LEARNING MATERIAL

ON

DIGITAL ELECTRONICS

(3RD SEMESTER)

DEPARTMENT OF INFORMATION TECHNOLOGY

Prepared by
Ms. P.Bhabani
Lect.ETC
Govt.Polytechnic,Bhubaneswar

UNIT-1: BASICS OF DIGITAL ELECTRONICS

1.1 Number System-Binary, Octal, Decimal, Hexadecimal - Conversion from one system to another number system.

In digital electronics, the number system is used for representing the information. The four most common number system are:

1. Decimal number system (Base- 10)
2. Binary number system (Base- 2)
3. Octal number system (Base-8)
4. Hexadecimal number system (Base- 16)

The Base or Radix of the number system is the total number of digit used in the number system.
For example, in decimal number system, it represents digit from 0-9 then the base of the system is 10.

Important terms:

Bit: A bit is the smallest unit of information. Bits in computer are grouped to form a lager unit of information

Byte: A byte is a combination of eight bits. Eight bits represent a character and is called a byte.

Nibble: A nibble is a combination of four bits, in other words a nibble is half a byte.

Word: a word is a combination of 16 bits, 32 bits or 64 bits depending on the computer. 16 is known as quad word.

Kilobyte (kb): A kilobyte has a memory unit of 1024 (2^{10})

Megabyte: a megabyte has a memory unit if 1,048,576 (2^{20}). This is approximately 1 million bytes.

Gigabytes (Gb): A gigabyte has a memory unit of 1,073,741,824 (2**30). This approximately billion bytes.

Terabytes: a terabyte has a memory unit of (2**40). This is approximately 1 trillion bytes. **Note:**

1024 byte = 1 kilobyte (1kb)
1024 kilobytes = 1 megabytes (1mb)
1024 megabyte = 1 gigabytes (1 Gb)
1024 gigabytes =1 terabyte (1tb)

Decimal Number System

Decimal number system has base 10 as it uses 10 digits from 0 to 9. In decimal number system, the successive positions to the left of the decimal point represent units, tens, hundreds, thousands, and so on.

Each position represents a specific power of the base (10). For example, the decimal number 1234 consists of the digit 4 in the units position, 3 in the tens position, 2 in the hundreds position, and 1 in the thousands position. Its value can be written as

(1 x 1000+(2 x 100+(3 x 10+(4 x l)
(1 x 10^3+(2 x 10^2+ (3 x 10^1+ (4 x $l0^0$)
1000 + 200 + 30 + 4
1234

Binary Number System

Characteristics of the binary number system are as follows −

- Uses two digits, 0 and 1
- Also called as base 2 number system
- First position in a binary number represents a **0** power of the base (2). Example 2^0
- Last position in a binary number represents a **x** power of the base (2). Example 2^x where **x** represents the last position-1 **Example**

Binary Number: 10101_2

Octal Number System

Characteristics of the octal number system are as follows −

- Uses eight digits, 0,1,2,3,4,5,6,7
- Also called as base 8 number system
- First position in an octal number represents a **0** power of the base (8). Example 8^0
- Last position in an octal number represents a **x** power of the base (8). Example 8^x where **x** represents the last position - 1 **Example**

Octal Number: 12570_8

Hexadecimal Number System

Characteristics of hexadecimal number system are as follows −

- Uses 10 digits and 6 letters, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Letters represent the numbers starting from 10. A = 10. B = 11, C = 12, D = 13, E = 14, F = 15
- Also called as base 16 number system
- First position in a hexadecimal number represents a **0** power of the bas e (16). Example, 16^0
- Last position in a hexadecimal number represents a **x** power of the base (16). Example 16^x where **x** represents the last position - 1

Example

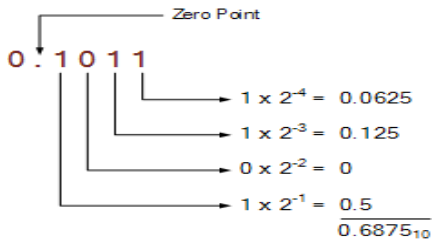
Hexadecimal Number: $19FDE_{16}$

Conversion of Number System

Binary to Decimal

Step	Binary Number	Decimal Number
Step 1	10101_2	$((1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	10101_2	$(16 + 0 + 4 + 0 + 1)_{10}$
Step 3	10101_2	21_{10}

Example 1



Example 2

$1101.0111 = (1\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0) + (0\times2^{-1}) + (1\times2^{-2}) + (1\times2^{-3}) + (1\times2^{-4})$

$= 8 + 4 + 0 + 1 + 0 + 1/4 + 1/8 + 1/16$

$= 8 + 4 + 0 + 1 + 0 + 0.25 + 0.125 + 0.0625 = 13.4375_{10}$

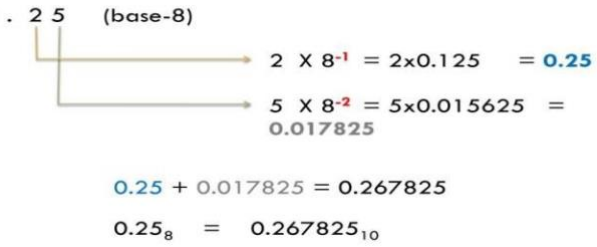
Hence the decimal equivalent number of 1101.0111_2 is given as: 13.4375_{10}

Octal to Decimal

Step	Octal Number	Decimal Number
Step 1	12570_8	$((1 \times 8^4) + (2 \times 8^3) + (5 \times 8^2) + (7 \times 8^1) + (0 \times 8^0))_{10}$

Step 2	12570_8	$(4096 + 1024 + 320 + 56 + 0)_{10}$
Step 3	12570_8	5496_{10}

Example 1



Hexadecimal to Decimal

Step	Binary Number	Decimal Number
Step 1	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (F \times 16^2) + (D \times 16^1) + (E \times 16^0))_{10}$
Step 2	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (15 \times 16^2) + (13 \times 16^1) + (14 \times 16^0))_{10}$
Step 3	$19FDE_{16}$	$(65536 + 36864 + 3840 + 208 + 14)_{10}$
Step 4	$19FDE_{16}$	106462_{10}

Example 1

Convert $(E F . B1)_{16} = (?)_{10}$

$= E \times 16^1 + F \times 16^0 + B \times 16^{-1} + 1 \times 16^{-2}$

$= 14 \times 16 + 15 \times 1 + 11 \times (1 / 16) + 1 \times (1 / 256)$

$= 224 + 15 + (0. 6 8 75) + (0. 0 0 3 9 0 6 2 5)$

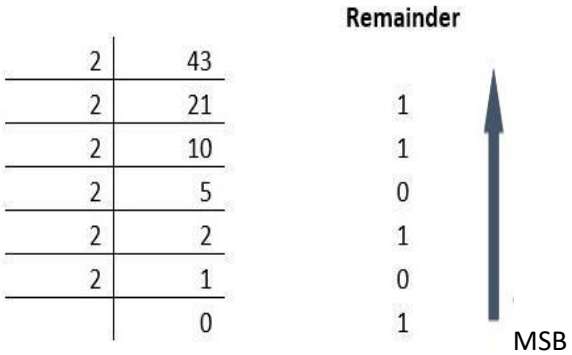
$= 239 + 0. 6914$

$= 239.691406$

Therefore $(E F . B 1)_{16} = (2 3 9.6 9 1 4 0 6)_{10}$

Decimal to Binary

Decimal numbers can be converted to binary by repeated division of the number by 2 while recording the remainder.



The remainders are to be read from bottom to top to obtain the binary equivalent. 43_{10}

$= 101011_2$

$(458.692)_{10} = (?)_2$

.692 x 2 = 1.384	1	MSB
.384 x 2 = 0.768	0	
.768 x 2 = 1.536	1	
.536 x 2 = 1.072	1	LSB

Decimal to Octal

Decimal numbers can be converted to octal by repeated division of the number by 8 while recording the remainder.

		Remainder
8	473	
8	59	1
8	7	3
	0	7

Reading the remainders from bottom to top,

$473_{10} = 731_8$

Convert $(965.198)_{10}$ to $(?)_8$

0.198 * 8 = 1.584 MSB
0.584 * 8 = 4.672
0.672 * 8 = 5.376

0.376 * 8 = 3.008
0.008 * 8 = 0.064
0.064 * 8 = 0.512 LSB

$(0.198)_{10} = (0.145300)_8$.

Decimal to Hexadecimal

Decimal numbers can be converted to octal by repeated division of the number by 16 while recording the remainder.

		Remainder
16	423	
16	26	7
16	1	A
	0	1

Reading the remainders from bottom to top we get,

$423_{10} = 1A7_{16}$

$(0.06640625)_{10}=( )_{16}$

Multiplication	Resultant integer part

0.06640625 x 16=1.0625	1
0.0625 x 16 =1.0	1
0 x 16=0.0	0

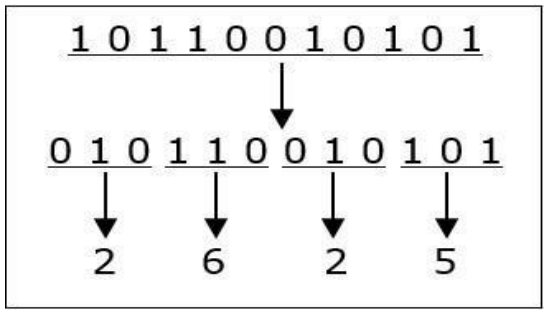
$(0.06640625)_{10}=(0.110)_{16}$

Binary to Octal and Vice Versa

To convert a binary number to octal number, these steps are followed –

Starting from the least significant bit, make groups of three bits.

- If there are one or two bits less in making the groups, 0s can be added after the most significant bit
- Convert each group into its equivalent octal number



$1011001010_{12} = 2625_8$

To convert an **octal number to binary**, each octal digit is converted to its 3-bit binary equivalent according to this table.

Octal Digit	0	1	2	3	4	5	6	7
Binary Equivalent	000	001	010	011	100	101	110	111

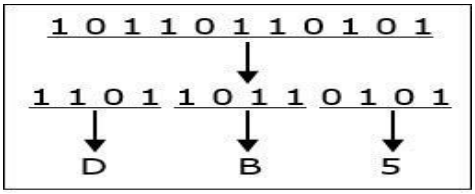
$54673_8 = 101100110111011_2$

Binary to Hexadecimal

To convert a binary number to hexadecimal number, these steps are followed –

Starting from the least significant bit, make groups of four bits.

- If there are one or two bits less in making the groups, 0s can be added after the most significant bit.
- Convert each group into its equivalent hexadecimal number.



$10110110101_2 = DB5_{16}$

To convert a **hexadecimal number to binary**, each hexadecimal digit is converted to its 4-bit binary equivalent according to this table.

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

Number System Relationship

The following table depicts the relationship between decimal, binary, octal and hexadecimal number systems.

HEXADECIMAL	DECIMAL	OCTAL	BINARY
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011

4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
A	10	12	1010
B	11	13	1011
C	12	14	1100
D	13	15	1101
E	14	16	1110
F	15	17	1111

Octal to Hexadecimal and vice versa

While converting from octal to hexadecimal, first convert the octal number into binary digit and then further from binary to hexadecimal.

Convert 536 from octal to hexadecimal number

$(536)_8$ = (101) (011) (110)

$=(101011110)_2$

Now forming the group of 4 binary bits to obtain its hexadecimal equivalent,

$(101011110)_2$= (0001) (0101) (1110)

$= (15E)_{16}$

So the hexadecimal number of 536 is 15E.

While converting from hexadecimal to octal, first convert the hexadecimal number into binary digit and then further from binary to octal.

Convert 15E hexadecimal number to Octal number

$(15E)_{16} = (0001)\ (0101)\ (1110)$

$= (000101011110)_2$

Now forming the group of 3 binary bits to obtain its octal equivalent,

$(000101011110)_2 = (101)\ (011)\ (110) = (536)_8$

1.2 Arithmetic Operation-Addition, Subtraction, Multiplication, Division, 1's & 2's complement of Binary numbers& Subtraction using complements method

ADDITION:

The four rules of binary addition are:

0 + 0 = 0

0 + 1 = 1
1 + 0 = 1

1 + 1 = 0 with a carry-over of 1 **Find the**

sum of the following numbers:

i) 10101 and 11011

Solution:
10101 and 11011

```
  1 1 1 1    Carry overs
  1 0 1 0 1
  1 1 0 1 1
1 1 0 0 0 0
```

ii) 11001 and 111

Solution:
11001 and 111

```
  1 1 1 1    Carry overs
  1 1 0 0 1
      1 1 1
1 0 0 0 0 0
```

iii) 10101.101 and 1101.011

Solution:

10101.101 and 1101.011

```
  1 1   1 1   1 1    Carry overs
  1 0 1 0 1 . 1 0 1
    1 1 0 1 . 0 1 1
1 0 0 0 1 1 . 0 0 0
```

SUBTRACTION:

The four rules of binary subtraction are:

0 - 0 = 0

1 - 0 = 1

1 - 1 = 0

0 - 1 = 1 with a borrow of 1 **Subtract**

the following numbers:

i) 101 from 1001 Solution:

101 from 1001

```
  1 Borrow
  1 0 0 1
    1 0 1
    1 0 0
```

ii) 111 from 1000

Solution:

111 from 1000

```
    1 Borrow
  1 0 0 0
    1 1 1
  0 0 0 1
```

iii) 11010.101 from 101100.011

Solution:

11010.101 from 101100.011

```
    1     1  1    Borrow

  1 0 1 1 0 0 . 0 1 1
    1 1 0 1 0 . 1 0 1
  1 0 0 0 1 . 1 1 0
```

MULTIPLICATION:

The rules for multiplication

$0 \times 0 = 0$
$1 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 1 = 1 (\textit{there is no carry or borrow for this})$

Examples of Binary Multiplication

```
  10101
×   101
-------
  10101
 000000
1010100
-------
1101001
```

```
   1011.01
×    110.1
---------
   101101
  0000000
 1011010 0
1011010 00
-----------
1001001.001
```

DIVISION:

The main rules of the binary division include:

- 1÷1 = 1
- 1÷0 = 0
- 0÷1 = Meaningless
- 0÷0 = Meaningless

Evaluate:

(i) 11001 ÷ 101

Solution:

```
101) 11001 (101
     101
      101
      101
```

Hence the quotient is 101

(ii) 11101.01 ÷ 1100

Solution:

```
1100)  11101.01  (10.0111
     1100
       10101
       1100
         0010
         1100
           1100
           1100
```

Hence the quotient is 10.0111

SIGNED-MAGNITUDE REPRESENTATION

In this system, a number consists of a magnitude and a symbol which indicates whether the magnitude is positive or negative.

