Shakuntala Krishna Institute Of Technology

Subject – Digital Electronics & Computer Organization

Top – 15 Questions

1. Explain different number systems (Binary, Octal, Decimal, Hexadecimal). Write methods of conversion from one system to another.
2. Perform binary arithmetic operations (addition, subtraction using 1's and 2's complement).
3. State and prove De-Morgan's Theorems with truth tables.
4. What is a Karnaugh Map (K-Map)? Explain how Boolean expressions are minimized using K-Map (2-variable, 3-variable, 4-variable examples).
5. Explain the working of Half Adder and Full Adder with truth tables and logic diagrams.
6. Explain Half Subtractor and Full Subtractor with truth tables and logic diagrams.
7. Explain multiplexer and demultiplexer with truth tables and logic diagrams.
8. Explain decoder and encoder with examples.
9. What is a Flip-Flop? Explain SR, JK, D, T, and Master-Slave Flip-Flops with diagrams and truth tables.
10. Write the characteristic equations and characteristic tables of SR, JK, D, and T flip-flops and Explain conversion of flip-flops (SR to JK, JK to D, D to T, etc.) with proper methods.
11. Explain the difference between edge triggering and level triggering with diagrams.
12. What is a register? Explain classification of registers and working of a shift register with diagrams.
13. What is a counter? Differentiate between asynchronous (ripple) and synchronous counters with examples.
14. Explain the basic cell organization of static RAM (SRAM) and dynamic RAM (DRAM) with diagrams.
15. What is cache memory? Explain cache memory organization and virtual memory organization with examples.

1. **Explain different number systems (Binary, Octal, Decimal, Hexadecimal). Write methods of conversion from one system to another.**

Number System

A number is a mathematical object used to count, measure, and label. Numbers are represented by a string of digital symbols. A number system of base r is a system that uses distinct symbols for r digits. That is in a positional base$r$ numeral system r basic symbols (or digits) corresponding to the first r natural numbers including zero are used. To generate the rest of the numerals, the position of the symbol in the figure is used. The symbol in the last position has its own value, and as it moves to the left its value is multiplied by r. There are four systems of arithmetic used in digital system. These systems are Decimal, Binary, Hexadecimal and Octal.

System	Base	Digits
Binary	2	0 1
Octal	8	0 1 2 3 4 5 6 7
Decimal	10	0 1 2 3 4 5 6 7 8 9
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 A B C D E F

Decimal Number System: The Decimal number system has a base ten. This system uses ten distinct digits 0 1 2 3 4 5 6 7 8 9 to form any number. Each digit can be used individually or they can be grouped to form a numeric value.Each of decimal digits, 0 through 9, has a place value or weight depending on its position. The weights are units, tens, hundreds, thousands and so on. The same can be expressed as the powers of its base as 100 , 101 , 102 , 103 ⋯ $etc$for the integer partand 10−1 , 10−2 , 10−3 , 10−4 ⋯ etc for the fractional part. 100 , 101 , 102 , 103 ⋯ etcrepresents the units, tens, hundreds, thousands etc. and the quantities 10−1 , 10−2 , 10−3 , ⋯ etc represents one tenth, one hundredth, one thousandth etc. The integer part and fractional parts are separated by a decimal point. The position weights in decimal system is given as

⋯	10^3	10^2	10^1	10^0	·	10^{-1}	10^{-2}	10^{-3}	10^{-4}	⋯

Example:

(i) $7693 = 7 \times 10^3 + 6 \times 10^2 + 9 \times 10^1 + 3 \times 10^0$

$= 7 \times 1000 + 6 \times 100 + 9 \times 10 + 3 \times 1$

$= 7000 + 600 + 90 + 3$

(ii) $1936.46 = 1 \times 10^3 + 9 \times 10^2 + 3 \times 10^1 + 6 \times 10^0 + 4 \times 10^{-1} + 6 \times 10^{-2}$

$= 1000 + 900 + 30 + 6 + 0.4 + 0.06$

Binary Number System: The base of the binary number system is two. It uses the digits0 and 1 only. The two digits 0 and 1 are called a bit. The place value of each position can be expressed in terms of powers of 2 like 2 0 , 2 1 , 2 2 ,*etc* for integer part and 2 −1 , 2 −2 , 2 −3 ,*etc* for the fractional part. A binary point separates the integer and fractional part. The position weights in the binary is given as

...	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}	2^{-4}	...

Example : 10112 = (1×23) + (0×22) + (1×21) + (1×20) = 8 + 0 + 2 + 1 = 1110

4 $bit\ binary\ word \Longrightarrow nibble$

8 $bit\ binary\ word \Longrightarrow byte$

16 $bit\ binary\ word \Longrightarrow word$

32 $bit\ binary\ word \Longrightarrow double\ word$

Octal Number System: The base of the octal number system is eight. It uses eight digits 0 1 2 3 4 5 6 *and* 7 to form a number. The place value of each position can be expressed in terms of powers of 8 like 8 0 , 8 1 , 8 2 ,*etc* for integer part and 8 −1 , 8 −2 , 8 −3 ,*etc* for the fractional part. An octal point separates the integer and fractional part. Sets of 3-bit binary numbers can be represented by octal numbers (000, 001, 010,011, 100, 101,110,111) and this can be conveniently be used for entering data in the computer. The position weights in the octal system is given as

...	8^3	8^2	8^1	8^0	.	8^{-1}	8^{-2}	8^{-3}	8^{-4}	...

Example : 7458 = (7×82) + (4×81) + (5×80) = 448 + 32 + 5 = 48510

Hexadecimal Number System: The Hexadecimal number system has a base of 16. It has 16 distinct digit symbols. It uses the digits 0 1 2 3 4 5 6 7 8 9 plus the letters $A\ B\ C\ D\ E\ and\ F$. Any hexadecimal digit can be represented by a group of four bit binary sequence.That is the Hexadecimal numbersare represented by sets of 4-bit binary sequence (0000, 0001,0010, 0011, 0100,0101,0110, 0111,1000,1001,1010,1011,1100,1101,1110,1111). The position weight in the hexadecimal number system is given as

...	16^3	16^2	16^1	16^0	.	16^{-1}	16^{-2}	16^{-3}	16^{-4}	...

Number System

Decimal (Base 10)	Binary (Base 2)	Octal (Base 8)	Hexadecimal (Base 16)
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Example : 2F16 = (2×161) + (15×160) = 32 + 15 = 4710

2. Perform binary arithmetic operations (addition, subtraction using 1's and 2's complement).

Arithmetic operations such as addition, subtraction, multiplication and division can be performed on binary numbers.

Binary addition: The addition of two Binary numbers is very similar to addition of two decimal numbers. It is key to binary subtraction, multiplication and division.The following rules are followed while adding two binary numbers.

Augend (A)	+	Addend (B)	Carry (A)	Sum (B)	Result
0	+	0	0	0	0
0	+	1	0	1	1
1	+	0	01		1
1	+	1	1	0	10 ; read as 0 with a carry 1
1	+1	+1	1	1	11 ; read as 1 with a carry 1

Example:

1. Add the binary numbers (i) 1011 and 1110 (ii) 10.001 and 11.110

(i)

Binary Number		Equivalent Decimal
1 1 1	Carry	
1 0 1 1		11
+ 1 1 1 0		14
Sum= 1 1 0 0 1		25

(ii)

Binary Number		Equivalent Decimal
1	← Carry	
1 0 . 0 0 1		2.125
+ 1 1 . 1 1 0		+ 3.750
Sum= 1 0 1 . 1 1 1		5.875

Binary subtraction: The subtraction of two Binary numbers is very similar to subtraction of two decimal numbers. Subtraction is the inverse operation of addition. The following rules are used in subtracting two binary numbers.

Minuend	-	Subtrahend	Difference	Borrow	
0	-	0	0	0	
1	-	0	1	0	
1	-	1	0	0	
0	-	1	1	1	read as difference 1 with borrow 1

Example:

1. Subtract the binary numbers (i) 101 from 1001 (ii) 11and 10000

(i)

	Binary Number	Equivalent Decimal
	1001	9
-	101	-5
Difference =	100	4

(ii)

	Binary Number	Equivalent Decimal
	10000	16
-	11	-3
Difference =	1101	13

Subtraction using 1's Complement

Example: 7–5

- Minuend = 0111 (7 in decimal)
- Subtrahend = 0101 (5 in decimal)

Step 1: Take 1's complement of subtrahend

0101 ⇒ 1010

Step 2: Add minuend and complemented subtrahend

Minuend	**+**	**1's Complement of Subtrahend**	**=**	**Sum**
0111	+	1010	=	10001

Step 3: End-around carry (1) → add back

Sum (without carry)	+	Carry	=	Final Result
0001 (from 10001)	+	1000	=	0010

Answer = 0010 = +2

Subtraction using 2's Complement

Example: 7−5

Minuend = 0111

- Subtrahend = 0101

Step 1: Find 2's complement of subtrahend

0101 ⇒ 1010+1=1011

Step 2: Add

Minuend	+	2's Complement of Subtrahend	=	Sum
0111	+	1011	=	10010

Step 3: Discard carry → 0010

Answer = +2

3. State and prove De-Morgan's Theorems with truth tables.

DeMorgan's Theorems are basically two sets of rules or laws developed from the Boolean expressions for AND, OR and NOT using two input variables, A and B. These two rules or theorems allow the input variables to be negated and converted from one form of a Boolean function into an opposite form.

DeMorgan's first theorem states that two (or more) variables NOR´ed together is the same as the two variables inverted (Complement) and AND´ed, while the second theorem states that two (or

more) variables NAND´ed together is the same as the two terms inverted (Complement) and OR´ed. That is replace all the OR operators with AND operators, or all the AND operators with an OR operators.

DeMorgan's First Theorem

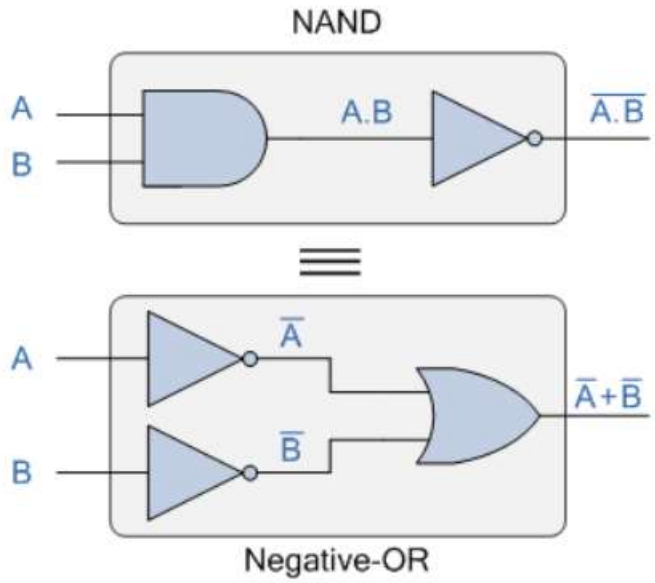
DeMorgan's First theorem proves that when two (or more) input variables are AND'ed and negated, they are equivalent to the OR of the complements of the individual variables. Thus the equivalent of the NAND function will be a negative-OR function, proving that A.B = A+B. We can show this operation using the following table.

