EC3352 DIGITAL SYSTEMS DESIGN

L T P C
3 0 2 4

COURSE OBJECTIVES

- To present the fundamentals of digital circuits and simplification methods
- To practice the design of various combinational digital circuits using logic gates
- To bring out the analysis and design procedures for synchronous and asynchronous Sequential circuits
- To learn integrated circuit families.
- To learn integrated circuit families.
- To introduce semiconductor memories and related technology

UNIT I BASIC CONCEPTS 9

Review of number systems-representation-conversions, Review of Boolean algebra- theorems, sum of product and product of sum simplification, canonical forms min term and max term, Simplification of Boolean expressions-Karnaugh map, completely and incompletely specified functions, Implementation of Boolean expressions using universal gates ,Tabulation methods.

UNIT II COMBINATIONAL LOGIC CIRCUITS 9

Problem formulation and design of combinational circuits - Code-Converters, Half and Full Adders, Binary Parallel Adder – Carry look ahead Adder, BCD Adder, Magnitude Comparator, Decoder, Encoder, Priority Encoder, Mux/Demux, Case study: Digital trans-receiver / 8 bit Arithmetic and logic unit, Parity Generator/Checker, Seven Segment display decoder.

UNIT III SYNCHRONOUS SEQUENTIAL CIRCUITS 9

Latches, Flip flops – SR, JK, T, D, Master/Slave FF, Triggering of FF, Analysis and design of clocked sequential circuits – Design - Moore/Mealy models, state minimization, state assignment,lock - out condition circuit implementation - Counters, Ripple Counters, Ring Counters, Shift registers, Universal Shift Register. Model Development: Designing of rolling display/real time clock.

UNIT IV ASYNCHRONOUS SEQUENTIAL CIRCUITS 9

Stable and Unstable states, output specifications, cycles and races, state reduction, race free assignments, Hazards, Essential Hazards, Fundamental and Pulse mode sequential circuits, Design of Hazard free circuits.

UNIT V LOGIC FAMILIES AND PROGRAMMABLE LOGIC DEVICES 9

Logic families- Propagation Delay, Fan - In and Fan - Out - Noise Margin - RTL ,TTL,ECL, CMOS - Comparison of Logic families - Implementation of combinational logic/sequential logic design using standard ICs, PROM, PLA and PAL, basic memory, static ROM,PROM,EPROM,EEPROM EAPROM.

45 PERIODS

PRACTICAL EXERCISES

30 PERIODS

1. Design of adders and subtractors & code converters.
2. Design of Multiplexers & Demultiplexers.
3. Design of Encoders and Decoders.
4. Design of Magnitude Comparators
5. Design and implementation of counters using flip-flops
6. Design and implementation of shift registers.

COURSE OUTCOMES

At the end of the course the students will be able to

CO1: Use Boolean algebra and simplification procedures relevant to digital logic.

CO2: Design various combinational digital circuits using logic gates.

CO3:Analyse and design synchronous sequential circuits.

CO4: Analyse and design asynchronous sequential circuits. .

CO5: Build logic gates and use programmable devices

TOTAL:75 PERIODS

TEXTBOOKS

1. M. Morris Mano and Michael D. Ciletti, 'Digital Design', Pearson, 5th Edition, 2013.(Unit – I-V)

REFERENCES

1. Charles H. Roth, Jr, 'Fundamentals of Logic Design', Jaico Books, 4th Edition, 2002.
2. William I. Fletcher, "An Engineering Approach to Digital Design", Prentice- Hall of India, 1980.
3. Floyd T.L., "Digital Fundamentals", Charles E. Merril publishing company,1982.
4. John. F. Wakerly, "Digital Design Principles and Practices", Pearson Education, 4 th Edition,2007.

Review of Number Systems

Numeral system is a writing system for expressing numbers, that is, a mathematical notation for representing numbers of a given set, using digits or other symbols in a consistent manner. The same sequence of symbols may represent different numbers in different numeral systems.

Number system plays an important role in digital systems. Number system is a mathematical object used to count, label and measure the task performed by the logic function.

1. Decimal Number System
2. Binary Number System
3. Octal Number System
4. Hexadecimal Number System

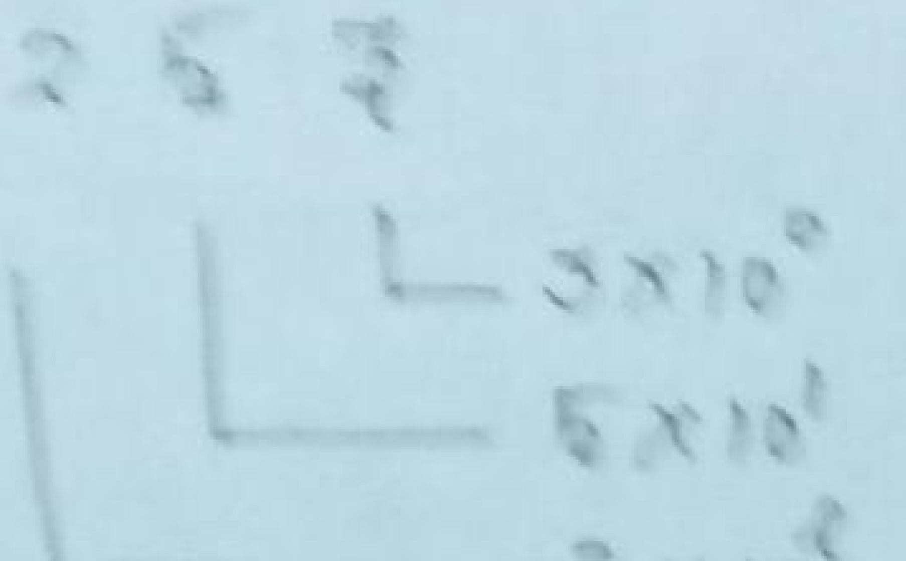
Number System Representation

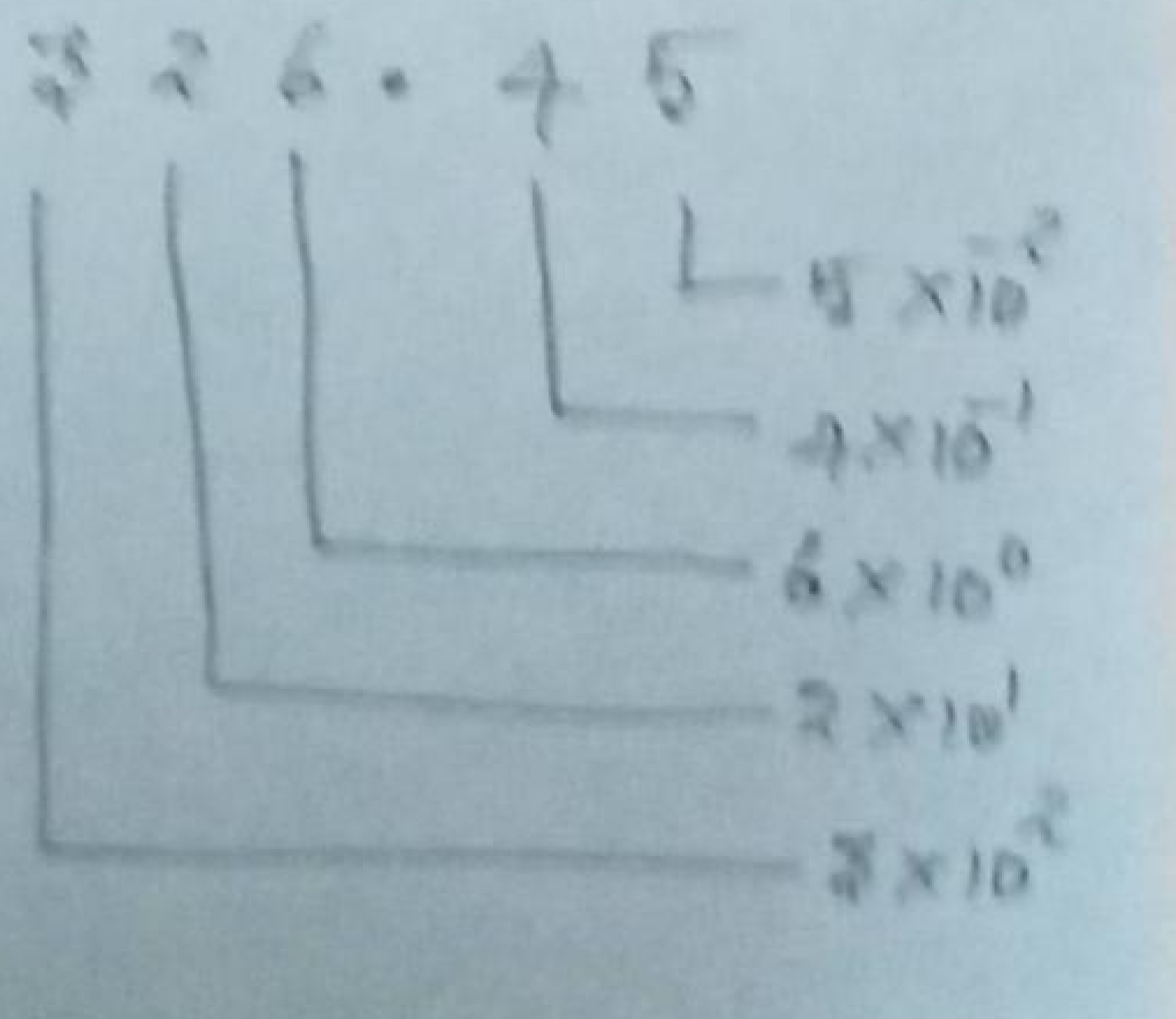
Decimal Number System

The number system which uses ten distinct digits 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 is known as decimal number system. It is also called as base 10 system.

- * Decimal number system has a base or radix of 10 $[(853)_{10}]$.
- * Each of the ten decimal digits, 0 through 9, has a place value or weights depending on its position.
- * The weights are units, tens, hundreds, thousands and so on ...

Ex $(257)_{10}$, $(726.45)_{10}$





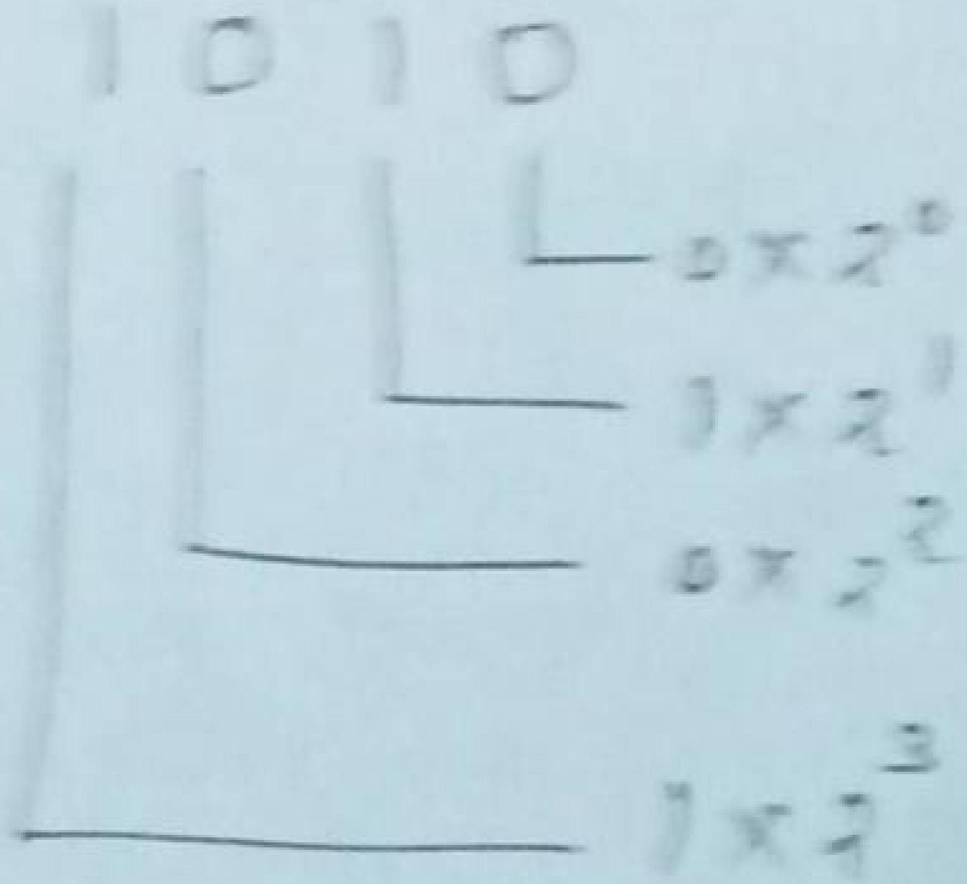
Binary Number System

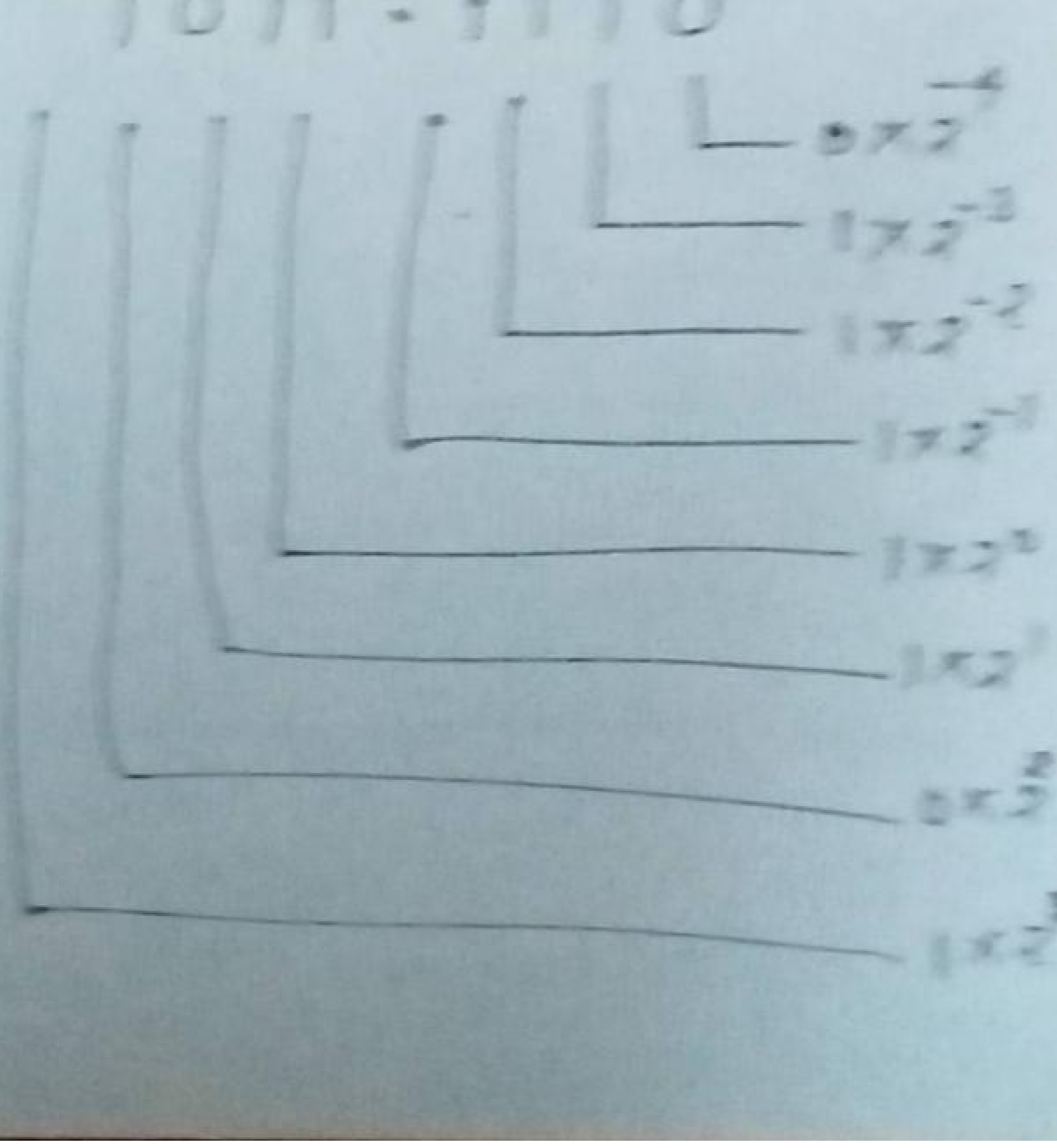
A number system that uses only two digits 0 and 1 is called a binary number system. The binary system is also called a base two system.

* The (symbol) number 0 and 1 are known as bits

* The weights or place value of each position can be expressed in terms of power of 2 $[2^0, 2^1, 2^2, ----]$.

Ex: $(1010)_2$, $(1011.1110)_2$





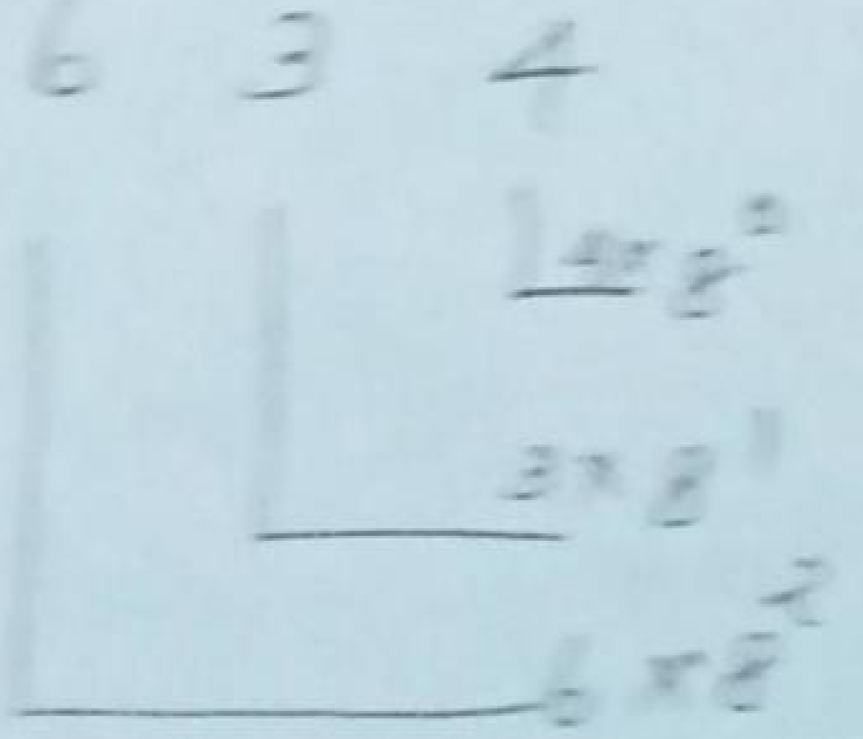
Octal Number System

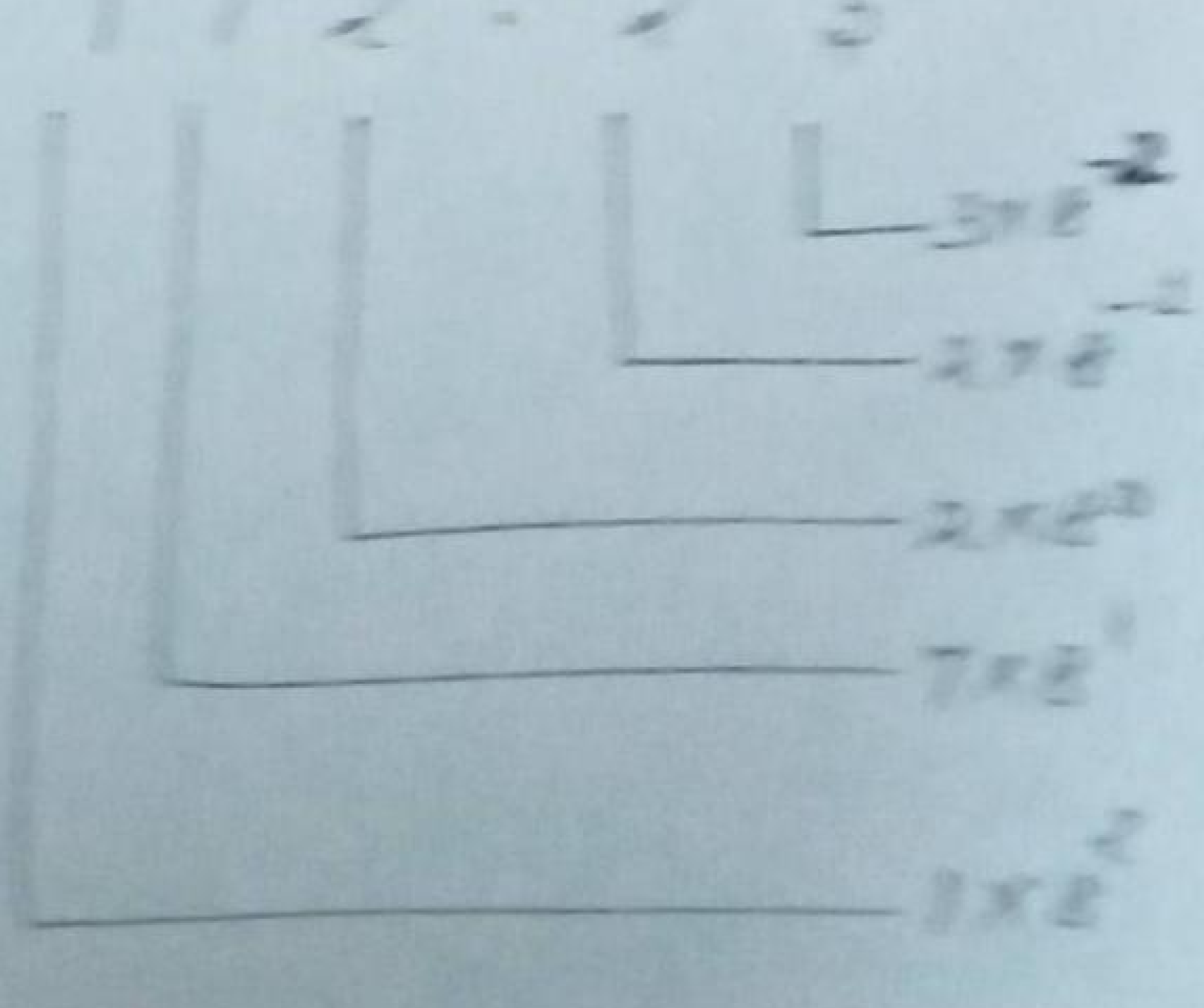
The number system which uses eight digits 0, 1, 2, 3, 4, 5, 6 and 7 is called an octal number system. It is also called as base eight system.

- Sets of 3-bit binary numbers can be represented by octal numbers $[\underset{7}{111}\ \underset{5}{101}]$.
- The weights or place value of each position can be represented in terms of power of 8 $[8^0, 8^1, 8^2, \ldots]$.

Ex:

$(634)_8$, $(172.23)_8$





Hexadecimal Number System

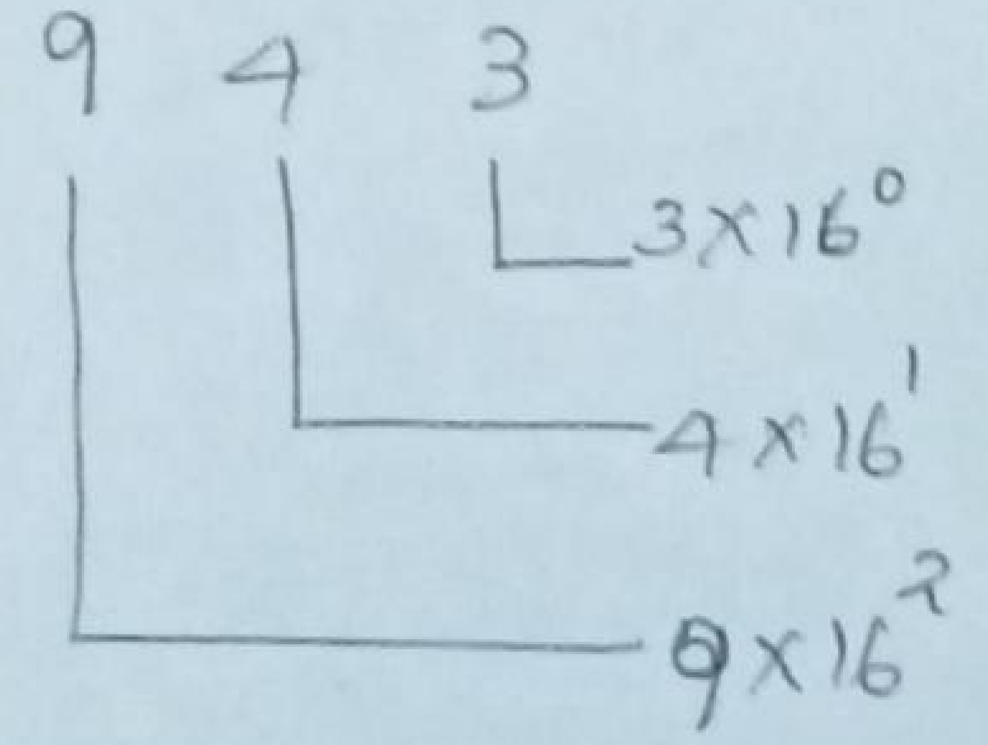
A number system that uses sixteen digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D and E is known as hexadecimal number system. It is also called as base sixteen system.

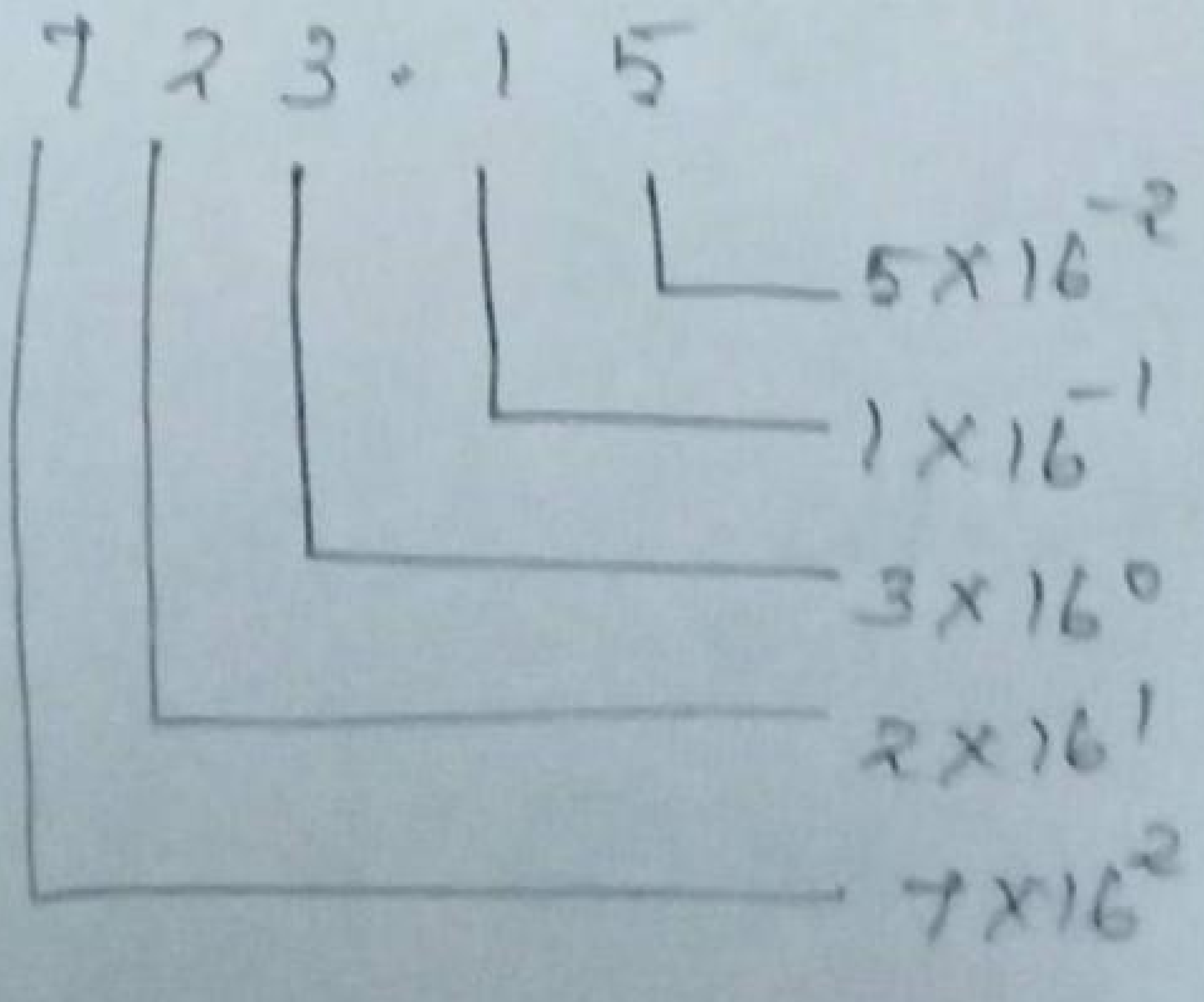
- * sets of 4-bit binary numbers can be represented by hexadecimal number $[\underbrace{1010}_{A}, \underbrace{0011}_{3}]$
- * The weights or place value of each position can be represented in terms of power of 16 $[16^0, 16^1, 16^2, \ldots]$.