In binary numbers system an extra bit position is used to represent the sign. This extra bit is called the SIGN BIT and is placed before the magnitude of the number to be represented. Generally, the MSB is the sign bit and the convention is that when the sign bit is 0, the number represented is positive and when the sign bit is 1, the number is negative.

$(00000000)_2 = +(0)_{10}$

$(10000000)_2 = -(0)_{10}$

1'S & 2'S COMPLEMENT OF BINARY NUMBERS

The 1's complement of a binary number can be obtained by substituting each 1 by 0 and 0 by 1.

Example:

Binary number-101100

1's complement = 010011

The 2's complement of a binary number is 1 added to the 1's complement of the binary number.

Example:

Binary number=101100

1's complement = 010011

2's complement= 1's complement + 1

=010100

SUBTRACTION USING COMPLEMENTS METHOD **1's**

complement subtraction:

- In 1's complement subtraction, add the 1's complement of subtrahend to minuend.
- If there is carry, then the carry is added to the LSB. This is called end around carry.
- If MSB is 0, the result is Positive.
- If the MSB is 1 then the result is Negative and is in its 1's complement form. Then take its 1's complement to get magnitude in binary.

  Evaluate:

  (i) 110101 – 100101 Solution:

  1's complement of 10011 is 011010. Hence

  Minued - 1 1 0 1 0 1

1's complement of subtrahend - 0 1 1 0 1 0

Carry over - 1 0 0 1 1 1 1

1

0 1 0 0 0 0

As MSB is 0, so the number is positive. Henc e the difference is 10000

(ii) 101011 – 111001 Solution:

1's complement of 111001 is 000110. Hence

Minued - 1 0 1 0 1 1

1's complement - 0 0 0 1 1 0

1 1 0 0 0 1

As MSB is 1, the number is negative. Take 1's complement of 10001 **Hence the difference is – 01 1 1 0**

(iii) 1011.001 – 110.10 Solution:

1's complement of 0110.100 is 1001.011 Hence

Minued - 1 0 1 1 . 0 0 1

1's complement of subtrahend - 1 0 0 1 . 0 1 1

Carry over - 1 0 1 0 0 . 1 0 0

1

0 1 0 0 . 1 0 1

Hence the required difference is 100.101

2's complement subtraction:

- In 2's complement, add 2's complement of subtrahend to the minuend.
- If there is a carry ignore it.
- If the MSB is 0, the result is Positive.
- If the MSB is 1 the result is Negative and is in 2's complement form. Then take its complement to get the magnitude in binary.

Evaluate:

10110 – 11010 Solution:

2's complement of 11010 is (00101 + 1) i.e. 00110. Hence

Minued - 1 0 1 1 0

2's complement of subtrahend - 0 0 1 1 0

Result of addition - 1 1 1 0 0

As MSB is 1, the number is negative. Take 2's complement of 11100

So, 1's complement of 11100 is 00011 2's complement is 00011+1 =00100 Hence the difference is – 100.

1.3 Digital Code & its application & distinguish between weighted & non-weight Code, Binary codes, excess3 and Gray codes.

- Digital coding is the process of using binary digits to represent letters, characters and other symbols in a digital format.
- There are several types of digital codes widely used today, but they use the same principle of combining binary numbers to represent a character.
- The most widely used Digital Codes are ASCII (American Standard Code Information Interchange), BCD (Binary Coded Decimal), EBCDIC (Extended Binary Coded Decimal Interchange Code) and Unicode.

Binary codes can be classified into two types.

- Weighted codes
- Unweighted codes

In weighted codes, each digit is assigned a specific weight according to its position.
Examples:8421code,2421code

The Non - Weighted Code are not positionally weighted. In other words, codes that are not assigned with any weight to each digit position.
Examples: Excess 3 code, Grey code

Excess 3 code

- Excess-3 code doesn't have any weights. So, it is a non-weighted code.
- We will get the Excess 3 code of a decimal number by adding three to the binary equivalent of that decimal number. Hence, it is called as Excess 3 code.
- It is a self-complementing code.

Convert the following numbers to Excess-3 code

(a) 87 **(b) 159**

Solution:

(a)

8	7
+3	+3
11	10
↓	↓
1011	1010

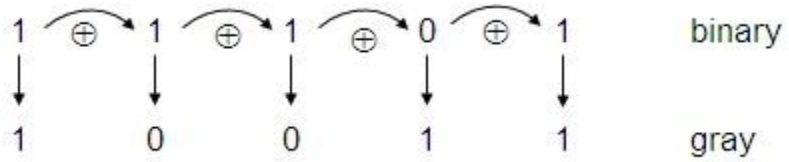
(b)

1	5	9
+3	+3	+3
4	8	12
↓	↓	↓
0100	1000	1100

Grey Code

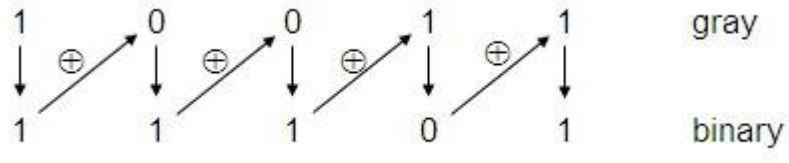
- The Grey code is a non-weighted code.
- No specific weight assigned to the bit positions.
- It exhibits only a single change from one code word to the next sequence. **Conversion of a Binary number to Gray code**

- The first bit(MSB) of the Gray code is the same as the first bit of the binary number
- The second bit of the Gray code equals the exclusive-OR, of the first and second bits of the binary number, i.e. it will be 1 if these binary code bits are different and 0 if they are the same.
- The third Gray code bit equals the exclusive-OR of the second and third bits of the binary number, and so on.



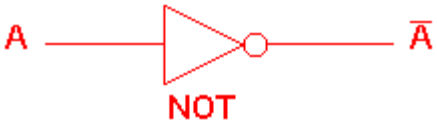
Conversion from Gray code to Binary

- The first binary bit(MSB) is the same as that of the first the Gray code bit.
- The second bit of the binary code equals the exclusive-OR, of the MSB and second bits of the gray code.
- Step 2 is repeated.



1.4 LOGIC GATES:

- Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on certain **logic**.
- In digital system, the basic gates are AND Gate, OR gate, NOT gate.
- NAND and NOR are called **universal gates** because all the other gates can be derived from it.
- Truth table is the list of all possible combination of input and the corresponding output. **NOT gate**



NOT gate	
A	$\overline{A}$
0	1
1	0

The NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an *inverter*. If the input variable is A, the inverted output is known as A'.

AND gate

2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

The AND gate is an electronic circuit that gives a **high** output (1) only if **all** its inputs are high. A dot (.) is used to show the AND operation i.e. A.B.

OR gate

2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

The OR gate is an electronic circuit that gives a high output (1) if **one or more** of its inputs are high. A plus (+) is used to show the OR operation.

NAND gate

2 Input NAND gate		
A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

This is an AND gate followed by a NOT gate. The outputs of all NAND gates are high if **any** of the inputs are low.

NOR gate

2 Input NOR gate		
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

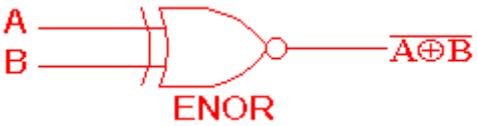
This is an OR gate followed by a NOT gate. The outputs of all NOR gates are low if **any** of the inputs are high.

EXOR gate

2 Input EXOR gate		
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

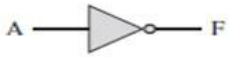
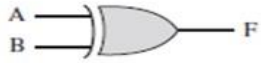
The '**Exclusive-OR**' gate is a circuit which will give a high output if **either, but not both**, of its two inputs are high. An encircled plus sign (⊕) is used to show the EOR operation.

EXNOR gate



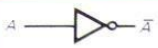
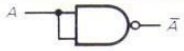
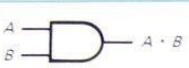
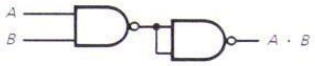
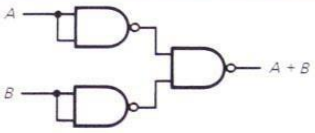
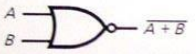
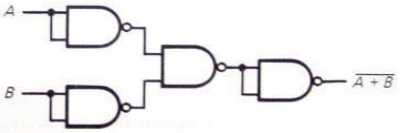
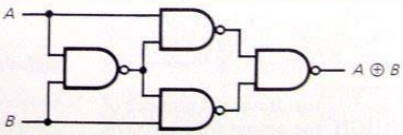
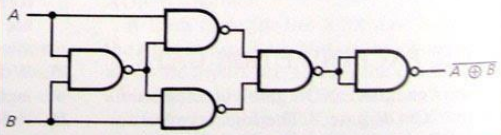
2 Input EXNOR gate		
A	B	$\overline{A⊕B}$
0	0	1
0	1	0
1	0	0
1	1	1

The '**Exclusive-NOR**' gate circuit does the opposite to the EOR gate. It will give a low output if **either, but not both**, of its two inputs are high. The symbol is an EXOR gate with a small circle on the output. The small circle represents inversion.

Name	Graphical Symbol	Algebraic Function	Truth Table
AND		$F = A \cdot B$ or $F = AB$	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = A + B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT		$F = \overline{A}$ or $F = A'$	A | F 0 | 1 1 | 0
NAND		$F = \overline{AB}$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = \overline{A + B}$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR		$F = A \oplus B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

1.5 UNIVERSAL GATES & ITS REALISATION:

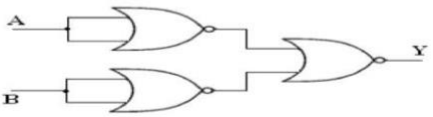
NAND and NOR are called **universal gates** because all the other gates can be derived from it.

LOGIC FUNCTION	SYMBOL	CIRCUIT USING NAND GATES ONLY
Inverter		
AND		
OR		
NOR		
XOR		
XNOR		

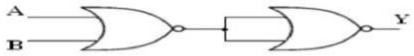
II. **Implementation using NOR gate:**

(a) NOT gate: $Y = A'$

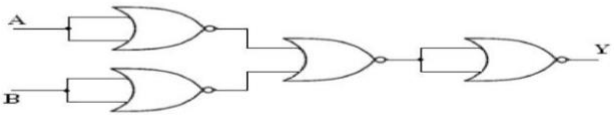
(b) AND gate: $Y = A \cdot B$



(c) OR gate: $Y = A + B$



(d) NAND gate: $Y = (AB)'$



(e) Ex-NOR gate: $Y = A \odot B = (A \oplus B)'$

1.6 BOOLEAN ALGEBRA, BOOLEAN EXPRESSION, DEMORGAN'S THEOREMS

- Boolean algebra is the set of rules used to simplify the logic expression without changing its functionality. ▪ It is used when number of variable is less (1,2,3)

Boolean Theorem:

1. INVERSION law

This law uses the NOT operation. The inversion law states that double inversion of a variable results in the original variable itself.

$\overline{\overline{A}} = A$

2. AND law

These laws use the AND operation. Therefore, they are called as **AND** laws.

(i) $A.0 = 0$ (ii) $A.1 = A$

(iii) $A.A = A$ (iv) $A.\overline{A} = 0$

3. OR law

These laws use the OR operation. Therefore, they are called as **OR** laws.

(i) $A + 0 = A$ (ii) $A + 1 = 1$

(iii) $A + A = A$ (iv) $A + \overline{A} = 1$

4. Commutative law

(i) $A.B = B.A$ (ii) $A + B = B + A$

Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

5. Associative law

(i) $(A.B).C = A.(B.C)$ (ii) $(A + B) + C = A + (B + C)$

This law states that when ORing/ANDing more than two variables, the result is the same regardless of the grouping of the variables.

6. Distributive law

Distributive law states the following condition.

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

This law states that ORing two or more variables and then ANDing the result with a single variable is equivalent to ANDing the single variable with each of the two or more variables and then ORing the product.

A+ (B.C) = (A+B). (A+C)

This law states that ANDing two or more variables and then ORing the result with a single variable is equivalent to ORing the single variable with each of the two or more variables and then ANDing the sum.

7. Absorption Laws

Absorption law involves in linking of a pair of binary operations.

i. A+AB = A ii. A(A+B) = A iii. A+$\bar{A}$B = A+B

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

8.Consensus Laws

i. AB + A'C + BC = AB + A'C ii. (A+B).

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

DEMORGAN'S THEOREM

- The first theorem states that the complement of a product is equal to the sum of the complements. That is if the variables are A and B, then

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

- The second theorem states that, the complement of a sum is equal to the product of the complements. In equation form, this can be written as

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

- Demorgan's Theorem can be proved for any number of variables; proof of these two theorems for two variables can be found in the below truth table

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

A study of truth table makes clear that columns 7 and 8 are equal. Therefore,

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

Similarly, columns 9 and 10 are equal. Therefore,

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

1.7 REPRESENT LOGIC EXPRESSION: SOP & POS FORMS

Sum Of Product (SOP)

- Sum of Product is the abbreviated form of SOP. Sum of product form is a form of expression in Boolean algebra in which different product terms of inputs are being summed together.
- This product is not arithmetical multiply but it is Boolean logical AND and the Sum is Boolean logical OR.
- Example:
- In SOP form, each product term is known as minterms
- The product of all literals, either with complement or without complement, is known as **minterm**. The min terms for two variable x and y are x'y', x'y, xy' and xy.
- If the variable value is 1, we will take the variable without its complement.

