SHARIF

Digital Electronics

chapter-1
Binary systems

□ Define digital Electronics. Write the advantages and disadvatages of digital Electronics.

Digital Electronics is the sub-branch of ~~elements~~ electronics that deals with digital signals for processing and controlling various systems and sub-systems. In various applications like sensors and actuators, usage of digital electronics is increasing extensively. In digital electronics, we facilitate two state or binary logic. There are two logic states including "0" (low) and "1" (high).

Advantages of digital electronics:

(1) Digital Electronics circuits are relatively easy to design.

(2) It has higher precision rate in terms of accuracy.

(3) Transmitted signals are not lost over long distance.

(4) Digital ~~systems sys~~ signals can be stored easily.

(5) Digital Electronics is more immune to "error" and "noise" than analog.

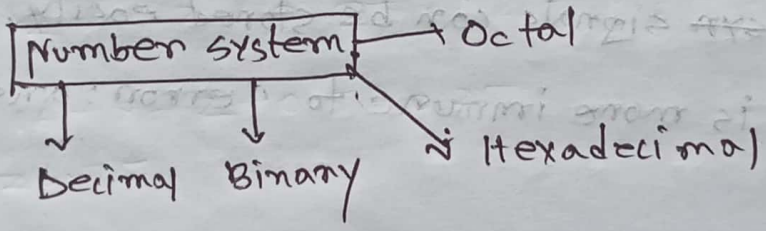
(6) The voltage at any point in a digital circuit can be either high or low, hence there is less chance of confusion.

Disadvantages of Digital Electronics:

The real world is analog in nature, all quantities like light, temperature, sound etc. Digital systems have to convert a continuous signal to a discrete one, which introduces small quantization errors. In order to reduce quantization errors, a large amount of data must be stored in Digital circuit.

Digital circuits only work with digital signals, so the process requires encoders and decoders. This increases the cost of the equipment.

Number system: In digital Electronics, a number system allows the information or values to be represented in the form of digits. Basically, a number system helps in representing the data within a digital computer.



Radix: The number of independent digits used in the number system is known as Radix or Base of the number system.

Decimal → 10 radix
Binary → 2 radix
Octal → 8 radix
Hexadecimal → 16 "

Bit A bit is a binary digit, the smallest increment of data on a computer. A bit can hold only one of two values : 0 or 1, corresponding to the electrical values of off and on state respectively.

Byte: Byte, the basic unit of information in computer storage and processing. A byte consists of 8 adjacent binary digits (bit) each of which consists of a 0 or 1.

Difference between Digital and Analog Electronics:

Analog	Digital
Analog Electronics is the branch of electronics which deals with the study of systems with analog system.	Digital Electronics is the branch of electronics that deals with the study of systems with digital signals.
Involves the use of continuous time signals.	uses discrete time signals or two state signals.
uses passive circuit components like resistors, capacitor etc.	uses active elements only.
Analog Electronic systems consume more power	Digital Electronics systems consume comparatively less power.
Analog Electronics have some power loss.	There is no power loss in case of digital electronics.
High noise and distortion	low noise and distortion.
Fm radios, telephone, TVs	automation, digital watches.

Binary Logic: Binary logic is a set of rules for dealing with logical arguments that must be true or false. Binary logic is used to describe, in a mathematical way, the manipulation and processing of binary information.

Truth Table: A truth table is a tabular representation of all the combinations of values for inputs and their corresponding outputs. It is basically used to check whether the propositional expression is true or false.

Binary number: Binary is a base-2 number system representing numbers using a pattern of ones and zeros.

The decimal equivalent of the binary number 11010.11 is 26.75.

$$(11010.11)_{10}$$

$$1\times2^4+1\times2^3+0\times2^2+1\times2^1+0\times2^0+1\times2^{-1}+1\times2^{-2}$$

$$= 16+8+2+0.5+0.25$$

$$= 26.75$$

An example of base-5 number is

$$(4021.2)_5 = 4\times5^3+0\times5^2+2\times5^1+1\times5^0+2\times5^{-1}$$

$$= 500+10+1+0.4$$

$$= (511.4)_{10}$$

An example of a hexadecimal number is

$(B65F)_{16} = 11\times16^3 + 6\times16^2 + 5\times16 + 15\times16^0$

$= 45056 + 1536 + 80 + 15$

$= (46687)_{10}$

Number Base conversion:

Number system conversion deals with the operations to change the base of the numbers. For example, to change a decimal number with base 10 to binary number with base 2.

Conversion from decimal to binary:

Example: 1-1

$2 \mid 41$

$2 \mid 20 - 1$

$2 \mid 10 - 0$

$2 \mid 5 - 0$

$2 \mid 2 - 1$

$2 \mid 1 - 0$

$0 - 1$

$\therefore (41)_{10} = (101001)_2$

Example 1.3: Covent $(0.6875)_{10}$ to binary

	Integer	fraction	coefficient
$0.6875 \times 2 =$	1	+ 0.375	$a_{-1} = 1$
$0.375 \times 2 =$	0	+ 0.75	$a_{-2} = 0$
$0.75 \times 2 =$	1	+ 0.5	$a_{-3} = 1$
$0.5 \times 2 =$	1	+ 0.0	$a_{-4} = 1$

$\therefore (0.6875)_{10} = (0.1011)_2$

Example 1.2 : Decimal to Octal

$$
\begin{array}{r|l}
8 & 153 \\
\hline
8 & 19 - 1 \\
\hline
8 & 2 - 3 \\
\hline
 & 0 - 2
\end{array}
\uparrow
$$

$\therefore (153)_{10} = (231)_8$

Example 1-4: Convert $(0.513)_{10}$ to octal

$0.513 \times 8 = 4.104$
$0.104 \times 8 = 0.832$
$0.832 \times 8 = 6.656$
$0.656 \times 8 = 5.248$
$0.248 \times 8 = 1.984$
$0.984 \times 8 = 7.872$

The answer, to seven significant figure is obtained from the integer part of the products :

$$(0.513)_{10} = (406517\ldots)_8$$

Octal and Hexadecimal numbers :

10 110 001 101 011 . 111 100 000 110
2 6 1 5 3 7 4 0 6

$= (26153.7406)_8$

SHARIF

binary to hexadecimal

$(10\ 1100\ 0110\ 1011.1111\ 0010)_2 = (2C6B.F2)_{16}$
2 C 6 B F 2

Octal to binary

$(673.124)_8 = (110111011.001010100)_2$

Hexadecimal to binary

$(306.D)_{16} = (0011000000110.1101)_2$

Complements: In number reprsesentation techniques, ~~the binary~~ The complement is used for representing the negetive decimal number in binary form. Different types of complement are possible of the binary number, but 1's and 2's complements are mostly used for binary numbers.

1's complement: We can find the 1's complement of the binary number by simply inventing the given number.

1010.1101 of 1's complement → 0101.0010

2's complement : ডানদিকের 0 থাকলে বসে যাবে। প্রথম 1 অপরিবর্তিত থাকবে তারপর অবশিষ্টগুলো অপর্যায় পরিবর্তিত হবে।

0001 → 1111
0010 → 1110
0011 → 1101
0111 → 1001

The r's complement:

If r is the base ~~system~~ of the number system, then there are two types of components that are possible. The r's complement is known as Radix complement.

Given a positive number N in base r with an integer part of n digits, the r's complement of N is defined as $r^n - N$ for $N \neq 0$.

The r's complement of a number exists for any base r (r greater than but not equal to 1) and may be obtained. The examples we use numbers with r=10 (decimal) and r=2 (binary).

The 10's complement of $(52520)_{10}$ is $10^5 - 52520 = 47480$

10 10's complement of $(0.3267)_{10} = 10^0 - 0.3267 = 0.6733$

The 10's complement of $(25.639)_{10}$ is $= 10^2 - 25.639 = 74.361$

The 2's complement of $(101100)_2$ is $(2^6)_{10} - (101100)_2$

$= (1000000 - 101100)_2$

$= 010100$

The 2's complement of (0.0110) is $1 - 0.0110$

$= 0.1010$

The (r-1)'s complement :

(r-1)'s complement is known as Diminished Radix complement. Given a positive number N in base r with an integer part of n digits and a fraction part of m digits, the (r-1)'s complement of N defined as $r^n - r^{-m} - N$.

The 9's complement of a decimal number is formed simply by subtracting every digit from 9. The 1's complement of a binary number is even simpler to form : the 1's are changed to 0's and the 0's to 1's. Since the (r-1)'s complement is very easily obtained, it is sometimes convenient to use it when r's complement is desired. The r's complement can be obtained from the (r-1)'s complement after the addition of r^{-m} to the least significant digit.

The 9's complement of $(52520)_{10}$ is $10^5 - 1 - 52520 = 47479$

The 9's complement of $(0.3267)_{10}$ is $1 - 10^{-4} - 0.3267$

$= 0.9999 - 0.3267$

$= 0.6732$

The 9's complement of $(25.639)_{10}$ is $10^2 - 10^{-3} - 25.639$

$= 100 - 0.001 - 25.639$

$= 74.36$

The 1's complement of $(101100)_2$ is $(2^6 - 1)_{10} - (101100)_2$

$= (111111 - 101100)_2$

$= 010011$

The 1's complement of $(0.0110)_2$ is $1-2^{-4}$ — 0.0110

$= (0.1111 - 0110)$

$= 0.1001$

10's complement:
ডানদিকে 0 থাকলে বাদ যাবে, তার প্রথম অংশ 10 থেকে বাকি গুলো 9 থেকে বাদ

2's complement:
ডানদিকে 0 থাকলে বাদ যাবে 1 প্রথম + অপরিবর্তিত থাকবে
তারপর অবশিষ্টগুলো সংখ্যা পরিবর্তিত হবে

9's complement:
যা থাকবে সব 9 থেকে বাদ

1's complement
বিপরীত

Subtraction

10's complement
carry end থাকলে carry end বাদে যা থাকে তাই রেজাল্ট
carry end না থাকলে – (10's complement)

2's complement
M-N
First N কে 2's complement বানাতে হবে

M+N
carry end থাকলে carry end বাদে যা আছে তাই
carry end না থাকলে
– (Ans's 2's complemen)

9's complement

carry end multiplied with true answer.

if no carry end

$-$ (Ans 9's complement)

1's complement

carry end থাকলে Answer +1

carry end না থাকলে $-$ (1's complement ans.)

Subtraction with r's complements:

The subtraction of two positive numbers $(M-N)$, both of base r, may be done as follows:

1. Add the minuend M to the r's complement of the subtrahend
2. Inspect the result obtained in step-1 for an end carry:
   (a) If an end carry occurs, discard it.
   (b) If an end carry does not occur, take the r's complement of the number obtained in step-1 and place a negative sign in front.

Example 1.5

Using 10's complement, subtract 72532 − 3250

M = 72532

N = 03250

10's complement of N = 96750

Now 72532
+ 96750
end carry → 1 / 69282

∴ Answer 69282

Example 1.6 : Subtract (3250 − 72532)10

m = 03250

N = 72532

10's complement of N = 27468

Now 03250
+ 27468
no carry / 30718

∴ −(69282) = − (10's complement of 30718)

Example 1-7: Using 2's complement M − N

M = 1010100

N = 1000100

2's complement of N = 0111100

Now

```
  1010100
+ 0111100
---------
  0010000
```

end carry = 1

Answer : 10000

(b) M = 1000100

N = 1010100

2's complement of N = 0101100

Now,

```
  1000100
+ 0101100
---------
  1110000
```

no carry

∴ − (2's complement of 1110000) is

$-(10000)_{Ans}$

Subtraction with (r-1)'s complement

The subtraction of M-N, both positive numbers in base r, may be calculated in the following manner.

1. Add the minuend to the (r-1)'s complement of the subtrahend N.

2. Inspect the result obtained in step 1 for an end carry
   (a) If an end carry occurs, add 1 to the least significant digit (end-around carry)
   (b) If an end carry does not occur, take the (r-1)'s complement of the number obtained in step 1 and place a negative sign in front of it.

Example 1.8

(a) M = 72532
N = 03250

9's complement of N = 96749

Now

```
  72532
+ 96749
--------
 169281
      1 +
--------
  69282
```

1 — End-around carry

(b) M = 03250
N = 72532

9's complement of N = 27467

Now, 03250
27467
No carry 30717

∴ −(9's complement of 30717) = −(69282)
= −69282

(a) M = 1010100
N = 1000100

1's complement of N = 0111011

Now, 1010100
0111011
1 0001111
+ 1
0010000

(b) M = 1000100
N = 1010100

1's complement of N = 0101011

Now
1000100
no carry 0101011
1101111

∴ −(1's complement of 1101111)

= − 0010000
= −10000

Comparison between 1's and 2's complements:

A comparison between 1's and 2's complements means the advantages and disadvantages of each. In case of 1's complement it it is easy to change 0's into 1's and 1's into 0's. The implementation of the 2's complement may be obtained in two ways. (1) by adding 1 to the least significant digit of the 1's complement and (2) by leaving all leadings 0's in the least significant positions and the first 1 unchanged and only then changing all 1's into 0's and all 0's ~~it~~ into 1's. During subtraction of two numbers by complements, the 2's complement is advantageous. in that only one anitmatic addition operation is required. The 1's complement requires two anithmatic additions when an end around carry occurs.

The 1's complement has the additional disadvantage of possessing two anithmetic zeros: one with all 0's and one with all 1's. To illustrate this let consider the subtraction of two equal binary numbers $1100 - 1100 = 0$

using 1's complement

$$\begin{array}{r} 1100 \\ 0011 \\ \hline 1111 \end{array}$$

$\therefore -0000$

Using 2's complement

```
  1100
  0100
 -----
 +0000
```

While the 2's complement has only one anitmatic zero, the 1's complement zero can be positive or negetive, which may compplicate matters.

1's complement is useful in logical manipulations, ~~since the chang of its~~ The 2's complement is used only in conjunction with arithmatic application.

SHARIF

Binary codes: Binary is a base-2 number system representing numbers using a pattern of ones and zeros

Decimal code: A decimal is a number that consists of a whole and a fractional part. Deimal codes are 10 based number system lies betreen 0 to 9.

BCD: Binary coded Decimal or BCD is the process for converting decimal numbers into their binary equivalents. The BCD is a straight assignment of the binary equivalen

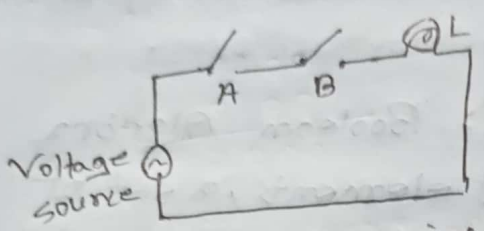
Binary stonage: The digital system must be able to physically store some information. When descrete elements of information are nepresented in binary form, the information storage medium must contain binary storage elements for stoning individual bits.

Registers: A register is a group of binary cells. Since a cell stores one bit of information, it follows that a register with n cells can store any discrete quantity of information that contains n bits.

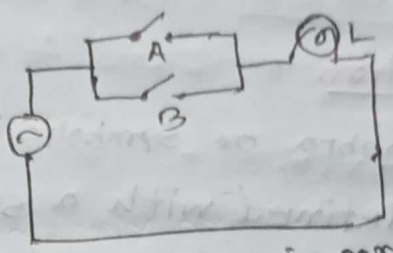
Logic gates: A logic gate is an electronic device that creates logical decisions depends on the various combinations of digital signals accessible on its output. There are seven basic gates such as AND, OR, XOR, NOT, NAND, NOR and XNOR.

Switching Circuits and Binary Signals:

The use of binary variables and the application of binary logic are demonstrated by the simple switching circuits of Fig. 1-4. Let the manual switches A and B represent two binary variables with values equal to 0 when the switch is open and 1 when the switch is closed. Similarly, let the lamp L represent a third binary variable equal to 1 when the light is on and 0 when off. For the switches in series, the light turns on if A and B are closed. For the switches in series, the light turns on if A or B is closed. It is obvious that the two circuit can be expressed by means of binary logic with the AND and OR operations, respectively:



(a) switches in series
logic AND



(b) switches in parallel
Logic OR

1.4: switching circuits that demonstate binary logic.

$L = A \cdot B$ for the circuit of Fig. 1.4(a)

$L = A + B$ for the circuit of Fig. 1.4(b)

Electronic digital circuits are sometimes called switching circuits because they behave like a switch, with the active element such as transistor either conducting (switch closed) or not conducting (switch open)

SHARIF

Chapter-2

Boolean Algebra and logic gates:

Basic Definition

- Boolean Algebra or Symbolic logic: Boolean Algebra may be defined with a set of elements, a set of operators and a number of unproved axioms or postulates.
- It defines rules for manipulating symbolic binary logic expressions.

A symbolic binary logic expression consists of binary variables and the operators AND, OR and NOT (ex. $(A+B)\cdot C'$)

- what is an algebra?

mathmatical system consisting of set of elements, set of operators and set of axioms.

Common Axioms or Postulates:

The postulates of a mathmatical system form the basic assumption from which it is possible to deduce the rules, theorems and properties of the systems.

- Closure: A set S is closed with respect to a binary operator, if for every pair of elements of S, the binary operator specifies a rule for obtaining an element of S. For example the set of natural numbers is closed w.r.t binary operator (+) but not closed w.r.t binary operator (−).

2. Associative law: A binary operator $*$ on a set S is said to be associative whenever

$(x*y)*z = x*(y*z)$ for all $x, y, z \in S$

3. Commutative law: A binary opeator $*$ on a set S is said to be commutative whenever:

$x*y = y*x$ for all $x, y \in S$

4. Identity element: A set S is said to have an identity element with respect to a binary operation $*$ on S if there exists an element $e \in S$ with the property.

$e*x = x*e = x$ for every $x \in S$

Example: The element 0 is an identity element w.r.t to operation + on the set of integers $I = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$

since $x+0 = 0+x = x$ for any $x \in I$

The set of natural numbers N has no identity element since 0 is excluded from the set.

5. Inverse: A set S having the identity element e with respect to a binary operator $*$ is said to have an inverse whenever, for every $x \in S$, there exists an element $y \in S$ such that,

$x*y = e$

Example: In the set of integers I with $e=0$, the inverse of element a is $(-a)$ since $a+(-a)=0$

6. Distributive law: If * and · are two binary operator on a set S, * is said to be distributive over.

$$x * (y \cdot z) = (x * y) \cdot (x * z)$$

The operator and postulates have the following meaning:

- The binary operator + defines addition
- The additive identity is 0
- The additive inverse defines subtraction.
- The binary operator · defines multiplication.
- The multiplicative identity is 1.
- The multiplicative inverse of $a = 1/a$ defines division

  i.e $a \cdot \frac{1}{a} = 1$
- The only distributive law applicable is that of · over +:

$$a \cdot (b+c) = (a \cdot b) + (a \cdot c)$$

Axiomatic definition of Boolean Algebra:

Boolean algebra is an algebric structure defined on a set of element B together with two binary operators + and · provided the following (Huntington) postulates are satisfied:

1. (a) Closure with respect to the operator +
   (b) Closure with respect to the operator ·.

2. (a) An identity element with respect to +; designated by 0:

$$x + 0 = 0 + x = x$$

(b) An identity element with respect to ·, designated by 1:

$$x \cdot 1 = 1 \cdot x = x$$

3. (a) Commutative with respect to +: $x + y = y + x$

(b) Commutative with respect to ·: $x \cdot y = y \cdot x$

4. (a) · is distributive over +: $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$

(b) + is distributive over ·: $x + (y \cdot z) = (x + y) \cdot (x + z)$

5. For every element $x \in B$, there exits an element $x' \in B$ (called the complement of x) such that

(a) $x + x' = 1$ and (b) $x \cdot x' = 0$

6. There exits at least two elements $x, y \in B$ such that

$$x \neq y.$$

Comparing Boolean algebra with arithmatic and ordinary algebra:

1. Huntington postulates do not include the associative law. However, this law holds for Boolean algebra and can be derived (for both operators) from the other postulates.

2. The distributive law of + over ·, i.e. $x + (y \cdot z) = (x + y) \cdot (x + z)$ is valid for Boolean algebra, but not for ordinary algebra.

3. Boolean Algebra does not have additive or multiplicative inverses; therefore there are no subtraction or division operations.

4. Postulate 5 defines an operator called complement which is not available in ordinary algebra.

5. Ordinary algebra deals with the real numbers, which constitute an infinite set of elements. Boolean algebra deals with the an yet undefined set of elements B, but in the two valued Boolean algebra defined below (and of interest in our subsequent use of this algebra), B is defined as a set with only two elements, 0 and 1

Two-Valued Boolean Algebra:

A two-valued Boolean algebra is defined on a set of two elements, $B = \{0, 1\}$, with rules for the two binary operators + and ·.

x	y	x · y
0	0	0
0	1	0
1	0	0
1	1	1

x	y	x + y
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

We must show now show that the Huntington postulates are valid for the set $B = \{0, 1\}$ and two binary operators defined above.

1. Closure is obvious from the tables in since the result of each operation is either 1 or 0 and $1, 0 \in B$

2. From the tables we see that:

(a) $0+0=0 \quad 0+1=1+0=1$

(b) $1 \cdot 1 = 1 \quad 1 \cdot 0 = 0 \cdot 1 = 0$

which establishes the two identity elements 0 for + and 1 for · as defined by postulate 2.

3. The commutative laws are obvious from the symmetry of the binary operator table.

4. (a) The distributive law $x \cdot (y+z) = (x \cdot y) + (x \cdot z)$ can be shown to hold true from the operator tables by forming a truth table of all possible values x, y and z. For each combination we derive $x \cdot (y+z)$ and show that the value is the same as $(x \cdot y) + (x \cdot z)$.

x y z	y+z	x·(y+z)	x·y	x·z	(x·y)+(x·z)
0 0 0	0	0	0	0	0
0 0 1	1	0	0	0	0
0 1 0	1	0	0	0	0
0 1 1	1	0	0	0	0
1 0 0	0	0	0	1	0
1 0 1	1	1	0	0	1
1 1 0	1	1	1	1	1
1 1 1	1	1	1		1

(b) The distributive law of + over · can be shown to hold true by means of a truth table similar to the one above

5. From the complement table it is easily shown that

(a) $x+x'=1$ since $0+0'=0+1=1$ and $1+1'=1+0=1$

(b) $x \cdot x'=0$ since $0 \cdot 0'=0 \cdot 1=0$ and $1 \cdot 1'=1 \cdot 0=0$

which verifies postulate 5.

6. Postulate 6 is satisfied because the two-valued boolean algebra has two distinct elements 1 and 0 with $1 \neq 0$

Basic Teorems and properties of Boolean Algebra:

The Duality principle

The dual of a Boolean expression is obtained by interchanging all ANDs and ORs, and all 0's and 1s

Example: The Dual of $A+(B \cdot C)+0$ is $A \cdot (B+C) \cdot 1$

The duality principle states that if E_1 and E_2 are Boolean expression then

$E_1 = E_2 \Leftrightarrow \text{dual}(E_1) = \text{dual}(E_2)$

where dual(E) is the dual of E. for example

$A+B(B \cdot C)+0 = (B \cdot C)+D \Leftrightarrow A \cdot (B+C) \cdot 1 = (B+C) \cdot D$

Basic theorems:

(a) $x+0=x$ (b) $x\cdot 1=x$

(a) $x+x'=1$ (b) $x\cdot x'=0$

(a) $x+x=x$ (b) $x\cdot x=x$

(a) $x+1=1$ (b) $x\cdot 0=0$

$(x')'=x$

(a) $x+y=y+x$ (b) $xy=yx$

(a) $x+(y+z)=(x+y)+z$ (b) $x(yz)=(xy)z$

(a) $x(y+z)=xy+xz$ (b) $x+yz=(x+y)(x+z)$

(a) $(x+y)'=x'y'$ (b) $(x\cdot y)'=x'+y'$

(a) $x+xy=x$ (b) $x(x+y)=x$

Theorem $x+x=x$

$$\begin{aligned} \therefore x+x &= (x+x)\cdot 1 \\ &= (x+x)(x+x') \\ &= x+xx' \\ &= x+0 \\ &= x \end{aligned}$$

Theorem-2

$x\cdot x=x$

$$\begin{aligned} \therefore x\cdot x &= xx+0 \\ &= xx+xx' \\ &= x(x+x') \\ &= x\cdot 1 \\ &= x \end{aligned}$$

Theorem 2(a) $x+1=1$

$\Rightarrow x+1 = 1.(x+1)$

$= (x+x')(x+1)$

$= x+x'1$

$= x+x'$

$= 1$

Theorem (2b): $x.0 = 0$ by duality.

