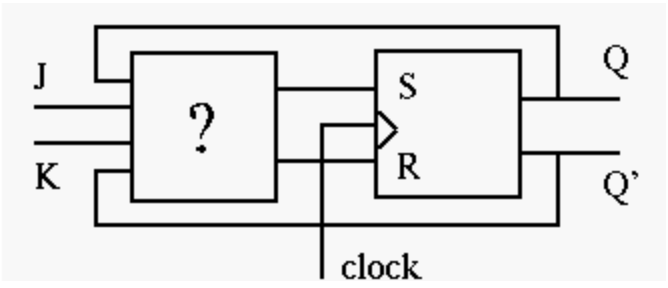
$S = D, \quad R = D'$



Convert a RS-FF to a JK-FF:

We need to design the circuit to generate the triggering signals S and R as functions of, J, K.

Consider the excitation table**:** The desired signal and as functions of, and current FF state can be obtained from the Karnaugh maps:



Q_t	Q_{t+1}	J	K	S	R
0	0	0	x	0	x
0	1	1	x	1	0
1	0	x	1	0	1
1	1	x	0	x	0

K-maps:

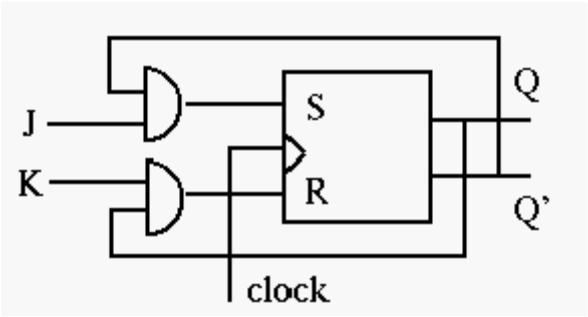
K \ QJ	00	01	11	10
0	0	1	X	X
1	0	1	0	0

S=Q'J

K \ QJ	00	01	11	10
0	X	0	0	0
1	X	0	1	1

R=QK

$S = Q'J, \quad R = QK$



The Master-Slave JK Flip-flop:

The Master-Slave Flip-Flop is basically two gated SR flip-flops connected together in a series configuration with the slave having an inverted clock pulse. The outputs from Q and Q from the "Slave" flip-flop are fed back to the inputs of the "Master" with the outputs of the "Master" flip-flop being connected to the two inputs of the "Slave" flip-flop. This feedback configuration from the slave's output to the master's input gives the characteristic toggle of the JK flip-flop as shown below.

The input signals J and K are connected to the gated "master" SR flip-flop which "locks" the input condition while the clock (Clk) input is "HIGH" at logic level "1". As the clock input of the "slave" flip-flop is the inverse (complement) of the "master" clock input, the "slave" SR flip-flop does not toggle. The outputs from the "master" flip-flop are only "seen" by the gated "slave" flip-flop when the clock input goes "LOW" to logic level "0". When the clock is "LOW", the outputs from the "master" flip-flop are latched and any additional changes to its inputs are ignored. The gated "slave" flip-flop now responds to the state of its inputs passed over by the "master" section. Then on the "Low-to-High" transition of the clock pulse the inputs of the "master" flip-flop are fed through to the gated inputs of the "slave" flip-flop and on the "High-to-Low" transition the same inputs are reflected on the output of the "slave" making this type of flip-flop edge or pulse-triggered. Then, the circuit accepts input data when the clock signal is "HIGH", and passes the data to the output on the falling-edge of the clock signal. In other words, the Master-Slave JK Flip-flop is a "Synchronous" device as it only passes data with the timing of the clock signal.

UNIT 4
Sequential circuit design and analysis

Sequential Circuit Design

- Steps in the design process for sequential circuits
- State Diagrams and State Tables
- Examples

- Steps in Design of a Sequential Circuit

1. Specification – A description of the sequential circuit. Should include a detailing of the inputs, the outputs, and the operation. Possibly assumes that you have knowledge of digital system basics.
2. Formulation: Generate a state diagram and/or a state table from the statement of the problem.
3. State Assignment: From a state table assign binary codes to the states.
4. Flip-flop Input Equation Generation: Select the type of flip-flop for the circuit and generate the needed input for the required state transitions
5. Output Equation Generation: Derive output logic equations for generation of the output from the inputs and current state.
6. Optimization: Optimize the input and output equations. Today, CAD systems are typically used for this in real systems.
7. Technology Mapping: Generate a logic diagram of the circuit using ANDs, ORs, Inverters, and F/Fs.
8. Verification: Use a HDL to verify the design.

Mealy and Moore

- Sequential machines are typically classified as either a Mealy machine or a Moore machine implementation.
- Moore machine: The outputs of the circuit depend only upon the current state of the circuit.
- Mealy machine: The outputs of the circuit depend upon both the current state of the circuit and the inputs.

An example to go through the steps

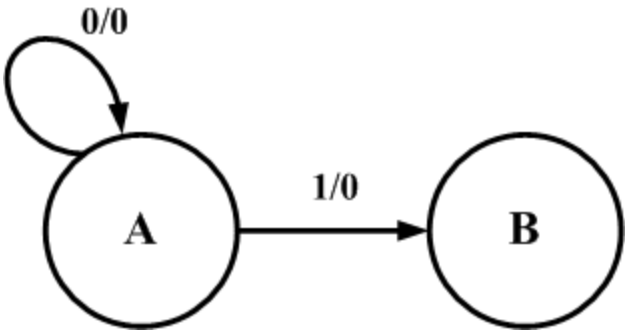
The specification: The circuit will have one input, X, and one output, Z. The output Z will be 0 except when the input sequence 1101 are the last 4 inputs received on X. In that case it will be a 1

Generation of a state diagram

- Create states and meaning for them.

State A – the last input was a 0 and previous inputs unknown. Can also be the reset state.
State B – the last input was a 1 and the previous input was a 0. The start of a new sequence possibly.

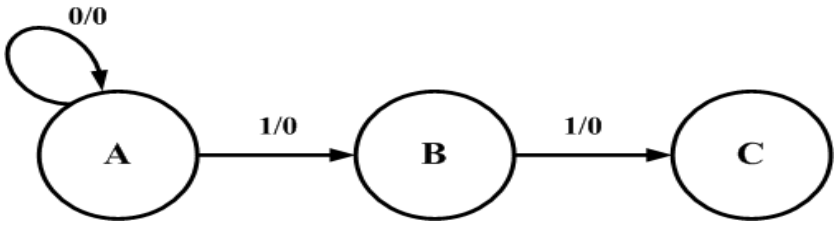
- Capture this in a state diagram



- Capture this in a state diagram

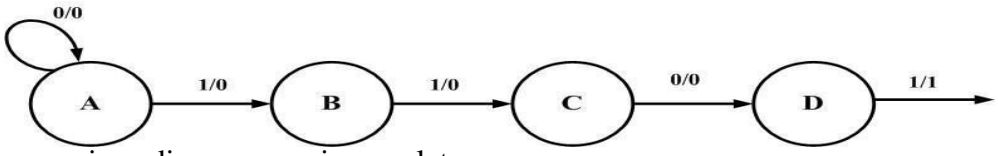
- Circles represent the states
- Lines and arcs represent the transition between states.
- The notation Input/output on the line or arc specifies the input that causes this transition and the output for this change of state.

- Add a state C – Have detected the input sequence 11 which is the start of the sequence



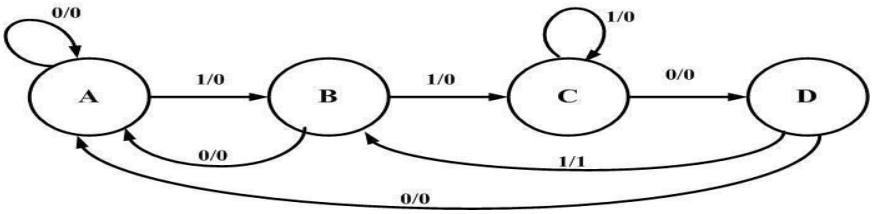
- Add a state D

  State D – have detected the 3rd input in the start of a sequence, a 0, now having 110. From State D, if the next input is a 1 the sequence has been detected and a 1 is output.



- The previous diagram was incomplete.

- In each state the next input could be a 0 or a 1. This must be included



- The state table
- This can be done directly from the state diagram

Prresent State	Next State X =0	Next State X=1	Output X=0	Output X=1
A	A	B	0	0
B	A	C	0	0
C	D	C	0	0
D	A	B	0	1

- Now need to do a state assignment

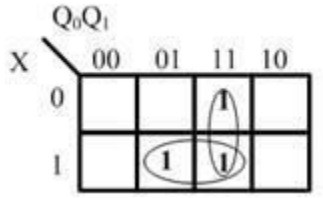
Select a state assignment

- Will select a gray encoding
- For this state A will be encoded 00, state B 01, state C 11 and state D 10

Prresent State	Next State X =0	Next State X=1	Output X=0	Output X=1
00	00	01	0	0
01	00	11	0	0
11	10	11	0	0
10	00	01	0	1

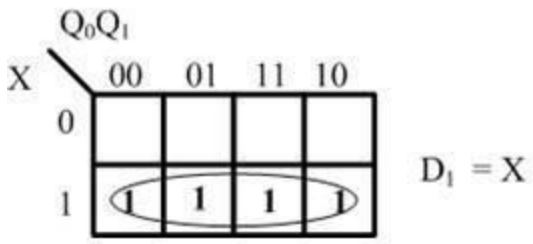
Flip-flop input equations

- Generate the equations for the flip-flop inputs
- Generate the D_0 equation



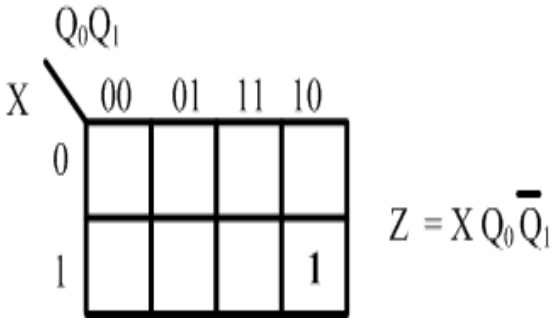
$D_0 = Q_0 Q_1 + X \overline{Q_1}$

- Generate the D_1 equation



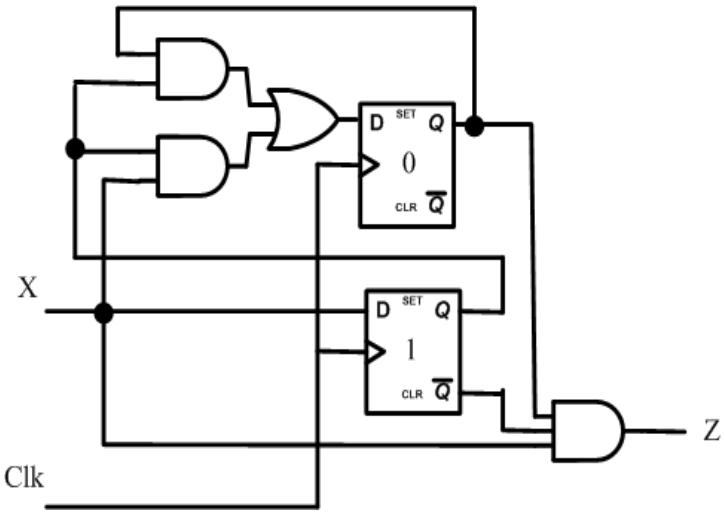
The output equation

- The next step is to generate the equation for the output Z and what is needed to generate it.
- Create a K-map from the truth table.



Now map to a circuit

- The circuit has 2 D type F/Fs



Shift registers:

In digital circuits, a **shift register** is a cascade of flip-flops sharing the same clock, in which the output of each flip-flop is connected to the "data" input of the next flip-flop in the chain, resulting in a circuit that shifts by one position the "bit array" stored in it, *shifting in* the data present at its input and *shifting out* the last bit in the array, at each transition of the clock input. More generally, a **shift register** may be multidimensional, such that its "data in" and stage outputs are themselves bit arrays: this is implemented simply by running several shift registers of the same bit-length in parallel.

Shift registers can have both parallel and serial inputs and outputs. These are often configured as **serial-in, parallel-out** (SIPO) or as **parallel-in, serial-out** (PISO). There are also types that have both serial and parallel input and types with serial and parallel output. There are also **bi-directional** shift registers which allow shifting in both directions: L→R or R→L. The serial input and last output of a shift register can also be connected to create a **circular shift register**

Shift registers are a type of logic circuits closely related to counters. They are basically for the storage and transfer of digital data.

Buffer register:

The buffer register is the simple set of registers. It is simply stores the binary word. The buffer may be controlled buffer. Most of the buffer registers used D Flip-flops.

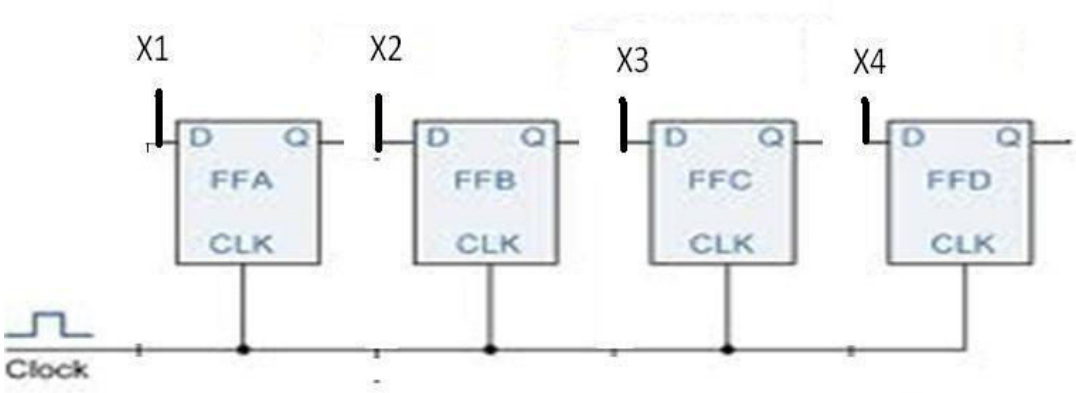


Figure: logic diagram of 4-bit buffer register

The figure shows a 4-bit buffer register. The binary word to be stored is applied to the data terminals. On the application of clock pulse, the output word becomes the same as the word applied at the terminals. i.e., the input word is loaded into the register by the application of clock pulse.

When the positive clock edge arrives, the stored word becomes:

$$Q_4Q_3Q_2Q_1=X_4X_3X_2X_1$$

$$Q=X$$

Controlled buffer register:

If goes LOW, all the FFs are RESET and the output becomes, Q=0000.

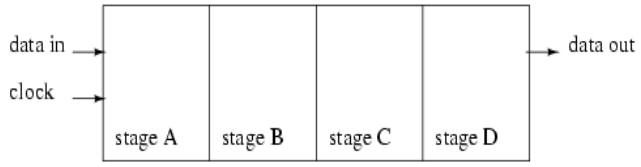
When is HIGH, the register is ready for action. LOAD is the control input. When LOAD is HIGH, the data bits X can reach the D inputs of FF‘s.

$$Q_4Q_3Q_2Q_1=X_4X_3X_2X_1$$

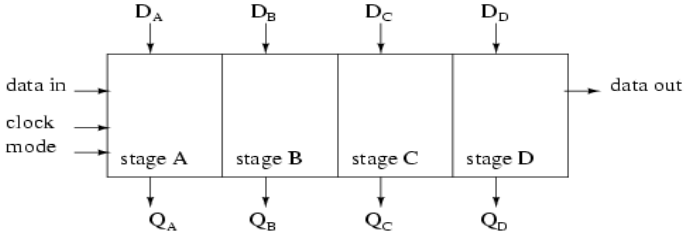
$$Q=X$$

When load is low, the X bits cannot reach the FF‘s.

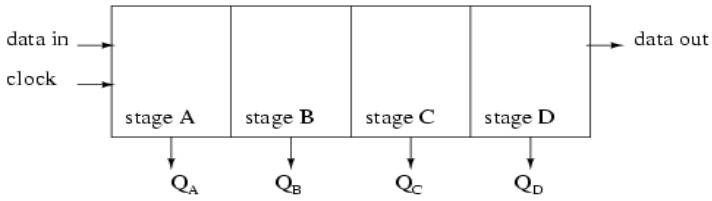
Data transmission in shift registers:



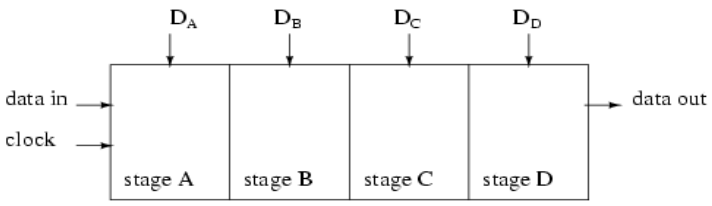
Serial-in, serial-out shift register with 4-stages



Parallel-in, parallel-out shift register with 4-stages



Serial-in, parallel-out shift register with 4-stages



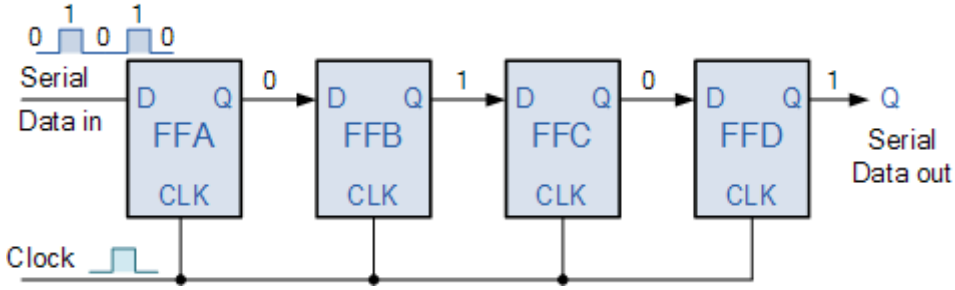
Parallel-in, serial-out shift register with 4-stages

A number of ff's connected together such that data may be shifted into and shifted out of them is called shift register. data may be shifted into or out of the register in serial form or in parallel form. There are four basic types of shift registers.

1. Serial in, serial out, shift right, shift registers
2. Serial in, serial out, shift left, shift registers
3. Parallel in, serial out shift registers
4. Parallel in, parallel out shift registers

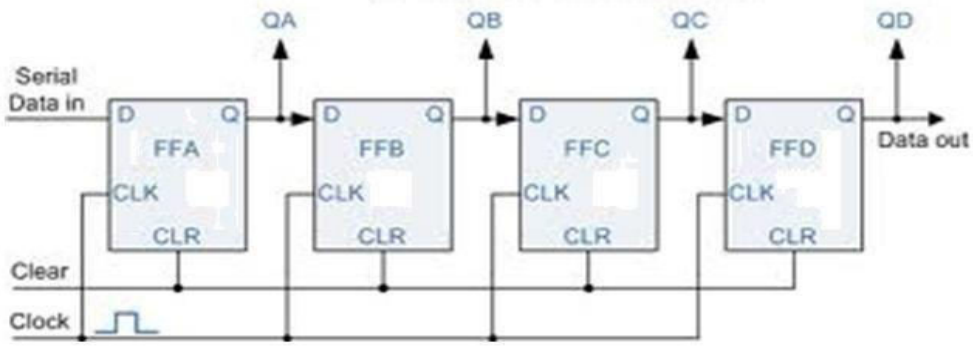
Serial IN, serial OUT, shift right, shift left register:

The logic diagram of 4-bit serial in serial out, right shift register with four stages. The register can store four bits of data. Serial data is applied at the input D of the first FF. the Q output of the first FF is connected to the D input of another FF. the data is outputted from the Q terminal of the last FF.



When serial data is transferred into a register, each new bit is clocked into the first FF at the positive going edge of each clock pulse. The bit that was previously stored by the first FF is transferred to the second FF. the bit that was stored by the Second FF is transferred to the third FF.

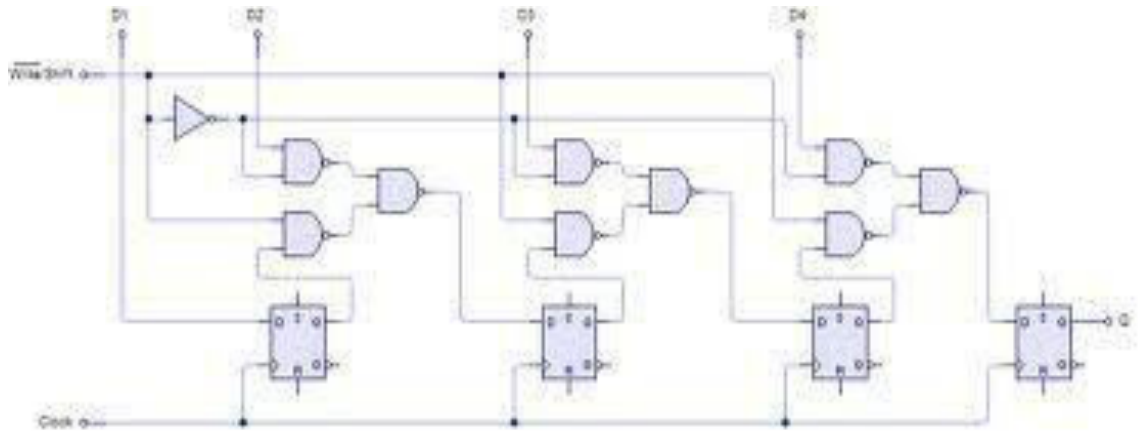
Serial-in, parallel-out, shift register:



In this type of register, the data bits are entered into the register serially, but the data stored in the register is shifted out in parallel form.