Verifying DeMorgan's First Theorem using Truth Table

Inputs		Truth Table Outputs For Each Term				
B	A	A.B	$\overline{A.B}$	$\overline{A}$	$\overline{B}$	$\overline{A}+\overline{B}$
0	0	0	**1**	1	1	**1**
0	1	0	**1**	0	1	**1**
1	0	0	**1**	1	0	**1**
1	1	1	**0**	0	0	**0**

We can also show that A.B = A+B using logic gates as shown.

DeMorgan's First Law Implementation using Logic Gates



The top logic gate arrangement of: A.B can be implemented using a standard NAND gate with inputs A and B. The lower logic gate arrangement first inverts the two inputs producing A and B. These then become the inputs to the OR gate. Therefore the output from the OR gate becomes: A+B

Then we can see here that a standard OR gate function with inverters (NOT gates) on each of its inputs is equivalent to a NAND gate function. So an individual NAND gate can be represented in this way as the equivalency of a NAND gate is a negative-OR.

DeMorgan's Second Theorem

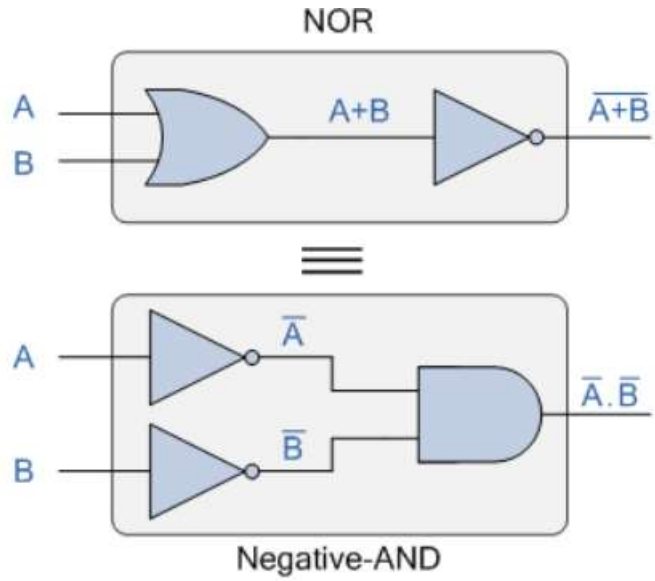
DeMorgan's Second theorem proves that when two (or more) input variables are OR'ed and negated, they are equivalent to the AND of the complements of the individual variables. Thus the equivalent of the NOR function is a negative-AND function proving that A+B = A.B, and again we can show operation this using the following truth table.

Verifying DeMorgan's Second Theorem using Truth Table

Inputs		Truth Table Outputs For Each Term				
B	A	A+B	$\overline{A+B}$	$\overline{A}$	$\overline{B}$	$\overline{A}.\overline{B}$
0	0	0	**1**	1	1	**1**
0	1	1	**0**	0	1	**0**
1	0	1	**0**	1	0	**0**
1	1	1	**0**	0	0	**0**

We can also show that A+B = A.B using the following logic gates example.

DeMorgan's Second Law Implementation using Logic Gates



The top logic gate arrangement of: A+B can be implemented using a standard NOR gate function using inputs A and B. The lower logic gate arrangement first inverts the two inputs, thus producing A and B. Thus then become the inputs to the AND gate. Therefore the output from the AND gate becomes: A.B

Then we can see that a standard AND gate function with inverters (NOT gates) on each of its inputs produces an equivalent output condition to a standard NOR gate function, and an individual NOR gate can be represented in this way as the equivalency of a NOR gate is a negative-AND.

Although we have used DeMorgan's theorems with only two input variables A and B, they are equally valid for use with three, four or more input variable expressions, for example:

For a 3-variable input

$A.B.C = A+B+C$

and also

$A+B+C = A.B.C$

For a 4-variable input

$A.B.C.D = A+B+C+D$

and also

A+B+C+D = A.B.C.D

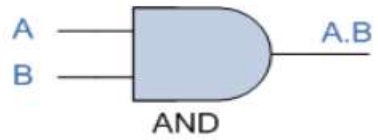
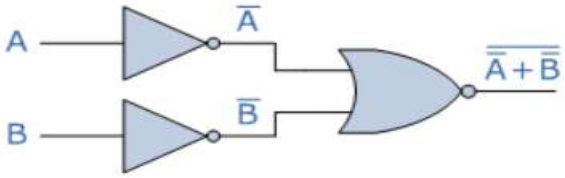
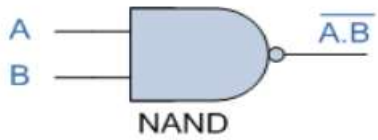
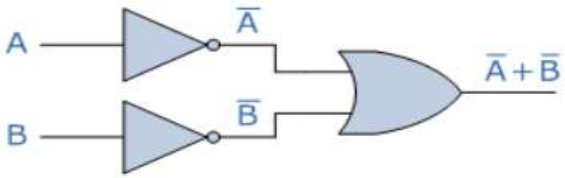
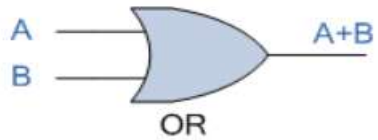
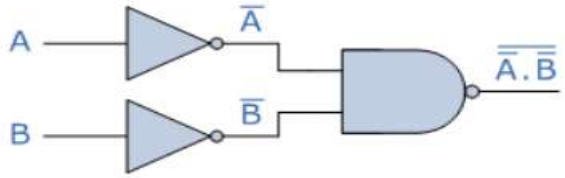
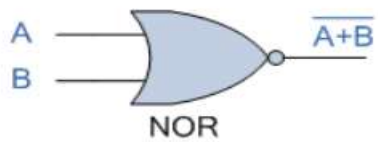
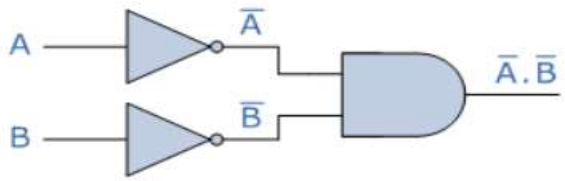
and so on.

DeMorgan's Equivalent Gates

We have seen here that by using DeMorgan's Theorems we can replace all of the AND (.) operators with an OR (+) and vice versa, and then complements each of the terms or variables in the expression by inverting it, that is 0's to 1's and 1's to 0's before inverting the entire function.

Thus to obtain the DeMorgan equivalent for an AND, NAND, OR or NOR gate, we simply add inverters (NOT-gates) to all inputs and outputs and change an AND symbol to an OR symbol or change an OR symbol to an AND symbol as shown in the following table.

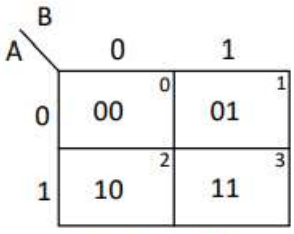
DeMorgan's Equivalent Gates

Standard Logic Gate	DeMorgan's Equivalent Gate
	
	
	
	

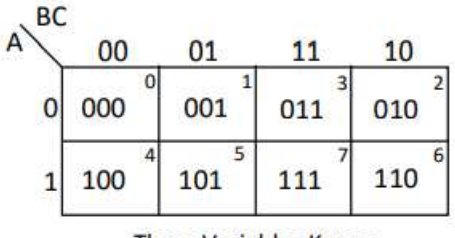
Then we have seen in this tutorial about DeMorgan's Thereom that the complement of two (or more) AND'ed input variables is equivalent to the OR of the complements of these variables, and that the complement of two (or more) OR'ed variables is equivalent to the AND of the complements of the variables as defined by *DeMorgan*.

4. What is a Karnaugh Map (K-Map)? Explain how Boolean expressions are minimized using K-Map (2-variable, 3-variable, 4-variable examples).

The Boolean theorems and the De-Morgan's theorems are useful in manipulating the logic expression. We can realize the logical expression using gates. The number of logic gates required for the realization of a logical expression should be reduced to a minimum possible value. One of the methods used to minimize the logical expression is K-map method. A Karnaugh map provides a pictorial method of grouping together expressions with common factors and therefore eliminating unwanted variables. The Karnaugh map can also be described as a special arrangement of a truth table. The K-map is a graphical device used to simplify a logical equation or to convert a truth table to its corresponding logic circuit in a simple, logical method. It is also known as Veitch diagram. A K-map is a diagram made up of squares and may be considered to be the graphic representation of the minterm canonical form. Each minterm is represented by a cell, and the cells are assembled in an orderly arrangement such that adjacent cell represent minterms which differ by one variable. The number of cells in a K-map depends upon the number of variables in the Boolean expression. Two variables map contain four cells, three variables map contain eight cells and n variables map contain 2n cells. Each row and column of the map is assigned by 0's and 1's as shown in figure.



Two Variables K-map
With cell number



Three Variables K-map
with cell number

This method can be done in two different ways, as discussed below.

Sum of Products (SOP) Form It is in the form of sum of three terms AB, AC, BC with each individual term is a product of two variables. Say A.B or A.C or B.C. Therefore such expressions are known as expression in SOP form. The sum and products in SOP form are not the actual additions or multiplications. In fact they are the OR and AND functions. In SOP form, 0 represents a bar and 1 represents an unbar. SOP form is represented by ∑.

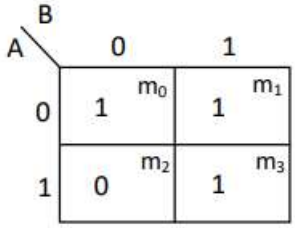
Boolean expression in SOP may or may not be in a standard form. First the expression is converted into SOP and then, 1's are marked in each cell corresponding to the minterm of expression and the remaining cells are marked with 0's.

Examples of SOP: 1. K-map for the Boolean expression $Y\ (A, B, C) = \bar{A}$ + B

In SOP form $\bar{A}\bar{B} + \bar{A}B + AB$

$\downarrow\downarrow \quad \downarrow\downarrow \quad \downarrow\downarrow$

$0\,0 \quad 0\,1 \quad 1\,1$



$\sum m(0,1,3)$

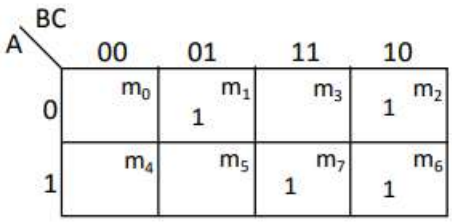
Result of $\bar{A}\bar{B} + \bar{A}B + AB$ is $\bar{A} + B$

2. K-map for the Boolean expression $Y(A, B, C) = AB + B\bar{C} + \bar{A}\bar{B}C$

In SOP form $ABC + AB\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C$

$\downarrow\downarrow\downarrow \quad \downarrow\downarrow\downarrow \quad \downarrow\downarrow\downarrow \quad \downarrow\downarrow\downarrow$

$1\,11 \quad 11\,0 \quad 01\,0 \quad 0\,0\,1$



$\sum m(1,2,6,7)$

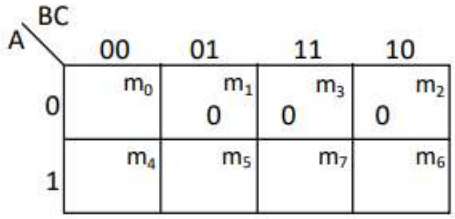
Result of $ABC + AB\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C$ is $AB + B\bar{C} + \bar{A}\bar{B}C$

Product of Sums (POS) Form It is in the form of product of three terms (A+B), (B+C), or (A+C) with each term is in the form of a sum of two variables. Such expressions are said to be in the product of sums (POS) form. In POS form, 0 represents an unbar and 1 represents a bar. POS form is represented by ∏

Example of POS:

In POS form $(B + \bar{C})(\bar{A} + \bar{B})(\bar{B} + C)$

$$\begin{array}{cccccc} \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0 & 1 & 1 & 1 & 1 & 0 \end{array}$$



$\prod m(1,3,2)$

Result of $(B + \bar{C})(\bar{A} + \bar{B})(\bar{B} + C)$ is $(A + \bar{C})(A + \bar{B})$

Steps for Minimization using K-Map

1. Draw the K-map grid (2, 3, or 4 variables).
2. Fill the cells with output values (1 for minterms, 0 otherwise).
3. Group adjacent 1's in **powers of 2** (1, 2, 4, 8...).
    - Groups may wrap around edges.
    - Larger groups → more simplification.
4. Write simplified expression from groups.