  If the variable value is 0, take its complement.
- The minterms can be represented by m_0, m_1, m_2, m_3……; the suffix indicates the decimal code corresponding to the minterm combination.
- **Types of Sum of Product (SOP) Forms**
  There are few different forms of Sum of Product.
  - Canonical SOP Form
  - Non-Canonical SOP Form
  - Minimal SOP Form

Canonical SOP Form

- Canonical SOP form means Canonical Sum of Products form. In this form, each product term contains all literals. So, these product terms are nothing but the min terms. Hence, canonical SOP form is also called as sum of min terms form.
- The expression of the canonical SOP is denoted with sign summation (∑), and the minterms in the bracket are taken when the output is true.
- The Boolean function of output is, f = p'qr + pq'r + pqr' + pqr. This is the canonical SOP form of output, f. We can also represent this function in following two notations. f=m3+m5+m6+m7
  f=∑ m (3,5,6,7)

Product of Sum(POS)

- **Product of Sum is the abbreviated for POS.** The product of Sum form is a form in which products of different sum terms of inputs are taken.
- These are not arithmetic product and sum but they are logical Boolean AND and OR respectively.
- Example:
- In POS form, each sum term is known as maxterms.
- The sum of all literals, either with complement or without complement, is known as **maxterm**. The maxterms for two variable x and y are x'+y', x'+y, x+y' and x+y.
- If the variable value is 0, we will take the variable without its complement.

  If the variable value is 1, take its complement.
- The maxterms can be represented by M_0, M_1, M_2, M_3……; the suffix indicates the decimal code corresponding to the maxterm combination.
- **Types of Sum Of Product (SOP) Forms**
  There are few different forms of Sum of Product.
  - Canonical SOP Form
  - Non-Canonical SOP Form
  - Minimal SOP Form

Canonical POS Form

- Canonical POS form means Canonical Product of Sums form. In this form, each sum term contains all literals. So, these sum terms are nothing but the Max terms. Hence, canonical POS form is also called as product of Max terms form.
- The expression this is denoted by ∏ and the max terms in the bracket are taken when the output is false.
- The Boolean function of output is, f = p+q+r. p+q+r'. p+q'+r. p'+q+r. This is the canonical POS form of output, f. We can also represent this function in following two notations.
  $f=M_0.M_1.M_2.M_4$ f=∏
  M (0,1,2,4)

Two variable minterms and maxterms.

x	y	Index	Minterm	Maxterm
0	0	0	$m_0 = \overline{x}\,\overline{y}$	$M_0 = x + y$
0	1	1	$m_1 = \overline{x}\,y$	$M_1 = x + \overline{y}$
1	0	2	$m_2 = x\,\overline{y}$	$M_2 = \overline{x} + y$
1	1	3	$m_3 = x\,y$	$M_3 = \overline{x} + \overline{y}$

Three variable minterms and maxterms

Variables			Min terms	Max terms
A	B	C	m_i	M_i
0	0	0	A' B' C' = m 0	A + B + C = M 0
0	0	1	A' B' C = m 1	A + B + C' = M 1
0	1	0	A' B C' = m 2	A + B' + C = M 2
0	1	1	A' B C = m 3	A + B' + C' = M 3
1	0	0	A B' C' = m 4	A' + B + C = M 4
1	0	1	A B' C = m 5	A' + B + C' = M 5
1	1	0	A B C' = m 6	A' + B' + C = M 6
1	1	1	A B C = m 7	A' + B' + C' = M 7

Conversion of SOP form to standard SOP form or Canonical SOP form

For getting the standard SOP form of the given non-standard SOP form, we will add all the variables in each product term which do not have all the variables. By using the Boolean algebraic law, (x + x' = 1) and by following the below steps we can easily convert the normal SOP function into standard SOP form.

- Multiply each non-standard product term by the sum of its missing variable and its complement.
- Repeat step 1, until all resulting product terms contain all variables ○ For each missing variable in the function, the number of product terms doubles.

Example:

Convert the non standard SOP function F = AB + A C + B C

F = A B + A C + B C

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

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

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

So, the standard SOP form of non-standard form is F = A B C + A B C' + A B' C + A' B C

Conversion of POS form to standard POS form or Canonical POS form

For getting the standard POS form of the given non-standard POS form, we will add all the variables in each product term that do not have all the variables. By using the Boolean algebraic law (x * x' = 0) and by following the below steps, we can easily convert the normal POS function into a standard POS form.

- By adding each non-standard sum term to the product of its missing variable and its complement, which results in 2 sum terms
- Applying Boolean algebraic law, x + y z = (x + y) * (x + z)
- By repeating step 1, until all resulting sum terms contain all variables By these three steps, we can convert the POS function into a standard POS function.

Example:

F = (p' + q + r) * (q' + r + s') * (p + q' + r' + s)

1. Term (p' + q + r)

As we can see that the variable s or s' is missing in this term. So we add s*s' = 0 in this term.

(p' + q + r + s*s') = (p' + q + r + s) * (p' + q + r + s')

2. Term (q' + r + s')

Similarly, we add p*p' = 1 in this term for getting the term containing all the variables.

(q' + r + s' + p*p') = (p + q' + r + s') * (p' + q' + r + s')

3. Term (q' + r + s')

Now, there is no need to add anything because all the variables are contained in this term.

So, the standard POS form equation of the function is

F = (p' + q + r + s)* (p' + q + r + s')* (p + q' + r + s')* (p' + q' + r + s') * (p + q' + r' + s)

Conversion of SOP form to POS form

For getting the POS form of the given SOP form expression, we will change the symbol ∏ to ∑. After that, we will write the numeric indexes of the variables which are missing in the boolean function.

There are the following steps used to convert the SOP function F = ∑ x, y, z (0, 2, 3, 5, 7) = x' y' z' + z y' z' + x y' z + xyz' + xyz into POS:

- In the first step, we change the operational sign to ∏.
- We find the missing indexes of the terms, 001, 110, and 100.
- We write the sum form of the noted terms.

```
001 = (x + y + z)

100 = (x + y' + z')

110 = (x + y' + z')
```

So, the POS form is:

```
F = Π x, y, z (1, 4, 6) = (x + y + z) * (x + y' + z') * (x + y' + z')
```

Conversion of POS to SOP form

For getting the SOP form from the POS form, we have to change the symbol ∏ to ∑. After that, we write the numeric indexes of missing variables of the given Boolean function.

There are the following steps to convert the POS function F = Π x, y, z (2, 3, 5) = x y' z' + x y' z + x y z' into SOP form:

1. In the first step, we change the operational sign to Σ.
2. Next, we find the missing indexes of the terms, 000, 110, 001, 100, and 111.
3. Finally, we write the product form of the noted terms.

```
000 = x' * y' * z'

001 = x' * y' * z

100 = x * y' * z'

110 = x * y* z'

111 = x * y * z
```

So the SOP form is:

```
F = Σ x, y, z (0, 1, 4, 6, 7) = (x' * y' * z') + (x' * y' * z) + (x * y' * z') + (x * y* z') + (x * y * z)
```

1.8 Karnaugh map

- Karnaugh map or K-Map method is most suitable for minimizing Boolean functions. With the help of the K-map method, we can find the simplest POS and SOP expression, which is known as the minimum expression.

- It is a graphical method, which consists of 2^n cells for 'n' variables. The adjacent cells are differed only in single bit position.
- Just like the truth table, a K-map contains all the possible values of input variables and their corresponding output values.
- The K-map method is used for expressions containing 2, 3, 4, and 5 variables.
- The K-map grid is filled using 0's and 1's. The K-map is solved by making groups. **These are the following steps used to solve the expressions using K-map:**

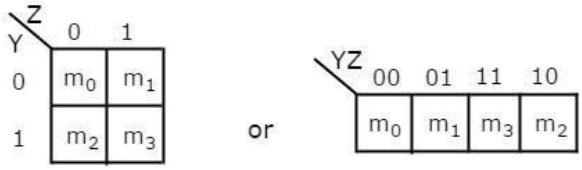
1. First, we find the K-map as per the number of variables. 2.

Find the maxterm and minterm in the given expression.

3. Fill cells of K-map for SOP with 1 respective to the minterm.
4. Fill cells of the block for POS with 0 respective to the maxterm.
5. Next, we create rectangular groups that contain total terms in the power of two like 2, 4, 8, … and try to cover as many elements as we can in one group.
6. With the help of these groups, we find the product terms and sum them up for the SOP form.

2 Variable K-Map

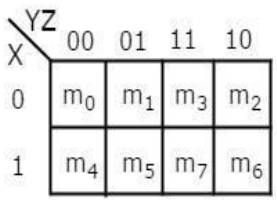
The number of cells in 2 variable K-map is four, since the number of variables is two. The following figure shows **2 variable K-Map**.



- There is only one possibility of grouping 4 adjacent min terms.
- The possible combinations of grouping 2 adjacent min terms are {(m_0, m_1), (m_2, m_3), (m_0, m_2) and (m_1, m_3)}.

3 Variable K-Map

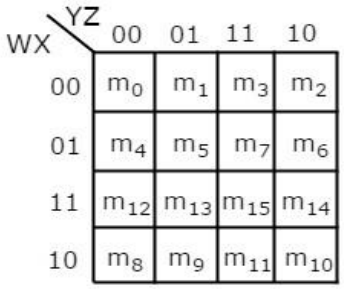
The number of cells in 3 variable K-map is eight, since the number of variables is three. The following figure shows **3 variable K-Map**.



- There is only one possibility of grouping 8 adjacent min terms.
- The possible combinations of grouping 4 adjacent min terms are {(m_0, m_1, m_3, m_2), (m_4, m_5, m_7, m_6), (m_0, m_1, m_4, m_5), (m_1, m_3, m_5, m_7), (m_3, m_2, m_7, m_6) and (m_2, m_0, m_6, m_4)}.
- The possible combinations of grouping 2 adjacent min terms are {(m_0, m_1), (m_1, m_3), (m_3, m_2), (m_2, m_0), (m_4, m_5), (m_5, m_7), (m_7, m_6), (m_6, m_4), (m_0, m_4), (m_1, m_5), (m_3, m_7) and (m_2, m_6)}.
- If x=0, then 3 variable K-map becomes 2 variable K-map.

4 Variable K-Map

The number of cells in 4 variable K-map is sixteen, since the number of variables is four. The following figure shows **4 variable K-Map**.



WX \ YZ	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

- There is only one possibility of grouping 16 adjacent minterm.
- Let R_1, R_2, R_3 and R_4 represents the min terms of first row, second row, third row and fourth row respectively. Similarly, C_1, C_2, C_3 and C_4 represents the min terms of first column, second column, third column and fourth column respectively. The possible combinations of grouping 8 adjacent min terms are {(R_1, R_2), (R_2, R_3), (R_3, R_4), (R_4, R_1), (C_1, C_2), (C_2, C_3), (C_3, C_4), (C_4, C_1)}.
- If w=0, then 4 variable K-map becomes 3 variable K-map.

Don't Care condition

- The "Don't Care" conditions allow us to replace the empty cell of a K-Map to form a grouping of the variables. While forming groups of cells, we can consider a "Don't Care" cell as either 1 or 0 or we can simply ignore that cell. Therefore, "Don't Care" condition can help us to form a larger group of cells.
- A Don't Care cell can be represented by a cross(X) or (d) in K-Maps representing a invalid combination.

UNIT-2: COMBINATIONAL LOGIC CIRCUITS

- The combinational logic circuits are the circuits that contain different types of logic gates.
- The output of the combinational circuit is determined from the present combination of inputs, regardless of the previous input.
- The input variables, logic gates, and output variables are the basic components of the combinational logic circuit.
- There are different types of combinational logic circuits, such as Adder, Subtractor, Decoder, Encoder, Multiplexer, and De-multiplexer.

Design procedure of Combinational circuits

- Find the required number of input variables and outputs from given specifications.
- Formulate the Truth table. If there are 'n' input variables, then there will be 2n possible combinations. For each combination of input, find the output values.
- Find the Boolean expressions for each output. If necessary, simplify those expressions.
- Implement the above Boolean expressions corresponding to each output by using Logic gates.

2.1 Half adder, Full adder, Half Subtractor, Full Subtractor, Serial and Parallel Binary 4 bit adder.

ADDER:

Adder circuit is a combinational digital circuit that is used for adding two numbers. A typical adder circuit produces a sum bit (denoted by S) and a carry bit (denoted by C) as the output. Adder circuits are of two types: Half adder and Full adder

HALF ADDER:

Half adder is a combinational arithmetic circuit that adds two single bit numbers and produces a sum and carry as the output.

There are two inputs named as A and B and the outputs are named as Sum (S) and Carry (C).

Truth Table

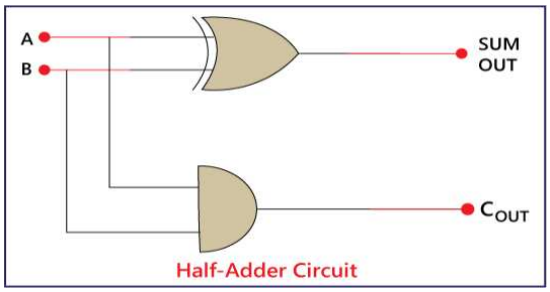
Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

From Truth table, the Boolean functions for each output as

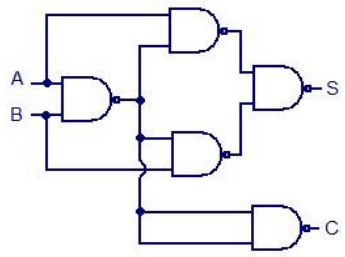
$S=A \oplus B$

$C=AB$

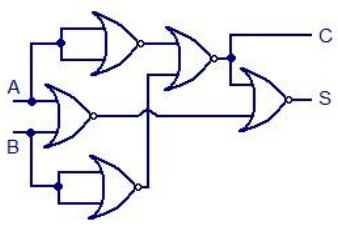
The logical diagram of Half adder is shown in the following figure.



NAND gates or NOR gates can be used for realizing the half adder in universal logic and the relevant circuit diagrams are shown in the figure below.



Half adder using NAND logic



Half adder using NOR logic

FULL ADDER

Full Adder is the adder which adds three single bit inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C_{in}. The output carry is designated as Carry and the normal output is designated as S which is SUM.