10 - A - 1010   12 - 1100 - C   14 - D - 1110

11 - B - 1011   13 - 1101 - D   15 - E - 1111

Ex: $(943)_{16}$, $(723 \cdot 15)_{16}$





Number System - Conversions

Decimal to Binary, Octal and Hexadecimal Conversions

Convert the following decimal numbers to Binary, Octal and Hexadecimal numbers.

i) $(455)_{10}$ ii) $(256.22)_{10}$

* The conversion of decimal to any number is by dividing the decimal number with the respective number system.

Decimal to Binary

i) $(455)_{10}$

```
2|455
 2|227 -1
 2|113 -1
  2|56 -1        ^ Top
  2|28 -0        |
  2|14 -0        |
   2|7 -0        |
   2|3 -1        | to
      1 -1       |
    -------------
                 Bottom
```

$(455)_{10} = (111000111)_2$

ii) $(256.22)_{10}$

2 | 256
2 | 128 – 0
2 | 64 – 0
2 | 32 – 0
2 | 16 – 0
2 | 8 – 0
2 | 4 – 0
2 | 2 – 0
1 – 0

Bottom to Top

	Integer
$.22 \times 2 = 0.44$	0
$.44 \times 2 = 0.88$	0
$.88 \times 2 = 1.76$	1
$.76 \times 2 = 1.52$	1

Top to Bottom

$(256.22) = (100000000.0011\cdots)_2$

Decimal to Octal

i) $(455)_{10}$

8 | 455
8 | 56 – 7
7 – 0

$(455)_{10} = (707)_8$

ii) $(256.22)_{10}$

8 | 256
8 | 32 – 0
4 – 0

	Integer
$.22 \times 8 = 1.76$	1
$.76 \times 8 = 6.08$	6
$.08 \times 8 = 0.64$	0
$.64 \times 8 = 5.12$	5

$(256.22)_{10} = (400.1605)_8$

Decimal to Hexadecimal

i) $(455)_{10}$

16 | 455
16 | 28 – 7
   1 – 12(C)

$(455)_{10} = (1C7)_{16}$

ii) $(256.22)_{10}$

16 | 256
16 | 16 – 0
   1 – 0

	Integer
.22 × 16 = 3.52	3
.52 × 16 = 8.32	8
.32 × 16 = 5.12	5
.12 × 16 = 1.92	1

$(256.22)_{10} = (100.3851)_{16}$

Binary to Decimal, Octal and Hexadecimal Conversion

* The conversion of binary to any number is by multiplying the binary number with binary number system along with each positional weight or place value $[2^0, 2^1, 2^2, \ldots]$.

Convert the following binary number to decimal, octal and hexadecimal. i) $(11101)_2$ ii) $(100010.011)_2$

Binary to Decimal

i) $(11101)_2$

1 1 1 0 1

$1 \times 2^0 = 1$

$0 \times 2^1 = 0$

$1 \times 2^2 = 4$

$1 \times 2^3 = 8$

$1 \times 2^4 = 16$

$\overline{29}$

$(11101)_2 = (29)_{10}$

ii) $(100010.011)_2$

$1\ 0\ 0\ 0\ 1\ 0\ .\ 0\ 1\ 1$

$1 \times 2^{-3} = 0.125$

$1 \times 2^{-2} = 0.25$

$0 \times 5^{-1} = 0.0$

$0 \times 2^{0} = 0.0$

$1 \times 2^{1} = 2.0$

$0 \times 2^{2} = 0.0$

$0 \times 2^{3} = 0.0$

$0 \times 2^{4} = 0.0$

$1 \times 2^{5} = 32.0$

$\underline{\overline{34.375}}$

$(100010.011)_2 = (34.375)_{10}$

Binary to Octal

i) $(11101)_2$

- * The octal number is a set of 3-bit binary number
- * ∴ The given binary number is splits into 3-bits from right side

$$11101 \Rightarrow \underbrace{011}_{3}\ \underbrace{101}_{5}$$

$$(11101)_2 = (35)_8$$

ii) $(100010.011)_2$

$$100010.011 \Rightarrow \overset{\longleftarrow}{\underbrace{100}_{4}\ \underbrace{010}_{2}}\,.\,\overset{\longrightarrow}{\underbrace{011}_{3}}$$

$$(100010.011)_2 = (42.3)_8$$

Binary to Hexadecimal

i) $(11101)_2$

* The hexadecimal number is a set of 4-bit binary number

* ∴ split the given binary number into 4-bit from right side

$$11101 \Rightarrow \underbrace{0001}_{1}\ \underbrace{1101}_{D}$$

$$(11101)_2 = (1D)_{16}$$

ii) $(100010.011)_2$

$$100010.011 \Rightarrow \overset{\longleftarrow}{\underbrace{10}\,\underbrace{0010}}\,.\,\overset{\longrightarrow}{\underbrace{011}}$$

$$\Downarrow$$

$$\underbrace{0010}_{2}\ \underbrace{0010}_{2}\,.\,\underbrace{0110}_{6}$$

$$(100010.011)_2 = (22.6)_{16}$$

Octal to Decimal, Binary and Hexadecimal Conversion

Convert the following octal number to decimal, binary and hexadecimal.

i) $(35)_8$ (ii) $(42.3)_8$

Octal to Decimal

i) $(35)_8$

3 5

$5 \times 8^0 = 5$

$3 \times 8^1 = 24$

$\overline{29}$

$(35)_8 = (29)_{10}$

ii) $(42.3)_8$

4 2 . 3

$3 \times 8^{-1} = 0.375$

$2 \times 8^0 = 2.000$

$4 \times 8^1 = 32.000$

$\overline{34.375}$

$(42.3)_8 = (34.375)_{10}$

Octal to Binary

* Octal number is a set of 3-bit binary number

* Therefore each digit in an octal number consists of 3-bits of binary number

i) $(35)_8$

$$3\ 5 \Rightarrow \underbrace{\overset{3}{\overset{\downarrow}{011}}}\ \underbrace{\overset{5}{\overset{\downarrow}{101}}}$$

$$(35)_8 = (011101)_2$$

ii) $(42.3)_8$

$$4\ 2\ .\ 3 \Rightarrow \overset{4}{\overset{\downarrow}{100}}\ \overset{2}{\overset{\downarrow}{010}}\ .\ \overset{3}{\overset{\downarrow}{011}}$$

$$(42.3)_8 = (100010.011)_2$$

Octal to Hexadecimal

* There is no direct conversion between octal to hexadecimal and hexadecimal to octal

* So, convert the octal number to binary and then convert binary to hexadecimal

i) $(35)_8$

$35 \Rightarrow \overset{3 \downarrow}{011}\,\overset{5 \downarrow}{101}$ → octal to binary

$011101 \Rightarrow \underbrace{01}\ \underbrace{1101}$ → binary to hexadecimal

$\Downarrow$

$\underbrace{0001}_{1}\ \underbrace{1101}_{D}$

$(35)_8 = (1D)_{16}$

ii) $(42.3)_8$

$42.3 \Rightarrow \overset{4 \downarrow}{100}\,\overset{2 \downarrow}{010} . \overset{3 \downarrow}{011}$

$100010.011 \Rightarrow \underbrace{\_\_10}\ \underbrace{0010} . \underbrace{011\_}$

$\Downarrow$

$\underbrace{0010}_{2}\ \underbrace{0010}_{2} . \underbrace{0110}_{6}$

$(42.3)_8 = (22.6)_{16}$

Hexadecimal to Decimal, Binary and Octal Conversion

Convert the following hexadecimal number to decimal, binary and octal.

i) $(1D)_{16}$ ii) $(22.6)_{16}$

Hexadecimal to Decimal

i) $(1D)_{16}$

1 D

(D) $13 \times 16^0 = 13$

$1 \times 16^1 = 16$

$\overline{\quad 29 \quad}$

$(1D)_{16} = (29)_{10}$

ii) $(22.6)_{16}$

2 2 . 6

$6 \times 16^{-1} = 0.375$

$2 \times 16^0 = 2.000$

$2 \times 16^1 = 32.000$

$\overline{\quad 34.375 \quad}$

$(22.6)_{16} = (34.375)_{10}$

Hexadecimal to Binary

i) $(1D)_{16}$

* Hexadecimal is a set of 4-bits binary number
* Therefore each digits in a hexadecimal has 4-bits of binary number

$$1\ D \Rightarrow \overset{1}{\overset{\downarrow}{0001}}\ \overset{D(13)}{\overset{\downarrow}{1101}}$$

$$(1D)_{16} = (00011101)_2$$

ii) $(22.6)_{16}$

$$2\ 2 \cdot 6 \Rightarrow \overset{2}{\overset{\downarrow}{0010}}\ \overset{2}{\overset{\downarrow}{0010}} \cdot \overset{6}{\overset{\downarrow}{0110}}$$

$$(22.6)_{16} = (00100010.0110)_2$$

Hexadecimal to Octal

- * There is no direct conversion between hexadecimal to octal
- * So, convert the hexadecimal number to binary and then convert binary to octal

i) $(1D)_{16}$

$$1\ D \Rightarrow \overset{1}{0001}\ \overset{D(13)}{1101}$$ → hexadecimal to binary

$$00011101 \Rightarrow -\underbrace{00}\,\underbrace{011}\,\underbrace{101}$$ → binary to octal

$$\Downarrow$$

$$\underbrace{000}_{0}\ \underbrace{011}_{3}\ \underbrace{101}_{5}$$

$$(1D)_{16} = (35)_8$$

ii) $(22.6)_{16}$

$$22.6 \Rightarrow \overset{2}{0010}\ \overset{2}{0010}\ .\ \overset{6}{0110}$$

$$00100010.0110 \Rightarrow \underset{\longleftarrow}{-\underbrace{001}\,\underbrace{000}\,\underbrace{010}}\,.\,\underset{\longrightarrow}{\underbrace{011}\,\underbrace{0--}}$$

$$\Downarrow$$

$$\underbrace{000}_{0}\ \underbrace{100}_{4}\ \underbrace{010}_{2}\,.\,\underbrace{011}_{3}\ \underbrace{000}_{\cancel{0}}$$ no need

$$(22.6)_{16} = (42.3)_8$$

Review of Boolean Algebra

Boolean algebra is a branch of mathematics that deals with operations on logical values with binary variables

* The Boolean variables are represented as binary numbers to represent truths (states): 1 = true (ON/High) and 0 = false (OFF/Low).

* Elementary algebra deals with numerical operations whereas Boolean algebra deals with logical operations.

* Elementary algebra is expressed using basic mathematical functions such as addition, subtraction, multiplication and division whereas Boolean algebra is expressed using basic notations such as conjunction, disjunction and negation.

* Boolean algebra is used to simplify the design of logic circuits. But, this method involves lengthy mathematical operations.

Boolean Logic Operations

Boolean function is an algebraic expression formed using binary constants, binary variables and basic operation symbols. Basic logical operations include the AND function (logical multiplication), the OR function (logical addition) and the NOT function (logical complementation).

Boolean function can be converted into a logic diagram composed of the AND, OR and NOT (inverter) gates.

1. Logical AND operation

The logical AND operation of two Boolean variables A and B, given as $Y = A \cdot B$. The common symbol for this operation is the multiplication sign(.).

The result of the AND operation on the variables A and B is logical '0' for all cases, except when both A and B are logical '1'. Usually, the dot denoting the AND function is omitted and A.B is written as AB.

Inputs		Output
A	B	$Y = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

2. **Logical OR Operation**

The logical OR operation between two Boolean variables A and B, gives as $Y = A+B$. The common symbol used for this logical addition operation is the plus sign(+). The result of the OR operation on the variables A and B is logical 1 when A or B (or both A, B) are logical 1.

Inputs		Output
A	B	$Y = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

3. **Logical Complementation (Inversion) / NOT Operation**

The logical inverse operation converts the logical 1 to the logical 0 and vice versa. This method is also called the NOT operation. The symbol used for this operation is a bar(—) over the function or the variable.

Input	Output
A	$Y = \overline{A}$
0	1
1	0

Several notations, such as adding an asterisk(*), a star (★), prime('), etc. over the variable, are used to indicate the NOT operation. "NOT A" or the complement of 'A' is represented by $\overline{A}$.

Basic Laws of Boolean Algebra

Logical operations can be expressed and minimized mathematically using the rules, laws and theorems of Boolean algebra. It is a convenient and systematic method of expressing and analyzing the operation of digital circuits and systems.

Boolean algebra uses binary arithmetic variables which have two distinct symbols '0' and '1'. These are called levels or states of logic.

- * Binary 1 represents - High Level
- * Binary 0 represents - Low Level

Three basic laws of Boolean algebra,

1. Commutative Law
2. Associative Law
3. Distributive Law

Commutative Law

1. $A+B = B+A$
2. $A \cdot B = B \cdot A$

According to this property, the order of the OR and AND operation conducted on the variables makes no difference.

Proof:

$$A \cdot B = B \cdot A$$

A	B	A·B
0	0	0
0	1	0
1	0	0
1	1	1

=

A	B	B·A
0	0	0
0	1	0
1	0	0
1	1	1

$$A + B = B + A$$

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	B+A
0	0	0
0	1	1
1	0	1
1	1	1

Associative Law

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

According to this law, it makes no difference in what order the variables are grouped during the OR/AND operation of several variables.

Proof: $A+(B+C) = (A+B)+C$

A	B	C	B+C	A+(B+C)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1

A	B	C	A+B	(A+B)+C
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1

A	B	C	B+C	A+(B+C)
1	0	0	0	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

=

A	B	C	A+B	(A+B)+C
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

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

A	B	C	B·C	A·(B·C)
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

=

A	B	C	A·B	(A·B)·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	0	0
1	1	0	1	0
1	1	1	1	1

Distributive Law

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

2. $A\cdot(B+C) = (A\cdot B)+A\cdot C$

According to property 1, the AND operation of several variables and then the OR operation of the result with a sigle variable is equivalent to the OR operation of the single variable with each of the several variables and then the 'AND' operation of the sums.

According to property 2, the 'OR' operation of several variables and then the 'AND' operation of the result with a single variable is equivalent to the 'AND' operation of the single variable with each of the several variables and then the 'OR' operation of the products.

Proof

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

A	B	C	B·C	A+(B·C)
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

=

A	B	C	A+B	A+C	(A+B)·(A+C)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Additional Laws

Absorption Laws

1. $\boxed{A + AB = A}$

$LHS = A + AB$

$= A(1+B)$

$= A \cdot 1$

$= A$

$= RHS$

Hence proved.

2. $\boxed{A \cdot (A+B) = A}$

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

$= A \cdot A + A \cdot B$

$= A + AB$

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

$= A$

$= RHS$

Hence proved.

3. $\boxed{A + \bar{A}B = A + B}$

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

Based on distributive law,

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

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

$\therefore$ LHS $= A + \bar{A}B$

$= (A+\bar{A})(A+B)$ $[\because A+\bar{A}=1]$

$= 1\cdot(A+B)$

$= A+B$

$=$ RHS

Hence proved.

<u>Consensus Laws</u>

1. $\boxed{AB+\bar{A}C+BC = AB+\bar{A}C}$

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

$= AB+\bar{A}C+BC\cdot 1$

$= AB+\bar{A}C+BC(A+\bar{A})$ $[\because A+\bar{A}=1]$

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

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

$= AB(1+C)+\bar{A}C(1+B)$

$= AB+\bar{A}C$ $[\because 1+C=1, 1+B=1]$

$=$ RHS

Hence proved.

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

LHS $= (A+B)(\bar{A}+C)(B+C+0)$

$= (A+B)(\bar{A}+C)(B+C+A\bar{A})$ $[\because A\cdot\bar{A}=0]$

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

(Distributive Law)
$A+BC = (A+B)(A+C)$

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

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

$$= RHS$$

Hence Proved

$$// \underbrace{(A+B)}_{A}\,(\underbrace{A}_{A}+\underbrace{B+C}_{B})$$

Based on absorption law,

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

$$\therefore (A+B)\left([A+B]+C\right) = A+B$$

$|||^{ly}$ $(\bar{A}+C)\left([\bar{A}+C]+B\right) = \bar{A}+C$ //

SUM OF PRODUCTS

The logical sum of two or more logical product terms is called a sum of products expression. It is basically an OR operation of AND operated variables

Ex:

$Y = AB + BC + AC$

$Y = AB + \bar{A}C + BC$ (SOP)

PRODUCT OF SUMS

The logical product of two or more logical sum terms is called a product of sums expression. It is basically an AND operation of OR operated variables.

Ex:

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

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

Minterm

A product term containing all the 'K' variables of the function in either complemented or uncomplemented form is called a minterm.

1 – uncomplement

0 – complement

The minterms of a n-variable function can be represented by $m_0, m_1, m_2, \ldots m_{n-1}$; the suffix indicates the decimal code corresponding to the minterm combination

Minterm Table [2 variable]

A	B	Minterm
0	0	$\overline{A}\,\overline{B}$
0	1	$\overline{A}\,B$
1	0	$A\,\overline{B}$
1	1	$A\,B$

JVA

Maxterm

A sum containing all the 'n' variables of the function in either complemented or uncomplemental form is called a max term.

1 - complement

0 - uncomplement

The maxterm of a n-variable function can be represented by $M_0, M_1, M_2, \ldots, M_{n-1}$; the suffix indicates the decimal code corresponding to the maxterm combination.

Maxterm Table [2 variable]

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

Implementation of Boolean Expressions using Universal Gates

Boolean algebra is used in describing and simplifying logic circuits. Simplification of Boolean logic expressions is very important because it reduces the hardware required to design a specific system.

The Boolean expression corresponding to a given gate network can be derived by systematically progressing from the input to the output of gates. The gating or logic network can be formed by interconnecting the OR, AND and NOT gates.

Logic Gates

Logic gate is an electronic circuit which makes logical decisions. To arrive at these decisions, the most common logic gates used are OR, AND, NOT, NAND and NOR gates.

Exclusive OR gate (E-OR), Exclusive NOR gate (E-NOR) are another logic gates which can be constructed using basic gates such as AND, OR and NOT gates.

OR Gate

The OR gate performs logical addition, commonly known as OR function. The OR gate has two or more inputs and only one output.

Operation

* Any one or all inputs are High (Logic 1) – Output is High (Logic 1)

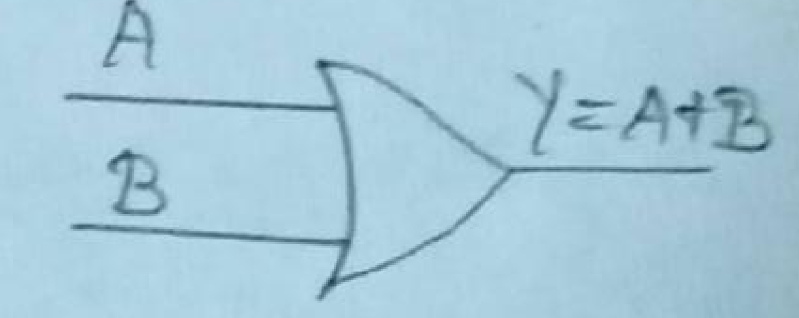
* When all inputs are Low (Logic 0) – Output is Low (Logic 0)

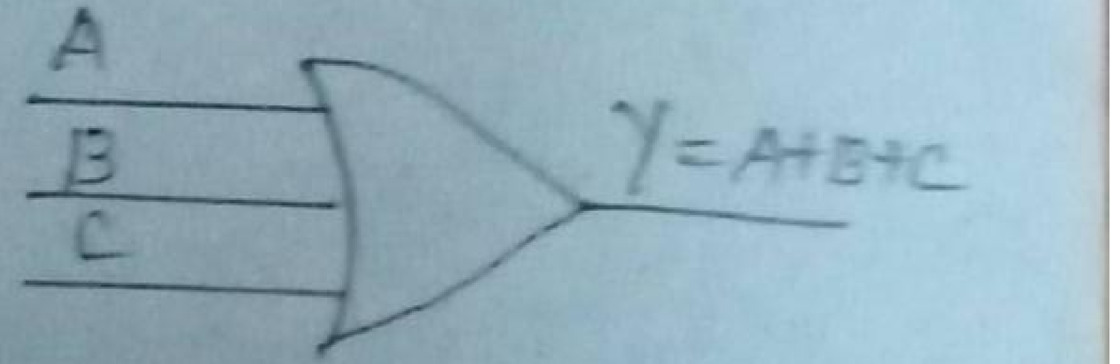
Truth Table

Inputs		Output
A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

Inputs			Output
A	B	C	Y=A+B+C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Logic Symbol





AND Gate

The AND gate performs logical multiplication, commonly known as AND function. The AND gate has two or more inputs and only one output.

Operation

* Only when all the inputs are HIGH (logic 1) } - Output is HIGH (logic 1)

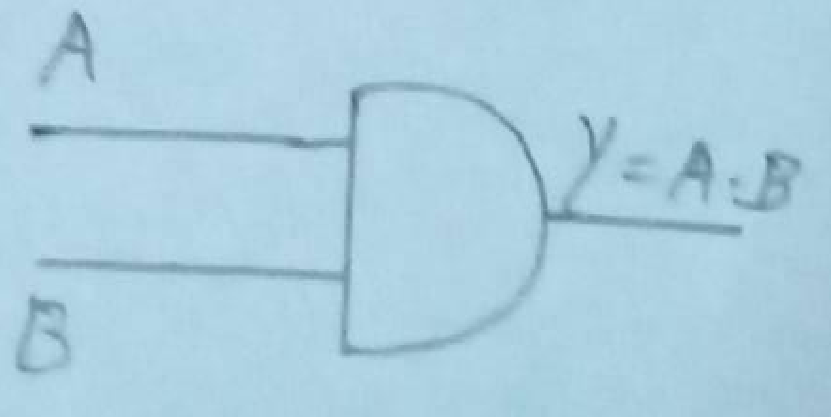
* Any one of the input or all inputs are LOW (Logic 0) } - Output is LOW (logic 0)

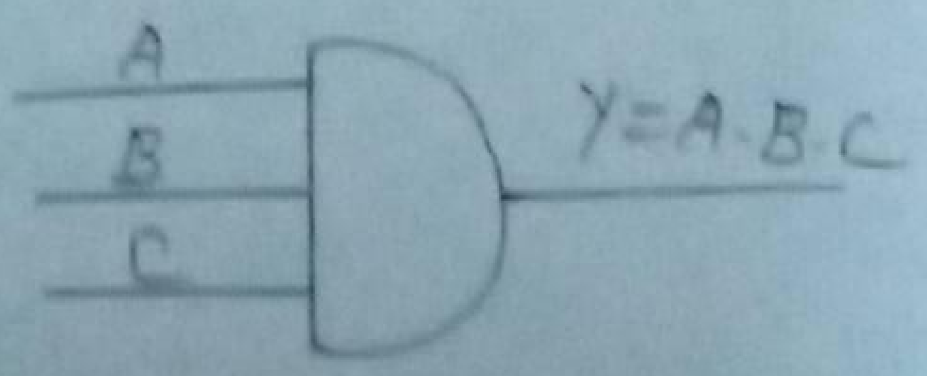
Truth Table

Inputs		Output
A	B	Y = A·B
0	0	0
0	1	0
1	0	0
1	1	1

Inputs			Output
A	B	C	Y = A·B·C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Logic symbol





NOT gate

The NOT gate performs basic logical function called inversion or complementation. The purpose of this gate is to convert one logic level into the opposite logic level. It has one input and one output.

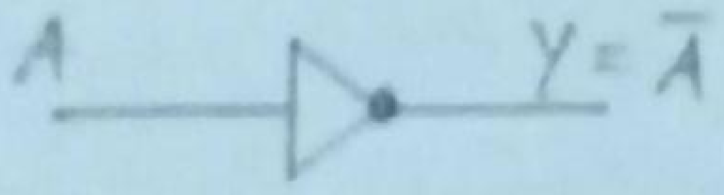
Operation

- * Input HIGH (logic 1) - Output LOW (logic 0)
- * Input LOW (logic 0) - Output HIGH (logic 1)

Truth Table

Input A	Output $Y = \bar{A}$
0	1
1	0

Logic Symbol



UNIVERSAL GATES

Universal gate is a gate which can implement any Boolean function without need to use any other gates. NAND and NOR gates are universal gates.

NAND Gate

NAND(gate) is a contraction of NOT-AND gates. It has two or more inputs and only one output.

Operation

* When all inputs are HIGH - Output is LOW

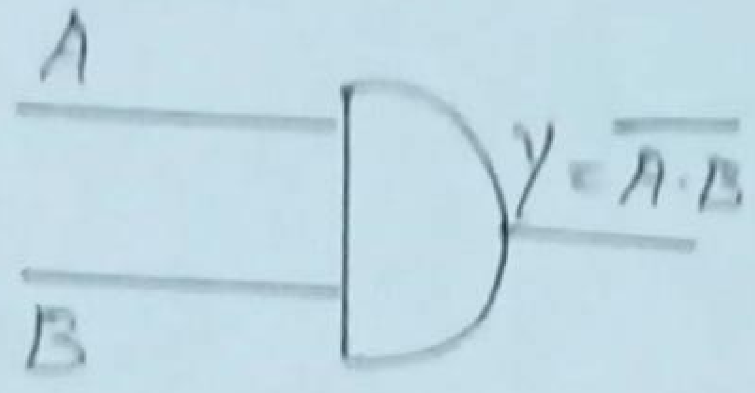
* Any one or all inputs are LOW - Output is HIGH

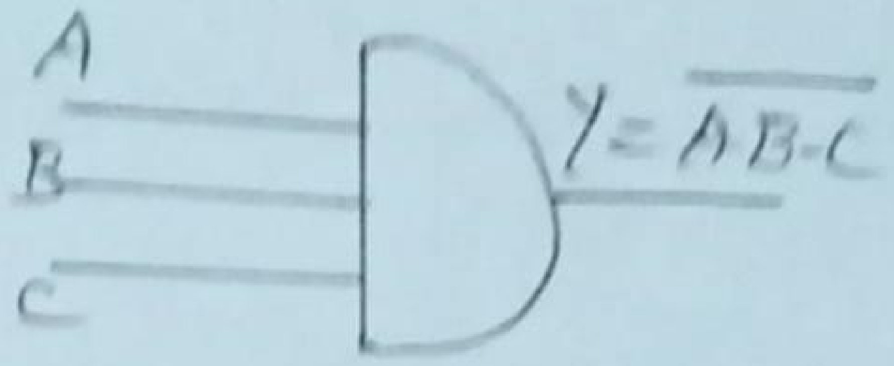
Truth Table

Inputs		Output
A	B	$Y=\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

Inputs			Output
A	B	C	$Y=\overline{A \cdot B \cdot C}$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Logic Symbol





NOR Gate

NOR is a contraction of NOT-OR gates. It has two or more inputs and only one output.

Operation

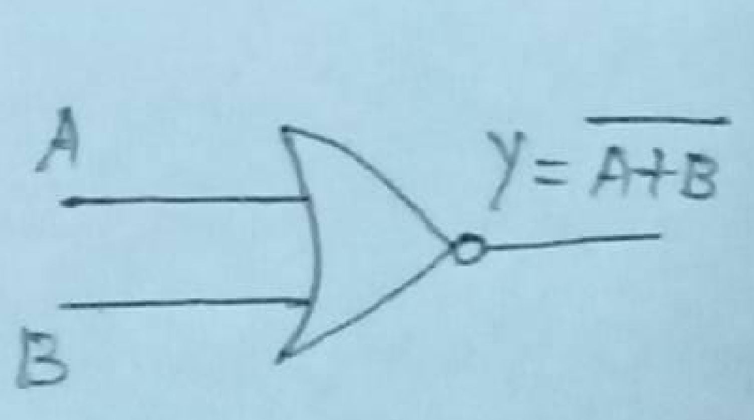
* When all inputs are LOW - Output is HIGH
* Any one or all inputs are HIGH } - Output is LOW

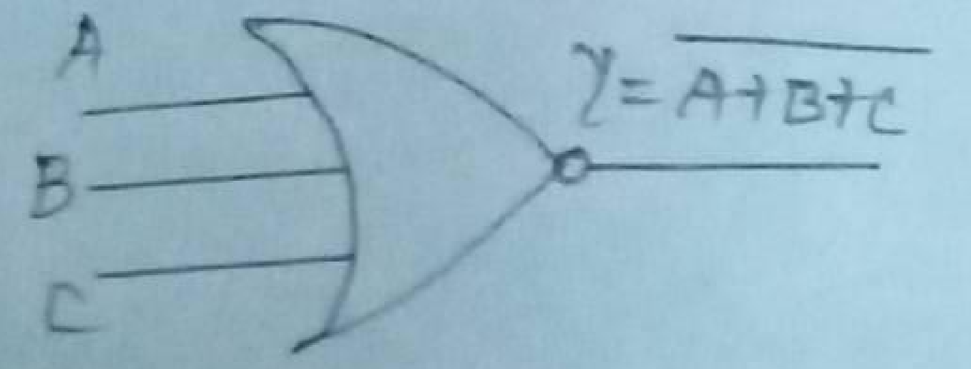
Truth Table

Inputs		Output
A	B	$Y=\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Inputs			Output
A	B	C	$Y=\overline{A+B+C}$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Logic Symbol





KARNAUGH MAP

Karnaugh map is a visual method used to simplify the algebraic expressions in Boolean functions without having to resort to complex theorems or equation manipulations.