1. 2-Variable K-Map

Format:

A \ B	**0**	**1**
0	F(0)	F(1)
1	F(2)	F(3)

Example:

Given function F(A,B)=Σm(1,3)

K-map:

A\B	0	1
0	0	1
1	0	1

Group the 1’s in column (B=1).

Simplified Expression:

F=B

2. 3-Variable K-Map

Format (Gray Code):

A\BC	00	01	11	10
0				
1				

Example:

Given F(A,B,C)=Σm(1,2,3,5,7)

Fill K-map:

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

👉 Grouping:

- (m1,m3,m5,m7) → B
- (m2,m3) → A’C

Simplified Expression:

F=B+A’C

---

3. 4-Variable K-Map

Format (Gray Code order for rows & columns):

AB \ CD	00	01	11	10
00				
01				
11				
10				

Example:

Given F(A,B,C,D)=Σm(0,2,5,7,8,10,13,15)

Fill K-map:

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

Grouping:

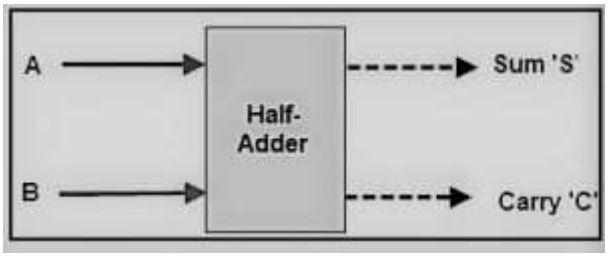
- (m0,m2,m8,m10) → A’C’
- (m5,m7,m13,m15) → AC
- (m10,m15) → BD

Simplified Expression:

F=A’C’+AC+BDF = A'C' + AC + BDF=A’C’+AC+BD

5. Explain the working of Half Adder and Full Adder with truth tables and logic diagrams.

Half adder adds two binary digits where the input bits are termed as augend and addend and the result will be two outputs one is the sum and the other is carry. To perform the sum operation, XOR is applied to both the inputs, and AND gate is applied to both inputs to produce carry.



HA Functional Diagram

Whereas in the full adder circuit, it adds 3 one-bit numbers, where two of the three bits can be referred to as operands and the other is termed as bit carried in. The produced output is 2-bit output and these can be referred to as output carry and sum.

By using a half adder, you can design simple addition with the help of logic gates.

Let's see an example of adding two single bits.

The 2-bit **half adder truth table** is as below:

INPUTS		OUTPUTS	
A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half Adder Truth Table

0+0 = 0

0+1 = 1

1+0 = 1

1+1 = 10

These are the least possible single-bit combinations. But the result for 1+1 is 10, the sum result must be re-written as a 2-bit output. Thus, the equations can be written as

0+0 = 00

0+1 = 01

1+0 = 01

1+1 = 10

The output ‘1’of ‘10’ is carry-out. ‘SUM’ is the normal output and ‘CARRY’ is the carry-out.

Now it has been cleared that a 1-bit adder can be easily implemented with the help of the XOR Gate for the output ‘SUM’ and an AND Gate for the ‘Carry’.

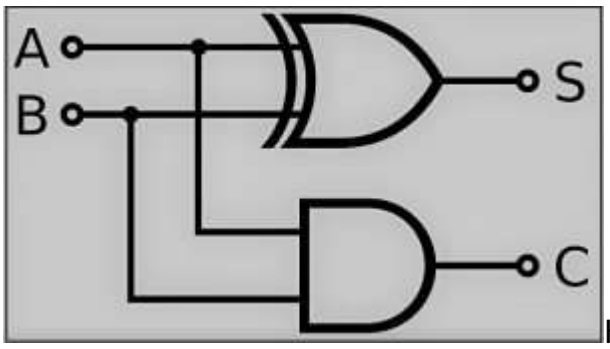
For instance, when we need to add, two 8-bit bytes together, then it can be implemented by using a full-adder logic circuit. The half-adder is useful when you want to add one binary digit quantities.

A way to develop two-binary digit adders would be to make a truth table and reduce it. When you want to make a three binary digit adder, the half adder addition operation is performed twice. In a similar way, when you decide to make a four-digit adder, the operation is performed one more time. With this theory, it was clear that the implementation is simple, but development is a time taking process.

The simplest expression uses the exclusive OR function:

Sum= A XOR B

Carry = A AND B



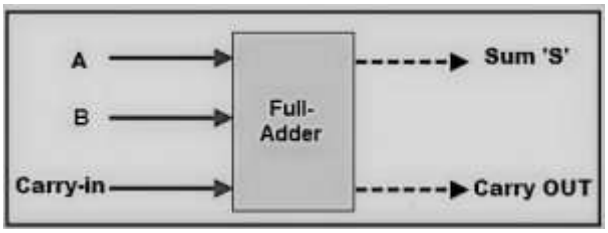
HA Logical Diagram

And an equivalent expression in terms of the basic AND, OR, and NOT is:

SUM=A.B+A.B’

Full Adder

This adder is difficult to implement when compared to half-adder.



Full Adder Functional Diagram

The difference between a half-adder and a full-adder is that the full-adder has three inputs and two outputs, whereas half adder has only two inputs and two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. When a full-adder logic is designed, you string eight of them together to create a byte-wide adder and cascade the carry bit from one adder to the next.

INPUTS			OUTPUT	
A	B	C-IN	C-OUT	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

Full Adder Truth Table

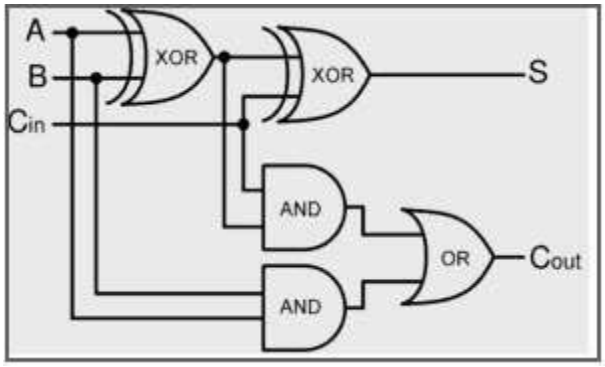
The output carry is designated as C-OUT and the normal output is represented as S which is ‘SUM’.

With the above **full adder truth-table**, the implementation of a full adder circuit can be understood easily. The SUM ‘S’ is produced in two steps:

1. By XORing the provided inputs ‘A’ and ‘B’
2. The result of A XOR B is then XORed with the C-IN

This generates SUM and C-OUT is true only when either two of three inputs are HIGH, then the C-OUT will be HIGH. So, we can implement a full adder circuit with the help of two half adder circuits. Initially, the half adder will be used to add A and B to produce a partial Sum and a second-half adder logic can be used to add C-IN to the Sum produced by the first half adder to get the final S output.

If any of the half adder logic produces a carry, there will be an output carry. So, C-OUT will be an OR function of the half-adder Carry outputs. Take a look at the implementation of the full adder circuit shown below.



Full Adder Logical Diagram

The implementation of larger logic diagrams is possible with the above full adder logic a simpler symbol is mostly used to represent the operation. Given below is a simpler schematic representation of a one-bit full adder.

With this type of symbol, we can add two bits together, taking a carry from the next lower order of magnitude, and sending a carry to the next higher order of magnitude. In a computer, for a multi-bit operation, each bit must be represented by a full adder and must be added simultaneously. Thus, to add two 8-bit numbers, you will need 8 full adders which can be formed by cascading two of the 4-bit blocks.

6. Explain Half Subtractor and Full Subtractor with truth tables and logic diagrams.

A **Half Subtractor** is a combinational circuit that subtracts one binary digit (bit) from another, producing a difference and a borrow-out. It takes two inputs: the minuend (A) and the subtrahend (B), and generates two outputs: the **Difference** (D) and the **Borrow-out** (Bo).

Operation

The Half Subtractor performs the operation A - B. The outputs are:

- **Difference (D)**: The result of the subtraction.
- **Borrow-out (Bo)**: Indicates if a borrow is needed from the next higher bit.

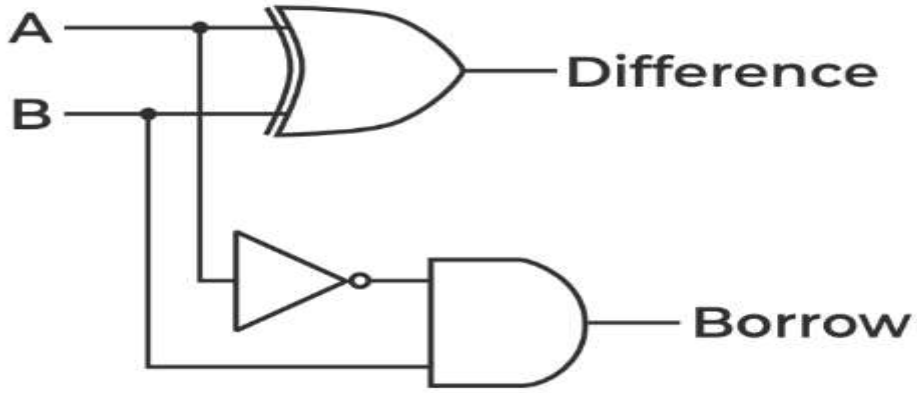
Truth Table

The truth table for a Half Subtractor is as follows:

A	B	Difference (D)	Borrow-out (Bo)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Explanation:

- **Difference (D)**: D = A ⊕ B (XOR of A and B).
- **Borrow-out (Bo)**: Bo = ¬A ∧ B (NOT A AND B).



A **Full Subtractor** is a combinational circuit that subtracts one binary digit from another while also accounting for a borrow-in from the previous stage. It takes three inputs: the minuend (A), the subtrahend (B), and the borrow-in (Bin), and produces two outputs: the **Difference** (D) and the **Borrow-out** (Bo).

Operation

The Full Subtractor performs the operation A - B - Bin. The outputs are:

- **Difference (D)**: The result of the subtraction.
- **Borrow-out (Bo)**: Indicates if a borrow is needed for the next higher bit.