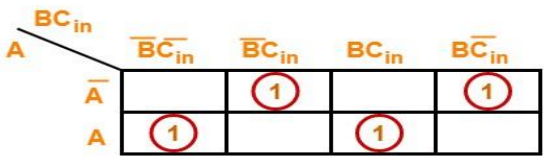
Block diagram

Truth Table

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

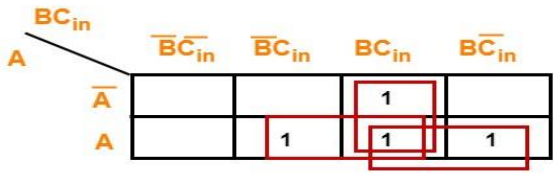
K-map for SUM and CARRY

For S:



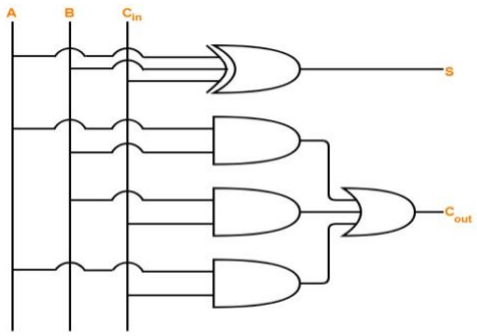
$$S = A \oplus B \oplus C_{in}$$

For C_{in}:



$$C_{out} = AB + BC_{in} + C_{in}A$$

Logical Diagram:



Full Adder Logic Diagram

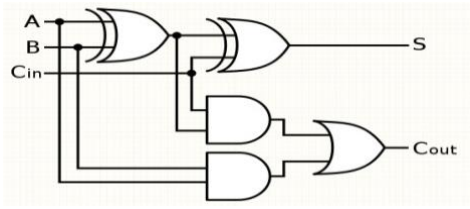
OR

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

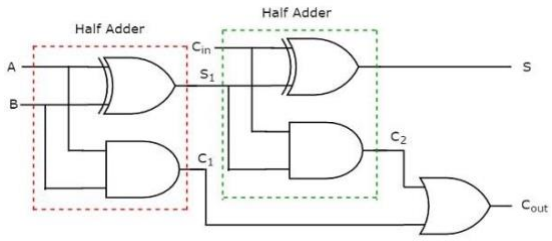
And

CARRY

$$C_o = \bar{A}BC_i + A\bar{B}C_i + AB\bar{C}_i + ABC_i$$
$$= C_i(\bar{A}B + A\bar{B}) + AB(\bar{C}_i + C_i)$$
$$= C_i(A \oplus B) + AB$$



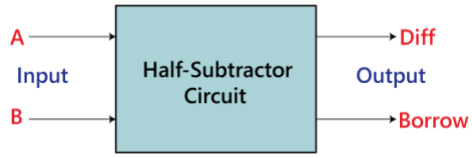
Full Addrer Using Half adder



HALF SUBTRACTOR

The half subtractor is a combinational circuit. It has two inputs and two outputs. This circuit is used to subtract two single bit binary numbers A and B. The **'difference**' and **'borrow'** are two output states of the half subtractor.

Block diagram



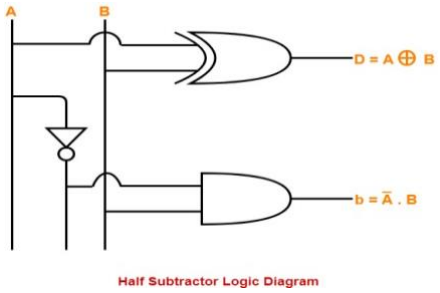
Truth Table

Inputs		Outputs	
A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

The SOP form of the **Diff** and **Borrow** is as follows:

Diff=A'B+AB'

Borrow = A'B

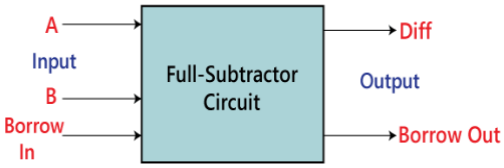


Half Subtractor Logic Diagram

FULL SUBTRACTOR

The full subtractor is used to subtract three 1-bit numbers A, B, and C, which are minuend, subtrahend, and borrow, respectively. The full subtractor has three input states and two output states i.e., diff and borrow.

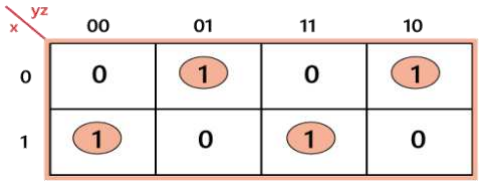
Block diagram



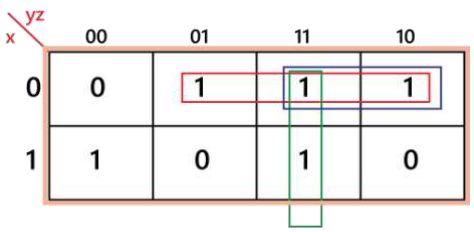
Truth Table

Inputs			Outputs	
A	B	$Borrow_{in}$	Diff	Borrow
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

The SOP form can be obtained with the help of K-map as:

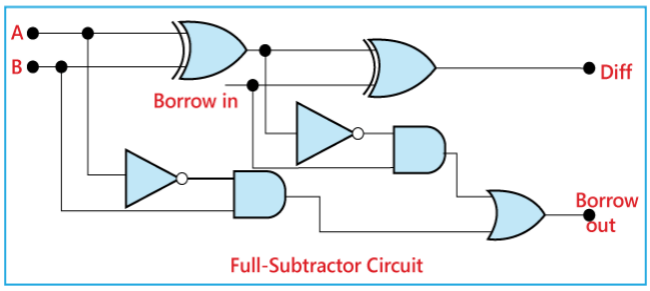


Diff=xy' z'+x' y' z+xyz+x'yz'

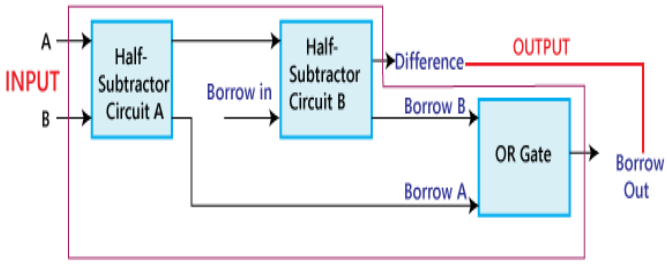


Borrow=x' z +x' y + yz

The full subtractor logic circuit diagram



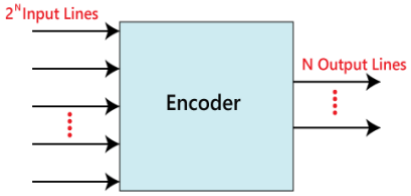
Construction of Full Subtractor Circuit using half subtractor:



2.2 Multiplexer (4:1), De- multiplexer (1:4), Decoder, Encoder, Digital comparator (3 Bit)

ENCODERS

The combinational circuits that change the binary information into N output lines are known as **Encoders**. The binary information is passed in the form of 2^N input lines. The output lines define the N-bit code for the binary information. At a time, only one input line is activated for simplicity.

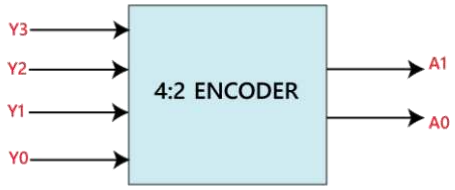


There are various types of encoders which are as follows:

4 to 2 line Encoder:

In 4 to 2 line encoder, there are total of four inputs, i.e., Y_0, Y_1, Y_2, and Y_3, and two outputs, i.e., A_0 and A_1. In 4input lines, one input-line is set to true at a time to get the respective binary code in the output side.

Block Diagram:



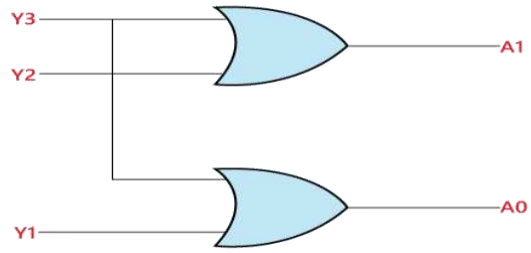
Truth Table:

INPUTS				OUTPUTS	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

The logical expression of the term A0 and A1 is as follows:

$A_1=Y_3+Y_2$
$A_0=Y_3+Y_1$

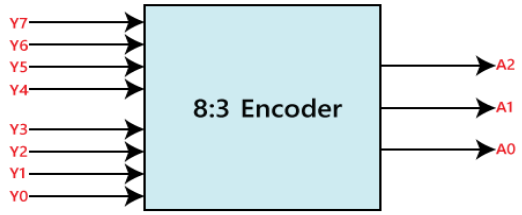
Logical circuit of the above expressions is given below:



8 to 3 line Encoder:

The 8 to 3 line Encoder is also known as **Octal to Binary Encoder**. In 8 to 3 line encoder, there is a total of eight inputs, i.e., Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, and Y_7 and three outputs, i.e., A_0, A1, and A_2. In 8-input lines, one input-line is set to true at a time to get the respective binary code in the output side.

Block Diagram:



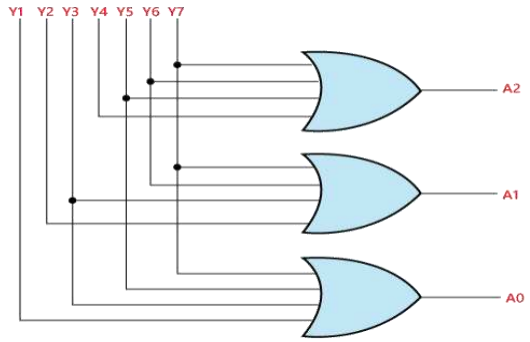
Truth Table:

INPUTS								OUTPUTS		
Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

The logical expression of the term A0, A1, and A2 are as follows:

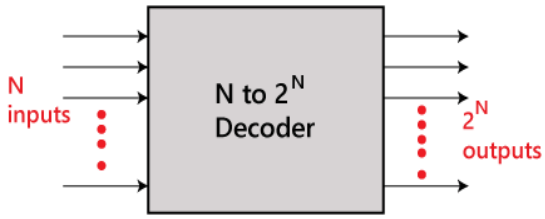
$A_2=Y_4+Y_5+Y_6+Y_7$
$A_1=Y_2+Y_3+Y_6+Y_7$
$A_0=Y_7+Y_5+Y_3+Y_1$

Logical circuit of the above expressions is given below:



DECODER

The combinational circuit that change the binary information into 2^N output lines is known as **Decoders.** The binary information is passed in the form of N input lines. The output lines define the 2^N-bit code for the binary information. In simple words, the **Decoder** performs the reverse operation of the **Encoder**. At a time, only one input line is activated for simplicity.

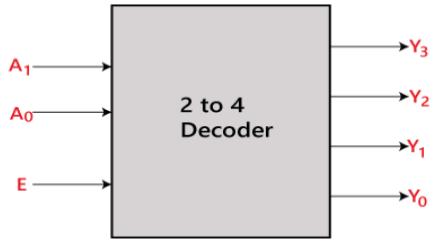


There are various types of decoders which are as follows:

2 to 4 line decoder:

In the 2 to 4 line decoder, there is a total of three inputs, i.e., A_0, and A_1 and E and four outputs, i.e., Y_0, Y_1, Y_2, and Y_3. For each combination of inputs, when the enable 'E' is set to 1, one of these four outputs will be 1.

Block Diagram:



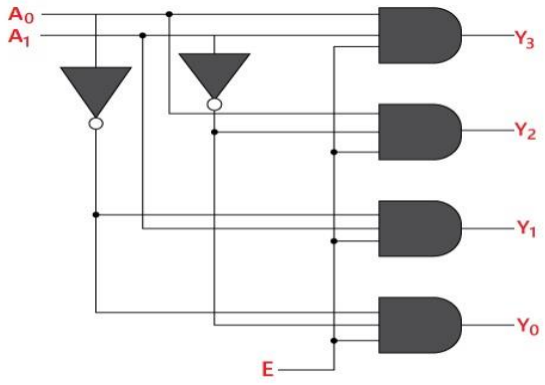
Truth Table:

Enable	INPUTS		OUTPUTS			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

The logical expression of the term Y0, Y0, Y2, and Y3 is as follows:

$Y_3=E.A_1.A_0$
$Y_2=E.A_1.A_0'$
$Y_1=E.A_1'.A_0$
$Y0=E.A_1'.A_0'$

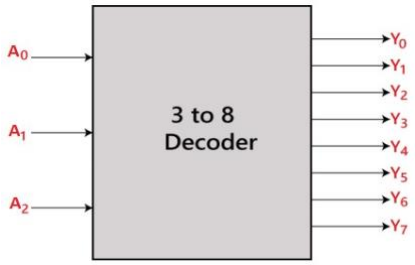
Logical circuit of the above expressions is given below:



3 to 8 line decoder:

The 3 to 8 line decoder is also known as **Binary to Octal Decoder**. In a 3 to 8 line decoder, there is a total of eight outputs, i.e., Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, and Y_7 and three outputs, i.e., A_0, A1, and A_2. This circuit has an enable input 'E'. Just like 2 to 4 line decoder, when enable 'E' is set to 1, one of these four outputs will be 1.

Block Diagram:



Truth Table:

Enable	INPUTS			Outputs							
E	A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

The logical expression of the term Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, and Y_7 is as follows:

$Y_0=A_0'.A_1'.A_2'$

$Y_1=A_0.A_1'.A_2'$

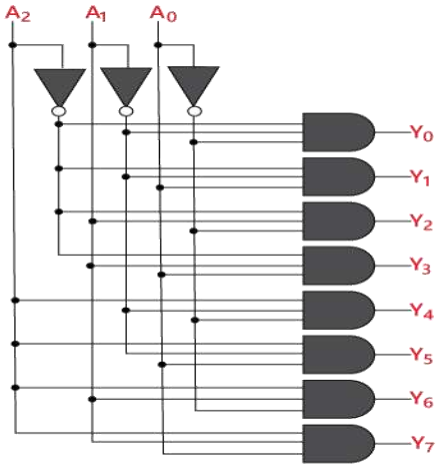
$Y_2=A_0'.A_1.A_2'$

$Y_3=A_0.A_1.A_2'$

$Y_4=A_0'.A_1'.A_2$

$Y_5=A_0.A_1'.A_2$

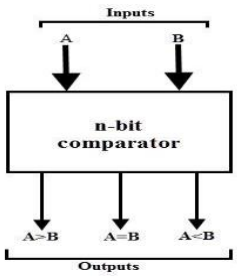
$Y_6=A_0'.A_1.A_2$ $Y_7=A_0.A_1.A_2$



Digital Comparator

A magnitude digital comparator is a combinational circuit that compares two digital or binary numbers (consider A and B) and determines their relative magnitudes in order to find out whether one number is equal, less than or greater than the other digital number.

Three binary variables are used to indicate the outcome of the comparison as A>B, A<B, or A=B. The below figure shows the block diagram of a n-bit comparator which compares the two numbers of n-bit length and generates their relation between themselves.



