Final Lab Report

Course No: 206

Course Title: Digital Logic Design

Submitted To:

Iffat Tamanna

Dept: CSE

Submitted By:

Md: Zobayer Hasan Nayem

Id: 19202103274

Section: 07

Dept: CSE

Lab - 01

Lab report:- 01

verify the Behavion of logic Gates using Truth Tables and familization with Digital Logic Gate.

Logic Gates below :-

1. OR Gate
2. AND Gate
3. NOT Gate
4. NOR Gate
5. NAND Gate
6. X-OR Gate
7. X-NOR Gate.

Table for Truth of OR Gates

$X = A + B$

input		output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

(2)

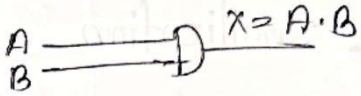
Truth table for AND Gates

$X = A \cdot B$

Input

A	B	$X = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

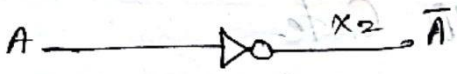
output



Truth table of NOT Gate

$X = \overline{A}$

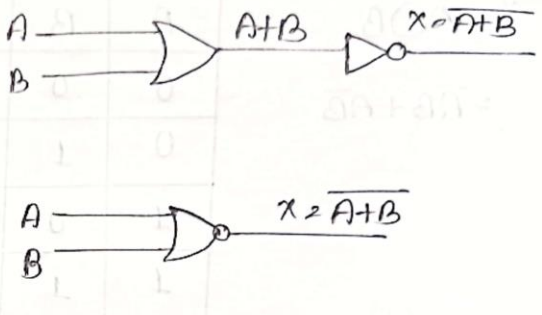
A	$X = \bar{A}$
0	1
1	0



Truth Table for NOR Gate

$X = \overline{A+B}$

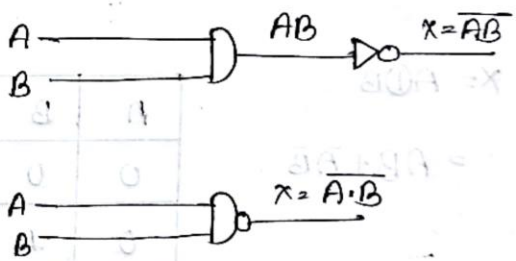
A	B	A+B	$\overline{A+B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0



Truth table for NAND GATE

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

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

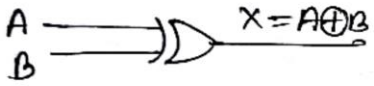


(4) Truth Table for x-OR Gate

$X = A \oplus B$

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

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

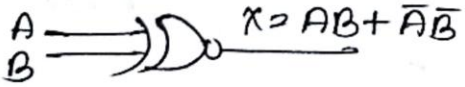


Truth Table for x-NOR Gate

$X = \overline{A \oplus B}$

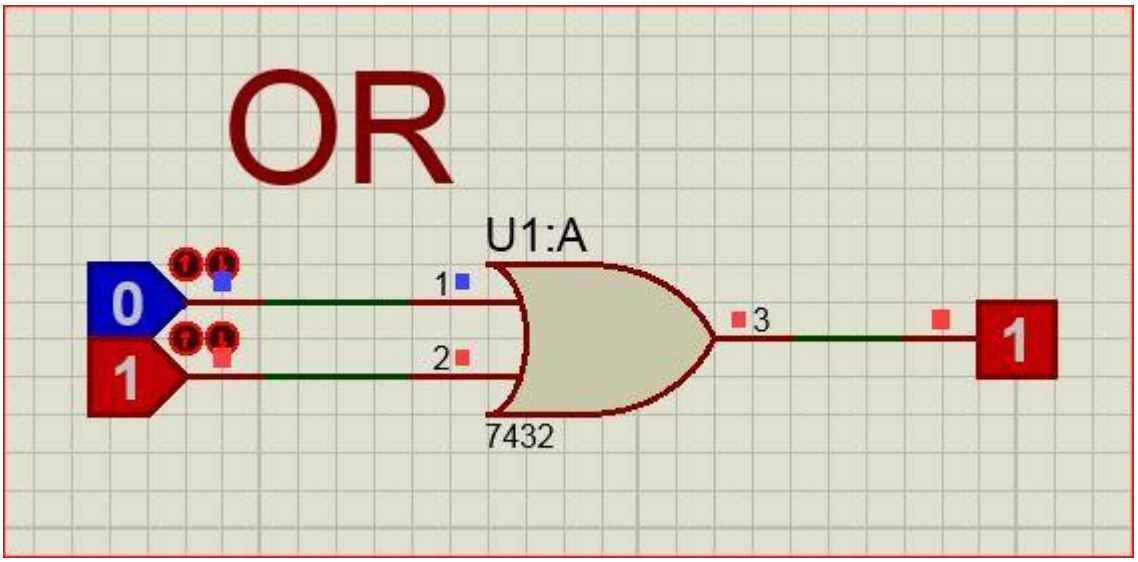
$= AB + \bar{A}\bar{B}$

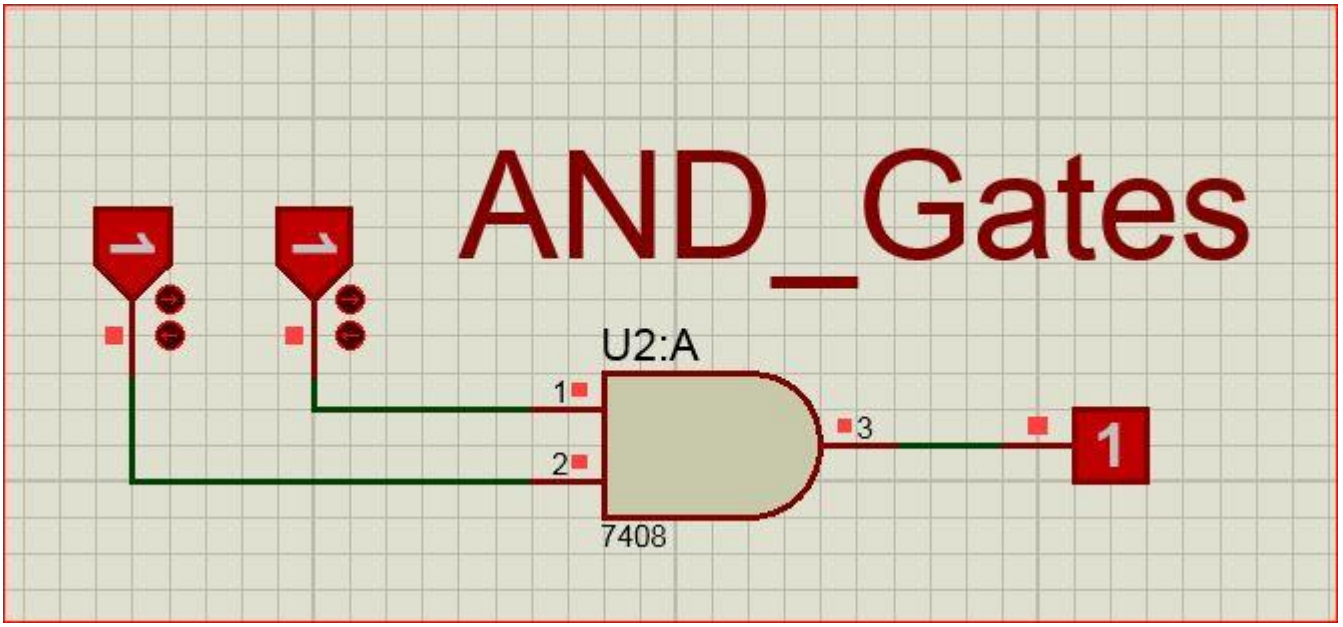
A	B	$\bar{A}$	$\bar{B}$	AB	$\bar{A}\bar{B}$	$AB + \bar{A}\bar{B}$
0	0	1	1	0	1	1
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	1	0	0	1	0	1

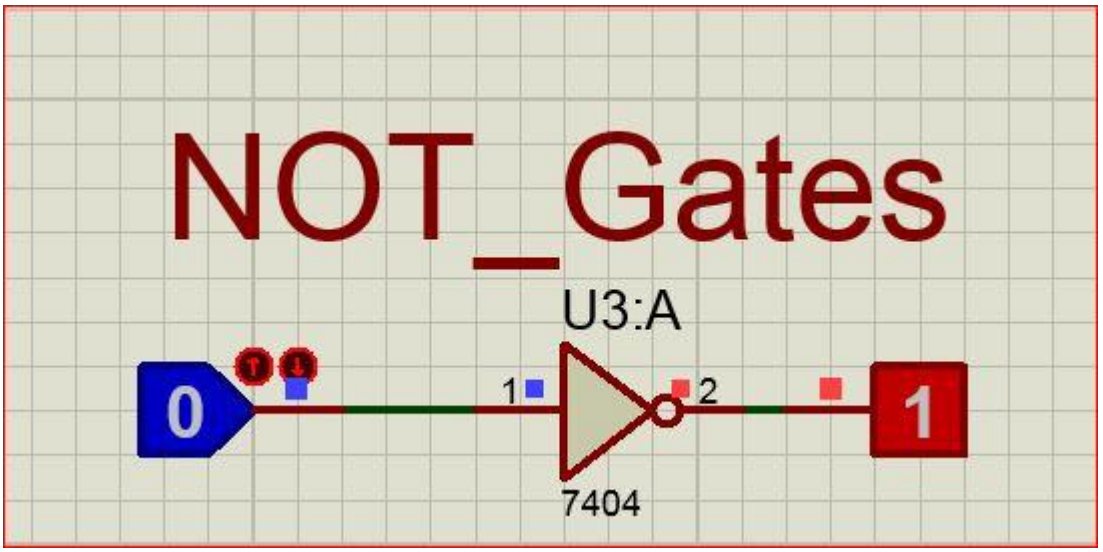


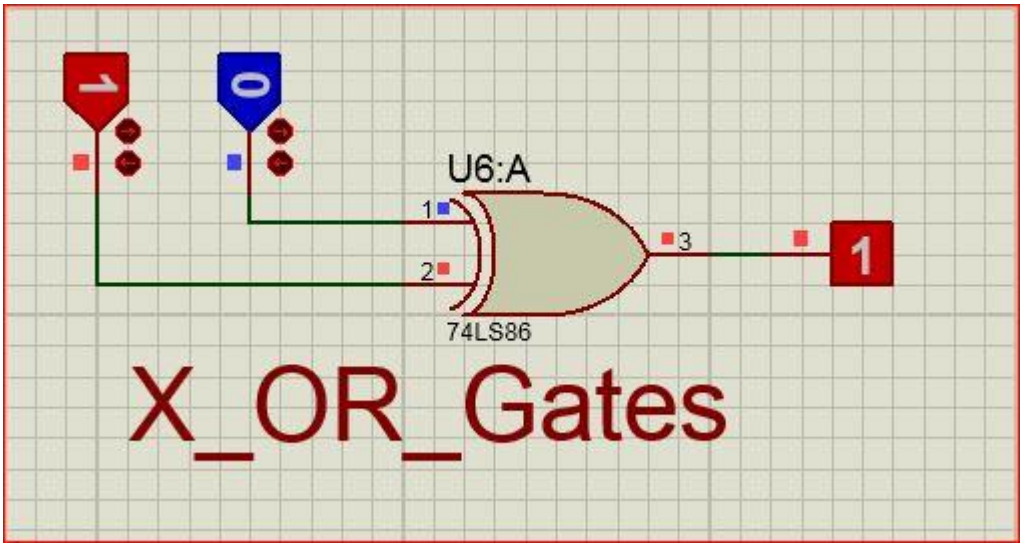
Conclusion :

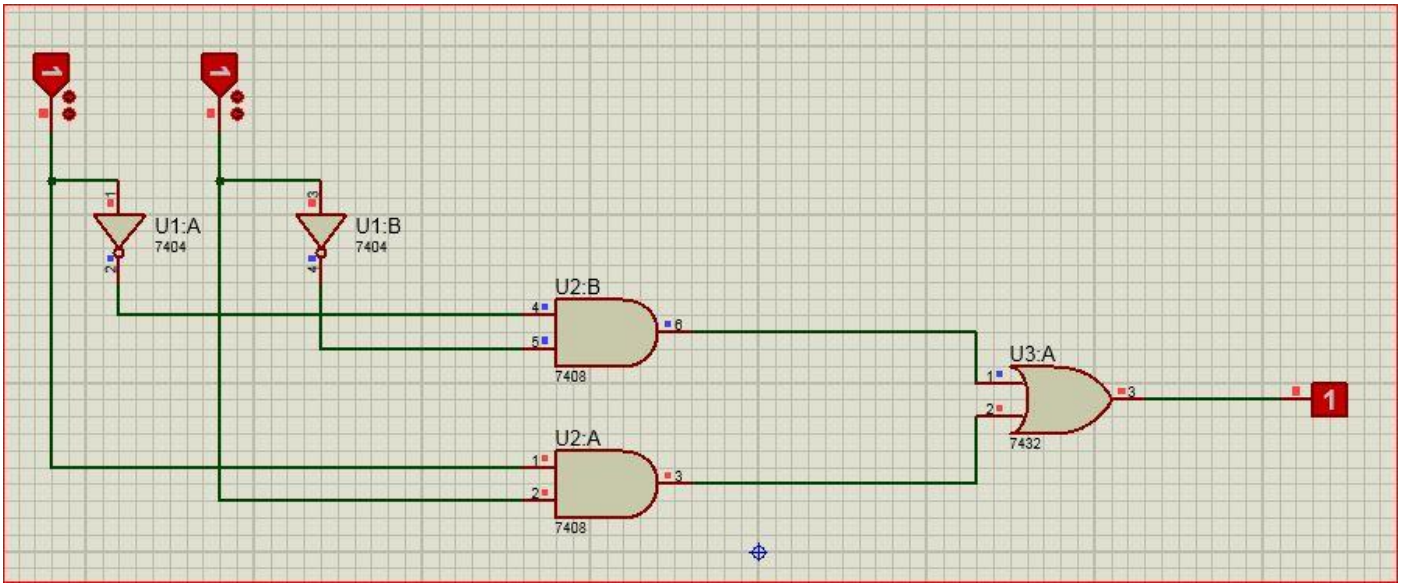
1. we have learnt how to implement circuits in proteus software.
2. we have understand the digit simulation of any circuit in the software.
3. we have verified the truth table for each input/output combination.
4. we repeated the process for all others logic gates.


OR
U1:A
0
1
1
2
3
1
7432

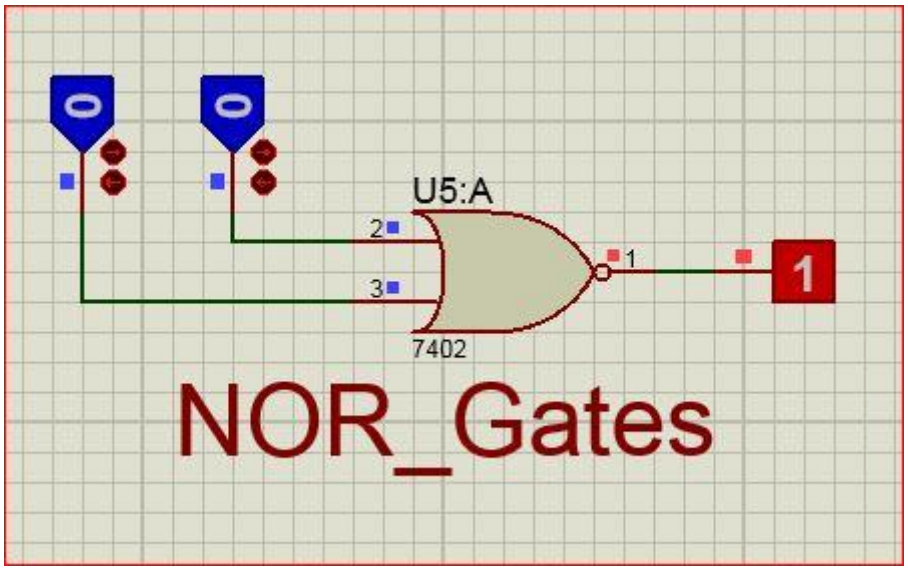

AND_Gates
U2:A
1
2
3
1
7408

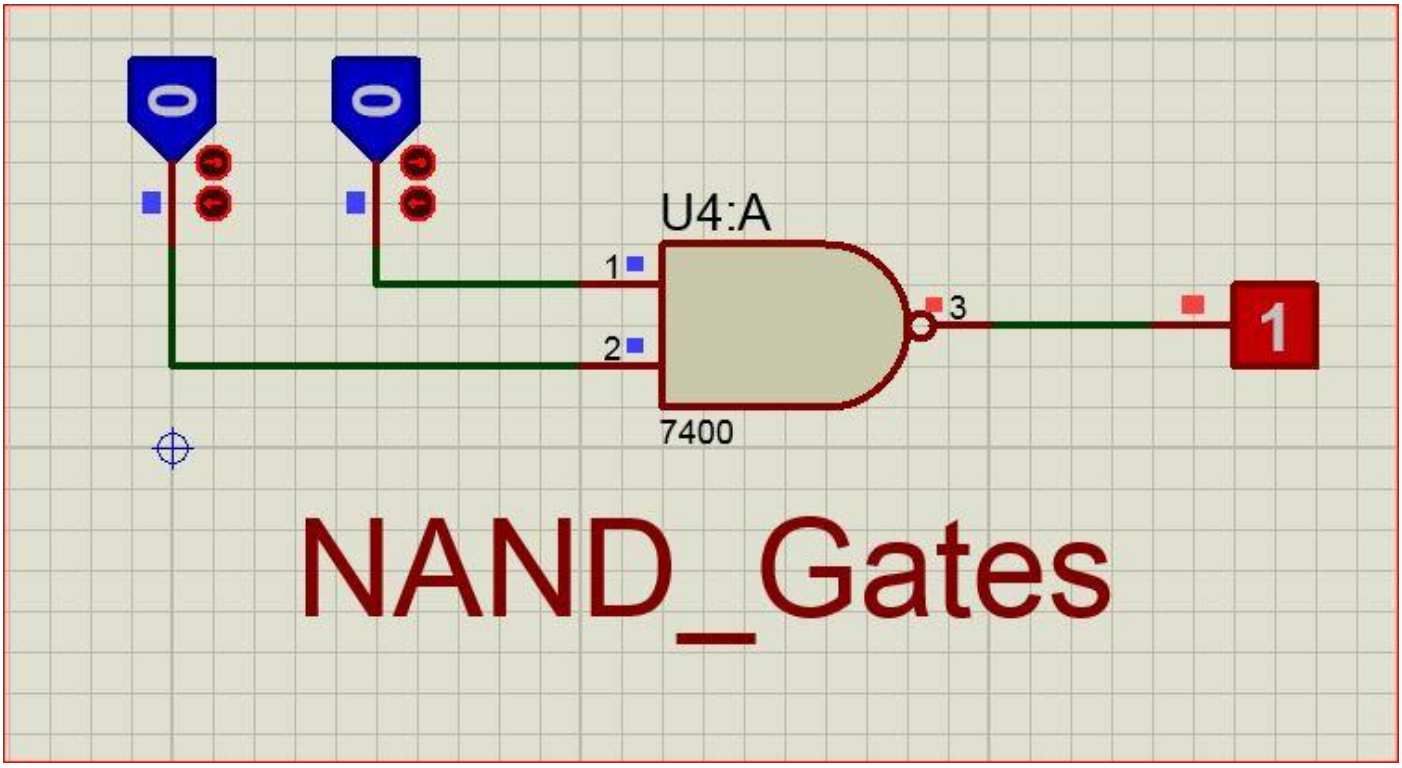

NOT_Gates
U3:A
0
1
2
1
7404


1
0
U6:A
1
2
3
74LS86
1
X_OR_Gates


1
1
U1:A
7404
U1:B
7404
U2:B
7408
U2:A
7408
U3:A
7432
1

X-NOR Gate


0
0
U5:A
2
3
1
7402
1
NOR_Gates


U4:A
1
2
3
7400
1
NAND_Gates

Lab – 02

Lab report : 2

Name of Experiment :

Implementation of basic gates Logic using universal Gates on protues.