Truth Table

The truth table for a Full Subtractor is as follows:

A	B	Bin	Difference (D)	Borrow-out (Bo)
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

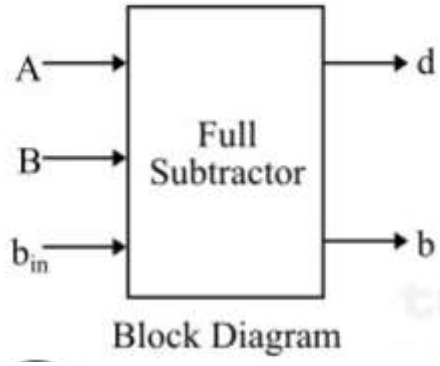
1	1	1	1	1

Explanation:

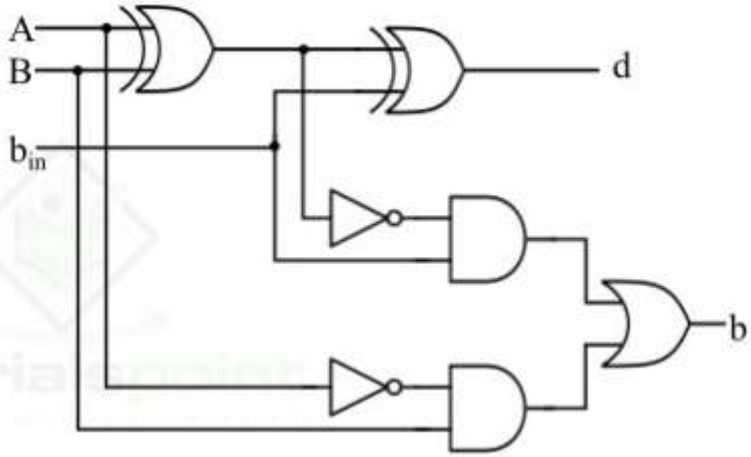
- **Difference (D)**: D = A ⊕ B ⊕ Bin (XOR of A, B, and Bin).
- **Borrow-out (Bo)**: Bo = ¬A ∧ B ∨ ¬A ∧ Bin ∨ B ∧ Bin.

Logic Expressions

- **Difference**: D = A ⊕ B ⊕ Bin
- **Borrow-out**: Bo = ¬A ∧ B ∨ ¬A ∧ Bin ∨ B ∧ Bin



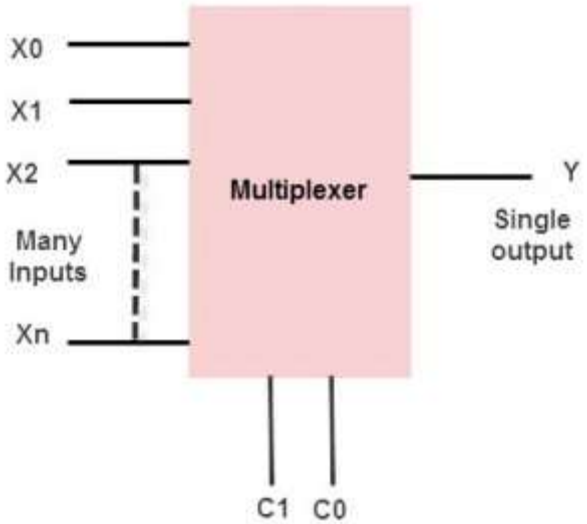
Block Diagram



7. Explain multiplexer and demultiplexer with truth tables and logic diagrams.

The multiplexer is a device that has multiple inputs and single line output. The select lines determine which input is connected to the output, and also increase the amount of data that can be sent over a network within a certain time. It is also called a data selector.

The single-pole multi-position switch is a simple example of a non-electronic circuit of the multiplexer, and it is widely used in many electronic circuits. The multiplexer is used to perform high-speed switching and is constructed by electronic components.



Multiplexer

Multiplexers are capable of handling both analog and digital applications. In analog applications, multiplexers are made up of relays and transistor switches, whereas in digital applications, the multiplexers are built from standard logic gates. When the multiplexer is used for digital applications, it is called a digital multiplexer.

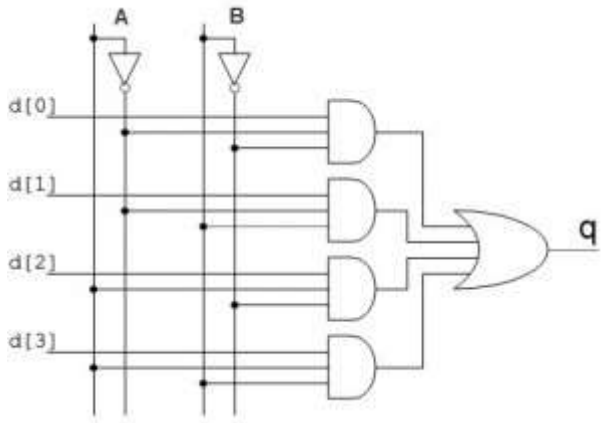
Multiplexer Types

Multiplexers are classified into four types:

- 2-1 multiplexer (1select line)
- 4-1 multiplexer (2 select lines)
- 8-1 multiplexer(3 select lines)
- 16-1 multiplexer (4 select lines)

4-to-1 Multiplexer

The 4X1 multiplexer comprises 4-input bits, 1- output bit, and 2- control bits. The four input bits are namely 0, D1, D2, and D3, respectively; only one of the input bits is transmitted to the output. The o/p 'q' depends on the value of control input AB. The control bit AB decides which of the i/p data bit should transmit the output. The following figure shows the 4X1 multiplexer circuit diagram using AND gates. For example, when the control bits AB =00, then the higher AND gates are allowed while remaining AND gates are restricted. Thus, data input D0 is transmitted to the output 'q"

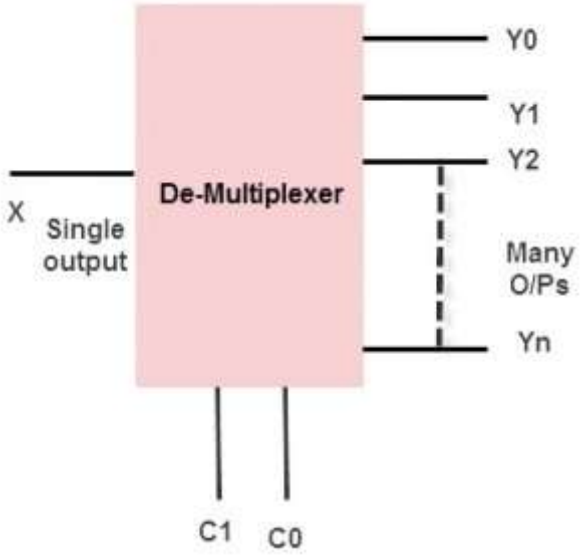


4X1 Mux

If the control input is changed to 11, then all gates are restricted except the bottom AND gate. In this case, D3 is transmitted to the output, and q=D0. If the control input is changed to AB =11, all gates are disabled except the bottom AND gate. In this case, D3 is transmitted to the output, and q = D3. The best example of a 4X1 multiplexer is IC 74153. In this IC, the o/p is the same as the i/p. Another example of a 4X1 multiplexer is IC 45352. In this IC, the o/p is the compliment of the i/p

De-multiplexer is also a device with one input and multiple output lines. It is used to send a signal to one of the many devices. The main difference between a multiplexer and a de-multiplexer is that

a multiplexer takes two or more signals and encodes them on a wire, whereas a de-multiplexer does reverse to what the multiplexer does.



Demultiplexer

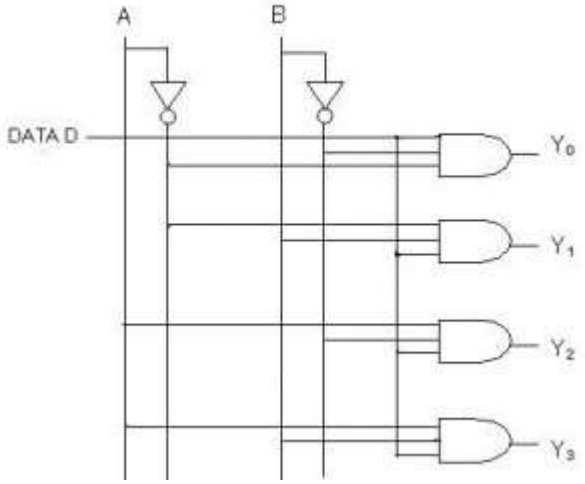
Types of Demultiplexer

Demultiplexers are classified into four types

- 1-2 demultiplexer (1 select line)
- 1-4 demultiplexer (2 select lines)
- 1-8 demultiplexer (3 select lines)
- 1-16 demultiplexer (4 select lines)

1-4 Demultiplexer

The 1-to-4 demultiplexer comprises 1- input bit, 4-output bits, and control bits. The 1X4 demultiplexer circuit diagram is shown below.



1X4 Demux

The i/p bit is considered as Data D. This data bit is transmitted to the data bit of the o/p lines, which depends on the AB value and the control i/p.

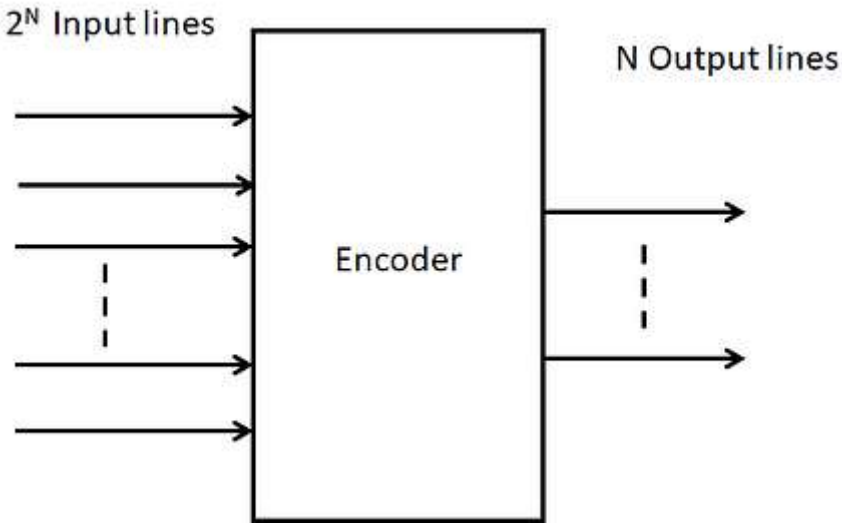
When the control i/p AB = 01, the upper second AND gate is permitted while the remaining AND gates are restricted. Thus, only data bit D is transmitted to the output, and Y1 = Data.

If the data bit D is low, the output Y1 is low. IF data bit D is high, the output Y1 is high. The value of the output Y1 depends upon the value of data bit D, the remaining outputs are in a low state.

If the control input changes to AB = 10, then all the gates are restricted except the third AND gate from the top. Then, data bit D is transmitted only to the output Y2; and, Y2 = Data. . The best example of 1X4 demultiplexer is IC 74155.

8. Explain decoder and encoder with examples.

An encoder in digital electronics is a combinational circuit that has 2 to the power n inputs and n outputs. The encoder produces a #binary code equivalent to the given input. The #encoder encodes information from 2^n inputs to n outputs.

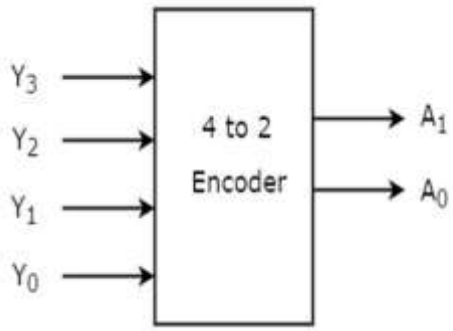


Block Diagram of Encoder

Types of Encoders:

4 to 2 line encoder:

4 can be written as 2^2 so the inputs are 4 and the outputs are 2. Let the outputs be A1 and A0 and the inputs be Y3, Y2, Y1, Y0. At any time any one of the inputs will be 1 and the respective Binary code will be the output.