Types of Magnitude Comparators

There are different kinds of magnitude comparators which include the following.

1-bit Magnitude Comparator

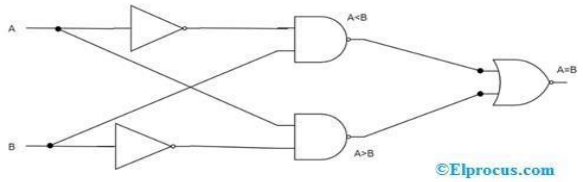
A comparator that compares two binary bits and produces three outputs based on the relative magnitudes of given binary bits is called a 1-bit magnitude comparator. Truth Table

The truth table for a 1-bit magnitude comparator is given below.

Inputs		*Outputs*		
A	*B*	*A > B*	*A = B*	*A < B*
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

The truth table derives the expressions of A<B, A>B and A=B as below

A<B – A′B
A>B – AB′
A=B – A′B′+AB
With these expressions, the Circuit diagram can be as follows



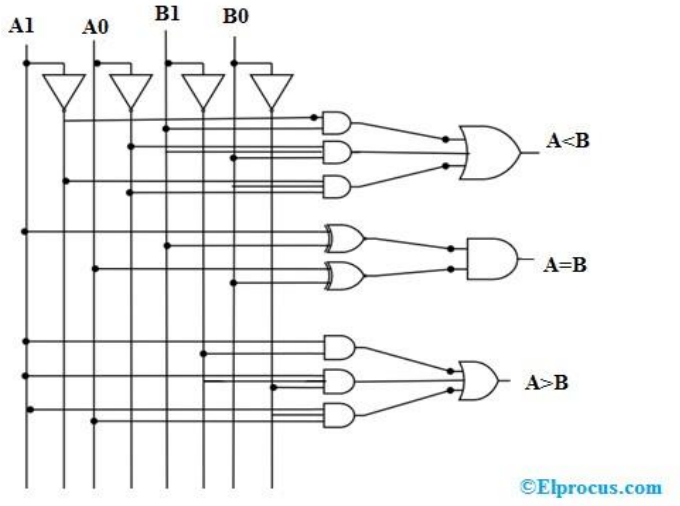
2-bit Magnitude Comparator

A comparator that compares two binary numbers (each number having 2 bits) and produces three outputs based on the relative magnitudes of given binary bits is called a 2-bit magnitude comparator. Truth Table

Inputs				Outputs		
A_1	A_0	B_1	B_0	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

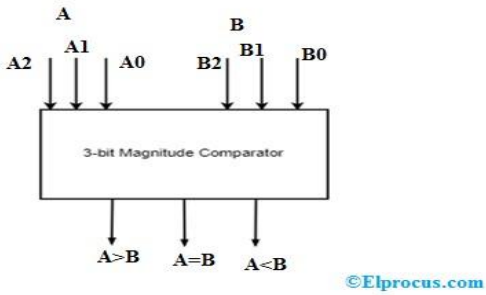
The truth table derives the expressions of A<B, A>B, and A=B as below
$A<B – A_1'B_1' + A_0'B_1B_0 + A_1'A_0'B_0$
$A>B – A_1B_1' + A_0B_1'B_0' + A_1A_0B_0'$

A=B – (A_0 Ex-Nor B_0) (A_1 Ex-Nor B_1)
With these expressions, the Circuit diagram can be as follows



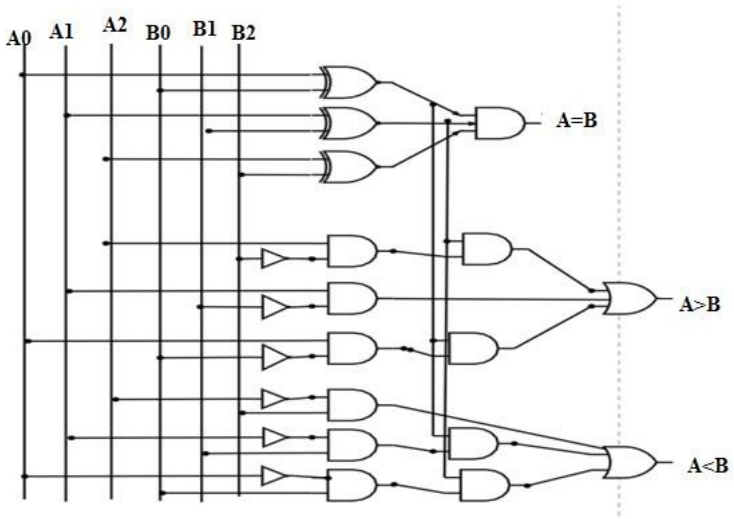
3-bit Magnitude Comparator

A comparator that compares two binary numbers (each number having 3 bits) and produces three outputs based on the relative magnitudes of given binary bits is called a 3-bit magnitude comparator.



The equal functions are $A_0 = B_0$, $A_1 = B_1$, $A_2 = B_2$
Then A=B
$= (A_0'B_0' + A_0B_0)(A_1'B_1' + A_1B_1)(A_2'B_2' + A_2B_2)$

The output is A< B in the cases of
$A_2<B_2$
$A_2 = B_2$ then $A_1<B_1$
$A_2 = B_2$, $A_1 = B_1$ then $A_0<B_0$
$A<B = A_2'B_2 + [(A_2'B_2' + A_2B_2) \times A_1'B_1] + [(A_2'B_2' + A_2B_2) \times [(A_1'B' + A_1B_1) \times A_0'B_0]$ The output is A> B in the cases of
$A_2>B_2$
$A_2 = B_2$ then $A_1>B_1$
$A_2 = B_2$, $A_1 = B_1$ then $A_0>B_0$
$A>B = A_2B_2' + + [(A_2'B_2' + A_2B_2) \times A_1B_1'] + [(A_2'B_2' + A_2B_2) \times [(A_1'B' + A_1B_1) \times A_0B_0']$



MULTIPLEXER

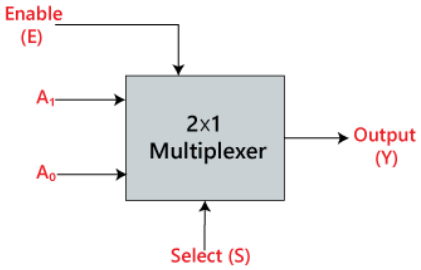
Multiplexer is a combinational circuit that has maximum of 2^n data inputs, 'n' selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines.

Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination will select only one data input. Multiplexer is also called as **Mux** There are various types of the multiplexer which are as follows:

2×1 Multiplexer:

In 2×1 multiplexer, there are only two inputs, i.e., A_0 and A_1, 1 selection line, i.e., S_0 and single outputs, i.e., Y. On the basis of the combination of inputs which are present at the selection line S, one of these 2 inputs will be connected to the output.

Block Diagram:



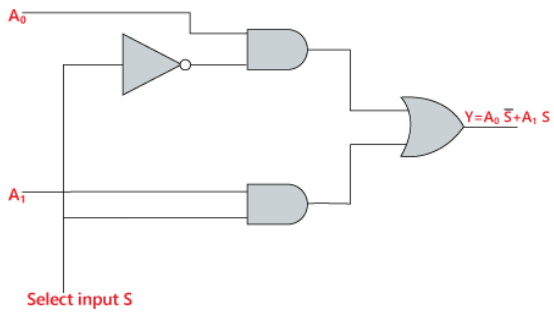
Truth Table:

INPUTS	Output
S_0	Y
0	A_0
1	A_1

The logical expression of the term Y is as follows:

$Y=S_0'.A_0+S_0.A_1$

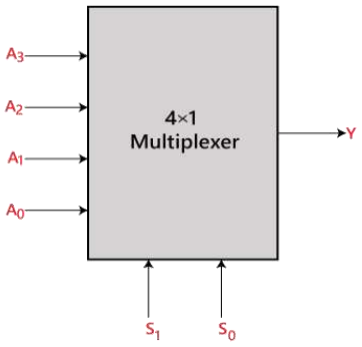
Logical circuit of the above expression is given below:



4×1 Multiplexer:

In the 4×1 multiplexer, there is a total of four inputs, i.e., A_0, A_1, A_2, and A_3, 2 selection lines, i.e., S_0 and S_1 and single output, i.e., Y. On the basis of the combination of inputs that are present at the selection lines S^0 and S_1, one of these 4 inputs are connected to the output.

Block Diagram:



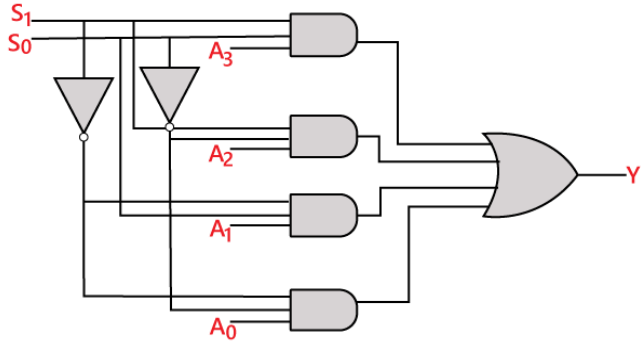
Truth Table:

INPUTS		Output
S_1	S_0	Y
0	0	A_0
0	1	A_1
1	0	A_2
1	1	A_3

The logical expression of the term Y is as follows:

$Y=S_1' S_0' A_0+S_1' S_0 A_1+S_1 S_0' A_2+S_1 S_0 A_3$

Logical circuit of the above expression is given below:



DE-MULTIPLEXER

De-Multiplexer is a combinational circuit that performs the reverse operation of Multiplexer. It has single input, 'n' selection lines and maximum of 2^n outputs. The input will be connected to one of these outputs based on the values of selection lines.

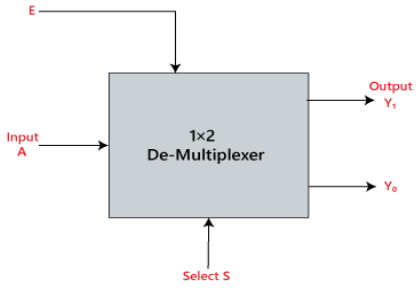
Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination can select only one output. De-Multiplexer is also called as De-Mux.

There are various types of De-multiplexer which are as follows:

1×2 De-multiplexer:

In the 1 to 2 De-multiplexer, there are only two outputs, i.e., Y_0, and Y_1, 1 selection lines, i.e., S_0, and single input, i.e., A. On the basis of the selection value, the input will be connected to one of the outputs.

Block Diagram:

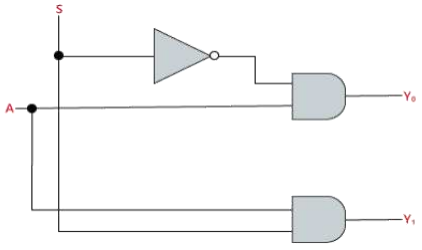


Truth Table:

INPUTS	Output	
S_0	Y_1	Y_0
0	0	A
1	A	0

The logical expression of the term Y is as follows:
$Y_0=S_0'.A$ $Y_1=S_0.A$

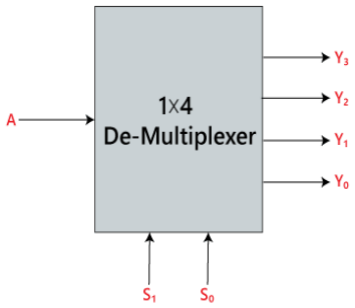
Logical circuit of the above expressions is given below:



1×4 De-multiplexer:

In 1 to 4 De-multiplexer, there are total of four outputs, i.e., Y_0, Y_1, Y_2, and Y_3, 2 selection lines, i.e., S_0 and S_1 and single input, i.e., A. On the basis of the combination of inputs which are present at the selection lines S_0 and S_1, the input be connected to one of the outputs.

Block Diagram:



Truth Table:

INPUTS		Output			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	A
0	1	0	0	A	0
1	0	0	A	0	0
1	1	A	0	0	0

The logical expression of the term Y is as follows:

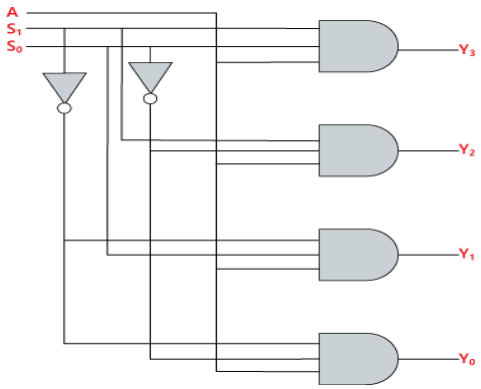
$Y_0 = S_1' S_0' A$

$y_1 = S_1' S_0 A$

$y_2 = S_1 S_0' A$

$y_3 = S_1 S_0 A$

Logical circuit of the above expressions is given below:



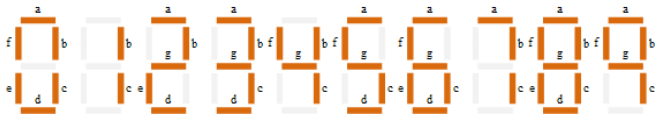
2.3 Seven segment Decoder (Definition, relevance, gate level of circuit Logic circuit, truth table, Applications of above)

Seven segment displays are the output display device that provides a way to display information in the form of image or text or decimal numbers.

One of the most commonly used decoder drive is a BCD to 7-segment decoder. In this decoder the outputs are used to drive seven segments.

It is widely used in digital clocks, basic calculators, electronic meters, and other electronic devices that display numerical information.

BCD to seven segment decoder has four input lines A, B, C and D and 7 output lines a, b, c, d, e, f and g. As the numbers which are greater than 9 are not a permitted combination it is assumed that they will not occur.