Theorem-3: $(x')' = x$

From postulate (5), we have $x+x'=1$ and $x.x'=0$ which defines the complement of x. The complement of x' is x and is also $(x')'$. Therefore, since the complement is unique, we have that $(x')' = x$.

Theorem 6(a): $x+xy = x$

$\therefore x+xy = x.1 + xy$

$= x(1+y)$

$= x.1$

$= x$

Theorem 6(b): $x(x+y) = x$ by duality

The theorems of Boolean algebra can be shown to hold true by means of truth tables. both sides of the relation are checked to yield identical results for all possible combinations of variables involved. The following truth table verifies the first absorption theorem.

x	y	xy	$x+xy$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

$\therefore x+xy = x$

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

$\therefore x \cdot 0 = 0 + x \cdot 0$

$= x \cdot x' + x \cdot 0$

$= x(x' + 0)$

$= x \cdot x'$

$= 0$

Theorem 3: $(x')' = x$

$(x')' = (x')' + 0$

$= (x')' + x \cdot x'$

$= ((x')' + x) \cdot ((x')' + x')$

$= (x + (x')') \cdot 1$

$= (x + (x')') \cdot (x + x')$

$= x + x'(x')'$

$= x + 0$

$= x$

Theorem $(x+y)' = x' \cdot y'$

We show that $x+y$ and $x' \cdot y'$ are complementary.

$L = x+y$

$L' = x' \cdot y'$

To be complement

$L \cdot L' = 0$ and $L + L' = 1$

$L \cdot L' = (x+y) \cdot (x' \cdot y')$

$= (x \cdot y') \cdot x + (x' \cdot y') \cdot y$

$= x' \cdot x \cdot y' + x' \cdot y \cdot y$

$= 0 \cdot y' + x' \cdot 0$

$= 0 + 0$

$= 0$

$L+L' = (x+y)+(x'y')$

$= (x+y+x')(x+y+y')$

$= (1+y)\cdot(1+x)$

$= 1\cdot 1$

$= 1$

∴ so it is proved that $(x+y)' = x'y'$

Theorem 5(b) = $(x\cdot y)' = x'+y'$

Let $L = x\cdot y$ and $L' = x'+y'$

$\therefore L\cdot L' = (x\cdot y)(x'+y')$

$= (x\cdot(x'+y'))\cdot(y\cdot(x'+y'))$

$= (x\cdot x' + xy')\cdot(y\cdot x' + y\cdot y')$

$= (x\cdot x')\,(y\cdot x')$

$= 0\cdot 0$

$= 0$

$L+L' = (x\cdot y)+(x'+y')$

$= (x+(x'+y'))(y+x'+y')$

$= (1+y')\cdot(1+x')$

$= 1\cdot 1$

$= 1$

$\therefore (x\cdot y)' = x'+y'$

Boolean functions

Boolean Function is an expression formed with binary variables, two binary operators OR and AND, the unary operator NOT, parentheses, and equal sign. For a given value of the variables, the function can be either 0 or 1.

For example,

$F_1 = xyz'$, $F_2 = x + y'z$, $F_3 = x'y'z + x'yz + xy'$,

$F_4 = xy' + x'z$.

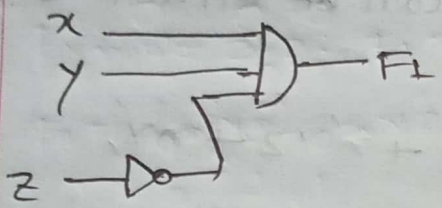
Boolean Function may be represented in a Truth Table.

From the table, we find that F_4 is same as F_3. Why?

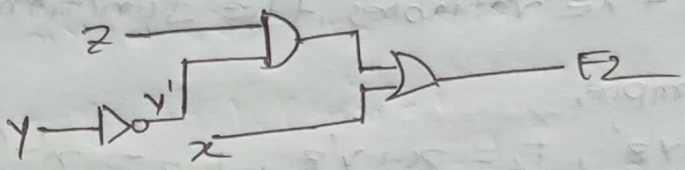
Two function of n binary variables are said to be equal if they have same value for all possible 2^n combinations of n variables.

x	y	z	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

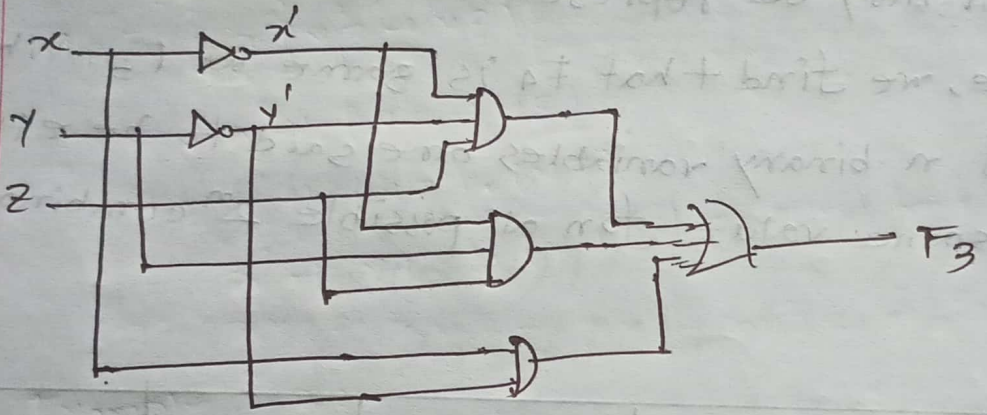
A Boolean function may be transformed from an algebric expression into a logic diagram composed of AND, OR and NOT gates.



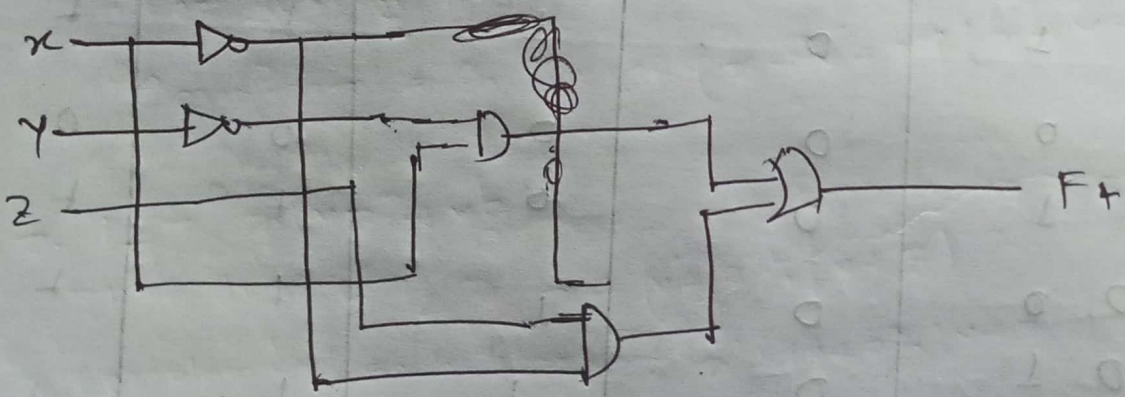
(a)



(b)



$F_3 = x'y'z + x'yz + xy'$



$F_1 = xy' + x'z$

Algebric manipulation: A literal is a primed or unprimed variable. When a boolean function is implemented with logic gates, each literal in the function designates an input to a gate, and each term is implemented with a gate. The minimization of the number of literals and the number of terms results in a circuit with less equipment.

Example 2.1:

1. $x + x'y = (x+x')(x+y) = 1 \cdot (x+y) = x+y$
2. $x(x'+y) = xx' + xy = xy + 0 = xy$
3. $x'y'z + x'yz + xy' = x'z(y'+y) + xy' = x'z + xy'$
4. $xy + x'z + yz = xy + x'z + yz(x+x')$
   $= xy + x'z + xyz + x'yz$
   $= xy(1+z) + x'z(1+y)$
   $= xy + x'z$
5. $(x+y)(x'+z)(y+z) = (x+y)(x'+z)$ by duality form.

Complement of a function:

The complement of a function F is F' and is obtained from an interchange of 0's for 1's and 1's for 0's in the value of F. The complement of a function may be deriered algebrically through De Morgan's theorem.

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

$= A' \cdot X'$

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

Example 2-2: $F_1 = x'yz' + x'y'z$, $F_2 = x(y'z' + yz)$

$F_1' = (x'yz' + x'y'z)' = (x'yz')' \cdot (x'y'z)'$

$= (x+y'+z)(x+y+z')$

$F_2' = \{x(y'z' + yz)\}'$

$= x' + (y'z' + yz)'$

$= x' + (y'z')' \cdot (yz)'$

$= x' + (y+z) \cdot (y'+z')$

Example 2-3:

1. $F_1 = x'yz' + x'y'z$

The dual of $F_1 = (x'+y+z') \cdot (x'+y'+z)$

Complement each literal $(x+y'+z)(x+y+z') = F_1'$

2. $F_2 = x(y'z' + yz)$

The dual of $F_2 = x + (y'+z') \cdot (y+z)$

Complement each literal: $x' + (y+z) \cdot (y'+z') = F_2'$

SHARIF

Canonical and standard forms:

If we take any expanded Boolean Expression where each term contains all Boolean variables in their true or complemented form, is also known as the canonical form of the expression.

Minterm: The product of all literals, either with complement or without complement, is known as minterm.

Consider two binary variables x an y combined with an AND operation. So there are four possible combinations $x'y', x'y, xy', xy$. Each of these four AND terms is called a minterm or a standard product. In a similar manner, n variables can be combined to form 2^n minterms.

Maxterm: The sum of all literals, either with complement or without complement is known as maxterm.

Consider two binary variables x and y combined with an OR operation. So there are four possible combinations : $x'+y', x+y', x'+y, x+y$. Each of these four OR terms is called a maxterm or a standard sum.

Canonical and Standard

Minterms and Maxterms for three Binary Variables.

x	y	z	Minterm Term	Minterm Designation	Maxterm Term	Maxterm Designation
0	0	0	$x'y'z'$	m_0	$x+y+z$	M_0
0	0	1	$x'y'z$	m_1	$x+y+z'$	M_1
0	1	0	$x'yz'$	m_2	$x+y'+z$	M_2
0	1	1	$x'yz$	m_3	$x+y'+z'$	M_3
1	0	0	$xy'z'$	m_4	$x'+y+z$	M_4
1	0	1	$xy'z$	m_5	$x'+y+z'$	M_5
1	1	0	xyz'	m_6	$x'+y'+z$	M_6
1	1	1	xyz	m_7	$x'+y'+z'$	M_7

$$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

$$f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$$

x	y	z	Function f_2	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	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

Complement of a boolean function. It may be read from the truth table by forming a minterm for each combination that produces 0 in the function then ORing these terms.

$$f_1' = x'y'z' + x'yz' + x'yz + xy'z + xyz'$$

Now if we take the complement of f_1', we obtain the function f_1:

$$f_1 = (x+y+z)(x+y'+z)(x+y'+z')(x'+y+z')(x'+y'+z)$$
$$= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6$$

Example 2.4 $F = A + B'C$

The function has three variables A, B and C
The first term A is missing two variables; therefore

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

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

The second term $B'C$ is missing one variable

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

Combining all terms, we have

$$F = A + B'C$$
$$= ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C$$
$$= m_7 + m_6 + m_5 + m_4 + m_4 + m_1$$

$$F(A,B,C) = \Sigma(1,4,5,6,7)$$

Product of Maxterms:

Example 2.5

~~$F = (xy + x')(xy + z)$~~

$\equiv$

$F = xy + x'z$

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

$= (x + x')(x' + y)(x + z)(y + z)$

$= (x' + y)(x + z)(y + z)$

The function has three variables: x, y and z. Each OR terms ~~using~~ is missing one variable:

Therefore

$x' + y + zz' = (x' + y + z)(x' + y + z')$

$x + z + yy' = (x + z + y)(x + z + y')$

$y + z + xx' = (y + z + x)(y + z + x')$

Combining all terms

$F = (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z')$

$= M_0 M_2 M_4 M_5$

$\therefore F(x, y, z) = \Pi(0, 2, 4, 5)$

Conversions between Canonical Forms

The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function

As an example, consider the function $F(A,B,C) = \Sigma(1,4,5,6,7)$

This has a complement that can be expressed as

$$F'(A,B,C) = \Sigma(0,2,3) = m_0 + m_2 + m_3$$

If we take the compliment of F' by De morgan's theorem, we obtain F in a different form:

$$F = (m_0 + m_2 + m_3)' = m_0' \cdot m_2' \cdot m_3' = M_0 M_2 M_3$$
$$= \Pi(0,2,3)$$

steps:

1. Change the operational symbols used in equation such as Σ, Π.
2. Use the De morgan's principal to write the indexes of the terms that are not presented in the given form of an equation or the index numbers of the Boolean function.

Fan out: In digital electronics, the fan out is the number of gate input driven by the output of another single logic gate. In most designs, logic gates are connected to form more complex circuits.

Fan in: Fan in is a term that defines the maximum number of digital inputs that a single logic gate can accept.

power dissipation: Power dissipation is the supplied power required to operate the gate. This parameter is expressed in milliwatts (mw).

Noise mangin: Noise margin is the maximum noise voltage added to the input signal of a digital circuit that does not cause an undesirable change in the circuit output.

SHARIF

Chapter - 3

Simplification of Boolean Function

What is K-map?

A Karnaugh map (K-map) is a visual method used to simplify the algebric expression in boolean functions without having to resort equation manipulation. It is a special version of a truth table that makes it easier to simplify boolean expression.

Simplification of boolean function:

Simplification of boolean functions, the algebric forms of functions can often be simplified, which leads to simpler and chean implementation.

Advantages:

1. Simplification of Digital Logic circuits.
2. Reduce numbers of gates used in the circuit. Thus reduces cost.
3. Only produces specified results — no indefinite or may be type result.
4. Increase speed of calculation.
5. Helps cheaking error in progroms.

Two and three variable maps:

x \ y	0	1
0	$x'y'$	$x'y$
1	xy'	xy

Example 3-1

$$F = x'yz + xy'z' + xy'z + xy'z$$

x \ yz	$\bar{y}\bar{z}$ 00	$\bar{y}z$ 01	yz 11	$y\bar{z}$ 10
$\bar{x}$ 0			1	1
x 1	1	1		

$$F = x'y + x\bar{y}$$

Example 3.2

$$F = x'yz + xy'z' + xyz + xyz'$$

x \ yz	$\bar{y}\bar{z}$ 00	$\bar{y}z$ 01	yz 11	$y\bar{z}$ 10
$\bar{x}$ 0			1	
x 1	1		1	1

$$F = yz + xz'$$

Example 3.3

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

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

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

Example 3.4

$F(x,y,z) = \Sigma(0,2,4,5,6)$

x \ YZ	$\bar{Y}\bar{Z}$ 00	$\bar{Y}Z$ 01	YZ 11	$Y\bar{Z}$ 10
$\bar{x}$ 0	1			1
x 1	1	1		1

$F = \bar{Z} + x\bar{Y}$

Four variable map

Example 3.5 :

$F(w,x,y,z) = \Sigma(0,1,2,4,5,6,8,9,12,13,14)$

wx \ yz	00	01	11	10
00	1	1		1
01	1	1		1
11	1	1		1
10	1	1		

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

Example: 3.6 : $F = A'B'c' + B'cD' + A'BcD' + AB'c'$

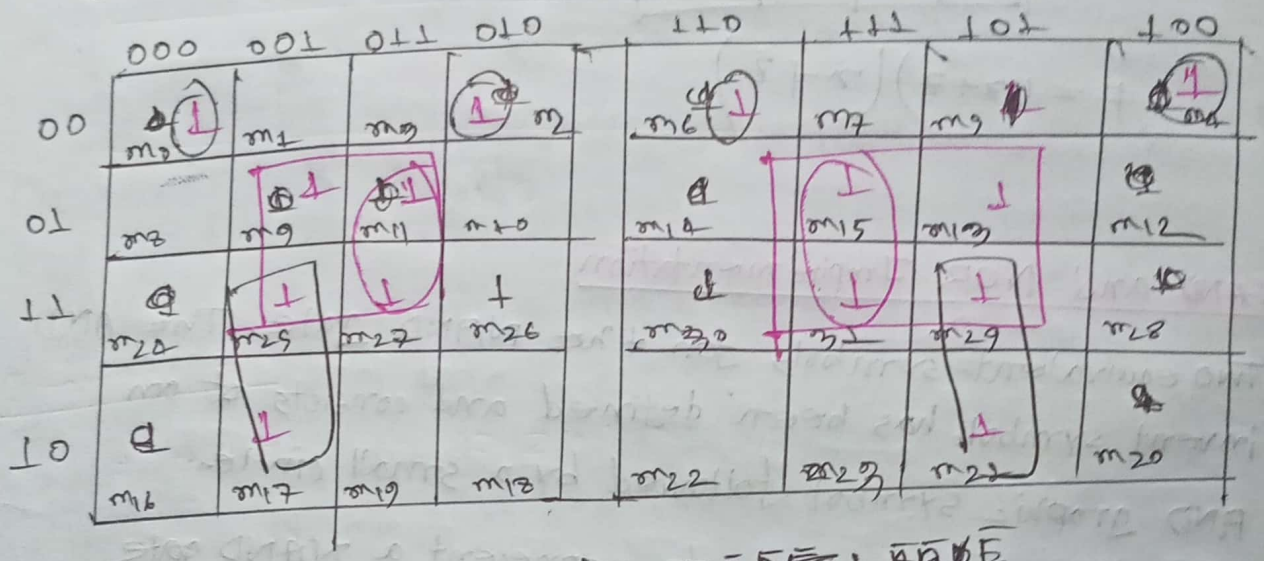
AB \ CD	$\bar{C}\bar{D}$ 00	$\bar{C}D$ 01	CD 11	$C\bar{D}$ 10
$\bar{A}\bar{B}$ 00	1	1		1
$\bar{A}B$ 01				1
AB 11				
$A\bar{B}$ 10	1	1		1

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

Five and six variables map

Example 3.7

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



$$F = \bar{B}\bar{E} + A\bar{D}\bar{E} + \cancel{\bar{A}\bar{B}\bar{D}\bar{E}} + \bar{A}\bar{B}\bar{D}\bar{E}$$

$$= \cancel{\bar{B}\bar{E} + A\bar{D}\bar{E} + \bar{A}\bar{B}\bar{E}'}$$

Product of sums simplification:

$$F(A,B,C,D) = \Sigma(0,1,2,5,8,9,10)$$

CD / AB	00	01	11	10
00	1	1	0	2
01	0	1	0	0
11	0	0	0	0
10	1	1	0	1

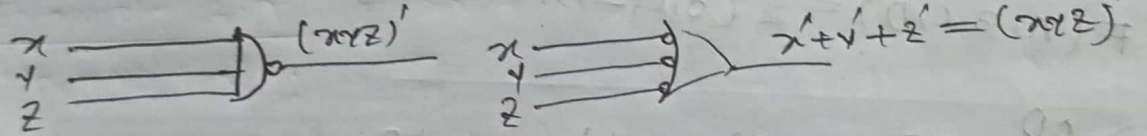
$$F = (\bar{A}+\bar{B})(\bar{C}+D)(B'+D)$$

	00	01	11	10
0	0			0
1		0	0	

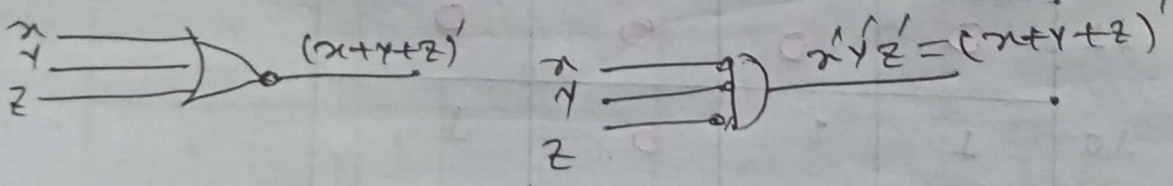
$F = (x+z)(\bar{x}+\bar{z})$

NAND and NOR Implementation

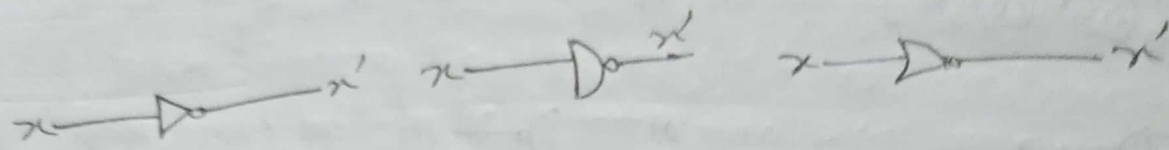
Two equivalent symbols for the NAND gate. The AND invert symbol has been defined and consists of an AND graphic symbol followed by a small circle. Instead, it is possible to represent a NAND gate by an OR graphic symbol preceded by small circles in all the inputs. The invert OR and AND invert follow De Morgan's law.



Similarly the OR-invert and invert AND follow the De Morgan's law:



A one input NAND or NOR gate behaves like an inverter



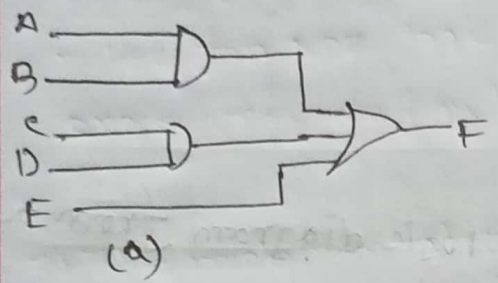
NAND Implementation

The rule for obtaining the NAND logic diagram from a boolean function is as follows:

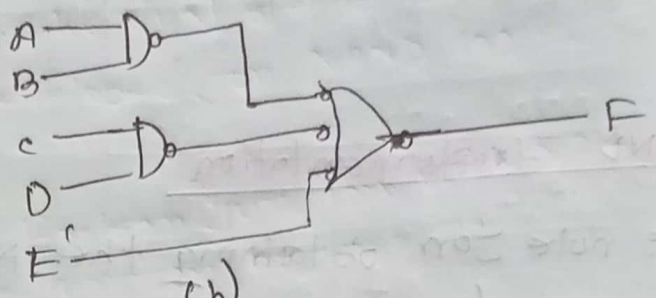
1. Simplify the function and express it in sum of products.
2. Draw a NAND gate for each product ~~of SOPA~~ term of the function that has at least two literals. The inputs to each NAND gate are the literals of the term. This constitutes a group of first level gates.
3. Draw a single NAND gate (using the AND inverter or invert OR graphic symbol) in the second level, with inputs coming from outputs of first level gates.
4. A term with a single literal requires an inverter in the first level or may be complemented and applied as an input to the second level NAND gate.

$F = AB + CD + E$

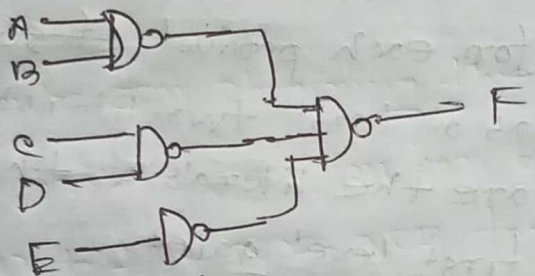
$= [(AB)' \cdot (CD)' \cdot E']'$



(a)



(b)



(c)

Three ways implementation $F = AB + CD + E$

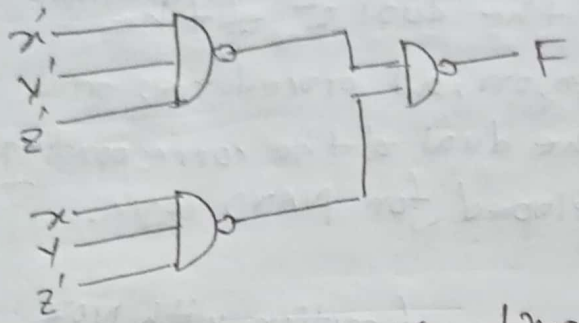
Example 3.9 $F(x,y,z) = \Sigma(0,6)$ Implementation with NAND Gate.