Implementation all basic gates using NAND and NOR Gates.

Equipment :

1. NAND Gate
2. Logic Gate
3. Logic probe

A universal gate is a gate which can implement any Boolean function without need to use any other gate type.

So, The NAND and NOR gates are universal gates.

(2.)

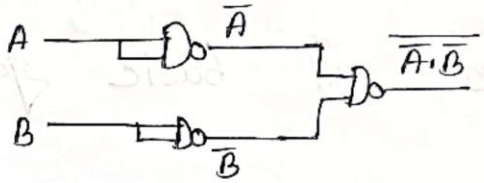
We will show that the AND, OR and NOT operation can be performed using only NAND gates.

OR Gate using NAND Gate.

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

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

$= A + B$ [OR Gate]



AND Gate using NAND Gate

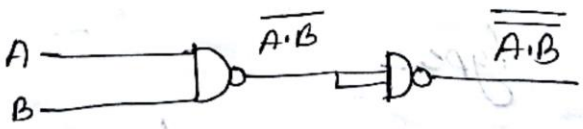
AND Logic $X = A \cdot B$

$X = \overline{AB}$ [using NAND]

$= \overline{\overline{AB}}$ [Double NAN]

$= AB$

= AND Gate

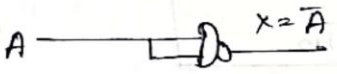


NOT Gate using NAND Gate

$X = \overline{A \cdot A}$ [NOT Logic and using NAND Gate]

$= \overline{A}$

= NOT Gate



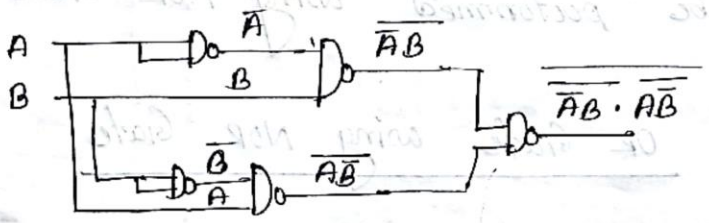
XOR Gate using NAND Gate

$X = \overline{\overline{A}B \cdot A\overline{B}}$ [output of XOR Gate using NAND]

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

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

= XOR Gate.



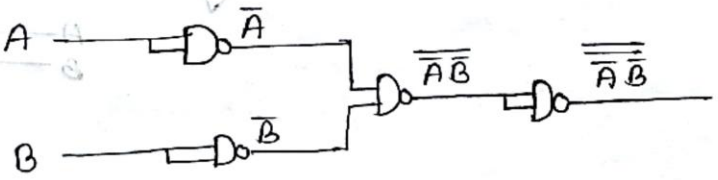
NOR Gate using NAND Gate

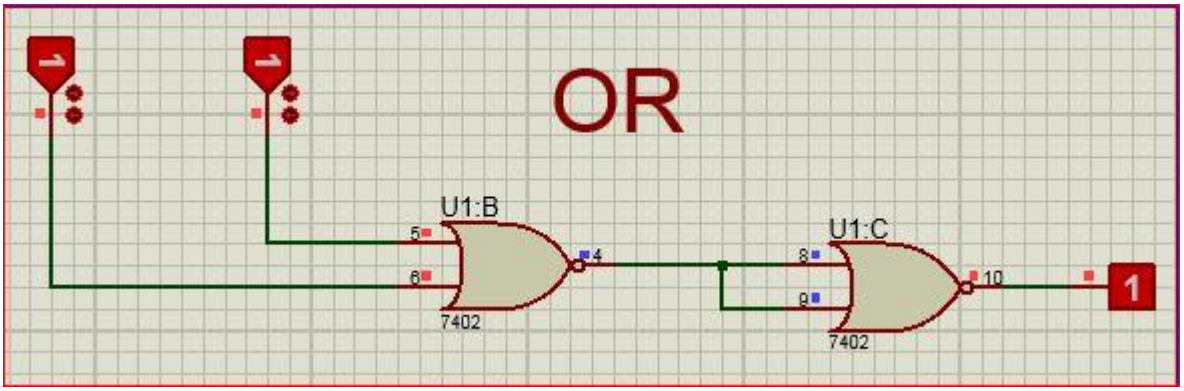
$X = \overline{\overline{A}\overline{B}}$

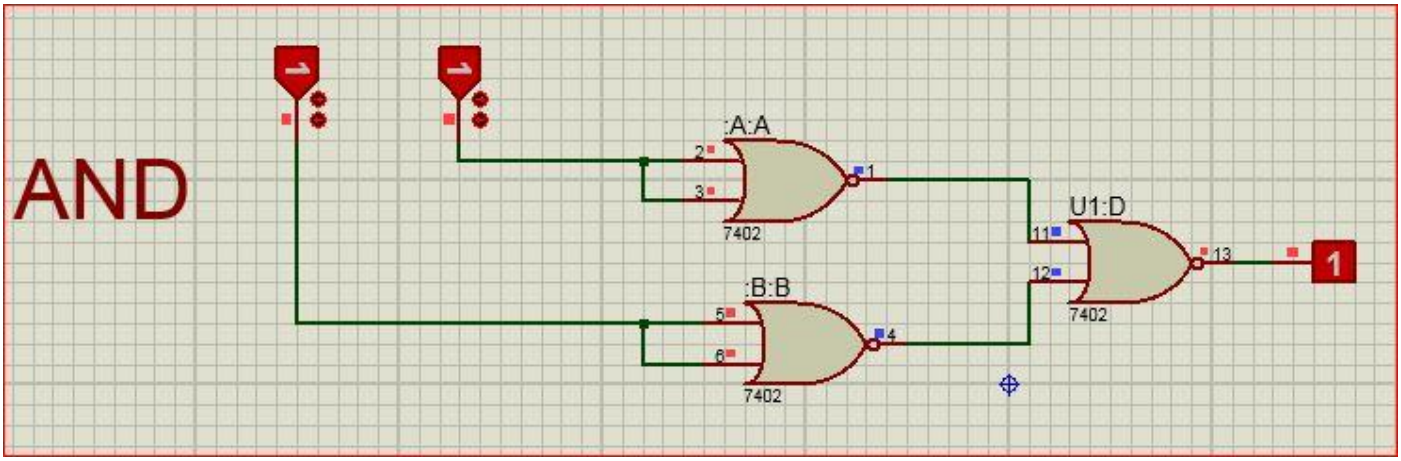
$= \overline{\overline{A} + \overline{B}}$

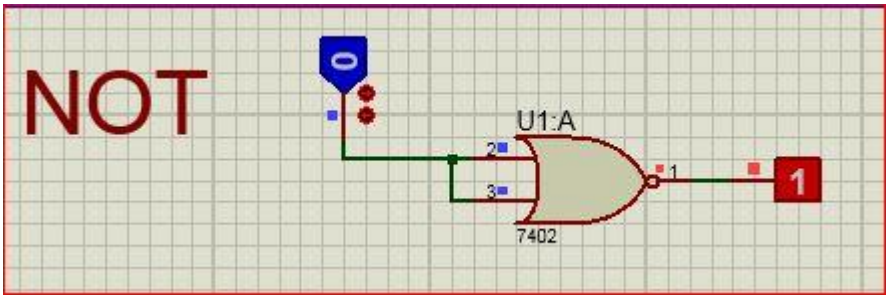
$= \overline{A + B}$

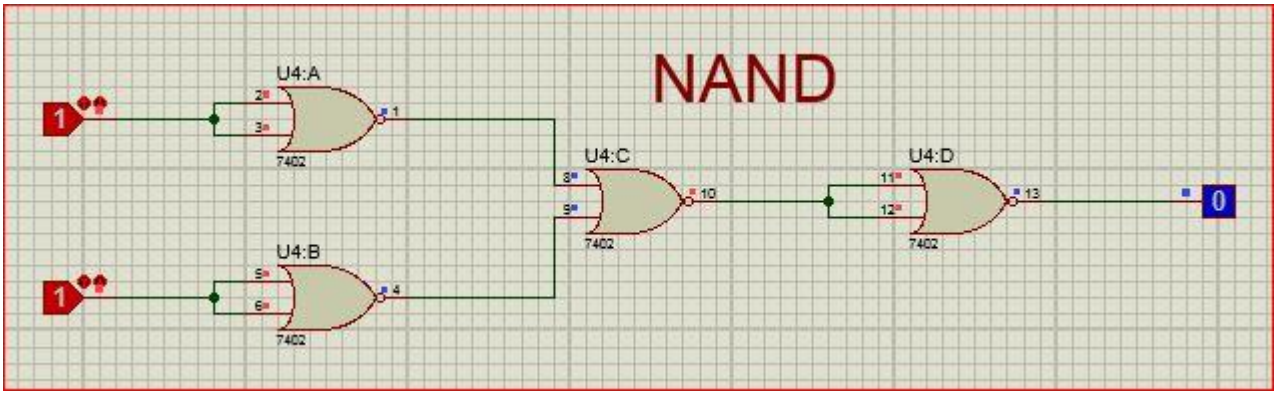
= NOR

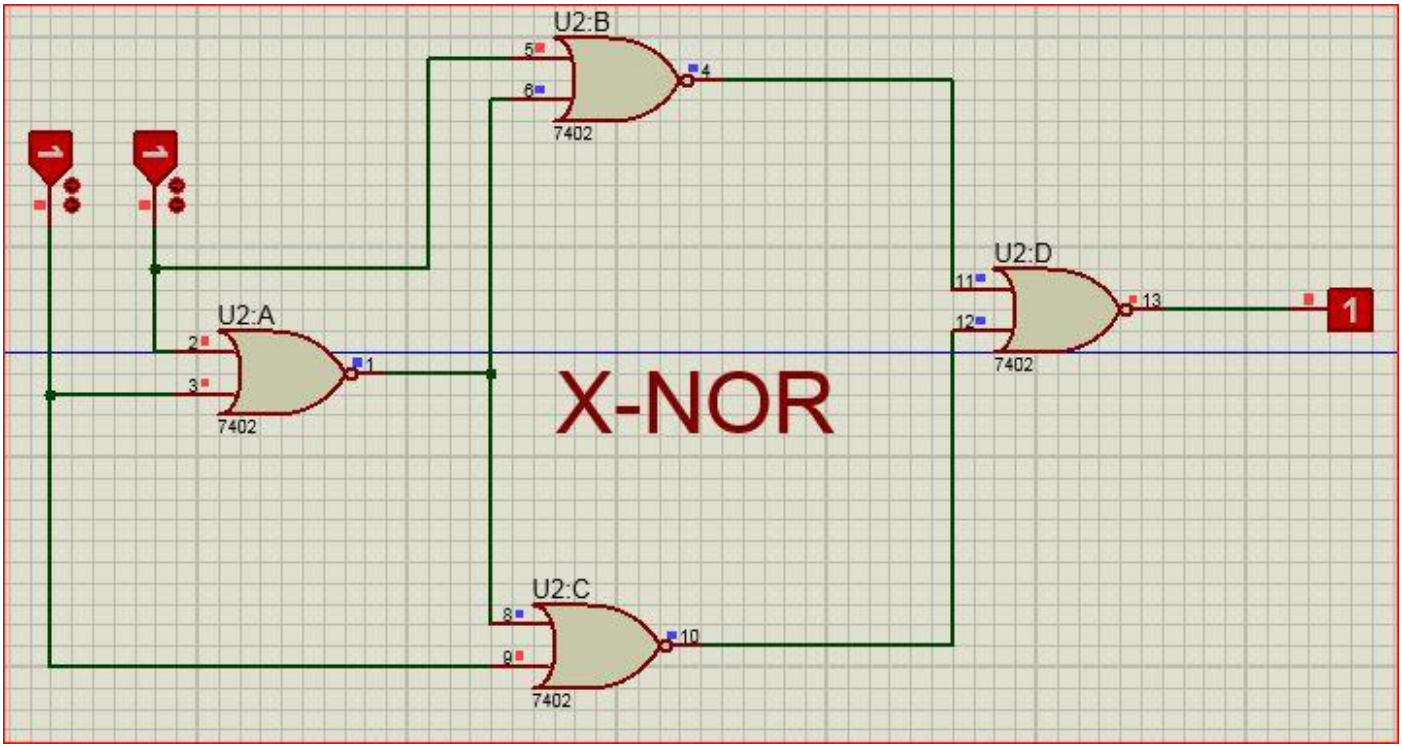


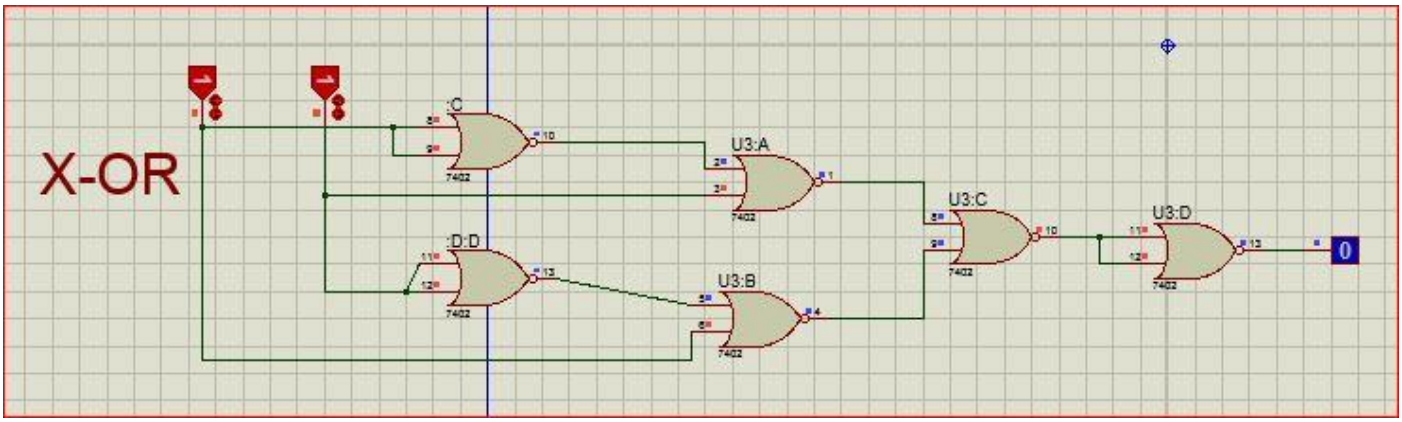

OR
U1:B
U1:C
7402
7402
1


AND
:A:A
:B:B
U1:D
7402
7402
7402
1


NOT
U1:A
7402
1


NAND
U4:A
U4:B
U4:C
U4:D
7402
1
0


X-NOR
U2:A
U2:B
U2:C
U2:D
7402
1


X-OR
U3:A
U3:B
U3:C
U3:D
7402
0

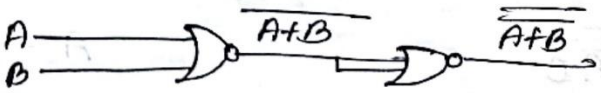
(1) Implemention all gates using NOR Gate

we will show that the AND, OR and NOT operation can be performed using NOR Gate.

OR Gate using NOR Gate.

$x = \overline{\overline{A+B}}$ [double NOR Gate use]

$= A+B$ [OR Gate Logic]

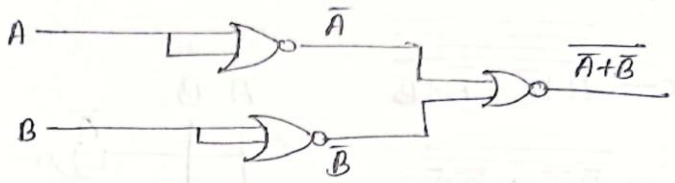


AND using NOR Gate

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

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

$= A \cdot B$ [AND Gate Logic]



NOT Gate using NOR Gate

$X = \overline{A + A}$

$= \overline{A}$ [NOT Gate]

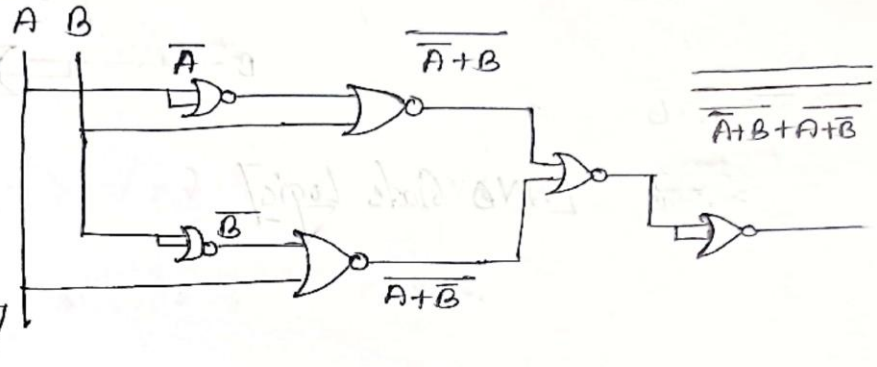
(3) X-OR using NOR Gate

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

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

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

$$= A \oplus B \quad [\text{X-OR Gate}]$$



X-NOR using NOR Gate

$$X = \overline{A+\overline{(A+B)} + \overline{B}+\overline{(A+B)}}$$

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

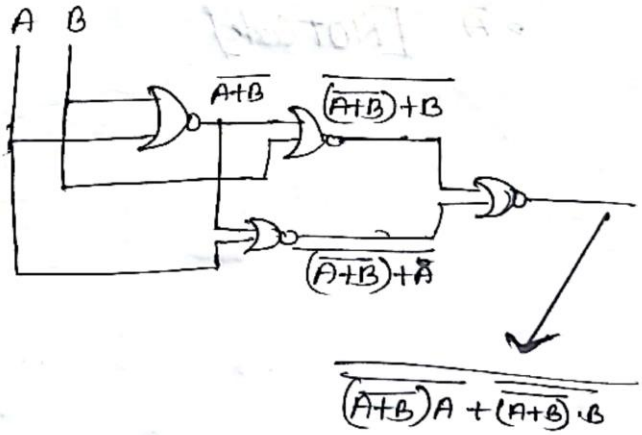
$$= \overline{\overline{A}\cdot(A+B) + \overline{B}\cdot(A+B)}$$

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

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

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

$$= \overline{A}\overline{B} + AB \quad [\text{X-NOR Gate}]$$



(4) X-NOR Gate using NAND Gate

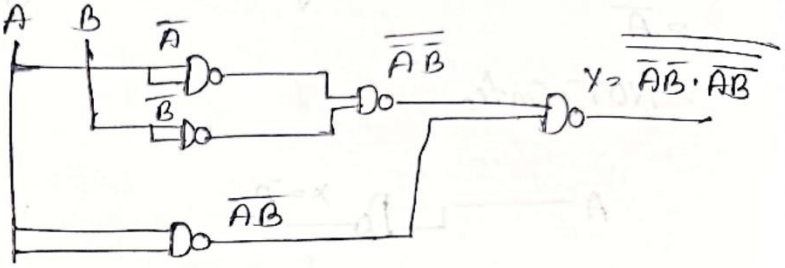
$X = \overline{\overline{A\bar{B}} \cdot \overline{\bar{A}B}}$ [Output of X-NOR using NAND]