Digit	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0
10	1	0	1	0	×	×	×	×	×	×	×
11	1	0	1	1	×	×	×	×	×	×	×
12	1	1	0	0	×	×	×	×	×	×	×
13	1	1	0	1	×	×	×	×	×	×	×
14	1	1	1	0	×	×	×	×	×	×	×
15	1	1	1	1	×	×	×	×	×	×	×

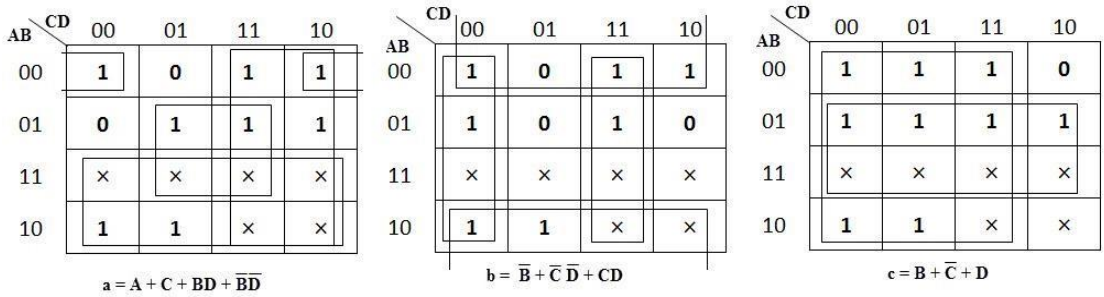
From the above truth table, the Boolean expressions of each output functions can be written as

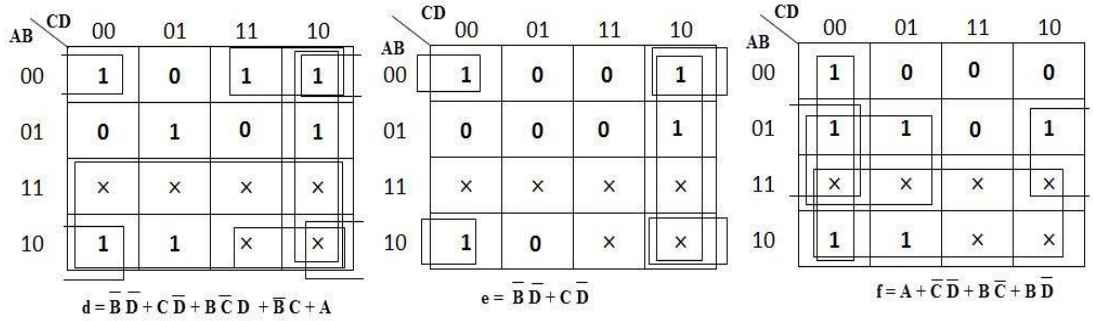
$a = F_1 (A, B, C, D) = \sum m (0, 2, 3, 5, 7, 8, 9)$ $b = F_2 (A, B, C, D) = \sum m (0, 1, 2, 3, 4, 7, 8, 9)$ $c = F_3 (A, B, C, D) = \sum m (0, 1, 3, 4, 5, 6, 7, 8, 9)$ $d = F_4 (A, B, C, D) = \sum m (0, 2, 3, 5, 6, 8)$ $e = F_5 (A, B, C, D) = \sum m (0, 2, 6, 8)$

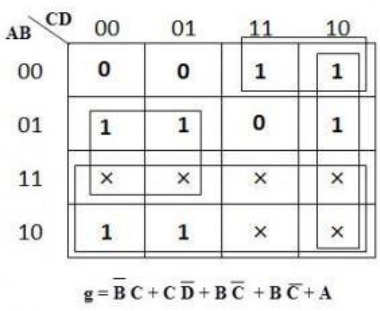
f = F_6 (A, B, C, D) = ∑m (0, 4, 5, 6, 8, 9) g =

F7 (A, B, C, D) = ∑m (2, 3, 4, 5, 6, 8, 9)

K-Map Simplification







From the above simplification, we get the output values as

a = A + C + BD + B̄ D̄

b = B̄ + C̄ D̄ + CD

c = B + C̄ + D

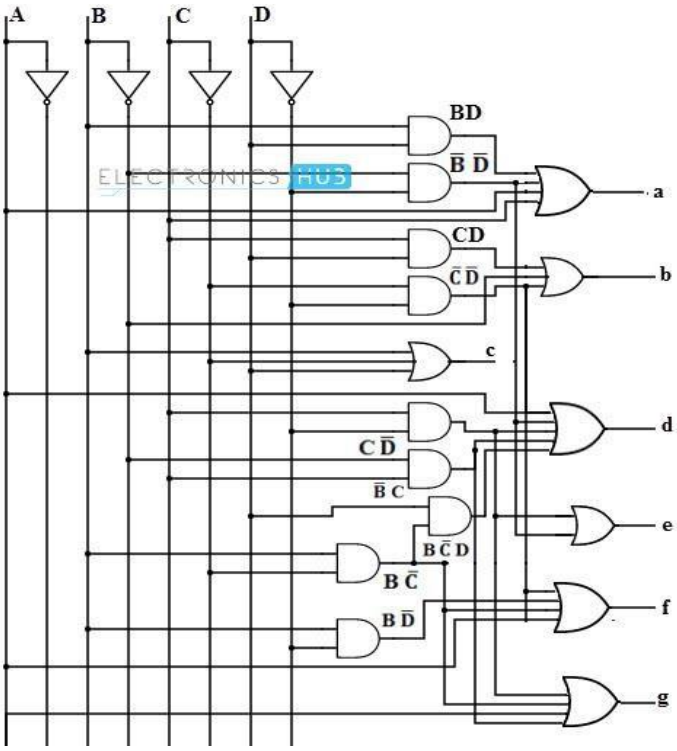
d = B̄ D̄ + C D̄ + B C̄ D + B̄ C + A

e = B̄ D̄ + C D̄

f = A + C̄ D̄ + B C̄ + B D̄

g = A + B C̄ + B̄ C + C D̄

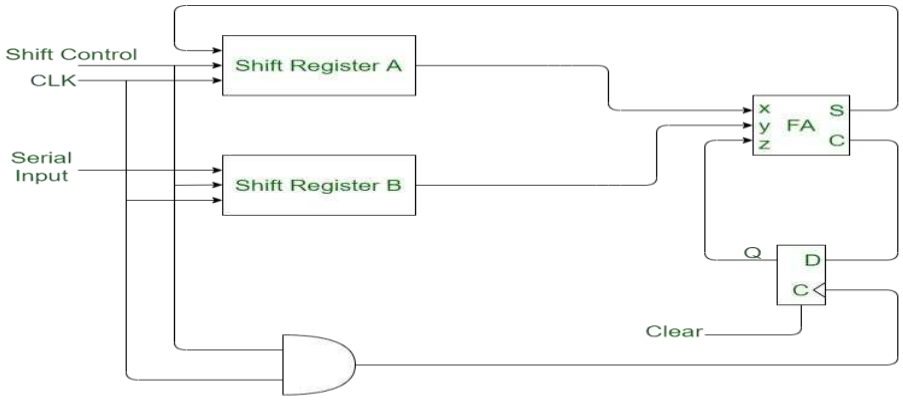
The logic circuit for each output signal of 7- segment display as follow



SERIAL BINARY ADDER

Serial binary adder is a combinational logic circuit that performs the addition of two binary numbers in serial form. Serial binary adder performs bit by bit addition. Two shift registers are used to store the binary numbers that are to be added.

A single full adder is used to add one pair of bits at a time along with the carry. The carry output from the full adder is applied to a D flip-flop. After that output is used as carry for next significant bits. The sum bit from the output of the full adder can be transferred into a third shift register.



Block Diagram of Serial Binary Adder

Shift Registers :

Shift Register is a group of flip flops used to store multiple bits of data. There are two shift registers used in the serial binary adder. In one shift register augend is stored and in other shift register addend is stored.

Full Adder :

Full adder is the combinational circuit which takes three inputs and gives two outputs as sum and carry.

The circuit adds one pair at a time with the help of it. **D Flip-flop :**

The carry output from the full adder is applied on the D flip-flop. Further, the output of D flip-flop is used as a carry input for the next pair of significant bits.

Working Process:

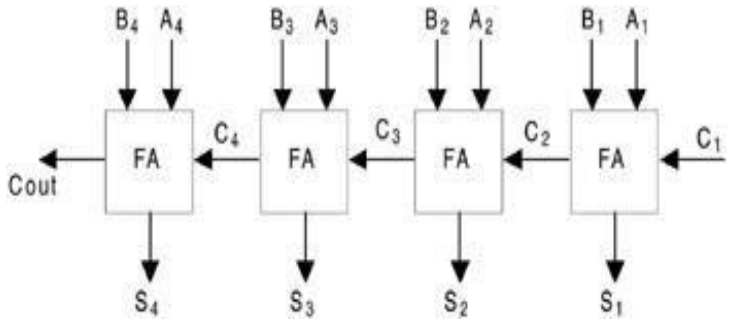
Following is the procedure of addition using serial binary adder:

- **Step-1:**
  The two shift registers A and B are used to store the numbers to be added.
- **Step-2:**
  A single full adder is used to add one pair of bits at a time along with the carry.
- **Step-3:**
  The contents of the shift registers shift from left to right and their output starting from a and b are fed into a single full adder along with the output of the carry flip-flop upon application of each clock pulse.
- **Step-4:**
  The sum output of the full adder is fed to the most significant bit of the sum register.
- **Step-5:**
  The content of sum register is also shifted to right when clock pulse is applied.
- **Step-6:**
  After applying four clock pulse the addition of two registers (A & B) contents are stored in sum register.

PARALLEL BINARY ADDER

A digital circuit that is used to perform the addition of two binary numbers & an i/p carry, where the length of one bit is larger than another bit and operates in parallel with equivalent pairs of bits.
The arrangement of parallel adder can be done by arranging the full adders (FAs) in a chain model where the carry o/p from every full adder (FA1) can be linked to the carry i/p of the next full adder (FA2) within the chain.

Two 4-bit binary numbers $B_4B_3B_2B_1$ and $A_4A_3A_2A_1$ are to be added with a carry input C_1. This can be done by cascading four full adder circuits as shown in Figure. The least significant bits A_1, B_1, and C_1 are added to the produce sum output S_1 and carry output C_2. Carry output C_2 is then added to the next significant bits A_2 and B_2 producing sum output S_2 and carry output C_3. C_3 is then added to A_3 and B_3 and so on. Thus finally producing the four-bit sum output $S_4S_3S_2S_1$ and final carry output Cout. Such type of four-bit binary adder is commercially available in an IC package.



UNIT-3: SEQUENTIAL LOGIC CIRCUITS

3.1 Principle of flip-flops operation, its Types

- A flip flop is an electronic circuit with two stable states that can be used to store binary data. The stored data can be changed by applying varying inputs.
- Flip flop is formed using logic gates such as two NAND and NOR gates which are in turn made of transistors.
- As flip flops have two stable states, hence they are called as Bistable multivibrators. The two stable states are High (logic 1) and Low (logic 0).
- Flip-flops and latches are fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems. Flip-flops and latches are used as data storage elements.

Latches vs Flip-Flops

- Latches and flip flops are both 1bit binary data storage devices.
- The main difference between a latch and a flip flop is the triggering mechanism. Latches are transparent when enabled, whereas flip flops are dependent on the transition of the clock signal i.e. either positive edge or negative edge.

LATCH	FLIP – FLOP
Latches do not require clock signal.	Flip – flops have clock signals
A latch is an asynchronous device.	A flip – flop is a synchronous device.
Latches are transparent devices i.e. when they are enabled, the output changes immediately if the input changes.	A transition from low to high or high to low of the clock signal will cause the flip – flop to either change its output or retain it depending on the input signal.
A latch is a Level Sensitive device (Level Triggering is involved).	A flip – flop is an edge sensitive device (Edge Triggering is involved).
Latches are simpler to design as there is no clock signal (no careful routing of clock signal is required).	When compare to latches, flip – flops are more complex to design as they have clock signal and it has to be carefully routed. This is because all the flip – flops in a design should have a clock signal and the delay in the clock reaching each flip – flop must be minimum or negligible.
The operation of a latch is faster as they do not have to wait for any clock signal.	Flip - flops are comparatively slower than latches due to clock signal.
The power requirement of a latch is less.	Power requirement of a flip – flop is more.
A latch works based on the enable signal.	A flip – flop works based on the clock signal.

Types of flip flops

Based on their operations, flip flops are basically 4 types. They are

i. S-R flip flop
ii. D flip flop
iii. J-K flip flop iv. T flip flop

3.2 SR Flip Flop using NAND, NOR Latch (un clocked)

The **SR flip-flop** also known as a *SR Latch*. It has two inputs, one is “SET” labelled **S** and other is “RESET” labelled **R**.

The reset input resets the flip-flop back to its original state with an output Q that will be either at a logic level “1” or logic “0” depending upon this set/reset condition.

Then the SR flip-flop actually has three inputs, Set, Reset and its current output Q relating to its current state or history. The term “Flip-flop” relates to the actual operation of the device, as it can be “flipped” into one logic Set state or “flopped” back into the opposing logic Reset state.

SR Flip Flop using NAND:

The basic single bit set-reset SR flip-flop is to connect together a pair of cross-coupled 2-input NAND gates, there is feedback from each output to one of the other NAND gate inputs. It consists of two inputs, one called the *Set*, S and the other called the *Reset*, R with two corresponding outputs Q and its inverse or complement $\bar{Q}$.

Operation

- When S=0 and R= 0, then both the outputs of gate G1 and G2 try to become 1, which is not possible. There is a race between the outputs.

- When S=0 and R= 1, then the output of G2 is 1 and that of G1 is 0 (Q=0) and the memory is said to be reset.
- When S=1 and R= 0, then the output of G2 is 0 and that of G1 is 1 (Q=1) and the memory is said to be set.
- When S=1 and R= 1, then the output remains unchanged.

According to inputs S and R, the data logic 0 or logic 1 can be loaded or stored into the logic circuit.

SR Flip Flop using NOR:

The basic single bit set-reset SR flip-flop is to connect together a pair of cross-coupled 2-input NOR gates, there is feedback from each output to one of the other NOR gate inputs. It consists of two inputs, one called the *Set*, S and the other called the *Reset*, R with two corresponding outputs Q and its inverse or complement $\bar{Q}$.

Operation

- When S=0 and R= 0, the output remains unchanged.
- When S=0 and R= 1, then the output of G1is 0 (Q=0) and that of G2 is 1 and the memory is reset.
- When S=1 and R= 0, then the output of G1 is 1 (Q=1) and that of G2 is 0 and the memory is set.
- When S=1 and R= 1, then the outputs of both the gates try to become 0, which is not possible, and there is a race between the outputs.

3.3 C l o c k e d SR, D, JK, T, JK Master Slave flip-flops-Symbol, logic Circuit, truth table and applications

Clocked SR flip-flop

A clocked SR flip-flop is designed by adding two NAND gates to a basic NAND gate un-clocked flip-flop. These two NAND gates are used to apply clock pulse. The simple S-R flip flop will be set or reset any time by changing the inputs S and R.