$F(x,y,z) = \Sigma(0,6)$

The first step to simplify the function in sum of products form. By using map method we gate

x \ yz	00	01	11	10
0	1			
1				1

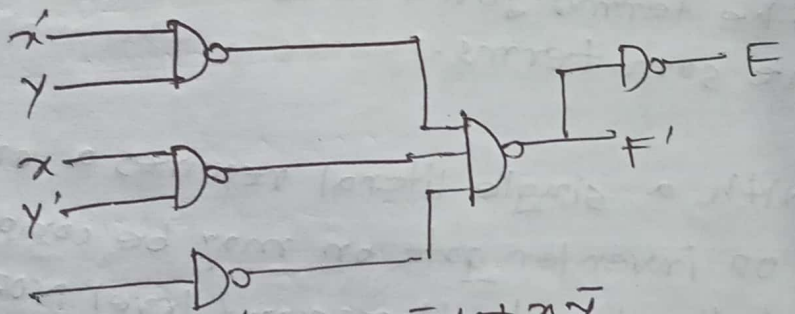
$F = \bar{x}\bar{y}\bar{z} + xyz'$



Next we try to simplify the complement of the function in sum of product. This is done by combining 0's in the map.

x \ yz	00	01	11	10
0		0	0	0
1	0	0	0	

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



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

If output F is required it is necessary to add a one-input NAND gate to invert the Function because first it was on the state of F'.

NOR Implementation

The NOR implementation is the dual of the NAND function. For this reason, all procedures and rules for NOR logic are the dual of the corresponding procedures and rules developed for NAND logic.

⇒ The implementation of Boolean function with NOR gates requires that the function be simplified in product of sum forms.

⇒ The rule for obtaining the NOR logic diagram from a boolean function can be derived from OR-AND to the NOR-NOR transformation. It is similar to the three step NAND rule, except that the simplified expression must be in the product of sums and the terms for the first level NOR gates are the sum terms.

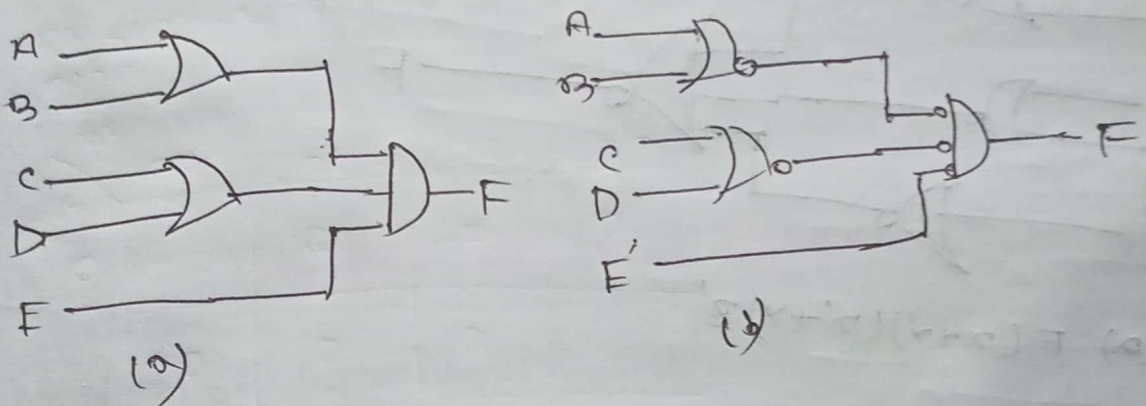
⇒ A term with a single literal requires a one input NOR or inverter gate or may be complemented and applied directly to the second level NOR gate.

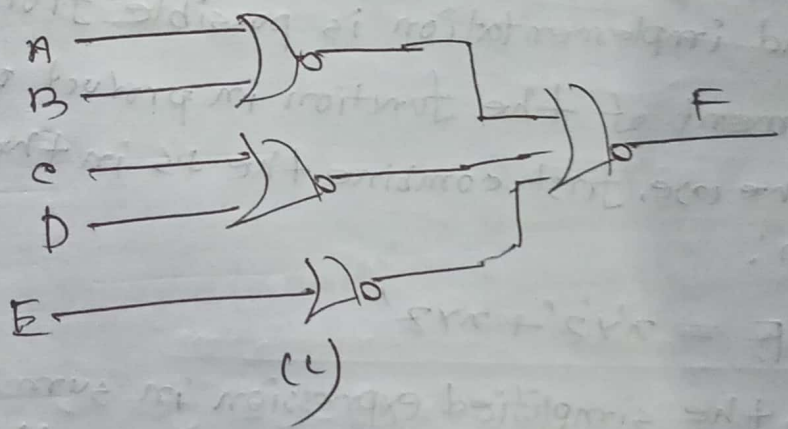
Alternative :

A ~~scond~~ second way to implement a function with NOR gates would be to use the expression for the complement of the function in product of the sum.

This will give a two level implementation for F' and a three level implementation if the normal output F is required.

⇒ To obtain the simplified product of sum expression for the complement of the function, it is necessary to combine the 1's in the map and then complement the function.





Example 3-10 :-

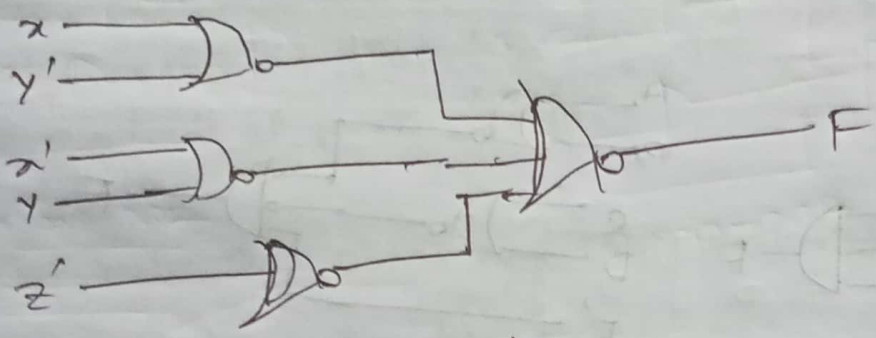
$F(x,y,z) = \Sigma(0,6)$ with NOR implementation

First combines the 0's in the map to obtain

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

This is the complement of the function in sum of products. Complement F' to obtain the simplified function in product of sum as required for NOR implementation

$$F = (x+y')(x'+y)z'$$



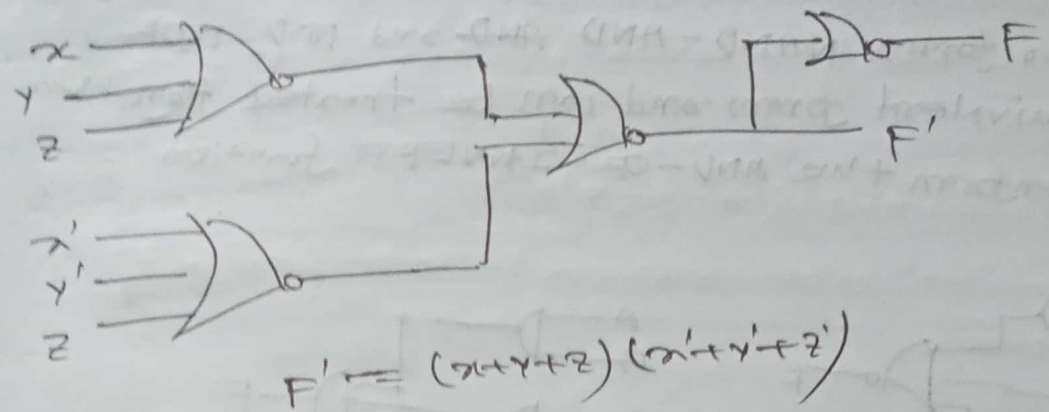
(a) $F(x+y')(x'+y)z'$

A second implementation is possible from the complement of the function in product of sums. For the case, first combine the 1's in the map to obtain:

$$F = x'y'z' + xyz'$$

This is the simplified expression in sum of products. Complement this product to obtain the complement of the function in product of sum as required for NOR implementations.

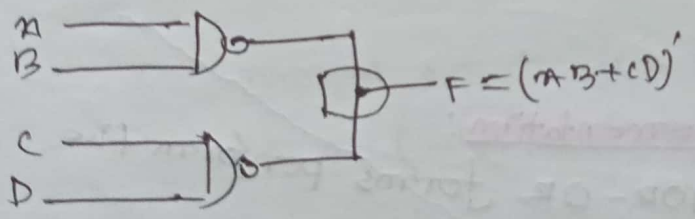
$$F' = (x+y+z)(x+y+z)$$



$F' = (x+y+z)(x'+y'+z')$

other two level implementation

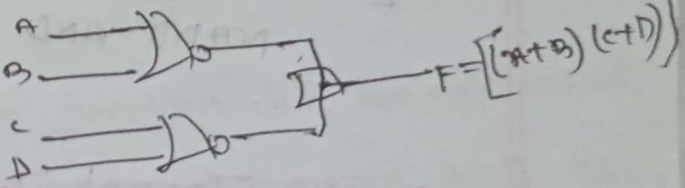
Wired logic: Some NAND and NOR gates allow the possibility of a wire connection between the outputs of two gates to provide a specific logic function. This type of logic is called wired logic.



↓

$F = (AB)' \cdot (CD)'$

$= (AB+CD)'$



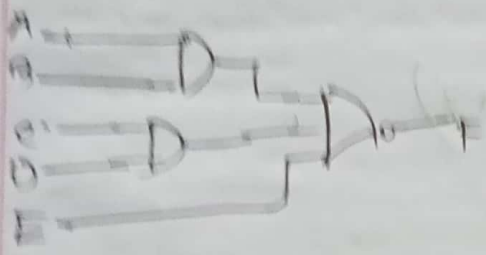
↓

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

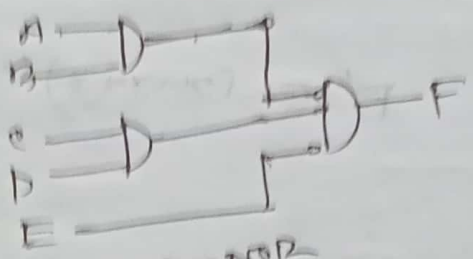
$= [(A+B)(C+D)]'$

AND-OR-Invert Implementation:

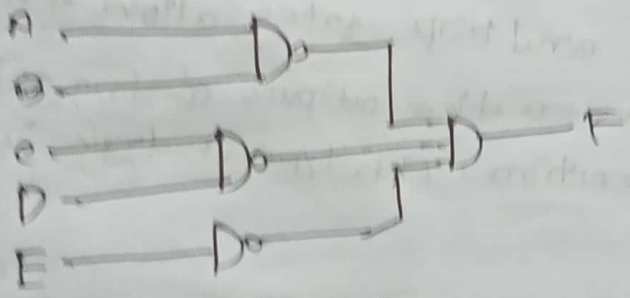
The two forms NAND-AND and AND-NOR are equivalent forms and can be treated together. Both perform the AND-OR-INVERT function.



(a) AND-NOR



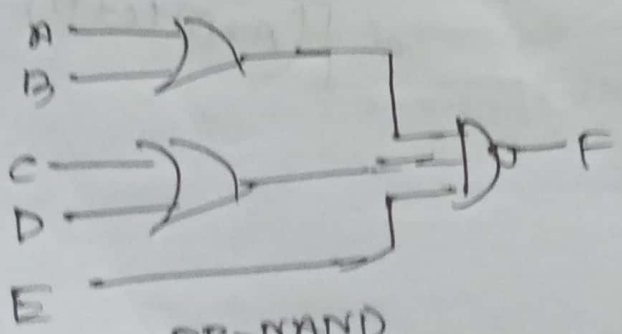
(b) AND-NOR



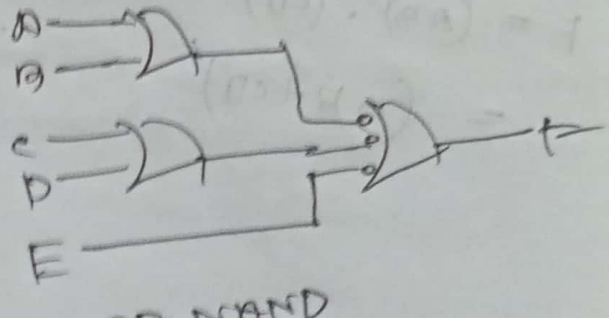
NAND-AND

OR-AND Invert Implementation:

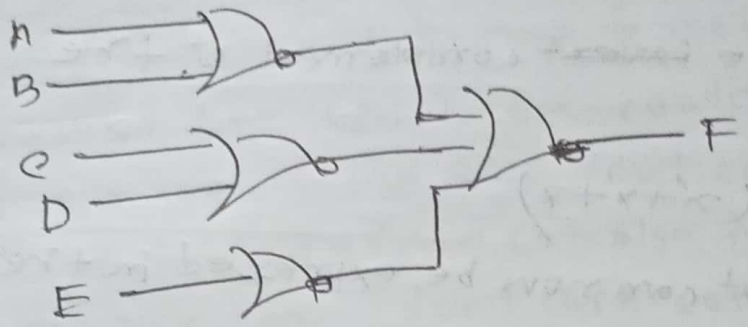
The OR-NAND and NOR-OR forms perform the OR-AND Invert function



OR-NAND



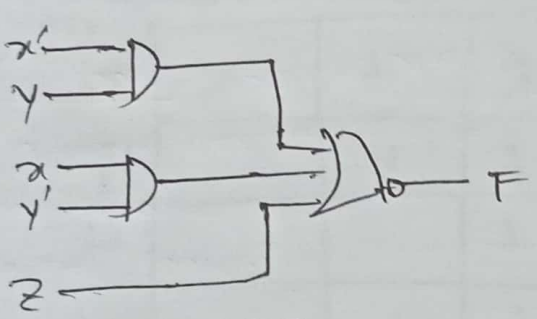
OR-NAND



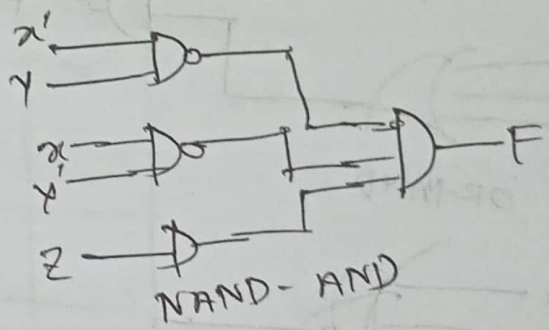
NOR-OR

Example 3.11

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



AND-NOR



NAND-AND

The normal output for the function can be expressed

$F = (x'y + xy' + z)'$ which is in the AND-OR invert.

The OR-AND invert forms require a simplified expression of the complement of the function in product of sums. To obtain this expression, we must first combine the 1's in the map,

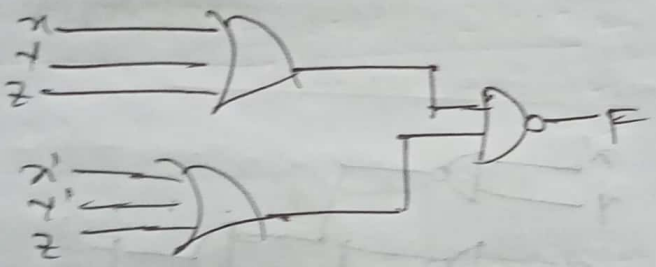
$F = x'y'z' + xyz'$

Then we take the ~~inverse~~ complement of the function:

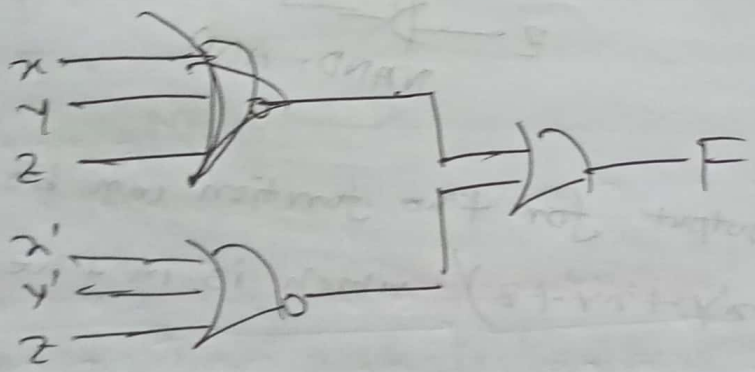
$F' = (x+y+z)(x'+y'+z)$

The normal output can now be expressed in the form:

$F = [(x+y+z)(x'+y'+z)]'$



OR-NAND



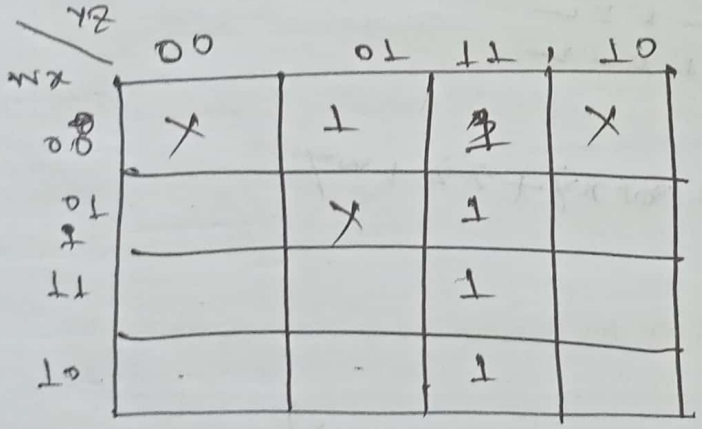
NOR-OR

Don't care condition:

The blank cells of k-map to make a group of the variables is called the "Don't care" condition. To make a group of cells, we can use the don't care cells as either 0 or 1, and if required, we can also ignore that cell. We mainly use the "don't care" cell to make a large group of cell.

Example 3.12

$F(w,x,y,z) = \Sigma(1,3,7,11,15)$, $d(w,x,y,z) = \Sigma(0,2,5)$



$\bar{w}z + yz$

Tabulation method: Tabulation method is a very useful and convenient tool for simplification of boolean function for large numbers of variables.

Example 3.13

wxyz
0 0000 ✓
1 0001 ✓
2 0010 ✓
8 1000 ✓
10 1010 ✓
11 1011 ✓
14 1110 ✓
15 1111

wxyz
0,1 000-
0,2 00-0 ✓
0,8 -000 ✓
2,10 -010 ✓
8,10 10-0 ✓
10,11 101- ✓
10,14 1-10 ✓
11,15 1-11 ✓
14,15 111- ✓

wxyz
0,2,8,10 -0-0
0,8,2,10 -0-0
10,11,14,15 1--1
10,14,11,15 1--1

prime implipicants $F = \bar{w}x\bar{y} + \bar{x}\bar{z} + wy$

Example 3-14

$F(w,x,y,z) = \Sigma(1,4,6,7,8,9,10,11,15)$

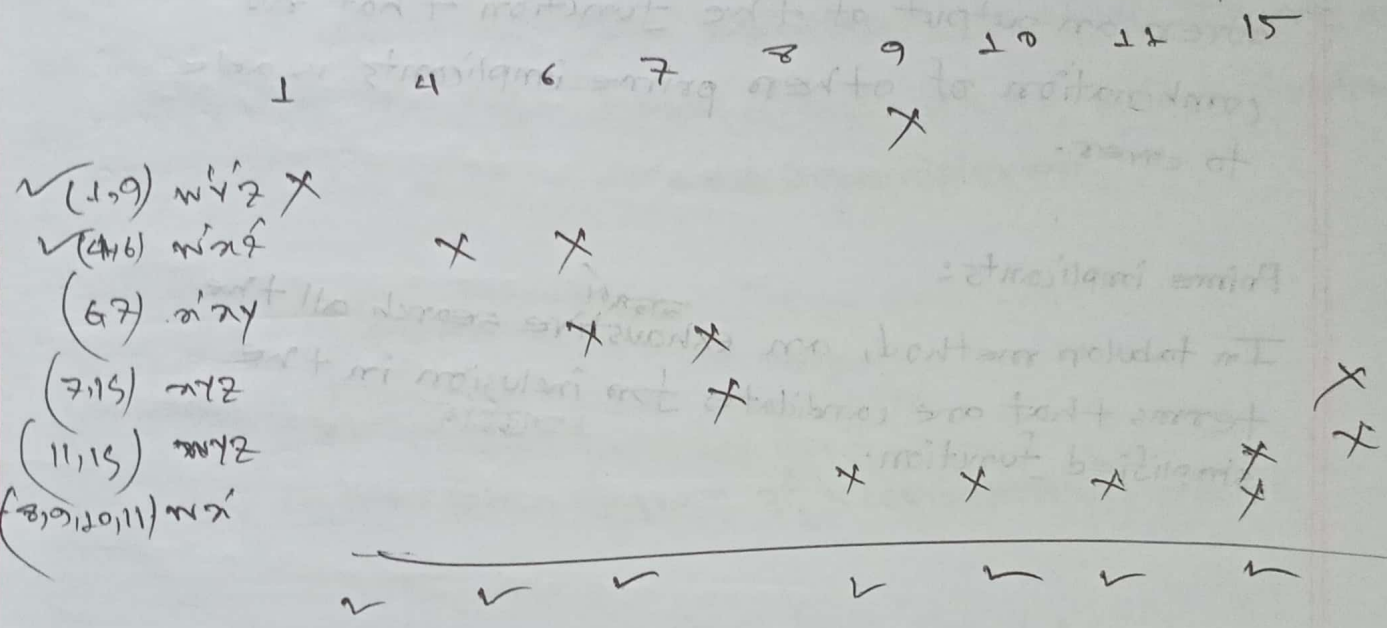
wxyz
1 0001 ✓
4 0100 ✓
8 1000 ✓
6 0110 ✓
9 1001 ✓
10 1010 ✓
7 0111 ✓
11 1011 ✓
15 1111 ✓

wxyz
1,9 -001
4,6 01-0
8,9 100- ✓
8,10 10-0
6,7 011-
9,11 10-1 ✓
10,11 101- ✓
7,15 -111
11,15 1-11

wxyz
8,9,10,11 10--
8,9,10,11 10--

$F = x'y'z' + w'xz' + xyz + wx'$

Selection of Prime implicants:



$\therefore F = x'y'z + w'xz' + wx' + xyz$

* Prime ~~and~~ implicants and Essential Implicants

Essential prime implicants:

Essential prime implicants are prime implicants that cover an output of the function that no combination of others prime implicants is able to cover.

Prime implicants:

In tabular method, an exhaustive search all the terms that are candidates for inclusion in the simplified function.

Chapter-4

Combinational logic

What is combinational circuit ?

The combinational logic circuits are the circuits that contain different types of logic gates. A combinational circuit consists of input variables, logic gates and output variables. There are different types of combinational circuit such as Adder, Subtractor, Decoder, Encoder, multiplexer and Demultiplexer.

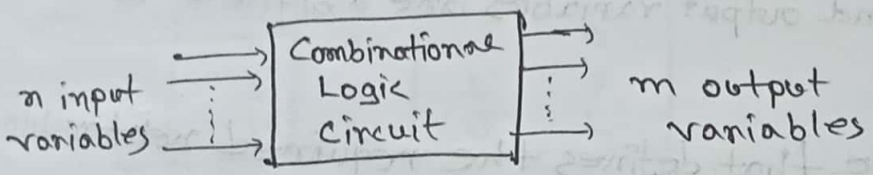


Figure : Block diagram of a combinational circuit.

Characteristics of the combinational logic circuit :

- At any instant of time, the output of the combinational circuits depends only on the present input terminals.
- The combinational circuit doesnot have any backup or previous memory. The present state of the circuit is not affected by the previous state of the input.
- The n number of inputs and m number of outputs are possible in combinational logic circuit.

Design procedure

The designs of combinational circuits starts from the verbal outline of the problem and ends in a logic circuit diagram.

(1) The problem is stated

(2) The number of available input variables and required output variables is determined.

(3) The input and output variables are assigned letter symbols.

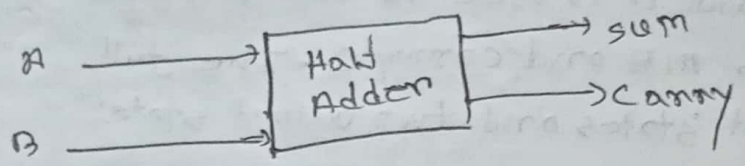
(4) The truth table that defines the required relationships between inputs and outputs is derived.

(5) The simplified Boolean function for each output is obtained.

(6) The logic diagram is drawn.

Half Adder

The half adder is a basic building block of adding two numbers as two input and produce two outputs. The adder is used to perform OR operation of two single bit binary numbers. The augent and addent bits are two input states, any 'carry' and 'sum' are two output states of the half adder.