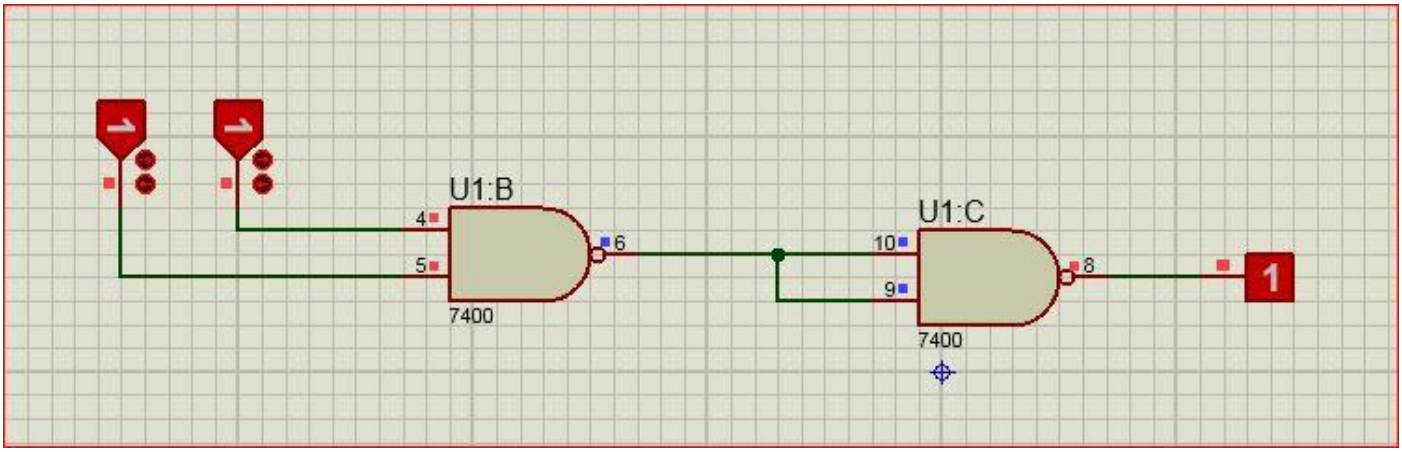
$= \overline{\overline{A} \cdot \overline{\bar{B}} + AB}$

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

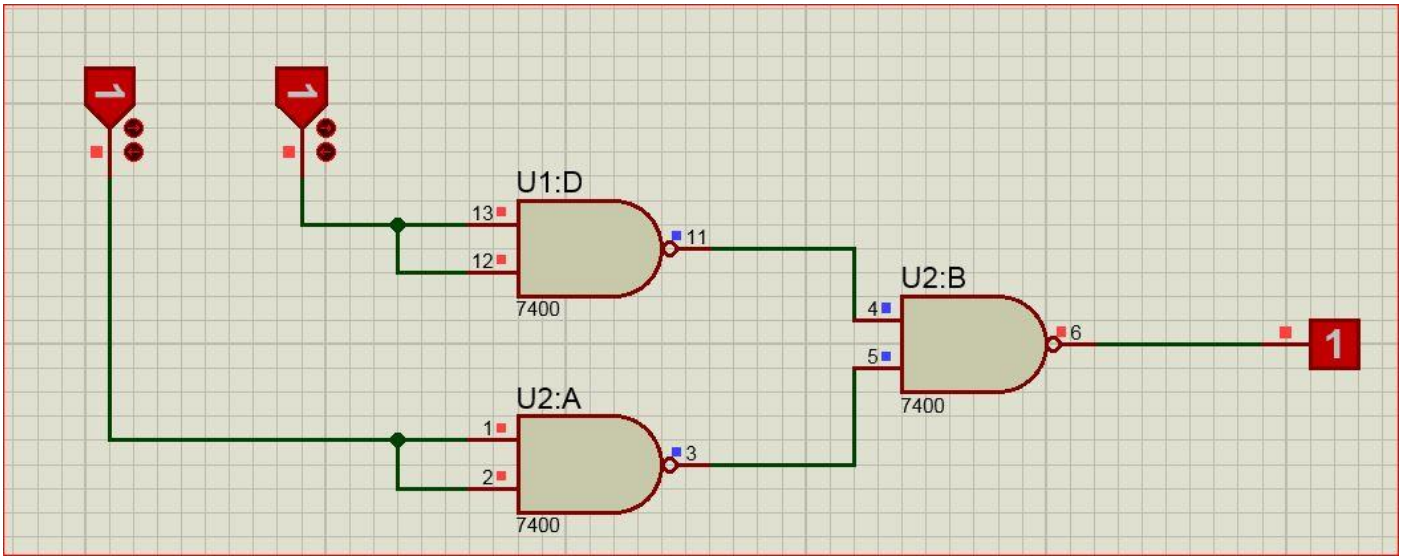
$= \overline{A \oplus B}$

$=$ X-NOR Gate.

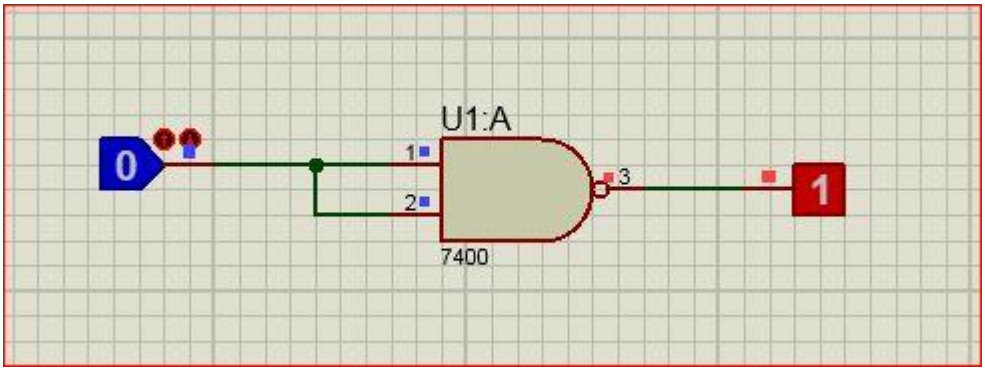




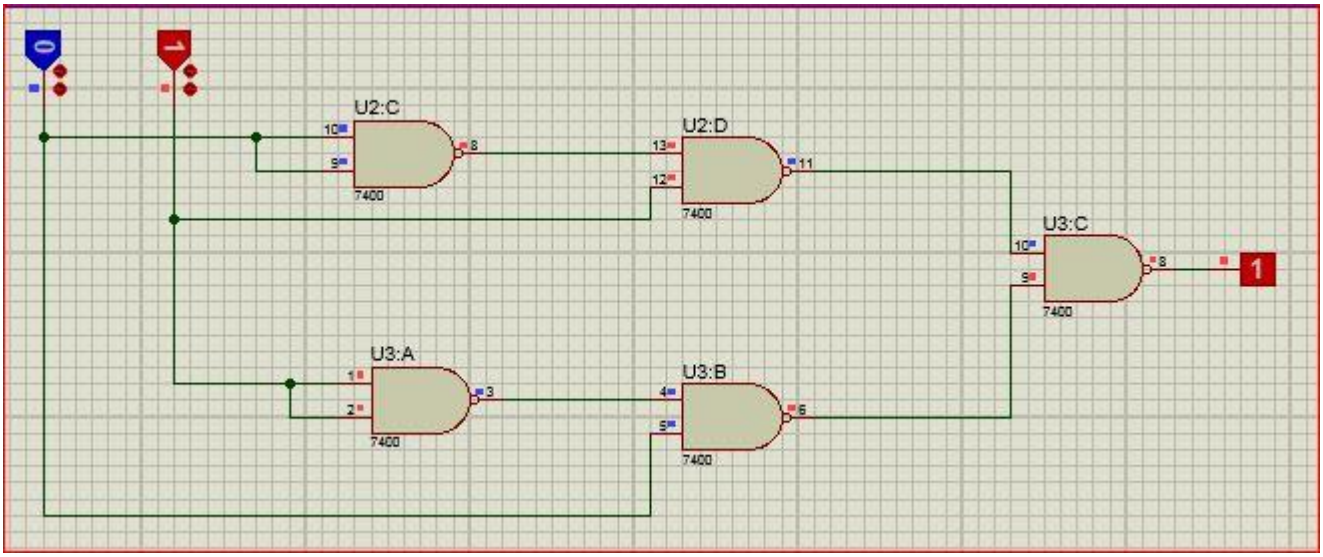
AND Using NAND Gate



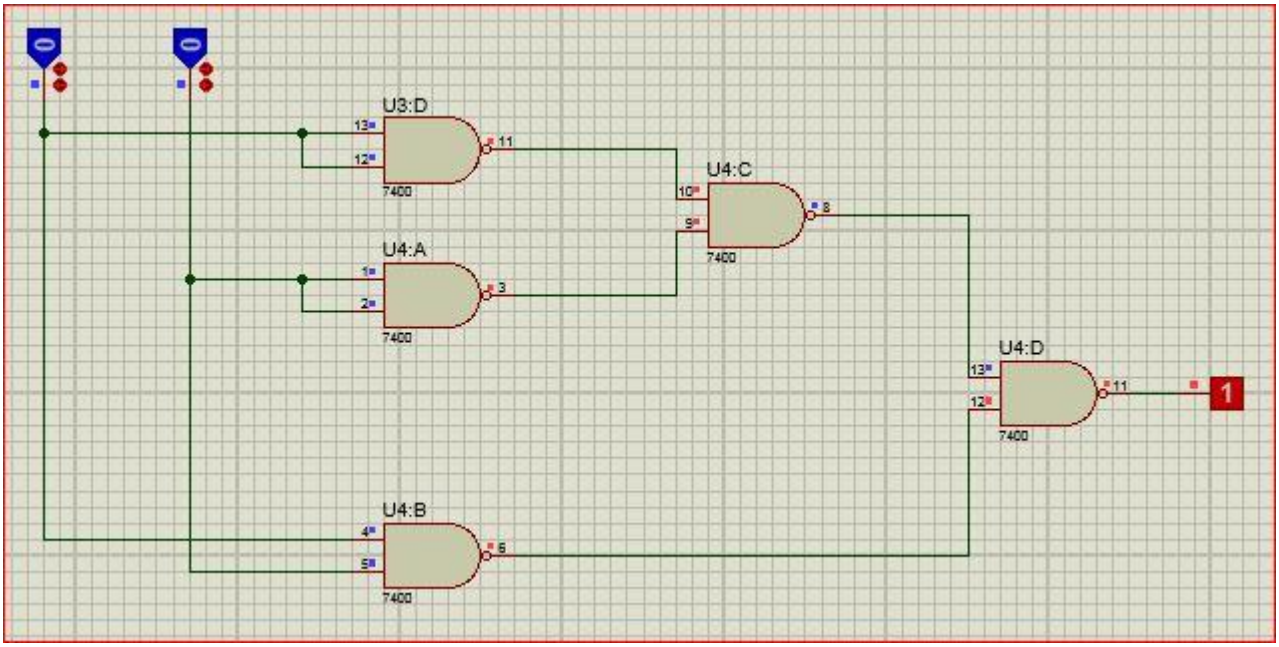
OR Using NAND Gate



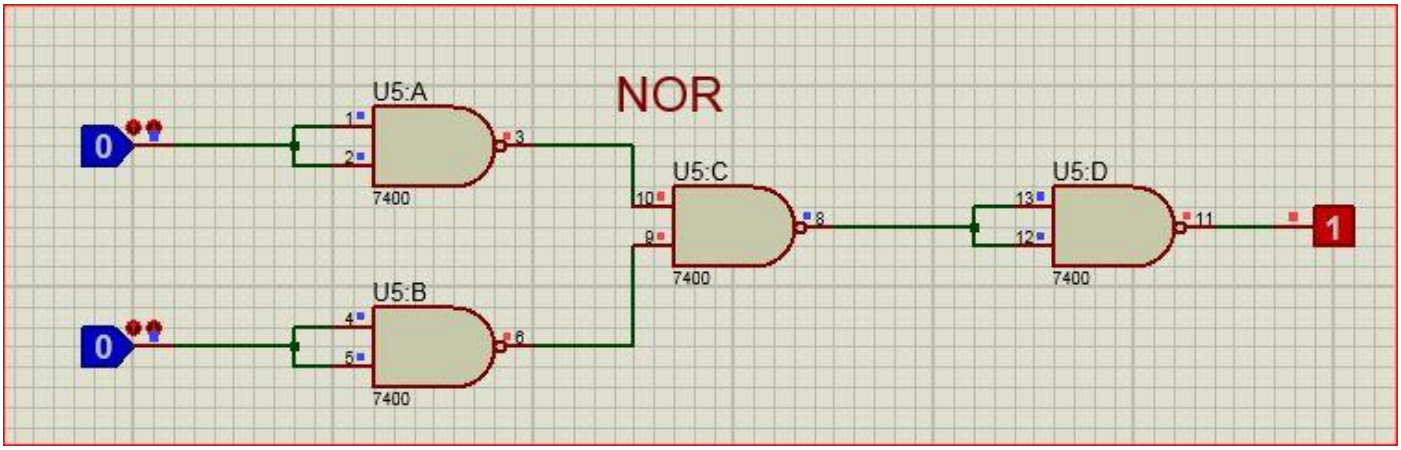
NOT Using NAND



X-OR Using NAND Gate



X_NOR Using NAND Gate



N-OR Gate Using NAND Gate

Conclusion:

1. we have learn't how to implement basic gates from univarsal gates.
2. we have learn what is NAND and what is NOR.
3. we got two types of variant of each gates from NAND and NOR.
4. we have also learn how to implement circuits in proteus software.
5. Lastly we understand the NAND an NOR gates to build up basic gates.

Lab-03

19202103274

Name of experiment : Implementation of Boolean function

Equipment :

(i) Two input AND Gate
(ii) Two input OR Gate
(iii) NOT Gate
(iv) Logic Gate
(v) Logic probe

Describe : The implementation of Boolean function by using Logic gates involves connecting output of one Logic gate to the input of another gates. Commonly used Logic Gates are AND, OR, NOT Gates. This Boolean function implementation is very simple and easy form.

19202103274

$F = \bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D} + ABC\bar{D} + A\bar{B}C\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D + A\bar{B}CD$

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

19202103274

Implementation the Boolean function

$F = \bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D} + ABC\bar{D} + A\bar{B}C\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D + A\bar{B}CD$

$\Rightarrow \bar{A}C\bar{D}(\bar{B}+B) + ABC\bar{D} + A\bar{B}C\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D + A\bar{B}CD$ [Distributive Law]

$\Rightarrow \bar{A}C\bar{D}\cdot 1 + ABC\bar{D} + A\bar{B}\bar{D}(C+\bar{C}) + A\bar{B}D(\bar{C}+C)$ [Complement Law]

$\Rightarrow \bar{A}C\bar{D}\cdot 1 + ABC\bar{D} + A\bar{B}\bar{D}\cdot 1 + A\bar{B}D\cdot 1$ [Identity Law]

$\Rightarrow \bar{A}C\bar{D} + ABC\bar{D} + A\bar{B}\bar{D} + A\bar{B}D$

$\Rightarrow C\bar{D}(\bar{A}+AB) + A\bar{B}(\bar{D}+D)$ [Distributive Law]

$\Rightarrow C\bar{D}((\bar{A}+A)(\bar{A}+B)) + A\bar{B}\cdot 1$ [Absorption Law]

$\Rightarrow C\bar{D}(1\cdot\bar{A}+B) + A\bar{B}$

$\Rightarrow CD(\bar{A}+B) + A\bar{B}$

A

1920210327 4

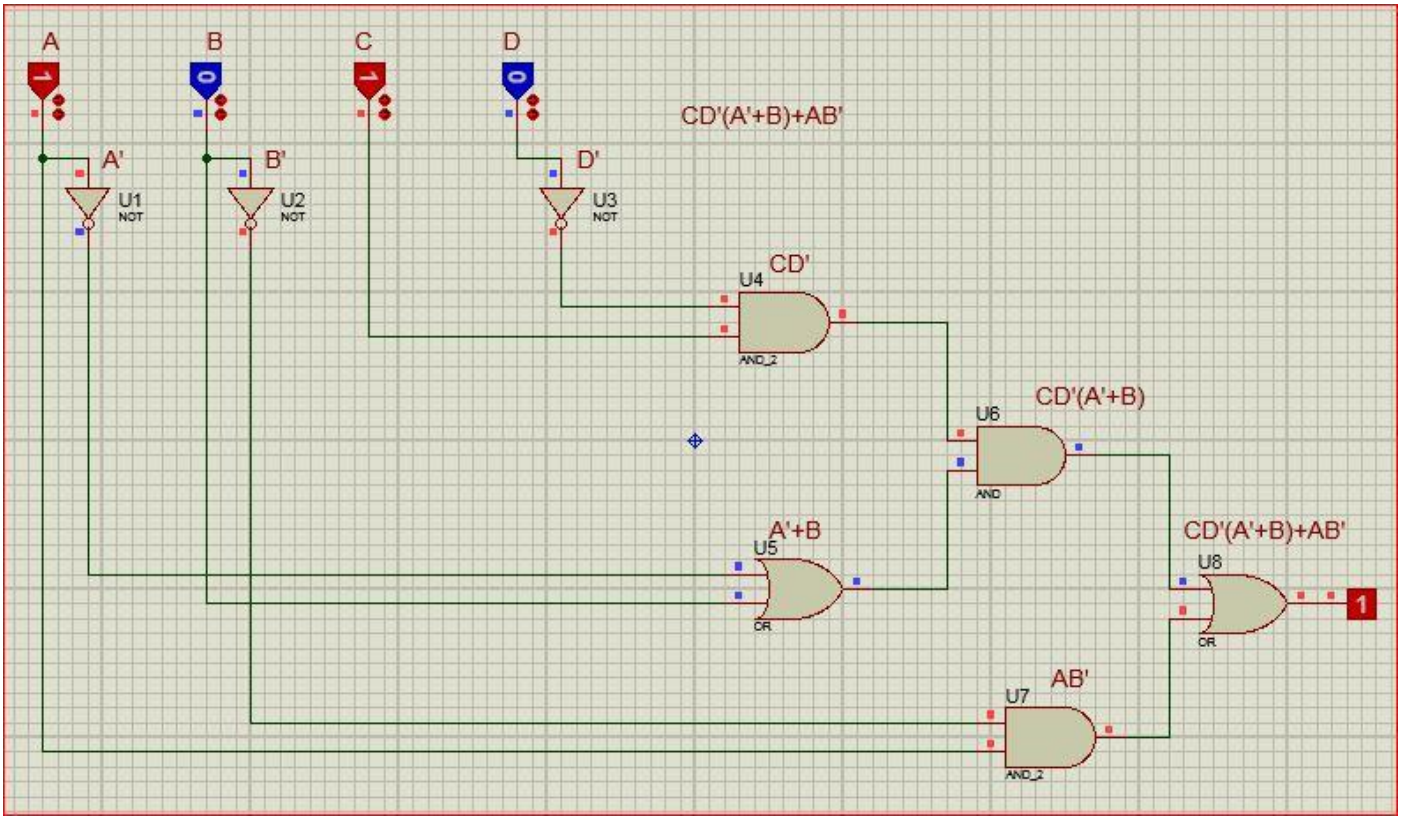
Conclusion:

① Boolean function can be implemention using the basic gates.

② Logic gates are very small circuit that implement Boolean operaton.

③ Boolean function are easy to implement by truth table & Logic gates

④ we design the circut to proteus software.