The simplification of the switching function using Boolean laws and theorems becomes complex with the increase in the number of variables and terms. The Karnaugh map technique provides a systematic method for simplifying and manipulating switching expressions.

In this technique, the information contained in a truth table or available in the POS or SOP form is represented on the Karnaugh map (K-map). Karnaugh map is a modified form of a truth table.

In an n-variable K-map, there are 2^n cells. Each cell corresponds to one combination of n-variables. Therefore for each row of the truth table, ie., for each minterm and for each maxterm, there is one specific cell in the K-map.

Two Variable K-map

B \ A	0	1
0	00 0	10 2
1	01 1	11 3

$2^2 = 4$ cells

Three Variable K-map

C \ AB	00	01	11	10
0	000 0	010 2	110 6	100 4
1	001 1	011 3	111 7	101 5

$2^3 = 8$ cells

Four Variable K-map

CD \ AB	00	01	11	10
00	0000 0	0100 4	1100 12	1000 8
01	0001 1	0101 5	1101 13	1001 9
11	0011 3	0111 7	1111 15	1011 11
10	0010 2	0110 6	1110 14	1010 10

$2^4 = 16$ cells

≡

AB \ CD	00	01	11	10
00	0000 0	0001 1	0011 3	0010 2
01	0100 4	0101 5	0111 7	0110 6
11	1100 12	1101 13	1111 15	1110 14
10	1000 8	1001 9	1011 11	1010 10

$2^4 = 16$ cells

Five Variable K-map

DE \ ABC	000	001	011	010	110	111	101	100
00	00000 0	00100 4	01100 12	01000 8	11000 24	11100 28	10100 20	10000 16
01	00001 1	00101 5	01101 13	01001 9	11001 25	11101 29	10101 21	10001 17
11	00011 3	00111 7	01111 15	01011 11	11011 27	11111 31	10111 23	10011 19
10	00010 2	00110 6	01110 14	01010 10	11010 26	11110 30	10110 22	10010 18

$2^5 = 32$ cells

QUINE McCLUSKEY METHOD / TABULAR METHOD

[TABULATION METHOD]

Quine McCluskey method also known as tabulation method which is used to minimize the boolean function. It simplifies boolean expression into the simplified form using prime implicants. This method is convenient to simplify boolean expressions with more than four input variables.

Quine McCluskey method is a tabular method based on the concept of prime implicants. Prime implicant is a product or sum term, which cannot be further reduced by combining with any other product or sum terms of the given boolean function.

Karnaugh map and Quine McCluskey methods are well known methods to simplify Boolean expression. K-map method becomes complex beyond five variable Boolean expression. Quine McCluskey method is computer based technique for minimization of Boolean function and it is faster than K-map method.

PROBLEM FORMULATION AND DESIGN OF COMBINATIONAL CIRCUITS

Combinational logic circuits are circuits in which the output at any time depends upon the combination of the input signals present at that instant only, and does not depend upon any past conditions.

Examples

1. Adders
2. Subtractors
3. Comparators
4. Decoders
5. Encoders
6. Multiplexers
7. Demultiplexers
8. Parity generators
9. Parity checkers
10. Code converters

ADDER

Adder is a combinational logic circuit that performs addition of numbers.

Types

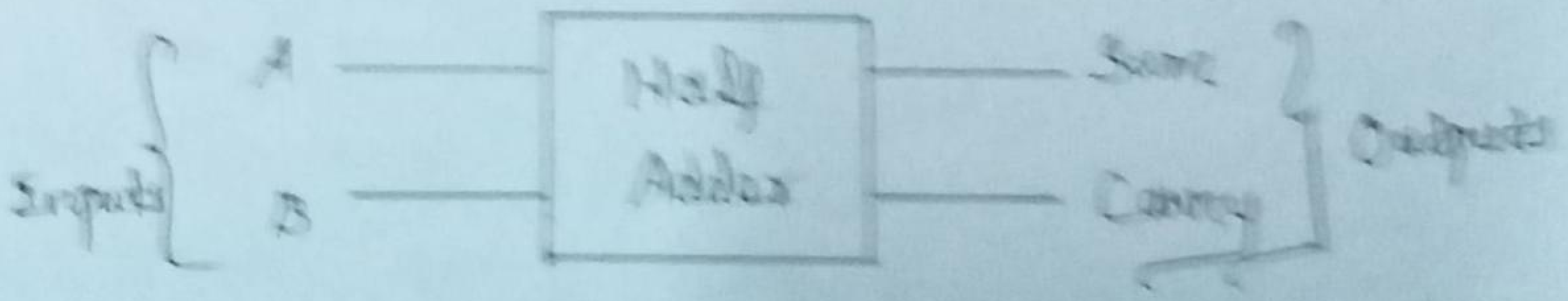
1. Half Adder
2. Full Adder

Half Adder

Half adder is the simplest combinational logic circuit which performs the arithmetic addition of two binary digits.

* It has two inputs and two outputs
* Inputs – A, B (two single bit numbers)
* Outputs – Sum, Carry

Block Diagram



Truth Table

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Operation

* From the truth table the sum output is 1 when any one of the input is 1

* The sum output is '0' when both the inputs are '0'

* The sum output is '0' when both the inputs are '1'

From the truth table,

* The carry output is '0' when both the inputs are '0'

* The carry output is '0' when any one of the input is '1'

* The carry output is '1' when both the input are '1'

Case 1: A = 0, B = 0

Sum = 0+0
= 0

Carry = 0

Case 2: $A=0, B=0$

Sum $= 0+1 = 1$ Carry $= 0$

Case 3: $A=1, B=0$

Sum $= 1+0 = 1$ Carry $= 0$

Case 4: $A=1, B=1$

Sum $= 1+1 = 0$ Carry $= 1$

$$\begin{array}{r} 1\,+ \\ 1 \\ \hline 1\;0 \\ \uparrow\;\uparrow \end{array}$$

Carry sum

K-map

Sum

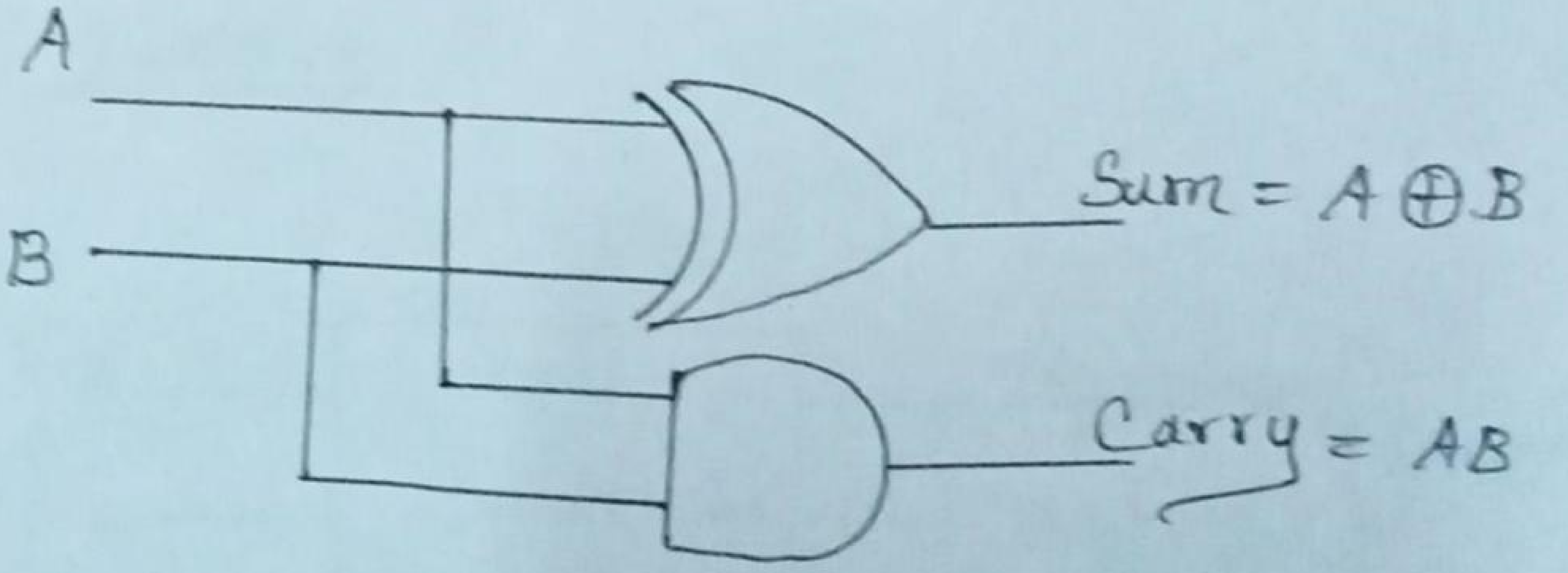
B \ A	$\bar{A}$ 0	A 1
$\bar{B}$ 0	00 0	~~0~~10 ①
B 1	01 ①	11 0

Carry

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

$$\text{Sum} = A\bar{B} + \bar{A}B$$
$$= A \oplus B$$

$$\text{Carry} = AB$$



SUBTRACTOR

Subtractor is a combinational logic circuit that performs subtraction of numbers.

Types

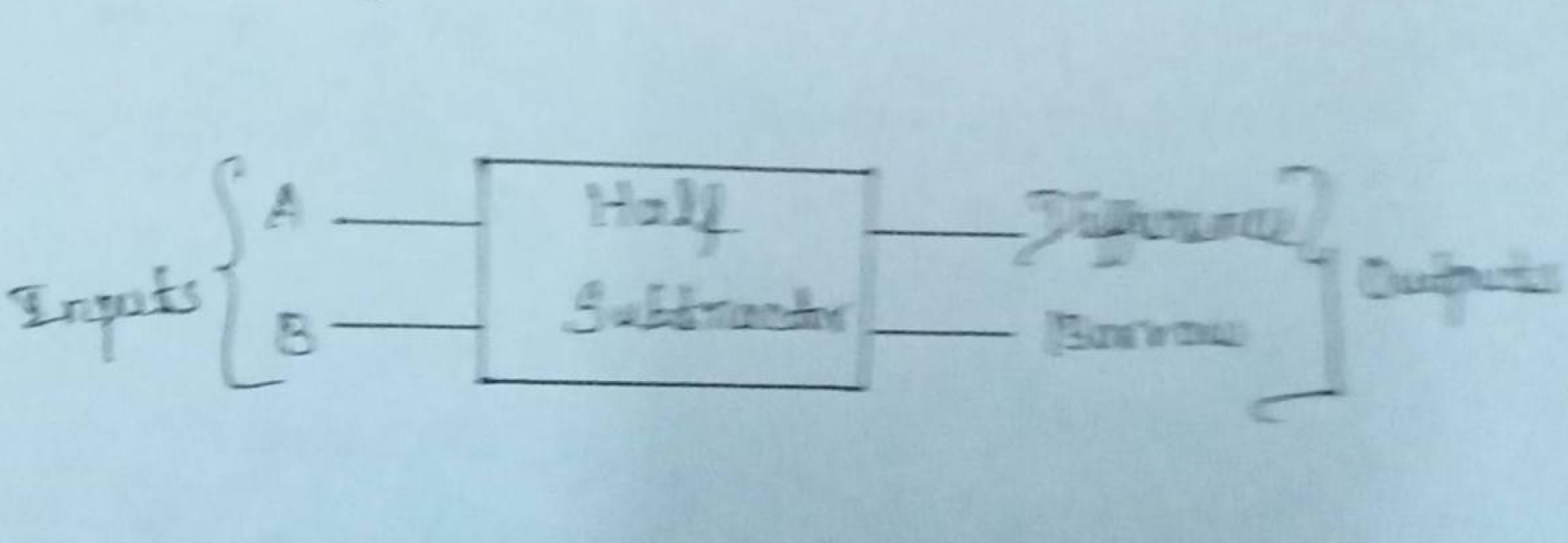
1. Half Subtractor
2. Full Subtractor

Half Subtractor

Half subtractor is the simplest combinational logic circuit which performs the arithmetic addi of two binary digits.

- * It has two inputs and two outputs
- * Inputs - A, B (two single bit numbers)
- * Outputs - Difference, Borrow

Block Diagram



Truth Table

Inputs		Outputs	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Operation

Case 1: A = 0 , B = 0

Difference = 0 − 0 Borrow = 0
= 0

Case 2: A = 0 / B = 1

Difference = 0 − 1 Borrow = 1
= 1

Borrow
↳1 0
0 −
1
―――
1
―――
↑
Difference

Case 3: A = 1, B = 0

Difference = 1 − 0 Borrow = 0
= 1

Case 4: A = 1, B = 1

Difference = 1 − 1 Borrow = 0
= 0

K-map

Difference

B \ A	0	1
0	00 0	10 1
1	01 1	11 0

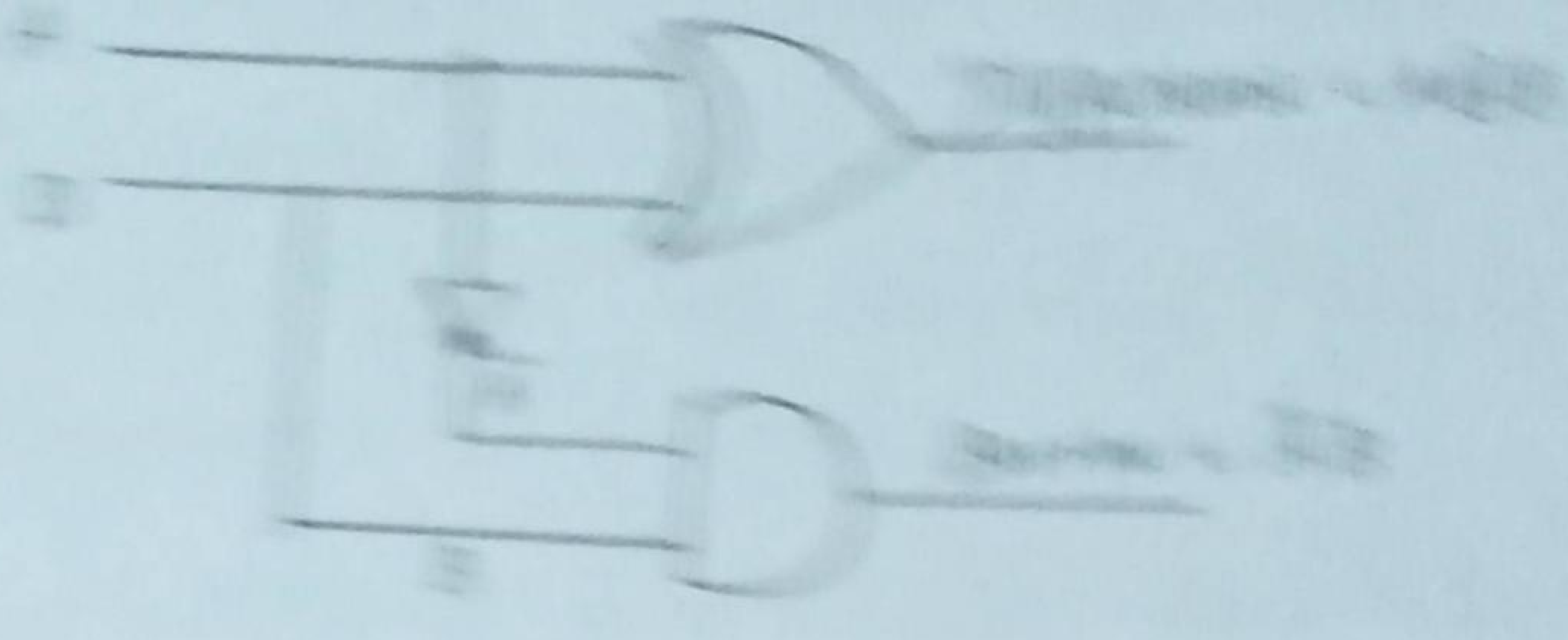
Difference = $A\bar{B} + \bar{A}B = A \oplus B$

Borrow

B \ A	0	1
0	00 0	10 0
1	01 1	11 0

Borrow = $\bar{A}B$

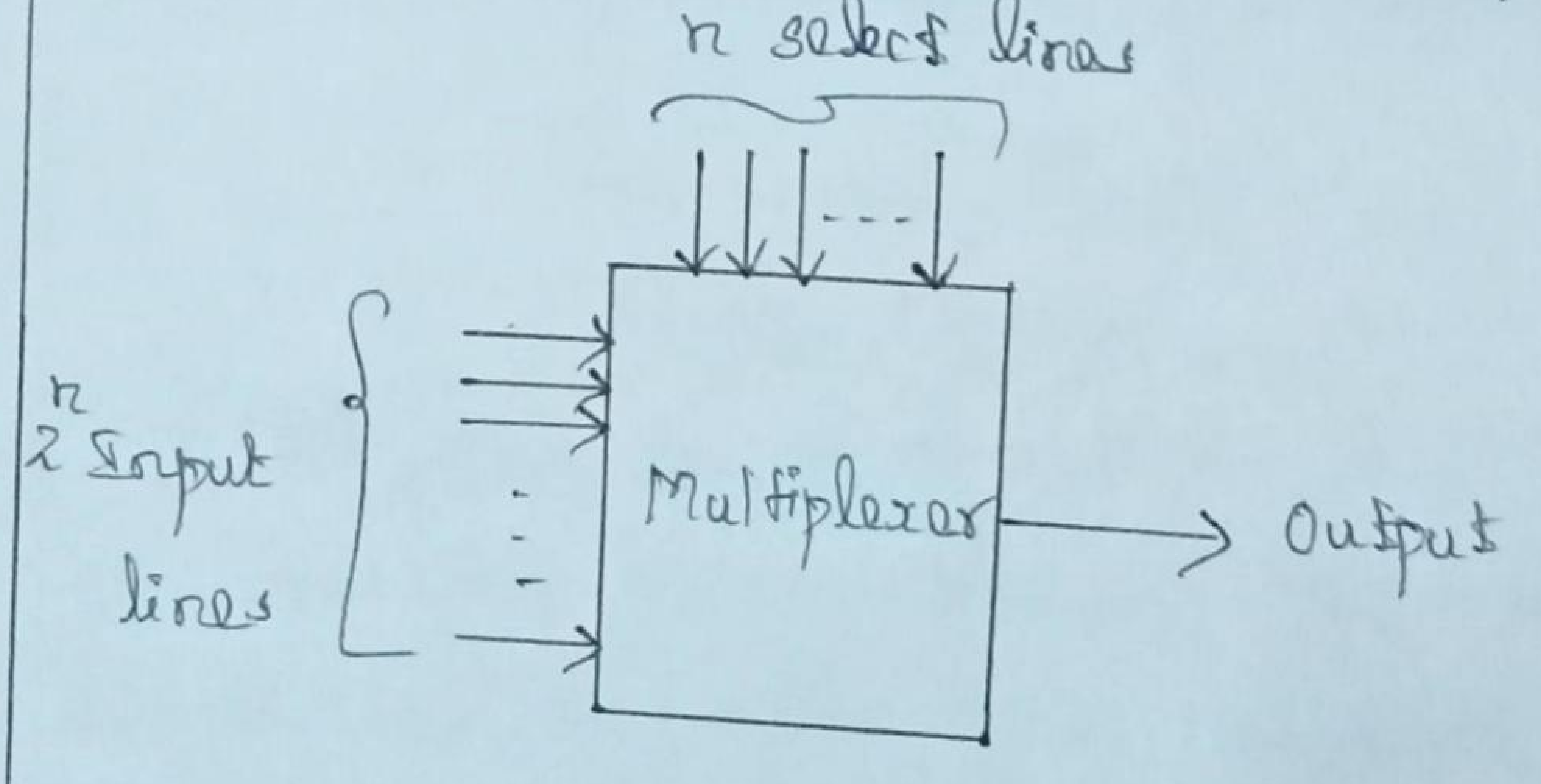
Logic Diagram



MULTIPLEXER

Multiplexer is a combinational logic circuit that selects one digital information from several sources and transmits the selected information on a single output line.

Multiplexer is also called data selector since it selects one of many inputs and steers transmits the information to the output.

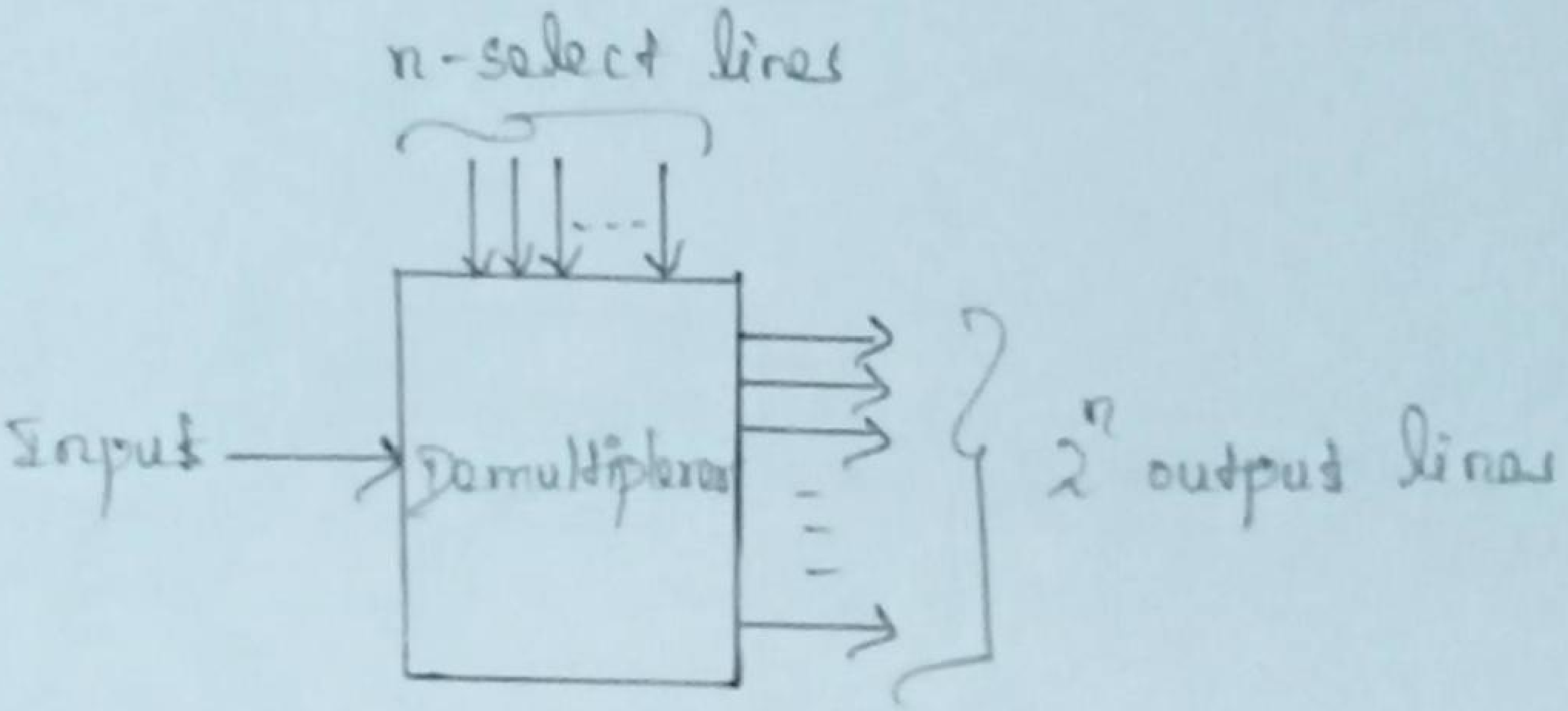


Block Diagram

The multiplexer has several data-input lines and a single output line. The selection of a particular input line is controlled by a set of selection lines. The selection lines decide the number of input lines of a particular multiplexer.

DEMULTIPLEXER

Demultiplexer is a combinational logic circuit that receives information on a single input and transmits the same information over one of several [2^n] output lines.



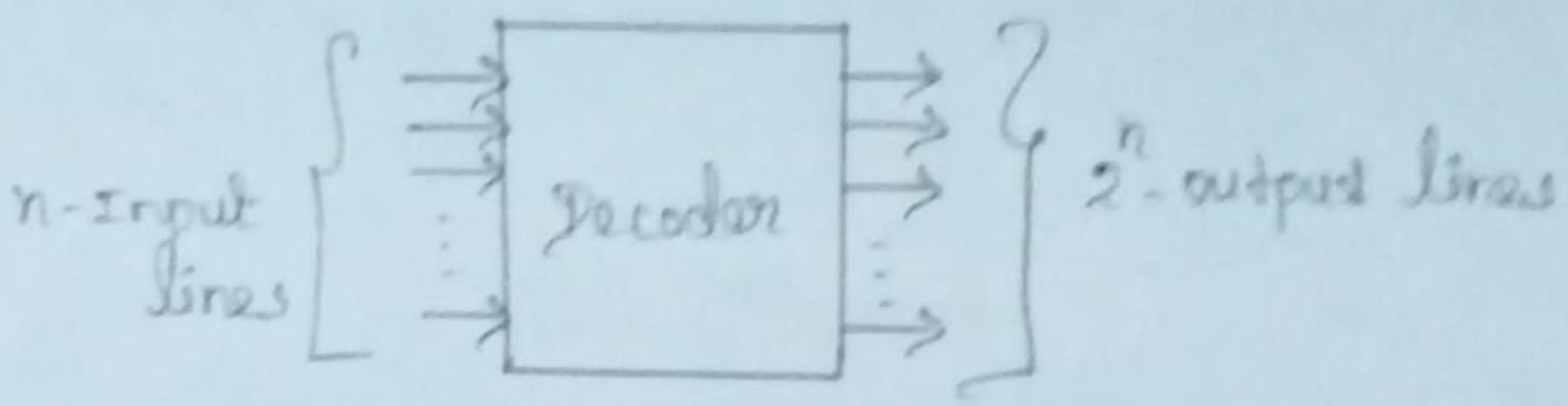
Block Diagram

The operation of demultiplex is opposite to the operation of multiplexer. The demultiplexer circuit has one input signal, n-select signals and 2^n output signals. The selection inputs determine to which output the data input will be connected.

As the serial data is changed to parallel data, ie., the input caused to appear on one of the 2n output lines, the demultiplexer is also called a distributor or a serial to parallel converter.

DECODER

Decoder is a combinational logic circuit that converts n-bit binary input (Information) code into 2^n output lines. Each output line will be activated for only one of the possible combinations of inputs.

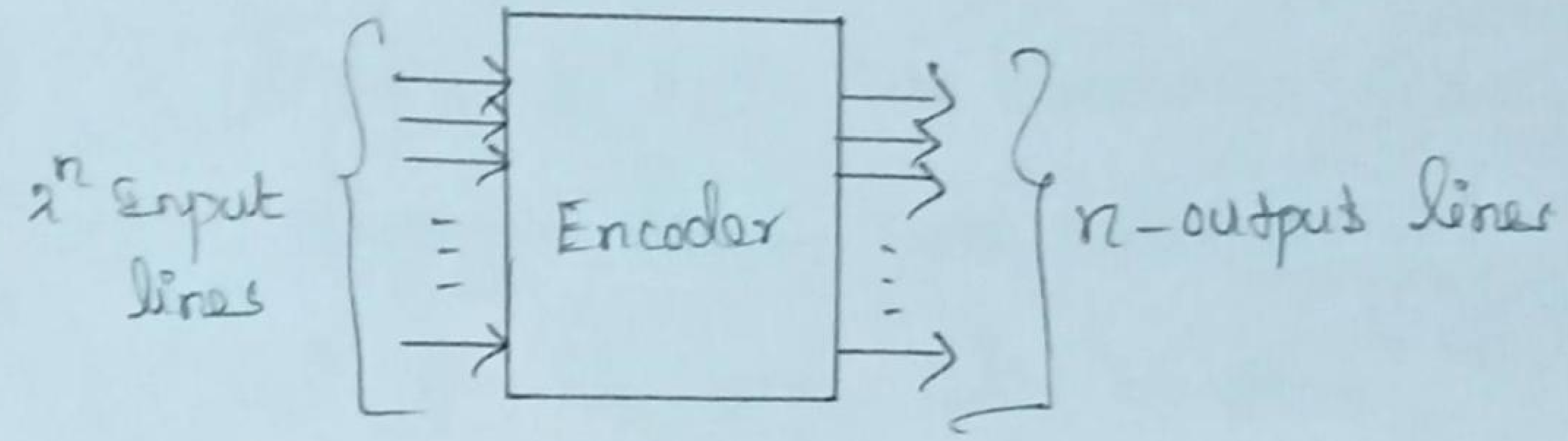


Block Diagram

Decoder is similar to demultiplexer but without any data input. In a decoder, the number of output is greater than the number of inputs. If the number of inputs and outputs are equal in a digital system then it can be called converters.