Truth Table

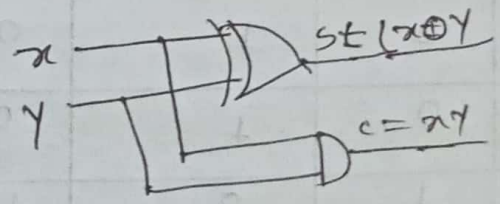
Inputs		Outputs	
A	B	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

In the above table, A, and B two inputs and sum, carry two outputs.

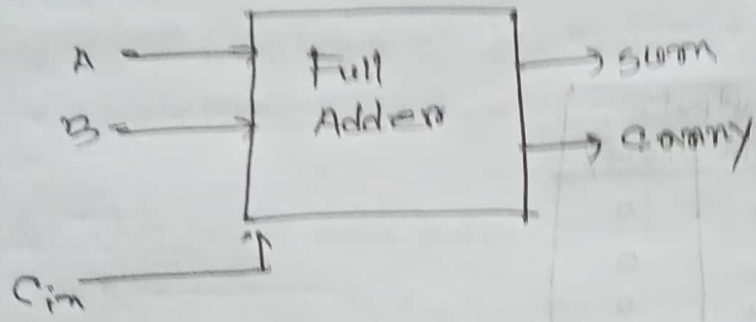
$$Sum = x'y + xy' = x \oplus y$$

$$Carry = xy$$

The logic diagram for the implementation of the half adder is



Full adder : The full adder is used to add three 1 bit binary numbers A, B and carry C. The full adder has three input states and two output states i.e sum and carry.



Consider the three input are x, y, z and two outputs are s and c

The characteristic truth table is

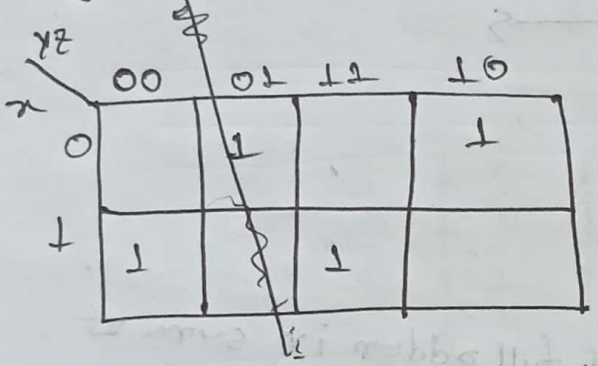
Input			Output	
x	y	z	s	c
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

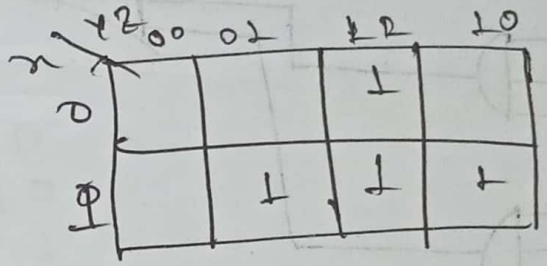
From the truth Table

$S = x'y'z + x'yz' + xy'z' + xyz$

$C = xy + xz + yz$

Using map method the most simplified method is





$S = xyz + xy'z' + x'yz' + x'y'z$

$C = xy + xz + yz$

$= x'(y'z + yz') + x(y'z' + yz)$

$= x'(y \oplus z) + x(y \odot z)$

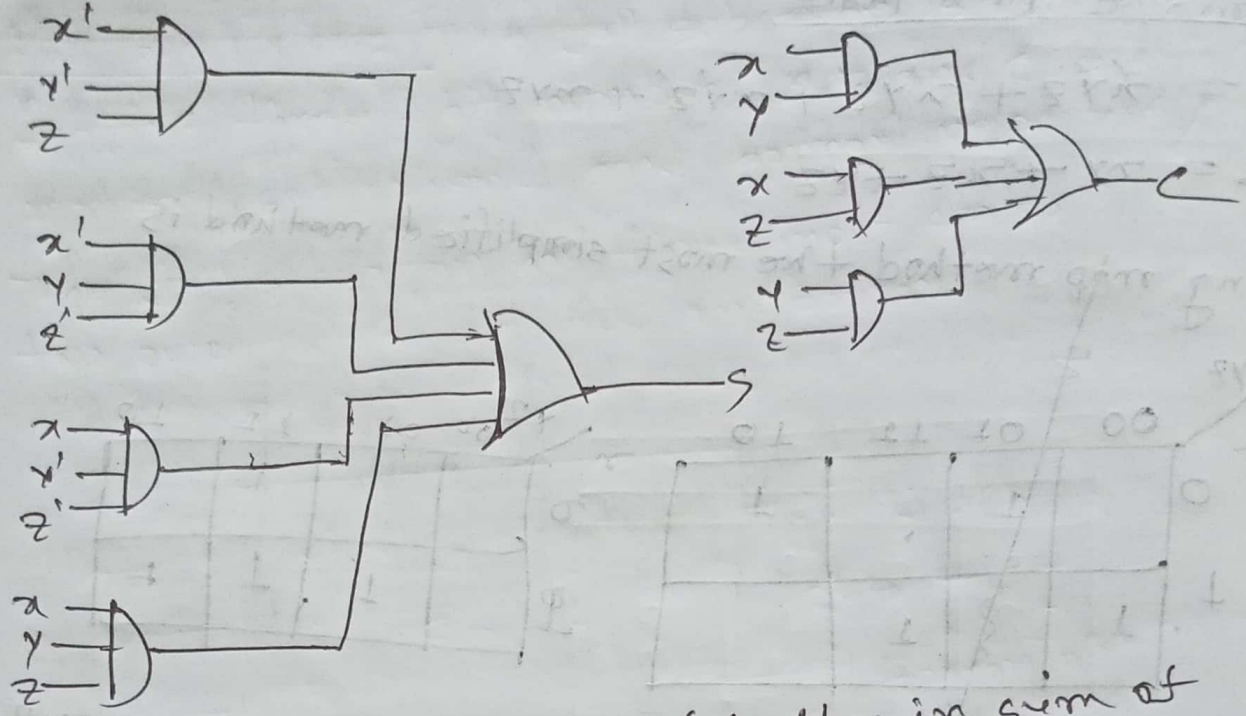
$= x'(y \oplus z) + x(\overline{y \oplus z})$

$= x \oplus y \oplus z$

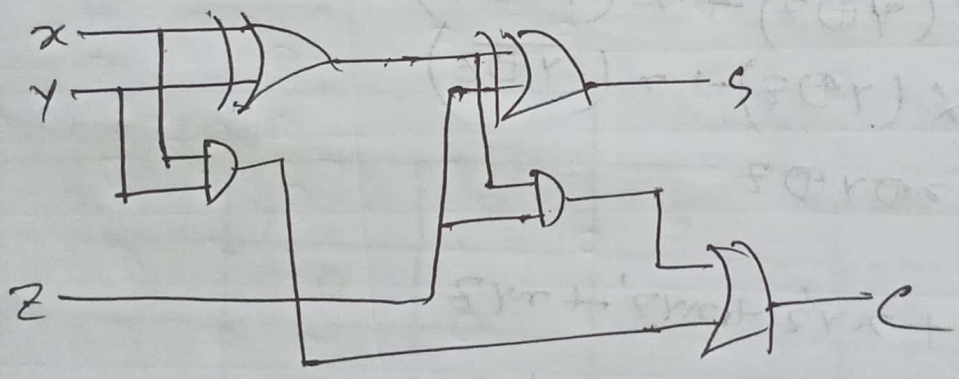
$C = x'yz + xy'z + xyz' + xyz$

$= z(x'y + xy') + xy(z' + z)$

$= z(x \oplus y) + xy$



Implementation of full adder in sum of products.



Implementation of a full adder with two half adder and an OR gate.

Subtractors:

The subtraction of two binary numbers may be accomplished by taking the compliment of the subtrahend and adding it to the minuend.

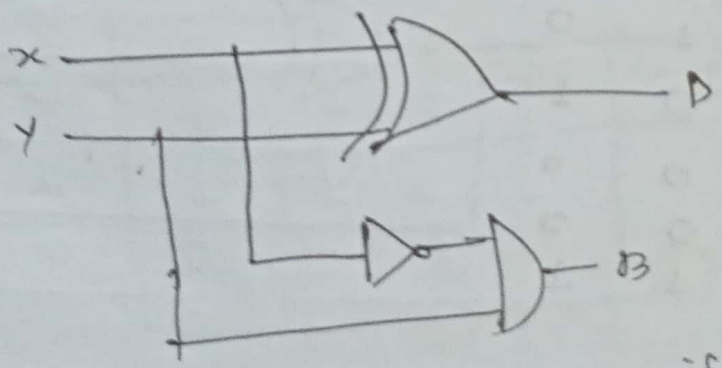
Half subtractor : A half subtractor is a combinational circuit that subtracts two bits and produce their difference.

x	y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

$D = x'y + xy'$

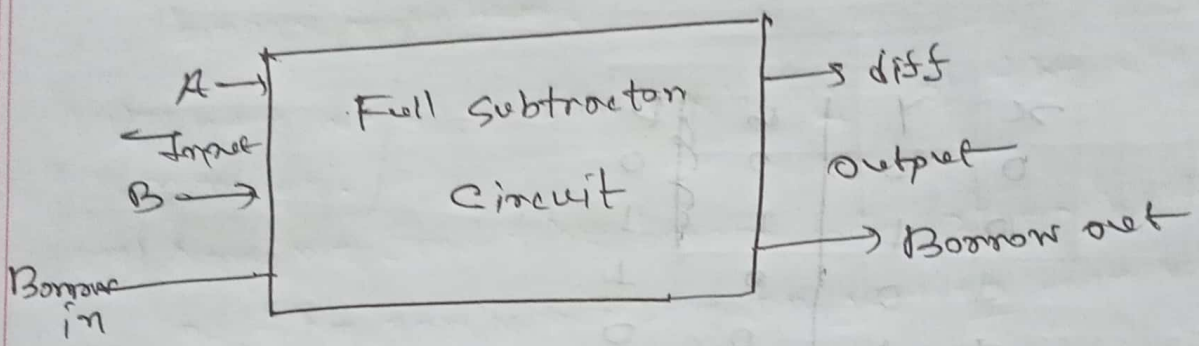
$= x \oplus y$

$B = x'y$



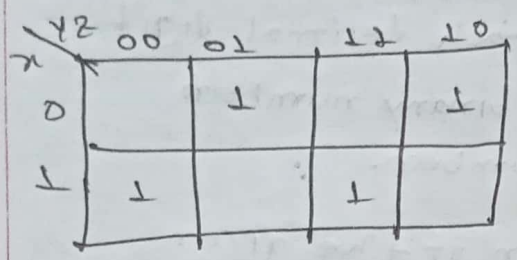
Half subtractor circuit.

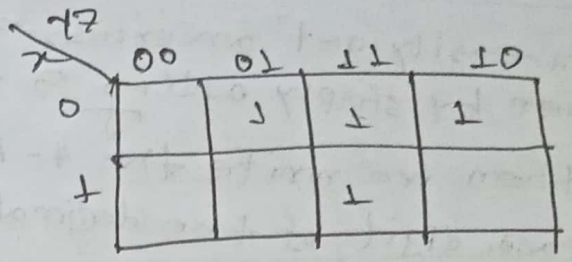
Full Subtractor: The full subtractor is used to subtract three 1 bit numbers A, B and C, which are minuend, subtrahend and borrow respectively. The full subtractor has three input states and two output states i.e diff and borrow



x	y	z	B	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Simplification using map





$D = x'y'z + x'yz + xy'z' + xyz$

$B = yz + x'z + x'y$

=

তারপর Diagram draw করতেহবে।

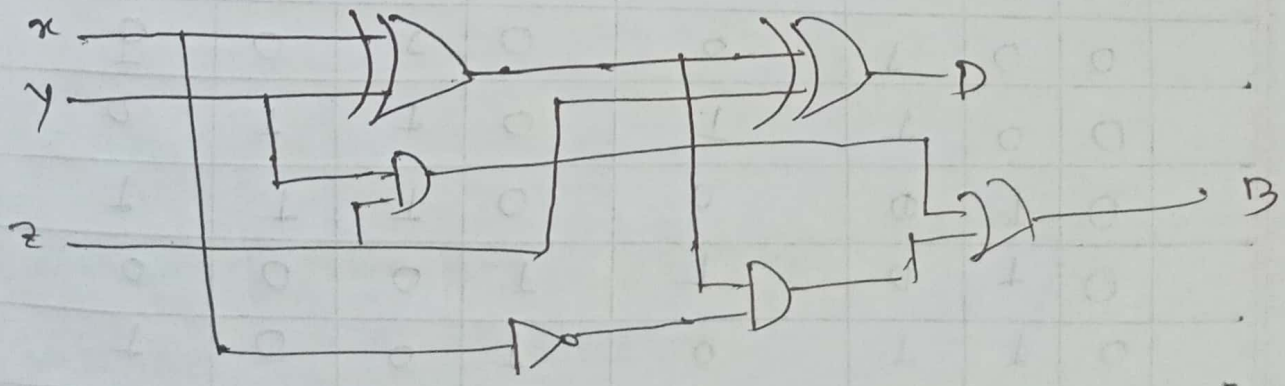
এখন গেটাকে যদি Exclusive - or এর মাধ্যমে represent করতে চাই

$D = x \oplus y \oplus z$ [সূত্র থেকে]

$B = x'y'z + x'yz' + x'yz + xyz$

$= x'(y'z + yz') + yz(x + x')$

$= x'(y \oplus z) + yz$



Full subtractor.

BCD to Excess 3-code conversion

We can easily get an excess-3 code of a decimal number by simply adding 3 to each decimal digit And then we write the 4-bit binary numbers for each digit of the decimal number.

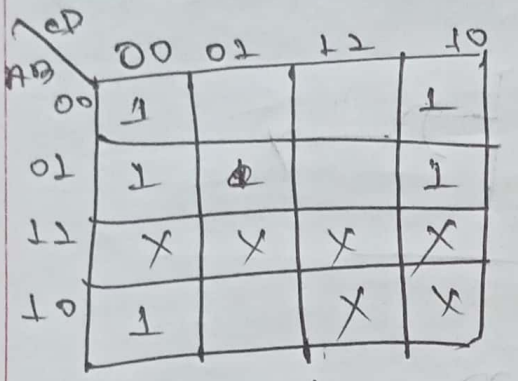
(i) We find the decimal number of the given binary number.

(ii) Then we add 3 in each digit of the decimal number.

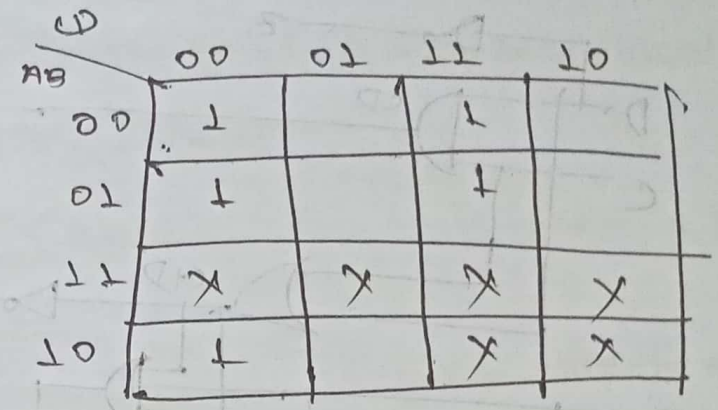
(iii) Now, we find the binary code of each digit of the newly number for getting excess 3-code

Input BCD				Output excess-3 code			
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

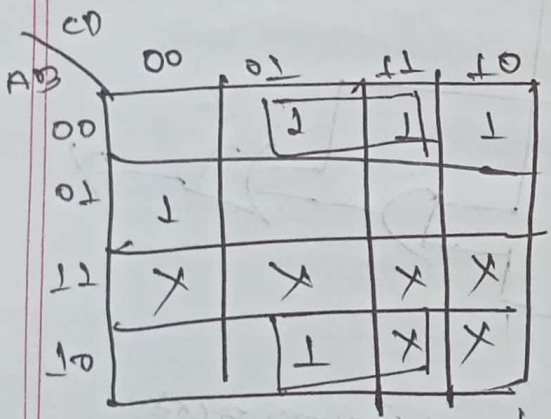
We note that four binary variables may have 16 bit combinations only 10 are listed in truth table and six bit are don't care condition.



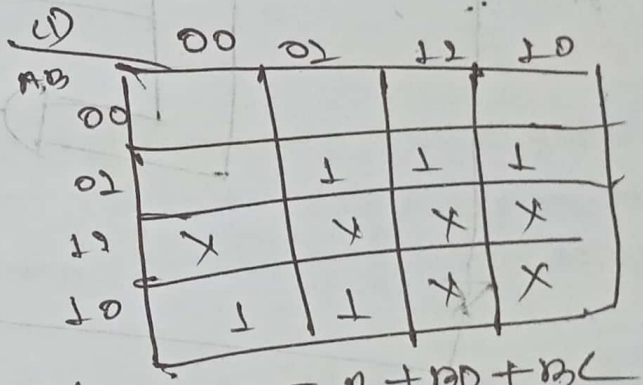
$Z = D'$



$Y = CD + C'D'$



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



$W = A + BD + BC$

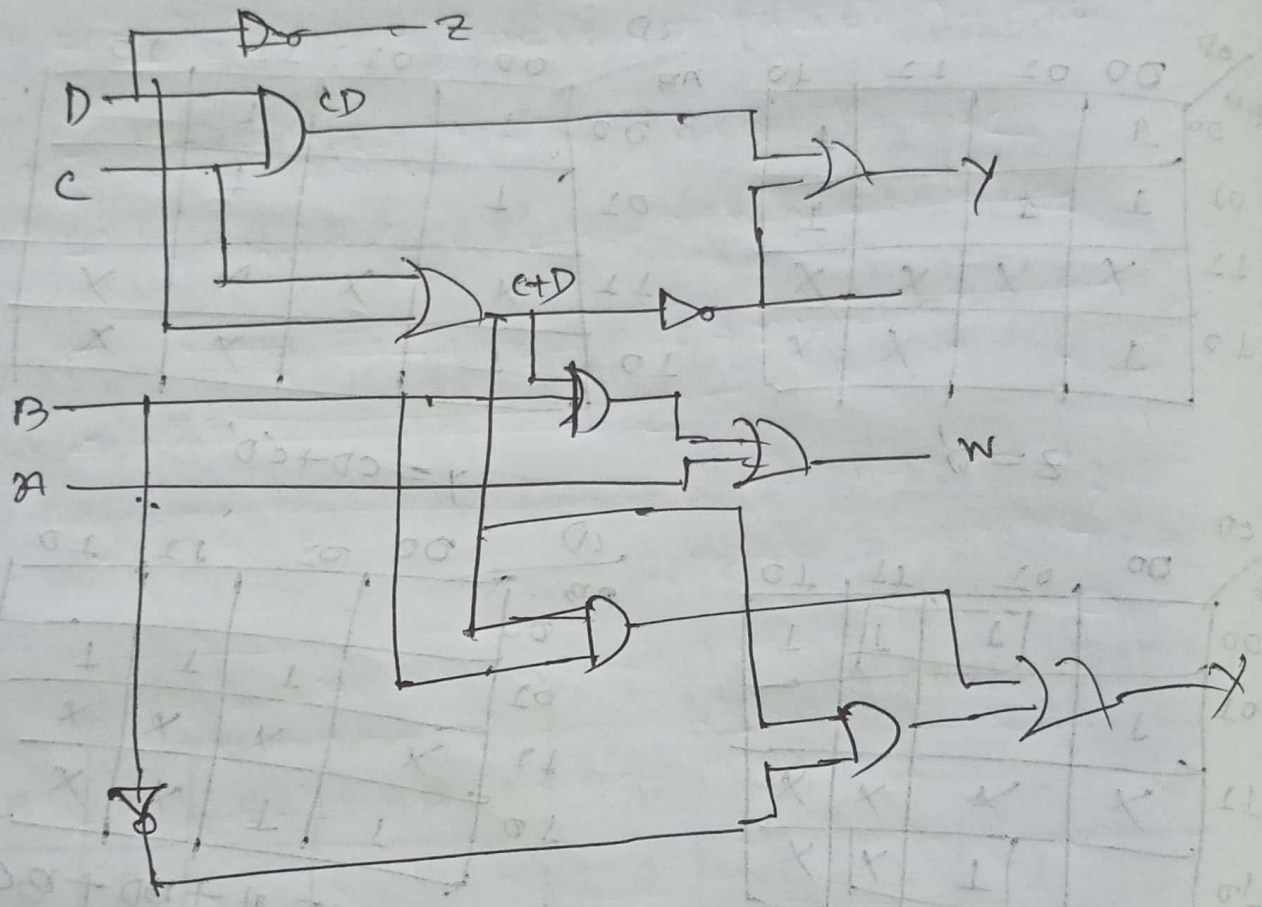
$Z = D'$

$Y = CD + C'D' = CD + (C+D)'$

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

$W = A + BC + BD = A + B(C+D)$

For getting common gate



Logic diagram for BCD - to excess 3 code converter.

SIMRIF

Analysis procedure: The design of a combinational circuit starts from the verbal specifications of a required function and culminates with a set of output Boolean functions or a logic diagram. The analysis of a combinational circuit is somewhat the reverse process.

To obtain the output Boolean functions from a logic diagram:

1. Label with arbitrary symbols all gate outputs that are a function of the input variables. Obtain the Boolean functions for each gate.

2. Label with other arbitrary symbols those gates which are a function of input variables and/or previously labeled gates. Find the Boolean functions for these gates.

3. Repeat the process outlined in step-2 until the outputs of the circuit are obtained.

4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables only.

From Fig 4.9

$F_2 = AB + BC + AC$

$T_1 = A + B + C$

$T_2 = ABC$

$T_3 = F_2' T_1$

$F_1 = T_2 + T_3$

Now

$F_1 = T_2 + T_3$

$= F_2' T_1 + ABC = (AB+BC+AC)'(A+B+C) + ABC$

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

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

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

$= A'BC + A'BC + A'BC + ABC$

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

$(A+B+C) + ABC$

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

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

Universal gate: The NAND gate is said to be a universal gate because any digital system can be implemented with it.

Any boolean function can be implemented with NAND gates, we need only show that the logical operations AND, OR and NOT can be implemented with NAND gates.

A convenient way to implement a combinational circuit with NAND gates is to obtain the simplified

Boolean functions in terms of AND, OR and NOT and convert the functions to NAND logic by De Morgan's Theorem.