A
B
C
D
CD'(A'+B)+AB'
A'
B'
D'
U1
NOT
U2
NOT
U3
NOT
CD'
U4
AND_2
CD'(A'+B)
U6
AND
A'+B
U5
OR
CD'(A'+B)+AB'
U8
OR
1
AB'
U7
AND_2

Lab-04

ID: 19202103274

Lab Report → 04

Name of Experiment :

Construct and test various adders and sub-traction circuit.

Equipment :

1: 2 input AND Gate
2. 3 input AND Gate
3. 2 input OR Gate
4. 4 input OR Gate
5. X-OR Gate
6. Not Gate
7. 3 input OR Gate

Description:

An adders is a digital circuit that performs addition of numbers. In many computers and others kinds of processors adders are used in the arithmetic Logic units or other signed numbers representations require more logic around the basic adders. There are two types of adders; ① Half adders
② full adders.

1920210327Y

Half adder:- The half adder circuit has two inputs (x and y) which add two input digit and generates a carry and a sum. Half adder circuit doesn't consider the previous carry.

Full adder:- The full adder circuit has three input (x, y and z) where z is a previous carry and (x and y) are a literal. The full adder generate a carry and sum. It also consider the previous carry.

Half Adders

19202103274

This is the truth table of Half adders

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

result

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

$C = xy$

This is the truth table of Full adders

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

19202103274

K-map of carray

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

$C = xz + xy + yz$

K-map of Sum

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

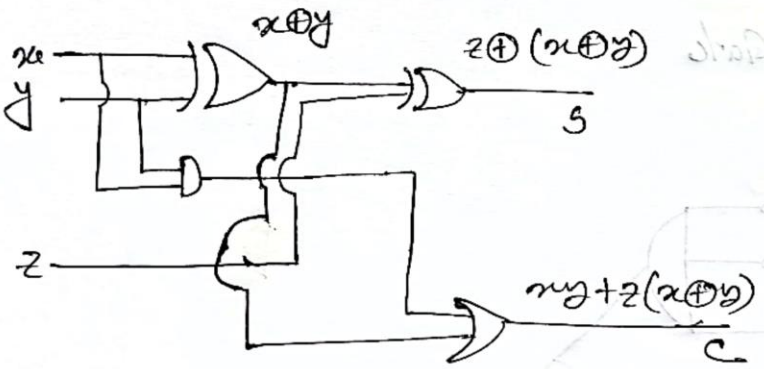
$S = x'y'z + x'yz' + xy'z' + xyz$

19202103274

Circuit of full adder using half adder:

Equipment

1. 2 AND Gate
2. 2 XOR Gate
3. OR Gate



19202103274

Simplification Full adders using half adders

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

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

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

$= z \oplus (x \oplus y)$ [Combine two half adders]

$C = xy + yz + xz$

$= xy + yz(x + x') + xz$

$= xy + xyz + x'yz + xz$

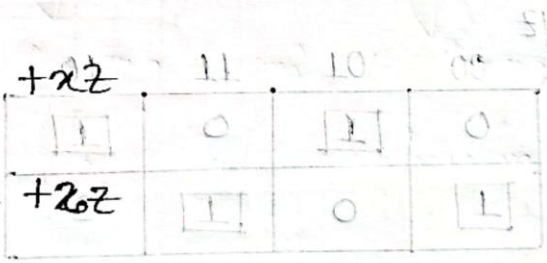
$= xy + x'yz + xyz + xz$

$= xy + x'yz + xz(y + 1)$

$= xy + x'yz + xz \cdot 1$

$= xy + x'yz + xz(y + y')$

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



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

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

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

$= xy + z(x \oplus y)$ [combine two half adders]

So, we can say, Full adders make with two half adders.

Subtractor

19202103274

Description :-

A subtractor logic circuit for calculating the difference between two numbers. The minuend and the numbers to be subtracted.

Half Subtractor :-

The half subtractor is also a building block for subtracting two binary numbers. It has two inputs and two outputs. This circuit is used to subtract two single bit binary numbers A and B. The D = Sub and B = borrow are two Output states of half subtractor.

Full Subtractor :-

The full subtractor is used to subtract three 1-bit numbers A, B, and C, which are minured subtrahend and borrow respectivly. The full subtractor has three input states and two outputs State. Sub = D and borrow = B

19202103274

Half Subtractor

A	B	Bn	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Half Subtractor implement with K-map of Borrow

A \ B	0	1
0	0	1
1	0	0

$B = \bar{A}B$

Half Subtractor implement with K-map of (D)

A \ B	0	1
0	0	1
1	1	0

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

19202103274

Full Subtractor

A	B	C	Bn	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

(A, B) → literal

C → Previous Borrow

K-map for Borrow

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

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

1902103274

K-map for D = Sub

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

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

Conclusion:-

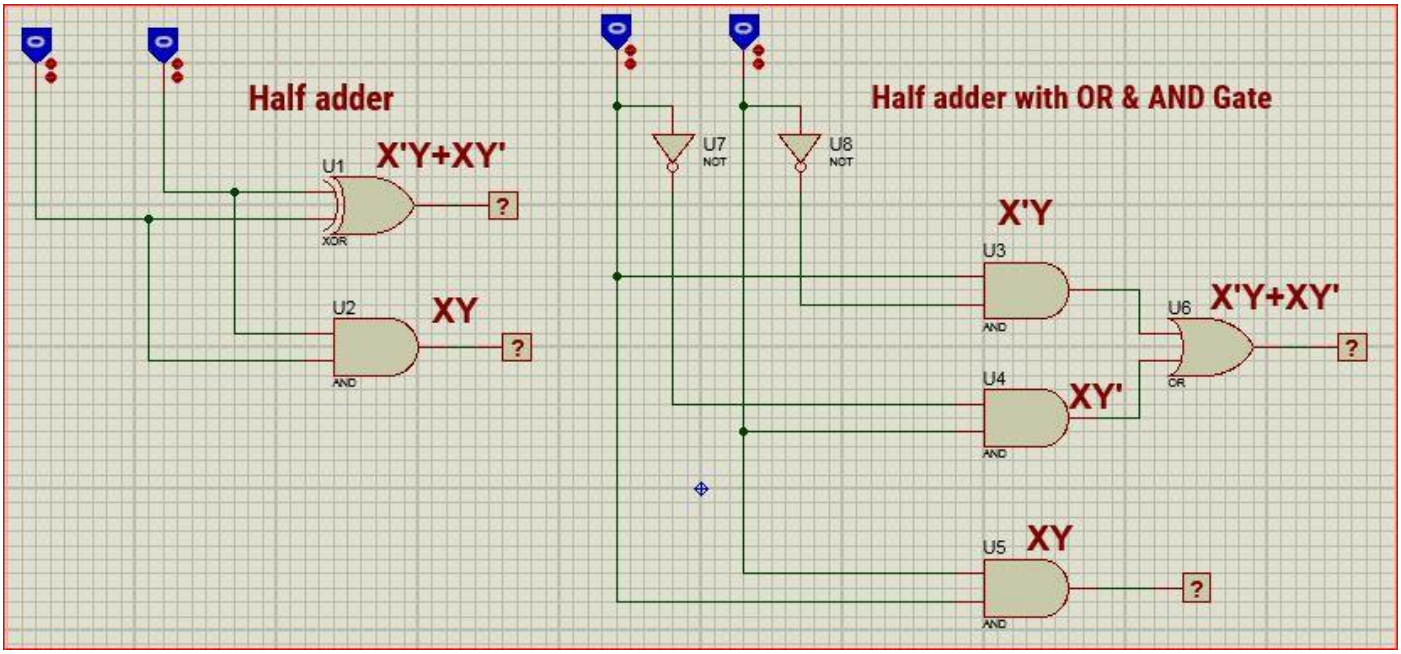
1. We have learn't how to implement Sub tractor and Full Subtractor.

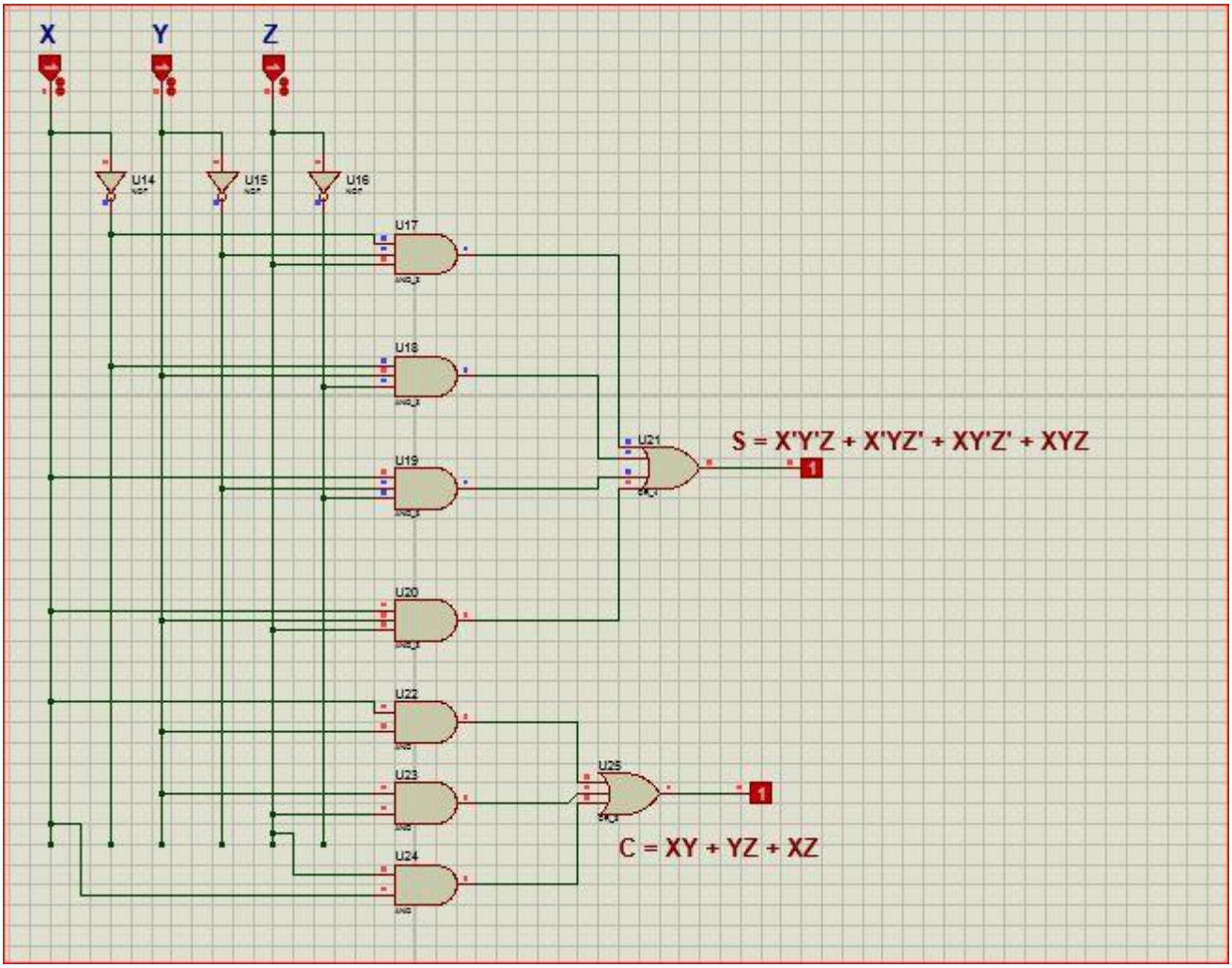
2. We also know how to work Half adder and how to get full adder from two half adder.

3. We learn't that, Half adder is an adder which adds two binary digits together, resulting in a sum and carry.

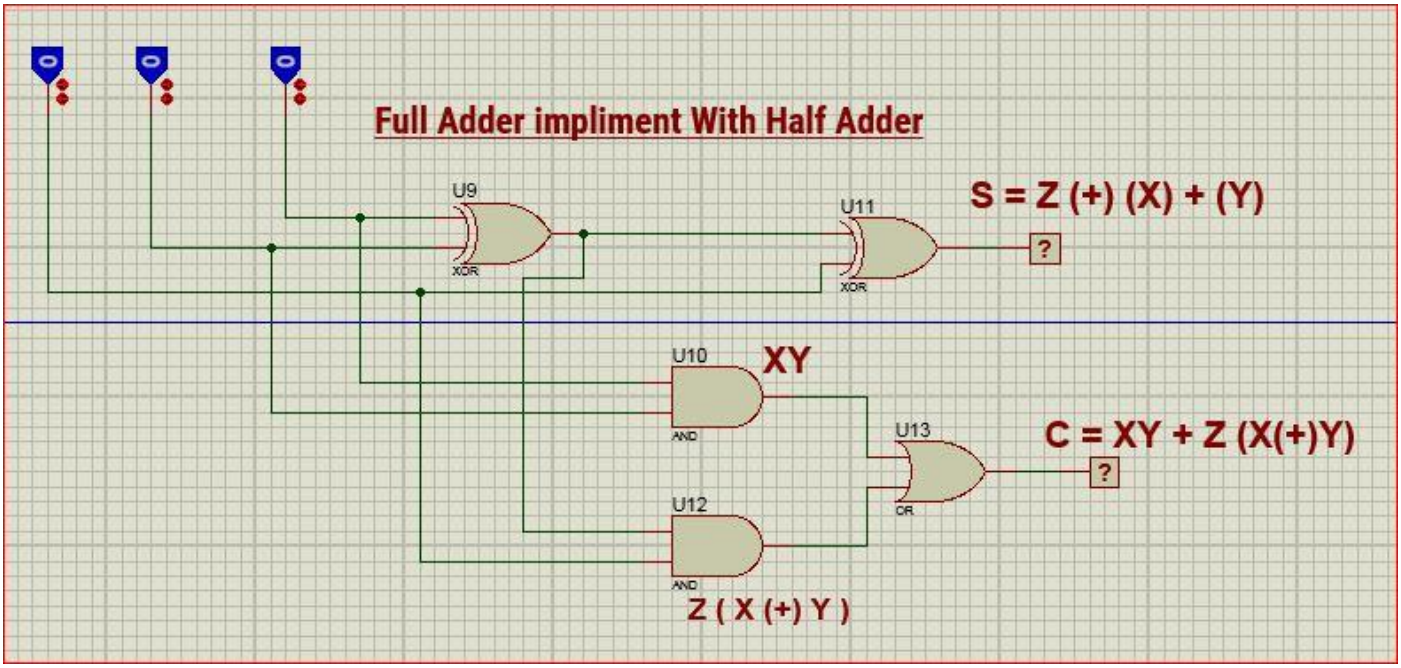
4. We also learn't Half Subtractor is another type of combinational circuit which is used to perform subtraction of two bits.

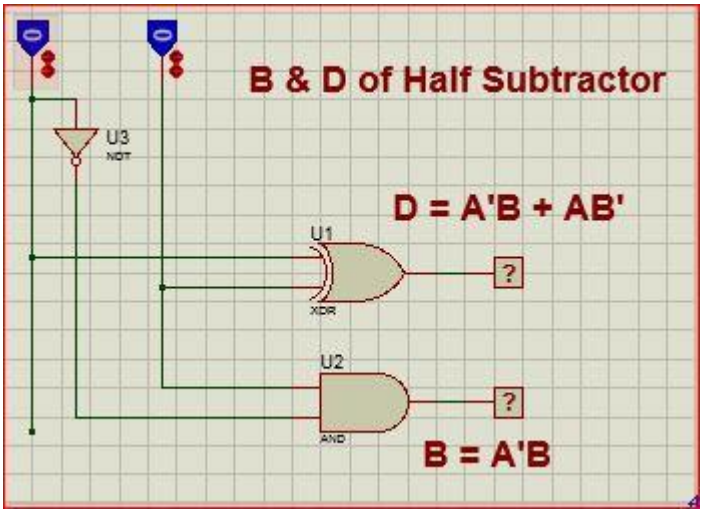
5. We also know, How to implements Half adder, full adder, and Half Subtractor and full Subtractor desing in protues software.

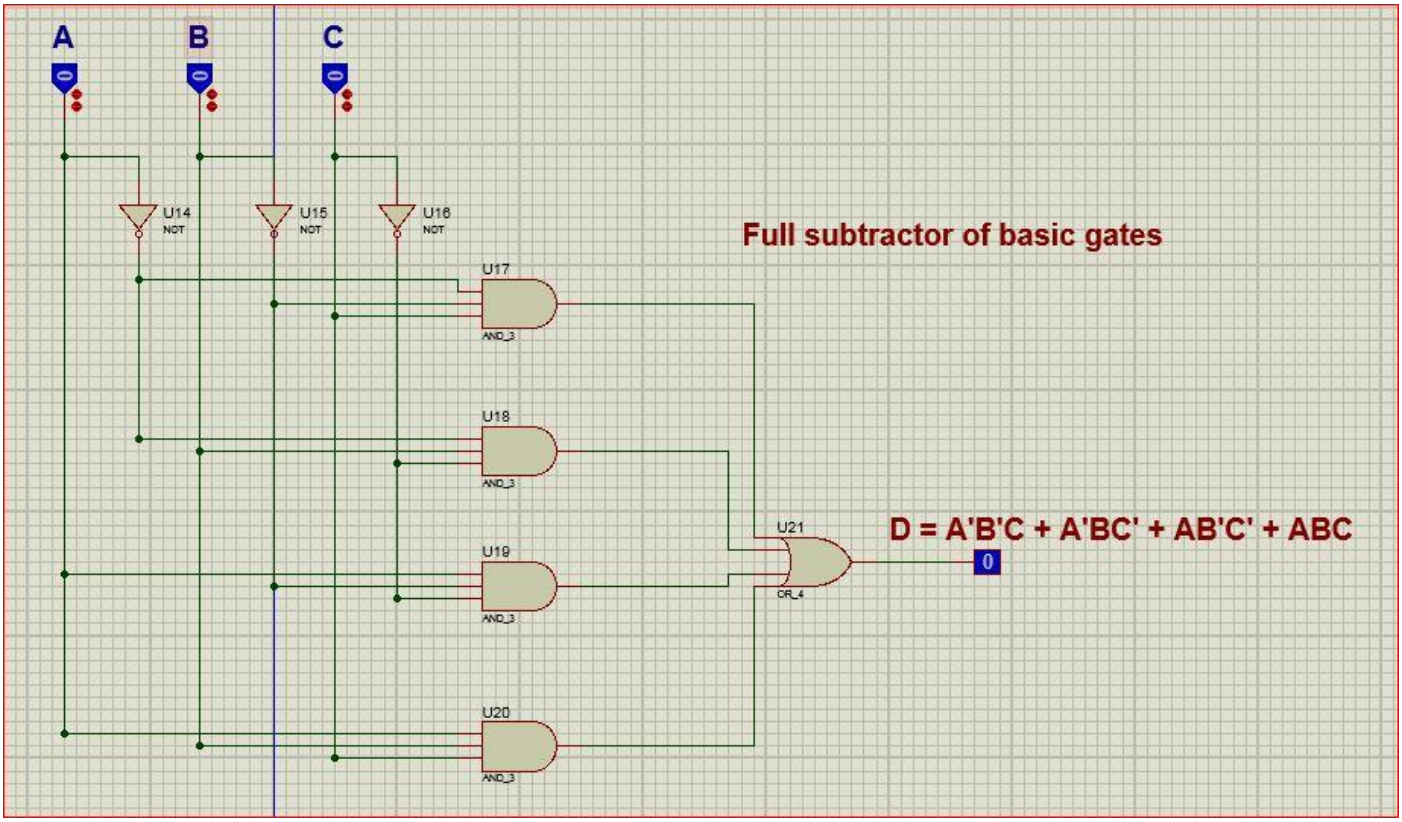

Half adder
X'Y+XY'
U1
XOR
U2
XY
AND
Half adder with OR & AND Gate
U7
NOT
U8
NOT
X'Y
U3
AND
U6
X'Y+XY'
OR
U4
XY'
AND
U5
XY
AND


X
Y
Z
U14
U15
U16
U17
U18
U19
U20
U21
S = X'Y'Z + X'YZ' + XY'Z' + XYZ
U22
U23
U24
U25
C = XY + YZ + XZ

Full Adder


Full Adder impliment With Half Adder
U9
XOR
U11
XOR
S = Z (+) (X) + (Y)
?
U10
AND
XY
U13
OR
C = XY + Z (X(+)Y)
?
U12
AND
Z (X (+) Y)


B & D of Half Subtractor
U3
NOT
D = A'B + AB'
U1
XOR
?
U2
AND
?
B = A'B



Full Subtractor

Lab-05

Name of Experiment : Design of code converters like BCD to Excess-3.