Block Diagram of 4 to 2 Encoder

The following is the truth table of 4 to 2 encoder

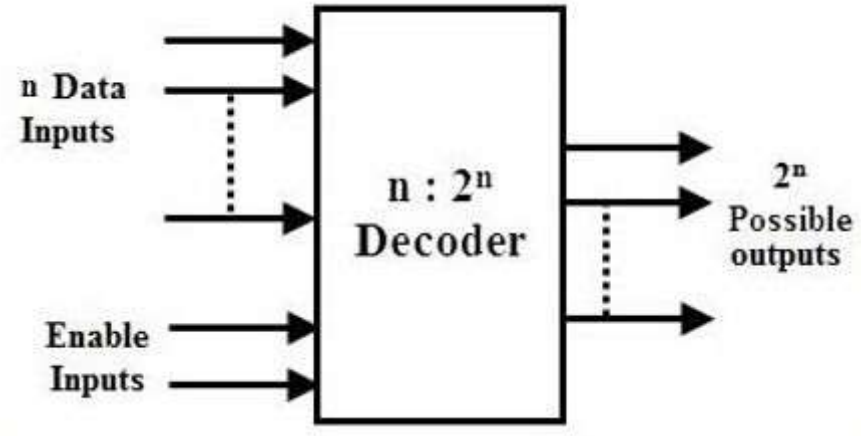
INPUTS				OUTPUTS	
Y3	Y2	Y1	Y0	A1	A2
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

The logical expressions for A1 and A0 are

A1 = Y3 + Y2

A0 = Y3 + Y1

A decoder in digital electronics is a combinational circuit that has n inputs and 2 to the power of n outputs. The output of the decoder is a maximum of 2^n unique output lines. The binary information from n input lines is converted to a maximum of 2^n unique output lines in the decoder. The operation of decoder is reverse to that of encoder.

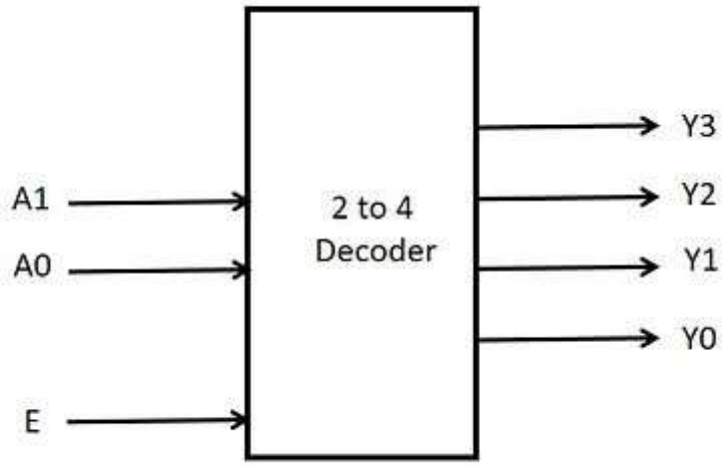


Block Diagram of Encoder

Types of Decoders:

2 to 4 decoder:

The 2 to 4 decoder has 2 input and 4 output lines. Let the inputs be A1 and A0 and the outputs be Y3, Y2, Y1, Y0. The following is the block diagram of 2 to 4 decoder



When the enable pin is set i.e., when E=1, for each input combination given at input lines one of the outputs will be high.

The following is the truth table of 2 to 4 decoder

Enable	Input		Output			
E	A1	A0	Y3	Y2	Y1	Y0
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

The Boolean expressions for each output can be written from the above truth table as follows

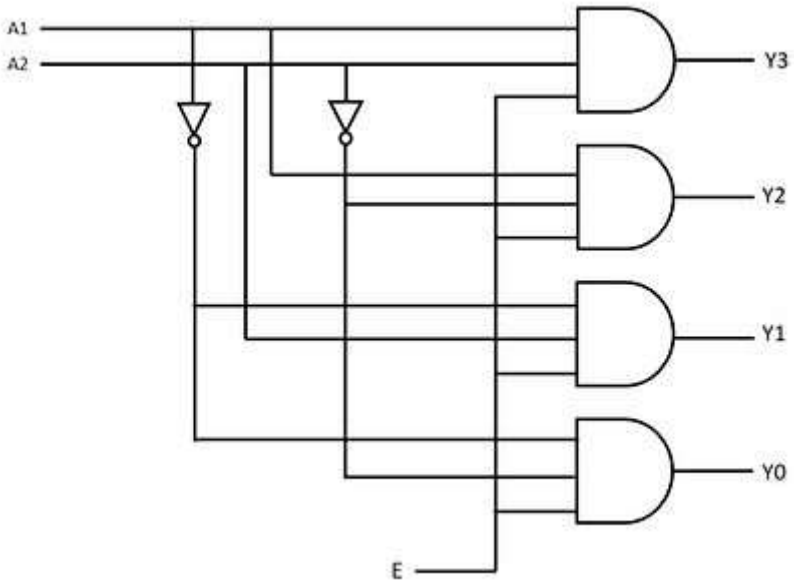
Y3 = E.A1.A0

Y2 = E.A1.(A0)'

Y1 = E.(A1)'.A0

Y0 = E.(A1)'.(A0)'

The above Boolean functions can be implemented using logic gates as follows



9. What is a Flip-Flop? Explain SR, JK, D, T, and Master-Slave Flip-Flops with diagrams and truth tables.

A flip-flop is a fundamental sequential logic circuit with two stable states that can store a single bit of binary data ($0 or\ 1$).It is controlled by input signals and a clock pulse and is used in memory and other digital systems. The main types are SR, JK, D, and T, along with the Master Slave variation, each with its own truth table and logic.

SR (Set-Reset) flip-flop

The SR flip-flop is one of the simplest sequential logic circuits, constructed from two cross-coupled NOR or NAND gates. It has two inputs, Set (S) and Reset (R), and two

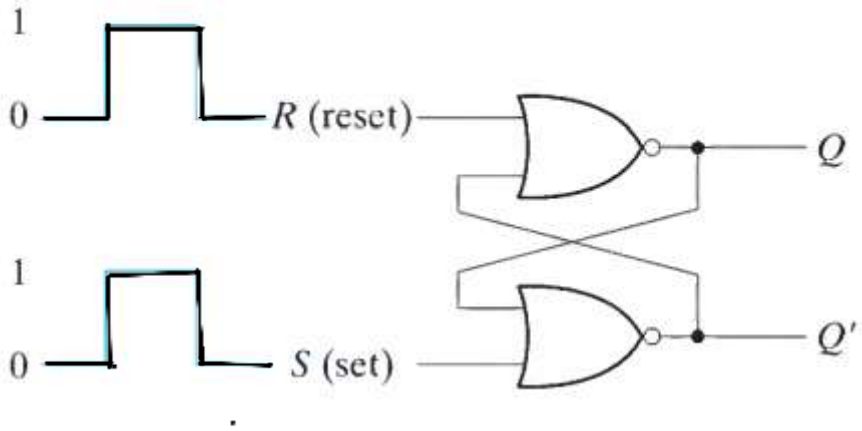
complementary

outputs, Q and Q'.

Logic Diagram using NOR gates

Truth Table

S	R	Q_{n+1}	Comments
0	0	Q_n	**Hold State:** Retains previous value.
0	1	0	**Reset State:** The Q output becomes 0.
1	0	1	**Set State:** The Q output becomes 1.
1	1	Invalid	**Invalid State:** Creates an unpredictable and unstable output, so this input combination must be avoided.



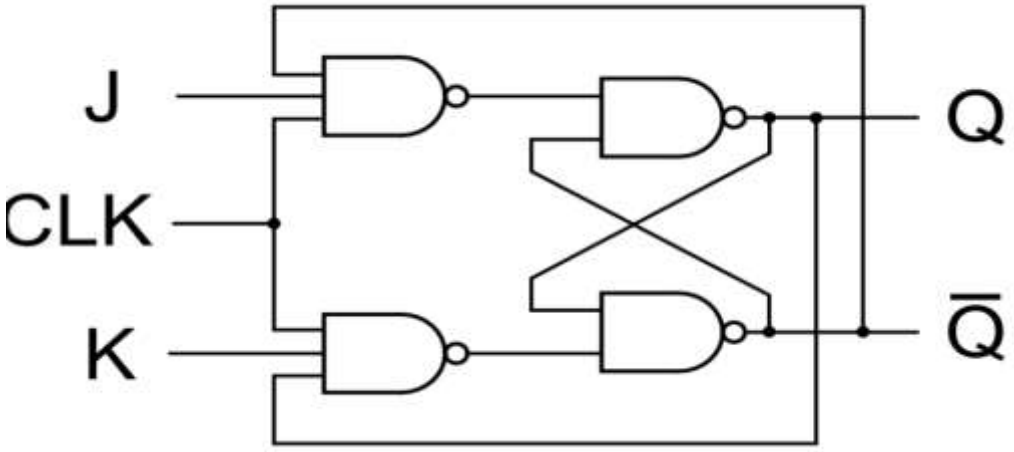
JK flip-flop

The JK flip-flop is a refined version of the SR flip-flop that overcomes the invalid output state when both inputs are high. When both J and K are 1, the output "toggles" or inverts its state on the next clock pulse.

Logic Diagram

Truth Table

J	K	$Qn+1$	Comments
0	0	Qn	**Hold State:** Retains previous value.
0	1	0	**Reset State:** The Q output becomes 0.
1	0	1	**Set State:** The Q output becomes 1.
1	1	$\overline{Q_n}$	**Toggle State:** The output switches to its complement.



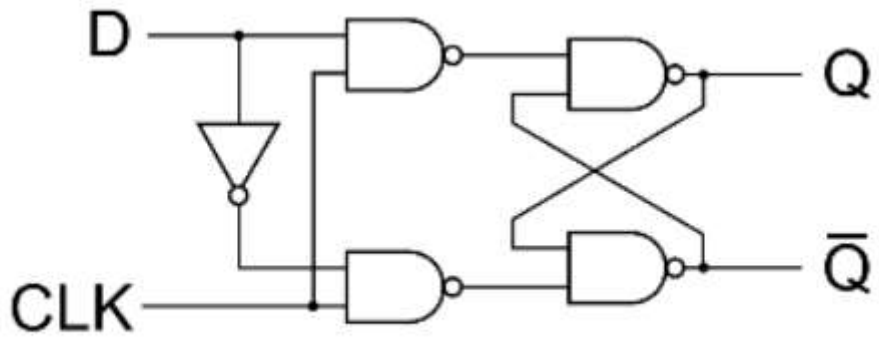
D (Delay) flip-flop

The D flip-flop, or Data flip-flop, stores the value of the single Data (D) input at a specific point in time and outputs it to Q. It is often used in shift registers and memory units.

Logic Diagram

Truth Table

D	Qn+1	Comments
0	0	**Reset State:** The output Q is 0 on the clock edge.
1	1	**Set State:** The output Q is 1 on the clock edge.