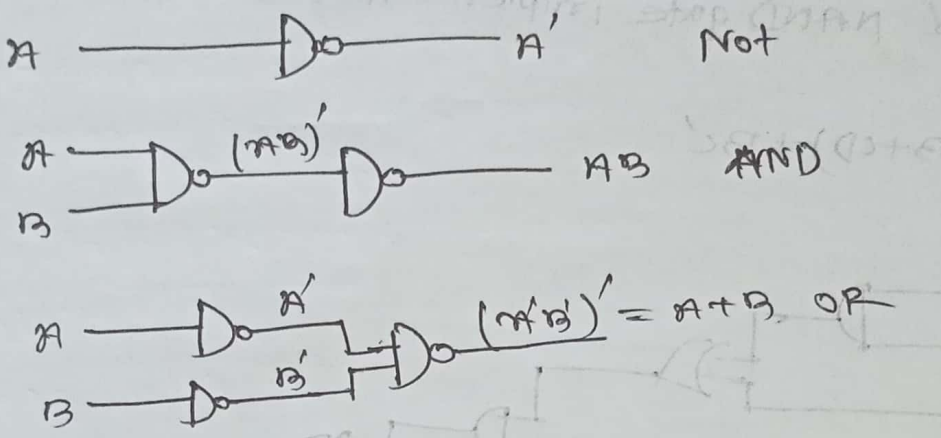


Figure: Implementation of NOT, OR, AND by NAND gate

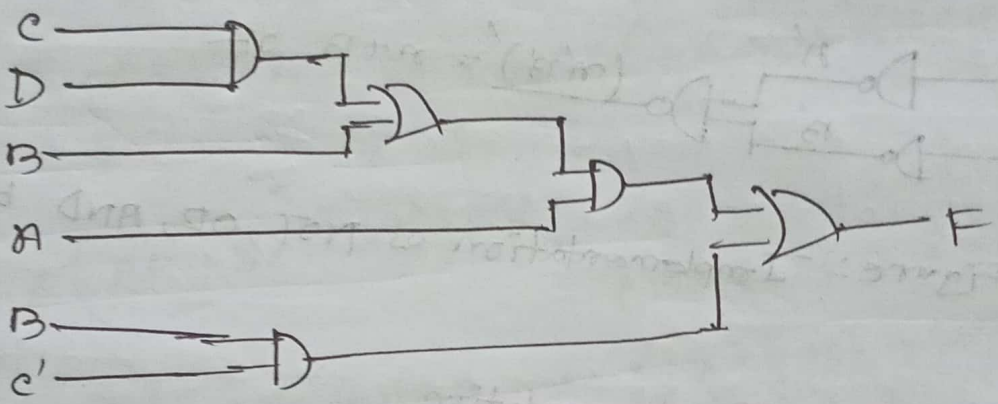
Boolean function Implementation
Block Diagram Method

Procedure (In case of NAND)

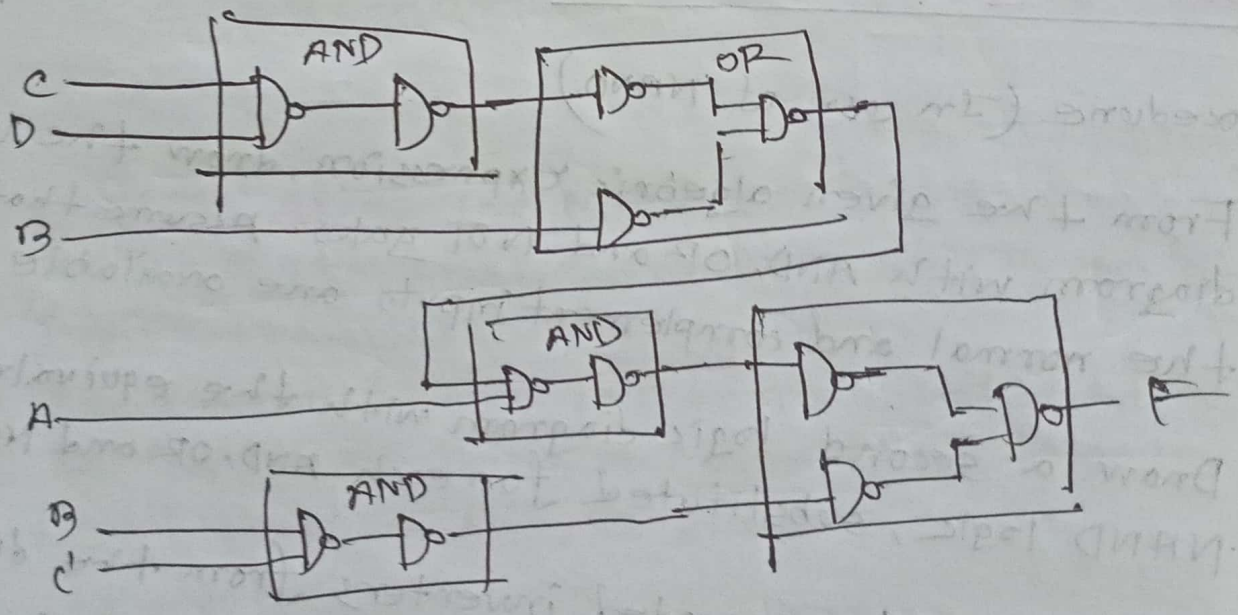
1. From the given algebraic expression, draw the logic diagram with AND, OR and NOT gates. Assume that both the normal and complement inputs are available.
2. Draw a second logic diagram with the equivalent NAND logic, substituted for each AND, OR and NOT gate.
3. Remove any two cascaded inverters from the program since double inversion does not perform a logic function. Remove inverters connected to single external inputs

and complement the corresponding input variable
The new logic diagram obtained is the
required NAND gate implementation.

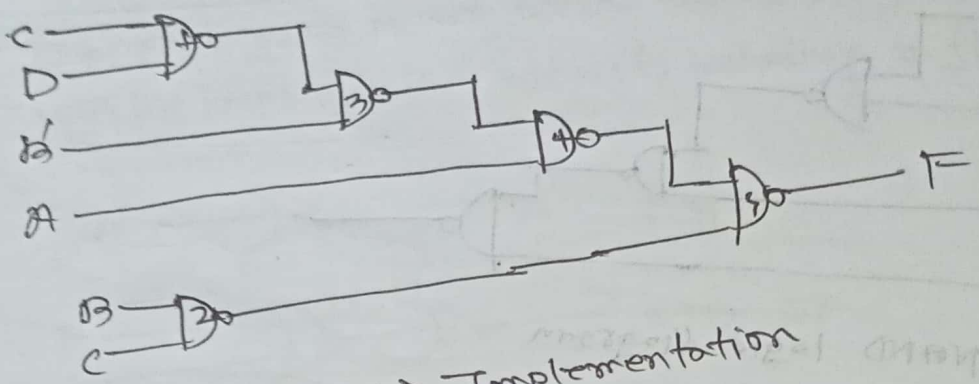
$F = A(B+CD)+BC'$



(a) AND/OR Implementation



(b) substituting equivalent NAND functions



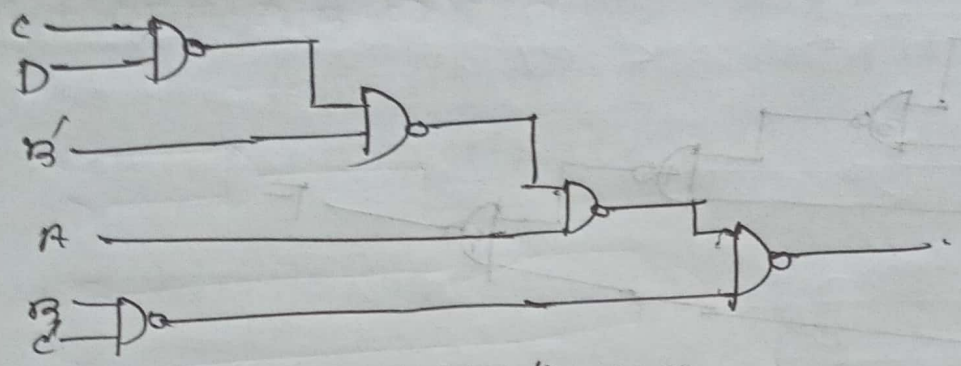
NAND Implementation

Block Diagram Transformation :

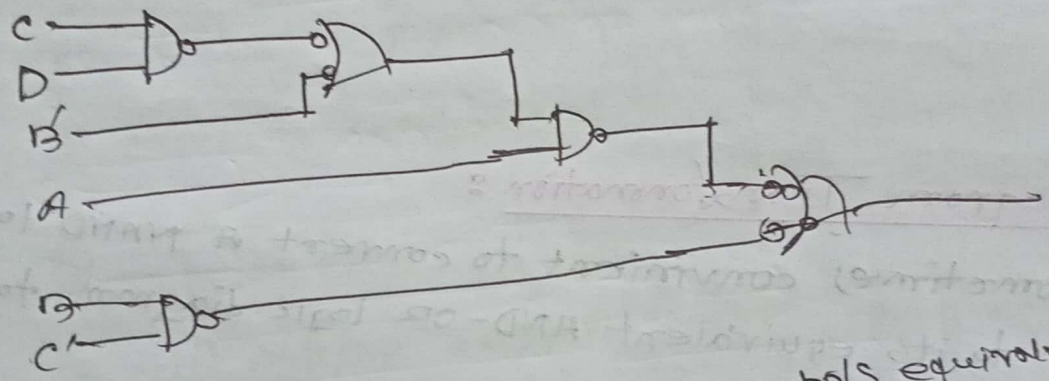
It is sometimes convinient to convert A NAND logic diagram to its equivalent AND-OR logic diagram to facilitate the analysis procedure.

The conversion of A NAND logic diagram to An AND-OR diagram is acheived through a change in symbols from AND invert to invert OR. The first level to be changed to an invert OR symbol should be the last level. These changes produce pairs of circles along the same line, and these can be removed since they represent double complementation.

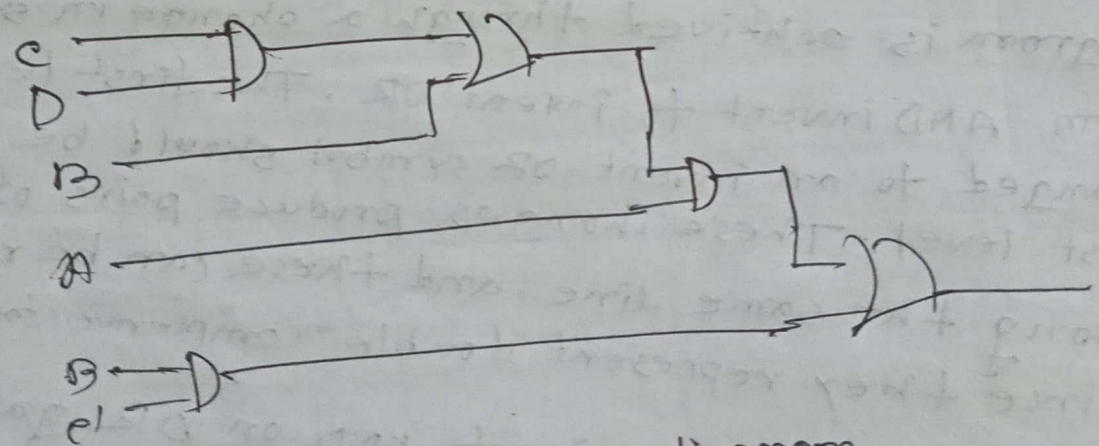
Moreover, a one input AND or OR gate can be removed since it does not perform A logical function. A one input AND/OR with a circle in the input/output is changed to an inverter circuit.



NAND logic diagram



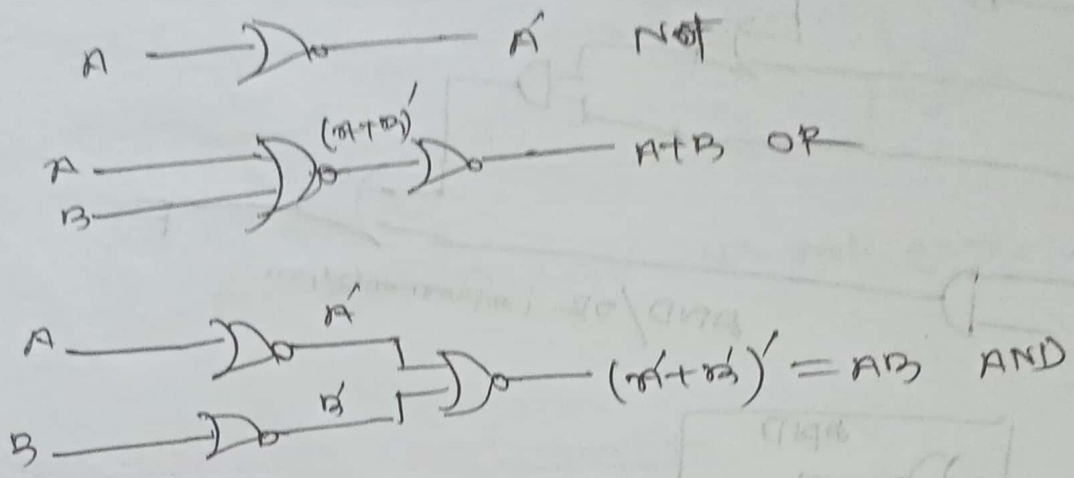
Substitution of invert OR symbols equivalent



AND-OR logic diagram

Universal Gate (NOR)

The NOR gate is universal because any Boolean function can be implemented with it, including a flip-flop circuit.



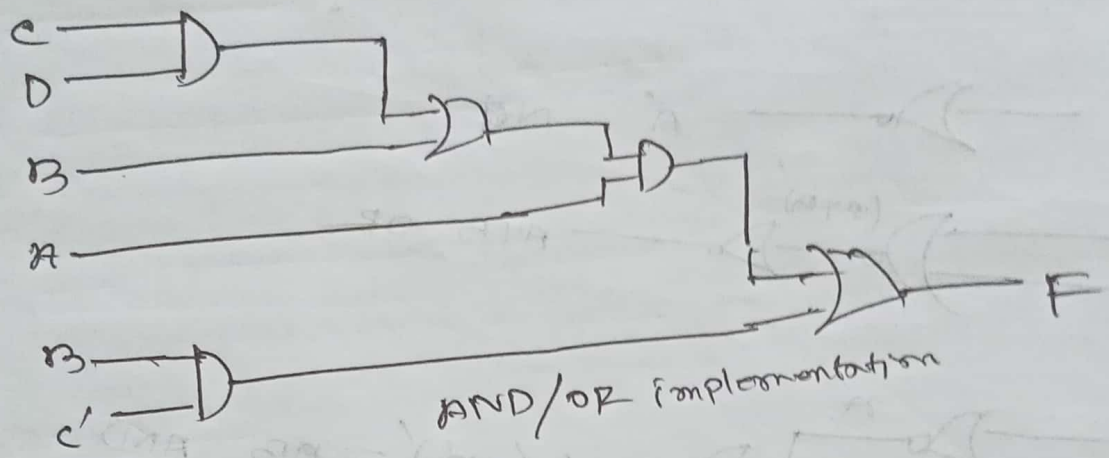
Boolean function Implementation Block diagram Method:

1. Draw the AND-OR logic diagram from the given algebraic expression, Assume that both the normal and the complement inputs are available.

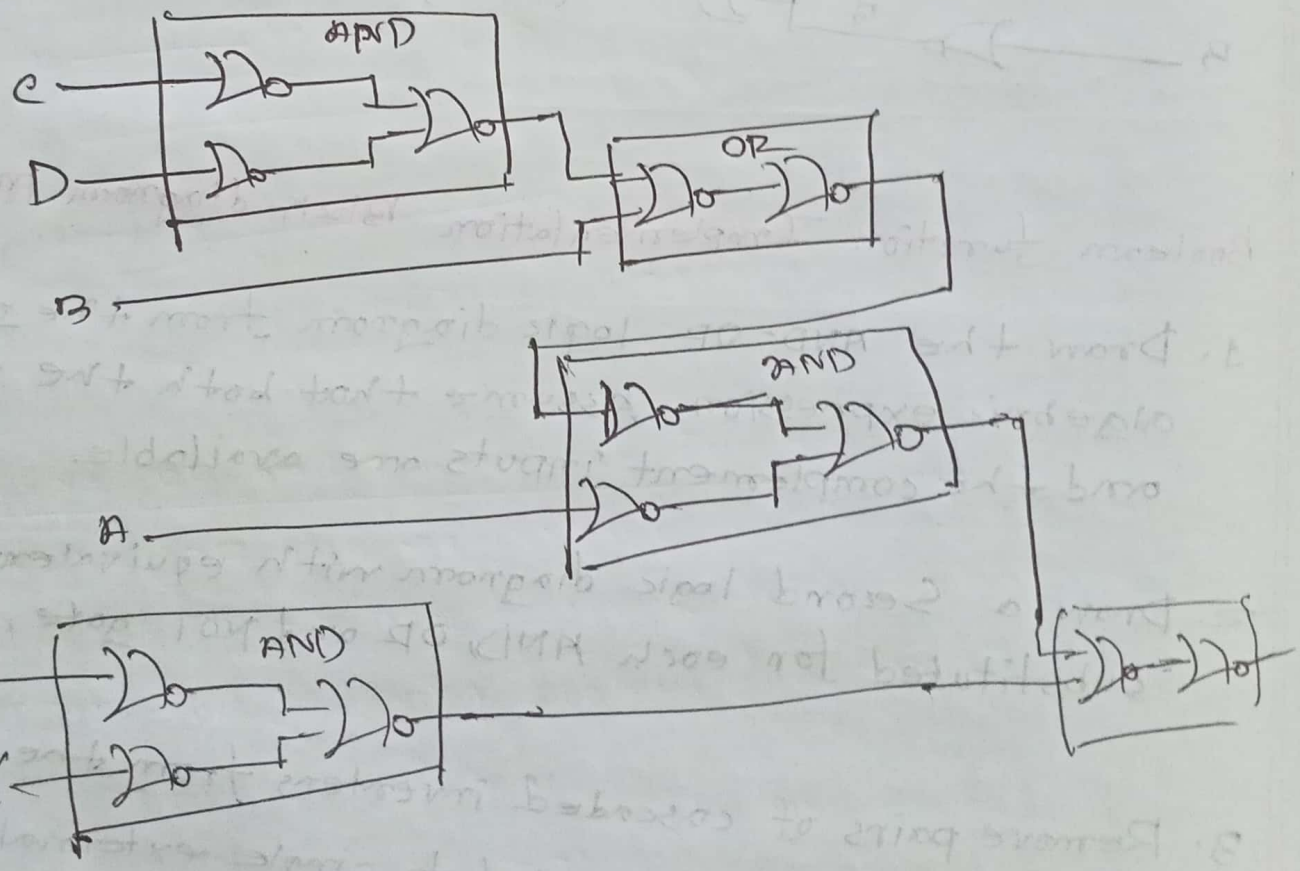
2. Draw a second logic diagram with equivalent NOR logic substituted for each AND, OR and NOT gate.

3. Remove pairs of cascaded inverters from the diagram. Remove inverters connected to single external inputs and complement the corresponding input variable.

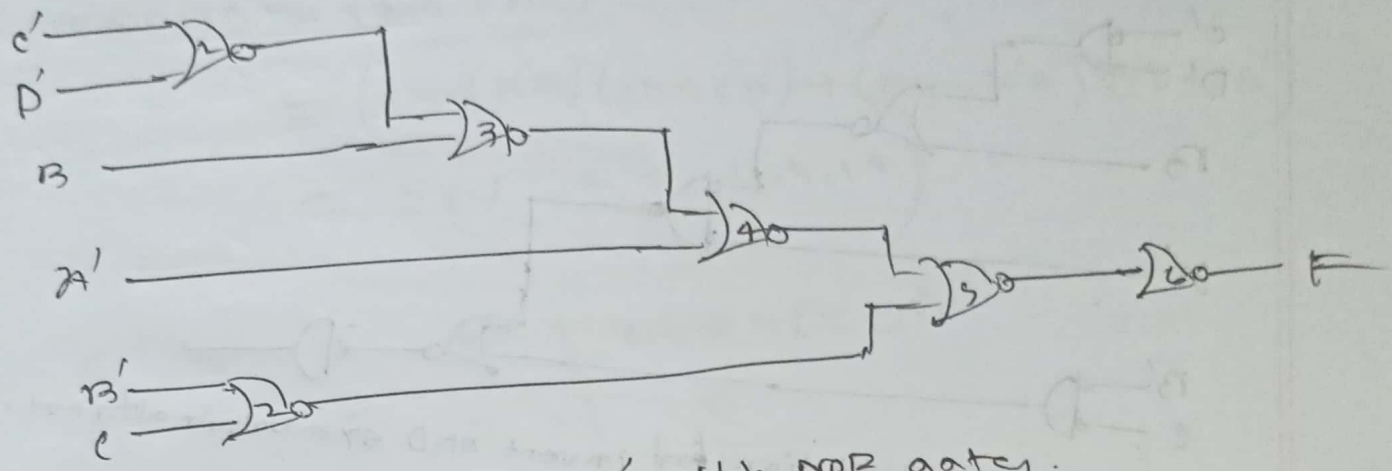
$F = A(B + CD) + BC'$



AND/OR implementation



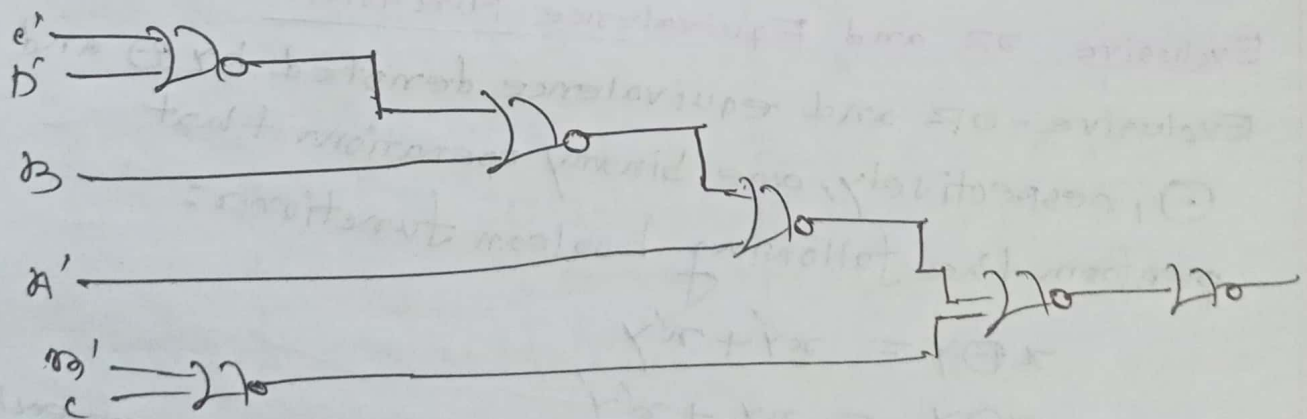
Substituting equivalent NOR



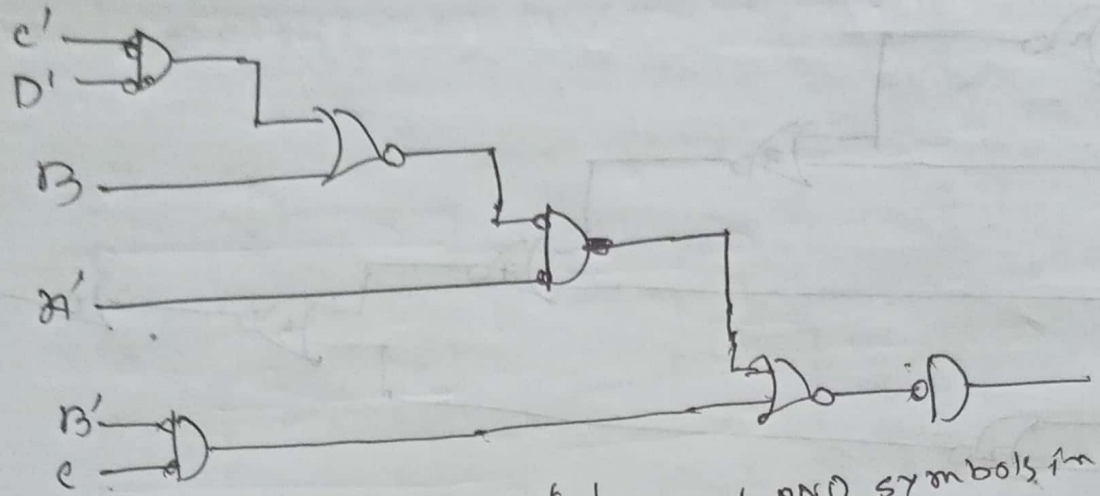
$F = A(B + CD) + BC'$ with NOR gates.

Block diagram Transformation

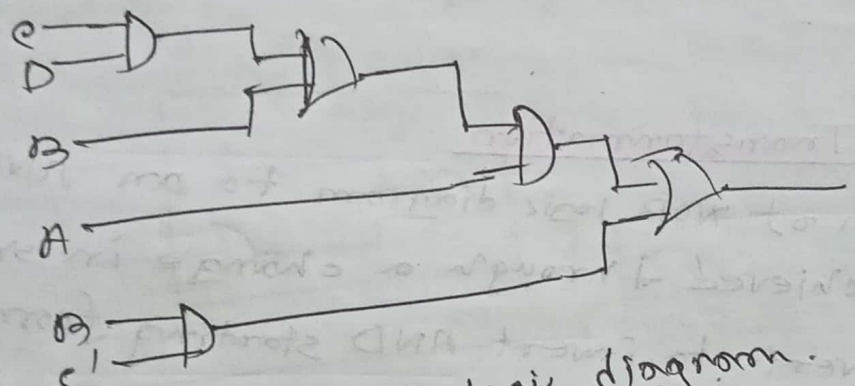
The conversion of NOR logic diagram to an AND-OR diagram is achieved through a change in symbols from OR invert to invert AND starting from the last level and in alternate level.



NOR logic diagram



Substitution of invert AND symbols in alternate levels



AND-OR logic diagram.