Describe : To understand the process of converting BCD to excess-3, it is requried to have knowledge of numbers system and numbers base conversion. The excess-3 binary code is an example of a self-complementary BCD code. A self-complementary binary code is a code which is always complimented in itself. By replacing the bit 1 to 0 and 0 to 1 of a numbers. The sum of all the 1's complement and the binary numbers of a decimal is equal to the binary numbers of decimal 9.

The process of BCD to excess-3. The Excess-3 code can be calculating added 3 (0011) to each BCD code.

Truth table

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

we will use the K-map method to design and make function for the conversion of BCD to Excess-3.

AB \ CP	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	x	x	x	x
10	1	1	x	x

$$w = A + BP + BC$$

AB \ CP	00	01	11	10
00		1	1	1
01	1	0	0	0
11	x	x	x	x
10	0	1	x	x

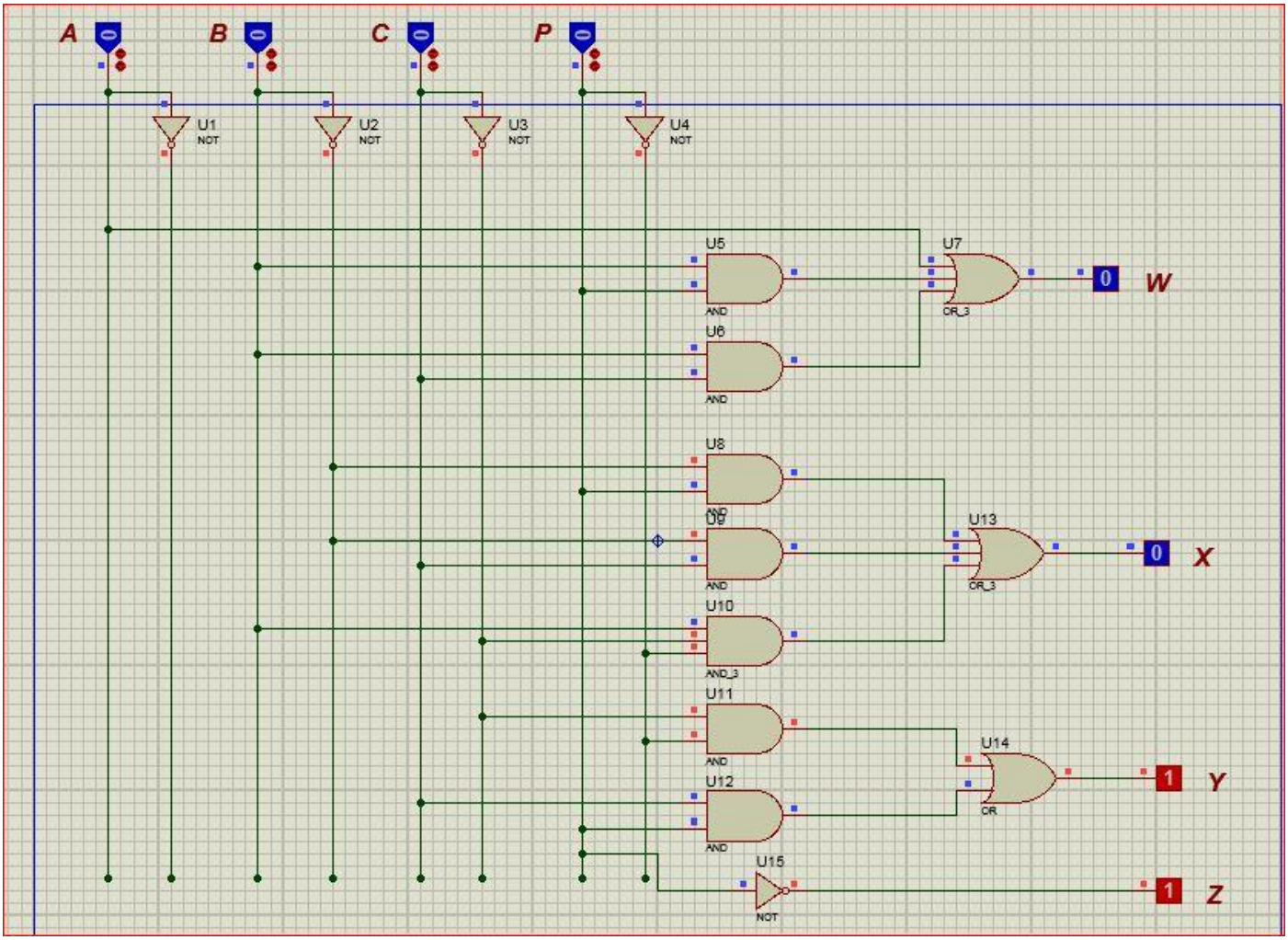
$$x = B'P + B'C + BC'P'$$

AB \ CP	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	x	x	x	x
10	1	0	x	x

$$y = C'P' + CP$$

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

$Z = D'$


A
B
C
P
U1
NOT
U2
NOT
U3
NOT
U4
NOT
U5
AND
U6
AND
U7
OR_3
0
W
U8
AND
U9
AND
U13
OR_3
0
X
U10
AND_3
U11
AND
U12
AND
U14
OR
1
Y
U15
NOT
1
Z

Conclusion:

(i) we have learnt how to convert BCD to Excess-3 code.

(ii) we have learnt that how to implement Code.

(iii) we also learnt that how to use k-map using to the BCD to Excess-03 code.

Lab Report → 05

ID: 19202103274

Name of Experiment : Construct and test combination Logic circuit parity Generator.

Equipment :

1. 8 input OR gate
2. 4 input AND gate
3. Not gate
4. Logic probe
5. Logic State

Description : A parity Bit is a Bit added to a string of Binary code. parity bit are a simple form of ecrcon detecting code. parity Bit are generally applied to the smallest units of a communication protocol, typically 8-bit octets (bytes). although they can also be applied separately to an entire massage string of bits. The parity bit ensure that

the total number of 1-bits in the string is even or odd. There are two various of parity bit: even parity bit and ODD parity Bit.

Even parity: parity Bit are added to transmitted massage to ensure that the number of bit with a value of one in aset of bits add up to even or odd numbers. Even and odd parities are the two variants of parity checking modes. even parity can be more clearly explained by means of an example Consider the transmitted massage 1010001, which has three ones in it. This turend into even parity by adding a one, making the sequence 1 1010001, so that there are four ones (an even number). If the transmitted massage has the form 1101001, which is already an even number, a zero is added to sustain the even parity.

Even_parity

This is the truth table of Even parity.

w	x	y	P	Even parity
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Ke-map

wx \ yp	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

$$f = w'x'y'p + w'x'yp' + w'xy'p' + w'xyp + wxy'p + wxy'p' + wxyp' + wx'yp$$

will Design the circuit in proteus Software.

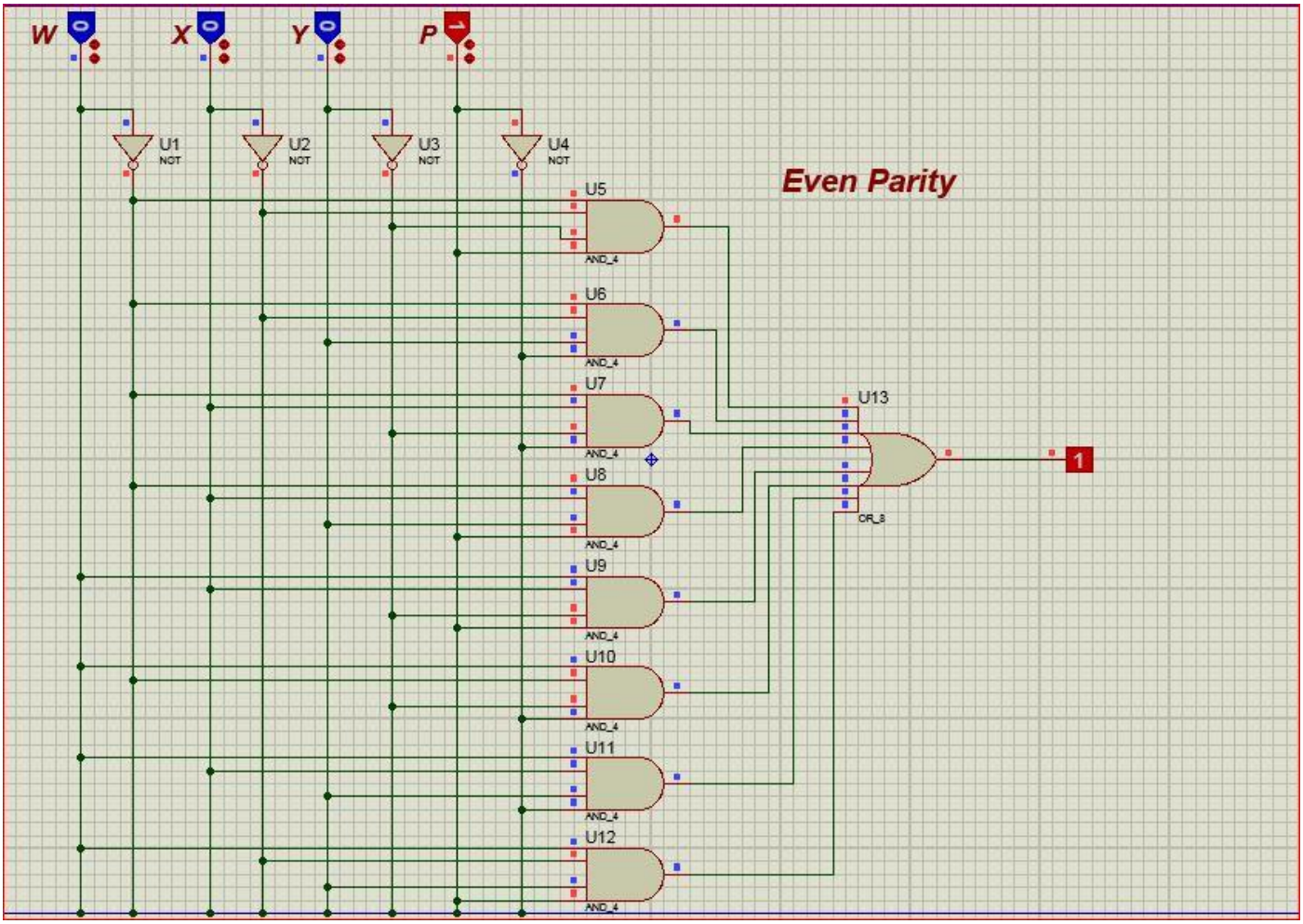
ODD parity: odd parity can be more clearly explained through an example. Consider the transmitted massage 1010001, which has three ones in it. This is turned into odd parity by adding a zero, making the sequence 0 1010001. Thus, the total number of ones remain at three, an odd number.

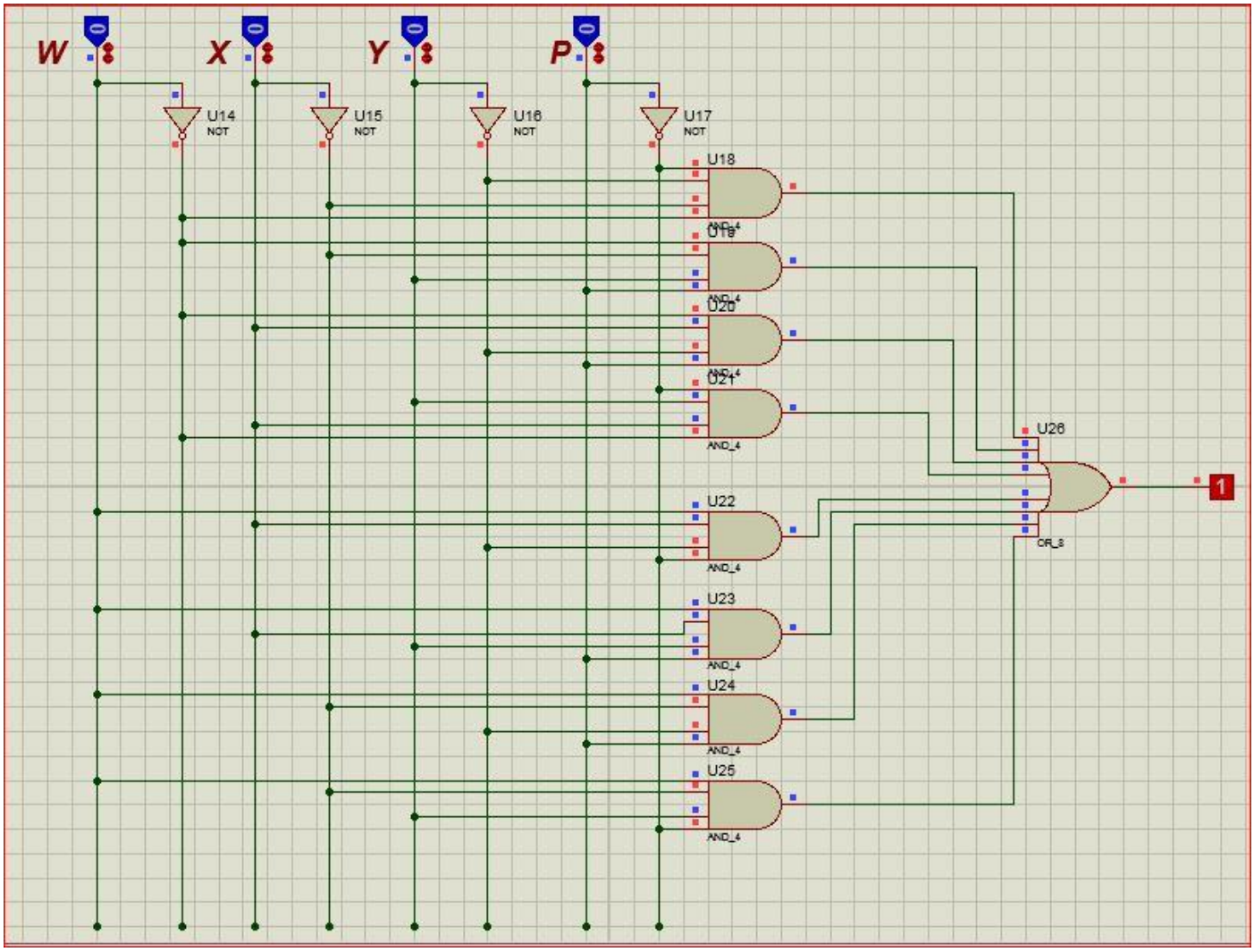
if the transmitted massage has the form 1101001, which has four ones in it, this can be turned into odd parity by adding a one, making the sequence 1 1101001.

K-map for odd parity

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

$$F = w'x'y'p' + w'x'yp + w'xy'p + w'xyp' + wxy'p' + wxyp + wx'y'p + wx'yp'$$


W
X
Y
P
U1
NOT
U2
NOT
U3
NOT
U4
NOT
Even Parity
U5
AND_4
U6
AND_4
U7
AND_4
U8
AND_4
U9
AND_4
U10
AND_4
U11
AND_4
U12
AND_4
U13
OR_8
1


W
X
Y
P
U14
NOT
U15
NOT
U16
NOT
U17
NOT
U18
U19
U20
U21
AND_4
U22
AND_4
U23
AND_4
U24
AND_4
U25
AND_4
U26
OR_8
1

ODD Parity

Lab-06

Lab-06

Name of Experiment: To check the operation of 2 to 4 line Decoder / 3 to 8 line Decoder and design circuit.

Equipment :

1. 3 input AND Gate
2. Not Gate
3. Logic probe
4. Logic State

Description : A decoder is a circuit that changes a code into a set of signals. It is called a decoder because it does the reverse of encoding, but we will began our study of ~~encoders~~ decoders.

A common type of decoder is the line decoder which taks an n-digit binary number and decodes into 2^n data lines. The simplest is the 2 to 4 Line decoders.

The truth table of 2 to 4 line decoders:

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

$D_0 = \overline{A}\,\overline{B}$

$D_1 = \overline{A}B$

$D_2 = A\overline{B}$

$D_3 = AB$

3 to 8 line Decoders

In 3 to 8 line decoders, it includes three inputs and eight outputs. Here the inputs are represented through [illegible], A, B, C. whereas the output are respresented through $D_0, D_1, D_2 - - D_7$

The selection of 8 output can be done based on three inputs.