ENCODER

Encoder is a combinational logic circuit that converts an active input signal into a coded output signal. It performs the inverse operation of a decoder.



Block Diagram

It has 2^n input lines, only one of which is active at any time and n-output lines. It encodes one of the active inputs to a coded binary output with 'n' bits.

In an encoder, the number of outputs is less than the number of inputs.

CODE CONVERTER

Code converter is a combinational logic circuit that changes the data in one form of binary code to another form of binary code.

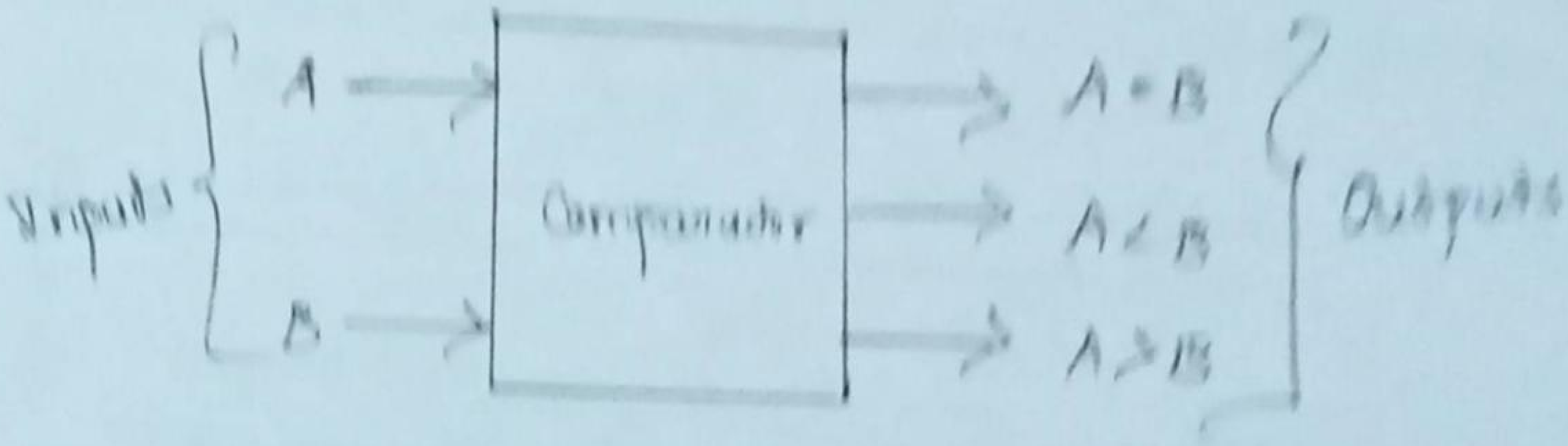
Examples

1. Binary to BCD Code Converter
2. BCD to Binary Code Converter
3. Binary to Gray Code Converter
4. Gray to Binary Code Converter
5. BCD to Excess-3 Code Converter
6. Excess-3 to BCD code Converter

COMPARATOR [Magnitude Comparator]

Magnitude comparator is a combinational logic circuit that compares the magnitude of two numbers and generates their relative magnitudes as output.

Inputs ⇒ A, B , Outputs ⇒ A=B, A<B, A>B



Block Diagram

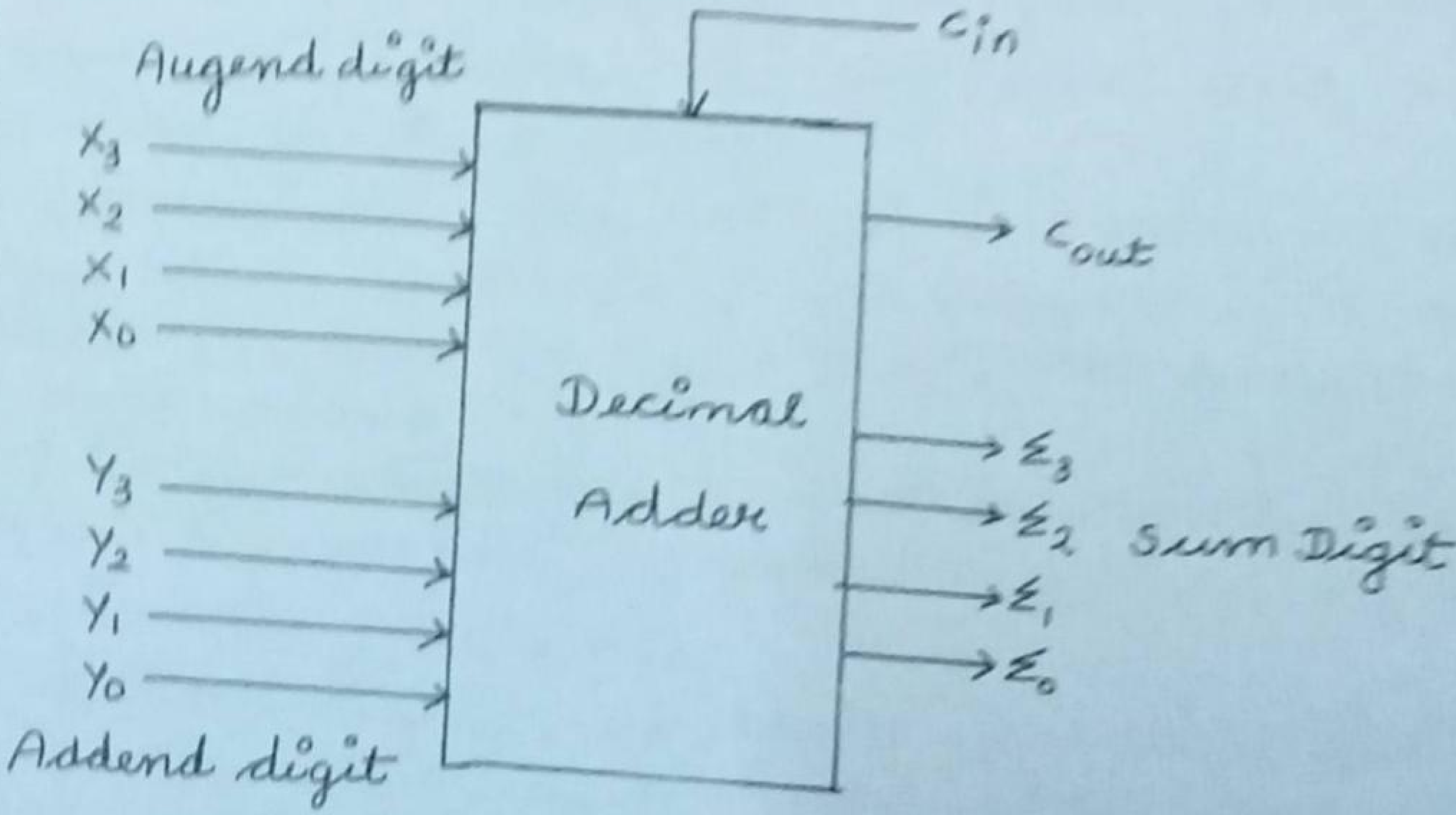
To implement the magnitude comparator, the Ex-NOR gates and AND gates are used. The EX-NOR gate can be used to find whether the two binary digits are equal or not, and the AND gates are used to find whether a binary digit is less than or greater than another bit.

- If A=B, then the output of EX-NOR gate will be '1'.
- If A≠B, then the output of EX-NOR gate will be '0'.

BCD Adder

A BCD adder is a circuit that adds two BCD digits in parallel and produces a sum digit which is also in BCD. A BCD adder must include the correction logic in its internal construction. A block diagram for BCD adder is shown as figure. This adder has two 4 bit BCD inputs and one carry input. It has a 4 bit sum output and one carry output. Here the sum is in BCD form.

Block diagram of a BCD adder



* Add two 4 bit BCD numbers using straigh binary addition
* If the four-bit sum is equal to or less than 9, the sum is in Proper BCD form and no correction is needed
* If the four-bit sum is greater than 9 or if a carry is generated from the sum, the sum is not in BCD form.

K-map Simplification

	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	0	1	1
10	0	0	1	1

$C_n = S_3S_2 + S_3S_1 + C_4$

* From table it is clear that correction is required when the sum output is greater than 9 i.e., when $C_4 = 1$ (or) $S_3 = 1$, add 0110 to get the B.C.D result.

* When $C_4 = 1$, the result is 16 and above; when $S_3 = 1$ and $S_2 = 1$, the result is 12 and above; when $S_3 = 1$ and $S_1 = 1$, the result is 10 and above.

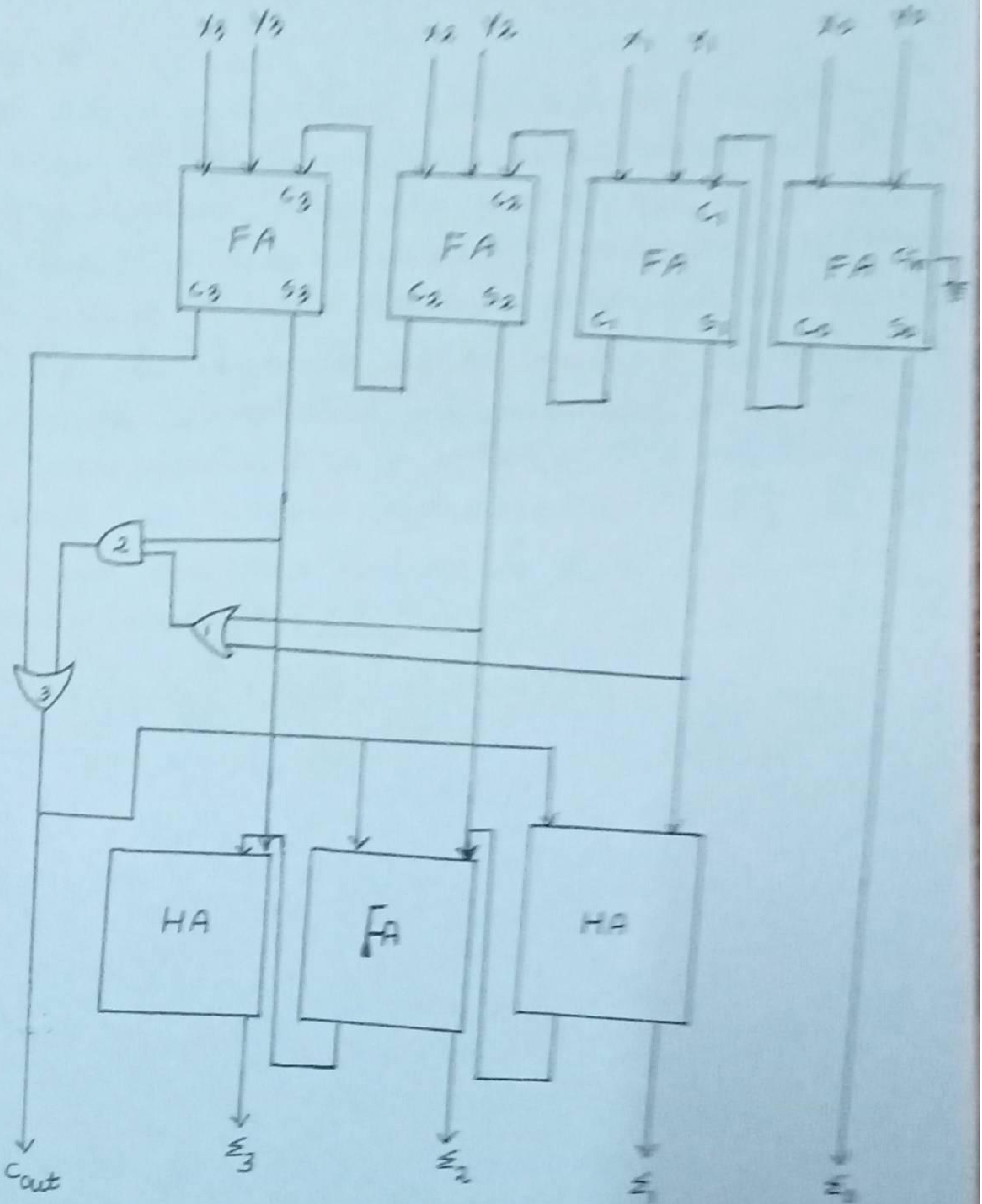
* The condition for correction can be written as an expression as follows

$C_n = C_4 + S_3(S_2 + S_1)$

* Alternatively, the above condition for correction can be obtained by K-map method.

* As discussed above, B.C.D adder must be capable of adding two 4-bit B.C.D numbers and the result has to be corrected to B.C.D, if the condition is satisfied by adding 0110.

BCD adder using full adders



Then the digit 6 (0110) Should be added to the sum to produce the BCD results. The Carry may be produced due to this addition and it is added to next decimal position.

Truth table

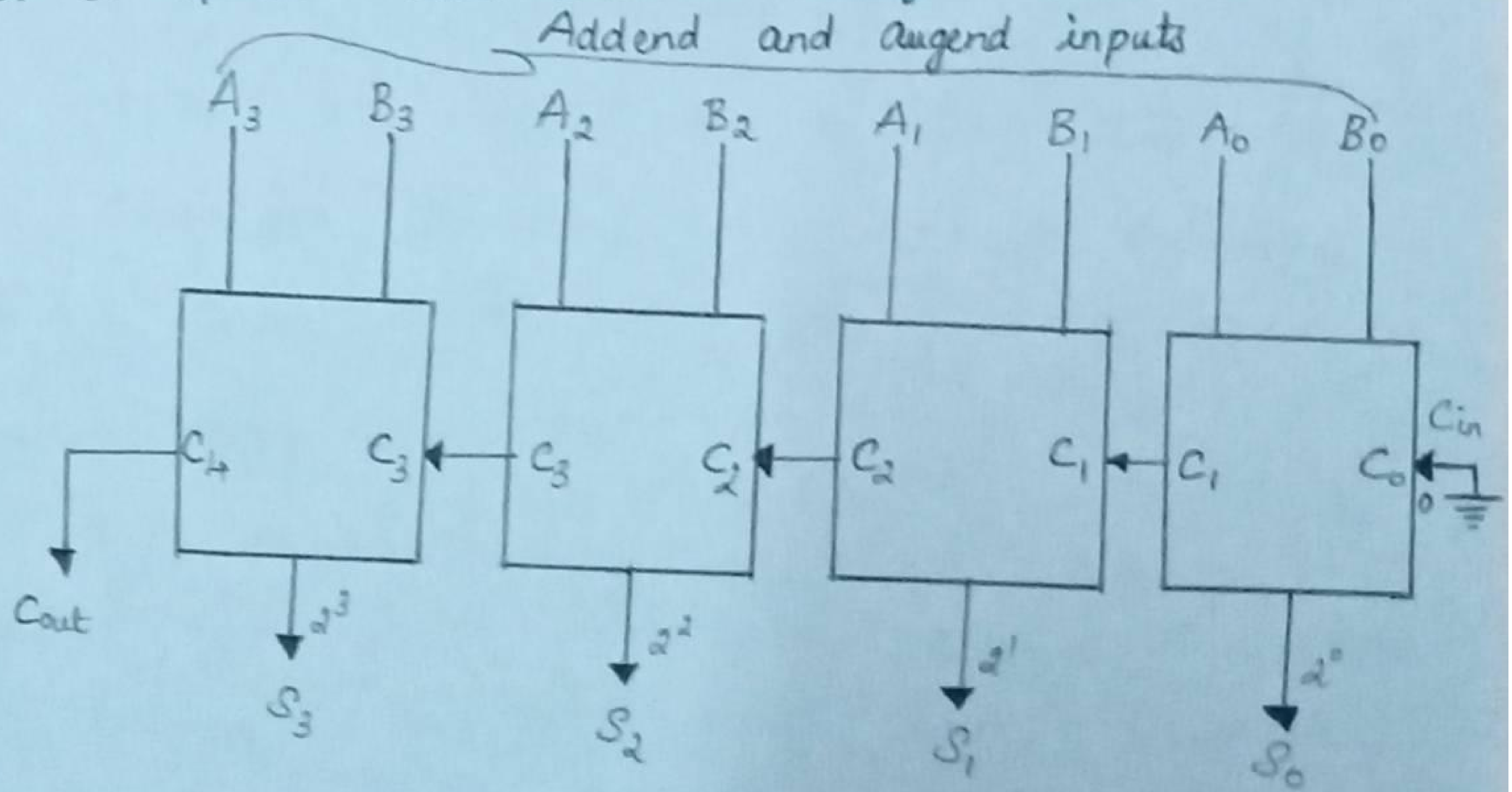
Decimal digit	uncorrected BCD					corrected BCD				
	C_3	S_3	S_2	S_1	S_0	C_{out}	S_3	S_2	S_1	S_0
0		0	0	0	0		0	0	0	0
1		0	0	0	1		0	0	0	1
2		0	0	1	0		0	0	1	0
3		0	0	1	1		0	0	1	1
4		0	1	0	0		0	1	0	0
5		0	1	0	1		0	1	0	1
6		0	1	1	0		0	1	1	0
7		0	1	1	1		0	1	1	1
8		1	0	0	0		1	0	0	0
9		1	0	0	1		1	0	0	1
10		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	1	0	0
15		1	1	1	1	1	0	1	0	1
16	1	0	0	0	0	1	0	1	1	0
17	1	0	0	0	1	1	0	1	1	1
18	1	0	0	1	0	1	1	0	0	0
19	1	0	0	1	1	1	1	0	0	1

PARALLEL BINARY ADDER

The Parallel binary adder is a Combinational logic circuit consists of various full adders in parallel structure and performs addition operation of two binary numbers.

* The addition of multibit numbers can be accomplished using several full adders.

* The 4-bit adder using full adder circuits is capable of adding two 4-bit numbers resulting in a 4-bit sum and a carry output.



4- bit parallel binary adder

* Inputs to each full adder will be A_i, B_i, C_i

* Outputs will be S_i and C_{i+1} where 'i' varies from 0 to 3.

Addition Operation

* Let the 4-bit numbers to be added be represented by $A_3A_2A_1A_0 = 1111$ and $B_3B_2B_1B_0 = 0011$

Significant place		:	4	3	2	1
Input carry		:	1	1	1	0
Augend input A		:	1	1	1	1
Addend input B		:	0	0	1	1
	1		0	0	1	0 ← Sum
	↑		S_3	S_2	S_1	S_0

Output carry

* The carry output of the lower order stage is connected to the carry input of the next higher order stage. Hence this type of adder is called ripple-carry adder.

* In the least significant stage A_0, B_0 and C_0 are added resulting in sum S_0 and carry C_1. This carry C_1 becomes the carry input to the second stage.

* Similarly in second stage, A_1, B_1 and C_1 are added resulting in S_1 and C_2

* In third stage A_2, B_2, and C_2 are added resulting in S_2 and C_3

* In the fourth stage, A_3, B_3 and C_3 are added resulting in S_3 and C_4 which is the output carry.

* Thus, the circuit results in a sum (S_3 S_2 S_1 S_0) and a carry output (Cout)

Limitations:

* Though the parallel binary adder is said to generate its output immediately after the inputs are applied, its speed of operation is limited by the carry propagation delay through all stages.

* In each full-adder, the carry input has to be generated from the previous full adder which has a inherent propagation delay (t_P).

* The propagation delay (t_P) of a full-adder is th time difference between the instant at which th inputs (A_i, B_i and C_i) are applied and the instant at which the outputs (S_i and C_{i+1}) are generate

* Therefore, in a 4-bit binary adder, the output in LSB stage is generated only after tp seconds.

* Similarly, the output in the second stage will b generated only after 2tp seconds from the tim the inputs are applied.

* The third stage will generate outputs only afte 3 tp seconds and the fourth

* There is a 4-bit binary adder where each full adder has a propagation delay of 50 ns, the output in the fourth stage will be generated only after $4t_p = 4 \times 50ns = 200ns$. The magnitude of such delay is prohibitive for high-speed computers.

Methods to reduce the delay:

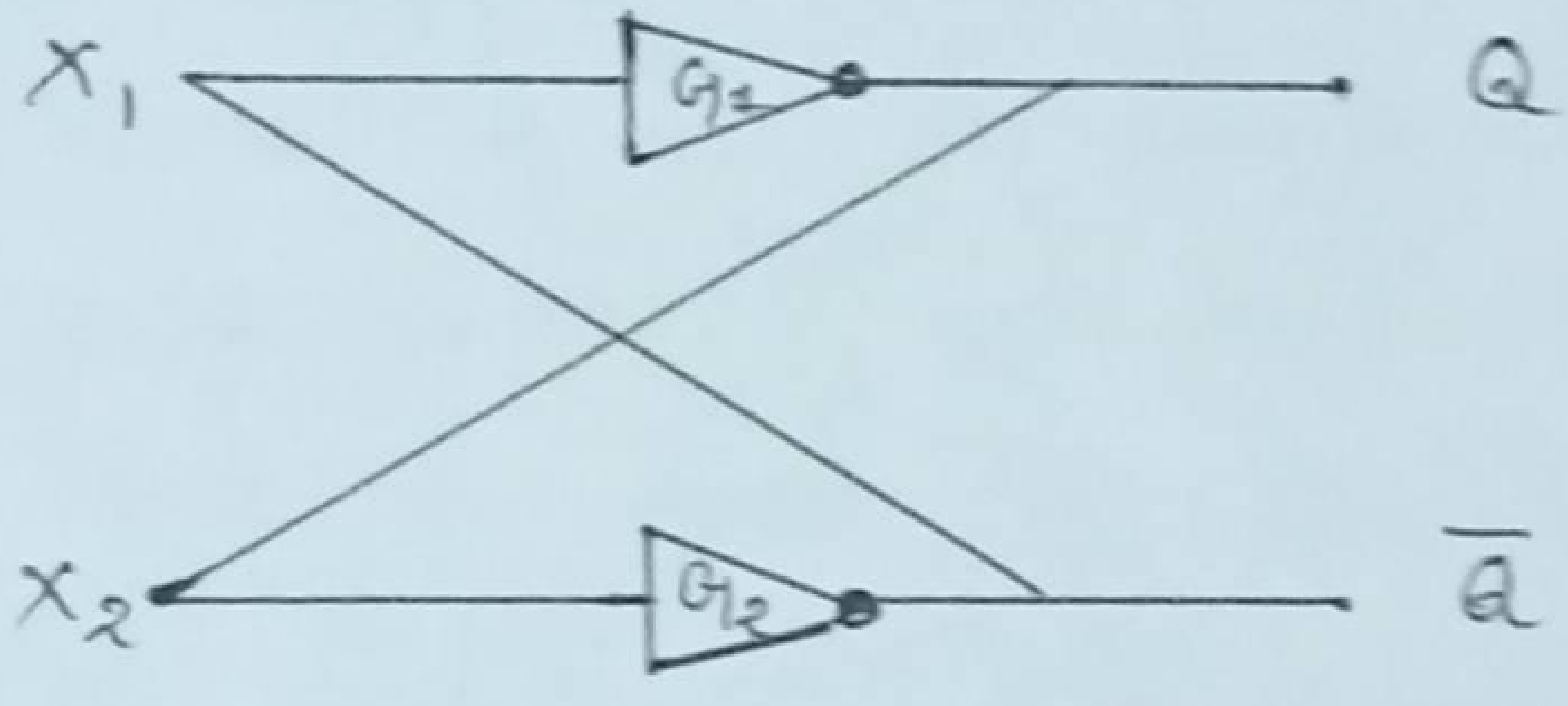
* One of the methods of speeding up this process is look-ahead carry addition which eliminates the ripple-carry delay.

* This method is based on the carry generate and the carry propagate functions of the full adder.

LATCHES

In a sequential digital logic circuit, data are stored in memory devices called flip-flops or latches.

Latches are the simplest kind of sequential circuit have only two states. It is a memory cell / storage cell, which is capable of storing one bit of information, ie, logic 1 or logic 0. In this type of sequential circuit one bit of information can be locked or latched. The basic latch consists of two inverters.



Basic Latch Diagram

The output Q of inverter G_1 is connected to the input X_2 of G_2 and the output $\overline{Q}$ of G_2 is connected to the input X_1 of G_1.

Set Reset (S R) Latch

The S R latch has two inputs, SET (S) and RESET (R) and two outputs Q and $\bar{Q}$. The two outputs are complement to each other.

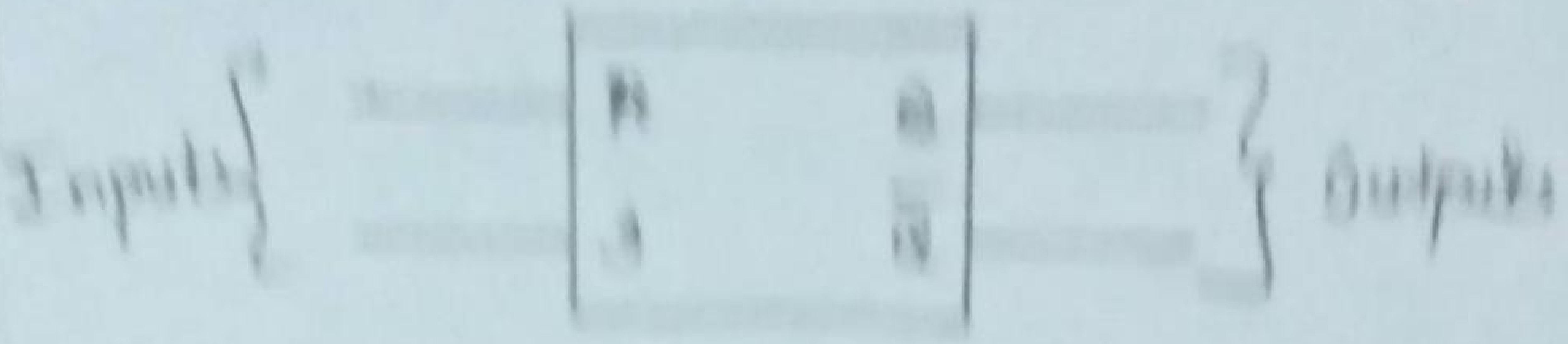


Fig: Block Diagram

The S R latch can be implemented using NOR gates or NAND gates.

S R latch using NOR gates

NOR based S R latch is implemented using two NOR gates connected back to back. The cross coupled connection from the output of one gate to the input of the other gate constitute a feedback path. For this reason, latches are classified as asynchronous sequential circuits.