Exclusive OR and Equivalence Functions:

Exclusive-OR and equivalence denoted by $\oplus$ and $\odot$, respectively, are binary operations that perform the following boolean functions:

$$x \oplus y = xy' + x'y$$

$$x \odot y = xy + x'y'$$

The two operations are the complements of each other.

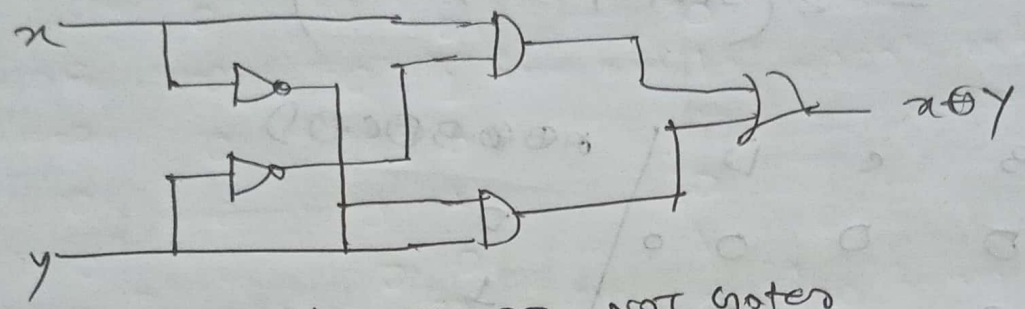
$$A \oplus B \oplus C \oplus D = (AB' + A'B) \oplus (CD' + C'D)$$

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

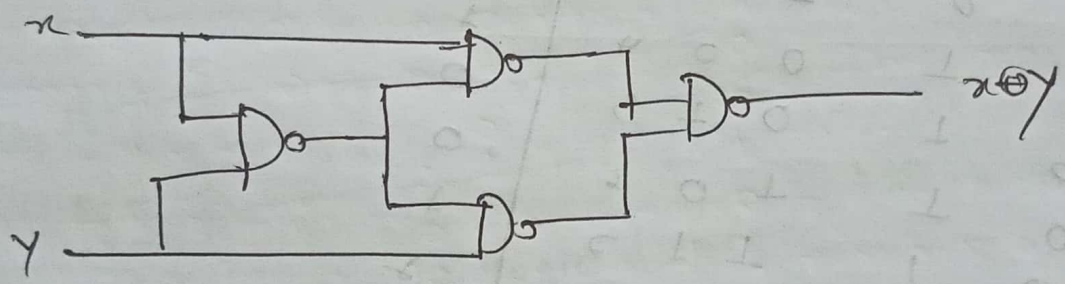
$$= \Sigma(1, 2, 4, 7, 8, 11, 13, 14)$$

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

যদি 0 even number হয় তবে $w \oplus x \oplus y \oplus z$



with AND-OR-NOT Gates



with NAND Gate

Parity Generator and Parity Checker:

- Parity bit is an extra bit included with a message to make the total number of 1's transmitted either odd / even.

* The circuit that generates the parity bit in the transmitter is called a parity generator.

* The circuit that checks the parity in the reciever is called a parity checker.

SHARIF

Two types of parity

Even Parity ⟶ Even number 1's

odd Parity ⟶ odd number 1's

* Even Parity Generator for 3 bit:

A	B	C	parity generator (P)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

even এর ক্ষেত্রে যেসব বিজোড় সংখ্যক 1 আছে ইনপুটে তাদের জোড় বানানোর জন্য output 1 দিয়েছি

odd parity generator for 3 bit

A	B	C	parity generator
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

এখানে ইনপুটে বিজোড় বানানোর জন্য আউটপুটে 1 টি ০ ও 1 দিতে হবে

parity cheaml: (3 bit)

A	B	C	P	even cheaker	odd cheaker
0	0	0	0	0	1
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	1

even number 1 থাকলে ডেটা বিকৃত

odd number 1 থাকলে ডেটা বিকৃত

SHARIF

Combinational logic with MSI and LSI

A binary parallel adder: A binary parallel adder is a digital function that produces the arithmatic sum of two binary numbers in parallel. It consists of full adders connected in cascade, with the output carry from one full-adder connected to the input carry of the next full adder.

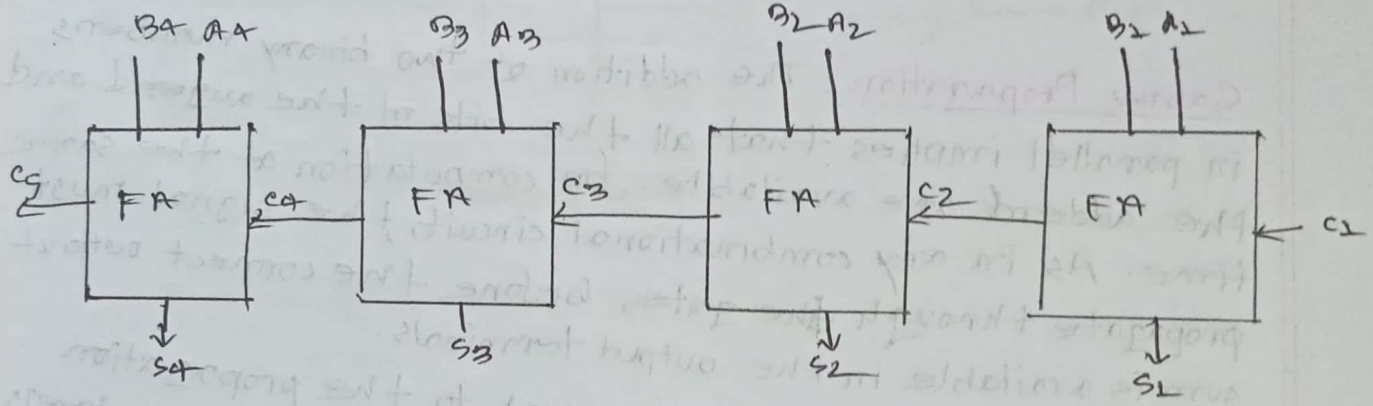


Figure: 4-bit full adders.

Example 5.1 Design a BCD-to-excess-3 code converter

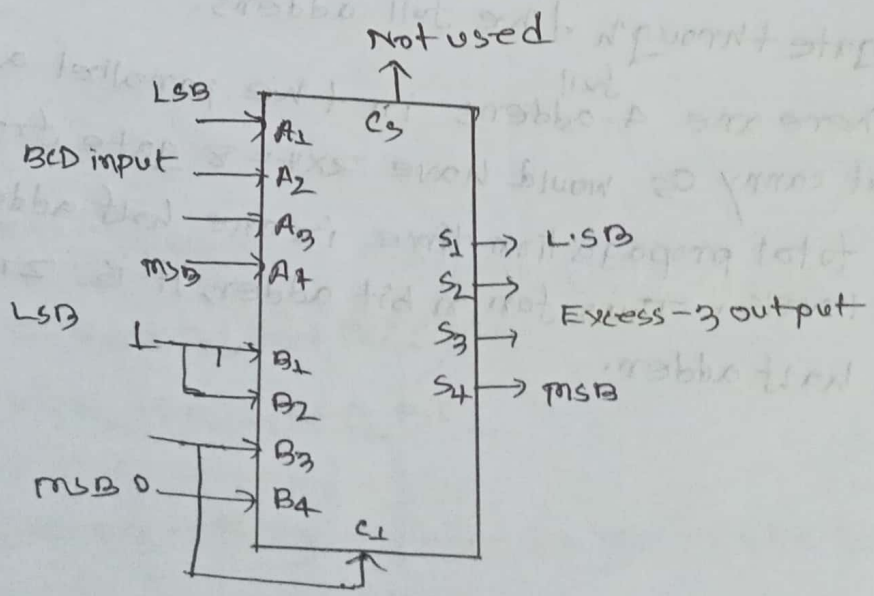
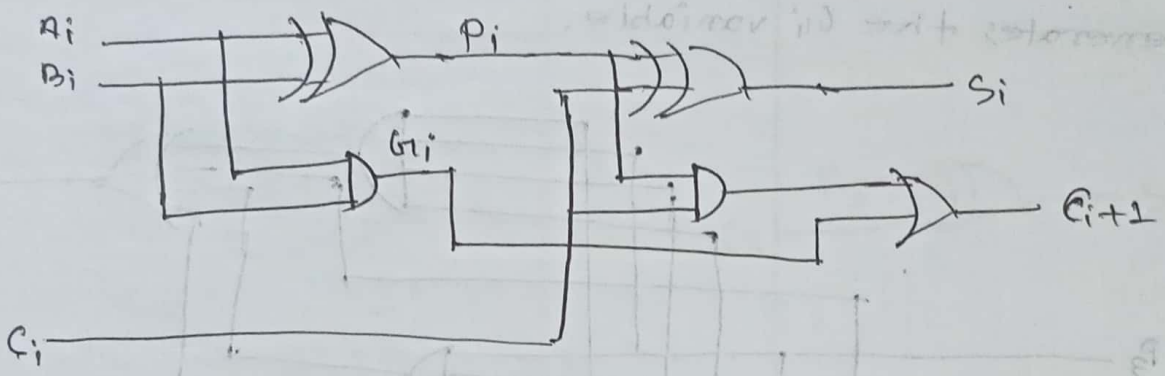


Figure: BCD to excess 3 code converter.

This figure is the 4-bit full adders MSI circuit. The BCD digit applied to inputs A. Inputs B are set set to a constant 0011. This is done by applying logic 1 to B_1 and B_2 and logic-0 to B_3 and B_4 and C_1. Logic-1 and logic-0 are physical signals whose values depend on the IC logic family used.

Canny Propagation: The addition of two binary numbers in parallel implies that all the bits of the augend and the addend are available for computation at the same time. As in any combinational circuit, the signal must propagate through the gates before the connect output sum is available in the output terminals.
The total propogation time is equal to the propagation delay of a typical gate times the number of get levels in the circuit. The longest propagation delay in a parallel adder is the time it takes the canny to propagate through the full adders.
If there are 4-full addens in the parallel adder, the output canny C_5 would have $2 \times 4 = 8$ gate from C_1 to C_5.
The total propagation time in one half adder plus eight gate levels. Thus for n bit adder, it is $2n+$ time in one half adder.

Look ahead canny: The most widely used technique employs the principle of look ahead canny. Consider the circuit.



Full adder circuit

$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$

The output sum and canny can be expressed as.

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_i C_i$$

Let $i=1$

$$C_2 = G_1 + P_1 C_1$$

let $i=2$

$$C_3 = G_2 + P_2 C_2$$

$$= G_2 + P_2 (G_1 + P_1 C_1)$$

$$= G_2 + P_2 G_1 + P_2 P_1 C_1$$

$i=3$,

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1$$

The combination of 4-bit parallel adder with a look-ahead carry scheme in shown figure. Each sum output requires two exculsive-OR gates. The output of the first exclusive-OR gate generates the Pi variables and the AND gate generates the Gi variable.

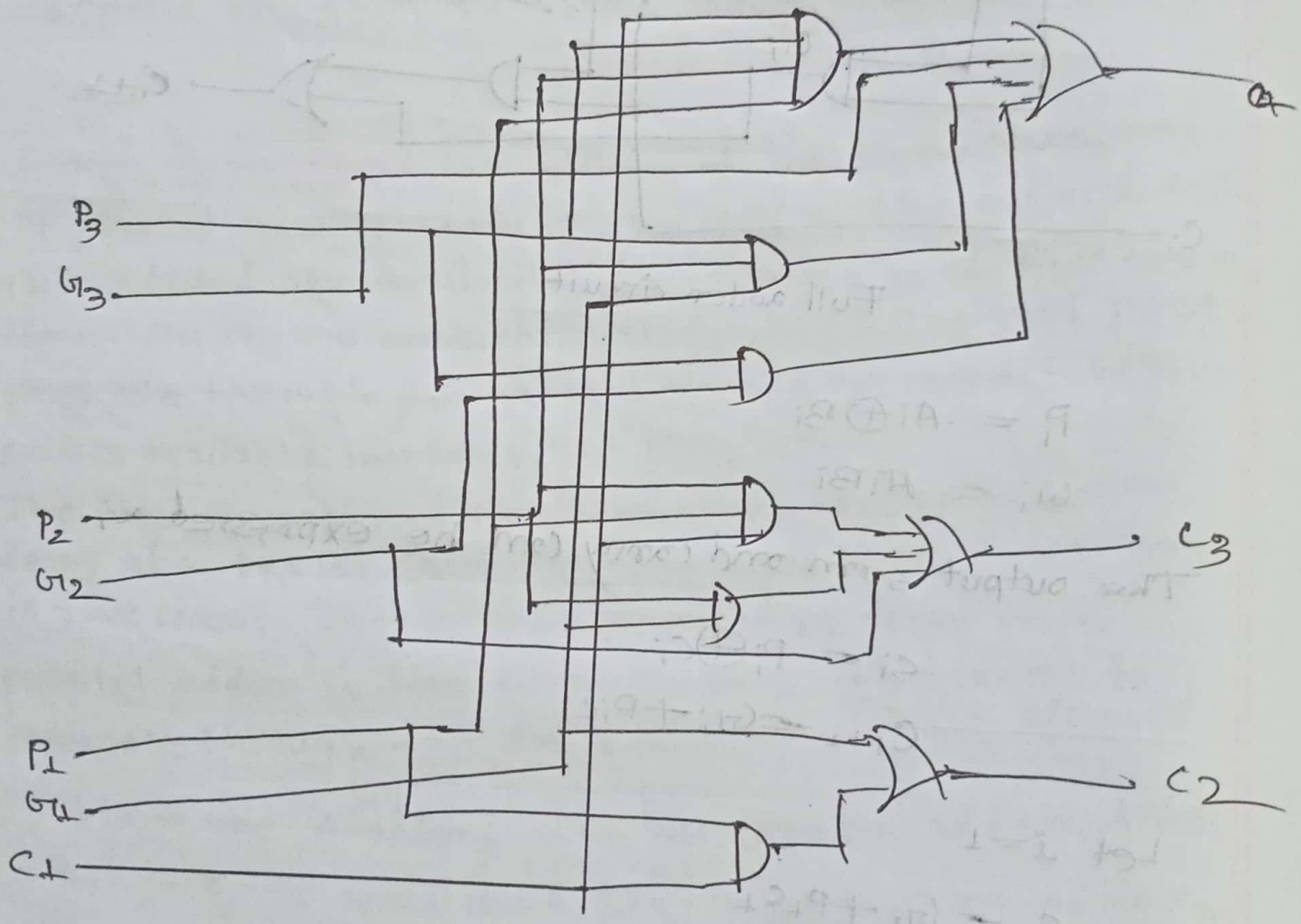


Fig: logic diagram of a look-ahead carry generator.

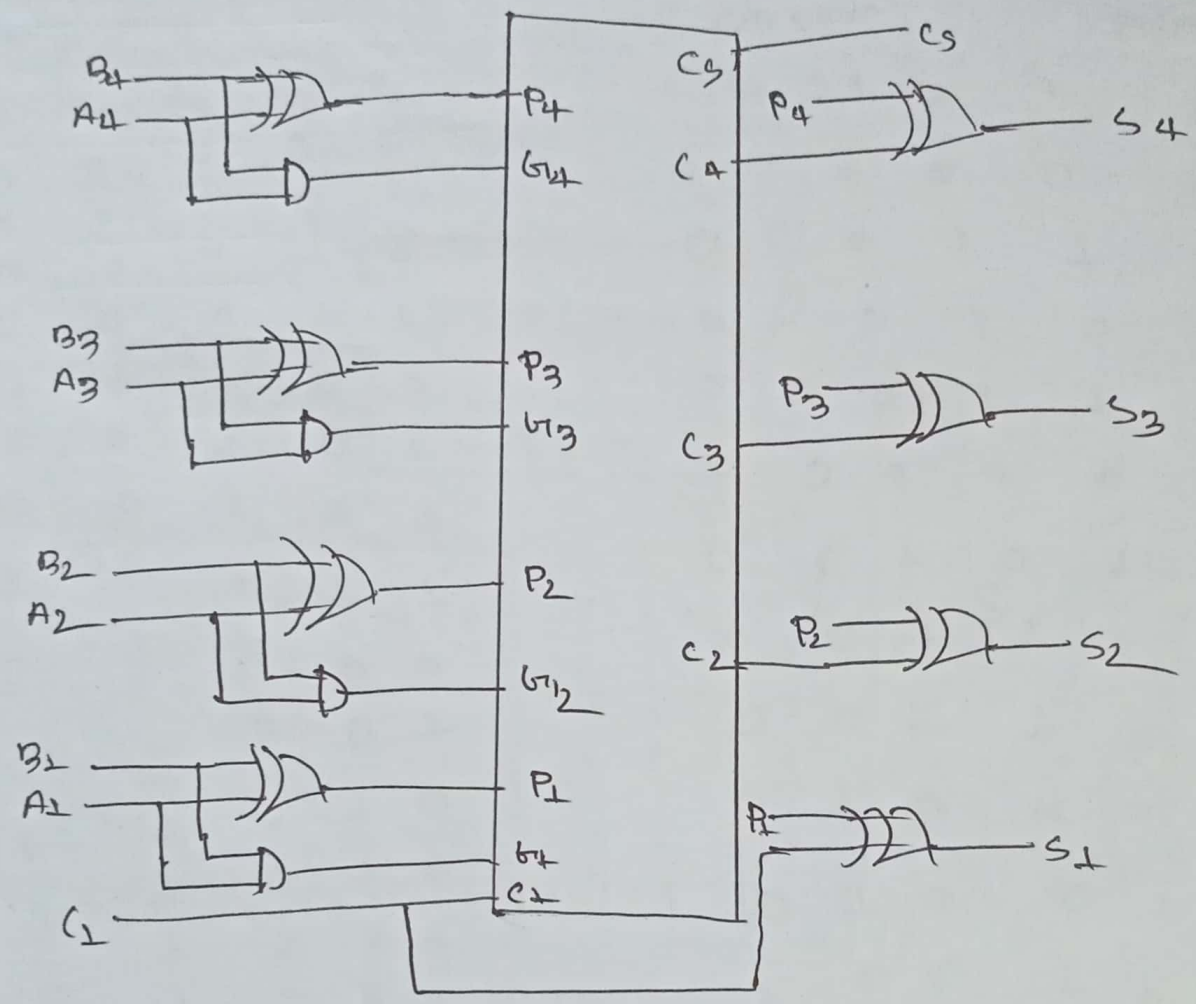


Fig: 4-bit Full addens with look ahead canny.

Q: constract a cincuit diagram of BCD adden from truth table.

BCD Adder: Consiben the anithmatic addition of decimal digits in BCD, together with a possible canny from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than 9+9+1 = 19. The 1 in the sum being an input canny.

Suppose we apply two BCD digits to a 4-bit binary adder. The adder will produce output from 0-19. These binary numbers are listed in the table:

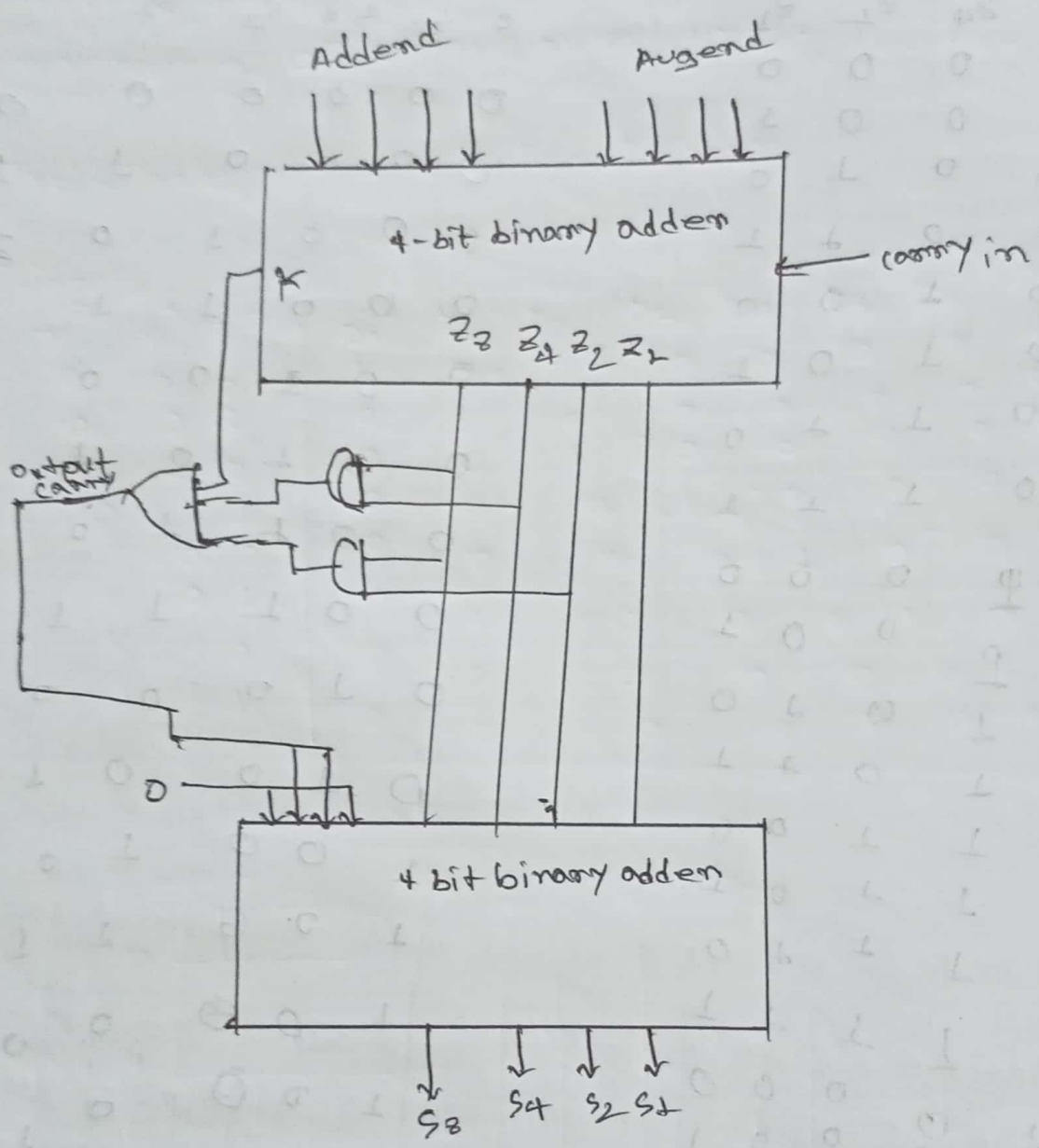
Binary sum

K	Z_8	Z_4	Z_2	Z_1
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1
0	0	1	0	0
0	0	1	0	1
0	0	1	1	0
0	0	1	1	1
0	1	0	0	0
0	1	0	0	1
0	1	0	1	0
0	1	0	1	1
0	1	1	0	0
0	1	1	0	1
0	1	1	1	0
0	1	1	1	1
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1

BCD sum

C	S_8	S_4	S_2	S_1	Decimal
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	2
0	0	0	1	1	3
0	0	1	0	0	4
0	0	1	0	1	5
0	0	1	1	0	6
0	0	1	1	1	7
0	1	0	0	0	8
0	1	0	0	1	9
1	0	0	0	0	10
1	0	0	0	1	11
1	0	0	1	0	12
1	0	0	1	1	13
1	0	1	0	0	14
1	0	1	0	1	15
1	0	1	1	0	16
1	0	1	1	1	17
1	1	0	0	0	18
1	1	0	0	1	19

Carry $c = K + Z_8Z_4 + Z_8Z_2$



Block diagram of BCD adder.

A BCD adder is a circuit that adds two BCD digits in parallel and produces a sum digit also in BCD. A BCD adder must include the connection logic in its external construction.