D Flip-Flop Circuit

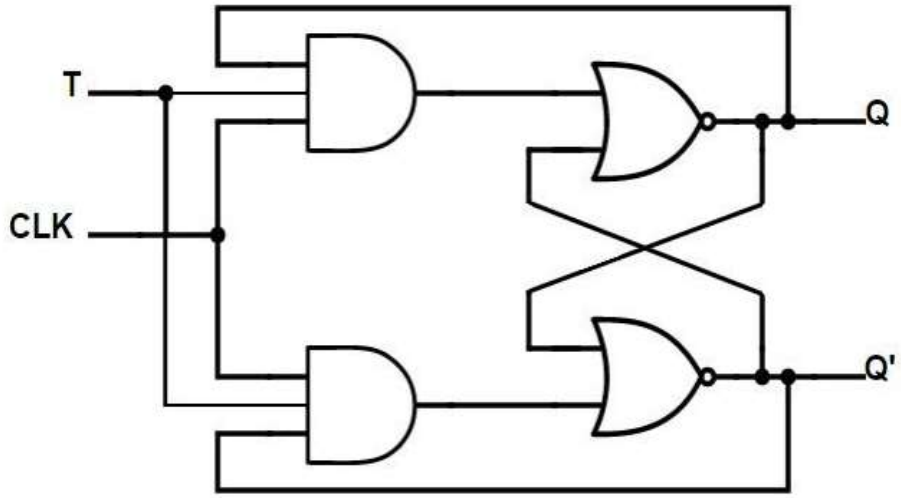
T (Toggle) flip-flop

The T flip-flop is a simplified version of the JK flip-flop created by connecting the J and K inputs together. It has a single T input, which controls whether the flip-flop holds its state or toggles it.

Logic Diagram

Truth Table

T	Qn+1	Comments
0	Qn	**Hold State:** Retains the previous value.
1	$\overline{Q_n}$	**Toggle State:** The output switches to its complement.



T Flip Flop Circuit

Master-Slave flip-flop

The master-slave configuration is a method of building a flip-flop by connecting two latches in a series. It is most commonly used for JK flip-flops to prevent the "race-around condition," where the output continuously toggles when J=K=1 and the clock pulse is high.

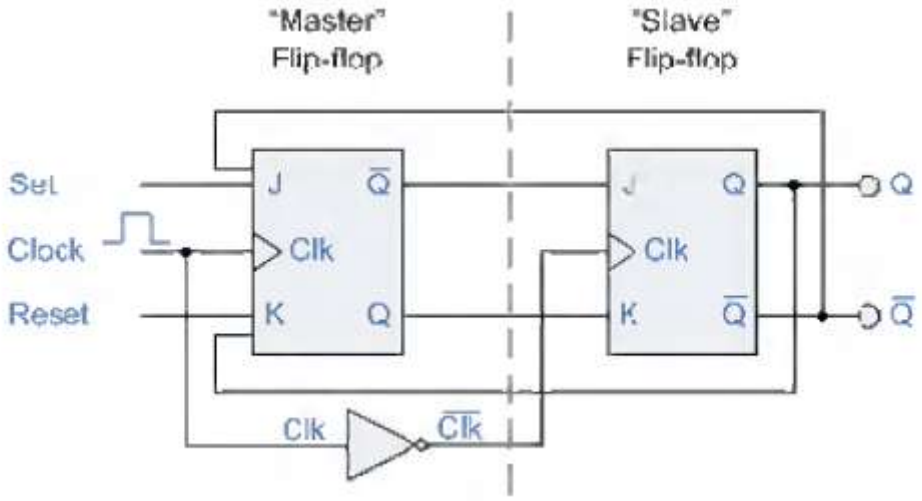
A master-slave flip-flop consists of two stages:

- **Master Stage:** A master flip-flop is enabled by the rising edge of the clock pulse and stores the input data.
- **Slave Stage:** A slave flip-flop is enabled by the falling edge of the clock pulse (an inverted signal) and transfers the data from the master to the final output.

Logic Diagram

Truth Table (for a Master-Slave JK flip-flop)

J	K	$Qn+1$	Comments
0	0	Q_n	**Hold State:** Retains previous value.
0	1	0	**Reset State:** The Q output becomes 0.
1	0	1	**Set State:** The Q output becomes 1.
1	1	$\overline{Q_n}$	**Toggle State:** The output switches to its complement.



10. Write the characteristic equations and characteristic tables of SR, JK, D, and T flip-flops and Explain conversion of flip-flops (SR to JK, JK to D, D to T, etc.) with proper methods.

Characteristic Equations and Tables of Flip-Flops

In digital electronics, flip-flops are fundamental sequential circuits used for storing binary data.

The **characteristic table** describes the next state (Q_{n+1}) based on the current state (Q_n) and inputs.

The **characteristic equation** mathematically expresses Q_{n+1} in terms of inputs and Q_n (where ' denotes NOT, + denotes OR, and juxtaposition or · denotes AND).

Below, I provide the characteristic tables and equations for SR, JK, D, and T flip-flops. These assume edge-triggered behavior with a clock signal, but the tables focus on the logical operation (ignoring clock for simplicity).

1. SR Flip-Flop

- **Inputs**: S (Set), R (Reset)
- **Behavior**: Sets output to 1 when S=1 and R=0; resets to 0 when S=0 and R=1; holds state when S=R=0; invalid (race condition or undefined) when S=R=1.
- **Characteristic Table**:

S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	Undefined
1	1	1	Undefined

- **Characteristic Equation**: Q_{n+1} = S + R' Q_n (valid only when S·R = 0; undefined otherwise).

2. JK Flip-Flop

- **Inputs**: J, K
- **Behavior**: Similar to SR, but toggles when J=K=1; no invalid state.
- **Characteristic Table**:

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1

1	1	0	1
1	1	1	0

- **Characteristic Equation**: Q_{n+1} = J Q_n' + K' Q_n

3. D Flip-Flop

- **Input**: D (Data)
- **Behavior**: The next state directly follows the D input; used for data storage.
- **Characteristic Table**:

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

- **Characteristic Equation**: Q_{n+1} = D

4. T Flip-Flop

- **Input**: T (Toggle)
- **Behavior**: Holds state when T=0; toggles when T=1.
- **Characteristic Table**:

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

- **Characteristic Equation**: Q_{n+1} = T $\oplus$ Q_n = T' Q_n + T Q_n'

Conversion of Flip-Flops

Flip-flop conversion involves modifying one type of flip-flop (source) to behave like another (target) by adding combinational logic (gates) to its inputs. This is useful when only certain flip-flops are available in hardware.

General Method for Conversion

1. **Identify Characteristic Equations**: Use the equations of both source and target flip-flops.
2. **Create a Conversion Table**: List all possible combinations of the target's inputs and current state Q_n. For each, compute the desired Q_{n+1} from the target's characteristic equation.
3. **Determine Source Inputs**: For each row, find the source flip-flop's input values that produce the same Q_{n+1} given Q_n.
4. **Minimize Logic**: Use Karnaugh maps (K-maps) or Boolean algebra to express the source inputs as functions of the target's inputs and Q_n.
5. **Implement Circuit**: Connect the logic gates to the source flip-flop's inputs. The clock and Q output remain the same.

   Below, I explain common conversions (SR to JK, JK to SR, JK to D, D to JK, JK to T, T to JK, D to T, T to D, SR to D, D to SR) with the derived logic expressions. For each, I'll provide the

conversion table and minimized expressions. (Diagrams would show gates connected to inputs; e.g., for JK to D, an inverter from D to K.)

1. SR to JK (Convert SR Flip-Flop to Act Like JK)

- Target: JK ($Q_{n+1} = J Q_n' + K' Q_n$)
- Source: SR ($Q_{n+1} = S + R' Q_n$)
- Conversion Table (Inputs: J, K, Q_n; Desired Q_{n+1}; Find S, R):

J	K	Q_n	Desired Q_{n+1}	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

- Minimized Expressions (using K-map):
  - $S = J Q_n'$
  - $R = K Q_n$
- Circuit: AND gate for S (J and Q_n'), AND gate for R (K and Q_n). Connect to SR inputs.

2. JK to SR (Convert JK to Act Like SR)

- Target: SR ($Q_{n+1} = S + R' Q_n$)
- Source: JK ($Q_{n+1} = J Q_n' + K' Q_n$)
- Conversion Table (Inputs: S, R, Q_n; Desired Q_{n+1}; Find J, K; avoid S=R=1 invalid):

S	R	Q_n	Desired Q_{n+1}	J	K
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	X
1	0	1	1	X	0
1	1	-	Invalid	-	-

- Minimized Expressions:
  - $J = S$
  - $K = R$
- Note: JK naturally avoids SR's invalid state. No extra gates needed if invalid is ignored, but add logic like AND(NOT S, R) if strict.

3. JK to D (Convert JK to Act Like D)

- Target: D ($Q_{n+1} = D$)
- Source: JK
- Conversion Table:

D	Q_n	Desired Q_{n+1}	J	K
0	0	0	0	X
0	1	0	0	1

1	0	1	1	X
1	1	1	X	0

- Minimized Expressions:
  - J = D
  - K = D'
- Circuit: Connect D directly to J; invert D to K (using NOT gate).

4. D to JK (Convert D to Act Like JK)

- Target: JK
- Source: D (Q_{n+1} = D)
- Conversion Table:

J	K	Q_n	Desired Q_{n+1}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

- Minimized Expression: D = J Q_n' + K' Q_n
- Circuit: Use OR gate with two AND gates: (J AND Q_n') OR (K' AND Q_n).

5. JK to T (Convert JK to Act Like T)

- Target: T (Q_{n+1} = T $\oplus$ Q_n)
- Source: JK
- Conversion Table:

T	Q_n	Desired Q_{n+1}	J	K
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	0	1

- Minimized Expressions:
  - J = T
  - K = T
- Circuit: Connect T to both J and K (no extra gates).

6. T to JK (Convert T to Act Like JK)

- Target: JK
- Source: T (Q_{n+1} = T $\oplus$ Q_n)
- Conversion Table:

J	K	Q_n	Desired Q_{n+1}	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1

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

- Minimized Expression: T = J Q_n' + K Q_n
- Circuit: OR gate with two AND gates: (J AND Q_n') OR (K AND Q_n).

7. D to T (Convert D to Act Like T)

- Target: T
- Source: D
- Conversion Table:

T	Q_n	Desired Q_{n+1}	D
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

- Minimized Expression: D = T $\oplus$ Q_n
- Circuit: XOR gate between T and Q_n connected to D input.

8. T to D (Convert T to Act Like D)

- Target: D
- Source: T
- Conversion Table:

D	Q_n	Desired Q_{n+1}	T
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

- Minimized Expression: T = D $\oplus$ Q_n
- Circuit: XOR gate between D and Q_n connected to T input.

9. SR to D (Convert SR to Act Like D)

- Target: D
- Source: SR
- Conversion Table:

D	Q_n	Desired Q_{n+1}	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

- Minimized Expressions:
  - S = D Q_n'
  - R = D' Q_n (or simply R = D')
- Circuit: AND for S (D and Q_n'), AND for R (D' and Q_n) or direct inverter if simplified.

10. D to SR (Convert D to Act Like SR)

- Target: SR
- Source: D
- Conversion Table (avoiding S=R=1):

S	R	Q_n	Desired Q_{n+1}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1

- Minimized Expression: D = S + R' Q_n
- Circuit: OR gate between S and (R' AND Q_n).
  These conversions ensure the source flip-flop mimics the target. For hardware, use gates like AND, OR, NOT, XOR as described. If implementing in FPGA or simulation, verify with timing considerations.

11.Explain the difference between edge triggering and level triggering with diagrams.

In digital electronics, edge triggering and level triggering are mechanisms that determine when a flip-flop or latch responds to input signals, typically controlled by a clock or enable signal. These concepts are fundamental to sequential circuits, governing how data is captured and stored. This document explains the differences between edge triggering and level triggering, their characteristics, and provides textual descriptions of timing diagrams to illustrate their behavior.