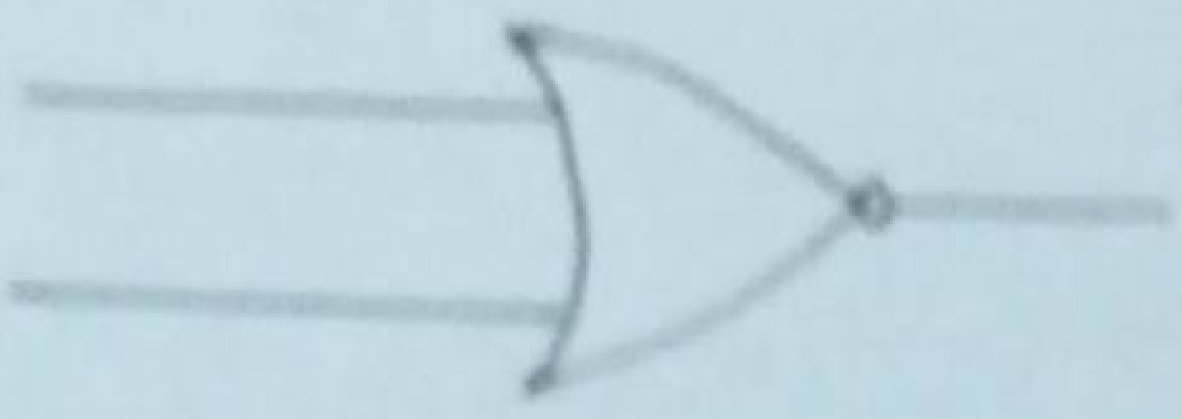


Fig: NOR Gate

Inputs		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Truth Table

- The output of a NOR gate is '0' if any one of the input is 1 and all the inputs are 1.
- The output of a NOR gate is '1' if both the inputs are 0

Logic Diagram

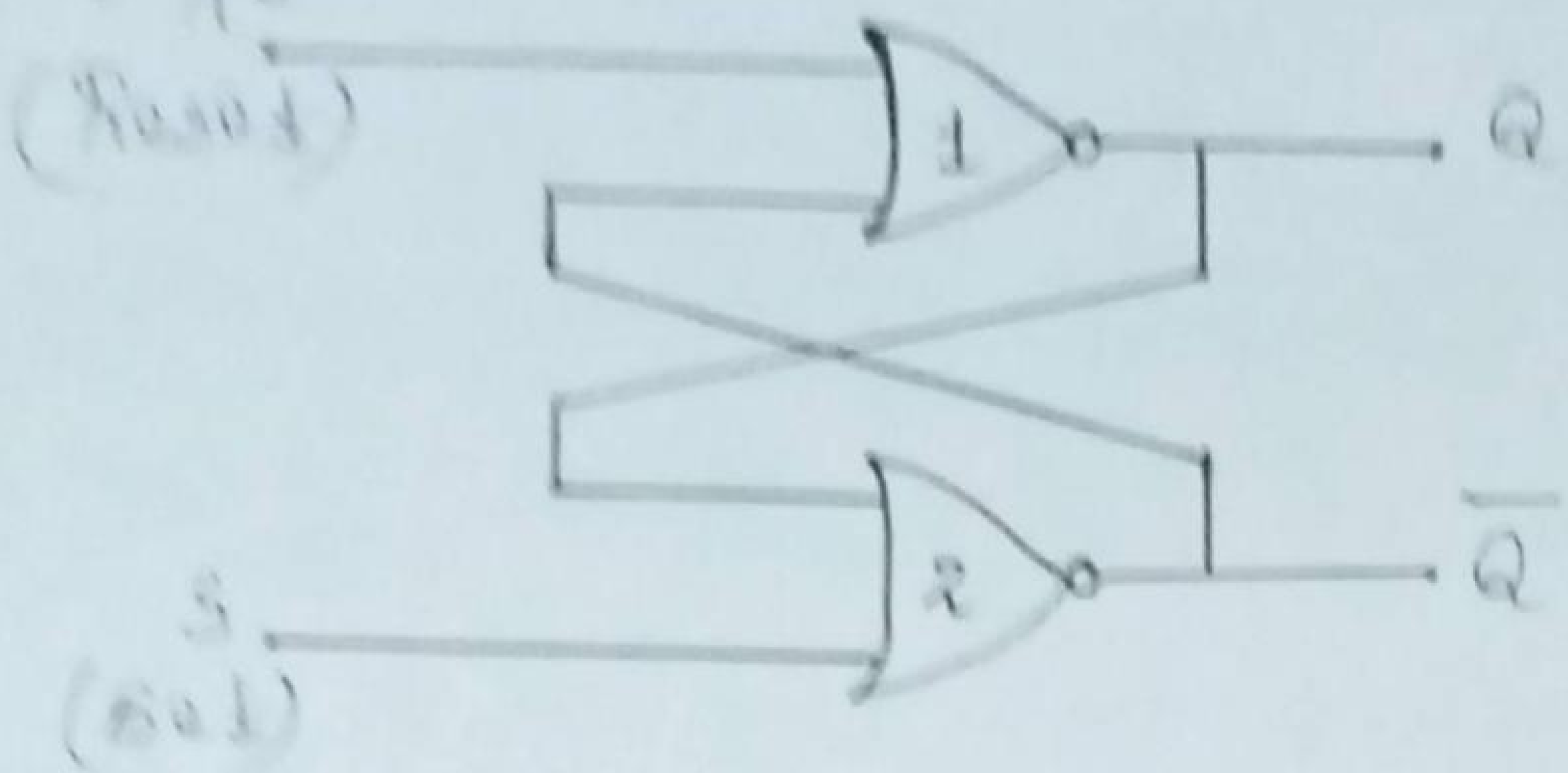


Fig: NOR Based S-R Latch

Truth Table

Inputs		Outputs		Action
S	R	Q_{n+1}	$\overline{Q}_{n+1}$	
0	0	Q_n	$\overline{Q_n}$	No Change
0	1	0	1	Reset
1	0	1	0	Set
1	1	?	?	Forbidden

Operation

Case 1 : S = 0 , R = 0

- Latch remains in its present state
- ie., the next state of the latch (Q_{n+1}) is just the present state

* The next state of the latch will be

$Q_{n+1} = 0$ if $Q_n = 0$

and $Q_{n+1} = 1$ if $Q_n = 1$

1. Assume $Q_n = 0$ and $\overline{Q_n} = 1$

* Inputs of NOR gate-1 are 1 and 0, and therefore its output $Q_{n+1} = 0$
* This $Q_{n+1} = 0$ is fed back to NOR gate-2 input
* Inputs of NOR gate-2 are 0 and 0, and therefore its output $\overline{Q_{n+1}} = 1$

Here output is,

$Q_{n+1} = 0$ & $\overline{Q_{n+1}} = 1$ [originally assumed]

2. Assume $Q_n = 1$ and $\overline{Q_n} = 0$

* Inputs of NOR gate-1 are 0 and 0, and therefore its output $Q_{n+1} = 1$
* This $Q_{n+1} = 1$ is fed back to NOR gate-2 input
* Inputs of NOR gate-2 are 1 and 0, and therefore its output $\overline{Q_{n+1}} = 0$

Here output is,

$Q_{n+1} = 1$ & $\overline{Q_{n+1}} = 0$ [originally assumed]

Case 2 : S = 0 , R = 1

* For NOR gate-1, one of the input is 1

* Another input is either '0' or '1', the output $Q_{n+1} = 0$

* Therefore, the input condition S=0 and R=1 will always resets the latch to '0'

* Both the inputs of NOR gate-2 are 0 and 0, therefore the output $\overline{Q_{n+1}} = 1$

Case 3: S=1 , R=0

* This input condition forces the output of NOR gate-2 to LOW, ie., $\overline{Q_{n+1}} = 0$

* This $\overline{Q_{n+1}} = 0$ is one of the input of NOR gate-1. Two inputs of NOR gate-1 are 0 and, therefore the output $Q_{n+1} = 1$

* Hence the condition S=1, R=0 will always sets the (gate) latch to '1.'

Case 4: S=1 , R=1

* Two inputs of NOR gate-1 are 1,1 and therefore the output $Q_{n+1} = 0$

|||ly

* Two inputs of NOR gate-2 are 1,1 and therefore the output $\overline{Q_{n+1}} = 0$

Here,
$Q_{n+1} = \overline{Q_{n+1}}$ - this is impossible. [Q_n & $\overline{Q_n}$ complement to each other)

Scanned by TapScanner

FLIP FLOPS

Flip flop is a device which stores a single bit binary information. Flip flops are synchronous sequential circuits change their states only when clock pulses are present.

The operation of the basic latch can be modified, by providing an additional control input that determines, when the state of the circuit is to be changed. The latch with the additional control input is called the flip-flop. The additional control input is either clock or enable input.

Storage elements that operates with signal levels rather than signal transitions are referred to as latches, those latches are controlled by a clock transition are called flip-flops. So, Latches are referred to as level sensitive devices whereas flip flops are referred to as edge sensitive devices.

FLIP FLOP

Flip flop is a device which stores a single bit binary data. The two states of a flip flop are logic '1' and logic '0'.

The operation of the basic latch can be modified, by providing an additional control input that determines, when the state of the circuit is to be changed. The latch with the additional control input is called the flip flop. The additional control input is either the clock or enable input.

Types

Flip flops are of different types depending on how their input and clock pulses cause transition between two states.

1. S-R Flip flop
2. J-K Flip flop
3. D Flip flop
4. T Flip flop

S-R Flip Flop.

SR Flip Flop - Set Reset Flip Flop

The S-R flip flop consists of two additional AND gates at the S and R inputs of S-R latch.

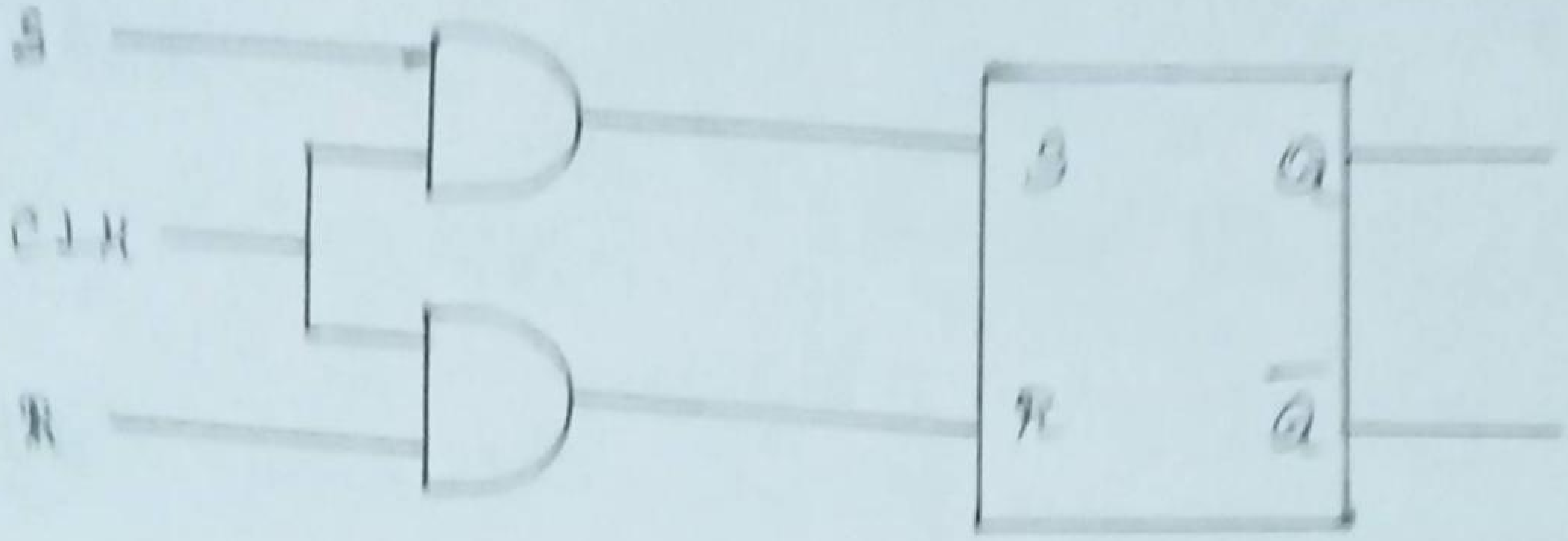


Fig: Block Diagram of S-R Flip Flop

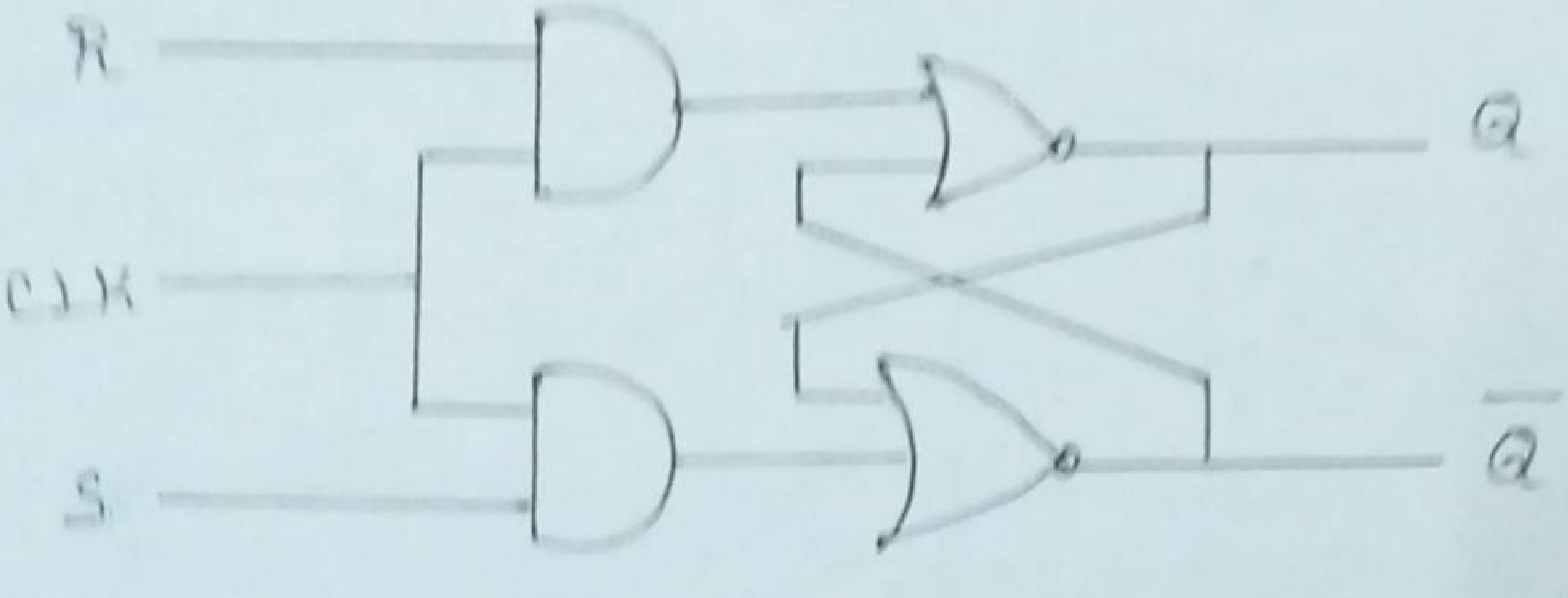


Fig: NOR Based S-R Flip Flop

* when the clock input is (zero) low, the output of both the AND gates are (zero) low

* Therefore, the changes in S and R inputs will not affect the output (Q) of the flip flop

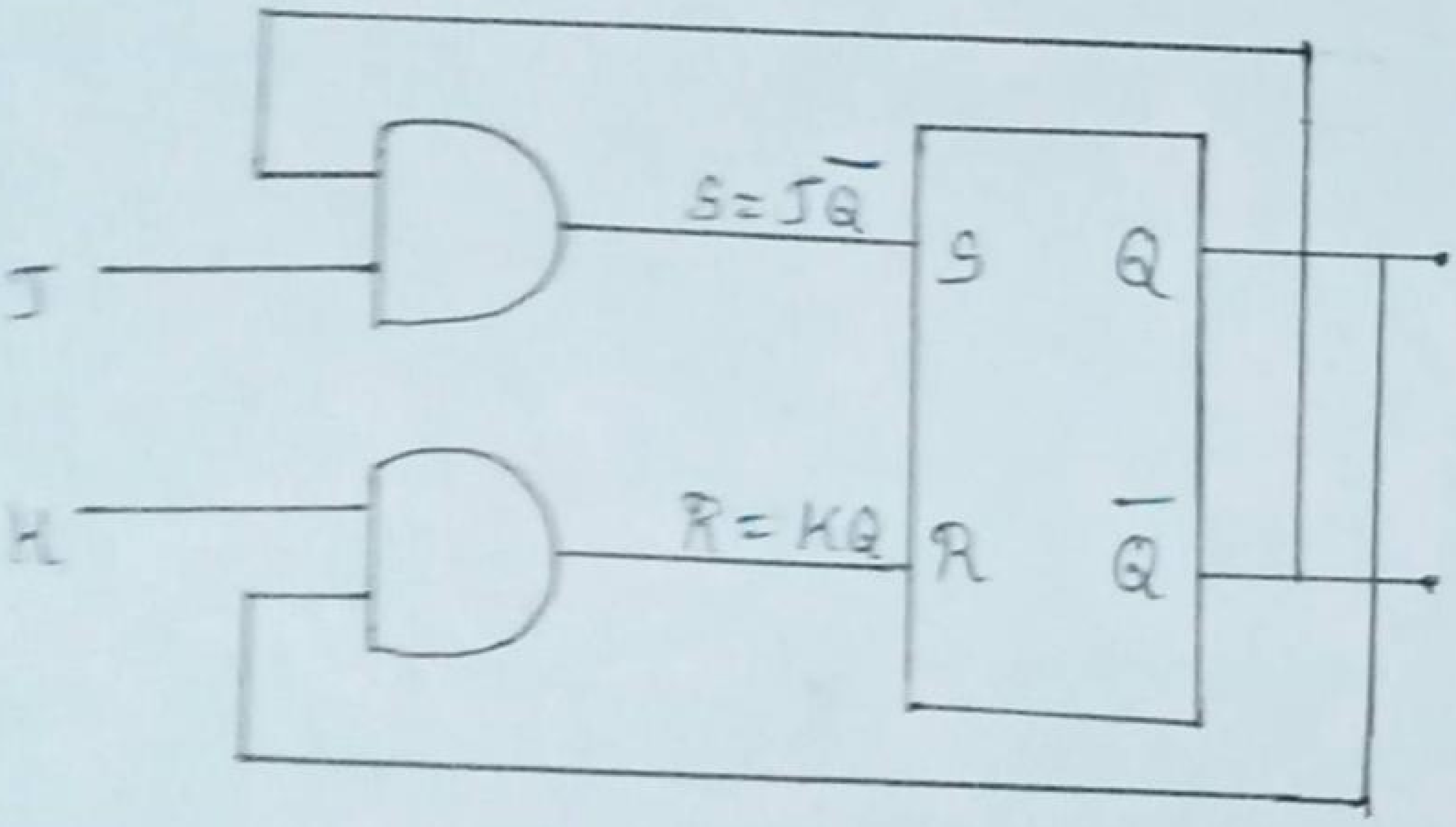
* when the clock input becomes HIGH, the values at S and R inputs will be passed to the output of AND gates and the output(s) of the flip flop will change according to the changes in S and R inputs as long as the clock input is HIGH.

* In this way, one can clock the flip flop so as to store either a 1 by applying S=1, R=0 (ie, set) or a 0 by applying S=0, R=1 (ie, reset) at any time and hold that bit of information for any desired period of time by applying a LOW at the clock input.

Present state	Clock Pulse	Data Inputs		Next State	Action
Qn	CLK	S	R	Qn+1	
0	0	0	0	0	No change
1	0	0	0	1	No change
0	1	0	0	0	No change
1	1	0	0	1	No change
0	0	0	1	0	No change
1	0	0	1	1	No change
0	1	0	1	0	Reset
1	1	0	1	0	Reset
0	0	1	0	0	No change
1	0	1	0	1	No change
0	1	1	0	1	Set
1	1	1	0	1	Set
0	0	1	1	0	No change
1	0	1	1	1	No change
0	1	1	1	?	Forbidden
1	1	1	1	?	Forbidden

J-K Flip Flop

J-K flip flop has a characteristic similar to that of an S-R flip flop. In addition, the indeterminate condition of the S-R flip flop is permitted in it. Inputs J and K behave like inputs S and R to set and reset the flip flop respectively. When J = K = 1, the flip flop output toggles, ie, switches to its complement state; if Q = 1, it switches to Q = 0 and if Q = 0, it switches to Q = 1.



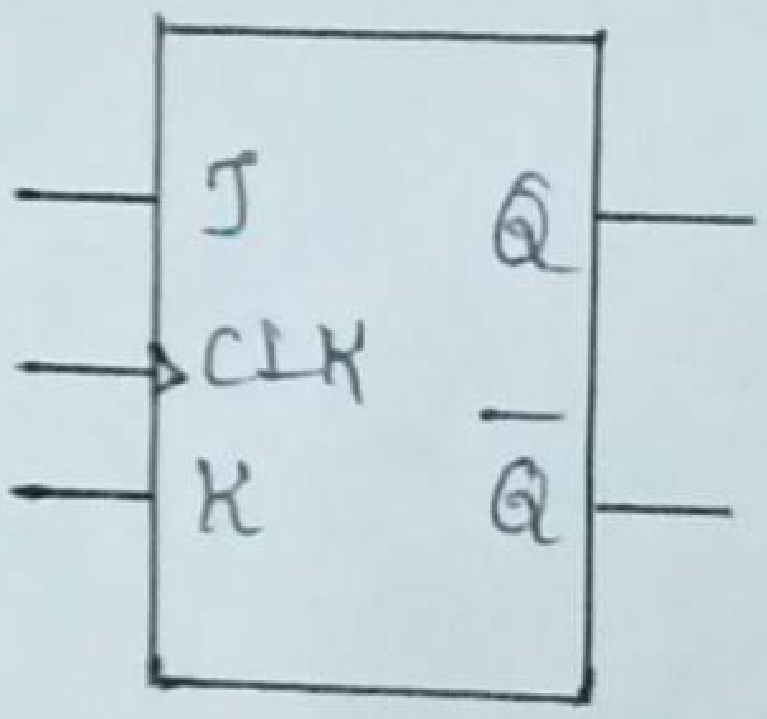


Fig: Graphic Symbol

Fig: J-K Flip Flop using SR Latch

JK flip flop can be obtained from the clocked SR flip flop by adding two AND gates. The data input J and the output $\overline{Q}$ are applied to the first AND gate, and its output ($J\overline{Q}$) is applied to the S-input of S-R flip flop.

Similarly, the data input K and output Q are applied to the second AND gate, and its output (K.Q) is applied to R input of SR flip flop.

D Flip Flop

D Flip Flop - Delay/Data Flip Flop

The delay flip flop has only one input called the Delayed (D) input and two outputs Q and $\overline{Q}$. It is constructed from an S-R flip flop by inserting an inverter between S and R and assigning the symbol D to the S input.

Basically D flip flop consists of a NAND flip flop with a gatting arrangement on its inputs.

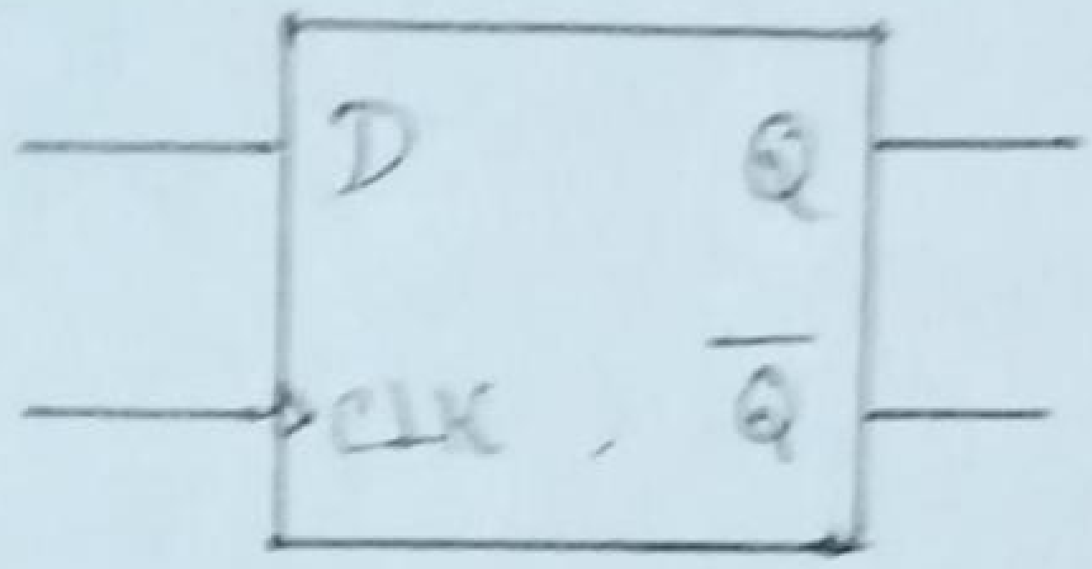


Fig: Logic Symbol

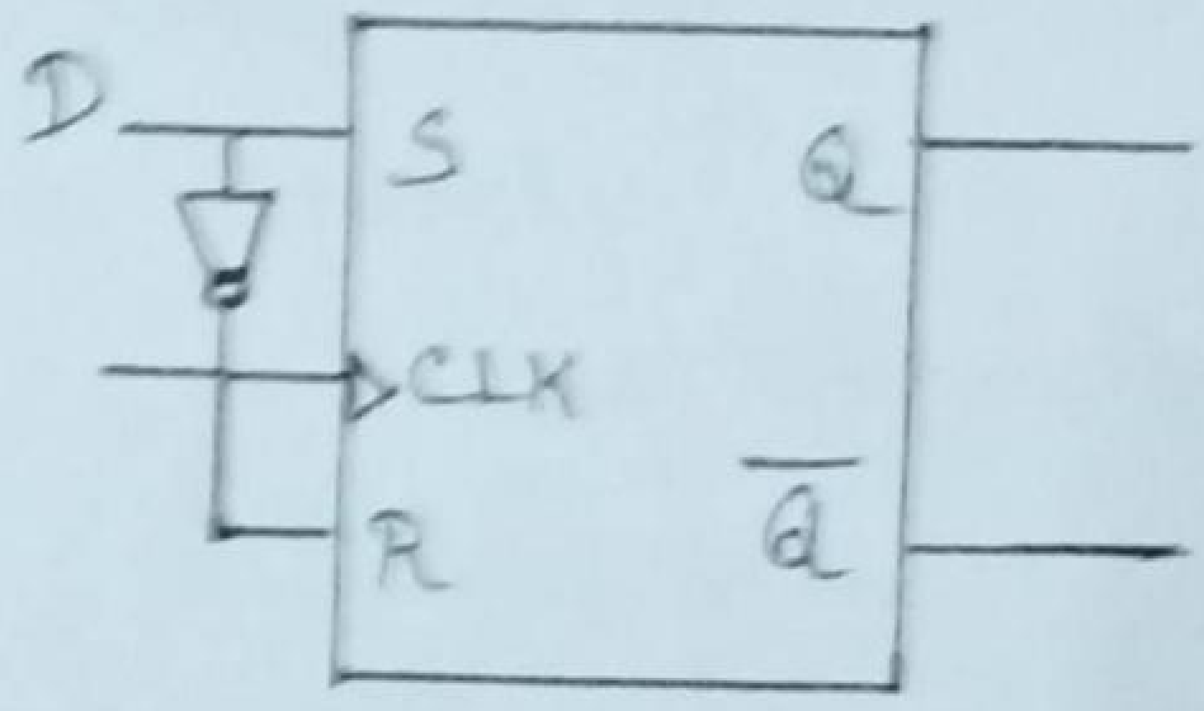


Fig: JK Flip Flop Using SR Flip Flop

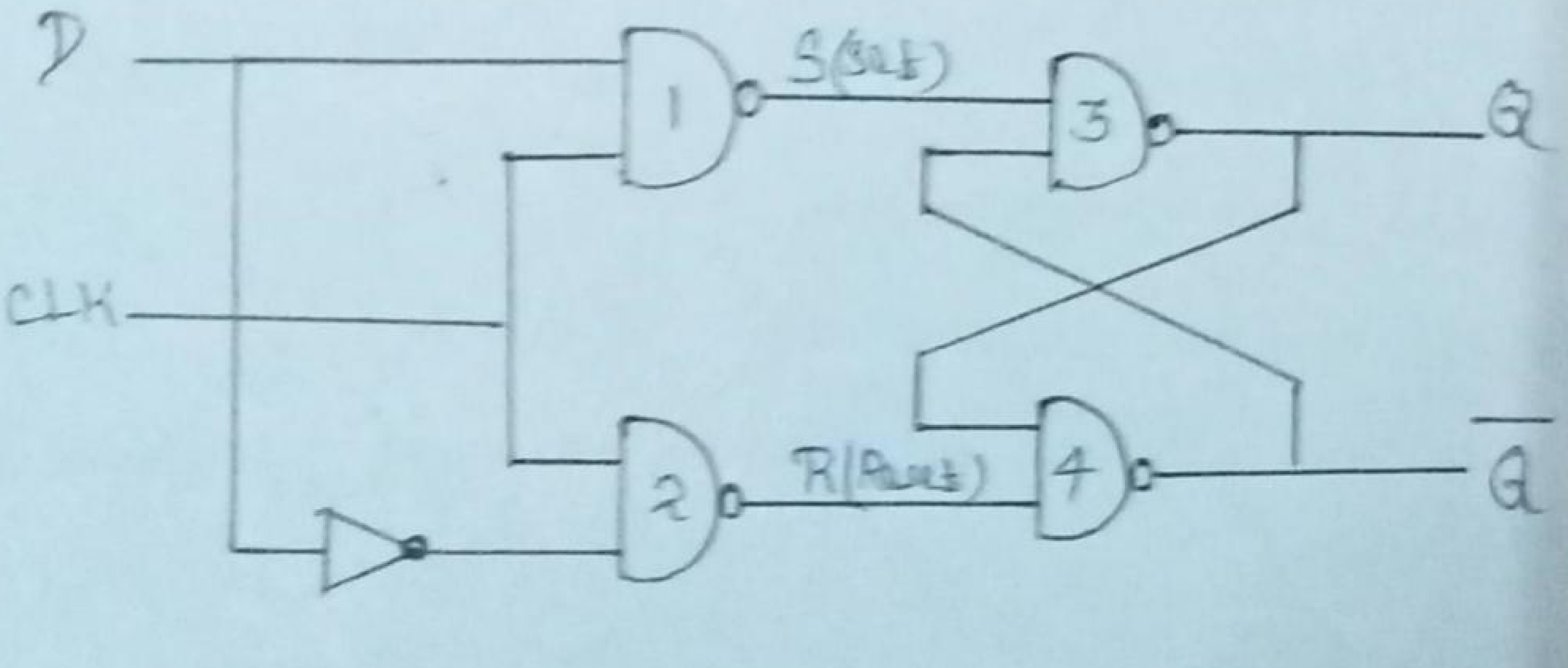


Fig: NAND Based JK Flip Flop

pulses to shift a bit from the input to the antput.