Truth table of 3 to 8 line Decoder

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

$D_0 = \bar{A}\bar{B}\bar{C}$

$D_1 = \bar{A}\bar{B}C$

$D_2 = \bar{A}B\bar{C}$

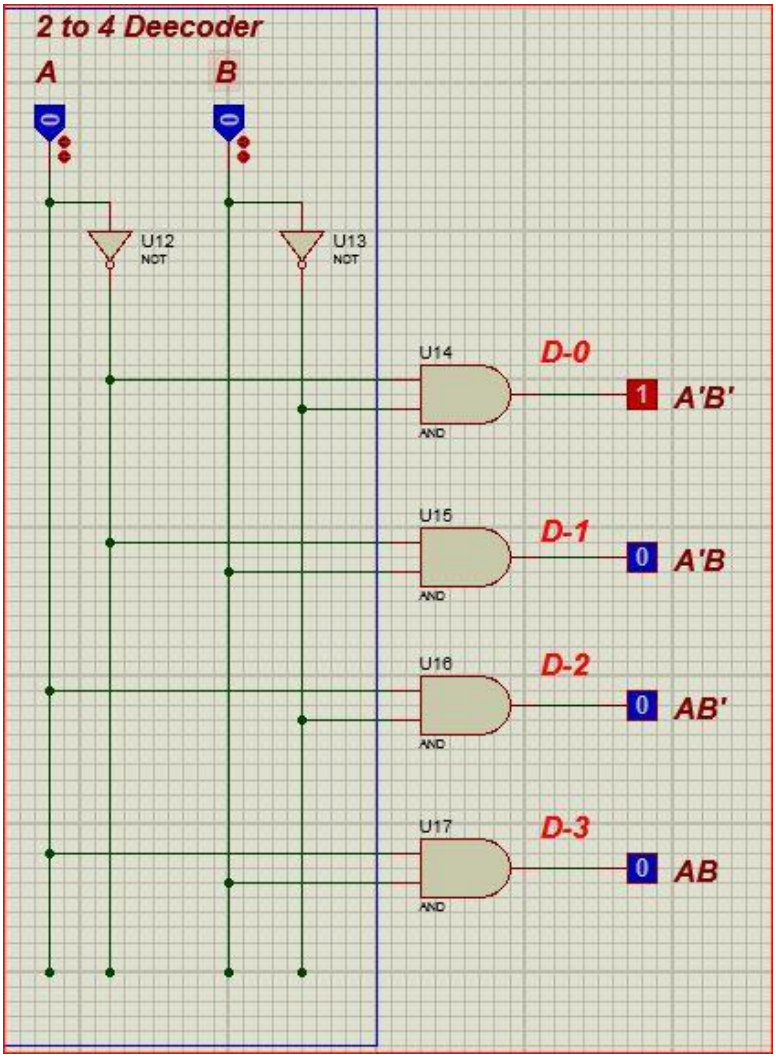
$D_3 = \bar{A}BC$

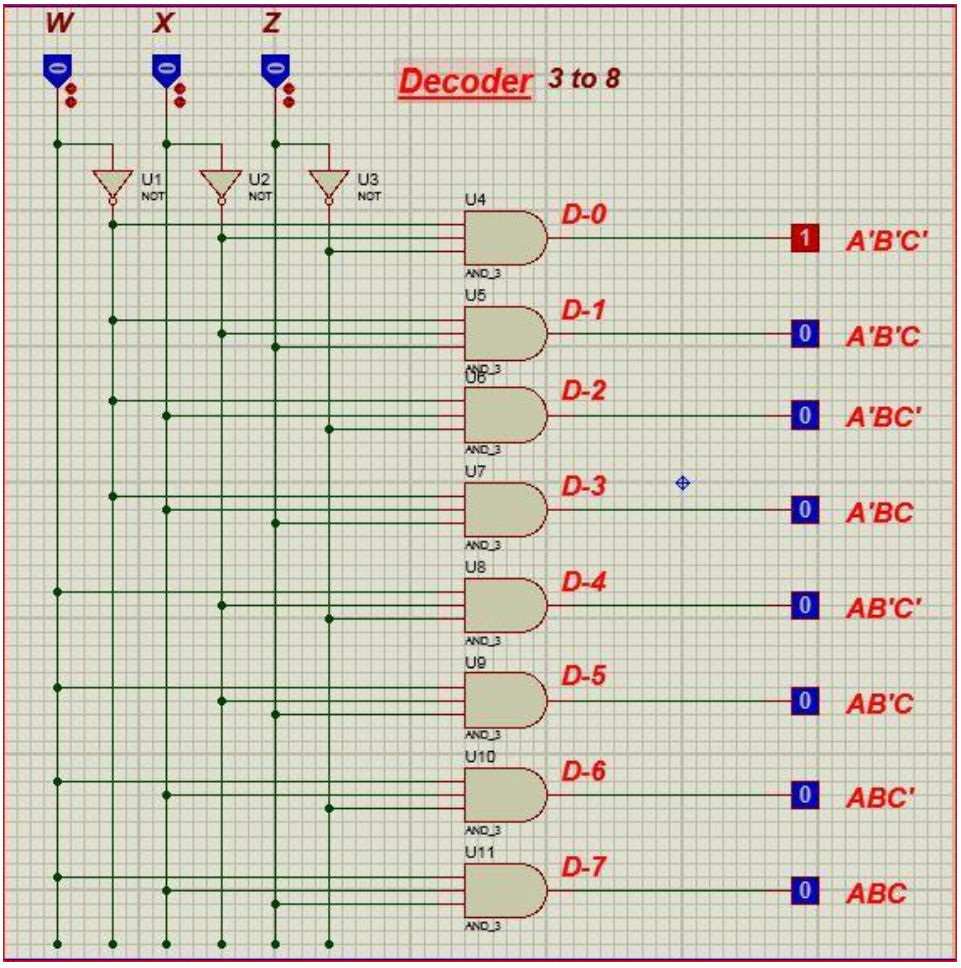
$D_4 = A\bar{B}\bar{C}$

$D_5 = A\bar{B}C$

$D_6 = AB\bar{C}$

$D_7 = ABC$


2 to 4 Deecoder
A
B
U12
NOT
U13
NOT
U14
D-0
1
A'B'
AND
U15
D-1
0
A'B
AND
U16
D-2
0
AB'
AND
U17
D-3
0
AB
AND


W
X
Z
Decoder 3 to 8
U1
NOT
U2
NOT
U3
NOT
U4
D-0
1
A'B'C'
AND_3
U5
D-1
0
A'B'C
AND_3
U6
D-2
0
A'BC'
AND_3
U7
D-3
0
A'BC
AND_3
U8
D-4
0
AB'C'
AND_3
U9
D-5
0
AB'C
AND_3
U10
D-6
0
ABC'
AND_3
U11
D-7
0
ABC
AND_3

Conclusion

1. We have learn that how to operate 2 to 4 line decoders.
2. We have also learn't how to operate 3 to 8 line decoders.
3. We have learn't how to find out the relation between input and output of a decoders.
4. We have also learn't how to implement a circuit of decoders using Basic gates.

Lab-07

Lab-07

Name of Experiment : To check the operation of Active Low decoders, Binary code Decimal encoders, Desiging of octal to Binary encoders and Hexadecimal to BCD encoders.

Equipment :

1. 2 input OR gates
2. 5 input OR gates
3. 8 input OR gate
4. Logic probe
5. Logic State
6. Not gate
7. 3 input AND gate.

Description : An encoders is a combinational circuit that performs the reverse operation of Decoders. It has maximum of 2^n input lines and 'n' output lines. It will produce a binary code equivalent to the input which is active High. Therefore the

the encoders encodes 2^n input lines with 'n' bits. it is optional to represent the enable signal in encoders.

Demux

Demultiplex (Demux) is the reverse of the multiplex progress — combining multiple unrelated analog or digital signal streams into one signal over a single shared medium, such as a single conductor of copper wire wire or fiber optic cable.

The demux is also known as 2-to-4 Demultiplexer which means that it has two select line and 4 output line.

truth table of Demux

E	A	B	D_0	D_1	D_2	D_3
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

Function :

$D_0 = \overline{\overline{E_0 + x + y}}$

$= E'x'y'$

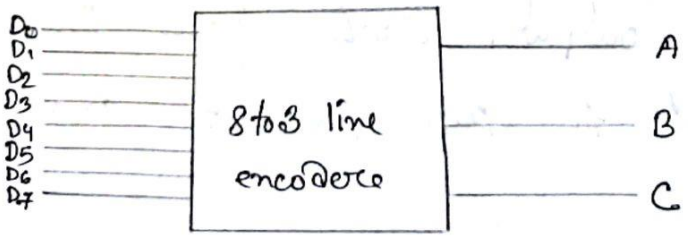
$D_1 = \overline{E'x'y}$

$D_2 = \overline{E'x'y'}$

$D_3 = \overline{E'xy}$

8 to 3 line Encoder

The 8 to 3 line encoder or octal to Binary encoder consist of 8 inputs : E_0 to E_7 and 3 outputs A B C. Each input line corresponds to each octal digit and three outputs generate corresponding binary code.



8 to 3 line encoders truth table

E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	A	B	C
1								0	0	0
	1							0	0	1
		1						0	1	0
			1					0	1	1
				1				1	0	0
					1			1	0	1
						1		1	1	0
							1	1	1	1

Function of 8 to 3 line Encoder

$$A = E_1 + E_3 + E_5 + E_7$$

$$B = E_2 + E_3 + E_6 + E_7$$

$$C = E_4 + E_5 + E_6 + E_7$$

Decimal to BCD Encoders

The 10 to 4 line encoders consist of 10 input this is the decimal numbers and 4 output this is the Binary coded Decimal numbers system. it contains 4 output. Each input line corresponds to each Decimal to BCD generate corresponding binary code.

Truth table of Decimal to BCD encoders

E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	A	B	C	D
1										0	0	0	0
	1									0	0	0	1
		1								0	0	1	0
			1							0	0	1	1
				1						0	1	0	0
					1					0	1	0	1
						1				0	1	1	0
							1			0	1	1	1
								1		1	0	0	0
									1	1	0	0	1

Function of Decimal to BCD Encoders

$A = E_8 + E_9$

$B = E_4 + E_5 + E_6 + E_7$

$C = E_2 + E_3 + E_6 + E_7$

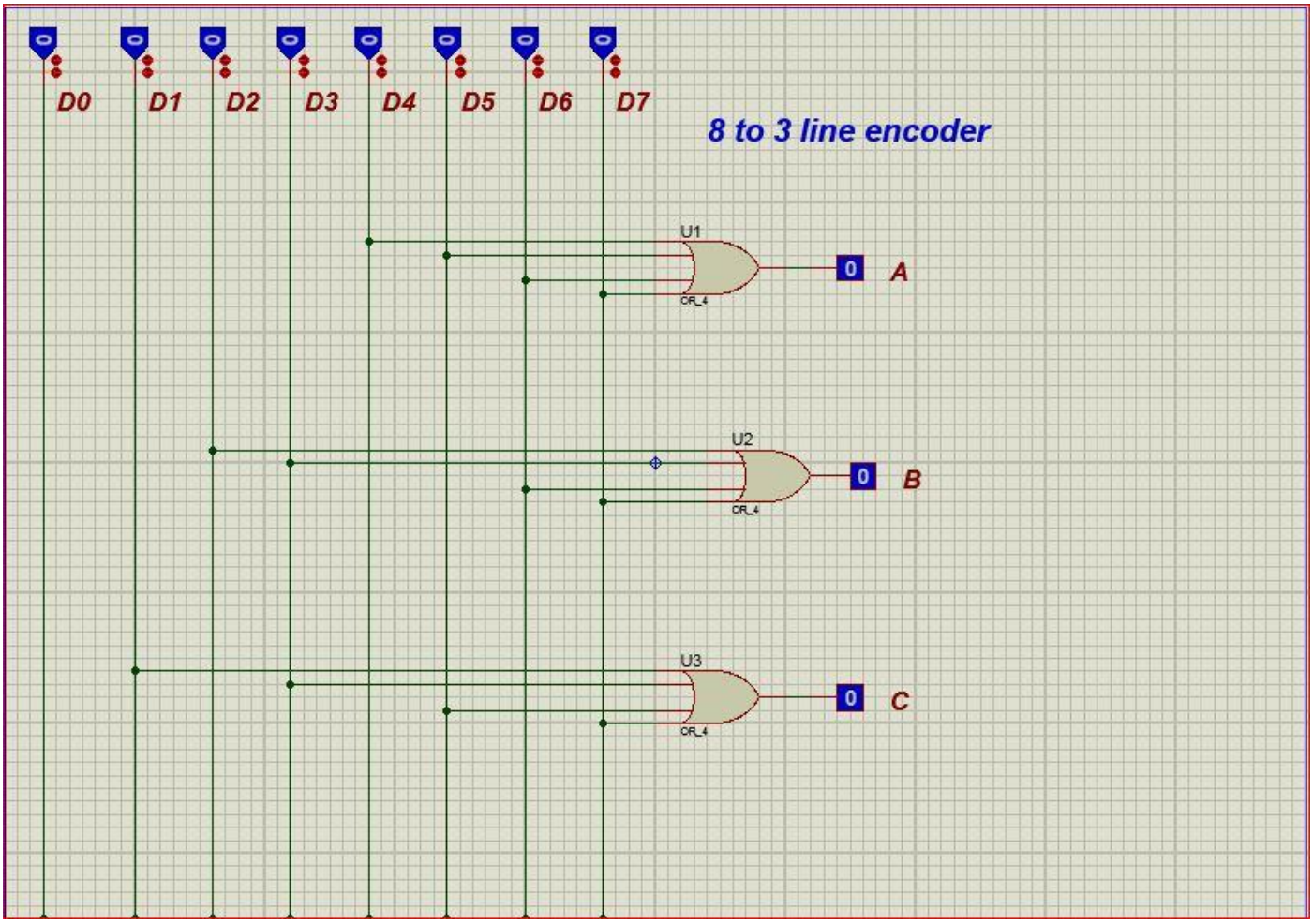
$D = E_1 + E_3 + E_5 + E_7 + E_9$

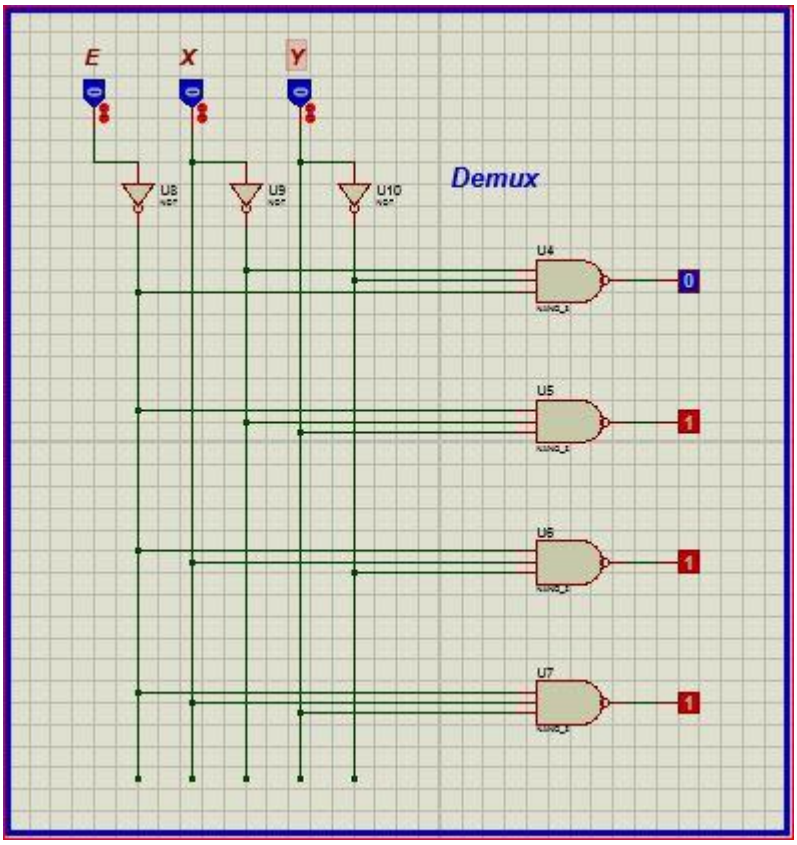
16 To 4 line Encoders

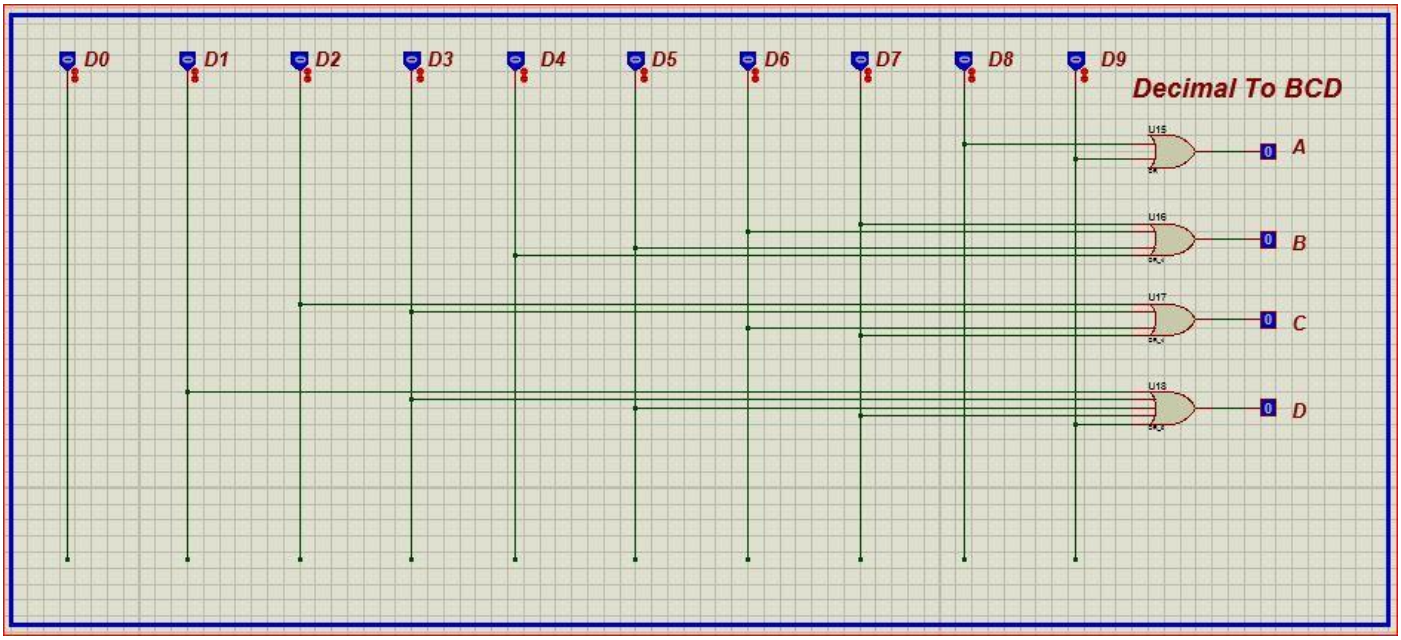
The Hexadecimal to Binary encoders usually consists of 16 line to 4 output lines. Each input line corresponds to the each decimal digit and 4 outputs corresponds to the BCD code. This encoders accepts the decoded decimal data as an inputs and encodes it to the BCD output which is avilable on the output lines.

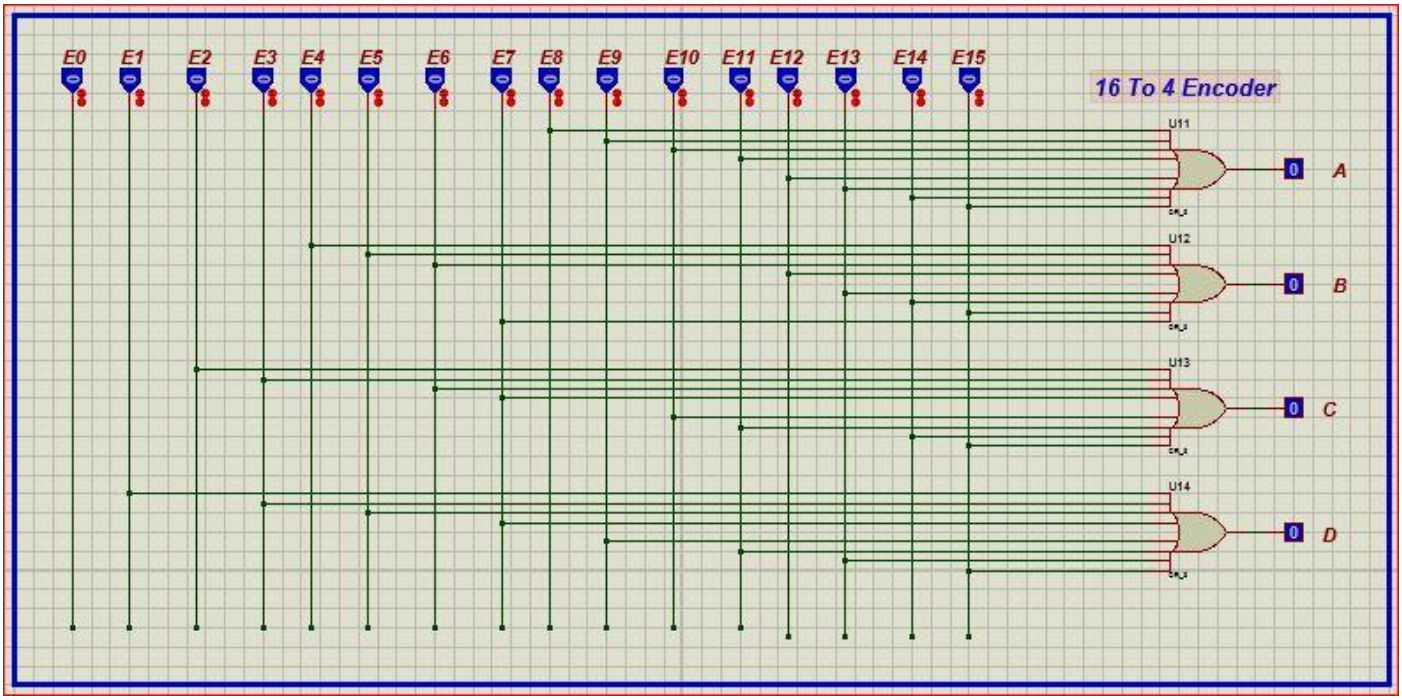
Truth table of 16 to 4 line encoders

E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	E_{11}	E_{12}	E_{13}	E_{14}	E_{15}	A	B	C	D
1																0	0	0	0
	1															0	0	0	1
		1														0	0	1	0
			1													0	0	1	1
				1												0	1	0	0
					1											0	1	0	1
						1										0	1	1	0
							1									0	1	1	1
								1								1	0	0	0
									1							1	0	0	1
										1						1	0	1	0
											1					1	0	1	1
												1				1	1	0	0
													1			1	1	0	1
														1		1	1	1	0
															1	1	1	1	1


D0
D1
D2
D3
D4
D5
D6
D7
8 to 3 line encoder
U1
OR_4
0
A
U2
OR_4
0
B
U3
OR_4
0
C


E
X
Y
U8
NOT
U9
NOT
U10
NOT
Demux
U4
0
U5
1
U6
1
U7
1


D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
Decimal To BCD
U15
U16
U17
U18
A
B
C
D


E0
E1
E2
E3
E4
E5
E6
E7
E8
E9
E10
E11
E12
E13
E14
E15
16 To 4 Encoder
U11
U12
U13
U14
A
B
C
D

Conclusion:

① We have learn't that How work Demux and Encoders.