Once the data bits are stored, each bit appears on its respective output line and all bits are available simultaneously, rather than on a bit-by-bit basis with the serial output. The serial-in, parallel out, shift register can be used as serial-in, serial out, shift register if the output is taken from the Q terminal of the last FF.

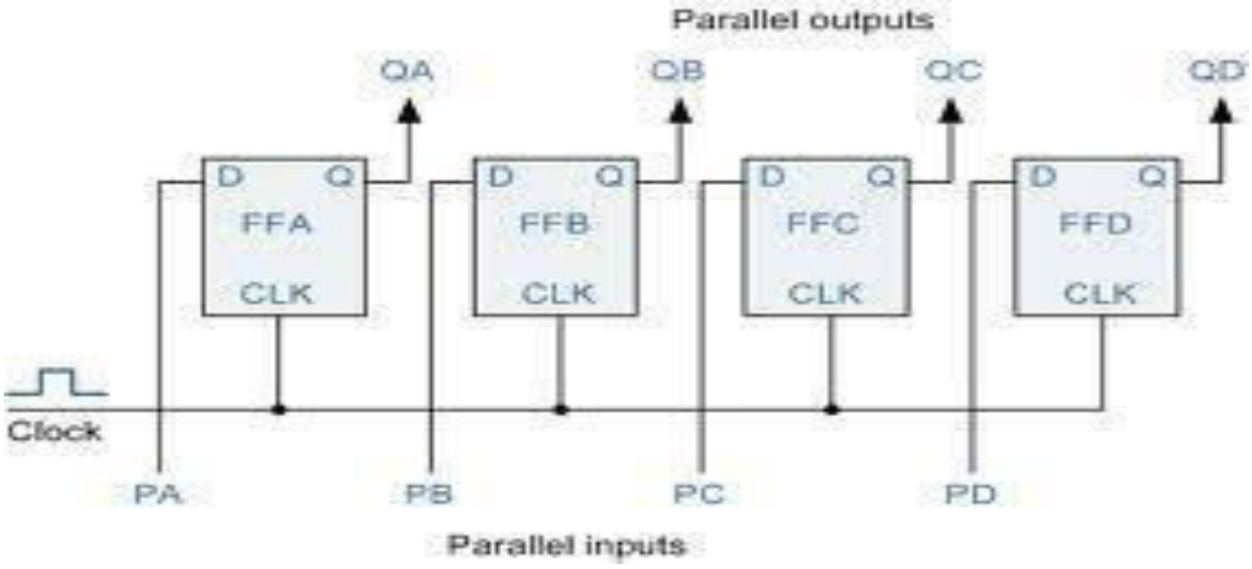
Parallel-in, serial-out, shift register:



For a parallel-in, serial out, shift register, the data bits are entered simultaneously into their respective stages on parallel lines, rather than on a bit-by-bit basis on one line as with serial data bits are transferred out of the register serially. On a bit-by-bit basis over a single line.

There are four data lines A,B,C,D through which the data is entered into the register in parallel form. The signal shift/ load allows the data to be entered in parallel form into the register and the data is shifted out serially from terminalQ4

Parallel-in, parallel-out, shift register



In a parallel-in, parallel-out shift register, the data is entered into the register in parallel form, and also the data is taken out of the register in parallel form. Data is applied to the D input terminals of the FF‘s. When a clock pulse is applied, at the positive going edge of the pulse, the D inputs are shifted into the Q outputs of the FFs. The register now stores the data. The stored data is available instantaneously for shifting out in parallel form.

Bidirectional shift register:

A bidirectional shift register is one which the data bits can be shifted from left to right or from right to left. A fig shows the logic diagram of a 4-bit serial-in, serial out, bidirectional shift register. Right/left is the mode signal, when right /left is a 1, the logic circuit works as a shift-register.the bidirectional operation is achieved by using the mode signal and two NAND gates and one OR gate for each stage.

A HIGH on the right/left control input enables the AND gates G1, G2, G3 and G4 and disables the AND gates G5,G6,G7 and G8, and the state of Q output of each FF is passed through the gate to the D input of the following FF. when a clock pulse occurs, the data bits are then effectively shifted one place to the right. A LOW on the right/left control inputs enables the AND gates G5, G6, G7 and G8 and disables the And gates G1, G2, G3 and G4 and the Q output of each FF is passed to the D input of the preceding FF. when a clock pulse occurs, the data bits are then effectively shifted one place to the left. Hence, the circuit works as a bidirectional shift register

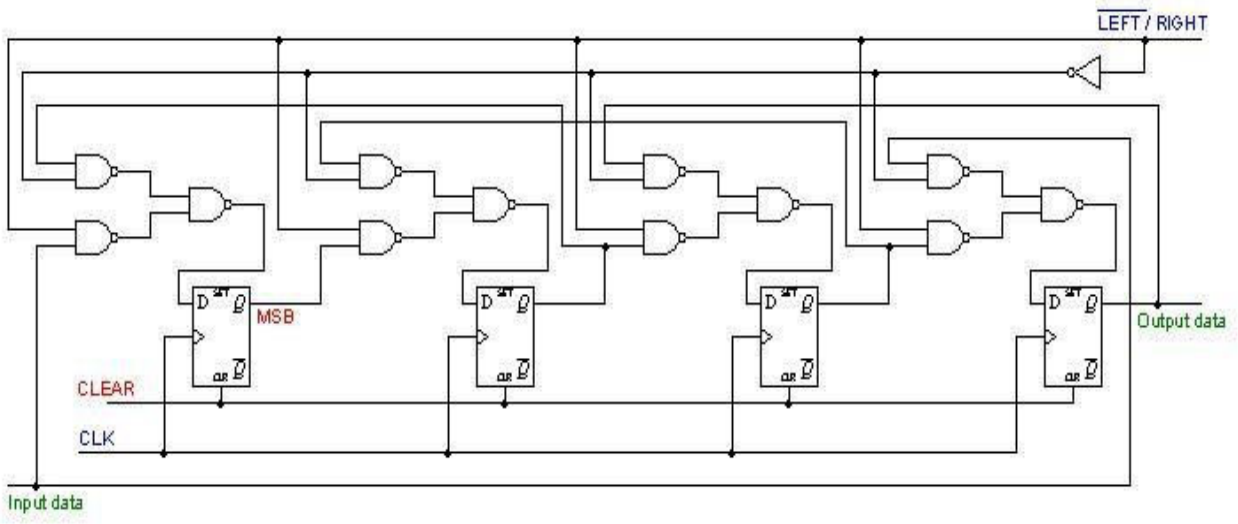


Figure: logic diagram of a 4-bit bidirectional shift register

Universal shift register:

A register is capable of shifting in one direction only is a unidirectional shift register. One that can shift both directions is a bidirectional shift register. If the register has both shifts and parallel load capabilities, it is referred to as a universal shift registers. Universal shift register is a bidirectional register, whose input can be either in serial form or in parallel form and whose output also can be in serial form or I parallel form.

The most general shift register has the following capabilities.

1. A clear control to clear the register to 0
2. A clock input to synchronize the operations
3. A shift-right control to enable the shift-right operation and serial input and output lines associated with the shift-right

4. A shift-left control to enable the shift-left operation and serial input and output lines associated with the shift-left
5. A parallel loads control to enable a parallel transfer and the n input lines associated with the parallel transfer
6. N parallel output lines
7. A control state that leaves the information in the register unchanged in the presence of the clock.

A universal shift register can be realized using multiplexers. The below fig shows the logic diagram of a 4-bit universal shift register that has all capabilities. It consists of 4 D flip-flops and four multiplexers. The four multiplexers have two common selection inputs s1 and s0. Input 0 in each multiplexer is selected when S1S0=00, input 1 is selected when S1S0=01 and input 2 is selected when S1S0=10 and input 4 is selected when S1S0=11. The selection inputs control the mode of operation of the register according to the functions entries. When S1S0=0, the present value of the register is applied to the D inputs of flip-flops. The condition forms a path from the output of each flip-flop into the input of the same flip-flop. The next clock edge transfers into each flip-flop the binary value it held previously, and no change of state occurs. When S1S0=01, terminal 1 of the multiplexer inputs have a path to the D inputs of the flip-flop. This causes a shift-right operation, with serial input transferred into flip-flopA4. When S1S0=10, a shift left operation results with the other serial input going into flip-flop A1. Finally when S1S0=11, the binary information on the parallel input lines is transferred into the register simultaneously during the next clock cycle

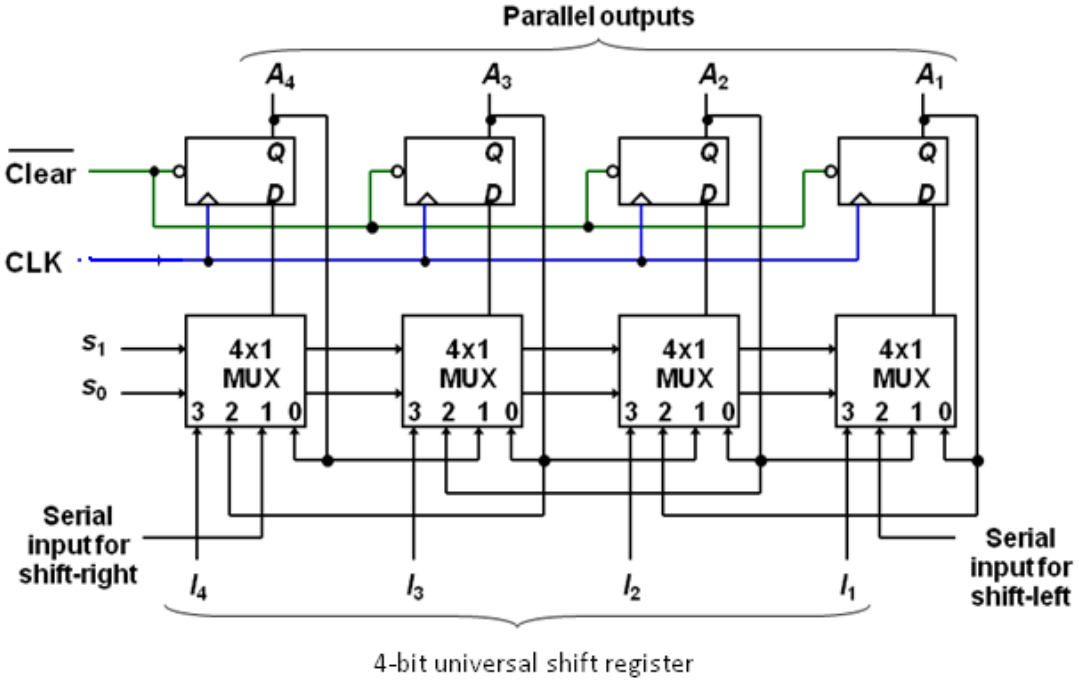


Figure: logic diagram 4-bit universal shift register

Function table for theregister

mode control		
S0	**S1**	**register operation**
0	0	No change
0	1	Shift Right
1	0	Shift left
1	1	Parallel load

Counters:

Counter is a device which stores (and sometimes displays) the number of times particular event or process has occurred, often in relationship to a clock signal. A Digital counter is a set of flip flops whose state change in response to pulses applied at the input to the counter. Counters may be asynchronous counters or synchronous counters. Asynchronous counters are also called ripple counters

In electronics counters can be implemented quite easily using register-type circuits such as the flip-flops and a wide variety of classifications exist:

- Asynchronous (ripple) counter – changing state bits are used as clocks to subsequent state flip-flops
- Synchronous counter – all state bits change under control of a singleclock
- Decade counter – counts through ten states per stage
- Up/down counter – counts both up and down, under command of a control input
- Ring counter – formed by a shift register with feedback connection in a ring
- Johnson counter – a *twisted* ring counter
- Cascaded counter
- Modulus counter.

Each is useful for different applications. Usually, counter circuits are digital in nature, and count in natural binary Many types of counter circuits are available as digital building blocks, for example a number of chips in the 4000 series implement different counters.

Occasionally there are advantages to using a counting sequence other than the natural binary sequence such as the binary coded decimal counter, a linear feed-back shift register counter, or a gray-code counter.

Counters are useful for digital clocks and timers, and in oven timers, VCR clocks, etc.

Asynchronous counters:

An asynchronous (ripple) counter is a single JK-type flip-flop, with its J (data) input fed from its own inverted output. This circuit can store one bit, and hence can count from zero to one before it overflows (starts over from 0). This counter will increment once for every clock cycle and takes two clock cycles to overflow, so every cycle it will alternate between a transition from 0 to 1 and a transition from 1 to 0. Notice that this creates a new clock with a 50% duty cycle at exactly half the frequency of the input clock. If this output is then used as the clock signal for a similarly arranged D flip-flop (remembering to invert the output to the input), one will get another 1 bit counter that counts half as fast. Putting them together yields a two-bit counter:

Two-bit ripple up-counter using negative edge triggered flip flop:

Two bit ripple counter used two flip-flops. There are four possible states from 2 – bit up-counting I.e. 00, 01, 10 and 11.

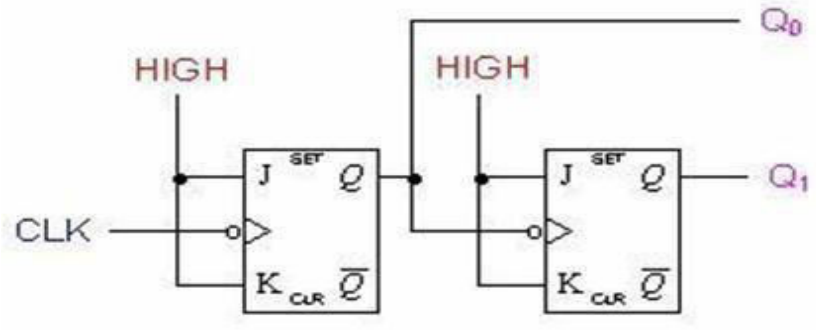
· The counter is initially assumed to be at a state 00 where the outputs of the tow flip-flops are noted as Q_1Q_0. Where Q_1 forms the MSB and Q_0 forms the LSB.

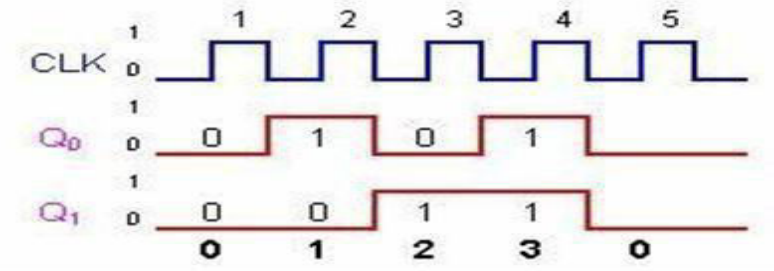
· For the negative edge of the first clock pulse, output of the first flip-flop FF_1 toggles its state. Thus Q_1 remains at 0 and Q_0 toggles to 1 and the counter state are now read as 01.

· During the next negative edge of the input clock pulse FF_1 toggles and $Q_0 = 0$. The output Q0 being a clock signal for the second flip-flop FF_2 and the present transition acts as a negative edge for FF_2 thus toggles its state $Q_1 = 1$. The counter state is now read as 10.

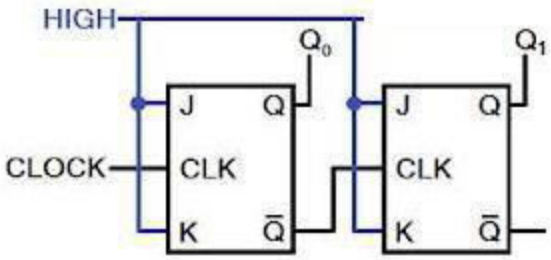
· For the next negative edge of the input clock to FF_1 output Q0 toggles to 1. But this transition from 0 to 1 being a positive edge for FF_2 output Q_1 remains at 1. The counter state is now read as 11.

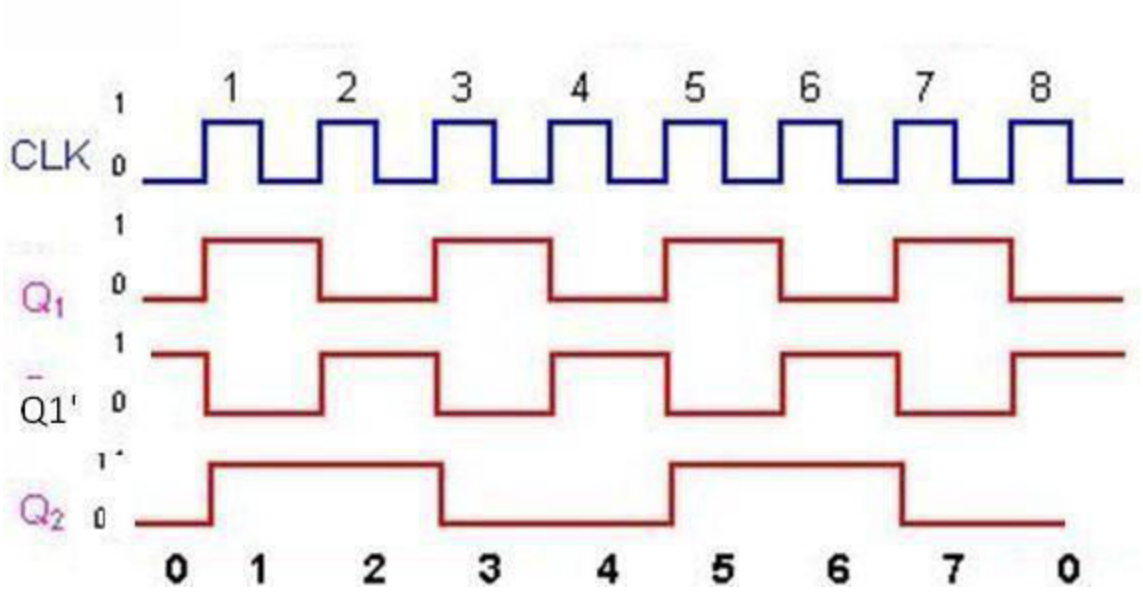
· For the next negative edge of the input clock, Q_0 toggles to 0. This transition from 1 to 0 acts as a negative edge clock for FF_2 and its output Q_1 toggles to 0. Thus the starting state 00 is attained. Figure shown below





Two-bit ripple down-counter using negative edge triggered flip flop:





A 2-bit down-counter counts in the order 0,3,2,1,0,1……,i.e, 00,11,10,01,00,11 …..,etc. the above fig. shows ripple down counter, using negative edge triggered J-K FFs and its timing diagram.

- For down counting, Q1‘ of FF1 is connected to the clock of Ff2. Let initially all the FF1 toggles, so, Q1 goes from a 0 to a 1 and Q1‘ goes from a 1 to a 0.

- The negative-going signal at Q1‘ is applied to the clock input of FF2, toggles Ff2 and, therefore, Q2 goes from a 0 to a 1.so, after one clock pulse Q2=1 and Q1=1, I.e., the state of the counter is 11.
- At the negative-going edge of the second clock pulse, Q1 changes from a 1 to a 0 and Q1‘ from a 0 to a 1.
- This positive-going signal at Q1‘ does not affect FF2 and, therefore, Q2 remains at a 1. Hence , the state of the counter after second clock pulse is 10
- At the negative going edge of the third clock pulse, FF1 toggles. So Q1, goes from a 0 to a 1 and Q1‘ from 1 to 0. This negative going signal at Q1‘ toggles FF2 and, so, Q2 changes from 1 to 0, hence, the state of the counter after the third clock pulse is 01.
- At the negative going edge of the fourth clock pulse, FF1 toggles. So Q1, goes from a 1 to a 0 and Q1‘ from 0 to 1. . This positive going signal at Q1‘ does not affect FF2 and, so, Q2 remains at 0, hence, the state of the counter after the fourth clock pulse is 00.