(ii) ~~In~~ Parallel Shifting

In parallel shifting operation, all the data (input or output) get shifted simultaneously during a single clock pulse.

- It is much faster than serial shifting.

* Shift registers are classified into the following four types based on how binary information is entered or shifted out:

1. Serial-in Serial-out (SISO)
2. Serial-in Parallel-out (SIPO)
3. Parallel in serial-out (PISO)
4. Parallel-in Parallel-out (PIPO).

I Serial-In-Serial-Out Shift Registers

* This type of shift register accepts data serially ie one bit at a time on a single input line

* Data may be shifted left (from low to high order bits) using shift left register or shifted right (from low to high order bits) using shift right registers.

* Shift Left Register

* A shift left register can be built by using J-K flipflop or D flip-flop.

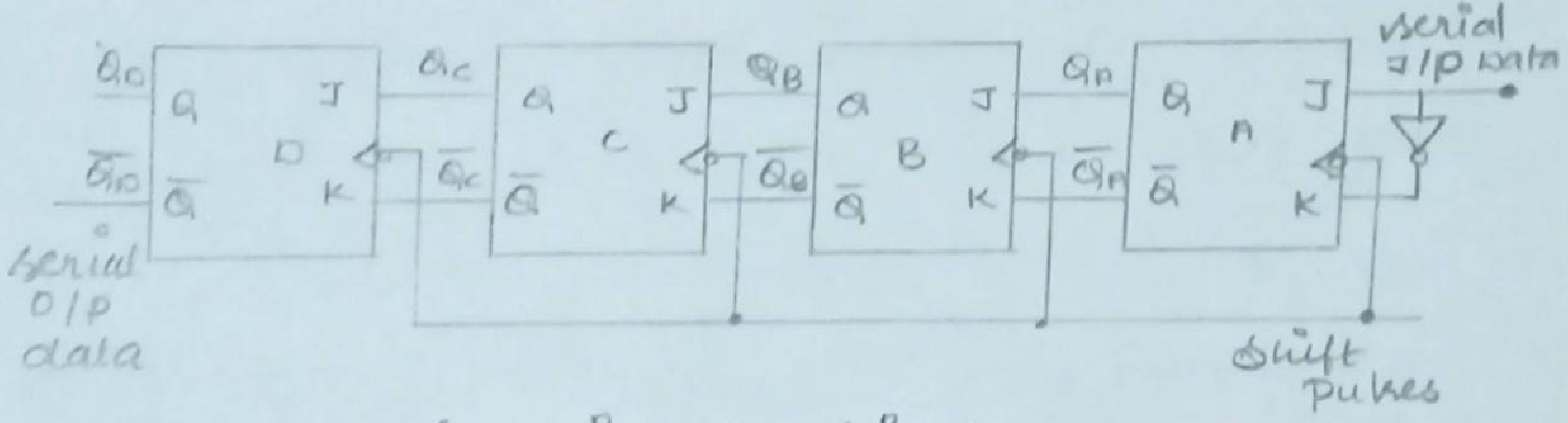
* A J-K flip flop based shift register requires connection of J and K inputs.

* In the shift registers using D flip flops delay (D) input of the rightmost flip-flop is used as a serial line

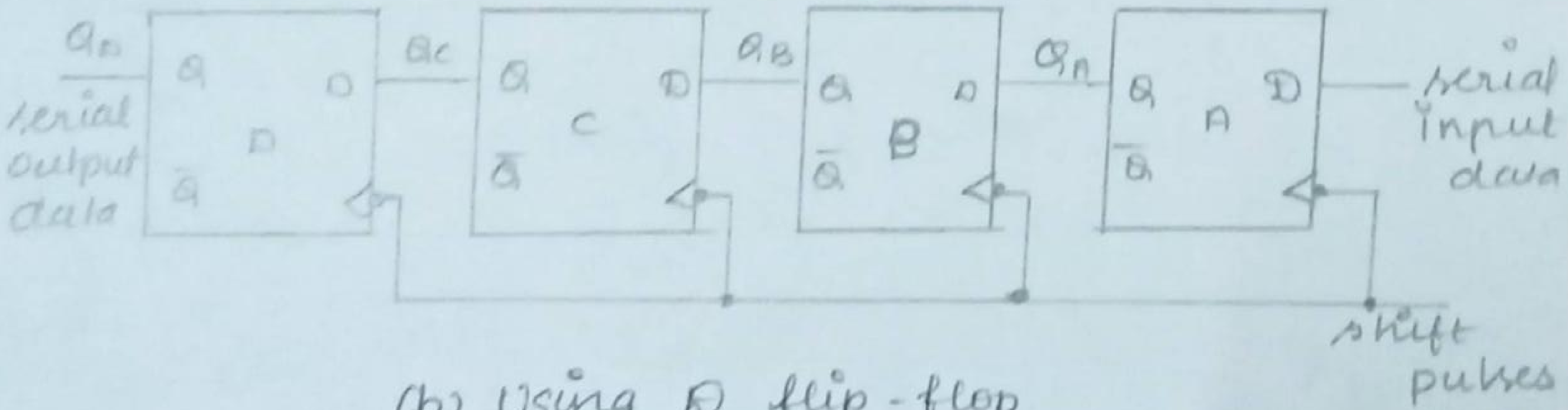
* The clock pulse is applied to all the flip flops simultaneously.

* When the shift or clock pulse

occurs, each flip-flop is set or reset according to the data at the respective flip-flop input.



(a) Using J-K flipflop



(b) Using D flip-flop

● Shift - Right registers

* A shift-right register can also be built using D flip-flops or J K flip-flops.

* when the first clock pulse is applied, flip-flop A is SET Thus storing the 1

Next, a 0 is applied to the serial input, making D=0 for flipflops A and D=1 for

[illegible]

[illegible]

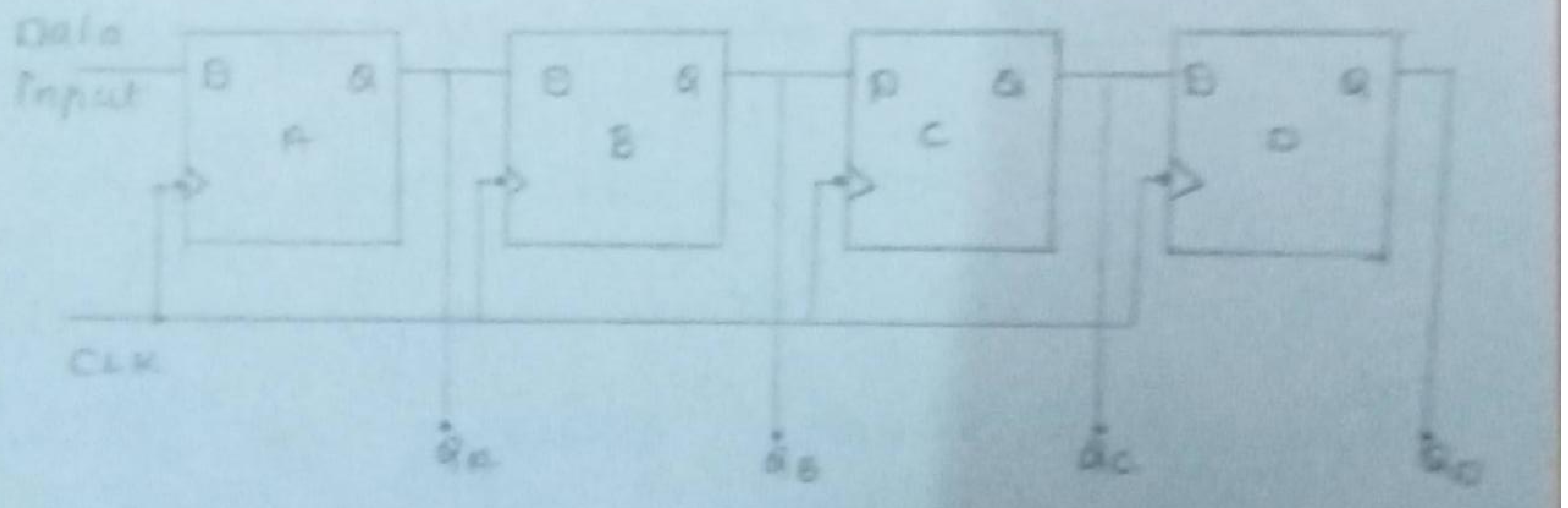
[illegible]

II Serial-In Parallel-out Shift Register

* It consist of one serial inputs and outputs are taken from all the flip-flops parallel

* In this register, data is shifted in serially but shifted out in parallel.

* Inorder to shift the data out in parallel, it is necessary to have all the data available at the outputs at the same time.



A 4-bit serial-in Parallel-out shift Registers

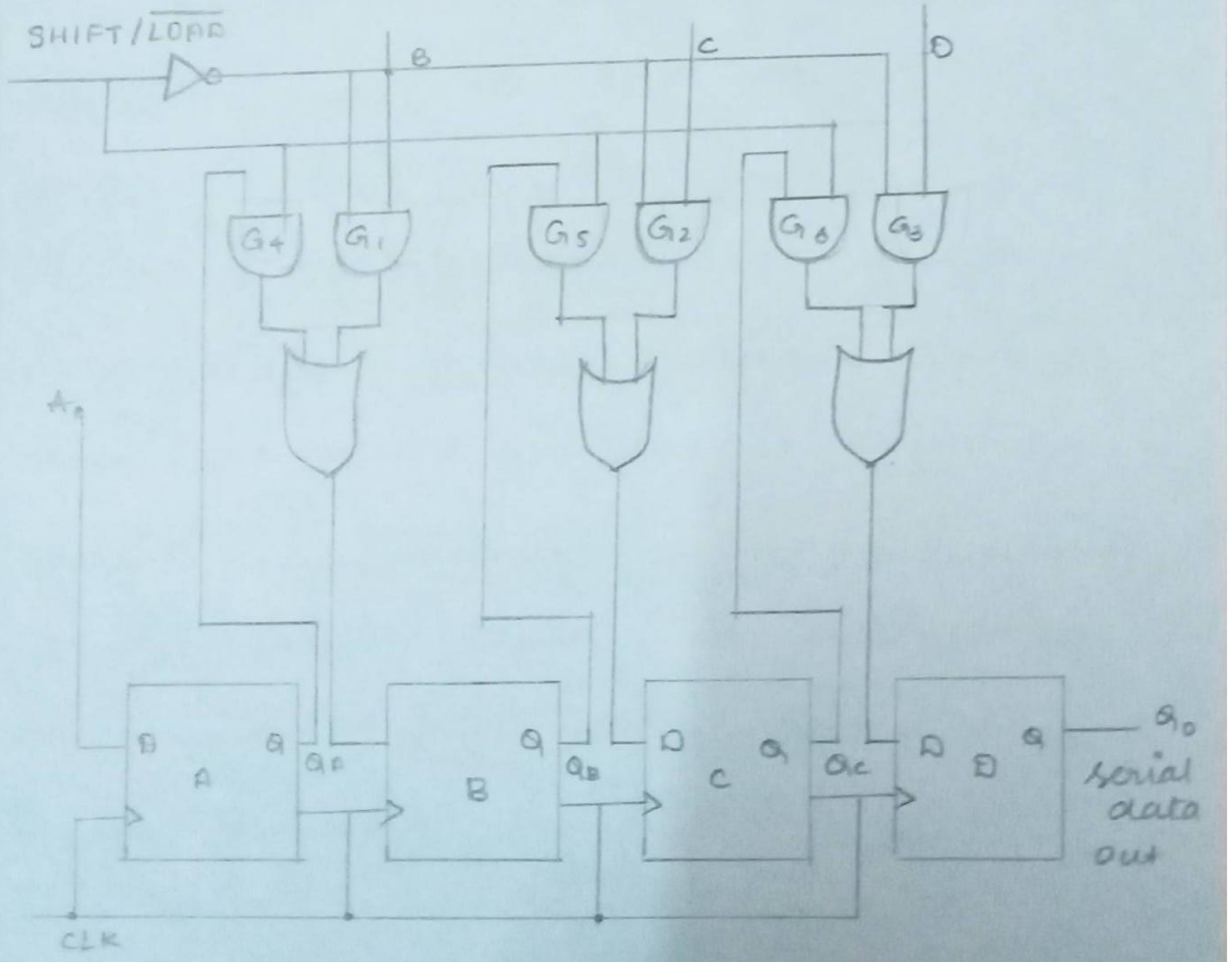
II Parallel in - Serial - out Shift Registers

* For a register with parallel data inputs, the bits are entered simultaneously into their respective flip-flops rather than a bit-by-bit basis on one line.

* Let A, B, C, D be the four parallel data-input lines and SHIFT/$\overline{LOAD}$ is a control input that allows the four bits of data at A, B, C and D inputs to enter into the register in parallel or shift the data in serial.

* When SHIFT/$\overline{LOAD}$ is low, AND gates G_1 through G_3 are enabled, allowing the data at parallel inputs A B, C and D to the D input of its respective flip-flop.

* when SHIFT/$\overline{\text{LOAD}}$ is HIGH, AND gates G_1 through G_3 are disabled and the remaining AND gates G_4 through G_6 are enabled, allowing the data bits to shift right from one stage to the next.



A 4-bit parallel-in-serial-out shift Registe

iv Parallel-in-Parallel-out Register

* In this type of register, data inputs can be shifted either in or out of the register in parallel

* The parallel entry of the data is carried out as like that of parallel-in serial-out shift registers

* In this register, there is no interconnection between successive flip-flops since no serial shifting is required. Therefore the moment of the parallel entry of the input data is accomplished, the respective bits will appear at the parallel outputs.

⊛ Universal Shift Registers

If the register has shift or parallel load capabilities then it is called a shift register with parallel load or universal shift registers.

* A register which is capable of shifting data to both the direction is called a bidirectional shift registers.

* A register that can shift in only one direction is called a uni-directional shift registers.

* shift registers can be used for converting serial data to parallel data and vice versa

* Some shift registers have necessary input and output terminal and also have both shift - right and shift-left capabilities.

* The most general shift registers has the capabilities listed below:

1. A clear control to clear the registers to 0.

2. A clk input for clock pulses to synchronize all operations

3. A shift right control to enable to 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 the serial input and output lines associated with the shift left.

5. A parallel load control to enable a parallel transfer and the n input lines associated with the parallel transfer.

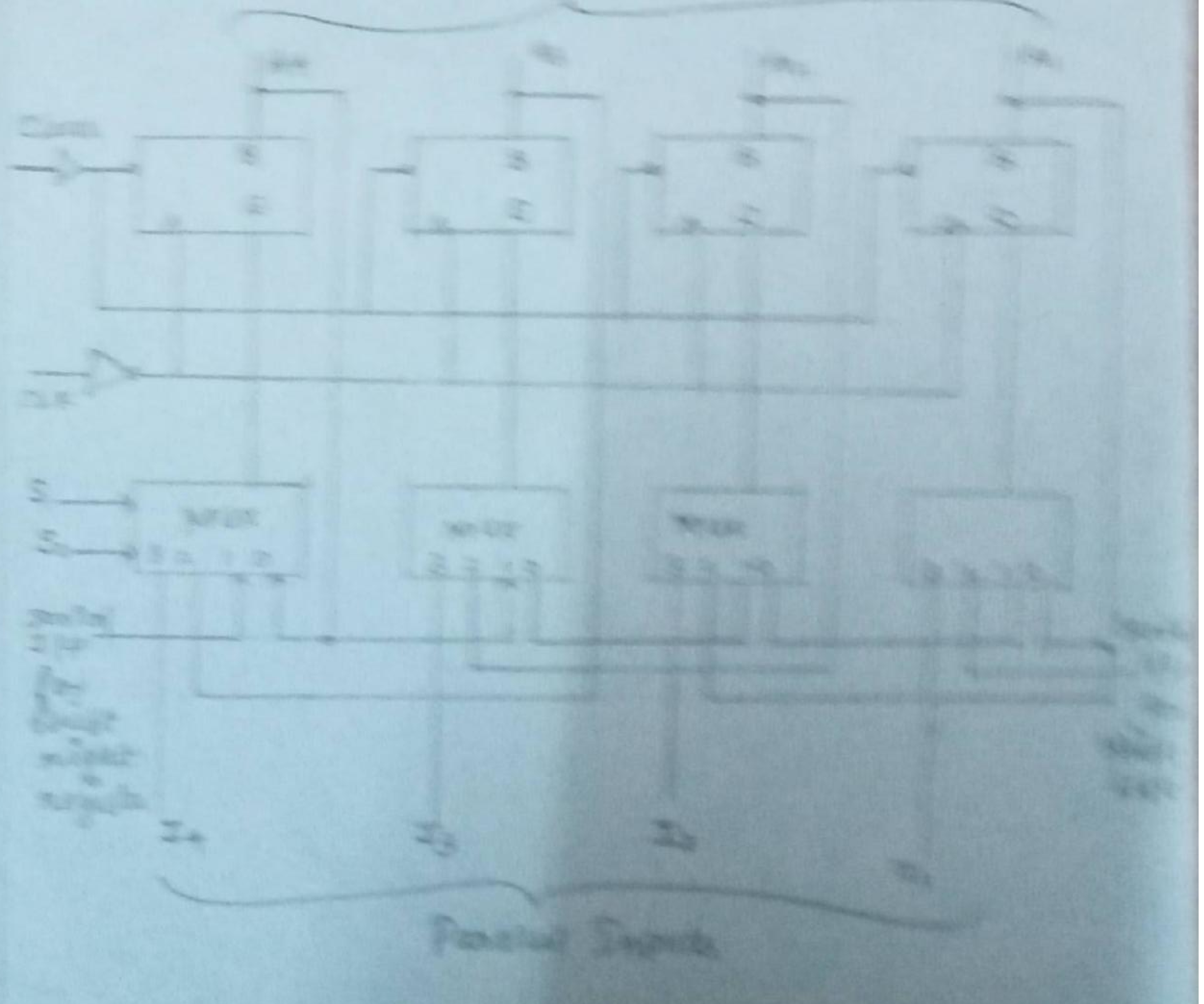
6. n parallel output lines.

7. A control line that leaves the information in the register unchanged even though clock pulses are continuously applied.

(i) Function table of bidirectional shift register.

Mode control		Register Operation
S_1	S_0	
0	0	No change
1	0	shift right
0	1	shift left
1	1	Parallel load

(ii) 4-bit Universal shift register



STABLE AND UNSTABLE STATES

Stable State - Fixed/Unchanging/Permanent State

A state or condition in which a system will remain unless or until it is disturbed by some outside force.

A stable state is one that will persist/continue indefinitely until the input changes.

A stable state is one that will be maintained even when the inputs revert to their inactive/neutral level.

The stable states in a flow table have specific output values associated with them.

The unstable states have unspecified output values denoted by a dash.

A system is in unstable state, when it will not return into the original location, when being displaced but instead passes into a new state of stable state.

ASYNCHRONOUS SEQUENTIAL CIRCUITS

Sequential circuits without clock pulses are called asynchronous sequential circuits.

Types

1. Fundamental mode asynchronous sequential circuits
2. Pulse mode asynchronous sequential circuits

Design of Fundamental Mode Asynchronous sequential Circuits.

- In fundamental mode circuits, the inputs and outputs are represented by levels rather than pulses.
- In the design of fundamental mode circuits, it is assumed that a change occurs in only one of the inputs and no change occurs in any other inputs until the circuit enters a stable state.

Design Procedure

Step 1: From the verbal description of the problem, formulate precisely what the circuit has to do and develop a state diagram specifying the Next state, Output

Step 2: Develop a primitive state table with Present state (PS), Next state (NS) and output (Z) from the state diagram obtained in step 1.

Step 3: Identify the redundant states by using the implicant chart and minimize the primitive state table by merging process.

Step 4: State Assignment – Assign state variables (secondary variables) to the rows of merged primitive state table and obtain Present State / Next State and Output table.

step 5: Excitation and Output table – Decide the flip flop to be used and obtain excitation and output table.

step 6: Excitation and Output map – Obtain the simplified expression for the excitation and output function by using K-map.

step 7: Circuit Diagram – Draw the schematic diagram.

Design of Pulse mode Asynchronous Sequential Circuits

The design of pulse mode asynchronous sequential circuits similar to the design of synchronous sequential circuits.

- In pulse mode sequential circuits, the inputs are pulses
- In this circuit, it is assumed that no two pulses will arrive at the same time
- Also, it is assumed that the duration of the pulses is long enough to cause state transition and is short enough so that there will not be more than one state transition for a single pulses.

The design procedure employed for fundamental mode circuits are also applicable to pulse mode asynchronous circuits.

PROBLEMS IN ASYNCHRONOUS CIRCUITS

The asynchronous sequential circuits have three disadvantages or problems,

1. Cycles
2. Races
3. Hazards

Cycles

If an input change induces a feedback transition through more than one unstable state, then such a situation is called a cycle.

- circuit goes through a unique sequence of unstable states = cycle

Row	NS			
	$x_1x_2 =$ 00	01	11	10
1	①			
2	1			
3	1			
4	1	④		

Table a: PS/NS Table

PS	NS (Y_1Y_2)			
y_1y_2	$x_1x_2 =$ 00	01	11	10
00	00 00			
01	01			
11	11			
10	10			

Races

In the case of cycle, only one feedback variable is unstable at any time for a change in an input variable. When two or more feedback variables change value in response to a change in an input variable, then a race condition is said to exist in an asynchronous sequential circuits.

Types

1. Critical Race
2. Non-Critical Race

PS	NS ($Y_1 Y_2$)			
$y_1 y_2$	$x_1 x_2 =$ 00	01	11	10
00	11 (1)	00 (5)	11 (9)	11 (13)
01	01 (2)	11 (6)	11 (10)	00 (14)
11	10 (3)	11 (7)	11 (11)	10 (15)
10	10 (4)	00 (8)	11 (12)	10 (16)

Hazards

[illegible]

Types

1. Static hazard

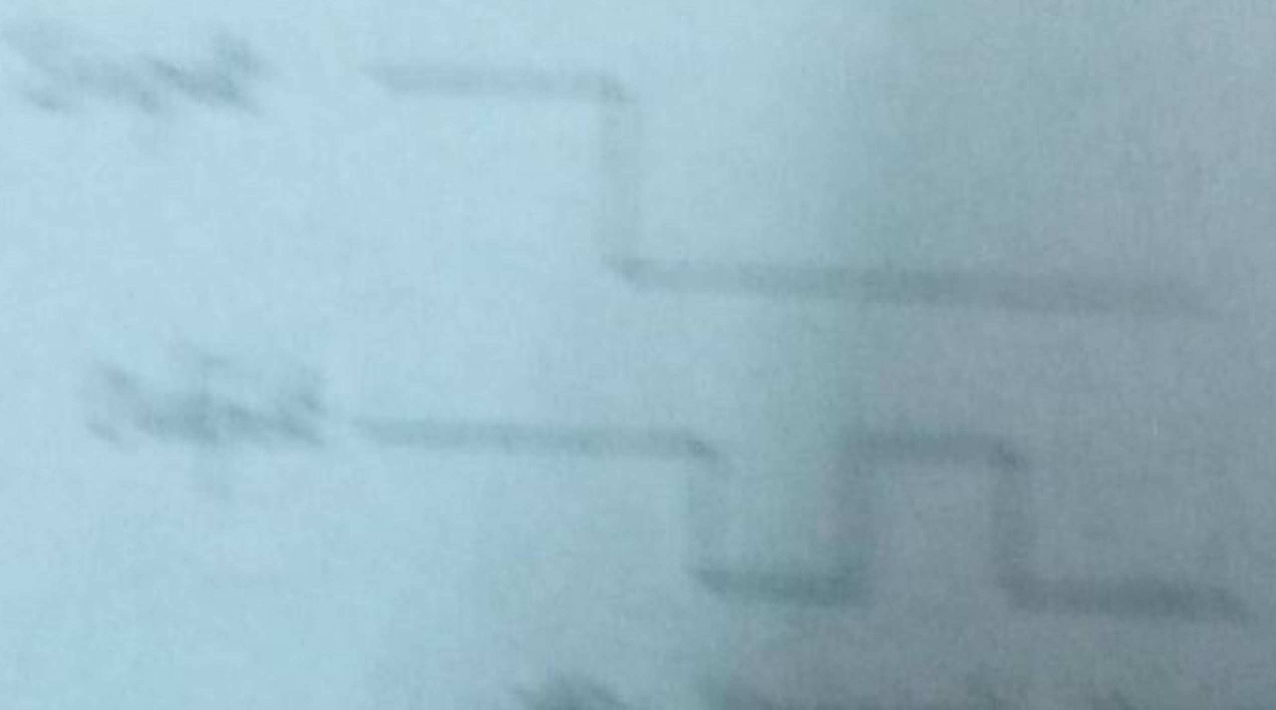
2. Dynamic hazard

Input
Output
Input
Output

static-1 hazard static-0 hazard

Fig: static hazard

[illegible]



DESIGN OF HAZARD FREE CIRCUITS

ad

Static Hazards Elimination

A transition between a pair of a adjacent input combinations which correspond to identical outputs (ie, both are '0's and 1's) contains a static hazard if it leads to the generation of momentary spurious output. Such hazards occur whenever there exists a pair of adjacent input combinations which produce the same output and there is no sub cube in K-map covering both combinations.

To design a static hazard free circuit,

* All adjacent input combinations, having same output occur within some sub cube of the corresponding function

* Every pair of adjacent '1' cells and every pair of adjacent '0' cells in the K-map of a function should be covered by at least one sub cube

The expression derived from such a collection of sub cubes is called hazard free function and it leads to the implementation of hazard free circuit.