Operation:

When the clock is absent (CLK=0), the outputs of both the gates G3 and G4 are 1 and there is no change in the outputs of G1 and G2. When the clock is present (CLK=1), the outputs of G3 and G4 are the function of the inputs S and R.

- When S=0 and R=0, then the outputs of G3 and G4 are 1, the outputs of G1 and G2 are unchanged.
- When S=0 and R=1, then the output of G3 is 1 and that of G4 is 0, the output of G1 is 0 and that of G2 is 1. The flip-flop is reset.
- When S=1 and R=0, then the output of G3 is 0 and that of G4 is 1, the output of G1 is 1 and that of G2 is 0. The flip-flop is set.
- When S=1 and R=1, then the outputs of G3 and G4 are 0, and the outputs of G1 and G2 try to become1, which is not possible. There is a race and the output is undefined and it is known as forbidden state.

S_n and R_n are the present inputs, Q_n is the present output, and Q_{n+1} is the output after the clock.

Case 1: For S=0, R=0 and CLK=0, the flip-flop simply remains in its present state. That is, Q remains unchanged. Even for S=0, R=0 and CLK=1, the flip-flop remains in its present state. This condition will not affect the outputs of flip-flop.

Case 2: For S=0, R=1 and CLK=0, the flip-flop remains in its present state. But, when CLK=1, the NAND gate-1 output will go to 1 and the NAND gate 2 output will go to 0. Now a 0 NAND gate 4 input forces $\bar{Q}$= 1 which in turn results in NAND gate 3 output Q=0. Thus, for S=0, R=1 and CLK=1, the flip-flop RESET to the 0 state.

Case 3: For S=1, R=0 and CLK=0, the flip-flop remains in its present state. But for S=1, R=0 and CLK=1, the set state of the flip-flop is reached. This causes the NAND gate 1 output to go 0 and the NAND gate 2 output to 1. Now, a 0 at NAND gate 3 input forces Q to 1 which in turn forces NAND gate 4 output $\bar{Q}$ to 0.

Case 4: An indeterminate condition occurs when all the inputs, namely CLK, S and R are equal to 1. This condition results in 0's in the outputs of gate 1 and 2 and 1's in both outputs Q and $\bar{Q}$. When the CLK input goes back to 0 (while S and R remain at 1), it is not possible to determine the next state, as it depends on whether the output of gate 1 or gate 2 goes to 1 first.

D FLIP-FLOP

The D flip-flop has only one input called the Delay (D) input and outputs Q and $\bar{Q}$. It can be constructed from an S-R flip-flop by inserting an inverter between S and R and assigning the symbol D to the S input.

Operation:

- When the CLK input is LOW, the D input has no effect, since the set and reset inputs of the NAND flipflop are kept HIGH.
- When the CLK goes HIGH, the Q output will take on the value of the D input.
  If CLK=1 and D=1, the NAND gate-1 output goes 0 and output of NAND gate goes 1. The basic NAND SR flip-flop output will be 1, i.e it follows D input. Similarly, for CLK =1 and D =0, the flip-flop output will be 0. If D changes while the CLK is HIGH, Q will follow and change quickly.

It is clear that the next state of the flip-flop at time Q_{n+1} follows the value of the input D when the clock pulse is applied. As transfer of data from the input to the output is delayed, it is known as delay(D) flip-flop.

The delay flip-flop is either used as a delay or as a latch to store 1 bit of binary information.

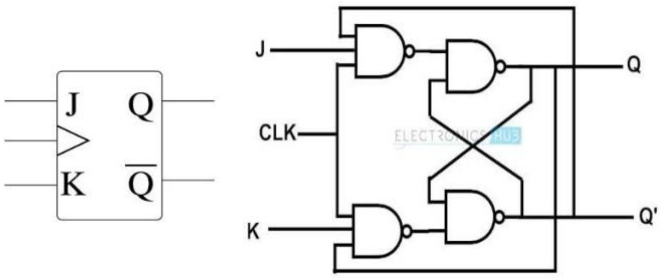
JK FLIP-FLOP:

- JK flip flop is a modified version of SR flip flop. Logic diagram consists of three input NAND gates replacing the two input NAND gates in SR flip flop and the inputs are replaced with J and K from S and R.
- The design of the JK flip flop is such that the three inputs to one NAND gate are J, clock signal along with a feedback signal from Q' and the three inputs to the other NAND are K, clock signal along with a feedback signal from Q. This arrangement eliminates the indeterminate state in SR f lip flop.

Operation

- When both the inputs J and K are LOW, then Q returns its previous state value i.e. it holds the previous data.

When we apply a clock pulse to the J K flip flop and the J input is low then irrespective of the other NAND gates, the NAND gate-1 output becomes HIGH. In the same manner, if the K input is low then output of NAND gate-2 is also HIGH. So thus the output remains in the same state i.e. no change in the state of flip flop.



J	K	$Q_{(t+1)}$	
0	0	$Q_{(t)}$	*unchanged*
0	1	0	*reset*
1	0	1	*set*
1	1	$\overline{Q}_{(t)}$	*output inversion*

- When J is LOW and K is HIGH, then flip flop will be in Reset state i.e. Q = 0, Q' = 1.

When we apply a clock pulse to the J K flip flop and the inputs are J is low and K is high the output of the NAND gate connected to J input becomes 1. Then Q becomes 0. This will reset the flip flop again to its previous state. So the Flip flop will be in RESET state.

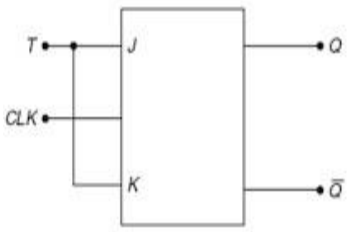
- When J is HIGH and K is LOW, then flip – flop will be in Set state i.e. Q = 1, Q' = 0

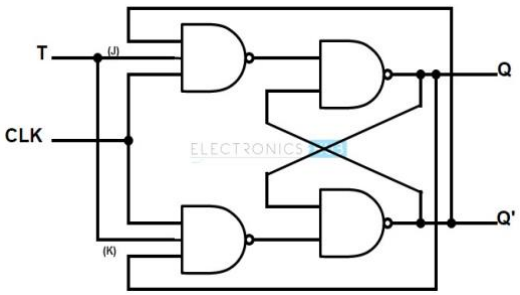
When we apply a clock pulse to the J K flip flop and the inputs are J is high and K is low the output of the NAND gate connected to K input becomes 1. Then Q' becomes 0. This will set the flip flop with the high clock input. So the Flip flop will be in SET state.

- When both the inputs J and K are HIGH, then flip – flop is in Toggle state. This means that the output will complement of the previous state.

T FLIP-FLOP:

A T flip flop is basically a single input version of JK flip flop. This modified form of JK flip-flop is obtained by connecting both inputs J and K together. This flip-flop has only one input along with the clock input.
These flip-flops are called T flip-flops because of their ability to complement its state (i.e.) Toggle, hence the name Toggle flip-flop.





T	Q_{n+1}
0	Q_n
1	$\overline{Q_n}$

- Here J and K input of the JK Flip-Flop is connected together and given to the T input.
- If the T input is in 0 state (i.e., J = K = 0) prior to a clock pulse, the Q output will not change with the clock pulse.
- If the T input is in 1 state (i.e., J = K = 1) prior to a clock pulse, the Q output will change to Q' with the clock pulse.
- We can conclude that if T = 1 and the device is clocked, then the output toggles its state.

3.4 Concept of Racing and how it can be avoided

When J=K=1 and if Q=0, a clock pulse of width t_p is applied, the output will change from 0 to 1 after a time interval Δt, where Δt is the propagation delay through two NAND gates in series. Now after Δt, we have J=K=1

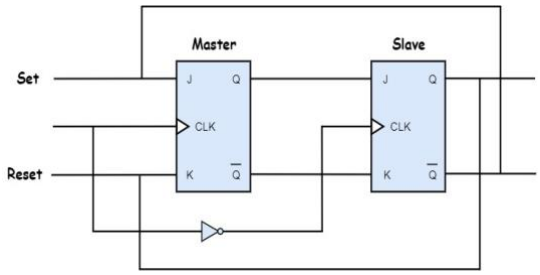
and Q=1 and after, another interval of Δt, output Q will become 0. Hence the output will oscillate back and forth between 0 and 1 in the duration t_p of the clock pulse width. So at the end of the clock pulse, the value of Q is ambiguous. This situation is known as Race- Around condition.

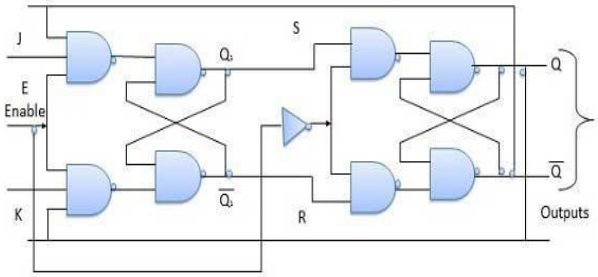
The race around condition can be avoided if t_p<Δt. Lumped delay lines can be used in series with the feedback connections in order to increase the loop delay beyond and hence to prevent the race around difficulty. The race around condition can also be avoided in Master- Slave flip-flop.

MASTER SLAVE JK FLIP FLOP:

- The Master-Slave Flip-Flop is basically a combination of two JK flip-flops connected together in a series configuration.
- The first flip-flop is called the *master*, and it is driven by the positive clock cycle. The second flip-flop is called the *slave*, and it is driven by the negative clock cycle.
- During the positive clock cycle, master flip-flop gives the intermediate output but slave flip-flop will not give the final output.
- During the negative clock cycle, slave flip-flop gets activated and copies the previous output of the master flip-flop and produces the final output.

Circuit Diagram





Truth Table

Inputs			Outputs		Comments
E	J	K	Q_{n+1}	$\overline{Q}_{n+1}$	
1	0	0	Q_n	$\overline{Q}_n$	No change
1	0	1	0	1	Rset
1	1	0	1	0	Set
1	1	1	$\overline{Q}_n$	Q_n	Toggle

Working of a master slave flip flop J = K = 0 (No change)

- When clock = 0, the slave becomes active and master is inactive. But since the S and R inputs have not changed, the slave outputs will also remain unchanged. Therefore, outputs will not change if J = K =0.

J = 0 and K = 1 (Reset)

- Clock = 1 − Master active, slave inactive. Therefore, outputs of the master become Q_1 = 0 and Q_1 bar = 1. That means S = 0 and R =1.
- Clock = 0 − Slave active, master inactive. Therefore, outputs of the slave become Q = 0 and Q bar = 1.
- Again clock = 1 − Master active, slave inactive. Therefore, even with the changed outputs Q = 0 and Q bar = 1 fed back to master, its output will be Q1 = 0 and Q1 bar = 1. That means S = 0 and R = 1.

- Hence with clock = 0 and slave becoming active the outputs of slave will remain Q = 0 and Q bar = 1. Thus we get a stable output from the Master slave. **J = 1 and K = 0 (Set)**
- Clock = 1 − Master active, slave inactive. Therefore, outputs of the master become Q_1 = 1 and Q_1 bar = 0. That means S = 1 and R =0.
- Clock = 0 − Slave active, master inactive. Therefore, outputs of the slave become Q = 1 and Q bar = 0.
- Again clock = 1 − then it can be shown that the outputs of the slave are stabilized to Q = 1 and Q bar = 0.

J = K = 1 (Toggle)

- Clock = 1 − Master active, slave inactive. Outputs of master will toggle. So S and R also will be inverted.
- Clock = 0 − Slave active, master inactive. Outputs of slave will toggle.

These changed output are returned back to the master inputs. But since clock = 0, the master is still inactive. So it does not respond to these changed outputs. This avoids the multiple toggling which leads to the race around condition. The master slave flip flop will avoid the race around condition.

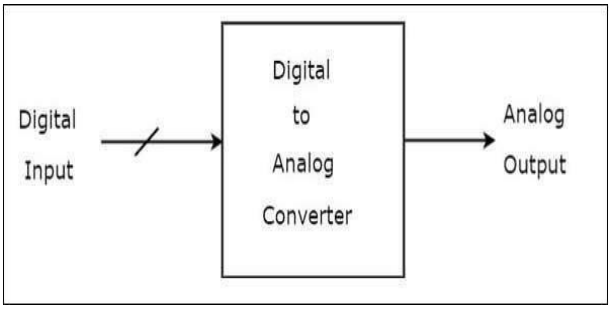
UNIT-5: A/D AND D/A CONVERTERS

5.1 Necessity of A/D and D/A converters.

- To convert the data from one form into another form
-

******To convert one form of signal with other

A Digital to Analog Converter (DAC) converts a digital input signal into an analog output signal. The digital signal is represented with a binary code, which is a combination of bits 0 and 1.

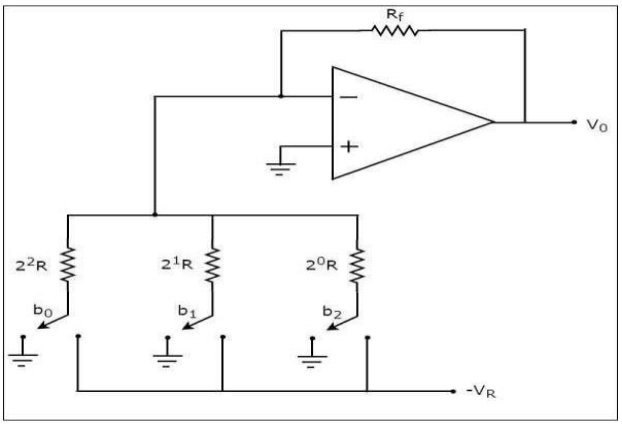


There are two types of DACs

- Weighted Resistor DAC
- R-2R Ladder DAC

5.2 D/A conversion using weighted resistors methods.

A weighted resistor DAC produces an analog output, which is almost equal to the digital (binary) input by using binary weighted resistors in the inverting adder circuit.