② We have learn't that the difference between Decoders and Encoders.

③ We also learn that how to Design Decimal to BCD Encoders.

④ We also learn that how to Design Octal to BCD Encoders.

⑤ We have learn that how to Design Hexa to BCD Encoders.

⑥ We also learn how to implement those encoders using basic gates in proteus.

Lab – 08

Lab-08

Name of Experiment: Implementation of mux, 2 to 1 mux, 4 to 1 mux, quadruple 2 to 1 line mux, 1 to 4 line demux.

Equipment:

1. 2 input AND Gate
2. Not Gate
3. 3 input AND Gate
4. 4 input OR Gate
5. 4 input AND Gate.
6. Logic probe
7. Logic state

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

Sins there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each

Combination can select only one output. De-multiplexers is also called as De-Mux.

Implement 2 to 1 line multiplexers:

A 2 to 1 multiplexers consists of two inputs D_0 & D_1. One select input s and one output Y. Depending on the select signal, the output is connected to either of the inputs. Since there are two input signals, only two ways are possible to connect the inputs to the outputs. So One select in needed to do these operations. if the select line is low, then the output will be switched to D_0 input, whereas if select line is high, then the output will be switched to D_1 input.

The value of the select input, the inputs. D_0, D_1 are produced at outputs. The output is D_0 when select value is $S = 0$ and the output is D_1 when select value is $S = 1$.

- The truth table of 2 to 1 mux

S	D_0	D_1	Y
0	0	x	0
0	1	x	1
1	x	0	0
1	x	1	1

Equation is $Y = \bar{S}D_0 + SD_1$

Implement 4 to 1 Mux

A 4 to 1 mux consists four data inputs lines as D_0 to D_3, two select lines as S_0 and S_1 and single output line Y. The select lines S_0 and S_1 select one of the four input lines to connect the output line. The four input combinations 00, 01, 10, 11 on the select lines respectively switches the input D_0, D_1, D_2 D_3 to the output. That means when $S_0 = 0$ and $S_1 = 0$ the output at Y is D_0, similarly Y is D_1 if the selected inputs $S_0 = 0$ and $S_1 = 1$ and so on.

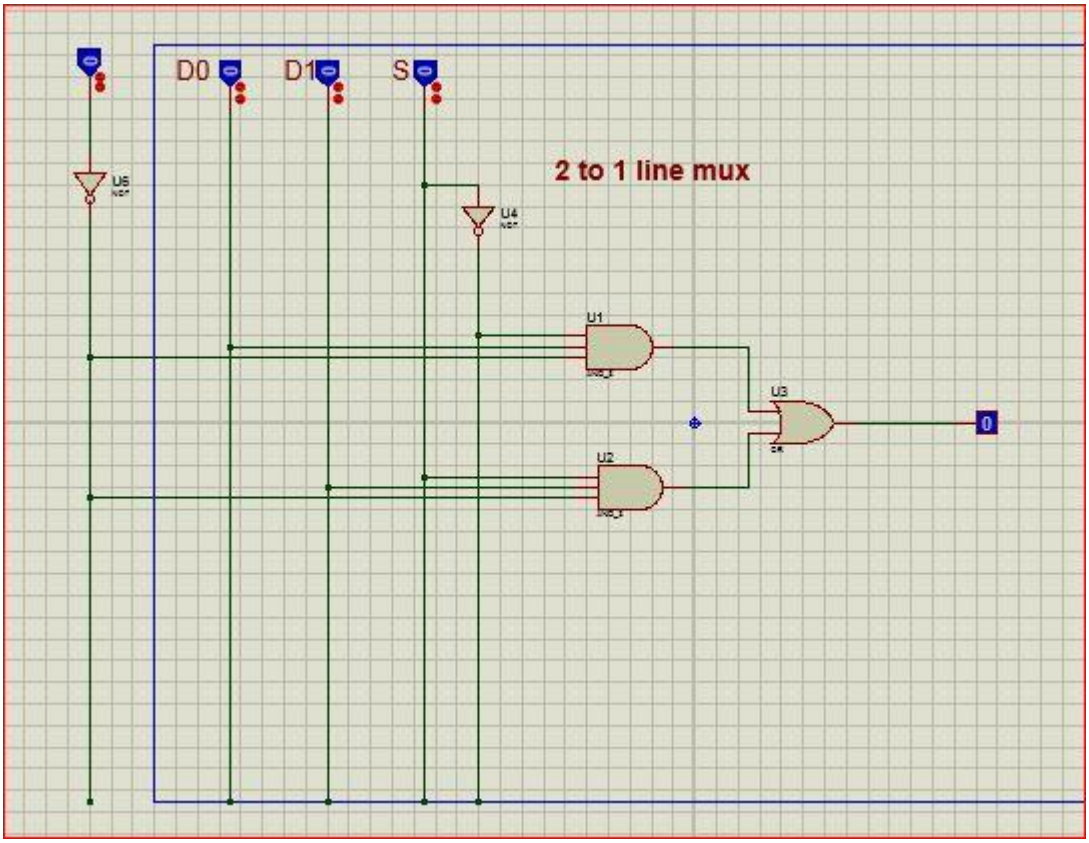
For 2 to 1 line quadruple mux

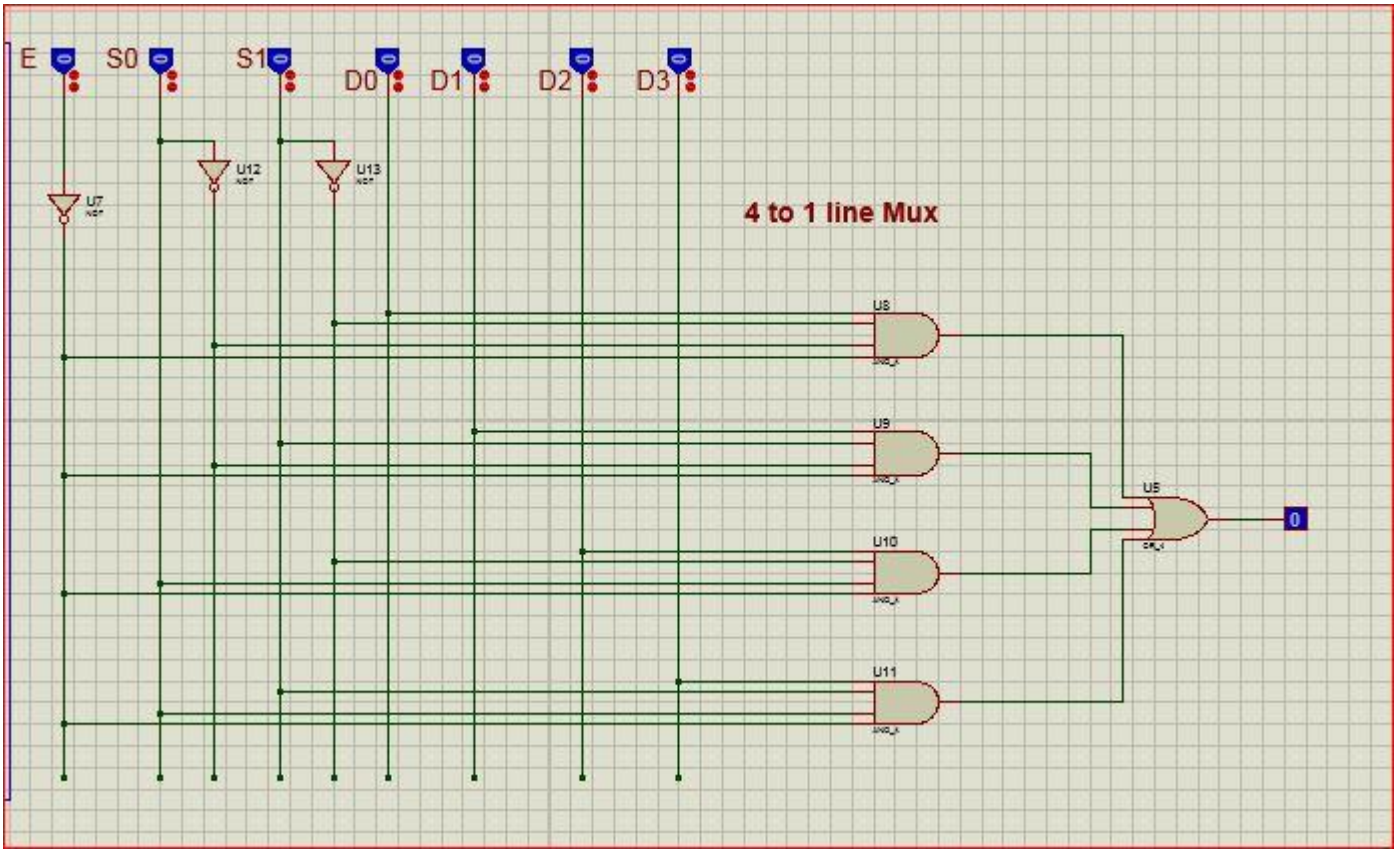
S	output
0	A
1	B

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

S_1	S_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Truth Table of 1 to 4 line demux


D0
D1
S
2 to 1 line mux
U6
U4
U1
U3
U2
0


E
S0
S1
D0
D1
D2
D3
U12
U13
U7
4 to 1 line Mux
U8
U9
U10
U11
0

Lab -09

Lab-09

Name of Experiment: Implement Designing of 4 bit magnitude comparator and magnitude comparison using IC. Implementing a full Adder using (a) Decoders (b) Multiplexers.

Equipment:

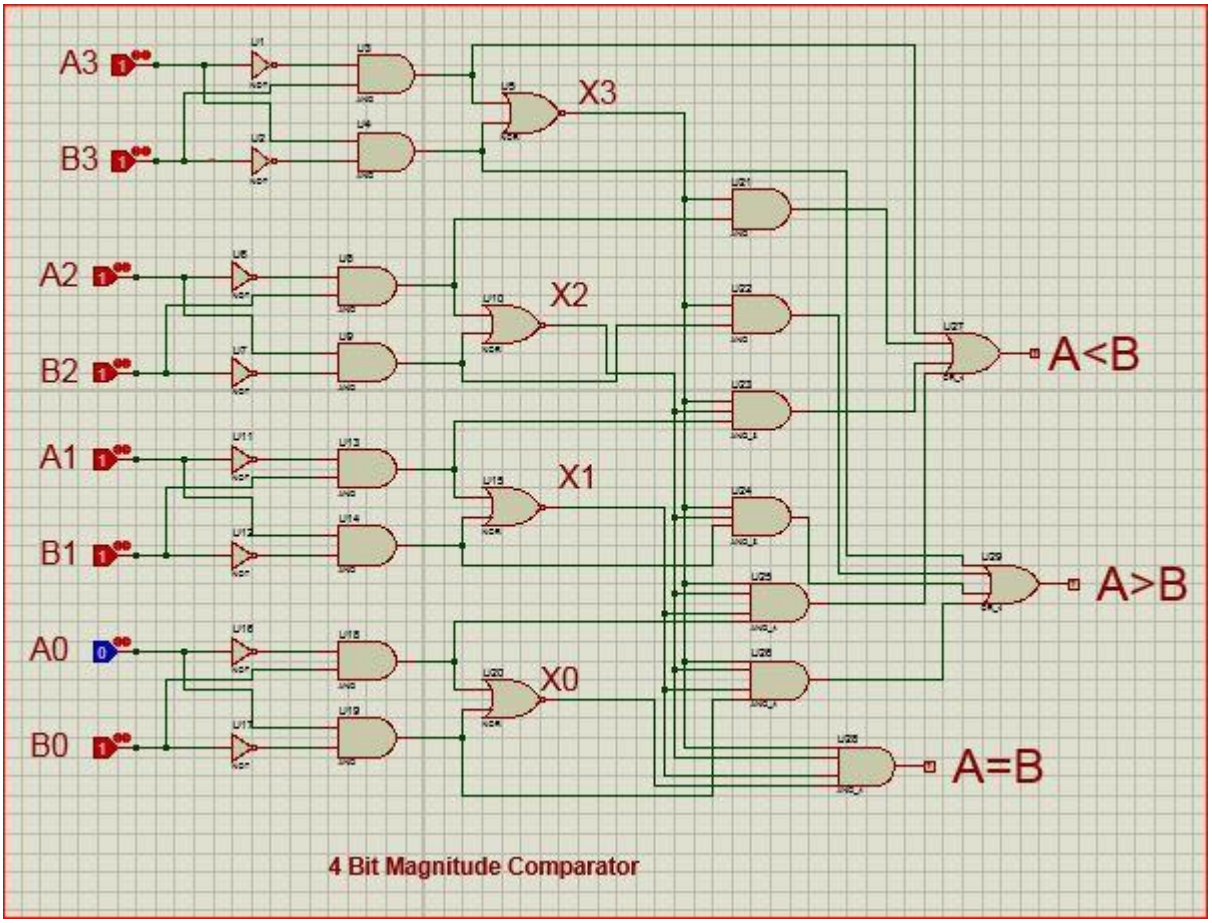
1. 2 input AND
2. 2 input OR
3. 4 input OR
4. 4 input AND
5. Logic state
6. Logic probe
7. Not Gate.

Description: mangnitude comparator is a combinational circuit that compares two digital or binary numbers in order to find out whether one binary number is equal, less than or greater than the other binary numbers.

We logically design a circuit for which we will have two inputs one for A and others for B and have three output terminals, one for $A > B$ condition, one for $A = B$ condition and one for $A < B$ condition.

4 bits magnitude Comparator:

A comparator used to compare two binary numbers each of four bits called a 4 bit magnitude comperator. It consists of eight inputs each for two four bit numbers and three outputs to generate less than, equal to and greater than between two binary numbers.


A3
B3
A2
B2
A1
B1
A0
B0
X3
X2
X1
X0
A<B
A>B
A=B
4 Bit Magnitude Comparator

Conclusion:-

1. we also learned How to works Comparators are used in central processing units.
2. we also learned How controls magnitude Comparator.
3. we also learned comperators are also used as process controllers.
4. we also learned How to design this Circuit in protues.

Lab-10

Lab :- 10

Experiment name:- Implementation of Construct, test and investigate the operation of various Flip-Flop, SR Flip-Flop, RS Flip-Flops, D Flip-Flop etc.

Equipment:

1. 2 input AND
2. 3 input AND
3. 4 input AND
4. Logic probe
5. Logic state
6. JK FF
7.

Description:- A Latch is a sample circuit that responds by switching its output between two states on the application of certain inputs. A digital Latch is the building block of sequential circuits. It is made using NOR or NAND Logic Gates. Latches have a feedback system. This means that the output of the Latch is given back to its inputs. This indicates that the outputs of a Latches depend on current as well as previous

inputs. A Latches has two inputs and two outputs. The outputs are complementary to each other.

SR Latch using NAND Gate

The construction is simple. Two inputs, S and R. Two outputs that are complementary to each others, Q and Q'. And the outputs are fed back to the gates. The SR Latch truth table and working of the SR Latch are given below.

Truth table of SR Latch.

S	R	Q	Q'
0	0	No Change	No Change
0	1	1	0
1	0	0	1
1	1	Not allowed	Not allowed

RS Latches using NOR Gate:

RS Latches have two inputs. S and R, S is called set and R called reset. The S input is used to produce High on Q. The R input is used to produce Low on Q. Q' is Q complementary output, so it always holds the opposite value of Q. The output of the S-R Latch depends on current as well as previous inputs or states. and its state (value stored) can change as soon as its inputs change. The circuit and the truth table of RS latch is shown below.

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

SR Flip-Flop :

There are majorly 4 types of flip-flop, with the most common one being SR Flip-flop. This simple flip-flop circuit has a set input (S) and reset input (R). In this system, ~~either~~ Once the outputs are established, the wiring of the circuits is maintained until "S" or "R" go high, or power is turned off.

As shown Adobe. it is the simplest and easiest to understands. The two outputs, as shown above, are the inverse of each others. The truth table of SR Flip-Flop is highlighted below.

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

JK Flip-Flop :

The JK Flip-Flop is an improvement on the SR flip-flop, where S=R=1 is not a problem. The input condition of J=K=1, gives an output inverting the output state. However, the outputs are the same when one tests the circuit practically.

JK Flip-Flop can function as set or reset.

J	K	Q	Q'
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	1
0	1	1	0
1	0	1	1
1	1	1	0

D Flip-flop :

D flip-flop is a better alternative that is very popular with digital electronics. They are commonly used for counters and shift registers and input synchronisation.

D	clock	Q	Q'
0	0	0	1
1	1	0	1
0	0	0	1
1	1	1	0

T Flip-flop :

A T flip-flop is like a JK flip-flop. These are basically a single input version of JK flip-flop The modified form of JK flip-flop is obtained by connecting both inputs J and K together. It has only one input along with the clock input.

Truth table of T flip-flop

T	Q	Q'
0	0	0
1	0	1
0	1	1
1	1	0

Conclusion:

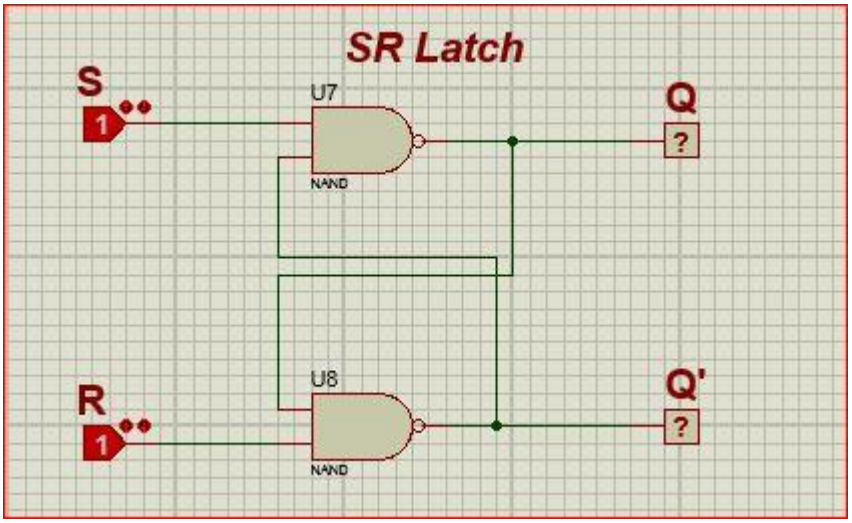
① we learned How to work Flip-Flop.