Two-bit ripple up-down counter using negative edge triggered flip flop:

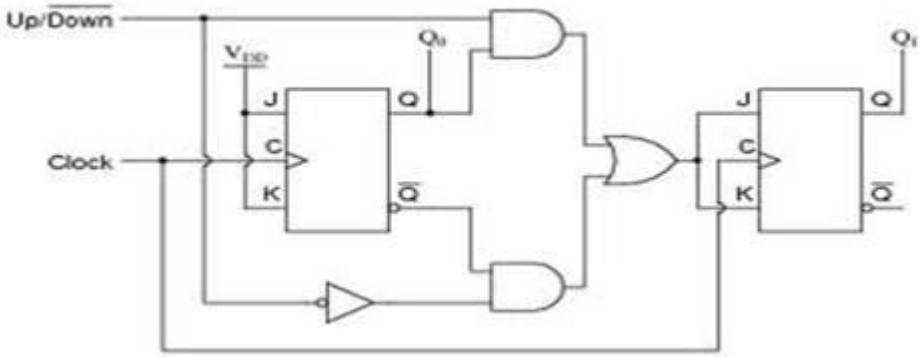


Figure: asynchronous 2-bit ripple up-down counter using negative edge triggered flip flop:

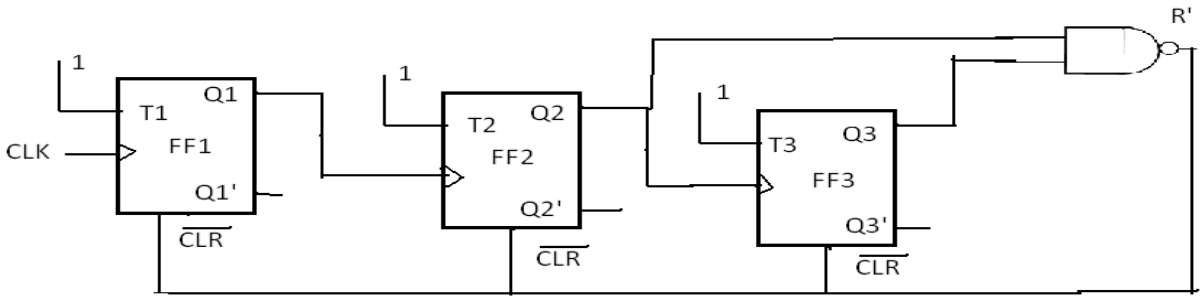
- As the name indicates an up-down counter is a counter which can count both in upward and downward directions. An up-down counter is also called a forward/backward counter or a bidirectional counter. So, a control signal or a mode signal M is required to choose the direction of count. When M=1 for up counting, Q1 is transmitted to clock of FF2 and when M=0 for down counting, Q1‘ is transmitted to clock of FF2. This is achieved by using two AND gates and one OR gates. The external clock signal is applied toFF1.
- Clock signal to FF2= (Q1.Up)+(Q1‘. Down)=Q1m+Q1‘M‘

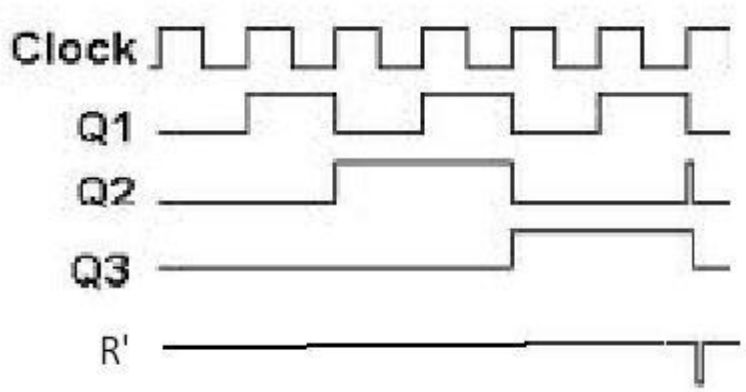
Design of Asynchronous counters:

To design a asynchronous counter, first we write the sequence , then tabulate the values of reset signal R for various states of the counter and obtain the minimal expression for R and R‘ using K-Map or any other method. Provide a feedback such that R and R‘ resets all the FF‘s after the desired count

Design of a Mod-6 asynchronous counter using T FFs:

A mod-6 counter has six stable states 000, 001, 010, 011, 100, and 101. When the sixth clock pulse is applied, the counter temporarily goes to 110 state, but immediately resets to 000 because of the feedback provided. it is ―divide by-6-counter‖, in the sense that it divides the input clock frequency by 6.it requires three FFs, because the smallest value of n satisfying the condition$N \leq 2^n$ is n=3; three FFs can have 8 possible states, out of which only six are utilized and the remaining two states 110and 111, are invalid. If initially the counter is in 000 state, then after the sixth clock pulse, it goes to 001, after the second clock pulse, it goes to 010, and so on.





After sixth clock pulse it goes to 000. For the design, write the truth table with present state outputs Q3, Q2 and Q1 as the variables, and reset R as the output and obtain an expression for R in terms of Q3, Q2, and Q1that decides the feedback into be provided. From the truth table, R=Q3Q2. For active-low Reset, R‘ is used. The reset pulse is of very short duration, of the order of nanoseconds and it is equal to the propagation delay time of the NAND gate used. The expression for R can also be determined as follows.

R=0 for 000 to 101, R=1 for 110, and R=X=for111

Therefore,

R=Q3Q2Q1‘+Q3Q2Q1=Q3Q2

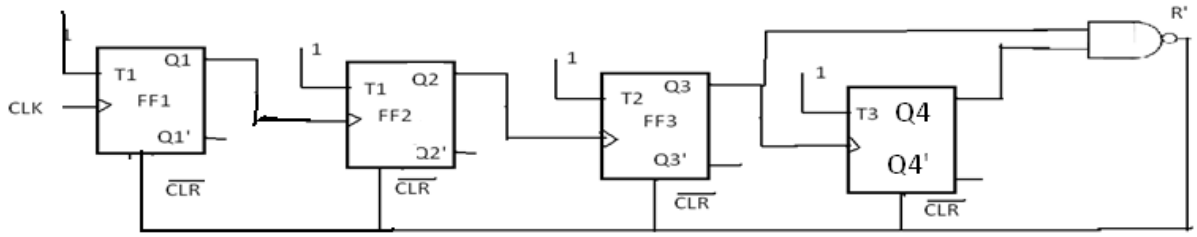
The logic diagram and timing diagram of Mod-6 counter is shown in the above fig.

The truth table is as shown in below.

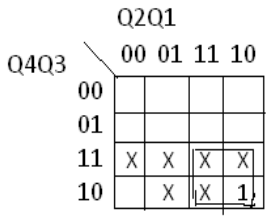
After pulses	States Q3	Q2	Q1	R
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
	↓	↓	↓	
	0	0	0	0
7	0	0	0	0

Design of a mod-10 asynchronous counter using T-flip-flops:

A mod-10 counter is a decade counter. It also called a BCD counter or a divide-by-10 counter. It requires four flip-flops (condition $10 \leq 2^n$ is n=4). So, there are 16 possible states, out of which ten are valid and remaining six are invalid. The counter has ten stable state, 0000 through 1001, i.e., it counts from 0 to 9. The initial state is 0000 and after nine clock pulses it goes to 1001. When the tenth clock pulse is applied, the counter goes to state 1010 temporarily, but because of the feedback provided, it resets to initial state 0000. So, there will be a glitch in the waveform of Q2. The state 1010 is a temporary state for which the reset signal R=1, R=0 for 0000 to 1001, and R=C for 1011 to 1111.



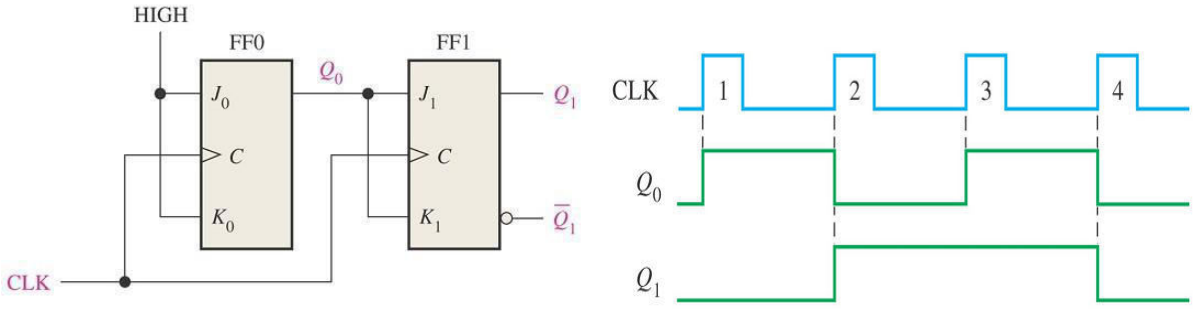
The count table and the K-Map for reset are shown in fig. from the K-Map R=Q4Q2. So, feedback is provided from second and fourth FFs. For active –HIGH reset, Q4Q2 is applied to the clear terminal. For active-LOW reset 4 2 is connected isof all Flip=flops.



After pulses	Count Q4	Q3	Q2	Q1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	0	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	0	1	0	1
10	0	0	0	0

Synchronous counters:

Asynchronous counters are serial counters. They are slow because each FF can change state only if all the preceding FFs have changed their state. if the clock frequency is very high, the asynchronous counter may skip some of the states. This problem is overcome in synchronous counters or parallel counters. Synchronous counters are counters in which all the flip flops are triggered simultaneously by the clock pulses Synchronous counters have a common clock pulse applied simultaneously to all flip-flops. A 2-Bit Synchronous Binary Counter



Design of synchronous counters:

For a systematic design of synchronous counters. The following procedure is used.

Step 1: State Diagram: draw the state diagram showing all the possible states state diagram which also be called nth transition diagrams, is a graphical means of depicting the sequence of states through which the counter progresses.

Step2: number of flip-flops: based on the description of the problem, determine the required number n of the flip-flops- the smallest value of n is such that the number of states $N \leq 2^n$--- and the desired counting sequence.

Step3: choice of flip-flops excitation table: select the type of flip-flop to be used and write the excitation table. An excitation table is a table that lists the present state (ps) , the next state(ns) and required excitations.

Step4: minimal expressions for excitations: obtain the minimal expressions for the excitations of the FF using K-maps drawn for the excitation of the flip-flops in terms of the present states and inputs.

Step5: logic diagram: draw a logic diagram based on the minimal expressions

Design of a synchronous 3-bit up-down counter using JK flip-flops:

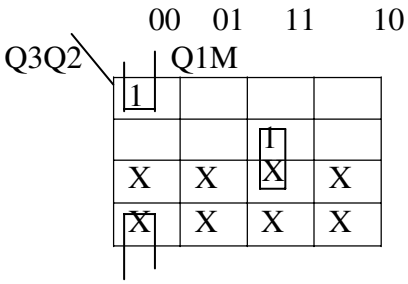
Step1: determine the number of flip-flops required. A 3-bit counter requires three FFs. It has 8 states (000,001,010,011,101,110,111) and all the states are valid. Hence no don‘t cares. For selecting up and down modes, a control or mode signal M is required. When the mode signal M=1 and counts down when M=0. The clock signal is applied to all the FFs simultaneously.

Step2: draw the state diagrams: the state diagram of the 3-bit up-down counter is drawn as

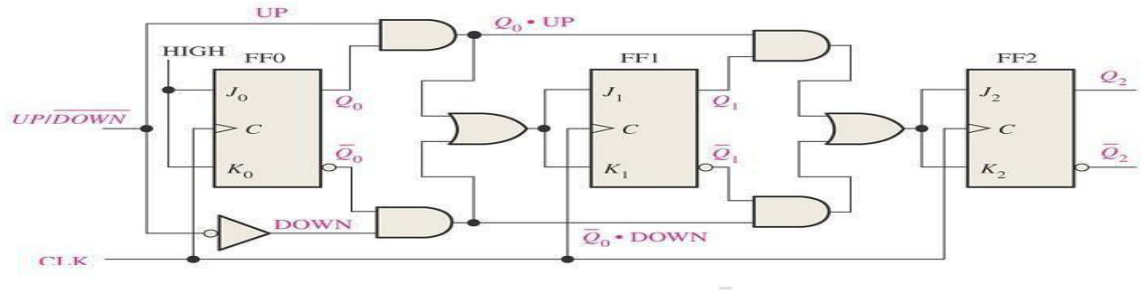
Step3: select the type of flip flop and draw the excitation table: JK flip-flops are selected and the excitation table of a 3-bit up-down counter using JK flip-flops is drawn as shown in fig.

PS			mode	NS			required excitations					
Q3	Q2	Q1	M	Q3	Q2	Q1	J3	K3	J2	K2	J1	K1
0	0	0	0	1	1	1	1	x	1	x	1	x
0	0	0	1	0	0	1	0	x	0	x	1	x
0	0	1	0	0	0	0	0	x	0	x	x	1
0	0	1	1	0	1	0	0	x	1	x	x	1
0	1	0	0	0	0	1	0	x	x	1	1	x
0	1	0	1	0	1	1	0	x	x	0	1	x
0	1	1	0	0	1	0	0	x	x	0	x	1
0	1	1	1	1	0	0	1	x	x	1	x	1
1	0	0	0	0	1	1	x	1	1	x	1	x
1	0	0	1	1	0	1	x	0	0	x	1	x
1	0	1	0	1	0	0	x	0	0	x	x	1
1	0	1	1	1	1	0	x	0	1	x	x	1
1	1	0	0	1	0	1	x	0	x	1	1	x
1	1	0	1	1	1	1	x	0	x	0	1	x
1	1	1	0	1	1	0	x	0	x	0	x	1
1	1	1	1	0	0	0	x	1	x	1	x	1

Step4: obtain the minimal expressions: From the excitation table we can conclude that J1=1 and K1=1, because all the entries for J1and K1 are either X or 1. The K-maps for J3, K3,J2 and K2 based on the excitation table and the minimal expression obtained from them are shown in fig.

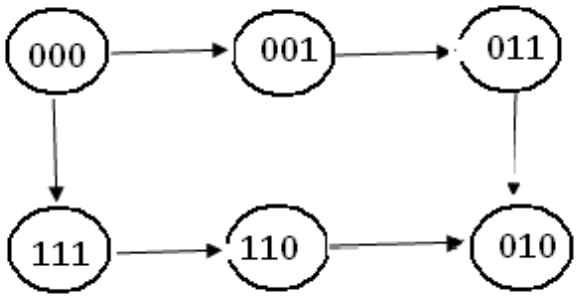


Step5: draw the logic diagram: a logic diagram using those minimal expressions can be drawn as shown in fig.



Design of a synchronous modulo-6 gray cod counter:

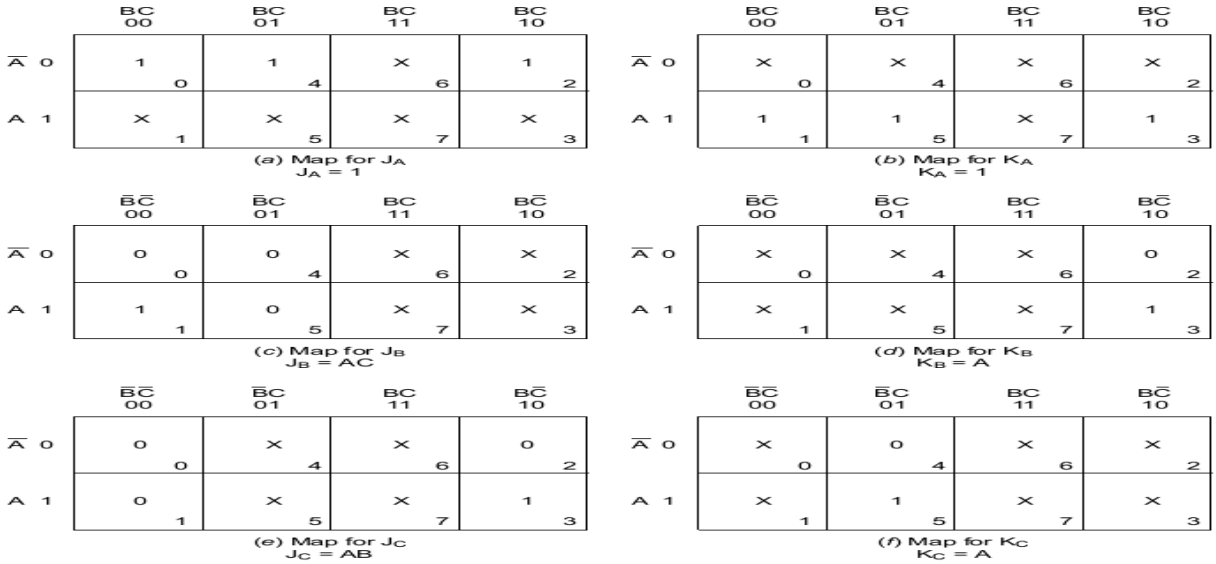
Step 1: the number of flip-flops: we know that the counting sequence for a modulo-6 gray code counter is 000, 001, 011, 010, 110, and 111. It requires n=3FFs ($N \leq 2^n$, i.e., $6 \leq 2^3$). 3 FFs can have 8 states. So the remaining two states 101 and 100 are invalid. The entries for excitation corresponding to invalid states are don‘t cares.
Step2: the state diagram: the state diagram of the mod-6 gray code converter is drawn as shown in fig.



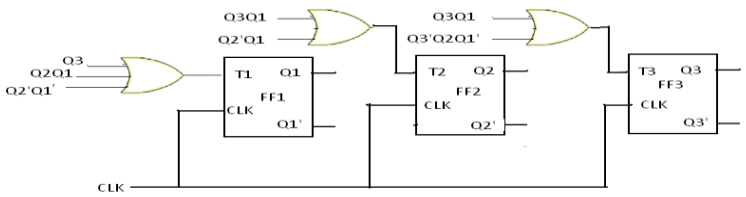
Step3: type of flip-flop and the excitation table: T flip-flops are selected and the excitation table of the mod-6 gray code counter using T-flip-flops is written as shown in fig.

PS			**NS**			**required excitations**		
Q3	**Q2**	**Q1**	**Q3**	**Q2**	**Q1**	**T3**	**T2**	**T1**
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	0
0	1	1	0	1	0	0	0	1
0	1	0	1	1	0	1	0	0
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

Step4: The minimal expressions: the K-maps for excitations of FFs T3,T2,and T1 in terms of outputs of FFs Q3,Q2, and Q1, their minimization and the minimal expressions for excitations obtained from them are shown if fig



Step5: the logic diagram: the logic diagram based on those minimal expressions is drawn as shown in fig.



Design of a synchronous BCD Up-Down counter using FFs:

Step1: the number of flip-flops: a BCD counter is a mod-10 counter has 10 states (0000 through 1001) and so it requires n=4FFs($N \leq 2^n$,, i.e., $10 \leq 2^4$). 4 FFS can have 16 states. So out of 16 states, six states (1010 through 1111) are invalid. For selecting up and down mode, a control or mode signal M is required. , it counts up when M=1 and counts down when M=0. The clock signal is applied to all FFs.

Step2: the state diagram: The state diagram of the mod-10 up-down counter is drawn as shown in fig.

Step3: types of flip-flops and excitation table: T flip-flops are selected and the excitation table of the modulo-10 up down counter using T flip-flops is drawn as shown in fig.

The remaining minterms are don't cares($\sum$d(20,21,22,23,24,25,26,37,28,29,30,31)) from the excitation table we can see that T1=1 and the expression for T4,T3,T2 are asfollows.

T4=$\sum$m(0,15,16,19)+d(20,21,22,23,24,25,26,27,28,29,30,31)

T3=$\sum$m(7,15,16,8)+d(20,21,22,23,24,25,26,27,28,29,30,31)

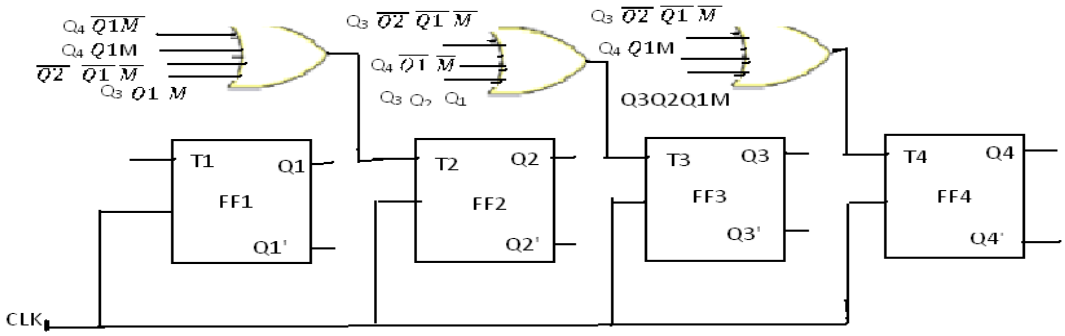
T2=$\sum$m(3,4,7,8,11,12,15,16)+d(20,21,22,23,24,25,26,27,28,29,30,31)

PS				**mode**	**NS**				**required excitations**			
Q4	**Q3**	**Q2**	**Q1**	**M**	**Q4**	**Q3**	**Q2**	**Q1**	**T4**	**T3**	**T2**	**T1**
0	0	0	0	0	1	0	0	1	1	0	0	1
0	0	0	0	1	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	0	0	1	0	0	1	1
0	0	1	0	1	0	0	1	1	0	0	0	1
0	0	1	1	0	0	0	1	0	0	0	0	1
0	0	1	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	0	0	1	1	0	1	1	1
0	1	0	0	1	0	1	0	1	0	0	0	1
0	1	0	1	0	0	1	0	0	0	0	0	1
0	1	0	1	1	0	1	1	0	0	0	1	1
0	1	1	0	0	0	1	0	1	0	0	1	1
0	1	1	0	1	0	1	1	1	0	0	0	1
0	1	1	1	0	0	1	1	0	0	0	0	1
0	1	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	1	1	0	0	1	0	0	0	1
1	0	0	1	0	1	0	0	0	0	0	0	1
1	0	0	1	1	0	0	0	0	1	0	0	1

Step4: The minimal expression: since there are 4 state variables and a mode signal, we require 5 variable kmaps. 20 conditions of Q4Q3Q2Q1M are valid and the remaining 12 combinations are invalid. So the entries for excitations corresponding to those invalid combinations are don‘t cares. Minimizing K-maps for T2 we get

T 2= Q4Q1‘M+Q4‘Q1M+Q2Q1‘M‘+Q3Q1‘M‘

Step5: the logic diagram: the logic diagram based on the above equation is shown in fig.



Shift register counters:

One of the applications of shift register is that they can be arranged to form several types of counters. The most widely used shift register counter is ring counter as well as the twisted ring counter.

Ring counter: this is the simplest shift register counter. The basic ring counter using D flip-flops is shown in fig. the realization of this counter using JK FFs. The Q output of each stage is connected to the D flip-flop connected back to the ring counter.