Consider the function,

$$f(A,B,C) = \Sigma(1,3,6,7)$$

Corresponding K-map,

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

Fig: K-map without hazard cover

From the K-map the simplified sum of product (SOP) expression is,

1 - uncomplement
0 - complement

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

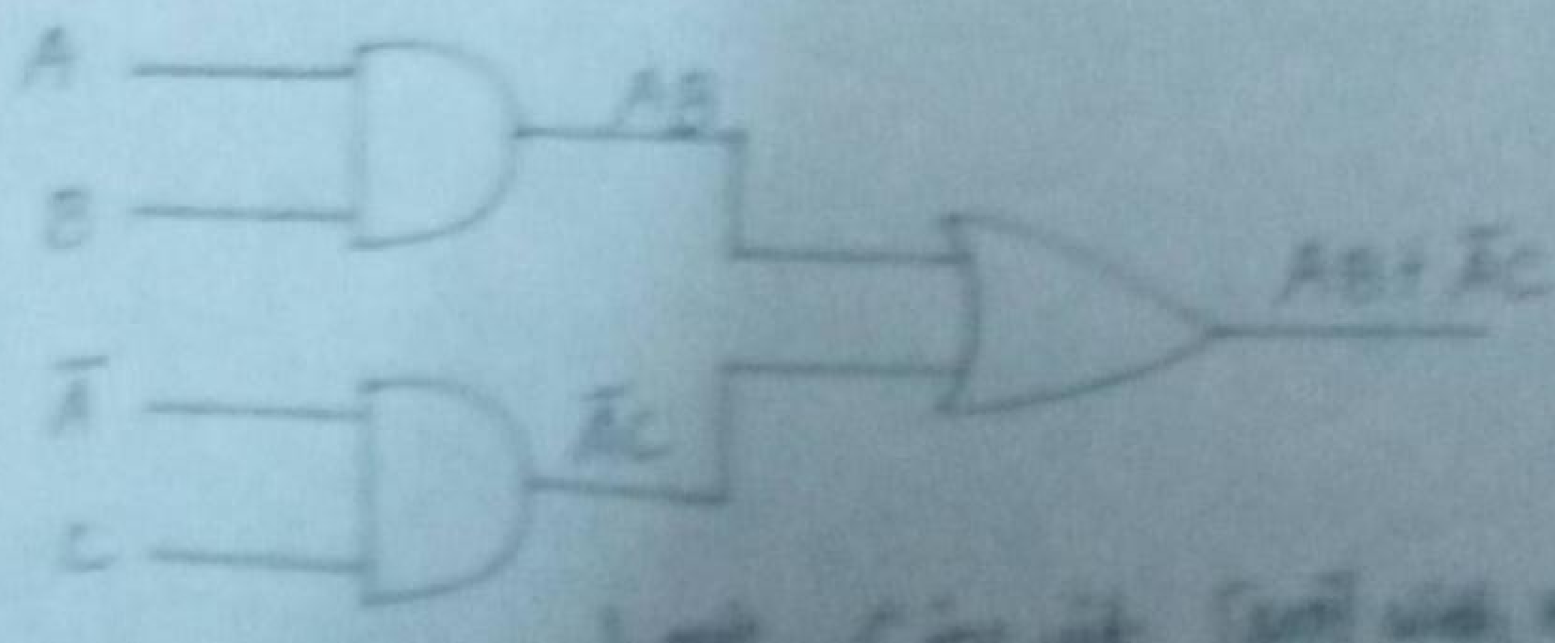
C \ AB	00	01	11	10
0	0	0	1	0
1	1	1	1	0

Fig: K-map without hazard cover

From the K-map the simplified product of sum (POS) expression is,

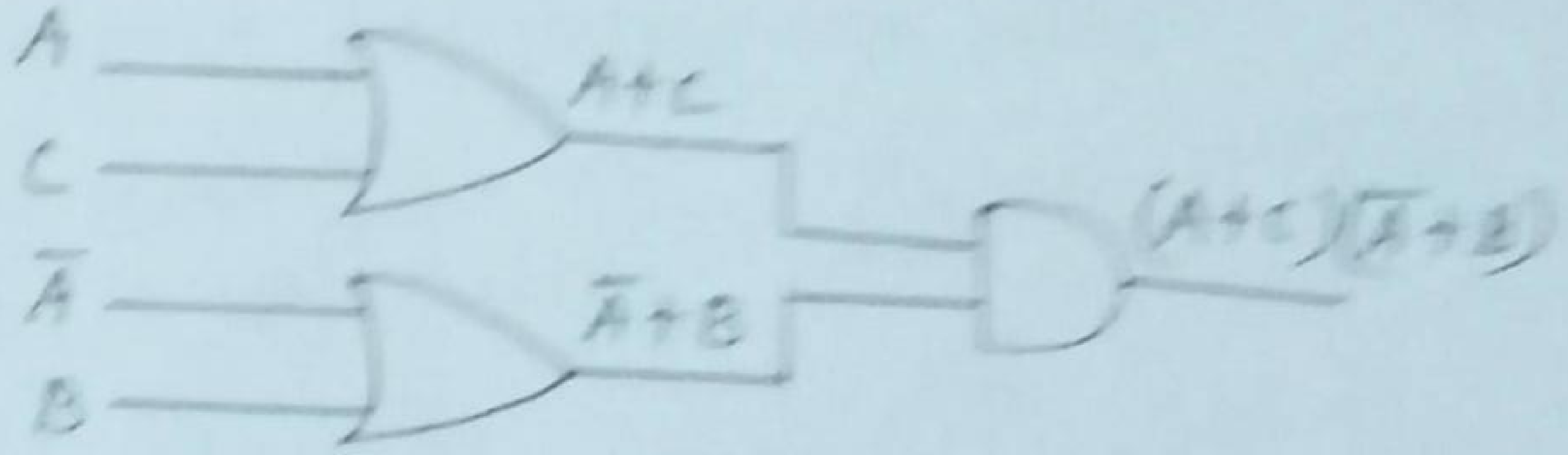
1 - complement
0 - uncomplement

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



Logic Circuit [SOP] with hazard

In the circuit, when the input (ABC) changes from 011 to 111 or 111 to 011, the output may momentarily go to '0' state due to unequal propagation delay in the AND gates, rather than remaining constant at '1' as per the function value.



Logic Circuit with hazard

III^{ly}

In this circuit, when the input changes from 000 to 100 or 100 to 000, the output may momentarily go to '1' state due to the unequal propagation delay in the OR gates, rather than remaining constant at '0' as per the function value.

These momentarily wrong outputs [Both SOP & POS circuit] are called glitches.

In order to make the logic circuit hazard free, every pair of adjacent '1' cells and every pair of adjacent '0' cells in the K-map should be covered by atleast one sub cube.

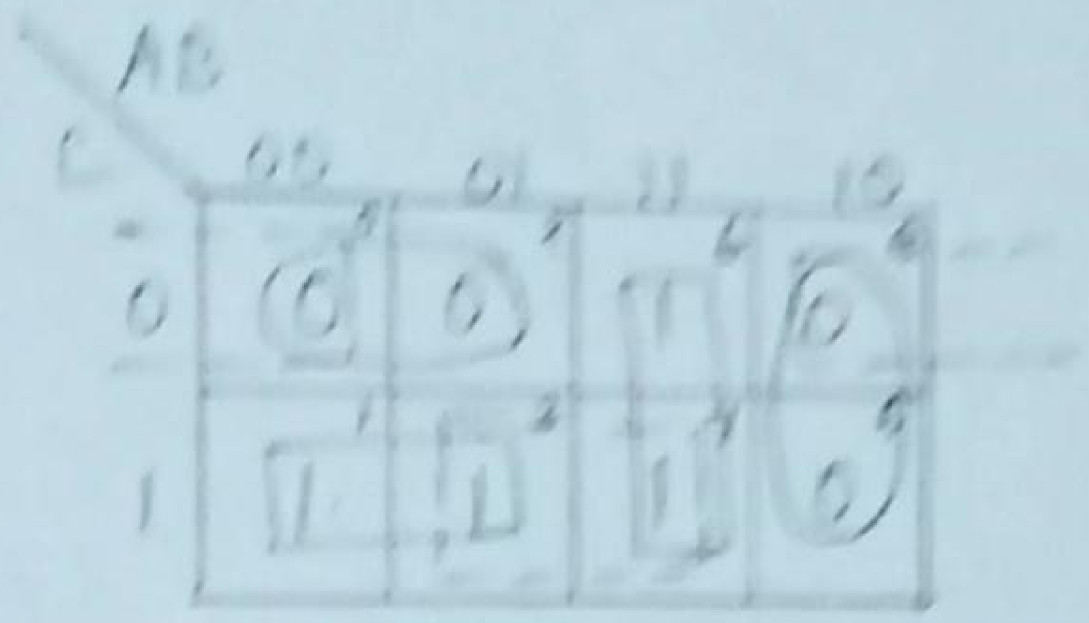


Fig: K-map with Hazard Cover

The hazard free expression is,

$$f = \underset{(SOP)}{AB + \bar{A}C + BC} \; ; \; f = \underset{(POS)}{(A+C)(\bar{A}+B)(B+C)}$$

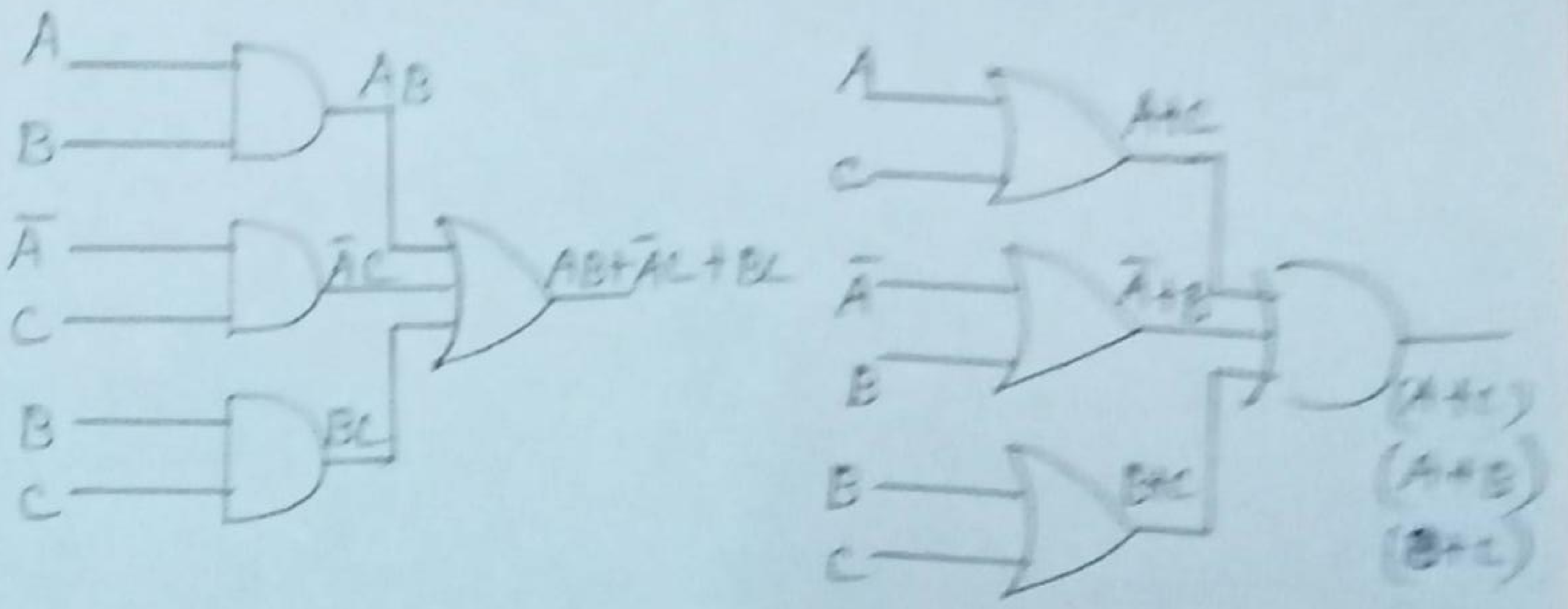


Fig: Hazard Free Logic Circuits

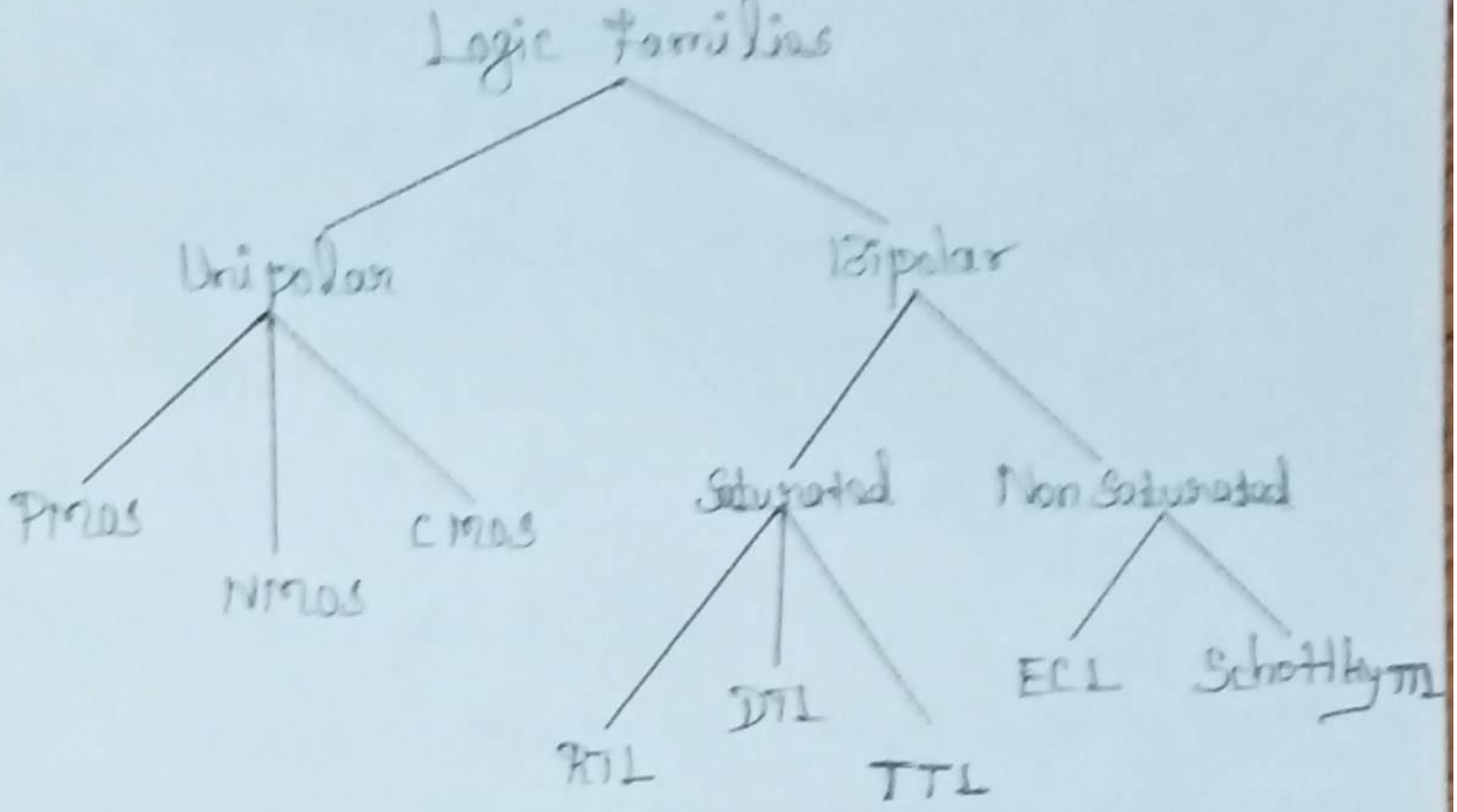
LOGIC FAMILIES

Logic family is a collection of different integrated circuit chips that have similar input, output and internal circuit characteristics but they perform different logic gate functions such as AND, OR, NOT ... etc..

Logic family may also refer to a set of techniques used to implement logic within very large scale integrated circuits such as central processors, memories or other complex functions.

* Some logic families use static techniques to minimize design complexity
* Some logic families use clocked dynamic techniques to minimize size, power consumption and delay
* Logic families may vary by speed, power consumption, cost, voltage and current levels

Classification



PMOS → P-Channel Metal Oxide Semiconductor

NMOS → N-Channel Metal Oxide Semiconductor

CMOS → Complementary Metal Oxide Semiconductor

RTL → Resistor Transistor Logic

DTL → Diode Transistor Logic

TTL → Transistor Transistor Logic

Logic family is a circuit technology that can be used to create many different types of gates [OR, AND, NOT, NAND ....]

Resistor Transistor Logic (RTL).

Resistor Transistor Logic is a class of digital circuits built using resistors as the input network and bipolar junction transistors as switching devices.

Ex: RTL NOR gate (Fig)

Construction and Working

The basic diagram of RTL NOR gate consisting of transistors and resistors. When the inputs A, B and C are at 0V [logic 0], the transistors are turned OFF. Hence the output goes to +Vcc ie, logic 1.

If either one or all input terminals are at +Vcc [logic 1], one transistor or all would be fully turned ON, thereby reducing the output voltage to almost 0V.

It is seen that the output is at logic 1 only when all the inputs are at logic 0 and the output is logic 0 either one or all the inputs are at logic 1, ie, the NOR logic function.

RESISTOR TRANSISTOR LOGIC

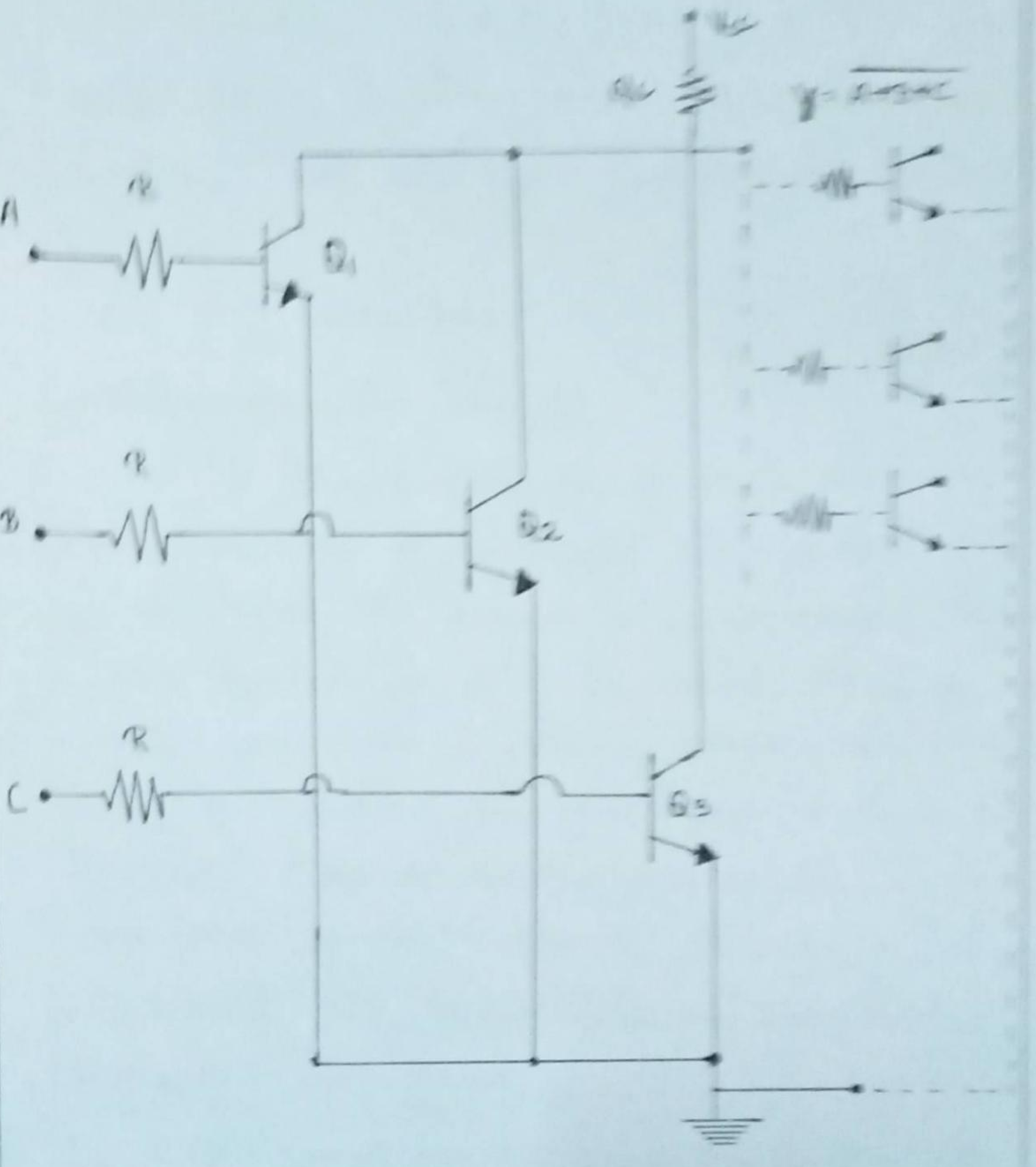


Fig: Block Diagram of RTL NOR Gate

TRANSISTOR - TRANSISTOR LOGIC (TTL)

Transistor - Transistor logic is a logic family built from bipolar junction transistors. Transistors perform both the logic function and the amplifying function.

Ex: TTL NAND Gate

(5M)

Construction and Working

The TTL circuit uses a single multi-emitter transistor that is fabricated with several emitters at its input. The number of emitters used depends on the desired fan-in of the circuit. Since a multi-emitter transistor is smaller in area than the diodes it replaces, the yield from a wafer is increased. Moreover, smaller area results in lower capacitance to the substrate, thereby reducing the circuit rise and fall times and hence increasing its speed.

Four transistors Q_1, Q_2, Q_3 and Q_4 are used. The output is taken from the collector of transistor Q_4. Each emitter of Q_1 acts like a diode. Therefore, transistor Q_1 and the 4kΩ resistor act like a 3-input AND gate and the rest of the circuit inverts the signal. Hence, the overall circuit acts like a three input NAND gate.

TRANSISTOR - TRANSISTOR Logic

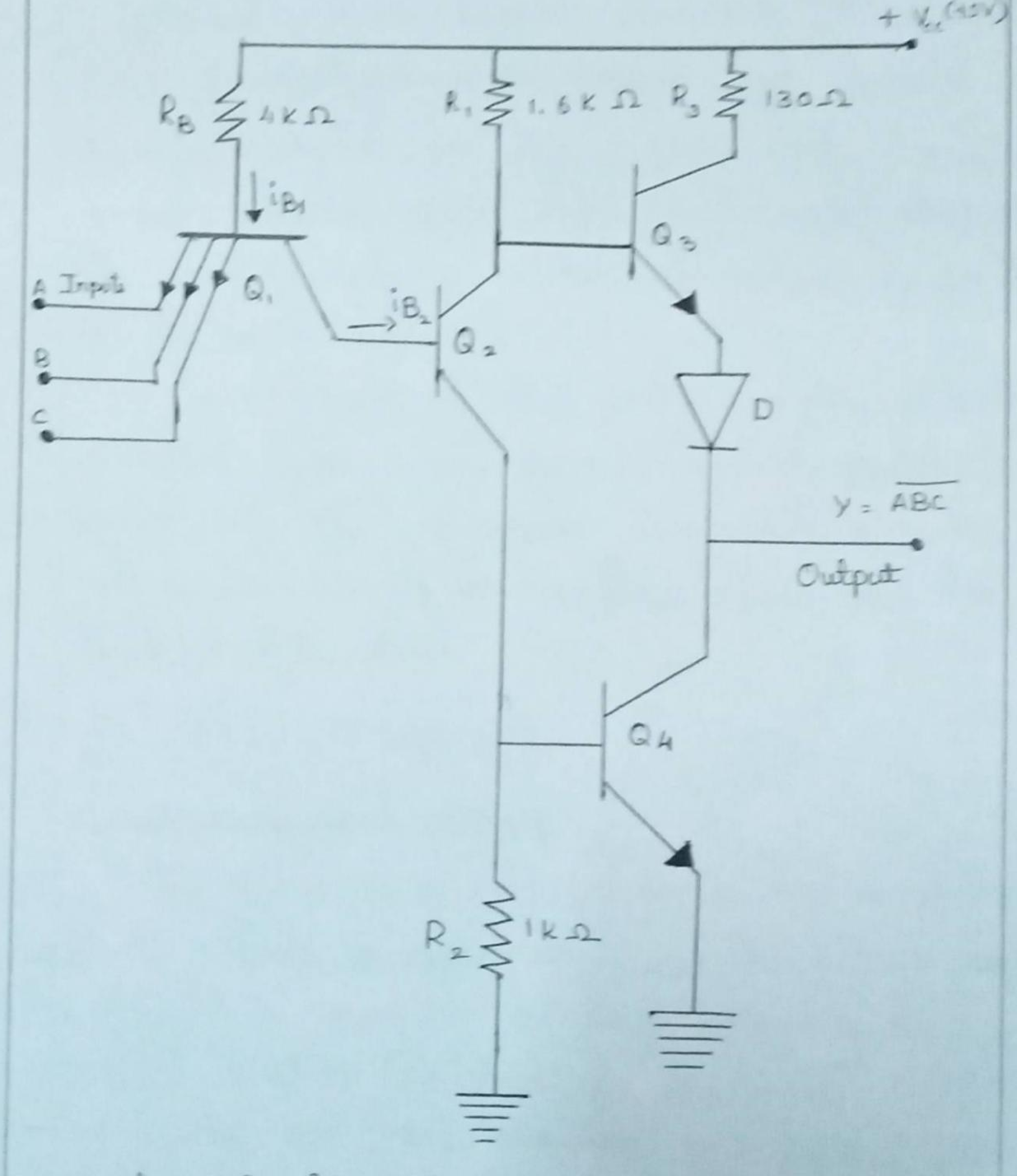


Fig: Circuit Diagram of TTL NAND Gate

EMITTER - COUPLED LOGIC [ECL]

Emitter - coupled logic is a high speed integrated circuit bipolar transistor logic family. ECL is considered as the fastest logic family. Emitter - coupled logic is a current - Mode logic [CML] or non - saturated digital logic family, which eliminates the turn - off delay of saturated transistors by operating in the active mode.

At present, the ECL family has the fastest switching speed among the commercially available digital ICs. The propagation delay time of a ECL gate is 1ns. Also it requires large silicon area and dissipates high power.

Ex: ECL OR/NOR Gate

JVA

Construction and Working

The basic ECL circuit can be used as an inverter if the output is taken at $V_{out\,1}$. This circuit can be expanded to more than one input by making transistor Q_1 parallel to other transistors for other inputs. By connecting one more transistor Q in parallel with Q_1, the circuit becomes a two-input ECL OR/NOR gate with inputs A and B.

EMITTER COUPLED LOGIC

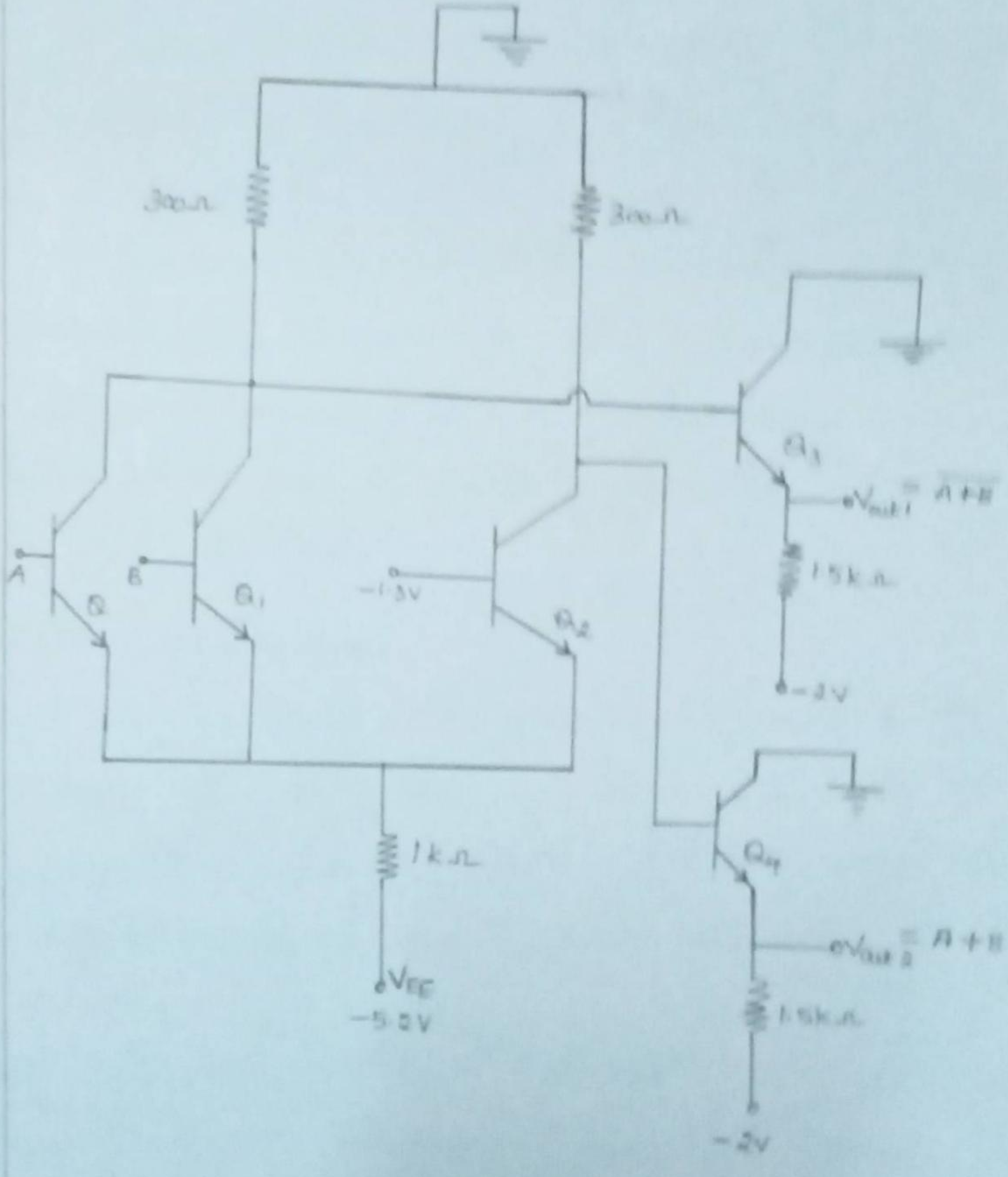


Fig: Circuit Diagram of ECL OR/NOR gate

COMPLEMENTARY METAL OXIDE SEMICONDUCTOR LOGIC (CMOS)

A circuit that uses complementary pairs of p-channel and n-channel MOSFETs is called CMOS family.