SHARIF

Magnitude Comparator: A magnitude comparator is a combinational circuit that compares two numbers A, B and determines their relative magnitudes. The outcome of the comparison is specified by three binary variables that indicates whether $A > B$, $A = B$, $A < B$.

$$A = A_3 A_2 A_1 A_0$$
$$B = B_3 B_2 B_1 B_0$$

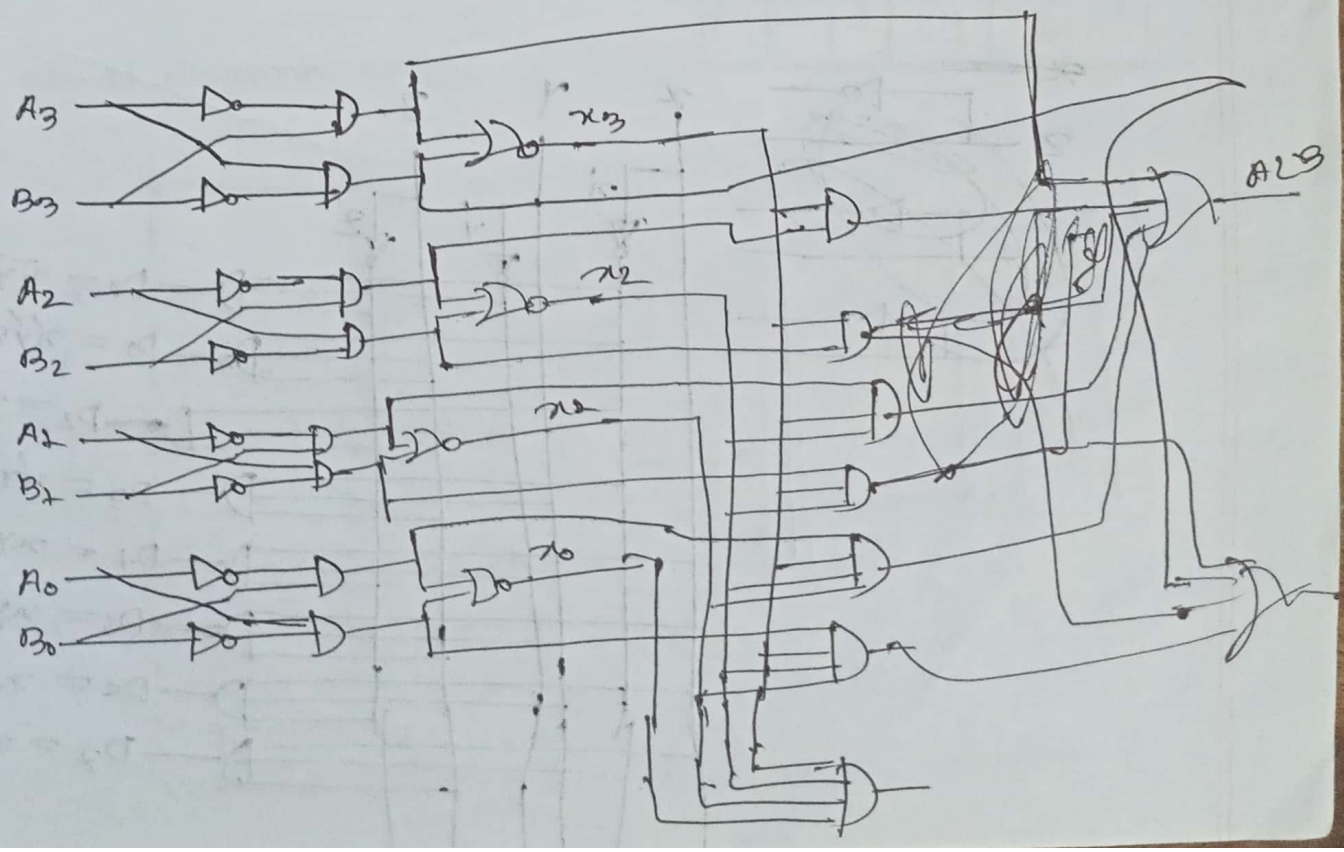
represent A an B is a bit number

if $x_i = A_i B_i + A_i' B_i'$; $A = B = x_3 x_2 x_1 x_0$

Then A and B are equal

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

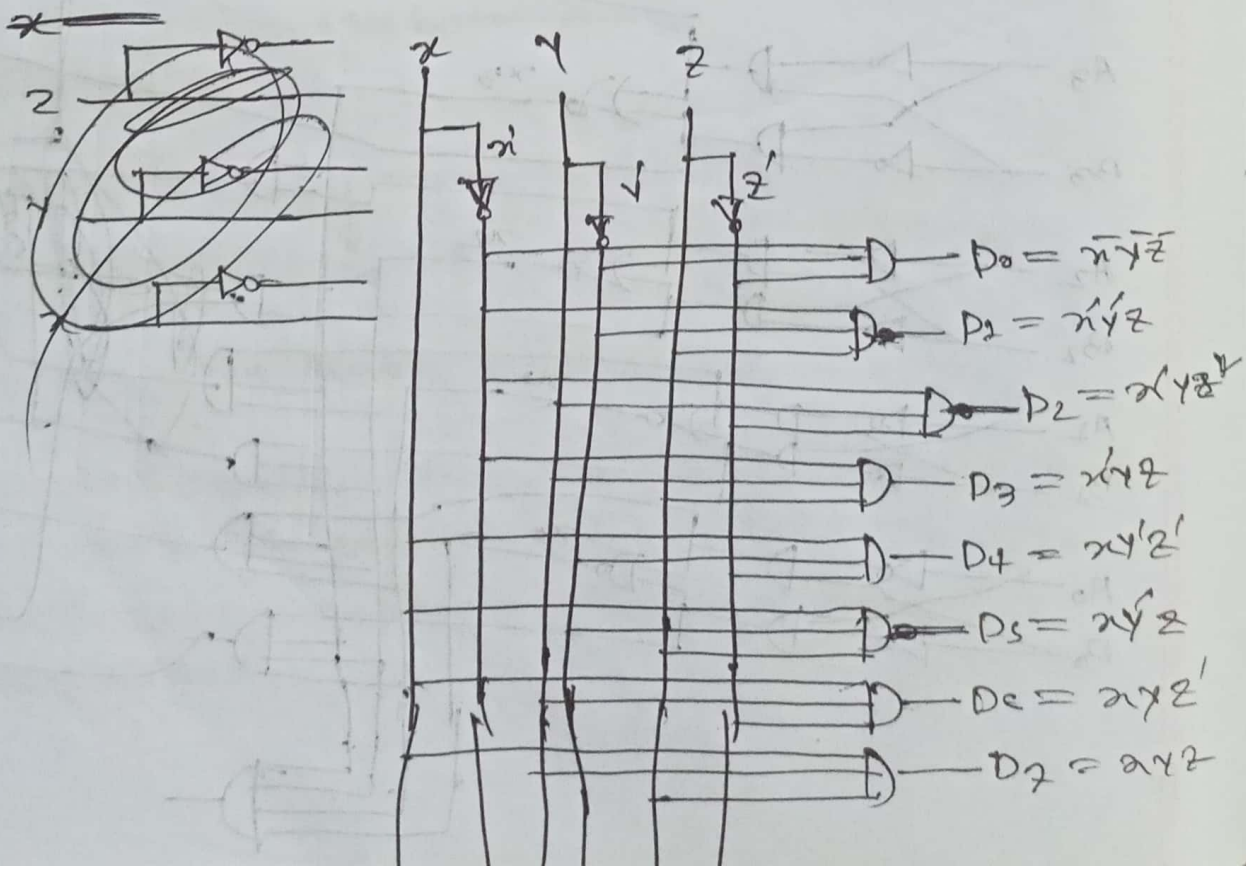
$$A < B = A_3' B_3 + x_3 A_2' B_2 + x_3 x_2 A_1' B_1 + x_3 x_2 x_1 A_0' B_0$$



Decoders: A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines.

Truth Table of 3 to 8 line Decoder

Inputs									
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



Example 5.2 বাড়ীর কাজ (If need)

Example 5-3 Implement a full adder circuit with a decoder and two OR gates.

Truth Table of full adder is

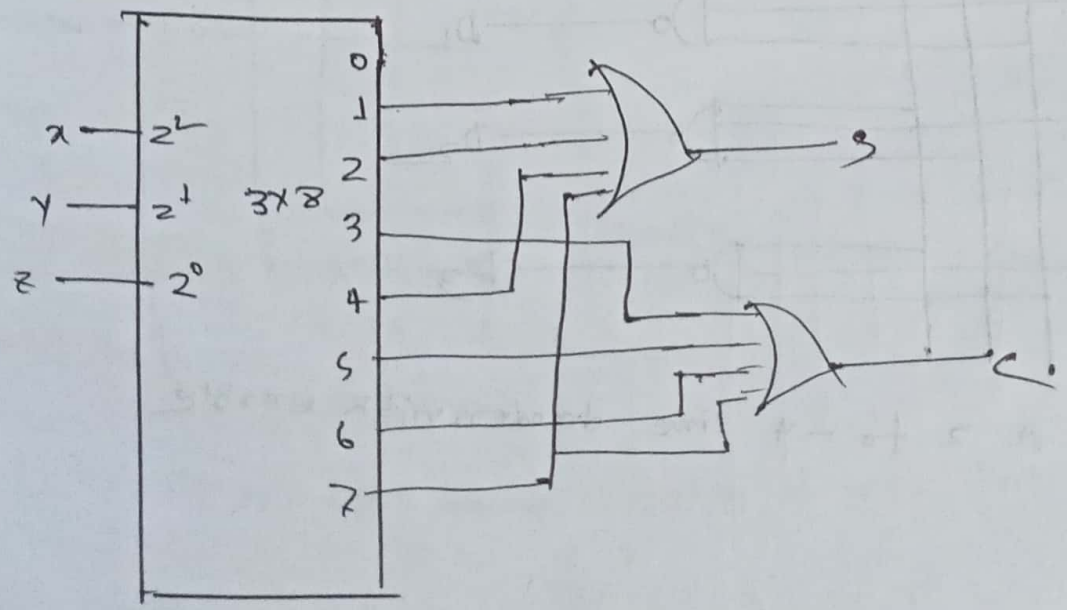
x	y	z	s	c
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

We obtain the functions for this truth table, the sum of products minterms are

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

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

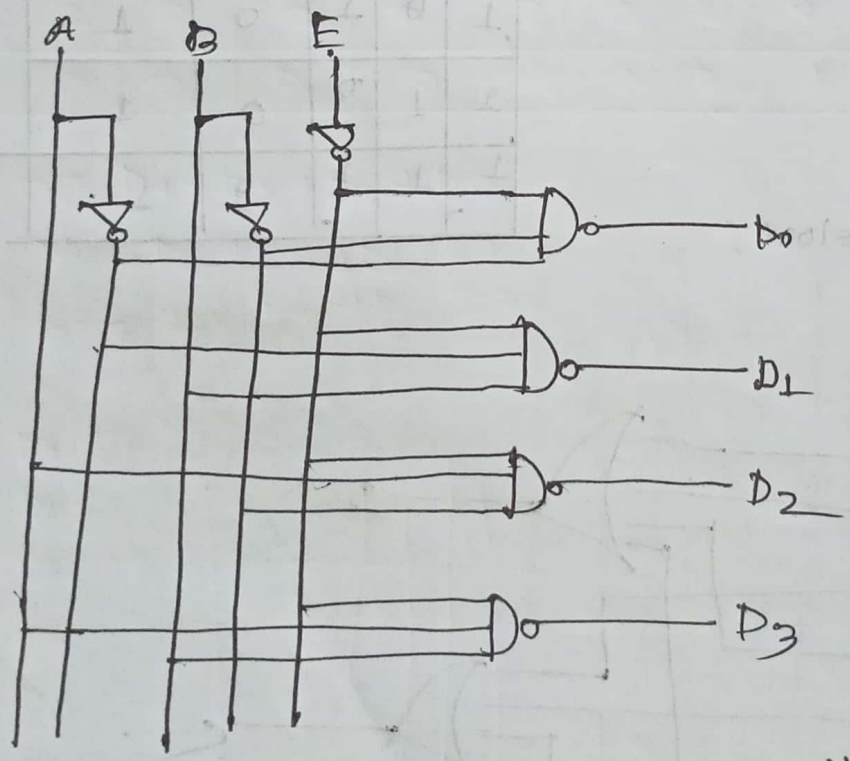
Circuit diagram below;



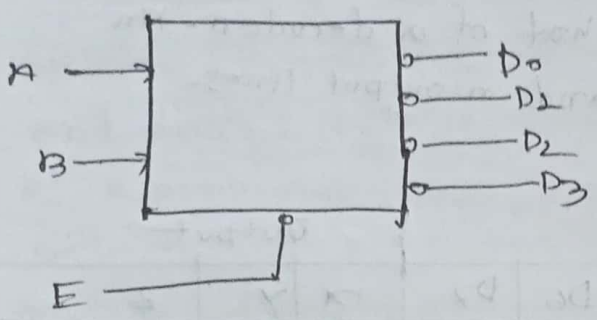
Implementation a full adder with a decoder.

Demultiplexer: A demultiplexer is a circuit that recieves information on a single line and transmits this information on one 2^n possible output lines.

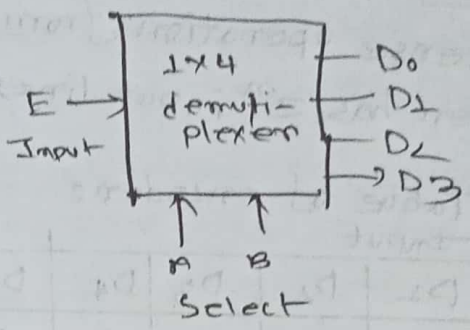
E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0



A 2 to -4 line decoder with enable

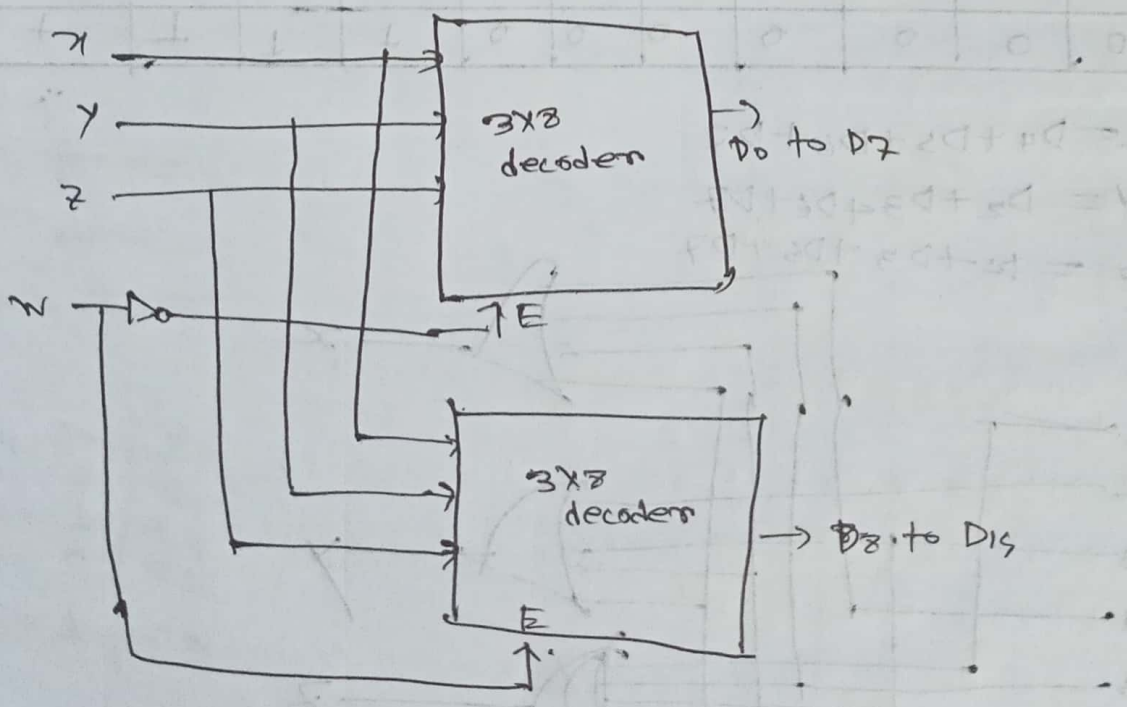


(a) Decoder with enable



(b) Demultiplexer

Q. Draw 4×16 decoder constructed with two 3×8 decoders



A 4×16 decoder constructed with two 3×8 decoders.

Encoder: A encoder is a digital function that produces a reverse operation from that of a decoder. An encoder has 2^n input lines and n output lines.

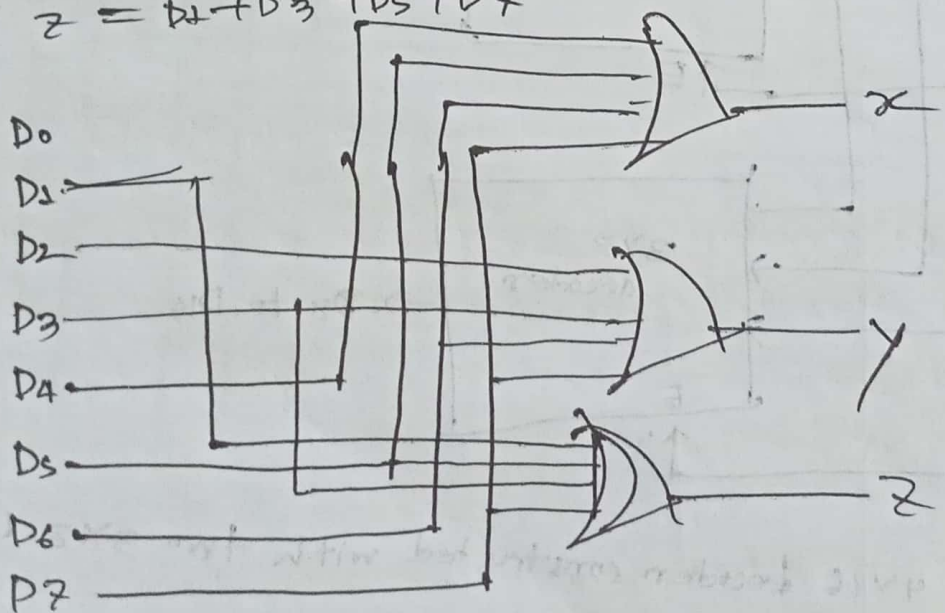
Truth Table of encoder:

Input								Output		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$x = D_4 + D_5 + D_6 + D_7$

$Y = D_2 + D_3 + D_6 + D_7$

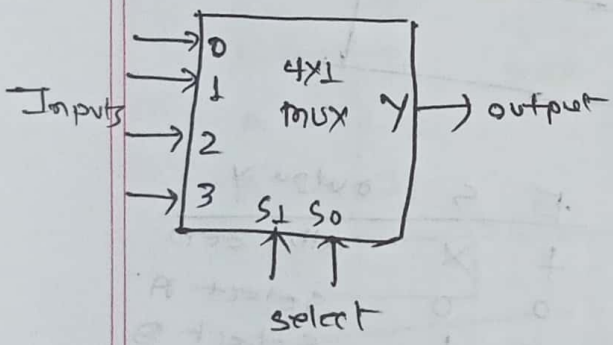
$z = D_1 + D_3 + D_5 + D_7$



octal to binary encoder.

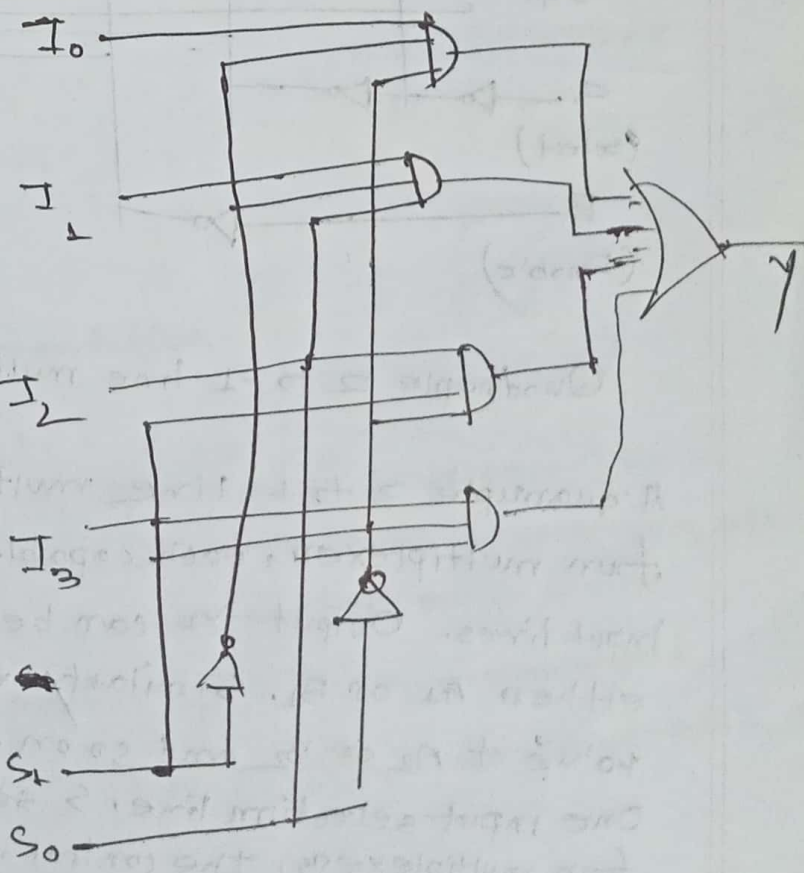
Multiplexer: A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

A multiplexer is also called a data selection, since it selects one of many inputs and steers the binary information to the output line.



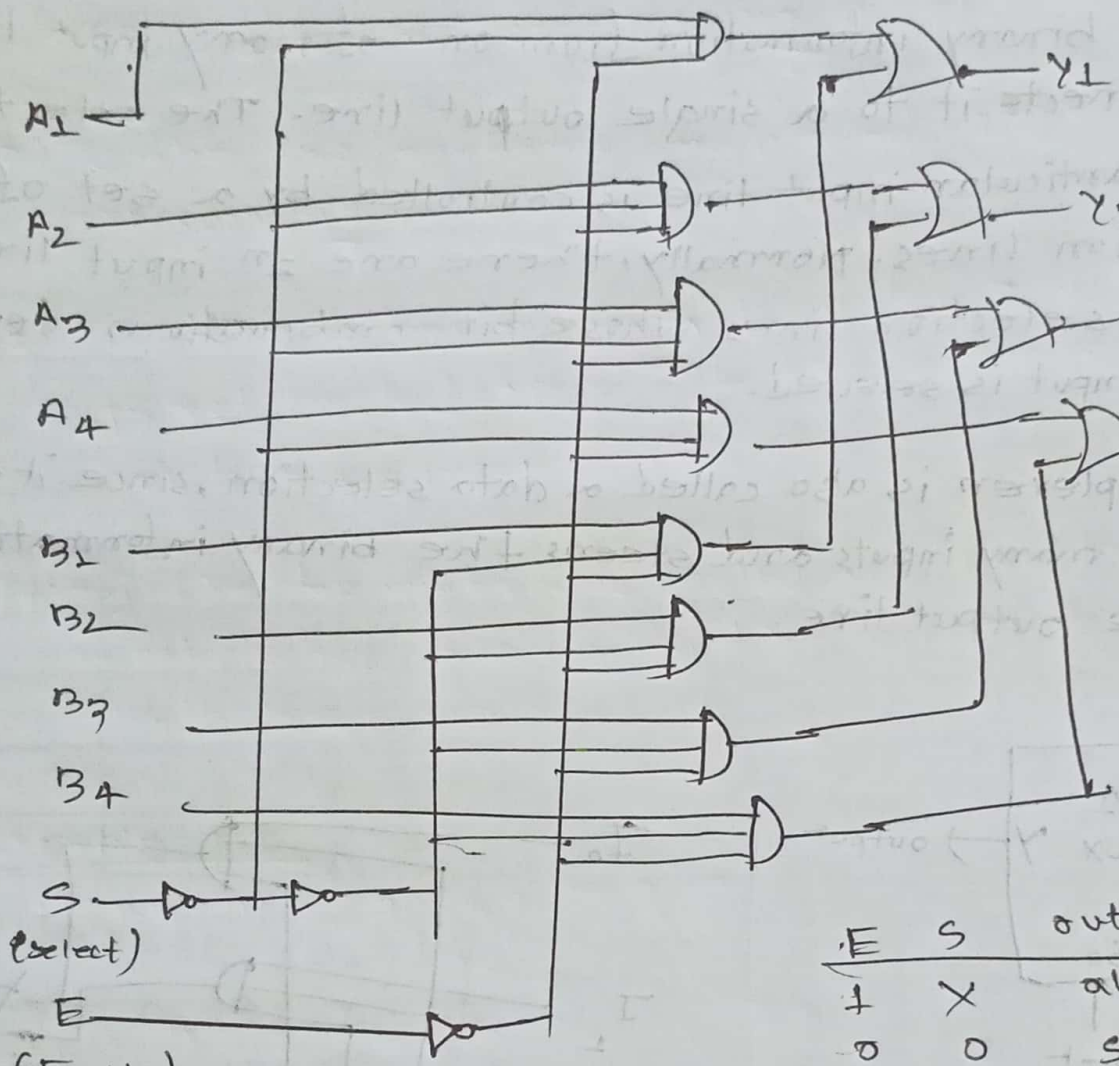
Block diagram

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



A 4 to 4 line multiplexer.