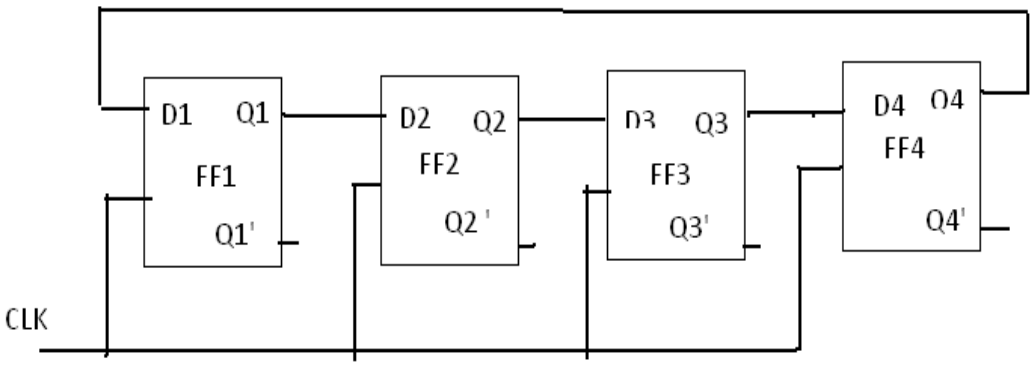
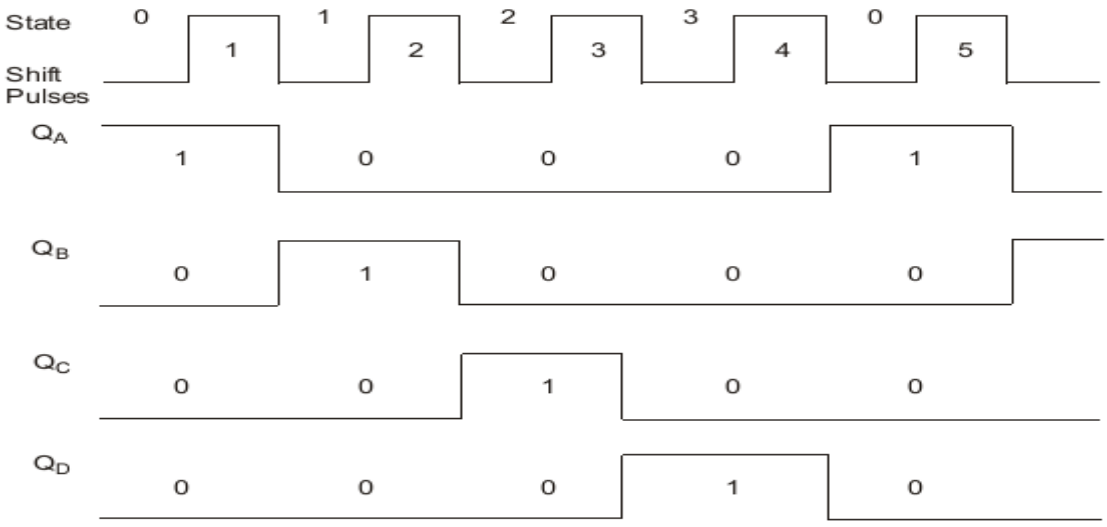


FIGURE: logic diagram of 4-bit ring counter using D flip-flops

Only a single 1 is in the register and is made to circulate around the register as long as clock pulses are applied. Initially the first FF is present to a 1. So, the initial state is 1000, i.e., Q1=1, Q2=0,Q3=0,Q4=0. After each clock pulse, the contents of the register are shifted to the right by one bit and Q4 is shifted back to Q1. The sequence repeats after four clock pulses. The number

of distinct states in the ring counter, i.e., the mod of the ring counter is equal to number of FFs used in the counter. An n-bit ring counter can count only n bits, where as n-bit ripple counter can count 2^n bits. So, the ring counter is uneconomical compared to a ripple counter but has advantage of requiring no decoder, since we can read the count by simply noting which FF is set. Since it is entirely a synchronous operation and requires no gates external FFs, it has the further advantage of being very fast.

Timing diagram:



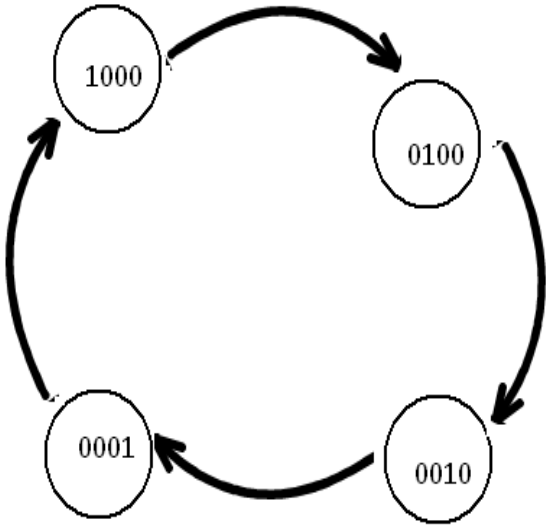


Figure: state diagram

Twisted Ring counter (Johnson counter):

This counter is obtained from a serial-in, serial-out shift register by providing feedback from the inverted output of the last FF to the D input of the first FF. the Q output of each is connected to the D input of the next stage, but the Q‘ output of the last stage is connected to the D input of the first stage, therefore, the name twisted ring counter. This feedback arrangement produces a unique sequence of states.

The logic diagram of a 4-bit Johnson counter using D FF is shown in fig. the realization of the same using J-K FFs is shown in fig.. The state diagram and the sequence table are shown in figure. The timing diagram of a Johnson counter is shown infigure.

Let initially all the FFs be reset, i.e., the state of the counter be 0000. After each clock pulse, the level of Q1 is shifted to Q2, the level of Q2to Q3, Q3 to Q4 and the level of Q4‘to Q1 and the sequences given in fig.

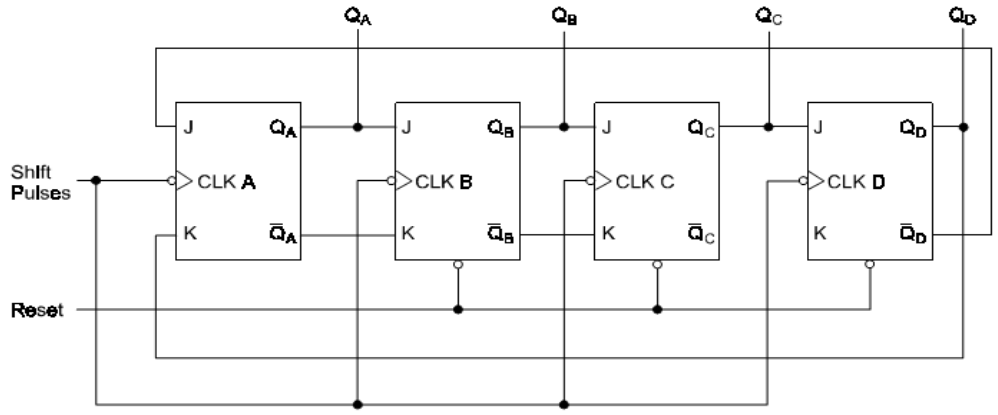


Figure: Johnson counter with JK flip-flops

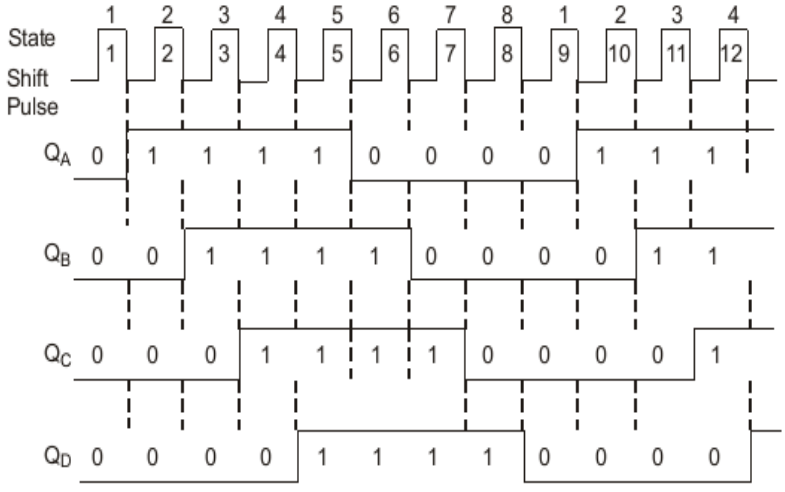
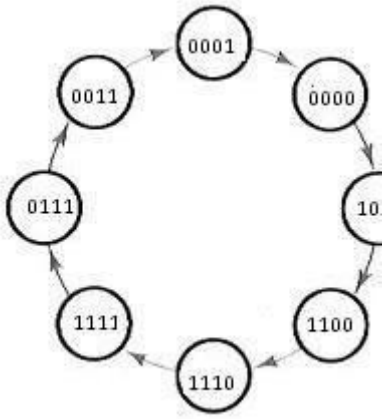


Figure: timing diagram

State diagram:



Q1	Q2	Q3	Q4	after clock pulse
0	0	0	0	0
1	0	0	0	1
1	1	0	0	2
1	1	1	0	3
1	1	1	1	4
0	1	1	1	5
0	0	1	1	6
0	0	0	1	7
0	0	0	0	8
1	0	0	0	9

Excitation table

Synthesis of sequential circuits:

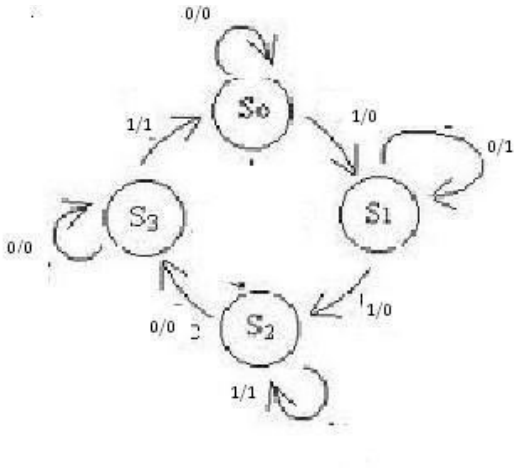
The synchronous or clocked sequential circuits are represented by two models.

1. Moore circuit: in this model, the output depends only on the present state of the flip-flops
2. Meelay circuit: in this model, the output depends on both present state of the flip-flop. And the inputs.

Sequential circuits are also called finite state machines (FSMs). This name is due to the fast that the functional behavior of these circuits can be represented using a finite number of states.

State diagram: the state diagram or state graph is a pictorial representation of the relationships between the present state, the input, the next state, and the output of a sequential circuit. The state diagram is a pictorial representation of the behavior of a sequential circuit.

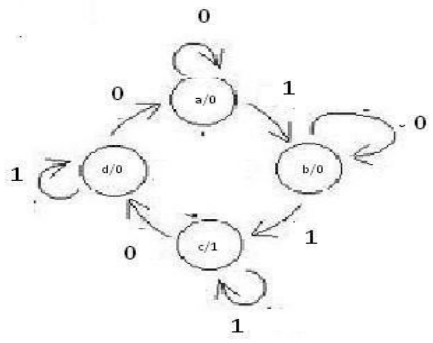
The state represented by a circle also called the node or vertex and the transition between states is indicated by directed lines connecting circle. a directed line connecting a circle with itself indicates that the next state is the same as the present state. The binary number inside each circle identifies the state represented by the circle. The direct lines are labeled with two binary numbers separated by a symbol. The input value is applied during the present state is labeled after the symbol.



	NS,O/P	
	INPUT X	
PS	X=0	X=1
a	a,0	b,0
b	b,1	c,0
c	d,0	c,1
d	d,0	a,1

Fig :a) state diagram (meelay circuit) fig: b) state table

In case of moore circuit ,the directed lines are labeled with only one binary number representing the input that causes the state transition. The output is indicated with in the circle below the present state, because the output depends only on the present state and not on the input.



	NS		
	INPUT X		
PS	X=0	X=1	O/P
a	a	b	0
b	b	c	0
c	d	c	1
d	a	d	0

Fig: a) state diagram (moore circuit) fig:b) state table

Serial binary adder:

Step1: word statement of the problem: the block diagram of a serial binary adder is shown in fig. it is a synchronous circuit with two input terminals designated X1and X2 which carry the two binary numbers to be added and one output terminal Z which represents the sum. The inputs and outputs consist of fixed-length sequences 0s and 1s.the output of the serial Z_i at time $t_i$is a function of the inputs $X_1(t_i)$ and $X_2(t_i)$ at that time t_{i-1} and of carry which had been generated at t_{i-1}. The carry which represent the past history of the serial adder may be a 0 or 1. The circuit has two states. If one state indicates that carry from the previous addition is a 0, the other state indicates that the carry from the previous addition is a 1

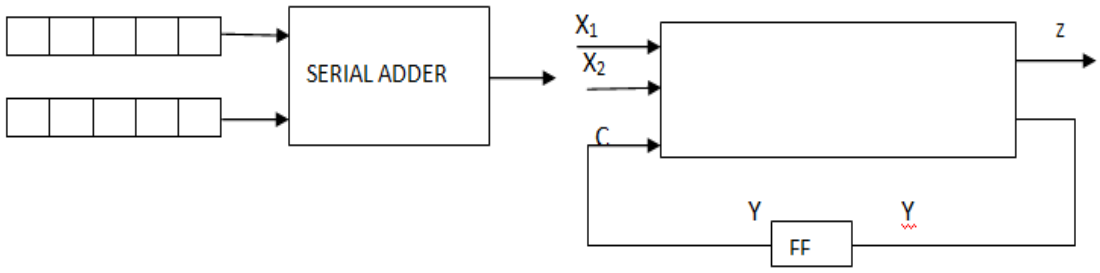
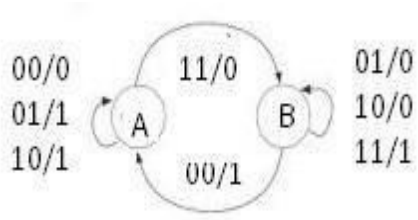


Figure: block diagram of serial binary adder

Step2 and 3: state diagram and state table: let a designate the state of the serial adder at t_i if a carry 0 was generated at t_{i-1}, and let b designate the state of the serial adder at t_i if carry 1 was generated at t_{i-1} .the state of the adder at that time when the present inputs are applied is referred to as the present state(PS) and the state to which the adder goes as a result of the new carry value is referred to as next state(NS).

The behavior of serial adder may be described by the state diagram and state table.



PS	NS ,O/P			
	X1 X2			
	0	0	1	1
	0	1	0	1
A	A,0	B,0	B,1	B,0
B	A,1	B,0	B,0	B,1

Figures: serial adder state diagram and state table

If the machine is in state B, i.e., carry from the previous addition is a 1, inputs X_1=0 and X_2=1 gives sum, 0 and carry 1. So the machine remains in state B and outputs a 0. Inputs X_1=1 and X_2=0 gives sum, 0 and carry 1. So the machine remains in state B and outputs a 0. Inputs X_1=1 and X_2=1 gives sum, 1 and carry 0. So the machine remains in state B and outputs a 1. Inputs X_1=0 and X_2=0 gives sum, 1 and carry 0. So the machine goes to state A and outputs a 1. The state table also gives the same information.

Setp4: reduced standard from state table: the machine is already in this form. So no need to do anything

Step5: state assignment and transition and output table:

The states, A=0 and B=1 have already been assigned. So, the transition and output table is as shown.

PS	NS				O/P			
	0	0	1	1	0	0	1	1
	0	1	0	1	0	1	0	1
0	0	0	0	1	0	1	1	1
1	0	1	1	1	1	0	0	1

STEP6: choose type of FF and excitation table: to write table, select the memory element the excitation table is as shown in fig.

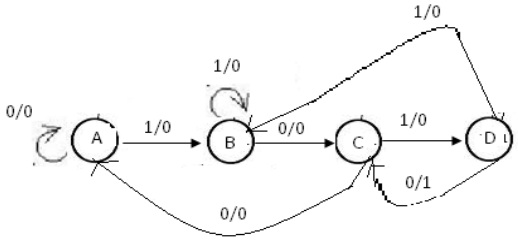
PS	I/P		NS	I/P-FF	O/P
y	x1	x2	Y	D	Z
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	1	1

Sequence detector:

Step1: word statement of the problem: a sequence detector is a sequential machine which produces an output 1 every time the desired sequence is detected and an output 0 at all other times

Suppose we want to design a sequence detector to detect the sequence 1010 and say that overlapping is permitted i.e., for example, if the input sequence is 01101010 the corresponding output sequence is 00000101.

Step2 and 3: state diagram and state table: the state diagram and the state table of the sequence detector. At the time t_1, the machine is assumed to be in the initial state designed arbitrarily as A. while in this state, the machine can receive first bit input, either a 0 o r a 1. If the input bit is 0, the machine does not start the detection process because the first bit in the desired sequence is a 1. If the input bit is a 1 the detection process starts.



PS	NS,Z	
	X=0	X=1
A	A,0	B,0
B	C,0	B,0
C	A,0	D,0
D	C,1	B,0

Figure: state diagram and state table of sequence detector

So, the machine goes to state B and outputs a 0. While in state B, the machinery may receive 0 or 1 bit. If the bit is 0, the machine goes to the next state, say state c, because the previous two bits are 10 which are a part of the valid sequence, and outputs 0.. if the bit is a 1, the two bits become 11 and this not a part of the valid sequence

Step4: reduced standard form state table: the machine is already in this form. So no need to do anything.

Step5: state assignment and transition and output table: there are four states therefore two states variables are required. Two state variables can have a maximum of four states, so, all states are utilized and thus there are no invalid states. Hence, there are no don't cares. Let a=00, B=01, C=10 and D=11 be the state assignment.

	NS(Y1Y2)		O/P(z)	
PS(y1y2	X=0	X=1	X=0	X=1
A= 0 0	0 0	0 1	0	0
B=0 1	1 0	0 1	0	0
C=1 0	0 0	1 1	0	0
D=1 1	1 1	0 1	1	0

Step6: choose type of flip-flops and form the excitation table: select the D flip-flops as memory elements and draw the excitation table.

					INPUTS -		
PS		**I/P**	**NS**		**FFS**		**O/P**
y1	Y2	X	Y1	Y2	D1	D2	Z
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	1	0	1	0	0
0	1	1	0	1	0	1	0
1	0	0	0	0	0	0	0
1	0	1	1	1	1	1	0
1	1	0	1	0	1	0	1
1	1	1	0	1	0	1	0

Step7: K-maps and minimal functions: based on the contents of the excitation table , draw the k- map and simplify them to obtain the minimal expressions for D1 and D2 in terms of y1, y2 and x as shown in fig. The expression for z (z=y1,y2) can be obtained directly from table

Step8: implementation: the logic diagram based on these minimal expressions

UNIT - V

Memory

- Random Access Memory - 335
- Types of ROM - 341 & Internet.
- Memory decoding - 342
- Address and data bus -
- Sequential Memory - 364
- Cache Memory -
- Programmable Logic Arrays - 356
- Memory Hierarchy interms of Capacity and Access time -

Memory:

A memory unit is a collection of cells capable of storing a large quantity of binary information.

→ A memory unit is a device which binary information is transferred for storage and from which information is retrieved when needed for processing.

→ When data processing takes place, informa from memory is transferred to selected registers in the processing unit. Intermediat data and final results obtained in the processing unit are transferred back to be stored in memory.

→ Binary information received from an i/p device is stored in memory, and informati transferred to an o/p device is taken from memory.

There are two types of memories that are used in digital system

- Random Access Memory (RAM)
- Read Only Memory (ROM)

RAM:

Random Access Memory stores new info for later use.

→ The process of storing new information into memory is referred to as a 'memory write' Operation.

→ The processing of transferring stored information out of memory is referred as a 'memory "read" operation.

→ RAM can perform both 'write' and 'read' Operations.

ROM:

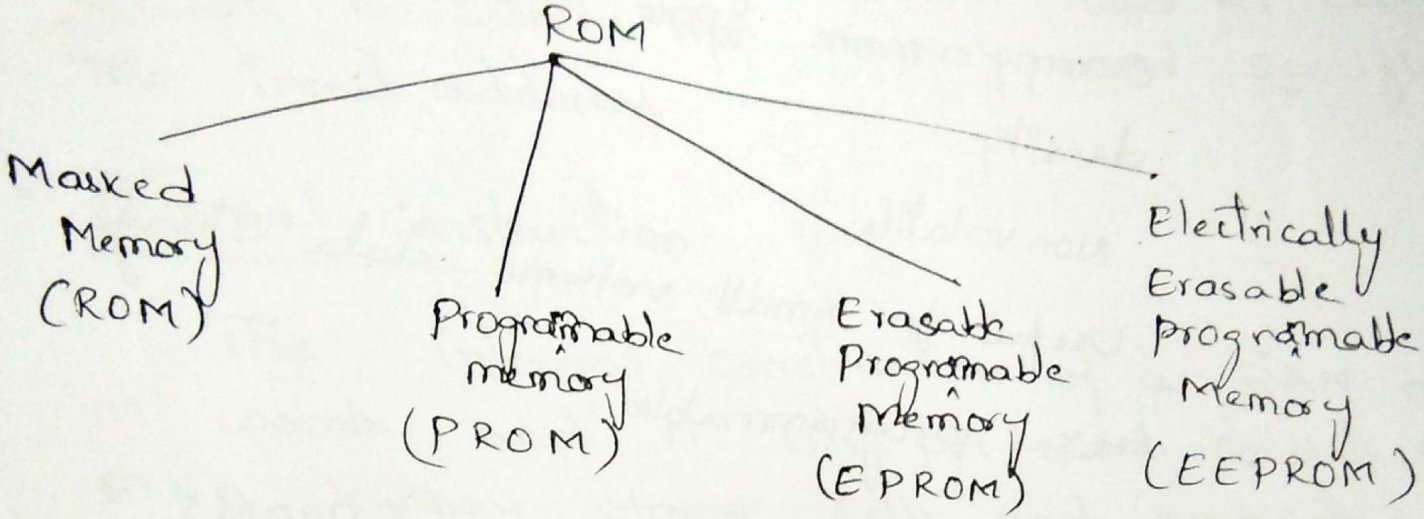
Read Only Memory. It can perform only the 'read' Operation. This means that suitable information is already stored inside memory and can be retrieved or read at any time. However, that information cannot be altered by writing.

→ ROM is a Programmable Logic Device (PLD). The binary information that is stored within such a device is specified in some fashion and then embedded within the hardware in a process is referred to a programming the device.