Edge triggering occurs when a flip-flop responds to input changes only at a specific transition (or “edge”) of the clock signal, either the rising edge (low-to-high) or the falling edge (high-to-low). The circuit ignores input changes during the rest of the clock cycle.

Edge triggering occurs when a flip-flop responds to input changes only at a specific transition (or “edge”) of the clock signal, either the rising edge (low-to-high) or the falling edge (high-to-low). The circuit ignores input changes during the rest of the clock cycle.

Key Differences Between Edge Triggering and Level Triggering

Feature	Edge Triggering	Level Triggering

Activation	Triggered at the rising or falling edge of the clock.	Triggered during the entire high or low clock level.
Timing Window	Narrow (only at the edge).	Wide (entire duration of the active level).
Circuit Type	Typically flip-flops (e.g., D, JK).	Typically latches (e.g., SR, D).
Stability	Less prone to glitches, suitable for synchronous systems.	More prone to glitches, used in simpler circuits.
Output Behavior	Output changes only at the clock edge.	Output follows input during active clock level.
Applications	Synchronous counters, registers, CPUs.	Data latches, asynchronous circuits.

12. What is a register? Explain classification of registers and working of a shift register with diagrams.

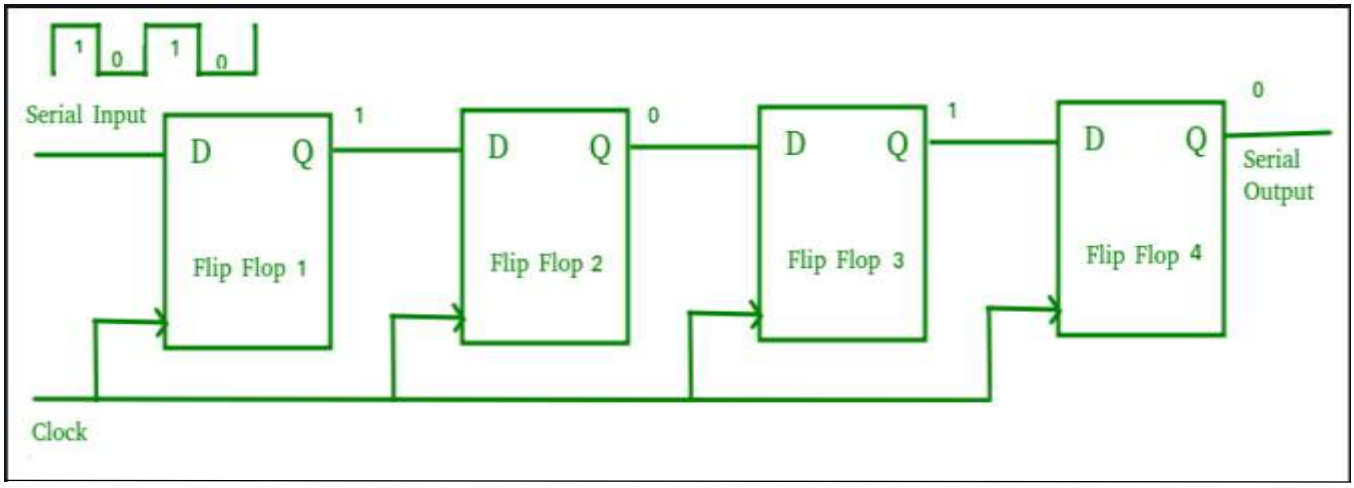
A register is a fundamental component in digital electronics, consisting of a group of flip-flops used to store multiple bits of data temporarily during processing in a digital system. Each flip-flop in a register stores one bit, so an n-bit register comprises n flipflops. Registers are essential in CPUs, memory units, and other digital circuits for tasks such as:

- Storing intermediate results during computations.
- Holding data for processing or transfer.
- Performing operations like shifting or counting.

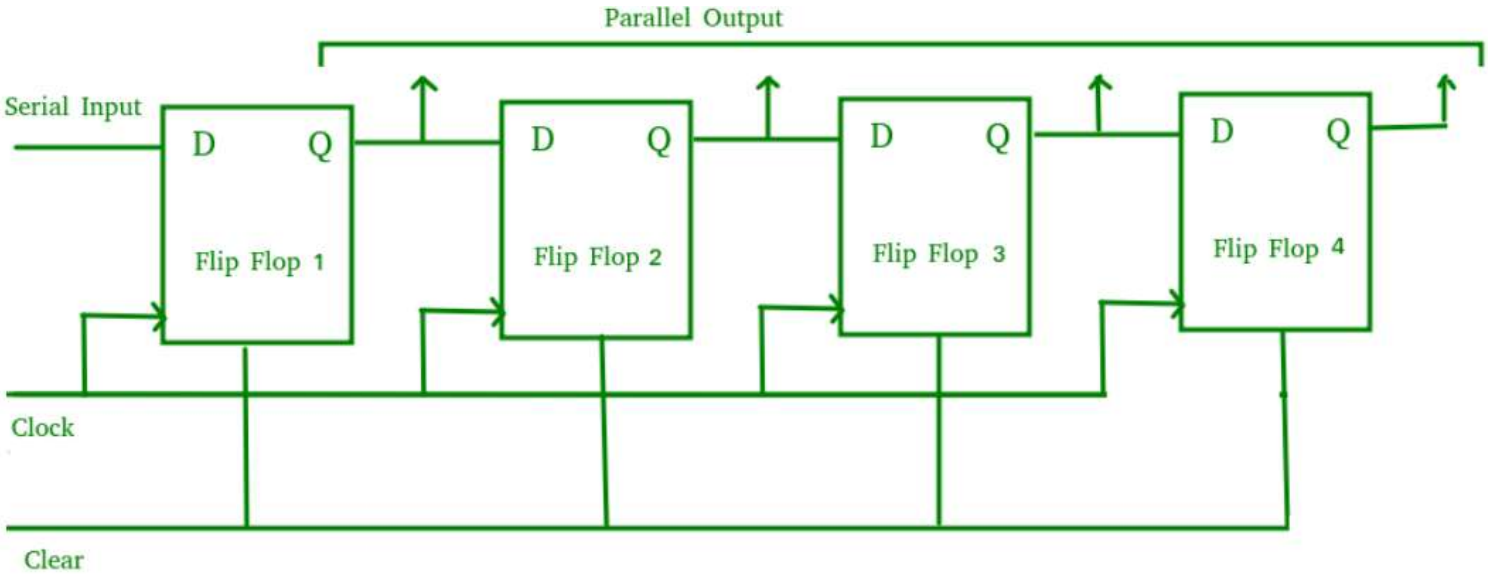
For example, a 4-bit register can store a 4-bit binary number (e.g., 1011) using four flip-flops, typically D flip-flops, synchronized by a clock signal.

Classification of Registers :-
Registers are classified based on how data is entered and retrieved. The main types are:

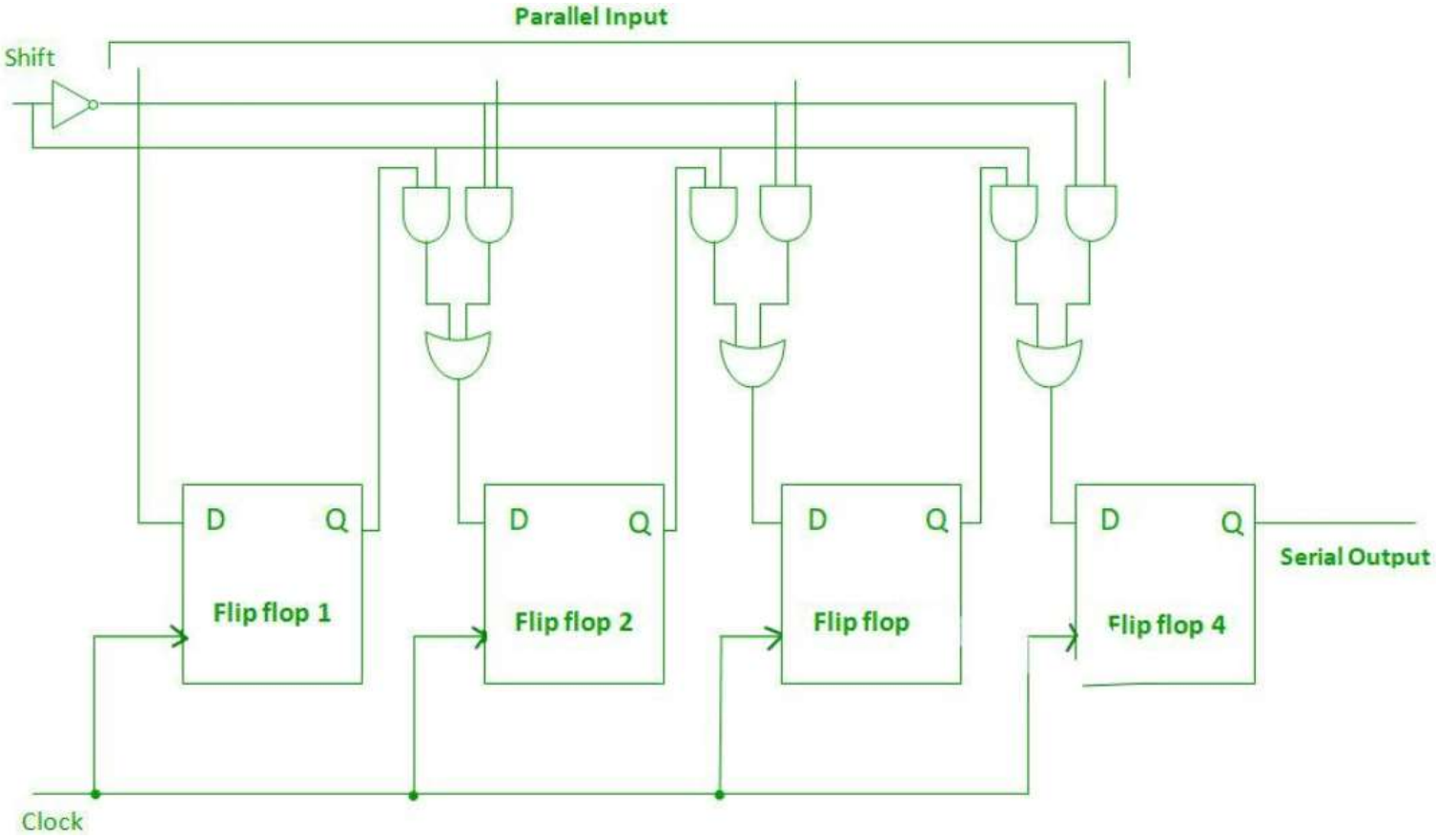
1. **Serial-In Serial-Out (SISO) Register:** Data is entered and retrieved sequentially, one bit at a time. Used in serial data communication.