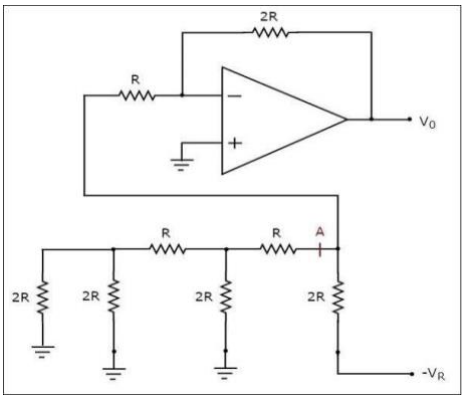


- It uses binary weighted resistor 2^1R, 2^2R……2^nR and summing amplifier.
- The MSB input is connected with lowest resistor and towards LSB the resistance value is made twice of previous resistor.
- The purpose of increasing the resistor value is to pass minimum current through the LSB resistance while maximum current through MSB resistance.
- Here n number of electronic switches are used to provide digital input.
  When switch is connected to ($-V_R$), b1=1
  When switch is connected to ground b_1 =0

- Output voltage= $V_R\left(\frac{b1}{2}+\frac{b2}{2^2}+\ldots\ldots\frac{bN}{2^N}\right)$
- Example:

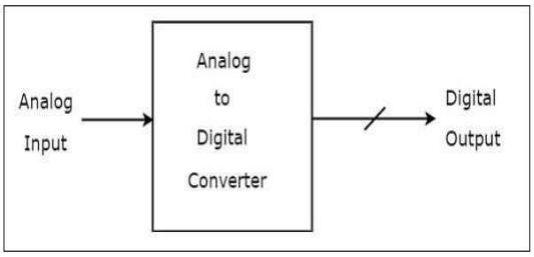
5.3 D/A conversion using R-2R ladder (Weighted resistors) network.

- R-2R ladder D/A converter produces an analog output, which is almost equal to the digital input by using R-2R ladder network in the inverting adder circuit.



- In R-2R ladder D/A converter, resistors of only two values i.e R and 2R are used.
- The inverting input terminal of the op-amp acts as summing junction for the ladder inputs. ▪ Output voltage= $V_R\left(\frac{bN}{2^N}+\ldots\ldots+\frac{b2}{2^2}+\frac{b1}{2}\right)$
- Example

********An Analog to Digital Converter **(ADC)** converts an analog signal into a digital signal. The digital signal is represented with a binary code, which is a combination of bits 0 and 1.



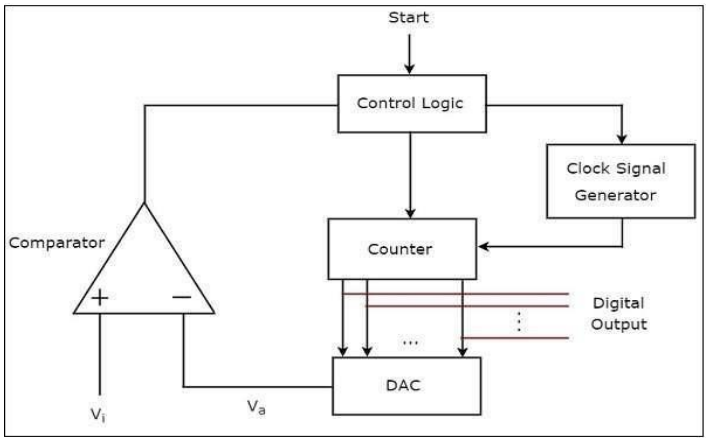
There are 3 types of ADCs

- Counter type ADC
- Successive Approximation ADC
- Flash type ADC

5.4 A/D conversion using counter method.

A **counter type ADC** produces a digital output, which is approximately equal to the analog input by using counter operation internally.

The **block diagram** of a counter type ADC is shown in the following figure –



The counter type ADC mainly consists of 5 blocks: Clock signal generator, Counter, DAC, Comparator and Control logic.

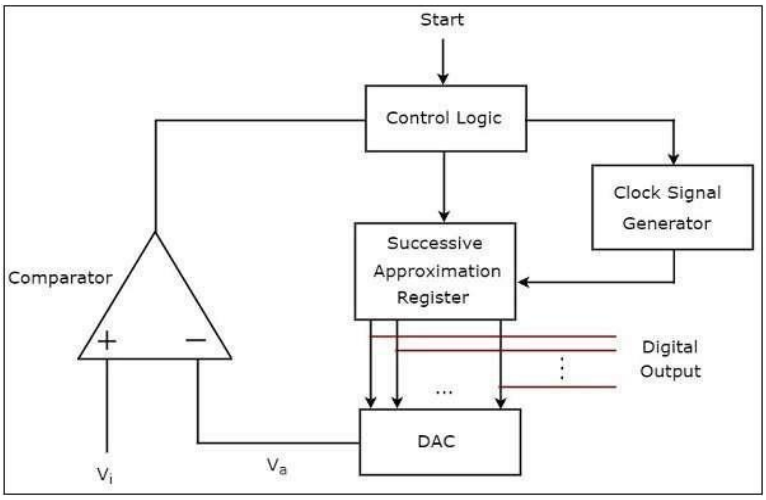
The **working** of a counter type ADC is as follows –

- The **control logic** resets the counter and enables the clock signal generator in order to send the clock pulses to the counter, when it received the start commanding signal.
- The **counter** gets incremented by one for every clock pulse and its value will be in binary (digital) format. This output of the counter is applied as an input of DAC.
- **DAC** converts the received binary (digital) input, which is the output of counter, into an analog output. Comparator compares this analog value, with the external analog input value.
- The **output of comparator** will be **'1'** as long as Vi is greater than. The operations mentioned in above two steps will be continued as long as the control logic receives '1' from the output of comparator.
- The **output of comparator** will be **'0'** when Va is less than or equal to Vi. So, the control logic receives '0' from the output of comparator. Then, the control logic disables the clock signal generator so that it doesn't send any clock pulse to the counter.
- At this instant, the output of the counter will be displayed as the **digital output**. It is almost equivalent to the corresponding external analog input value.

5.5 A/D conversion using Successive approximate method

A **successive approximation type ADC** produces a digital output, which is approximately equal to the analog input by using successive approximation technique internally.

The **block diagram** of a successive approximation ADC is shown in the following figure



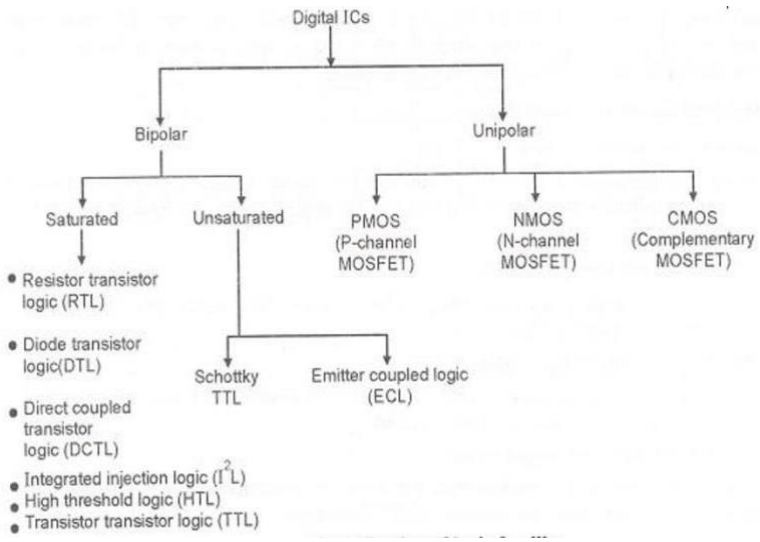
The successive approximation ADC mainly consists of 5 blocks− Clock signal generator, Successive Approximation Register (SAR), DAC, comparator and Control logic. The **working** of a successive approximation ADC is as follows −

- The **control logic** resets all the bits of SAR and enables the clock signal generator in order to send the clock pulses to SAR, when it received the start commanding signal.
- The binary (digital) data present in **SAR** will be updated for every clock pulse based on the output of comparator. The output of SAR is applied as an input of DAC.
- **DAC** converts the received digital input, which is the output of SAR, into an analog output. The comparator compares this analog value Va with the external analog input value Vi.
- The **output of a comparator** will be '1' as long as Vi is greater than Va. Similarly, the output of comparator will be '0', when Vi is less than or equal to Va .
- The operations mentioned in above steps will be continued until the digital output is a valid one.

UNIT-6: LOGIC FAMILIES

6.1 Various logic families &categories according to the IC fabrication process

- A logic family is a collection of different integrated circuit chips that have similar input, output and internal circuit characteristics but that perform different logic functions.
- The circuit design of the basic gate of each logic family is the same.
- The most important parameters for evaluating and comparing logical families include: logic levels, power dissipation, propagation delay, noise margin, Fan out



6.2 Characteristics of Digital ICs- Propagation Delay, fan-out, fan-in, Power Dissipation ,Noise Margin ,Power Supply requirement &Speed with Reference to logic families.

Propagation Delay

The time required to change the output from one logic state to another logic state after input is applied, is called the propagation delay of logic circuit. Its unit is nanoseconds.

fan-out

It is the no of the same gates that can be driven by a gate. High fan out is beneficial, as this reduces the requirement for additional drivers to drive more gates. fan-in

The fan-in of a logic gate is defined as the number of inputs (coming from similar circuits) that it can handle properly.

Power Dissipation

This is the amount of power dissipated in an IC. It is determined by the current, Icc. that it draws from the Vcc supply and equals Vcc Icc. This power is specified in mW. Lower power dissipation is desirable feature for any IC.

Noise Margin

Noise margin is the amount of noise that a CMOS circuit could withstand without compromising the operation of circuit.

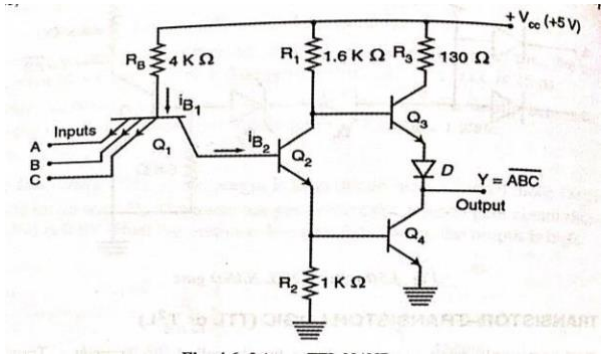
Power Supply requirement

Every IC requires a certain amount of electrical power to operate. The power is supplied by one or more powersupply voltage connected to the power pin (or pins) on the chip. Usually there is only one power-supply terminal on the chip and it is marked Vcc for TTL or VDD for for MOS devices. Obviously low power consumption is desirable features in any digital ICs.

6.3 Features, circuit operation &various applications of TTL(NAND), CMOS (NAND & NOR)

Transistor Transistor Logic (TTL)

The basic circuit for TTL logic family is the NAND gate. The TTL circuit uses a special 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.



Circuit operation:

- When any one of the input is logic 0, the emitter junctions of transistor T1 are forward biased. The required voltage to conduct T2 and T3 is greater than the voltage available at the base of T1 and hence T2 and T3 are in cut-off. The output voltage is equal to the supply voltage Vcc, output is in logic 1 state.
- When all the inputs are in logic 1 state, the emitter junction of T1 are reverse biased and the current supply by the source is sufficient to operate T2 and T3 in saturation and the output is in logic 0 state.

Application:

- Switching device in driving lamps and relays
- Processors of mini computers
- Used in printers and video display terminals

CMOS NAND:

1.18.2 CMOS NAND Gate

Figure 1.45 shows a two-inputs CMOS logic NAND gate. It consists of two *p*-channel and two *n*-channel MOSFETs. *p*-channel MOSFETs are connected in parallel and *n*-channel MOSFETs are connected in series. Here, Q_1 and Q_2 are *p*-channel MOSFETs and Q_3 and Q_4 are *n*-channel MOSFETs.

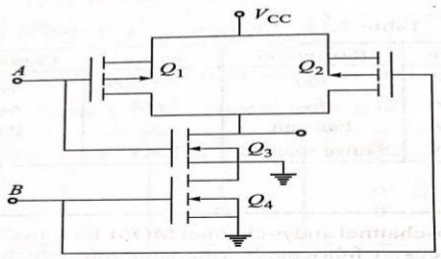


Fig. 1.45 CMOS NAND gate

Operation

- When the inputs are low, Q_1 and Q_2 are ON, Q_3 and Q_4 are OFF, and the output is high (V_{DD}).
- When any one of the inputs is low (0 V or –ve), then the corresponding MOSFET Q_1 or Q_2 is ON, Q_3 or Q_4 is ON, and the output is high.
- When the inputs are high (+ve voltage), Q_1 and Q_2 are OFF, Q_3 and Q_4 are ON, and the output is low.

The operation of the circuit is summarized in Table 1.16.

Table 1.16 Operations of CMOS NAND gate

A	*B*	Q_1	Q_2	Q_3	Q_4	V_o
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

CMOS NOR

1.18.3 CMOS NOR Gate

Figure 1.46 shows a two-inputs CMOS logic NOR gate. It consists of two *p*-channel and two *n*-channel MOSFETs. *n*-channel MOSFETs are connected in parallel and *p*-channel MOSFETs are connected in series. Here, Q_1 and Q_2 are *p*-channel MOSFETs and Q_3 and Q_4 are *n*-channel MOSFETs. When the input is low, *p*-channel MOSFETs are ON and *n*-channel MOSFETs are OFF. When the input is high, *p*-channel is OFF and *n*-channel is ON.

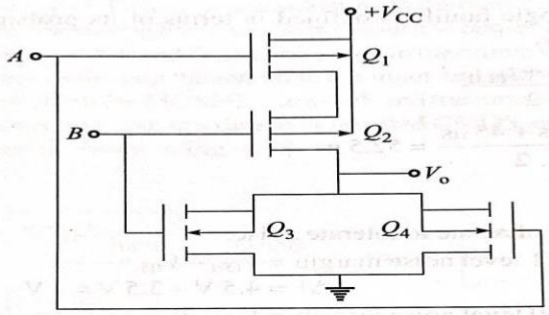


Fig. 1.46 CMOS NOR gate

Operation

- When inputs are low, Q_1 and Q_2 are ON, Q_3 and Q_4 are OFF, and the output is high (V_{DD}).
- When any one of the inputs is low (0 V or –ve), then the corresponding MOSFET Q_1 or Q_2 is ON, Q_3 or Q_4 is ON, and the output is low.
- When inputs are high (+ve voltage), Q_1 and Q_2 are OFF, Q_3 and Q_4 are ON, and the output is low.

Table 1.17 Operations of CMOS NOR gate

A	*B*	Q_1	Q_2	Q_3	Q_4	V_O
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	0
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0