Types of ROMs :-

ROM is a Read Only Memory.

The ROMs are classified as



1. Masked Memory (ROM) :-

- Masked program during manufacturing
- Can not be programmed.
- Nonvolatile, retain data even when power is shut off
- Cheaper than programmable devices
- Generally used to perform specific functions.
- Useful for fixed programme instructions.

2) Programmable Memory (PROM):-

- Programmed by blowing built-in fuses
- Cannot be reprogrammed
- Occupy more space, leads to low memory density.
- Non volatile
- Useful for small volume data storage.
- User programmable.

3) Erasable Programmable Memory (EPROM)

- Erasable Programmable ROM
- Programmed by storing charge on insula gates.
- Erasable with ultraviolet light
- Reprogrammable after erasure
- Non volatile

4) Electrically Erasable Programmable Memory (EEPRO

- Electrically erasable programmable ROM.
- Programmed by storing charges on insulated gates
- Non volatile.

Memory Decoding:

In addition to requiring storage componen in a memory unit, there is a need for decodin circuits to select the memory word specified the input address.

- **Internal Construction:**

The internal construction of a RAM of 'm' words and 'n' bits per word consists of mxn binary storage cells and associated decoding circuits for selecting individual wor

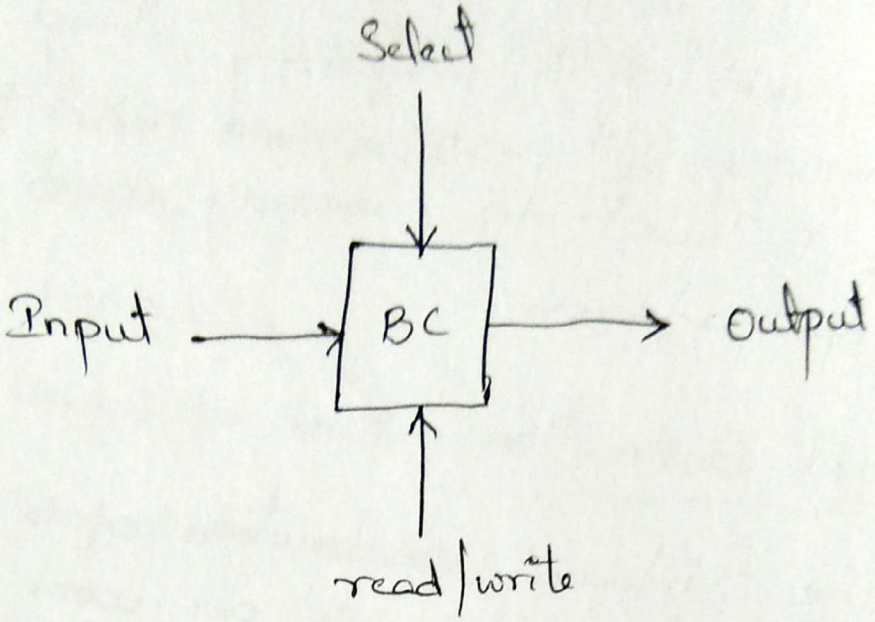
→ The binary storage cell is the basic build block of a memory unit.

→ Actually, the cell is an electronic circuit with four to six transistors.

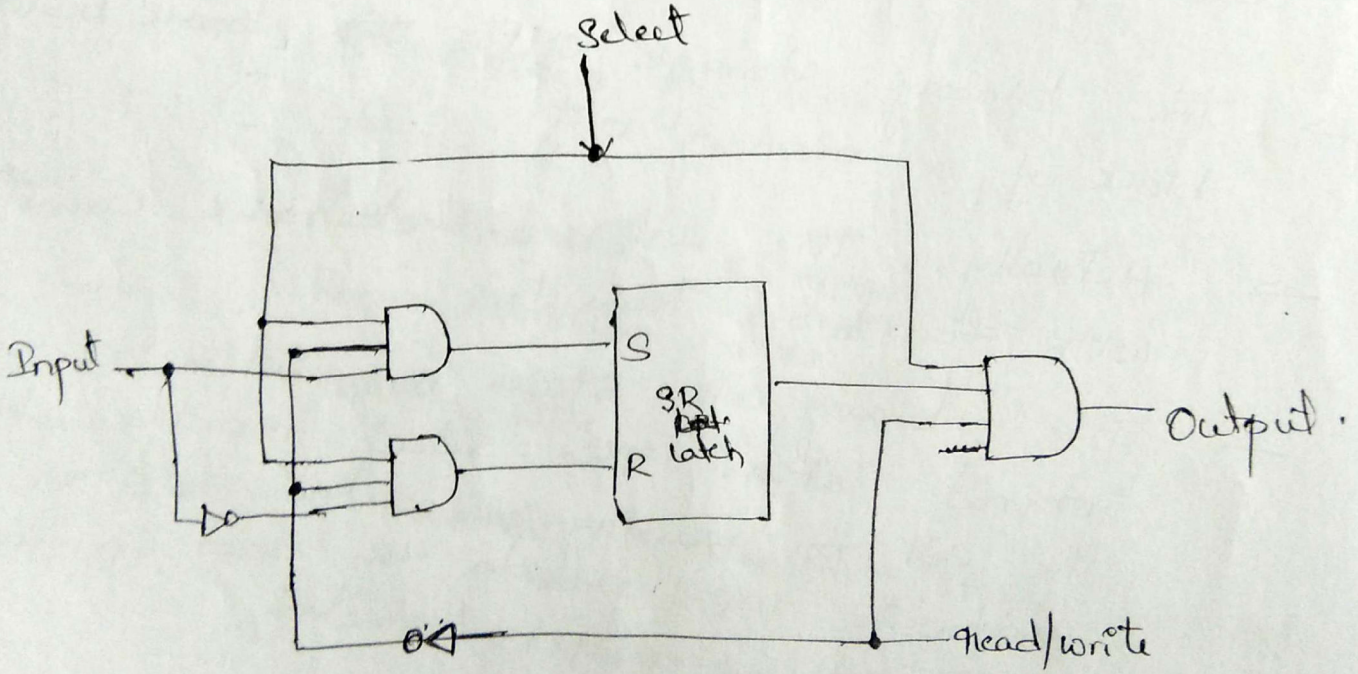
→ The binary cell stores one bit in its internal latch. The select input enables the cell for reading or writing; and the read/write input provides the read operation of the cell when it is selected

→ A '1' in the read/write input provids the 'read' operation and a '0' in the read/wri input provides the 'write' operation by formin a path from the input terminal to the latch.

Memory cell Block diagram:



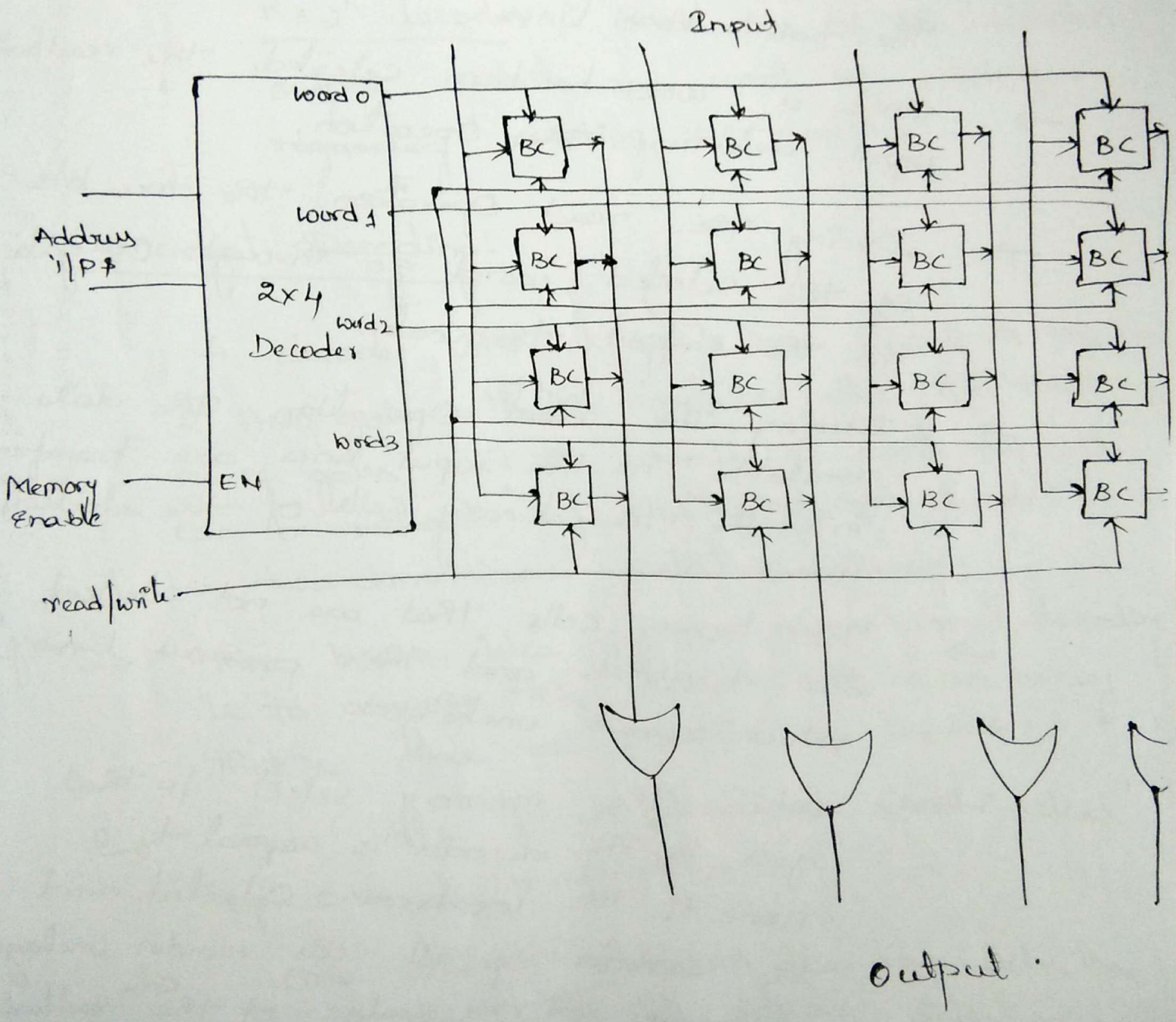
Memory cell logic diagram:



The Storage part of the cell is modeled by an SR latch with associated gates to form a D latch.

A logic construction of a 4x4 RAM

RAM consists of 4 words of 4 bits each and has a total of 16 binary cells. The small blocks labeled 'BC' represent the binary cell with its 3 i/p's and one o/p.



→ A memory with 4 words needs two address lines. The two address inputs go through a 2x4 decoder to select one of the four words.

→ The decoder is enabled with the memory enable input. When the memory enable is 0, all outputs of the decoder are 0 and none of the memory words are selected.

→ With the memory select at 1, one of the four words is selected, dictated by the value in the two address lines.

→ Once a word has been selected, the read/write input determines the operation.

→ During the read operation, the four bits of the selected word go through OR gates to the output terminals.

→ During the write operation, the data available in the input lines are transferred into the four binary cells of the selected word.

→ The binary cells that are not selected are disabled, and their previous binary values remain unchanged.

→ When the memory select i/p that goes to the decoder is equal to 0, none of the words are selected and the contents of all cells remain unchanged regardless of the value of the read/write input.

→ Commerical RAM may have a capacity of thousands of words, and each word may range from 1 to 64 bit.

→ A memory with 2^K words of n bits per word requires K address lines that go into a $K \times 2^K$ decoder. Each one of the decoder outputs selects one word of n bit for reading or writing.

Coincident Decoding.

A decoder with K inputs and 2^K outputs requires 2^K AND gate with K inputs per gate. The total no. of gates and the number of inputs per gate can be reduced by employing two decoders in a two-dimensional selection scheme.

→ The basic idea in two-dimensional decoding is to arrange the memory cells in an array that is close as possible to square.

→ Two $K/2$ i/p decoders are used instead of one K-i/p decoder.

→ One decoder performs the row selection and the other the column selection in a 2-D matrix configuration.

→ In the 2-D selection pattern for a 1K-word memory instead of using a single 10×1024 decoder, we use two 5×32 decoders.

→ With the single decoder, we need 1024 AND gates with 10 i/ps in each.

→ In 2-D, we need 64 AND gates with 5 i/ps in each.

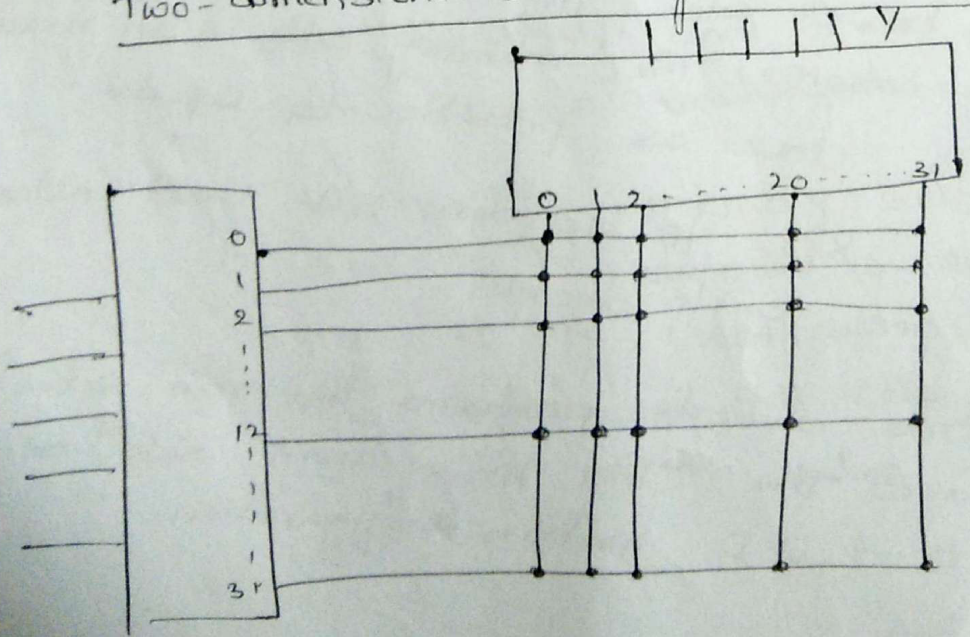
→ The 5 MSB of the address go to input X and the 5 LSB go to the i/p Y.

→ Each word within the memory array is selected by the coincidence of one X line and one Y line.

→ Thus, each word in memory is selected by coincidence between 1 of 32 rows and 1 of 32 columns, for the total of 1024 words.

* → Note that each intersection represents a word that may have any no. of bits.

Two-dimensional decoding structure for a 1K-word Memory

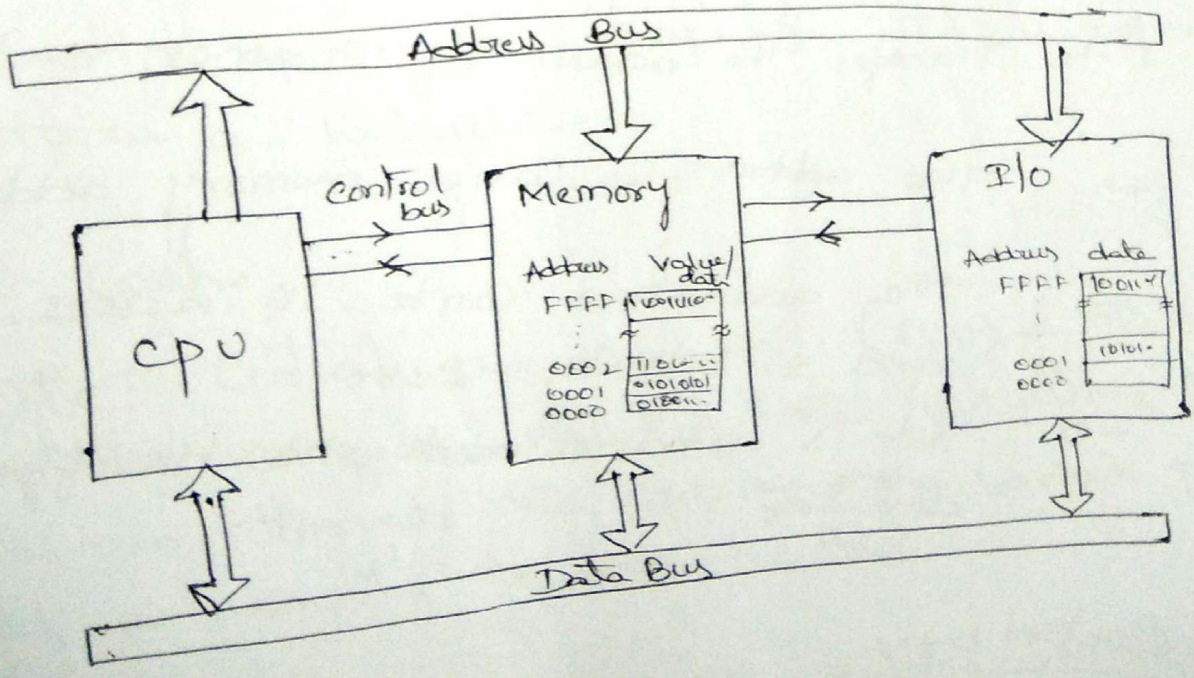


ddress and Data bus:

The bus is a medium used to transfer da and controls from one part to other part of Computers. Data is normally Transferred between main storage and the processor along a device called a bus, which is effectly means of sending multipl bits of data in paralled.

There are mainly three types of bus.

1) Data bus
2) Addrus bus
3) Control bus



Data bus:

As name tells that it is used to transfer d within microprocessor, I/o and memory devices.

→ It is bidirectional as microprocessor requires to send or receive data.

→ The data bus also works as bus in 8 bit long.

→ The word length of a processor depends on data bus, 8 bits in 8085, 16 bits in 8086, and 32 bits in 80386.

Address Bus:

It is group of wires or lines that are used to transfer the address of memory or I/o devices.

→ The address bus carries memory address.

→ The address lines can be 16 in 8085, 8086 - 20 bits and 32 bits in 80386.

→ It is unidirectional, the bits flow in only one direction.

Control bus:

→ The control bus carries the control signal between the units of computer.

→ The signals like READ/WRITE, START/HALT are carried by a control bus.

Cache Memory :

Cache Memory is a type of memory used to hold frequently used data. Cache memory is relatively small, but very fast. It essentially stores information that is likely to be used again.

→ Most web browsers use a cache to load regularly viewed webpages fast.

→ The Most important type of cache memory is the 'CPU Cache'.

→ For example, web browsers typically use a cache to make webpages load faster by storing a copy of the webpage files locally, such as on local computer. This is referred to as a 'webcache'.

CPU Cache :

The Central processing unit (CPU) is the brain of the computer. All of the instructions have to run through the CPU for the various parts of a computer to work together.

CPU chips have been getting smaller and faster as chip technology has advanced.

→ One of the slower aspects of computer is the interaction between the CPU chip and the main memory in the form of Random Access Memory (RAM). Installing more memory is not always a solution - the bottleneck is often the time it takes to access the memory.

→ So, A small form of memory located directly on the chip itself, which is called 'Cache'.

→ Cache is much smaller, but can be accessed much faster than the main memory.

→ The CPU cache stores the most frequently used pieces of information so they can be retrieved more quickly

How the CPU Cache works:

To carry out a particular instruction, the CPU needs a specific piece of information. The CPU will first check to see if this information is available in the CPU cache. If the information is found, this is called a 'Cache hit'. If the information is not found, this is called 'Cache miss', and the CPU goes on looking for the information elsewhere. In the case of cache miss, the piece of information will be found in the main memory but it will simply take longer.

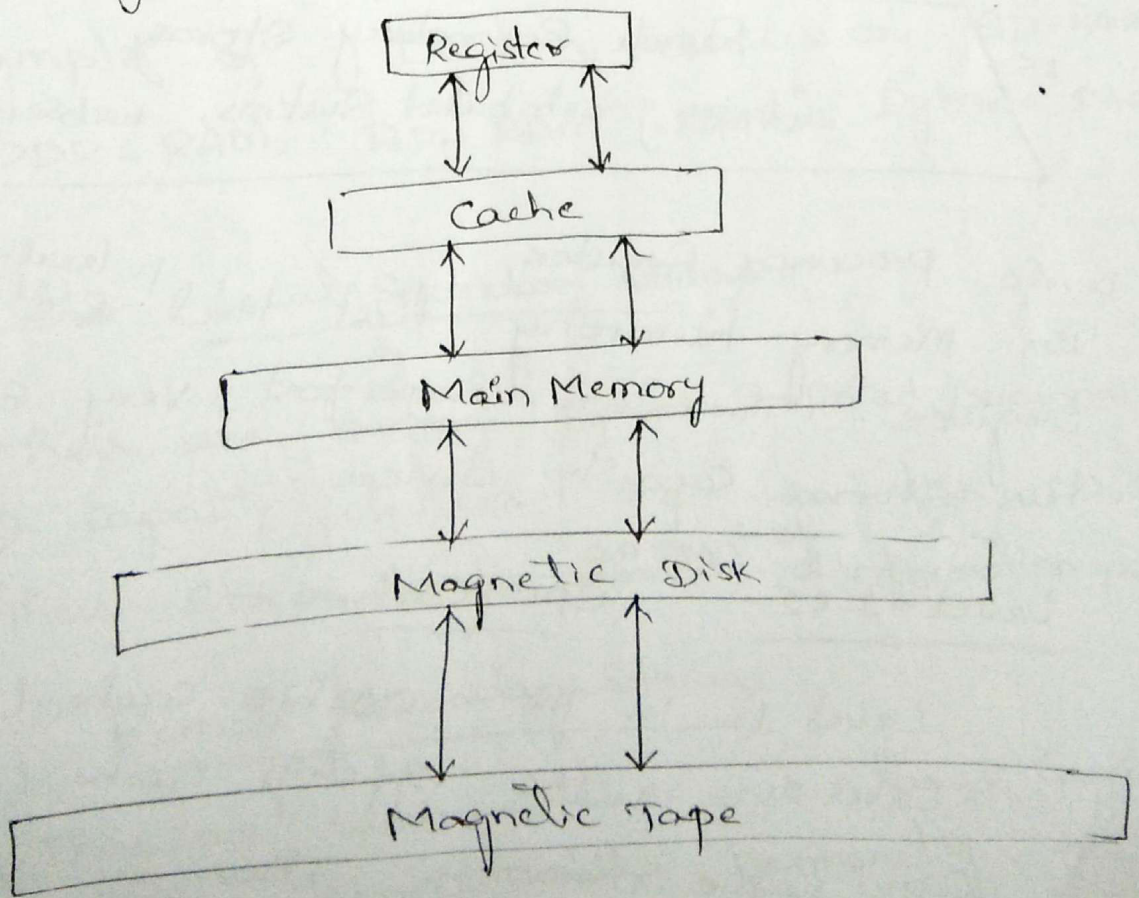
Memory Hierarchy:

The memory unit is an essential component in any digital computer since it is needed for storing programs and data.