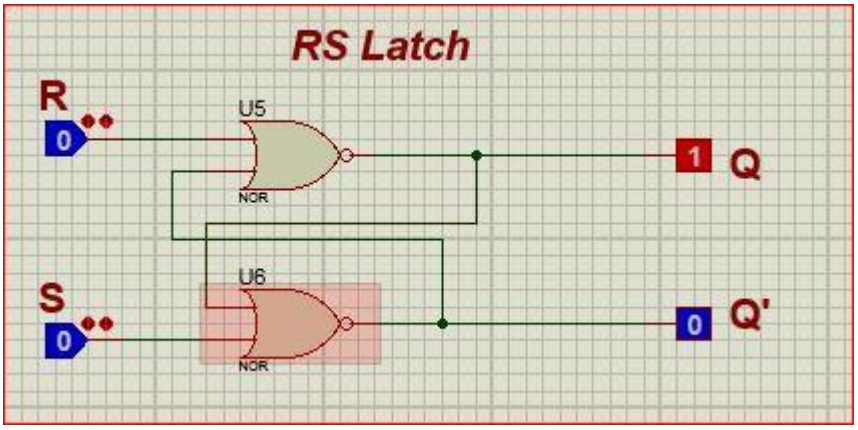
② we learned How to work Latches.

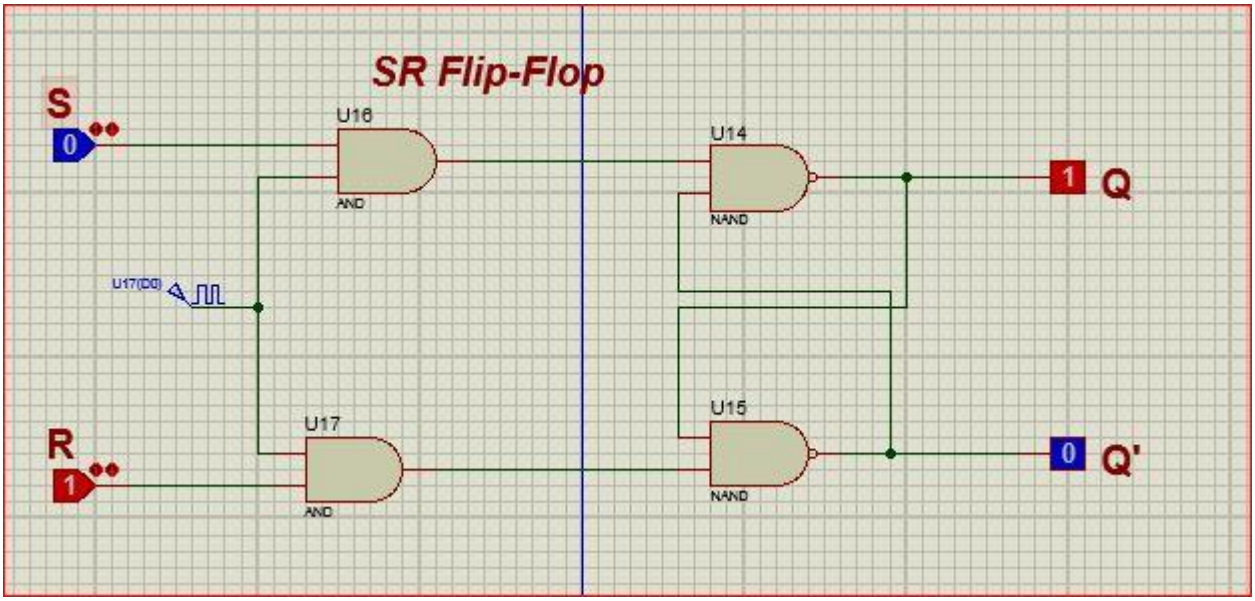
③ we also learned How to work SR Flip-Flop, RS flip-flop, JK flip-flop, T flip-flop.

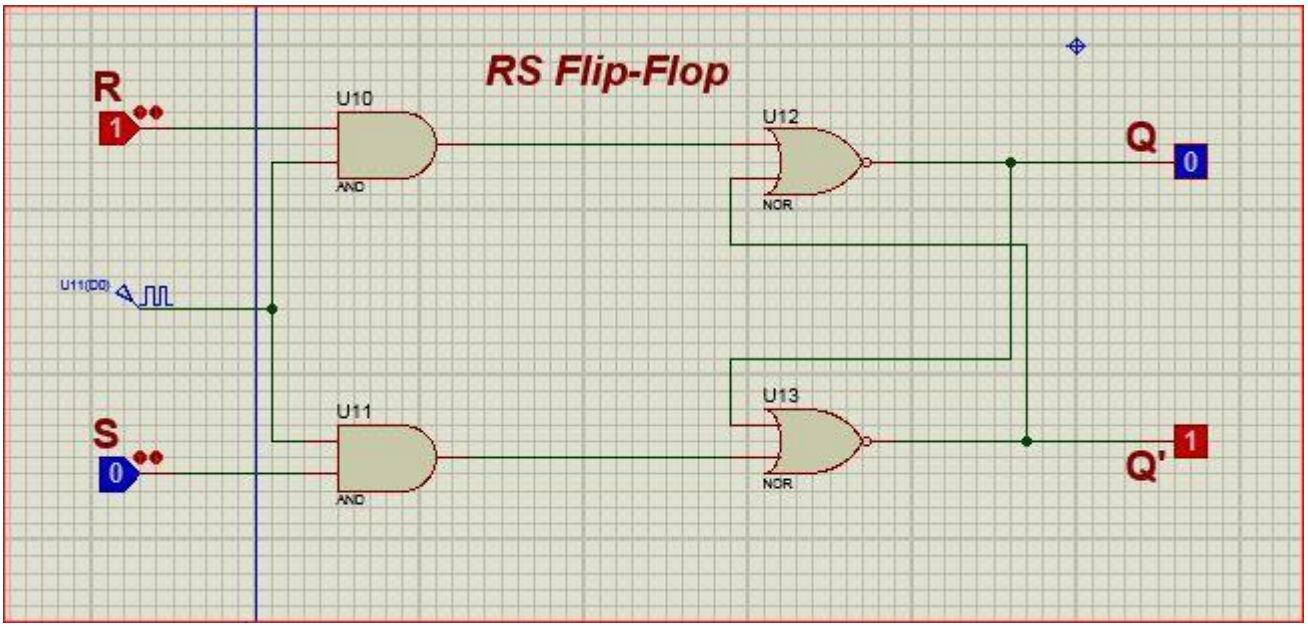
④ we also learned How to design this flip-flop protues.

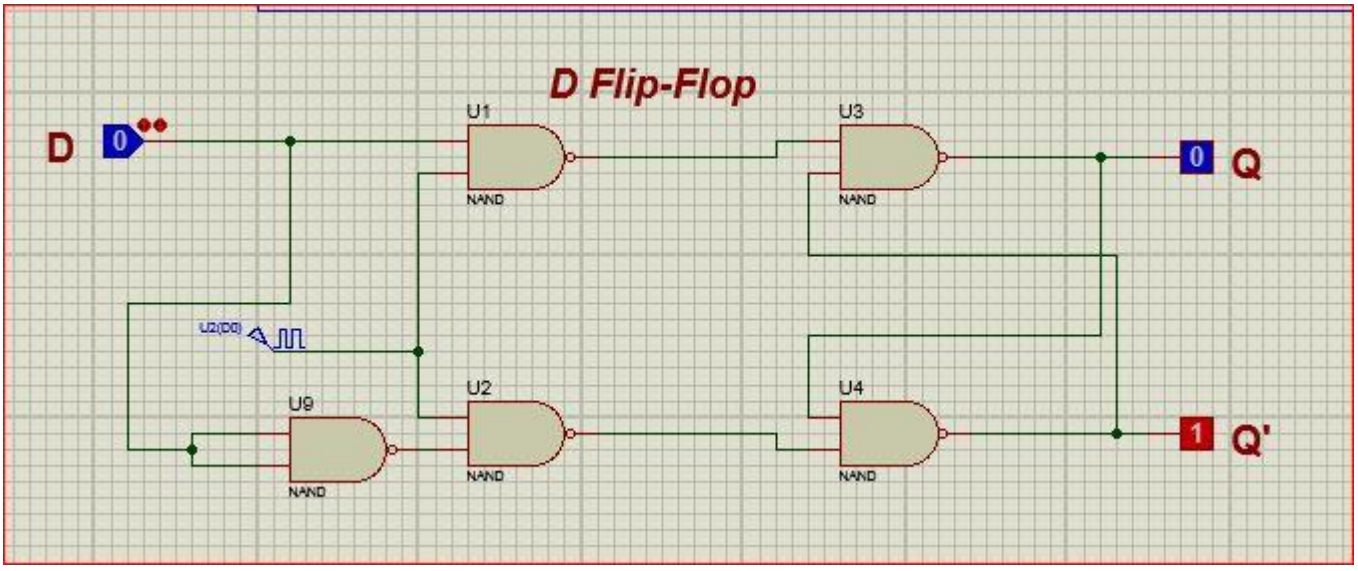
⑤ we also learned This Lab very easily.

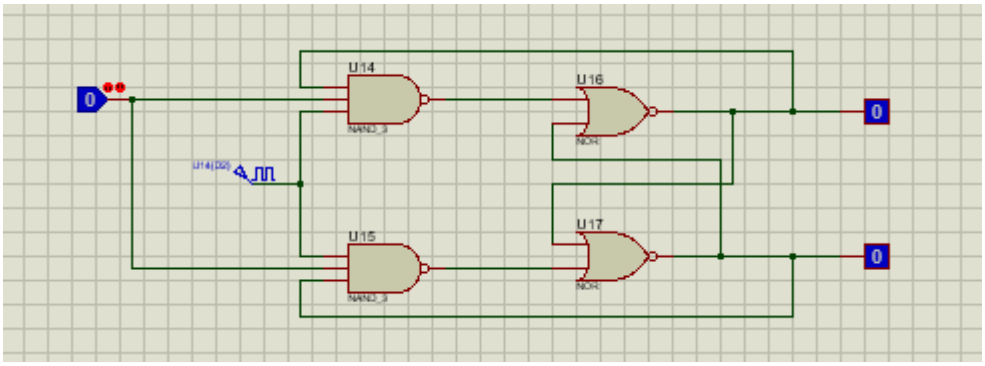

SR Latch
S
1
U7
NAND
Q
?
R
1
U8
NAND
Q'
?


RS Latch
R
0
U5
NOR
1
Q
S
0
U6
NOR
0
Q'

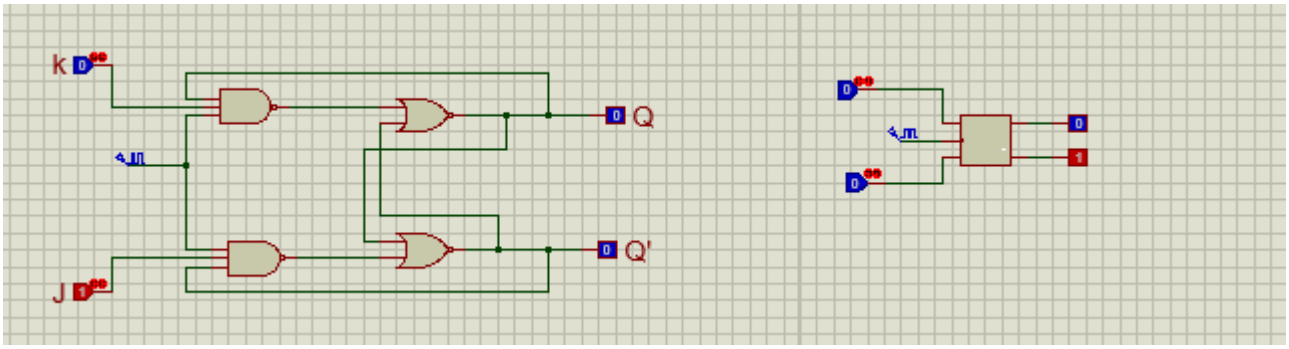

SR Flip-Flop
S
0
U16
AND
U14
NAND
1
Q
U17(DO)
R
1
U17
AND
U15
NAND
0
Q'


RS Flip-Flop
R
1
S
0
U10
AND
U11
AND
U12
NOR
U13
NOR
Q
0
Q'
1
U11(D0)


D Flip-Flop
D
0
U1
NAND
U2
NAND
U3
NAND
U4
NAND
U9
NAND
0
Q
1
Q'
U2(D0)


0
U14
U15
U16
U17
0
0

T Flip-Flop



JK Flip-Flop

Lab – 11

Lab-11

Name of Experiment: Construction of Counters and registers.

Equipment:

1. 2 input AND
2. 4 input AND
3. 3 input AND
4. JKFF
5. Logic probe
6. Logic State.

Description: Register is a group of flip-flop. Its basic function is to hold information within a digital system so as to make it avilable to the logic units during the computing process.

Counters – is essentially a register that goes through a predetermined sequence of states.

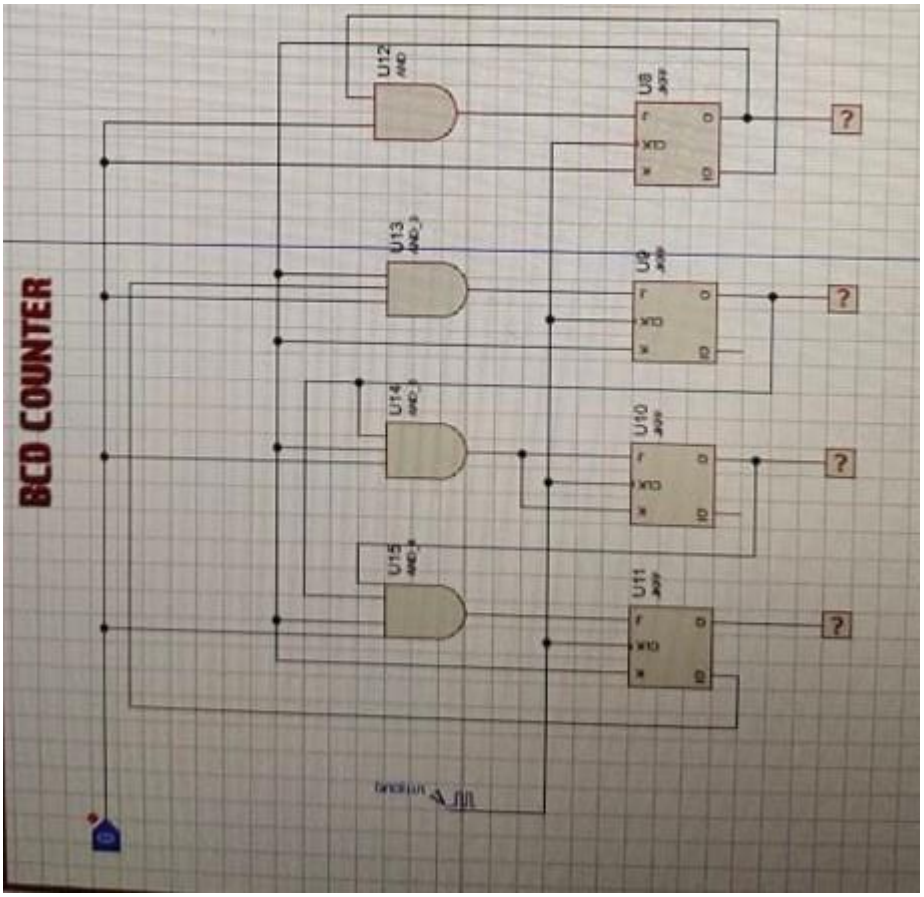
The gates in the counters are connected in such a way as to produce the prescribed sequence of binary states.

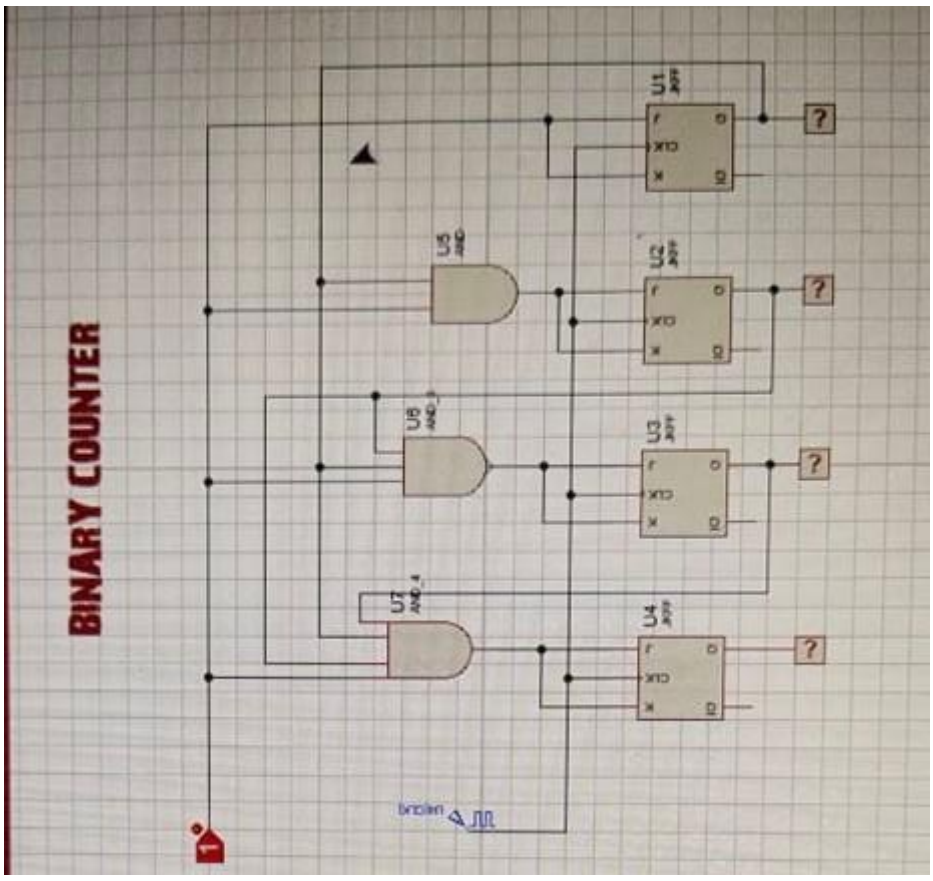
BCD Counters:

A BCD Counters is a series type of digital Counters. which is design to count 10 digit. It performs the operation of resetting automatically when there is a new clock input signal. As the counters counts ten unique combination of the applied input. it is termed a decode Counters. A four bit decode counters will operate as a BCD counters by skipping 6 output from the 24 outputs.

Binary Counter:

A binary counter is a hardware circuit that is made out of a series of flip-flop. The outputs of one flip-flop is sent to the inputs of the next flip-flop in the series. A binary Counter can be either asynchronous or Synchronous, depending on how the flip flop are connected together.


BCD COUNTER
U12
AND
U8
JKFF
U13
AND_3
U9
JKFF
U14
AND_3
U10
JKFF
U15
AND_4
U11
JKFF
?
CLK
J
K
Q
nQ


BINARY COUNTER
U1
JKFF
U5
AND
U2
JKFF
U6
AND_3
U3
JKFF
U7
AND_4
U4
JKFF
?
CLK
J
K
Q
nQ

Conclusion:

1. We have learned about Counters.
2. We have learned how to implements BCD Counter.
3. We also learned how to implement Binary counter.
4. We also have learned that how to implement this BCD Counter on protues.

The END