Quadruple 2 to 1 line multiplexer :



E	S	output Y
1	X	all 's zero
0	0	select A
0	1	select B

Quadruple 2 to 1 line multiplexer

A quaruple 2 to 1 line multiplexer is which has four multiplexer, each capable of selecting one of two input lines. Output Y_1 can be selected to be equal to either A_1 or B_1. Similarly output Y_2 may have the value of A_2 or B_2 and so on.

One input selection line, S select one of two lines in all four multiplexers. The control input E enables the multiplexers in the 0 state and disables them 1 state

Boolean function implementation

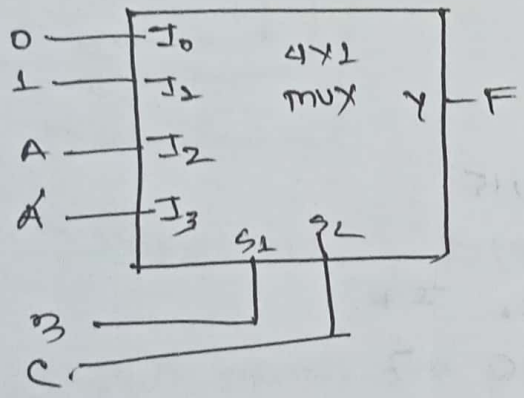
$F(A,B,C) = \Sigma(1,3,5,6)$

Truth Table

minterm	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

Implementation table

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



multiplexer implementation.

Operation/Rules/working:

List the inputs of the multiplexer and under them list all the minterms in two row. The first row lists all those minterms where A is complemented and the second row all the minterm with A uncomplemented,

Circle all the minterms of the function and inspect each column separately-

(1) If the two minterms in a column are not circled apply 0 to the corresponding multiplexer input.

(2) If the two minterms are circled, apply 1 to the corresponding multiplexer input.

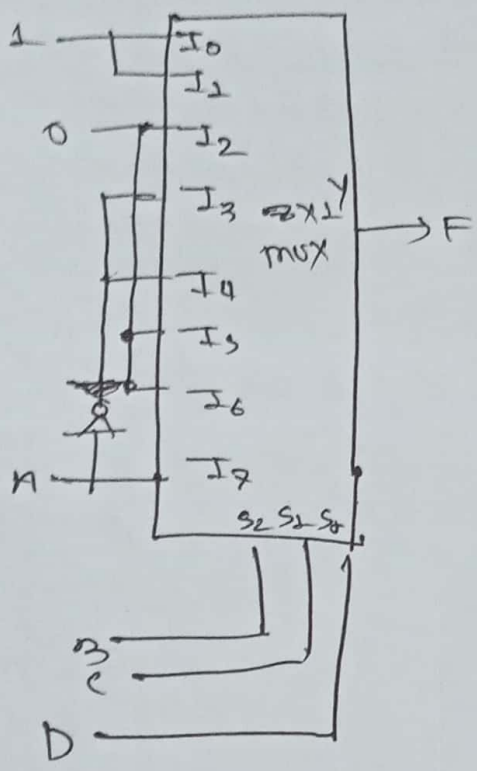
(3) If the bottom minterm is circled and the top is not circled, apply A to the corresponding multiplexer input

(4) If the top minterm is circled and the bottom is not circled, apply A' to the corresponding multiplexer input.

Example 5-4

$F(A,B,C,D) = \Sigma 0,1,3,4,8,9,15$

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



Implementing $F(A,B,C,D) = \Sigma(0,1,3,4,8,9,15)$

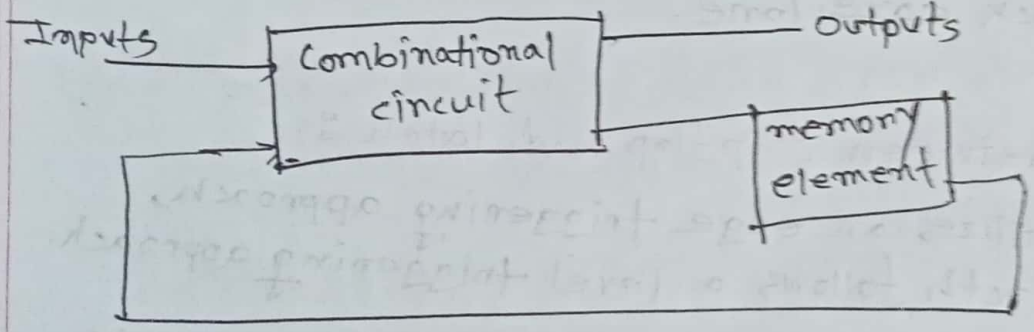
PLA: PLA stands for programmable Logic array. It is a fixed architecture logic device with programable AND gates followed by programble OR gates. PLA is basically a type of programable logic device used to build a reconfigurable ~~diti~~ digit circuit.

SAMRIT

Chapter-6

Sequential logic: Sequential logic is a type of logic circuit whose current outputs depend not only on the circuit inputs, but also on a memory of past input and states.

A sequential circuit refers to a special type of circuit. It consists of a series of various inputs and outputs. Hence the output depends on a combination of both present inputs as well as previous outputs. Thus, the sequential circuit consists of the combinational circuit along with its memory storage element.



block diagram of sequential circuit

There are two types of sequential circuit:

(i) Asynchronous sequential circuit

(ii) Synchronous sequential circuit.

(i) **Asynchronous sequential circuit:** This circuit don't use the clock signal. This types of circuits are operated through various pulses. Thus, the changes in input

can easily make a change in the state of circuit.

2. Synochronous Sequential circuit: This circuits are basically the digital sequential circuits where the clock signal goverm the feedback of the input for the next output generation.

Define latch: A latch is one kind of a logic circuit and it also known as a bistable-multivibrator. Because it has two stable states, namely active high as well as active low. It works like a storage device by holdings the data through a feedback core lane.

Difference between flipflop and latch :

(1) Flip-flop utilizes an edge triggering approach, where lacth follows a lerel triggering approach.

(2) In flip-flop the clock signal is present but in lateh the clock signal is absent.

(3) flip-flop can design using latches along with a clock. On the other hand lateh design using logic gates.

(4) Flip-flop is sensitive to the applied input and the clock signal. But latches are sensitive to the applied input signal only

(5) Flip-flop has slow operating speed where latches has comparatively fast operating speed.

(6) Flip-flop requires more power where as latches require comparatively less power.

(7) It is quite easy to perform circuit analysis, But Analyzing the circuit of latch is quite complex.

(8) Flip-flop performs synochronous operations. Where latches performs Asynochronous operations.

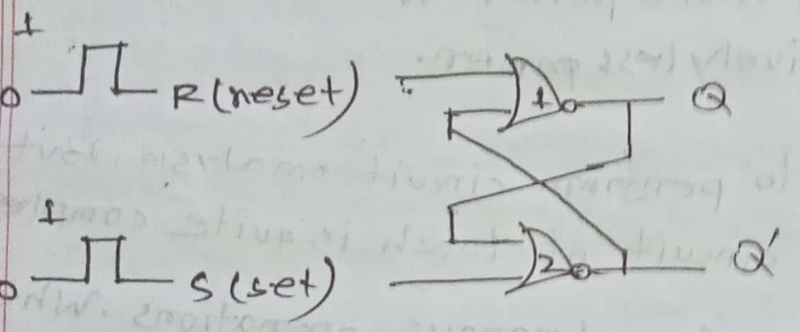
(9) Flip-flop are comparatively more robust where latches are comparatively lass robust.

(10) flip-flop requires more area but latches requires comparatively less area.

The memory elements used in clocked sequential circuits are called flip-flops.
These circuits are binary cells capable of storing one bit of information.

Basic flip-flop circuit: A flip-flop in electronics is a circuit with two stable states that can be used to store binary data. Flip-flop circuit can be constructed from two NAND or NOR gates.

Direct coupled RS flip-flop or SR latch:



(a) Logic diagram

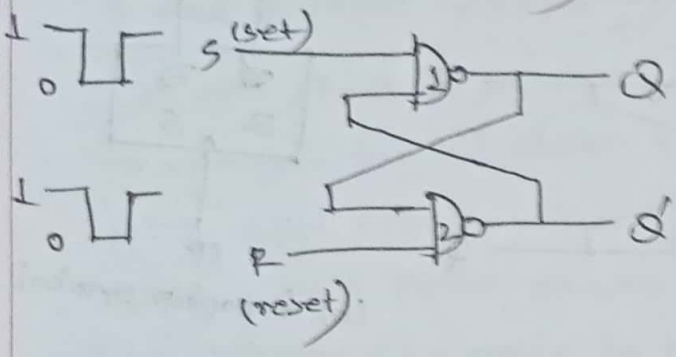
S	R	Q	Q'
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0

Operation: We know that output of a NOR gate is zero if any of its output is 1. And output is 1 when all the input is 0.

Here assume that set input 1 and reset input 0 since gate 2 has a input thus Q' must 0. so Q is obviously 1. When set input return to zero then output remain same because Q remain 1. It is same if reset is 1 and set is 0. Then Q=0, Q'=1. When 1,1 is applied in input then both Q and Q' is 0. This condition violet the fact that output Q and Q are the

complement of each other. We have to make sure that 1's are not applied to both inputs simultaneously.

Circuit Diagram (NAND)



(a) Logic diagram

S	R	Q	Q'
1	0	0	1
1	1	0	1
0	1	1	0
1	1	1	0
0	0	1	1

After S=1, R=1 output remains same

After S=0 R=1

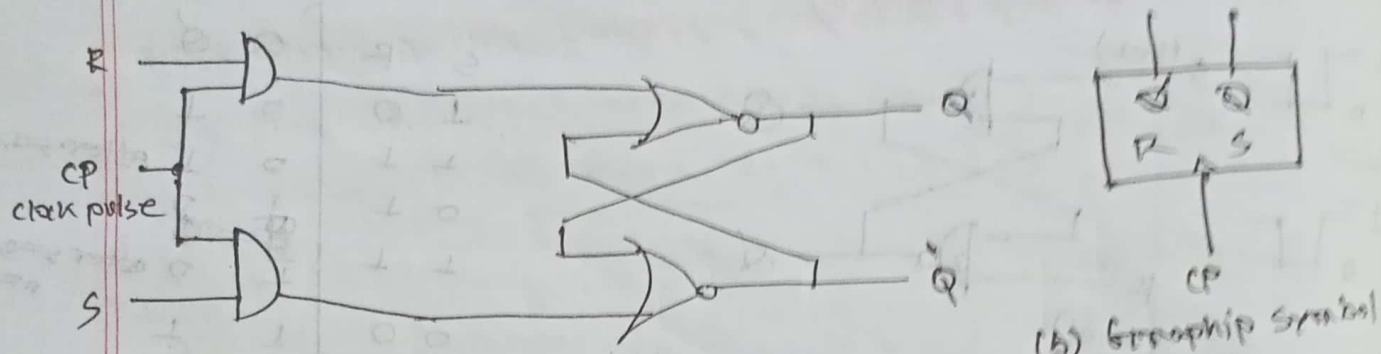
(b) Truth Table

Operation: We know that NAND gate produces output 1 if any of the input is 0. and if produces 0 if both of the input is 1.

We assume that, set is 0 and reset is 1 an input. Then Q must be 1 because NAND gate 1 has a input 0. Then Q' must be zero. If set is becomes 1 then both input is 1 there will no change output. same as set 1 and reset 0, input then Q=0 and Q'=1

When both of the input is 0 then Q and Q' becomes 1, that is the rules violations of a flip-flop So this condition must be avoid in any flip flop

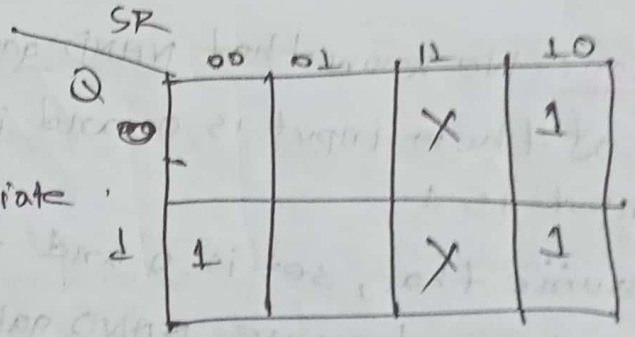
Clocked RS Flip-flop: The basic flip-flop as it stands is an asynchronous sequential circuit. By adding gates to the inputs of the basic circuit, the flip-flop can be made to respond to input levels during the occurrence of a clock pulse.



(a) Logic diagram (b) Graphic symbol

Q	S	R	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	Intermediate
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	Intermediate

Characteristic table



$Q(t+1) = S + R'Q$

$SR = 0$

characteristic equation

Figure: clocked RS flip-flop

The basic flip-flop as it stands is an asynchronous sequential circuit. By adding gates to the inputs of the basic circuit, the flip-flop can be made to respond to input levels during the occurance of a clock pulse.

The clocked RS flip-flop consists of a basic NOR flip-flop and two AND gates. The output of the two AND gate remain 0, as long as clock pulse is 0, regardless of the S and R input values.

When the clock pulse goes to 1, information from the S and R input is allowed to reach the basic flip-flop.

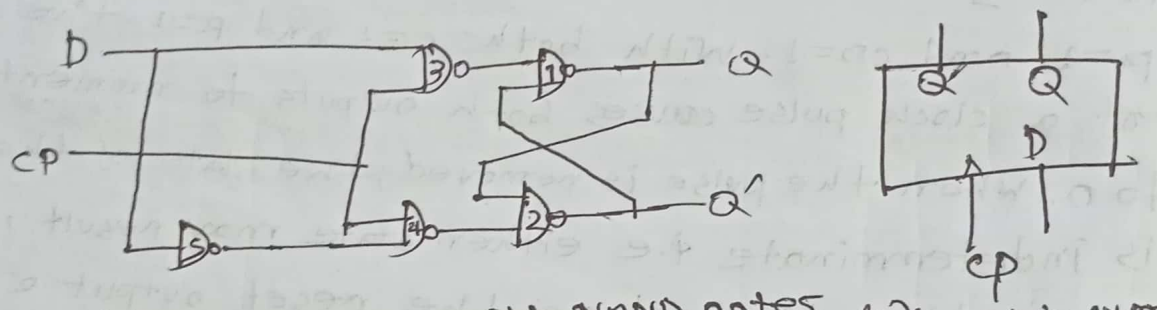
The set state is reached with S=1, R=0, and CP=1. To change to the clear state, the inputs must be S=0, R=1, and CP=1. With both S=1 and R=1, the occurrence of a clock pulse causes both outputs to momentarily go to 0. When the pulse is removed, the state of the flip-flop is indeterminate i.e either state may result, depending on whether the set or the reset output of the basic flip-flop remains a 1 longer before the transition to 0 at the end of the pulse.

From the characteristics table, for the flip-flop, this table summerise the operation of the flip flop in a tabular form. Q is the binary state of the flip-flop at a given time ~~refused~~ refered to present state, the S and R column give the possible values of the inputs, the Q(t+1) is the state of flip flop after the occurrance of a clock pulse refered to as next stage.

From the characteristics equation of the flip flop is derived in the map. The equation specifies the value of the next stage as a function of ~~a~~ the present state

and the input. We simplify the equation using map method, assuming indeterminate as a don't care (x) condition. Since S and R always remain different Thus SR=0, also a characteristics equation.

D flip-flop: D flip flop is a modification of the clocked RS flipflop.



(a) logic diagram with NAND gates (b) Graphic symbol

Q	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

(c) characteristic table

	0	1
0		1
1		1

(d) $Q(t+1)=D$ characteristic equation

Fig: clocked D flipflop

According to figure (a), NAND gate 1 and 2 form a basic flipflop and gate 3 and 4 modify it into a clocked RS flip flop. The input D goes directly to the S input and its compliment through gate 5, is applied to the R input.

As long as the clock pulse input is at 0, gates 3 and 4 have a 1 in their outputs, regardless of the value of the inputs.

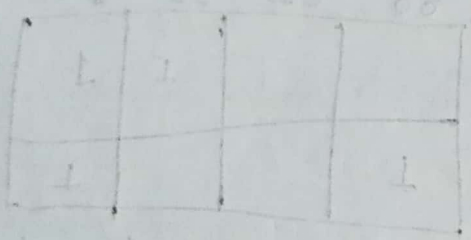
This confirms to the requirement that the two inputs of a basic NAND flip flop (1 and 2) remain initially at the 1 level. The D input is sampled during the occurrence of a clock pulse. If it is 1, the output of gate 3 goes to 0. switching the flip flop to the set state (unless it was already set). If it is 0, the output of gate 4 goes to 0, switching the flip-flop to the clear state.

The D flip-flop recieves the designation from its ability to transfer "data" into a flip-flop. It is basically an RS flip-flop with an inverter in the R input. The added inverter reduces the number of inputs.

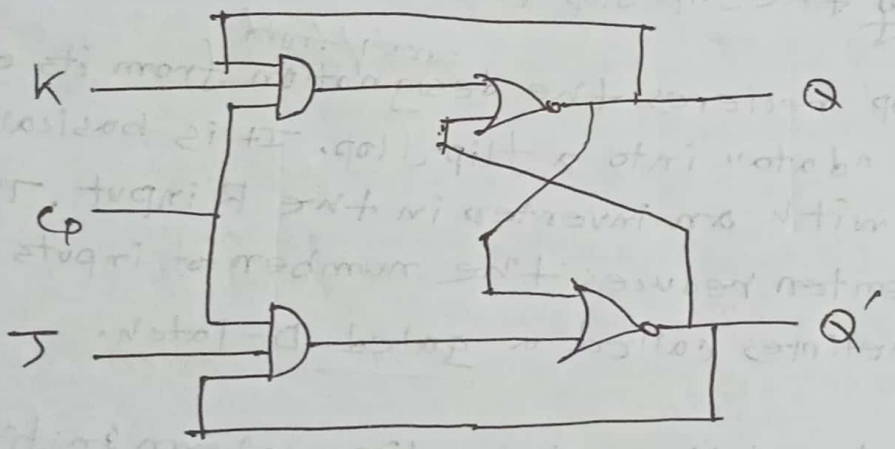
This is sometimes called a gated D-latch.

The characteristics table and equation shown in figure

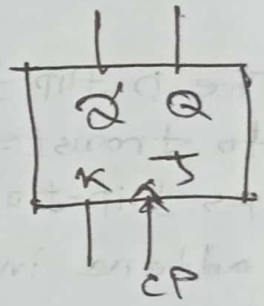
The value of the ~~most~~ next stage is the value of D at present state.



Jk Flip Flop: A JK flip-flop is a refinement of RS flip-flop in that the indeterminate state of the RS type is defined in the JK type. Inputs J and K behave like S and R to set and clear the flip-flop (note that in a JK flip-flop, the letter J is for set and K is for clear) when inputs are applied to both J and K simultaneously, the flip-flop switches to its complement state, that is, if Q=1 it switches to Q=0, and vice versa.



(a) Logic diagram



(b) Graphic symbol

Q	J	k	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	Ø

(c) Characteristic table

Q \ JK	00	01	11	10
0			1	1
1	1			1

$Q(t+1) = QK' + Q'J$

(d) Characteristic equation

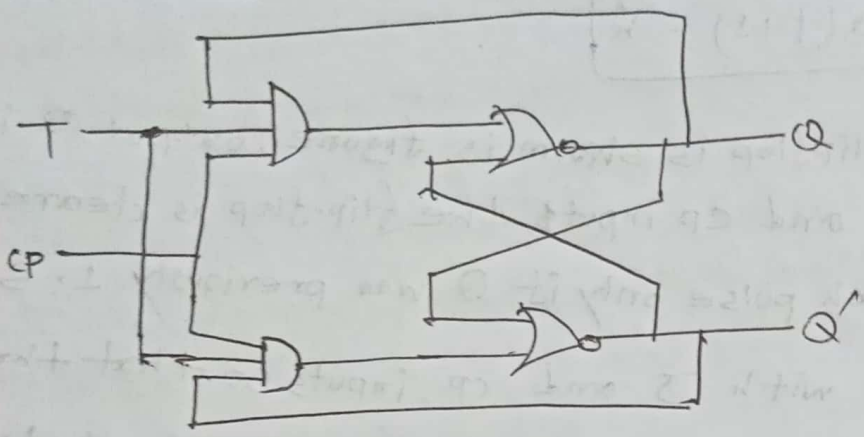
$Q = 0$ হলে $Q(t+1) = J$
$Q = 1$ হলে $Q(t+1) = \bar{K}$

A clocked JK flipflop is shown is figure, output Q is ANDed with K and CP inputs, the flip-flop is cleared during the clock pulse only if Q was previously 1. Similarly Q' is ANDed with J and CP, inputs so that the flip-flop is set with clock pulse only if Q' was previously 1.

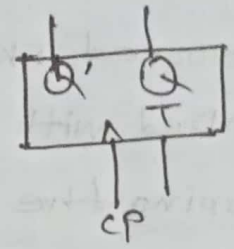
As shown in the characteristics table in Fig(c), the JK flip-flop behaves like an RS flip-flop, except when both J and K are equal to 1. When both J and K are 1, the clock pulse is transmitted through one AND gate only- the one whose input is connected to the flip-flop output which is presently equal to 1. Thus, if Q=1, the output of the upper AND gate becomes 1 upon application of a clock pulse, and the flip-flop is cleared.

If Q'=1, the output of the lower AND gate becomes a 1 and the flip-flop is set. In either case, the output state of the flip-flop is complemented.

T Flip flop



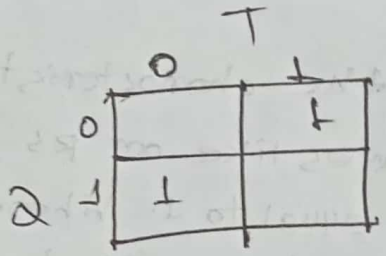
(a) Logic diagram



(b) Graphical symbol

Q	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

(c) Characteristic table



$Q(t+1) = TQ' + T'Q$

(d) Characteristic equation

Clocked T Flip-Flop

The T flip flop is a single input version of the JK flip-flop. As shown in figure, the T flipflop is obtained from a JK type if both inputs are tied together. The designation T comes from the ability of the flip flop to "toggle" on change state. Regardless of the present state of the flip flop, it assumes the complemented state when the clock pulse occurs while input T is logic-1

Triggering of flip flop : Clocked flip flops are triggered by pulses. A pulse starts from an initial value of 0, goes momentarily to 1 and after a short time, return to its 0 value.

A clock pulse may be either positive or negetive. A positive clock source remains at 0 during the interval between pulses and goes to 1 during the occurance of a pulse. The pulse goes through two signals transitions. from 0 to 1 and return from 1 to 0.

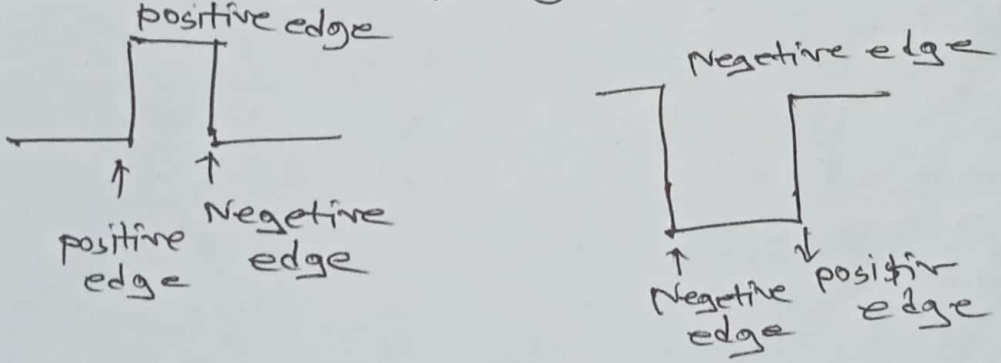


Figure: Definition of clock pulse

As shown in figure, the positive transition is defined as the positive edge and the negative transition of the negetive edge. This definition applies ~~to~~ also to negetive pulse.