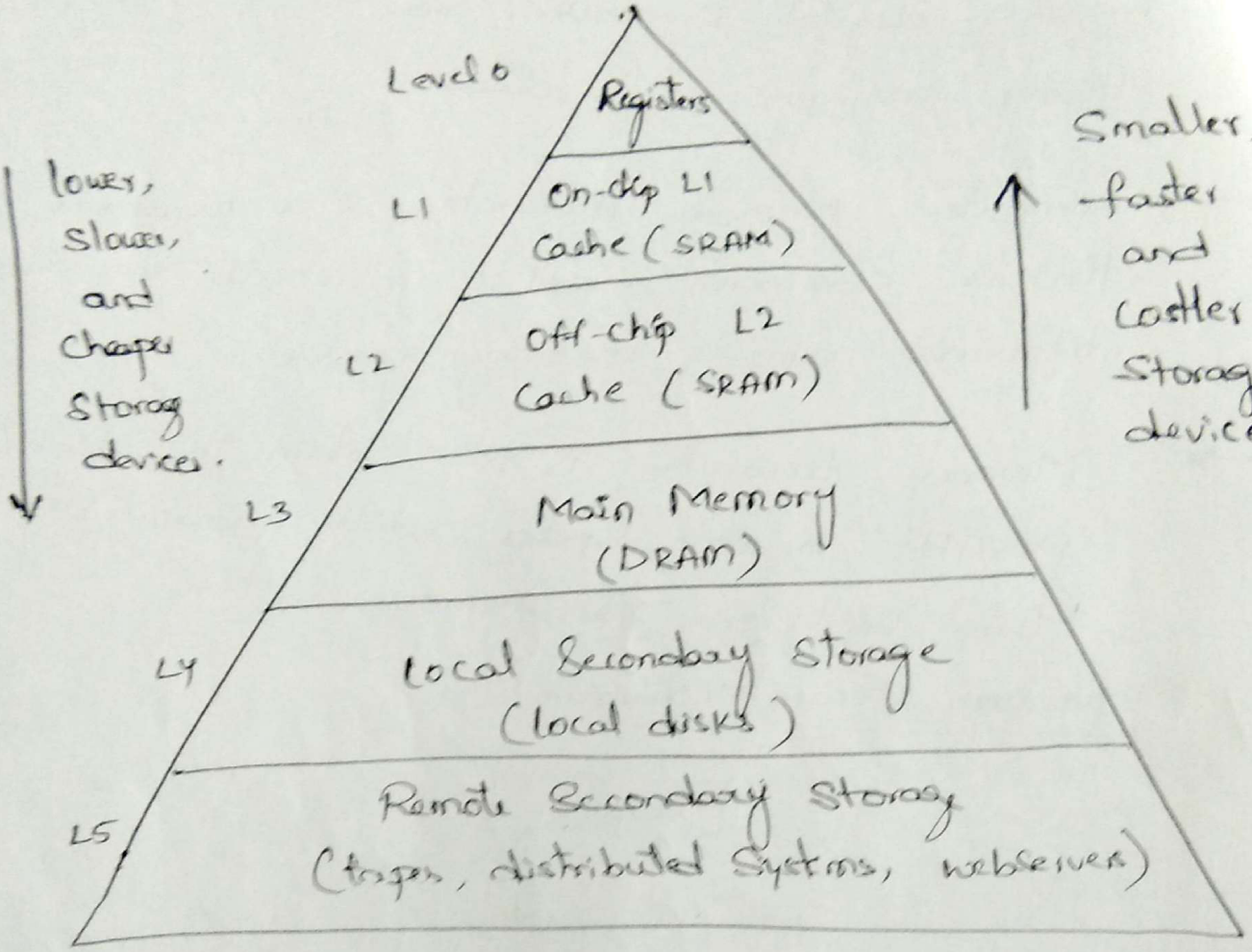
→ Computer Memory Hierarchy is a pyramid structure that is commonly used to illustrate the significant differences among memory types.

→ Memory hierarchy is to obtain the highest possible access speed while minimizing the total cost of the memory system.

Memory Hierarchy diagram:-



An Example Memory Hierarchy:



Level 0: Processor Registers.

→ In Memory hierarchy first level is level-0 this is CPU processor registers. They are superfast, very expensive and less storage capacity devices.

→ Level-1 & 2: CPU Cache:

Level 1 is the onchip cache-(L1) and
Level 2 is the offchip cache (L2) are the CPU cache memories. They are very faster, Expensive and small storage capacity devices.

Level 3: Main Memory:

MO Level-3 is the main memory. These are fast memory devices than the level 4 & 5 but comparitivly slower than level 1, 2 & 3. These are available in reasonable price. but the storage capacity is average.

→ Most of the main memory in a general purpose computer is made up of RAM, but a portion of the memory may be constructed with ROM chip

→ RAM - Random Access Memory.
ROM - Read Only Memory.

example of Level 3 devices are SD-RAM, DDR-RAM, RD-RAM (Rambus Dynamic RAM) etc

Level-4: Local Secondary Storage:

These are average speed, priced reasonable average capacity devices. Ex: Hard disk, CD, DVD, etc. These stores large amount of information permanently

Level-5: Remote Secondary Storage:

These are slow, cheap and large capacity devices. These stores data permanetly. These are used for backup storage. Examples: Flash drive, Tape devices like floppy, pendrives, SD-cards etc.

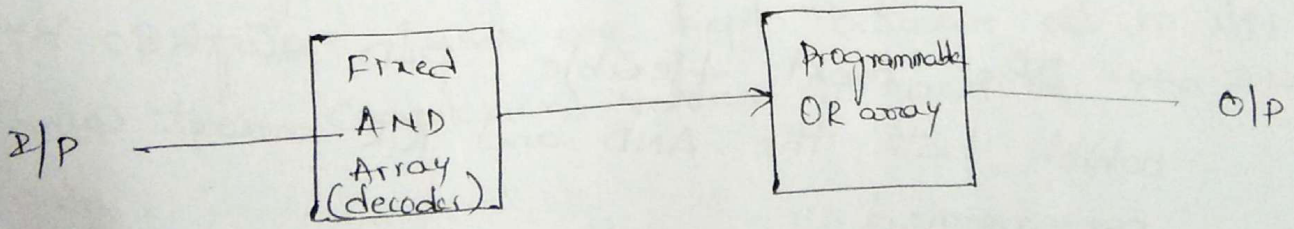
Combinational PLDs:

The PROM is a combinational Programmable Logic Device (PLD). an IC with programmable gates divided into an AND array and an OR array to provide an AND-OR Sum-of-product (SOP) implementation.

There are 3 major types of Combinational PLDs.

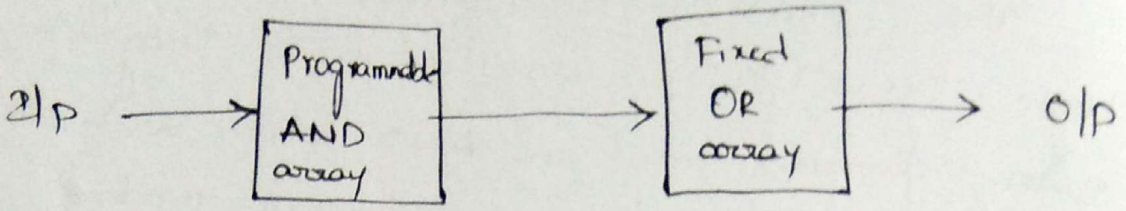
1. Programmable Read-only Memory (PROM)
2. Programmable Array Logic (PAL)
3. Programmable Logic Array (PLA)

1. PROM:



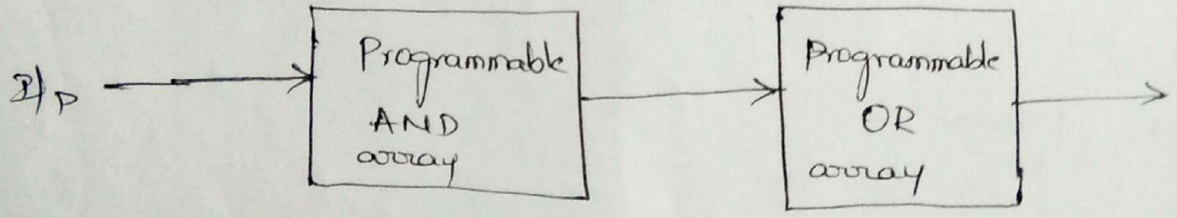
The PROM has a fixed AND array constructed as a decoder and a programmable OR array. The programmable OR gates implement the Boolean function in Sum-of-minterms form.

2. PAL :



The PAL has a programmable AND array and a fixed OR array. The AND gates are programmed to provide the product terms for the Boolean functions which are logically summed in each OR gate.

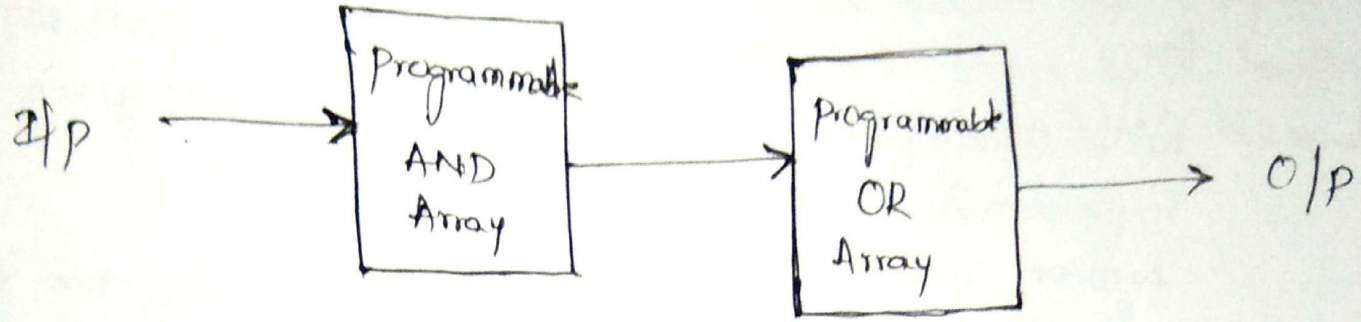
3. PLA :



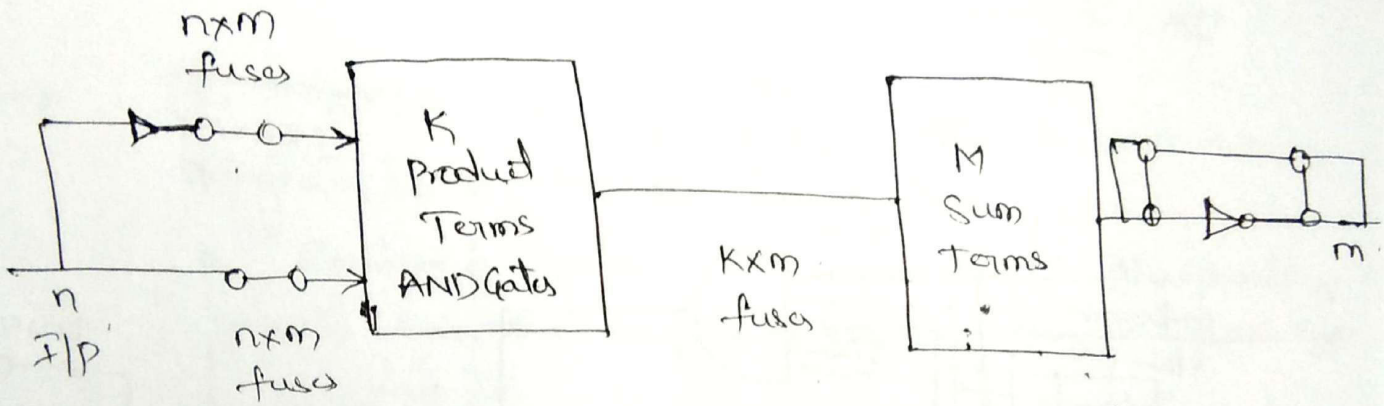
The most flexible PLD is the PLA, in which both the AND and OR arrays can be programmed.

The product terms in the AND array may be shared by an OR gate to provide the required Sum-of-product implementation.

Programmable Logic Array: (PLA)



Block diagram:



The ~~PLAs~~ ~~are~~ product terms is a group of K AND gates and Sum terms is a group of M OR gates. Fuses are kept between all n i/ps and their complement values to each of the AND gates.

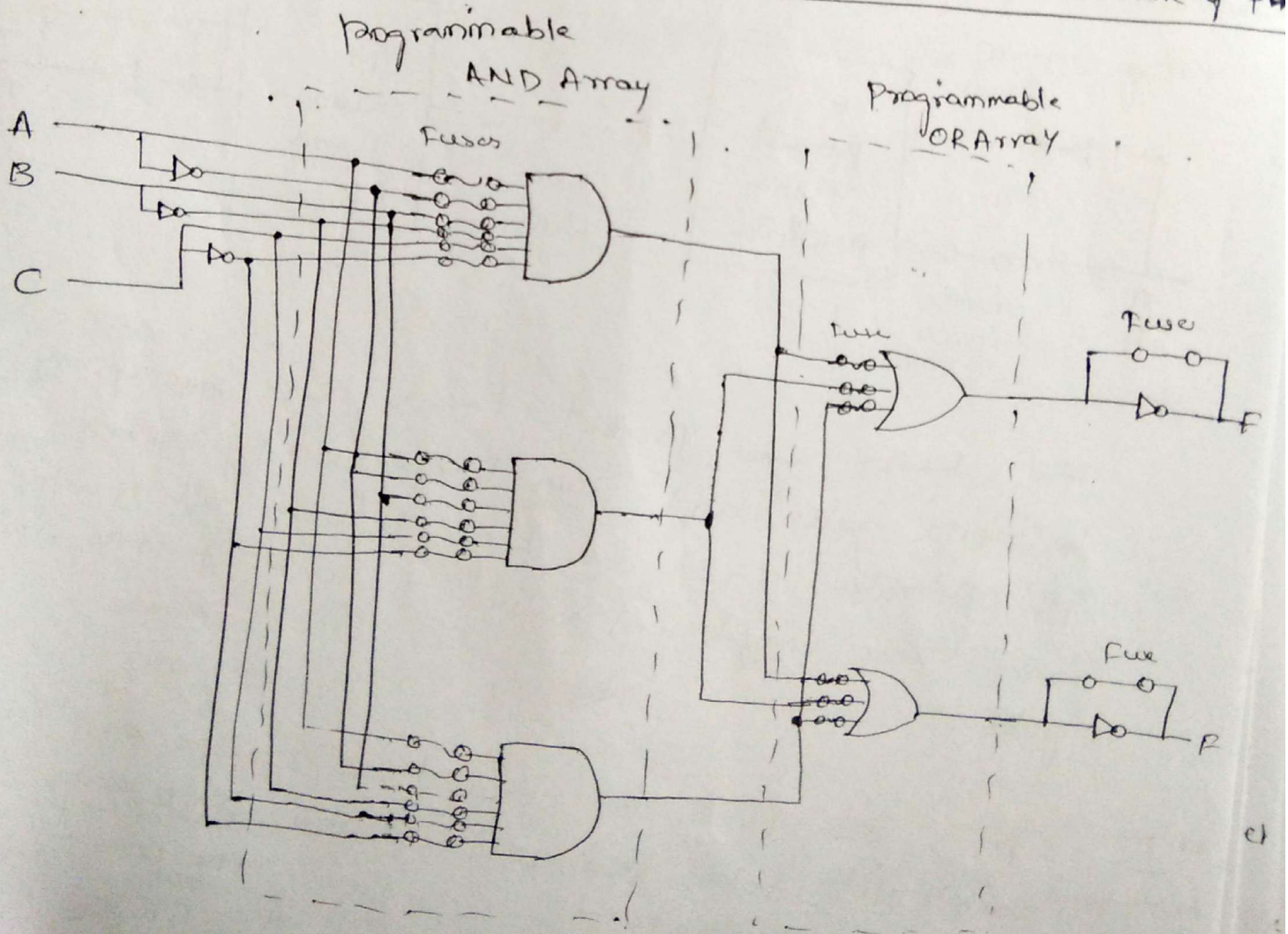
→ Fuses are used to connect all n i/ps and their complement values to each of the AND gates. Fuses are also used to connect the o/ps of the AND gates, and the i/ps of the OR gates.

→ The fuses at the inverters o/p generates the function in AND-OR form or in AND-OR-INVERT form.

→ With the fuse blown gives AND-OR-INVERT implementation. With the fuse not blown, gives AND-OR implementation.

→ The size of the PLA is specified by the no. of i/ps, product terms, and o/ps.

Internal connection of the General Structure of PLA



Each i/p and its complement are connected by fuse to the i/p's of all AND gates, the o/p of the OR gates. Two fuses are at the output Inverters. By blowing selected fuses and leaving other fuses it is easy to implement boolean function in SOP form.

The PLA's available are of two types

1) Mask-Programmable logic array (MPLA)
2) Field-Programmable logic array. (FPLA)

→ In MPLA, the consumer must submit a PLA program table to the manufacturers. Vendor produce a Customer made PLA according to the table which has the required paths b/w the i/p & o/p.

→ The FPLA can be programmed by the user by means of certain recommended procedures. Commercial H/w programmer units are available for use in conjunction with certain FPLA's.

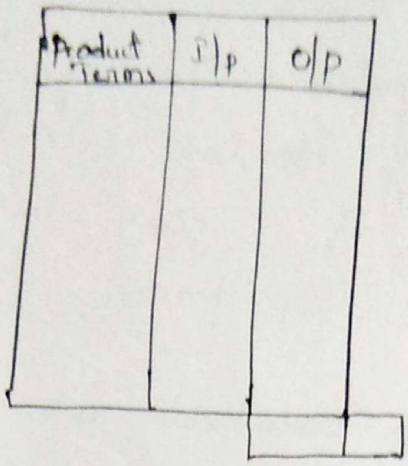
PLA Program Table:-

The PLA consists of 3 columns. The first column lists the product terms numerically. The second column specifies the path b/w on the AND gates and the OR gates.

→ Under each o/p variable, we write T (for true) if the o/p inverter is to be bypassed, and C (for complement) if the function is to be

Complemented with the o/p inverters.