Complementary metal oxide semiconductor is a type of metal oxide semiconductor field effect transistor fabrication process that uses complementary and symmetrical pairs of p-type and n-type MOSFETs for logic functions.

IVA

CMOS NAND Gate

A two input NAND gate which consists of two p-type units in parallel and two n-type units in series. Transistors Q_1 and Q_2 form one complementary connection and Q_3 and Q_4 form another complementary connection.

If both inputs are HIGH, both p-channel transistors turn OFF and both n-channel transistors turn ONN. The output has a low impedance to ground and produces a LOW state.

If any input is low, the associated n-channel transistor is turned OFF and the associated p-channel

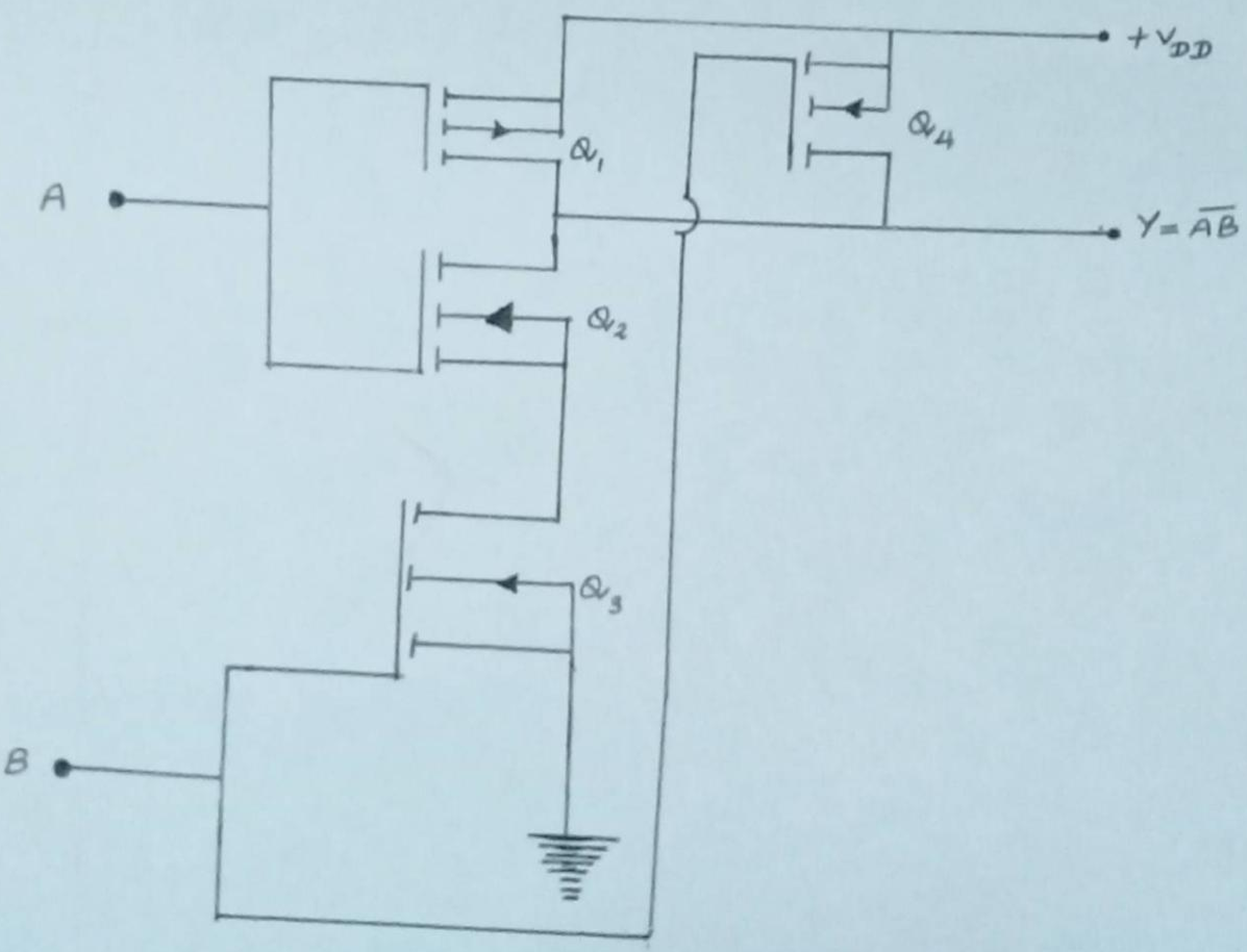


Fig: Circuit Diagram of CMOS NAND Gate

transistor is turned ON. The output is coupled to V_{DD} and goes to the HIGH state. This functions as a logic NAND gate.

PROGRAMMABLE LOGIC ARRAY [PLA]

Programmable Logic Array is a type of fixed architecture logic devices with programmable AND gates followed by programmable OR gates. The PLA is used to implement a complex combinational circuit.

In VLSI design, PLAs are used because the area required by the regular AND and OR array is less than the area required by randomly interconnected gates. Several PLAs include storage elements such as flip flops on the silicon chip, so that they can be used for sequential applications.

PLA is similar to a ROM in concept except that it does not provide full decoding of the variables and does not generate all the minterms as in the ROM. Thus, in a PLA, the decoder is replaced by a group of AND gates, each of which can be programmed to produce a product (AND) term of the input variables.

The AND and OR gates inside the PLA are initially fabricated with fuse among them. The specific boolean functions are implemented in sum-of-products (SOP) form by blowing appropriate fuses and leaving the desired connections. It is similar to the reprogramming of ROMs.

PROGRAMMABLE ARRAY LOGIC [PAL]

Programmable array logic is a type of fixed architecture logic devices with programmable AND gates followed by fixed OR gates. Because only the AND gates are programmable, the PAL is easier to program, but is not as flexible as the PLA. The PAL has the same AND and OR arrays. JVR

In PAL, the inputs to the AND gates are programmable, while the inputs to the OR gates are hard wired. Every AND gate in a PAL can be programmed to generate any desired product of the input variables and their complements. Each OR gate is hard-wired to only selected AND gate outputs.

Ex: 4-Inputs and 4-outputs PAL

Xs on the input side of AND gates represent the fusible links and Xs on the output side of AND gate are fixed connections. This limits each output function to sum of four product terms, it cannot be implemented with this PAL, one having more OR inputs would have to be used. If fewer than four product terms are required, the unneeded 1s can be made zero.

Read only Memory (ROM)

* A read only memory (ROM) is a semiconductor memory device used to store the information permanently. It performs only read operation and does not have a write capability.

* A ROM is programmed for a particular purpose for manufacturing process & user cannot alter its function.

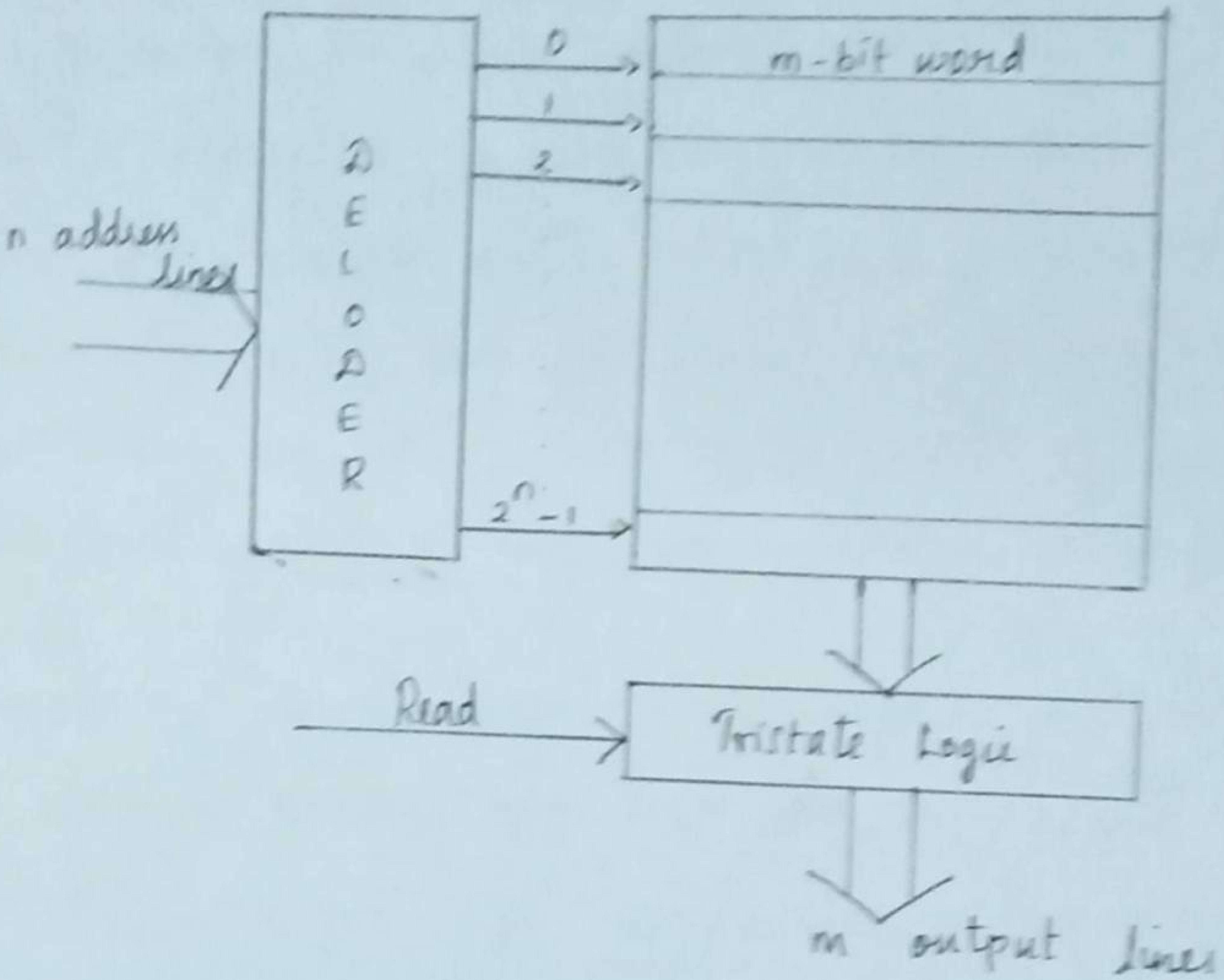
* The ROM is a combinational logic circuit. It includes both the decoder and OR gates within single IC package

* The ROM is used to implement complex combinational circuits within one IC package or as permenant storage package for binary information

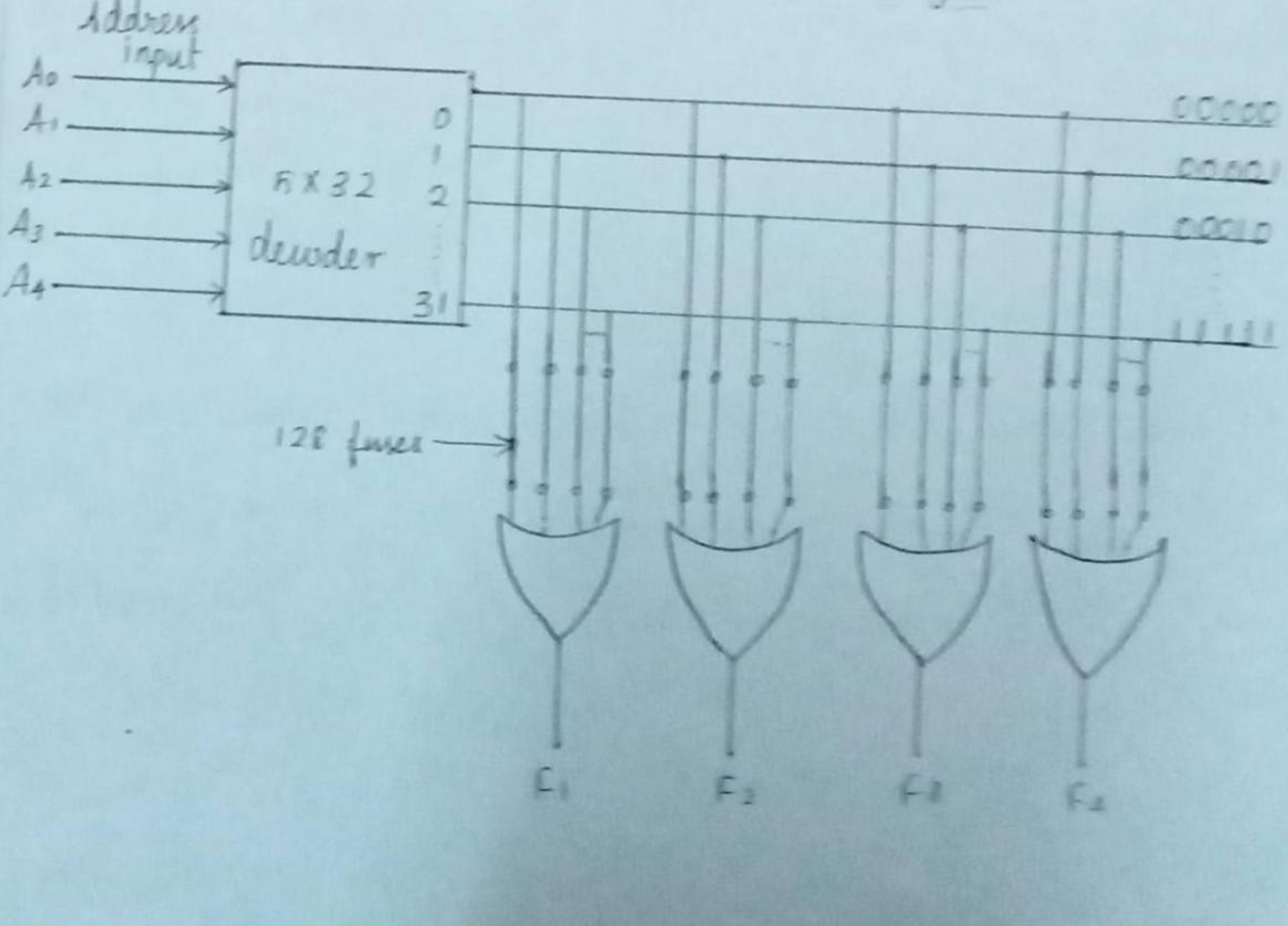
Types:

* PROM
* EP ROM
* EEPROM
* EAPROM

Block diagram of ROM



Logic construction of a 32x4 ROM using OR gates



PROM (Programmable ROM)

* In order to provide some flexibility in the possible applications of ROM, programmable ROMs (PROMs) have been introduced. The PROM can be programmed electrically by the user but cannot be reprogrammed

* PROMs are widely used in the control of electrical equipment such as washing machines and electric ovens

* PROMs are available in both bipolar and MOS technologies PROMs have 4bit or 8bit output words formats with capacities ranging in excess of 25000bits

Fuse technology used in PROM. The fuse links are placed between the emitter of each cell's transistor and its column line.

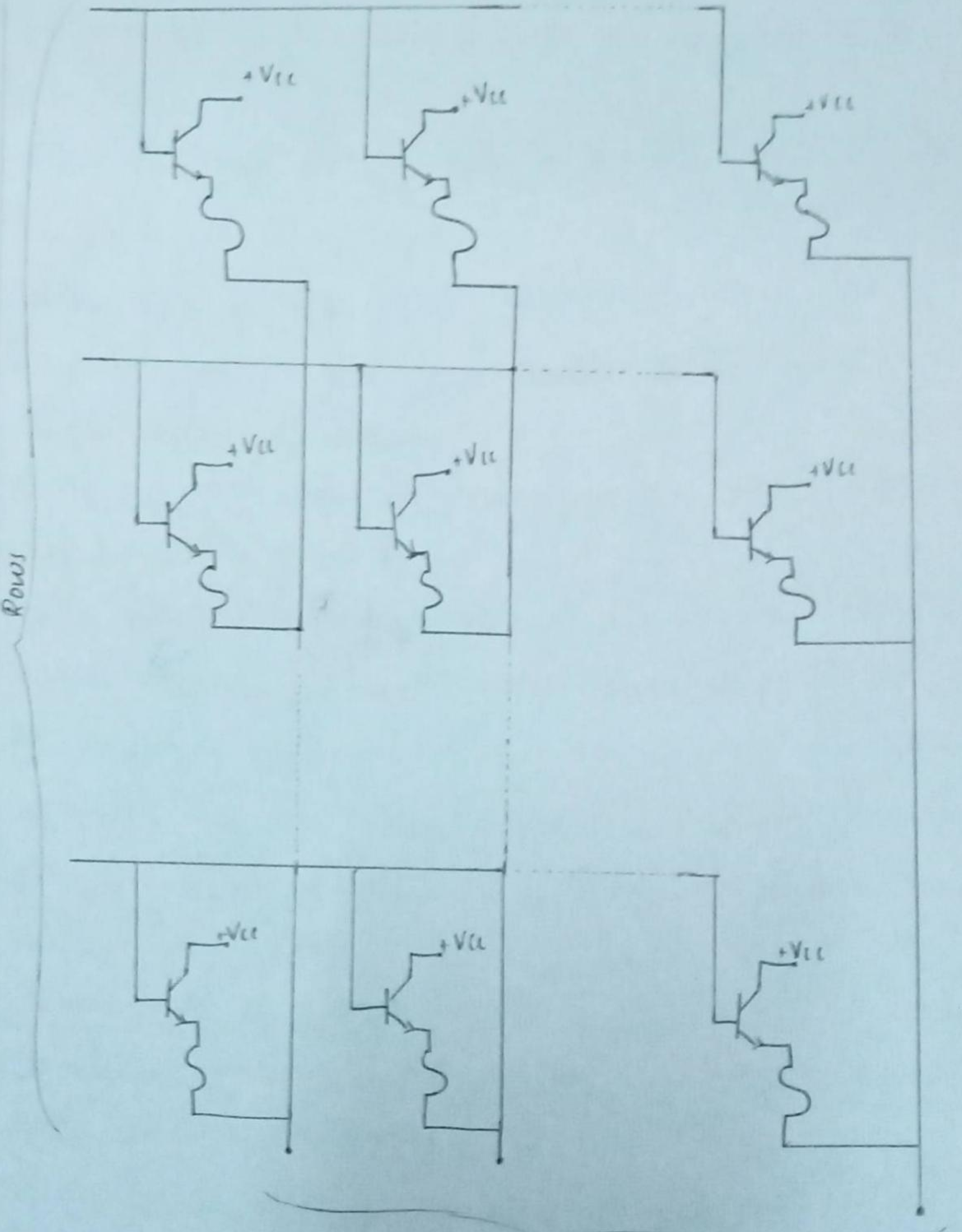
(i) metal links
(ii) silicon Links
(iii) P-n junctions

* The abone fuse technologies are irreversible The following MOS technologies used for the fabrication of programmable memories

(i) FAMOS ROM
(ii) MAOs PROM

* IC74186 is a TTL LSI 512-bit PROM. It is organized as 64 words of 8 bit each

Bipolar PROM array with fusible links



Erasable Programmable ROM (EPROM)

* A PROM device that can be erased and reprogrammed is called Erasable PROM. It uses an array of n-channel enhancement type MOSFETs with an insulated gate structure

* An additional floating gate is formed within the silicon dioxide (SiO_2) layer

* The floating gate is left unconnected while the normal control gate is connected to the row decoder output of EPROM

* The initial values of unprogrammed EPROM cells may be all 0s or all 1s

* To program a diffrent data, all cells in the EPROM must be erased This is done by illuminating the cells by a strong ultraviolet light having a wavelength.

* The IC 2764 is a 8 kB or 65536-bit EPROM organised as 8192 words of 8 bits each. It has 13 address lines and 8 data lines.

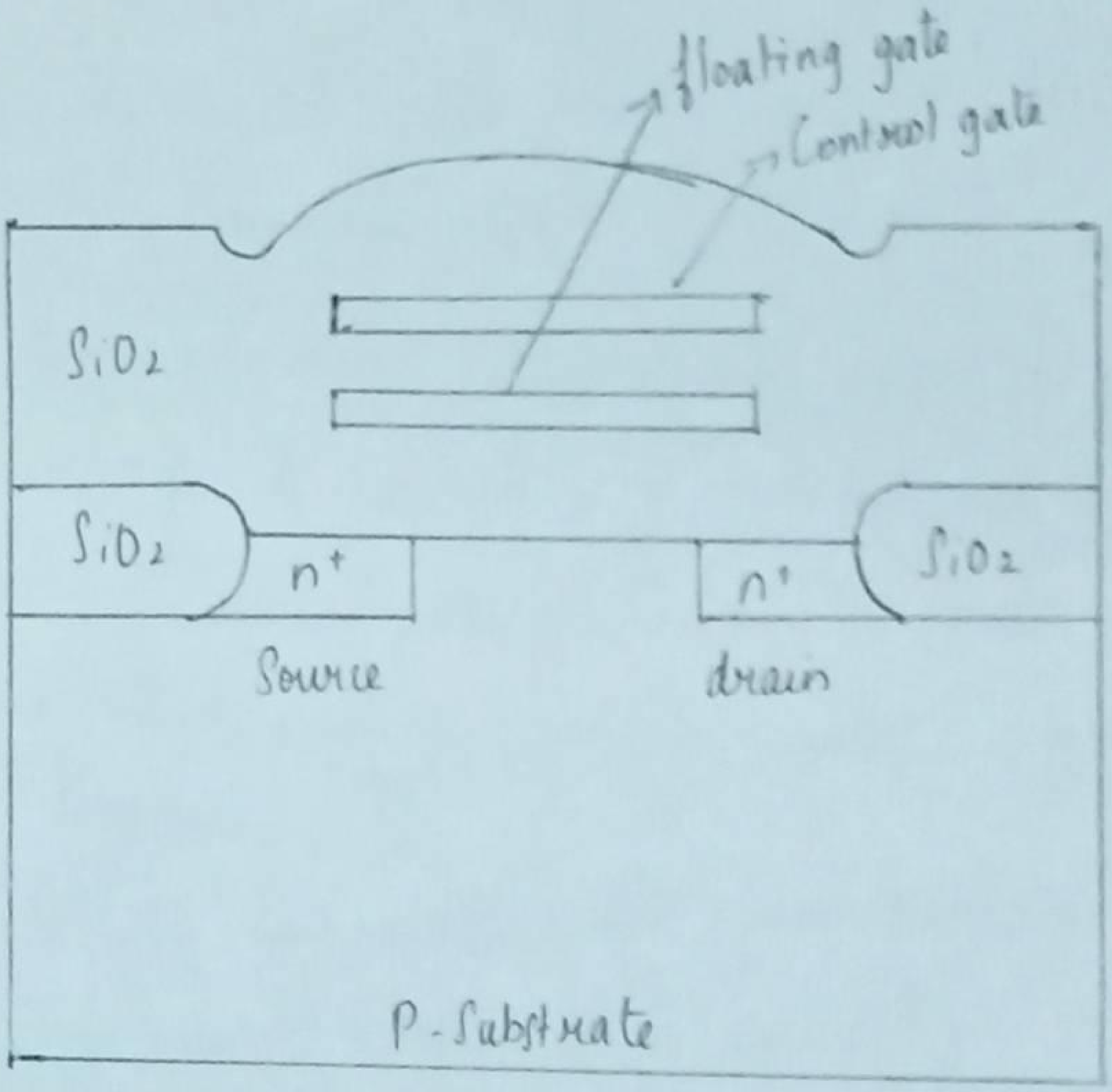
Disadvantages of EPROM:

* Changes in the selected memory locations cannot be made in reprogramming.

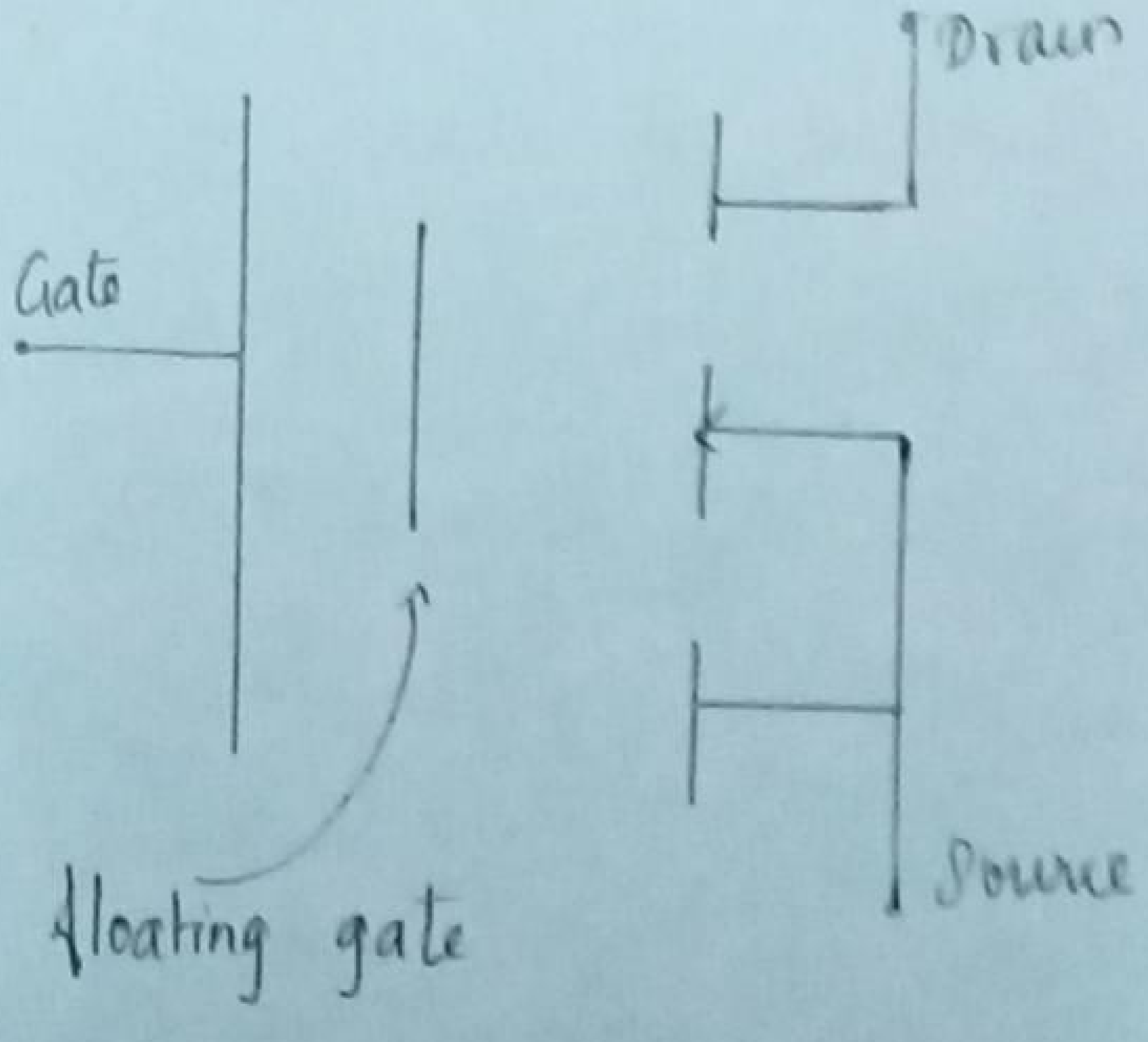
* The process of reprogramming cannot takes place with the IC in the circuit. This process takes about half an hour

EPROM

Structure



Symbol.



Electrically Erasable Programmable ROM (EEPROM)

* Another type of reprogrammable ROM device is EEPROM, which is also known as Electrically Alterable Programmable ROM.

* The EEPROM overcomes the disadvantages of EPROM.

* EEPROM can be erased and programmed by the application of controlled electric pulses to the IC in the circuit & thereby changes can be made in selected memory locations

* EEPROM is non-volatile like EPROM but does not require ultraviolet light to be erased

* EEPROM is a rugged, low power semiconductor device and it occupies less space

* It has the advantages of program flexibility, small size and semiconductor memory ruggedness

* The requirement of low power supports fields programming in portable devices for communication encoding, data formatting and conversion and program storage.

* With, EEPROM the programs can be altered remotely, possibly by telephone

Electronically Alterable Programmable ROM (EAP ROM)

* EAPROM stands for Electronically Alterable Programmable Read-only Memory.

* It is a type of PROM whose contents can be changed.

* It acts as a non-volatile storage device and its individual bits can be re-programmed during the course of system operation

* Computer acronyms. Memory, Memory terms, Storage device.

* The main difference between EEPROM and EAPROM is the content of EEPROM is erased by electric signals. the EAPROM is erased by electronically.

* A form of PROM in which the contents of selected memory locations can be changed by applying suitable electric signals, as in case of EAROM