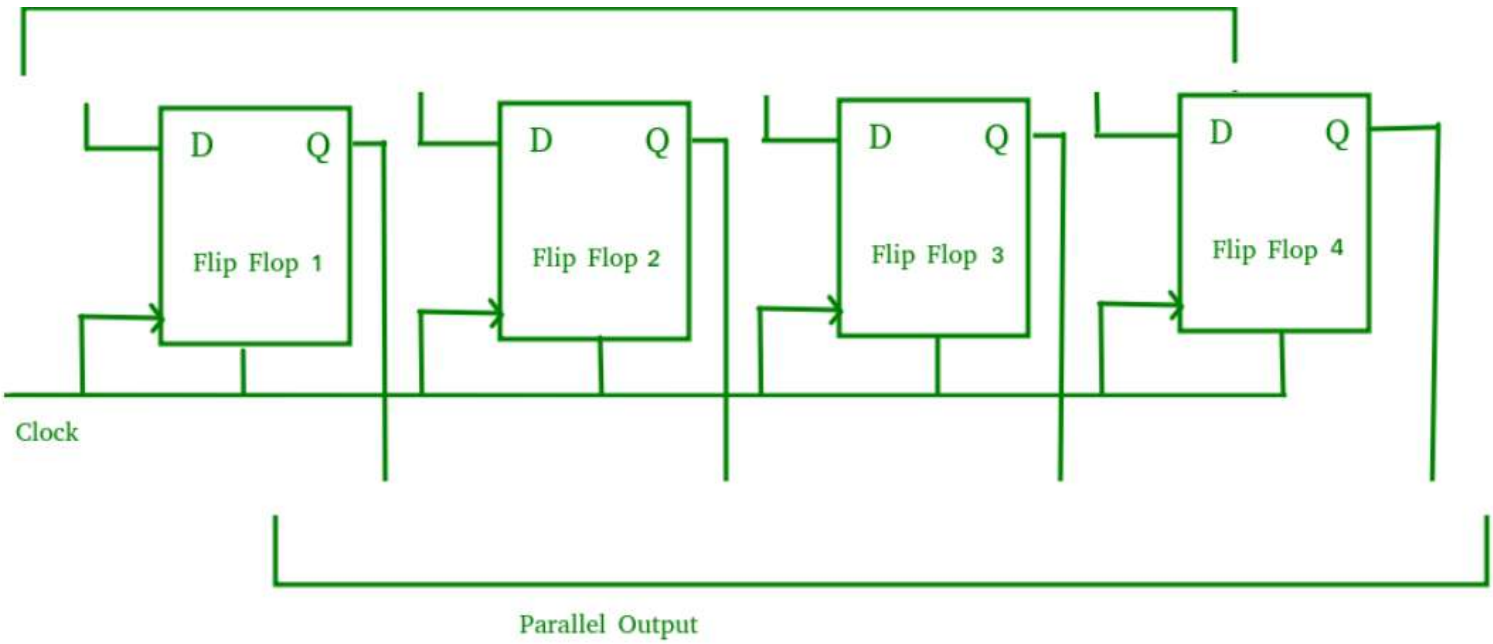
2. **Serial-In Parallel-Out (SIPO) Register:** Data is entered serially but retrieved in parallel (all bits simultaneously). Useful for serial-to-parallel data conversion.



3. **Parallel-In Serial-Out (PISO) Register:** Data is entered in parallel but retrieved serially. Used for parallel-to-serial data conversion.



4. **Parallel-In Parallel-Out (PIPO) Register:** Data is both entered and retrieved in parallel. Commonly used for temporary storage in processors.



5. **Universal Shift Register:** A versatile register that can operate as SISO, SIPO, PISO, or PIPO based on control inputs, supporting operations like left or right shifting.

Working of a Shift Register :-

A shift register is a type of register that shifts its stored data left or right by one bit position with each clock pulse. It is used in applications like data serialization, deserialization, and delay lines. This section explains the working of a Serial-In Serial-Out (SISO) shift register as an example.

Components of a SISO Shift Register:-

- **Flip-Flops:** Typically D flip-flops, each storing one bit.
- **Clock Signal:** Synchronizes the shifting process.
- **Serial Input:** The input data bit.
- **Serial Output:** The output data bit.

Operation :-

The SISO shift register consists of flip-flops connected in a chain, where the output of one flip flop feeds into the input of the next. Data is entered serially, shifted through the flip-flops with each clock pulse, and output serially. The steps are:

1. **Initialization:** All flip-flops are reset (e.g., to 0).
2. **Data Input:** A single bit is input to the first flip-flop at each clock pulse.
3. **Shifting:** With each clock pulse, data shifts to the next flip-flop.
4. **Output:** After n clock pulses (for an n-bit register), data appears at the output.

Example:-

Consider a 4-bit SISO shift register with input data 1010:

- Initial State: Q3 Q2 Q1 Q0 = 0000.
- Clock 1: Input = 1, Register = 1000.
- Clock 2: Input = 0, Register = 0100.
- Clock 3: Input = 1, Register = 1010.
- Clock 4: Input = 0, Register = 0101.

- Clock 5: Output starts, first bit (1) appears,

Register = 0010. After 8 clock pulses, the data 1010 is output serially

13. What is a counter? Differentiate between asynchronous (ripple) and synchronous counters with examples.

In digital electronics**, a counter** is a sequential logic circuit that consists of a series of flip-flops. As the name suggests, counters are used to count the number of occurrences of an input in terms of negative or positive edge transitions.

Based on the way the flip-flops are triggered, counters can be grouped into two categories: Synchronous counters and Asynchronous counters.

Here we will discuss how these two types of counters function and how they are different from each other.

Synchronous Counter

If the **clock** pulses are applied to all the flip-flops in a counter simultaneously, then such a counter is called as synchronous counter.

- In a synchronous counter, all the constituting flip-flops are clocked with the same clock input simultaneously. These are also known as *parallel counters*.
- Basically, all the flip-flops in a synchronous counter are arranged in a cascade connection and each flip-flop is individually connected to an external clock. It allows the clocking of all the flip-flops at the same time instant with the same clock input. It means the output of each flip-flop varies in synchronization with the clock input.
- Due to this, the common clock signal causes the change in the state of each individual flip-flop simultaneously. Resultantly it leads to no ripple effect, thus there is no propagation delay in a synchronous counter.
- Logic gates are used in synchronous counters to control the count sequence.

Asynchronous Counter

Asynchronous counters are also known as *serial counters* because the flip-flops that constitute the counter are connected serially and the input clock pulse is provided to the first flip-flop in the connection.

- The output of the first flip-flop acts as the input of the next adjacent flip-flop in the forward direction. In this manner, the clock input ripples through the counter. Hence, these counters are also known as *ripple counters*.
- Due to the ripple effect, the timing signal in an asynchronous counter gets delayed by some amount on passing through each flip flop. Hence, it results in a propagation delay.

Difference Between Synchronous and Asynchronous Counters

The following table highlights the major differences between Synchronous and Asynchronous Counters.

Key	**Synchronous Counter**	**Asynchronous Counter**
Trigger	In case of Synchronous Counters, all the constituent flip-flops are triggered with same clock simultaneously.	In case of Asynchronous Counters, there is triggering of different flip-flops with different clock.
Operation Speed	Operation speed of a synchronous counter is faster as compared to that of an asynchronous counter.	The operation speed of an asynchronous counter is comparatively slower than a synchronous counter.
Error Prone	Synchronous Counters are less error-prone; they hardly produce any decoding errors because each flip-flop is individually clocked.	Asynchronous Counters are more error-prone and produce decoding errors in the system.
Complexity	All the flip-flops in a synchronous counter coordinate with the clock, hence its design and implementation is complex as	In an asynchronous counter, the output of one flip-flop acts as the input of the next flip-flop, hence its

	compared to that of an asynchronous counter.	design and implementation is quite simple.
Sequence	A Synchronous counter can be operated in any desired count sequence, as it could get manipulated by changing the clock sequence.	An Asynchronous counter can operate only in a fixed count sequence, i.e., UP and DOWN.
Delay	There is no propagation delay observed in case of Synchronous Counters.	In case of asynchronous counters, there is a subsequent propagation delay from one flip-flop to another.

14. Explain the basic cell organization of static RAM (SRAM) and dynamic RAM (DRAM) with diagrams.

RAM stands for Random Access Memory. It is the internal memory of the CPU for storing data, program, and program result. It is a read/write memory which stores data until the computer is working. As soon as the computer is switched off, data is erased. Therefore, RAM is a volatile memory.

SRAM stands for Static Random Access Memory. Each memory cell of SRAM is made up of a flip-flop, a 1-bit storage device. SRAM uses a matrix of 6 transistors. In this memory circuit, capacitors are not used. Thus, in SRAM, there is no data leakage, so SRAM need not be refreshed regularly.

SRAM is a high speed random access memory which is used in special applications such as cache memory in computers and other embedded systems. However, SRAM is relatively expensive because it uses comparatively more number of chips that increase its manufacturing cost. SRAM is a volatile memory which means it retains the stored data as long as the power is supplied to the computer.

DRAM stands for **Dynamic Random Access Memory**. Each memory cell of DRAM is made up of one transistor and one capacitor. In DRAM, the data and information is stored in the form of an electric charged on the capacitor. Since capacitor loses its data (charge), thus DRAM must be continually refreshed several hundred times per second to maintain the data.

DRAM is a small sized and less expensive type of RAM. For this reason, it is used as RAM in most computer systems. However, DRAM is relatively slower and has a short data life than SRAM.

Parameter	**SRAM**	**DRAM**
Full Form	SRAM stands for Static Random Access Memory.	DRAM stands for Dynamic Random Access Memory.
Component	SRAM stores information with the help of transistors.	DRAM stores data using capacitors.
Need to Refresh	In SRAM, capacitors are not used which means refresh is not needed.	In DRAM, contents of a capacitor need to be refreshed periodically.
Speed	SRAM provides faster speed of data read/write.	DRAM provides slower speed of data read/write.
Power Consumption	SRAM consumes more power.	DRAM consumes less power.
Data Life	SRAM has long data life.	DRAM has short data life.
Cost	SRAM are expensive.	DRAM are less expensive.
Density	SRAM is a low density device.	DRAM is a high density device.
Usage	SRAMs are used as cache memory in computer and other computing devices.	DRAMs are used as main memory in computer systems.

15.What is cache memory? Explain cache memory organization and virtual memory organization with examples.

Cache memory increases the access speed of the CPU. It is not a technique but a memory unit, i.e. a storage device. In cache memory, recently used data is copied. Whenever the program is ready to be executed, it is fetched from the main memory and then copied to the cache memory. But, if its copy is already present in the cache memory, then the program is directly executed.

Virtual Memory increases the capacity of main memory. Virtual memory is not a storage unit, its a technique. In virtual memory, even such programs which have a larger size than the main memory are allowed to be executed.

Difference Between Virtual Memory and Cache Memory

Virtual Memory	**Cache Memory**
Virtual memory increases the capacity of main memory.	While cache memory increase the accessing speed of CPU.
Virtual memory is not a memory unit, its a technique.	Cache memory is exactly a memory unit.
The size of virtual memory is greater than the cache memory.	While the size of cache memory is less than the virtual memory.
Operating System manages the Virtual memory.	On the other hand hardware manages the cache memory.
In virtual memory, the program with size larger than the main memory are executed.	While in cache memory, recently used data is copied into.
In virtual memory, mapping frameworks is needed for mapping virtual address to physical address.	While in cache memory, no such mapping frameworks is needed.

Virtual Memory	Cache Memory
It is not as speedy as cache memory.	It is a fast memory.
Those data or programs are kept here that are not completely get placed in the main memory.	The frequently accessed data is kept in cache memory in order to reduce the access time of files.
Users are able to execute the programs that take up more memory than the main memory.	The time required by CPU to access the main memory is more than accessing the cache. That is the reason frequently accessed data is stored in cache memory so that accessing time can be minimized.