Ex:

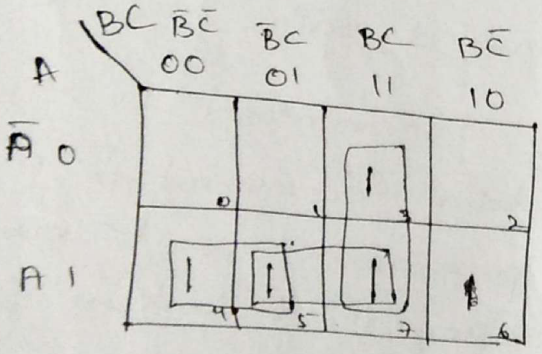


Ex:- 1) Implement the boolean functions F1 and F2 of a combinational logic circuit using PLA

where $F_1(A,B,C) = \Sigma(3,4,5,7)$

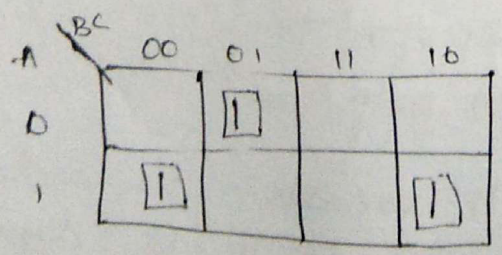
$F_2(A,B,C) = \Sigma(1,4,6)$

Solution:- $F_1(A,B,C) = \Sigma(3,4,5,7)$

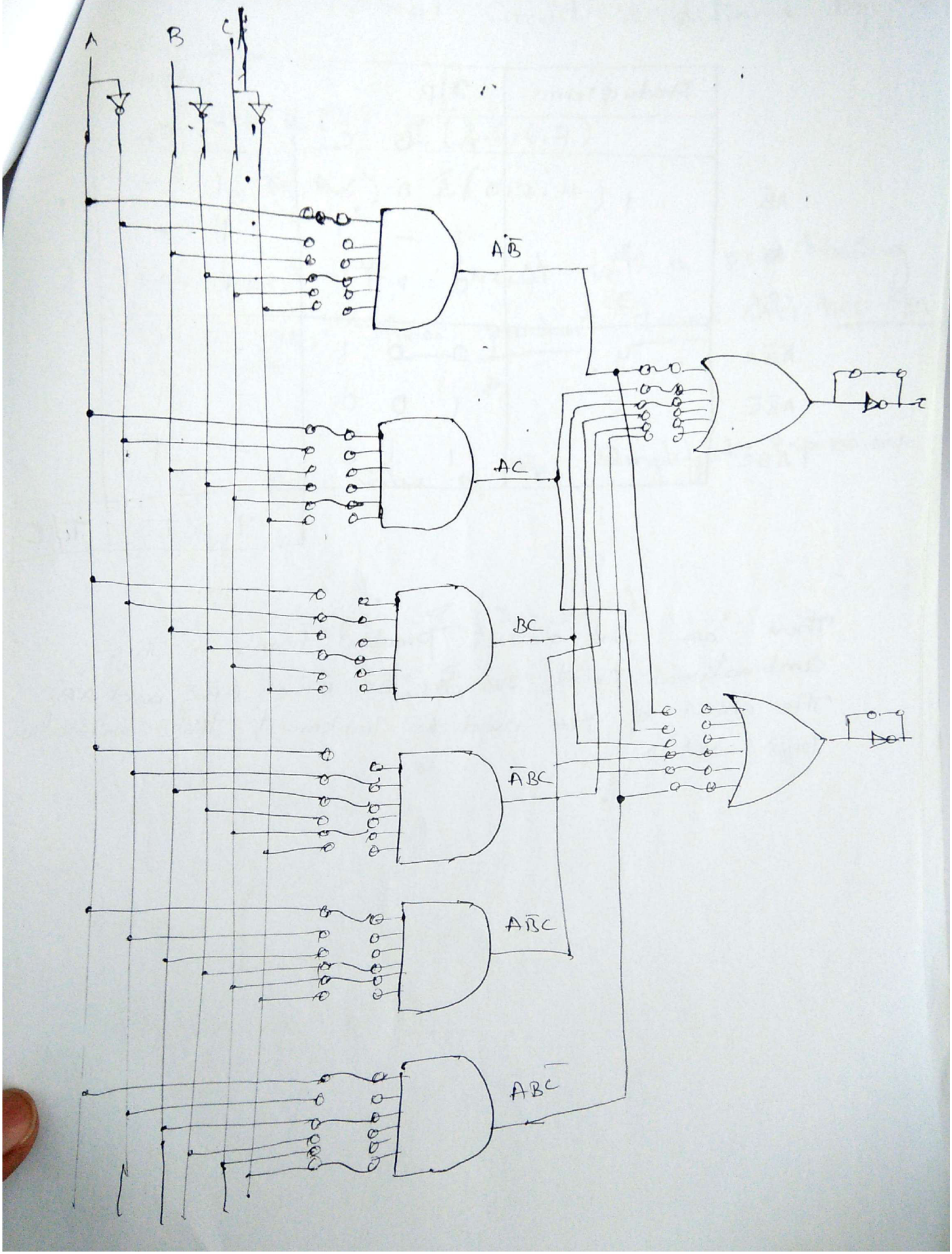


$F_1 = A\bar{B} + AC + BC$.

$F_2(A,B,C) = \Sigma(1,4,6)$



$F_2 = \bar{A}\bar{B}C + A\bar{B}\bar{C} + \bar{A}\bar{B}C$



PLA program table

	Product Terms	I/p			outputs	
		A	B	C	F_1	F_2
$A\bar{B}$	1	1	0	–	1	–
AC	2	1	–	1	1	–
BC	3	0	1	1	1	–
$\bar{A}\bar{B}C$	4	0	0	1	–	1
$A\bar{B}\bar{C}$	5	1	0	0	–	1
$AB\bar{C}$	6	1	1	0	–	1
					T	T

T/C

There are six distinct product terms in this combinational circuit, $A\bar{B}$, AC, BC, $\bar{A}\bar{B}C$, $A\bar{B}\bar{C}$ and $AB\bar{C}$. The circuit of PLA used to implement this combinational logic ckt is

(P.T.O)

A combinational circuit is defined by the functions

$F_1(A,B,C) = \Sigma(3,5,6,7)$

$F_2(A,B,C) = \Sigma(0,2,4)$

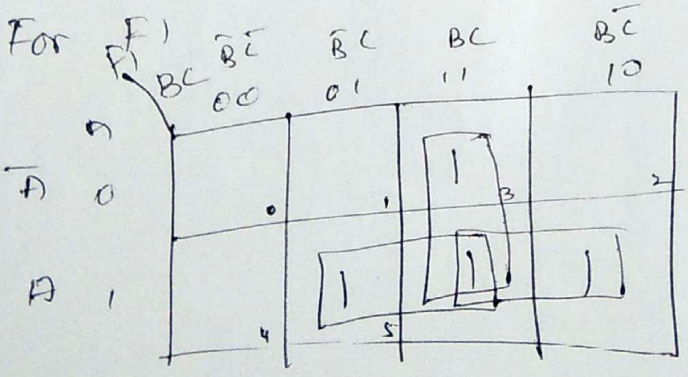
Implement the circuit with a PLA having three i/ps, three product terms and two o/ps

The True value of the Simplified expressions are

$F_1(A,B,C) = \Sigma(3,5,6,7)$

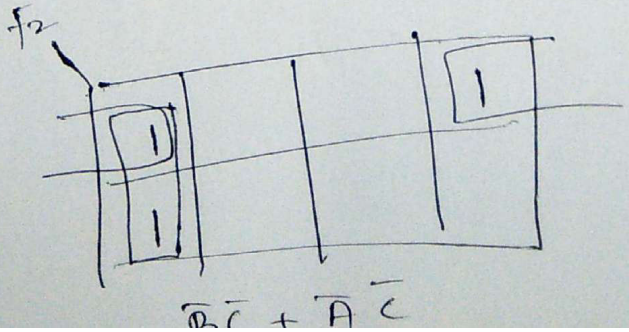
$F_2(A,B,C) = \Sigma(0,2,4)$

For F_1



$F_1 = AC + AB + BC$

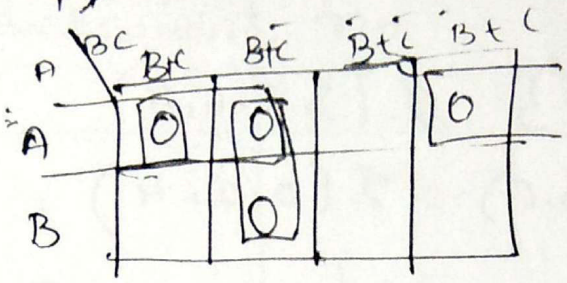
For F_2



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

The complemented value of the simplified expressions are

For $\bar{F_1}$



$$F_1 = (A+C)(B+\bar{C})(A+AB)$$

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

$$\bar{F_1} = \bar{B}\bar{C} + \bar{A}\bar{B} + \bar{A}\bar{C}$$

→ Circuit with a PLA having 3-I/P, 3-Product terms

Note:

(∵ $F_1 = BC + AB + A$

$\bar{F_1} = \overline{BC + AB + AC}$

Apply Demorgan theorem we will get final exp

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

UNIT NO 5

– Register Transfer and Microoperations –

Register Transfer Language

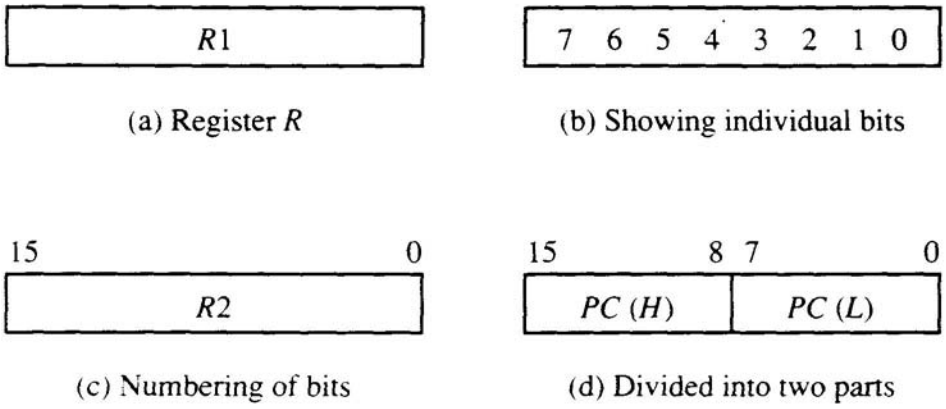
- Digital systems are composed of modules that are constructed from digital components, such as registers, decoders, arithmetic elements, and controllogic
- The modules are interconnected with common data and control paths to forma digital computersystem
- The operations executed on data stored in registers are called*microoperations*
- A microoperation is an elementary operation performed on the information stored in one or moreregisters
- Examples are shift, count, clear, andload
- Some of the digital components from before are registers thatimplement microoperations
- The internal hardware organization of a digital computer is best defined by specifying
    - The set of registers it contains and theirfunctions
    - The sequence of microoperations performed on the binaryinformation stored
    - The control that initiates the sequence ofmicrooperations

- Use symbols, rather than words, to specify the sequence ofmicrooperations
- The symbolic notation used is called a *register transferlanguage*
- A programming language is a procedure for writing symbols to specify a given computationalprocess
- Define symbols for various types of microoperations and describe associated hardware that can implement themicrooperations

– Register Transfer

- Designate computer registers by capital letters to denote itsfunction
- The register that holds an address for the memory unit is calledMAR
- The program counter register is calledPC
- IR is the instruction register and R1 is a processorregister
- The individual flip-flops in an *n*-bit register are numbered in sequence from 0to *n*-1
- Refer to Figure 4.1 for the different representations of aregister

Figure 4-1 Block diagram of register.



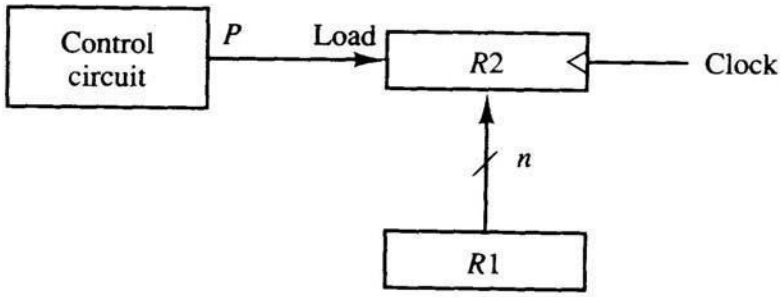
- Designate information transfer from one register to anotherby
  R2 ←R1

- This statement implies that the hardware isavailable
  - The outputs of the source must have a path to the inputs of thedestination
  - The destination register has a parallel loadcapability

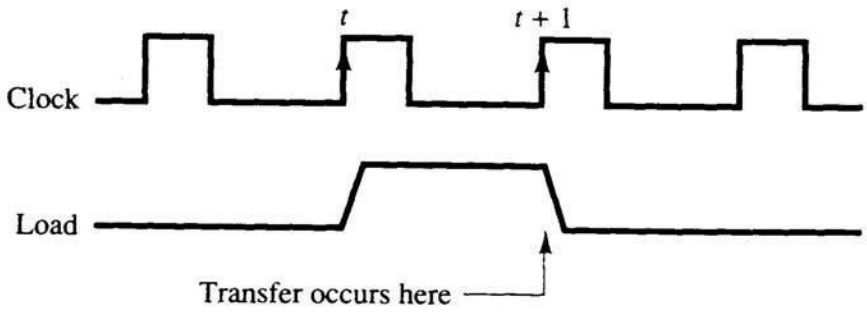
- If the transfer is to occur only under a predetermined control condition, designate it by
  If (P = 1) *then* (R2 ← R1)
  or,
  P: R2 ← R1,
  where P is a control function that can be either 0 or 1

- Every statement written in register transfer notation implies the presence of the required hardwareconstruction

Figure 4-2 Transfer from R1 to R2 when $P = 1$.



(a) Block diagram



(b) Timing diagram

- It is assumed that all transfers occur during a clock edgetransition
- All microoperations written on a single line are to be executed at the same time

  T: R2 ← R1, R1 ←R2

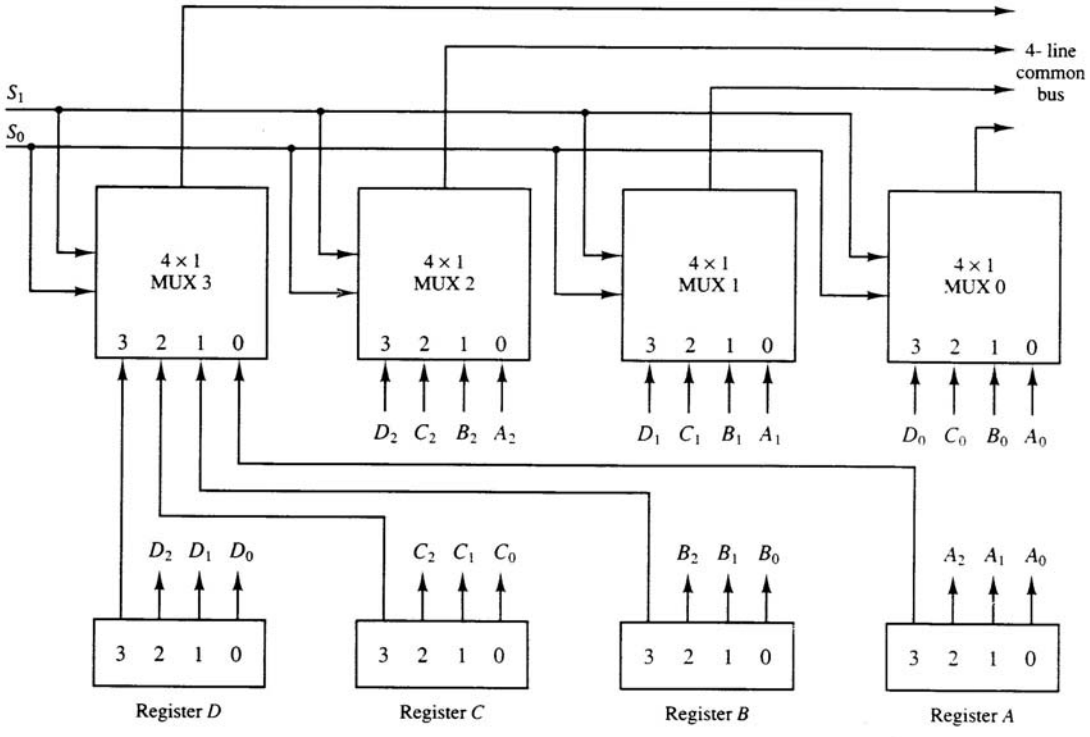
TABLE 4-1 Basic Symbols for Register Transfers

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	*MAR*, *R2*
Parentheses ( )	Denotes a part of a register	*R2*(0–7), *R2*(*L*)
Arrow ←	Denotes transfer of information	*R2* ← *R1*
Comma ,	Separates two microoperations	*R2* ← *R1*, *R1* ← *R2*

– Bus and Memory Transfers

- Rather than connecting wires between all registers, a common bus isused
- A bus structure consists of a set of common lines, one for each bit of aregister
- Control signals determine which register is selected by the bus duringeach transfer
- Multiplexers can be used to construct a commonbus
- Multiplexers select the source register whose binary information is then placed on thebus
- The select lines are connected to the selection inputs of the multiplexers and choose the bits of oneregister

Figure 4-3 Bus system for four registers.

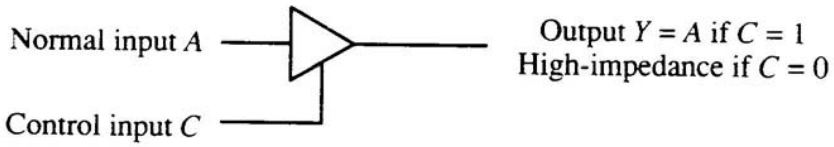


- In general, a bys system will multiplex *k* registers of *n* bits each to produce an *n*-line commonbus
- This requires *n* multiplexers – one for eachbit
- The size of each multiplexer must be *k* x1
- The number of select lines required is *logk*
- To transfer information from the bus to a register, the bus lines are connected to the inputs of all destination registers and the corresponding load control line must be activated
- Rather than listing each stepas

BUS ← C, R1 ← BUS,
use R1 ← C, since the bus isimplied

- Instead of using multiplexers, *three-state gates* can be used to construct the bus system
- A three-state gate is a digital circuit that exhibits threestates
- Two of the states are signals equivalent to logic 1 and0
- The third state is a *high-impedance* state – this behaves like an open circuit, which means the output is disconnected and does not have a logicsignificance

Figure 4-4 Graphic symbols for three-state buffer.



- The three-state buffer gate has a normal input and a control input which determines the outputstate
- With control 1, the output equals the normalinput
- With control 0, the gate goes to a high-impedancestate
- This enables a large number of three-state gate outputs to be connected with wires to form a common bus line without endangering loadingeffects

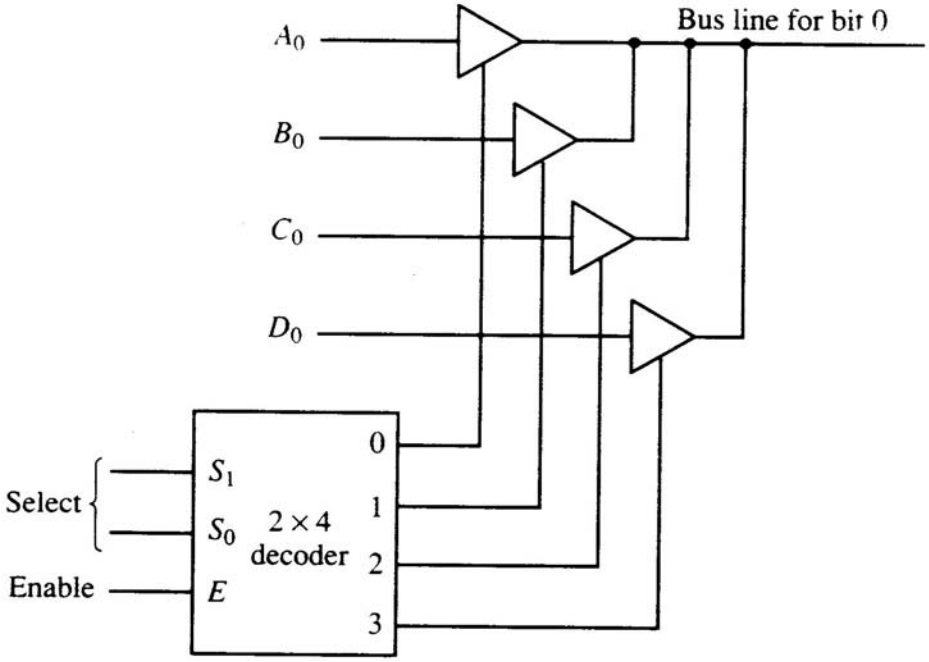


Figure 4-5 Bus line with three state-buffers.

- Decoders are used to ensure that no more than one control input is active at any giventime
- This circuit can replace the multiplexer in Figure4.3
- To construct a common bus for four registers of *n* bits each using three-state buffers, we need *n* circuits with four buffers ineach
- Only one decoder is necessary to select between the fourregisters

- Designate a memory word by the letterM
- It is necessary to specify the address of M when writing memory transfer operations
- Designate the address register by AR and the data register byDR
- The read operation can be stated as:

  Read: DR ←M[AR]
- The write operation can be stated as:

  Write: M[AR] ←R1

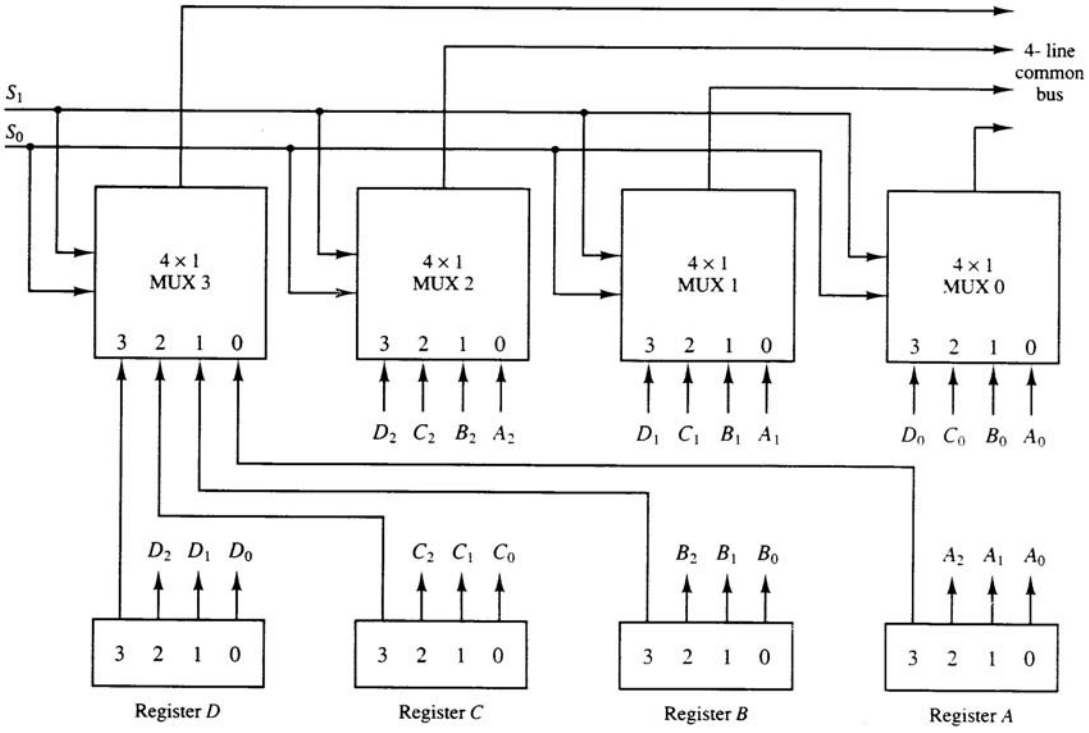
– Arithmetic Microoperations

- There are four categories of the most commonmicrooperations:
  - Register transfer: transfer binary information from one register toanother
  - Arithmetic: perform arithmetic operations on numeric data stored in registers

- Logic: perform bit manipulation operations on non-numeric data storedin registers
- Shift: perform shift operations on data stored inregisters

- The basic arithmetic microoperations are addition, subtraction, increment, decrement, andshift
- Example of addition: R3 ← R1+R2
- Subtraction is most often implemented through complementation andaddition
- Example of subtraction: R3 ← R1 + ~~R2~~ + 1 (strikethrough denotes bar on top – 1's complement ofR2)
- Adding 1 to the 1's complement produces the 2'scomplement
- Adding the contents of R1 to the 2's complement of R2 is equivalent to subtracting

Figure 4-3 Bus system for four registers.



- Multiply and divide are not included asmicrooperations
- A microoperation is one that can be executed by one clockpulse
- Multiply (divide) is implemented by a sequence of add and shift microoperations (subtract andshift)

- To implement the add microoperation with hardware, we need the registers that hold the data and the digital component that performs theaddition
- A full-adder adds two bits and a previouscarry

- A *binary adder* is a digital circuit that generates the arithmetic sum of two binary numbers of anylength
- A binary added is constructed with full-adder circuits connected incascade
- An *n*-bit binary adder requires *n*full-adders

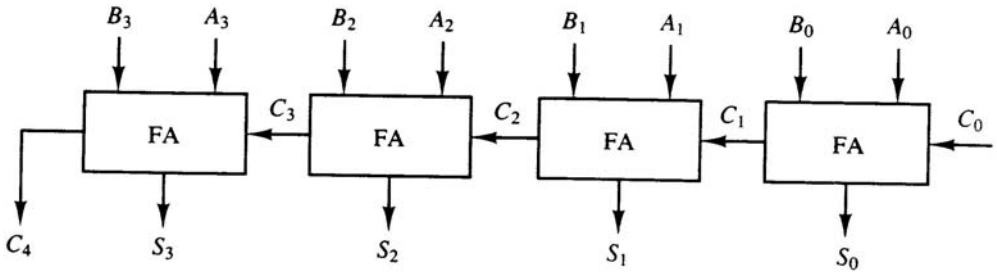


Figure 4-6 4-bit binary adder.

- The subtraction A-B can be carried out by the followingsteps
  - Take the 1's complement of B (invert eachbit)
  - Get the 2's complement by adding1
  - Add the result toA
- The addition and subtraction operations can be combined into one common circuit by including an XOR gate with eachfull-adder

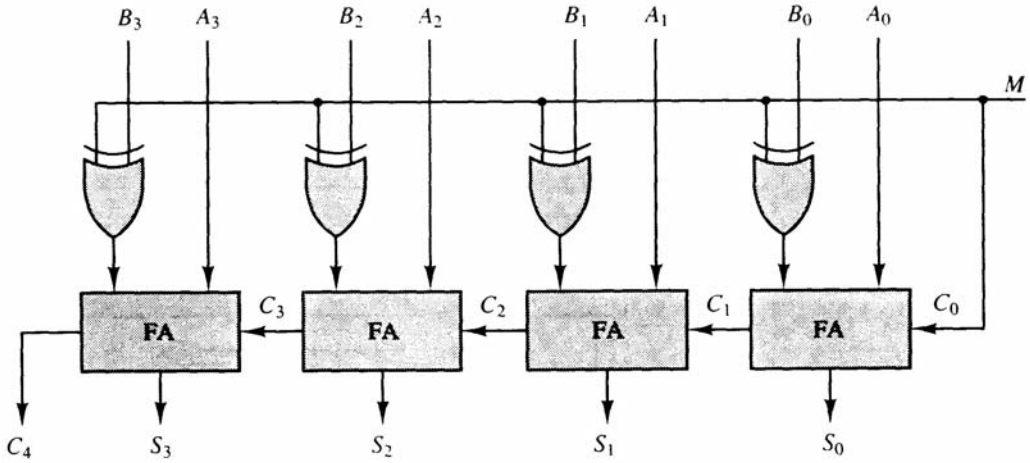


Figure 4-7 4-bit adder-subtractor.

- The increment microoperation adds one to a number in aregister
- This can be implemented by using a binary counter – every time the count enable is active, the count is incremented byone
- If the increment is to be performed independent of a particular register, then use half-adders connected incascade

- An *n*-bit binary incrementer requires *n*half-adders

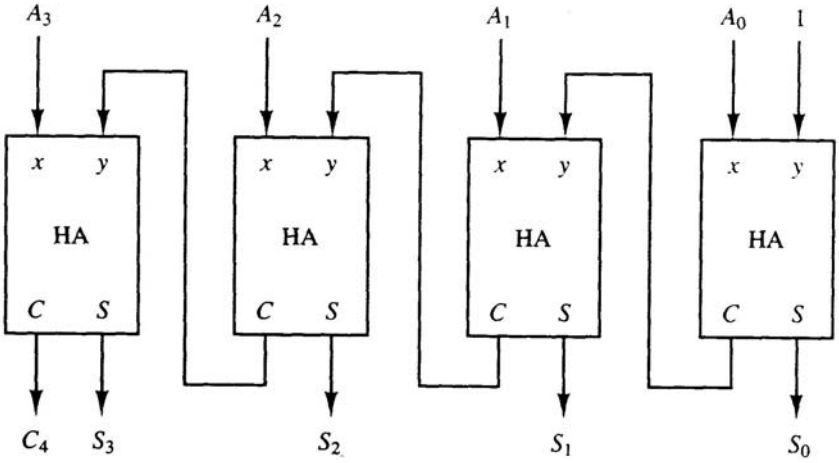


Figure 4-8 4-bit binary incrementer.

- Each of the arithmetic microoperations can be implemented in one composite arithmeticcircuit
- The basic component is the paralleladder
- Multiplexers are used to choose between the differentoperations
- The output of the binary adder is calculated from the followingsum:

$$D = A + Y + C_{in}$$

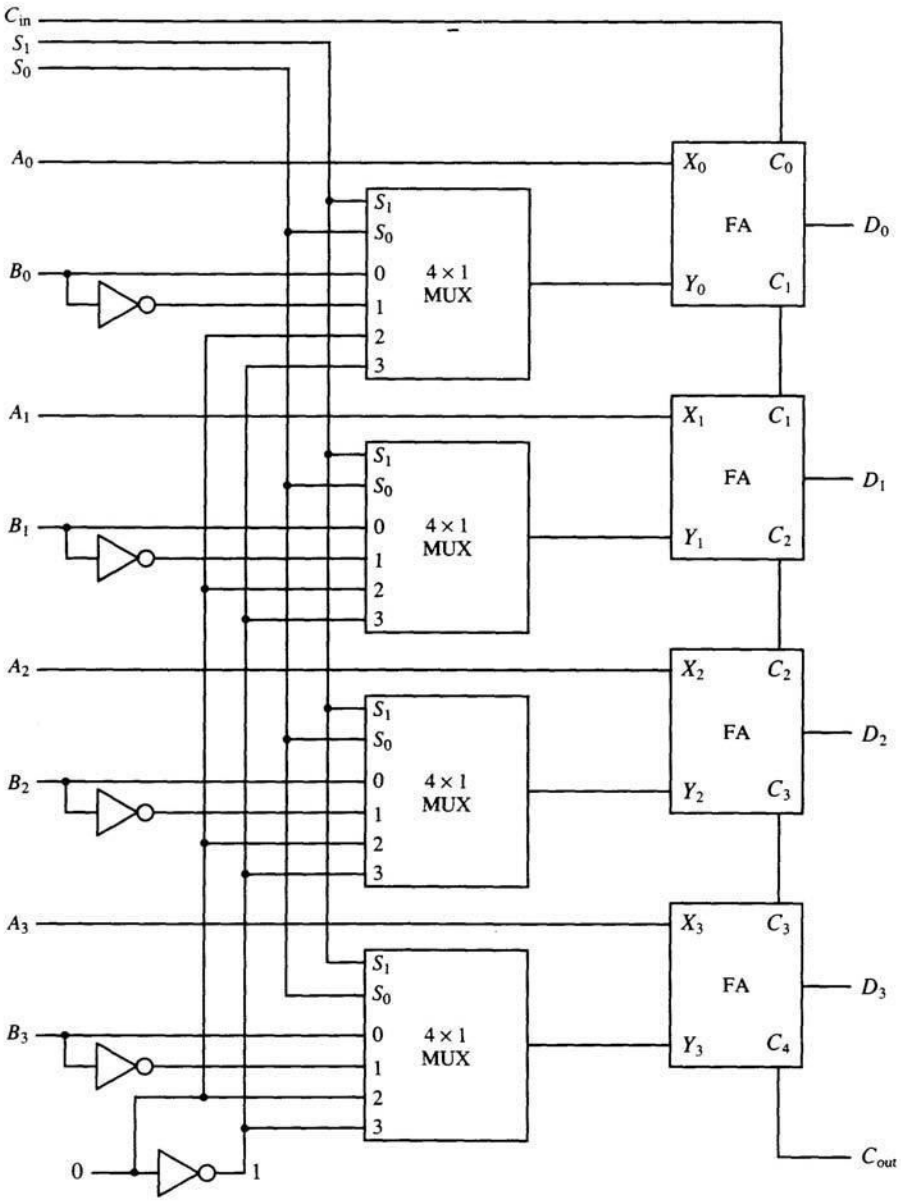


Figure 4-9 4-bit arithmetic circuit.

TABLE 4-4 Arithmetic Circuit Function Table

Select			Input	Output	
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

– Logic Microoperations

- Logic operations specify binary operations for strings of bits stored in registers and treat each bitseparately
- Example: the XOR of R1 and R2 is symbolizedby

  P: R1 ←R1⊕ R2
- Example: R1 = 1010 and R2 =1100

  1010 Content of R1
  1100 Content of R2
  0110 Content of R1 after P =1

- Symbols used for logicalmicrooperations:
  - OR: ∨
  - AND: ∧
  - XOR: ⊕

- The + sign has two different meanings: logical OR andsummation
- When + is in a microoperation, thensummation
- When + is in a control function, thenOR
- Example:

  P + Q: R1 ← R2 + R3, R4 ← R5 ∨ R6
- There are 16 different logic operations that can be performed with two binary variables

TABLE 4-5 Truth Tables for 16 Functions of Two Variables

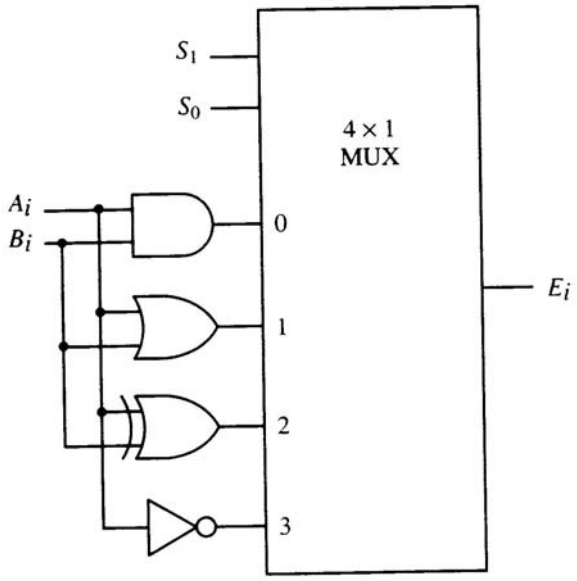
x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

TABLE 4-6 Sixteen Logic Microoperations

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

- The hardware implementation of logic microoperations requires that logic gates be inserted for each bit or pair of bits in theregisters
- All 16 microoperations can be derived from using four logicgates

Figure 4-10 One stage of logic circuit.



(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

(b) Function table

- Logic microoperations can be used to change bit values, delete a group of bits, or insert new bit values into aregister
- The *selective-set* operation sets to 1 the bits in A where there are corresponding 1's inB

  1010 Abefore
  1100 B (logic operand)
  1110 Aafter

  A ←A∨ B

- The *selective-complement* operation complements bits in A where there are corresponding 1's inB

  1010 Abefore
  1100 B (logic operand)
  0110 Aafter

  A ←A⊕ B

- The *selective-clear* operation clears to 0 the bits in A only where there are corresponding 1's inB

  1010 Abefore
  1100 B (logic operand)
  0010 Aafter

$A \leftarrow A \wedge \overline{B}$

- The *mask* operation is similar to the selective-clear operation, except that the bits of A are cleared only where there are corresponding 0's inB

  1010 Abefore
  1100 B (logic operand)
  1000 Aafter

  $A \leftarrow A \wedge B$

- The *insert* operation inserts a new value into a group ofbits
- This is done by first masking the bits to be replaced and then Oring them with the bits to beinserted

  01101010 Abefore
  0000 1111 B(mask)
  0000 1010 A aftermasking

  00001010 Abefore
  1001 0000 B (insert)
  1001 1010 A afterinsertion

- The *clear* operation compares the bits in A and B and produces an all 0's result if the two number areequal

  1010 A
  1010 B

  0000 $A \leftarrow A \oplus B$

– Shift Microoperations

- Shift microoperations are used for serial transfer ofdata
- They are also used in conjunction with arithmetic, logic, and other data-processing operations
- There are three types of shifts: logical, circular, andarithmetic
- A *logical shift* is one that transfers 0 through the serialinput
- The symbols *shl*and *shr*are for logical shift-left and shift-right by one position

  R1 ←shlR1

- The *circular shift* (aka rotate) circulates the bits of the register around thetwo ends without loss ofinformation
- The symbols *cil*and *cir*are for circular shift left andright

TABLE 4-7 Shift Microoperations

Symbolic designation	Description
$R \leftarrow \text{shl } R$	Shift-left register R
$R \leftarrow \text{shr } R$	Shift-right register R
$R \leftarrow \text{cil } R$	Circular shift-left register R
$R \leftarrow \text{cir } R$	Circular shift-right register R
$R \leftarrow \text{ashl } R$	Arithmetic shift-left R
$R \leftarrow \text{ashr } R$	Arithmetic shift-right R

- The *arithmetic shift* shifts a signed binary number to the left orright
- To the left is multiplying by 2, to the right is dividing by2
- Arithmetic shifts must leave the sign bitunchanged
- A sign reversal occurs if the bit in R_{n-1} changes in value after theshift
- This happens if the multiplication causes anoverflow
- An overflow flip-flop V_s can be used to detect the overflow

$$V_s = R_{n-1} \oplus R_{n-2}$$

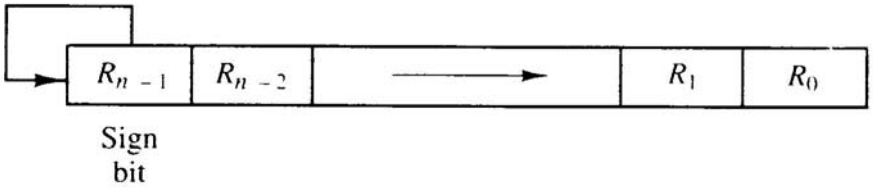
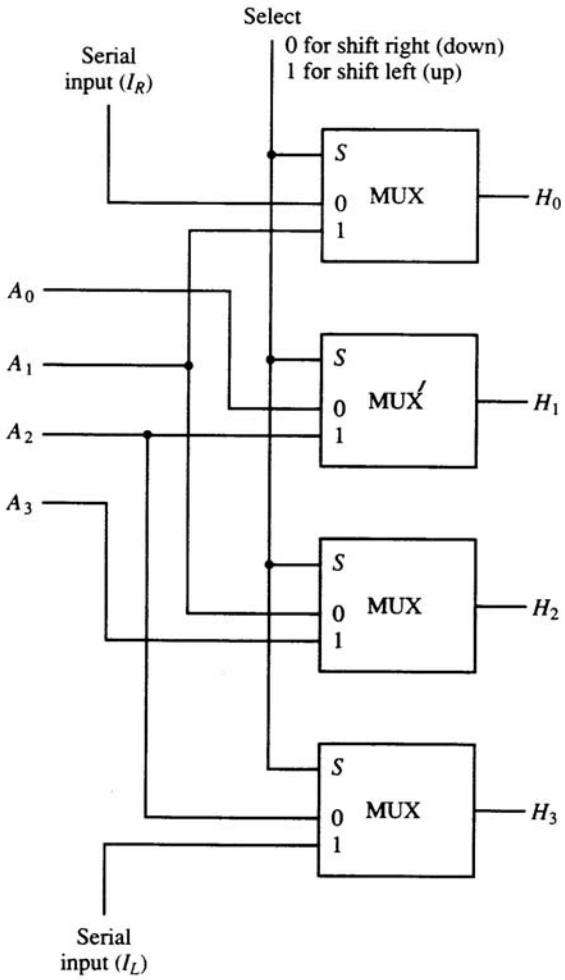


Figure 4-11 Arithmetic shift right.

- A bi-directional shift unit with parallel load could be used to implementthis
- Two clock pulses are necessary with this configuration: one to load the value and another toshift
- In a processor unit with many registers it is more efficient to implement the shift operation with a combinationalcircuit
- The content of a register to be shifted is first placed onto a common bus and the output is connected to the combinational shifter, the shifted number is then loaded back into theregister
- This can be constructed withmultiplexers



Function table

Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

Figure 4-12 4-bit combinational circuit shifter.

– Arithmetic Logic Shift Unit

- The *arithmetic logic unit (ALU)* is a common operational unit connected toa number of storageregisters
- To perform a microoperation, the contents of specified registers are placed in the inputs of theALU
- The ALU performs an operation and the result is then transferred to a destination register
- The ALU is a combinational circuit so that the entire register transfer operation from the source registers through the ALU and into the destination register can be performed during one clock pulseperiod

Figure 4-13 One stage of arithmetic logic shift unit.

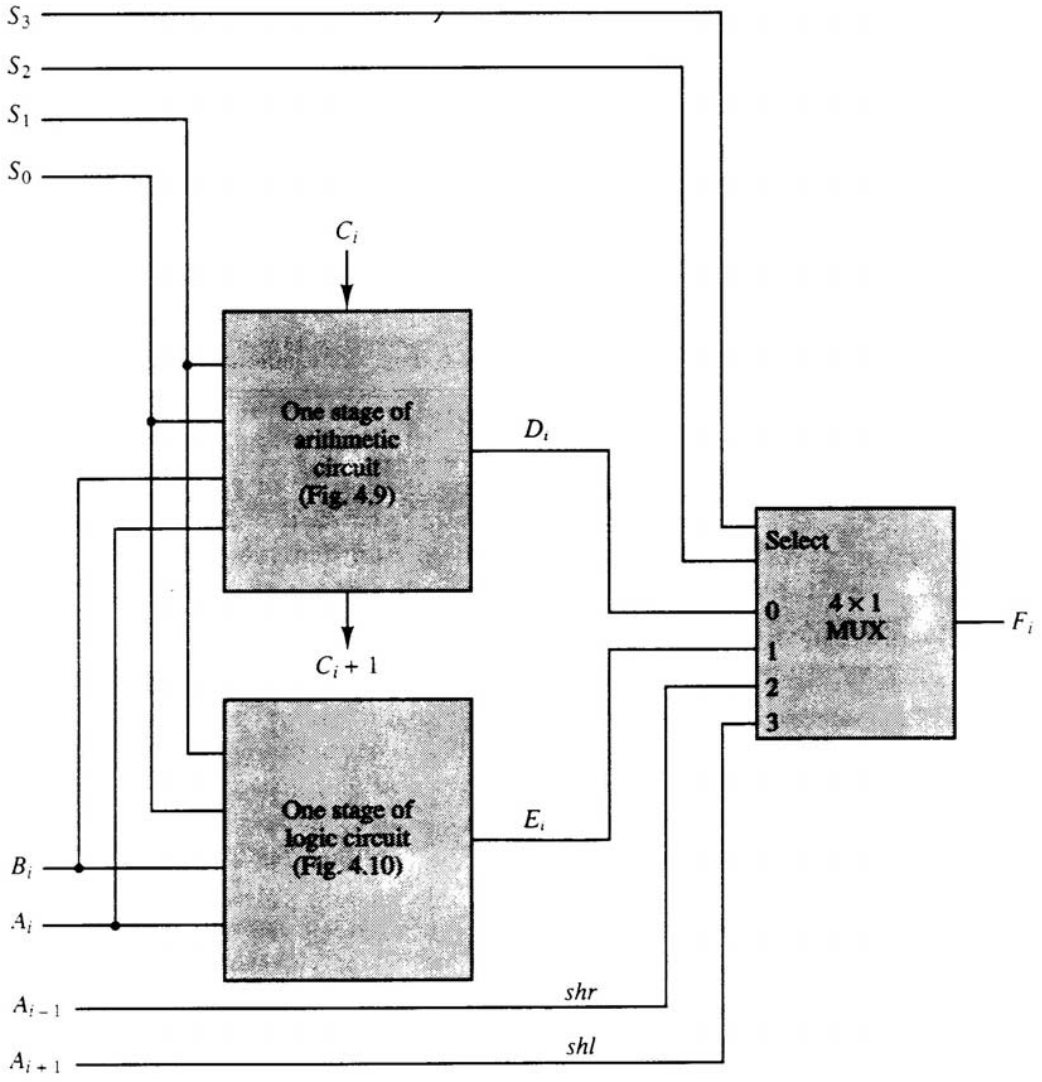


TABLE 4-8 Function Table for Arithmetic Logic Shift Unit

Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	$F = \text{shr } A$	Shift right A into F
1	1	×	×	×	$F = \text{shl } A$	Shift left A into F