
# Lab Report

# CSE – 206

## Presented By:


- **Name:** Tunazzinur Rahman Kabbo
- **Intake:** 44
- **Sec:** 07
- **ID:** 19202103268


CSE – 206

## Presented To:


❖ **Iffat Tamanna**
Lecturer, BUBT
Department of Computer Science & Engineering
Email: iffat@bubt.edu.bd

# LAB - 01

Name of the experiment: To verify the behavior of logic gates using truth table and implementation of all basic logic gates using Proteus.

Equipments:

(i) Proteus Software.
(ii) And Gate (7408)
(iii) OR Gate (7432)
(iv) NOT Gate (7404)
(v) NAND Gate (7400)
(vi) NOR Gate (7402)
(vii) XOR Gate (XOR)
(viii) XNOR Gate (4077)
(ix) Logic Probe.
(x) Logic state

## Description:

Logic gates are like devices that acts as a building block for digital circuits. They perform basic logical functions that are fundamental to digital circuits. Most electronic devices we use today will have some from of logic gates in them. For example, logic gates can be used in technologies such as smartphones etc. Logic gates are based on Boolean Algebra (0 and 1).

## Truth Table:

### AND Gate

=> A . B

| A | B | A.B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Output

Input

## OR Gate

=> A+B

| A | B | A+B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Output

Input

## NOT Gate

=> $\overline{A}$

| Input A | Output $\overline{A}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

## NAND Gate

=> $\overline{A.B}$

| A | B | A.B | $\overline{A.B}$ |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Output

Input

## NOR Gate

$\Rightarrow \overline{A+B}$

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

Output

Input

## XOR Gate

$\Rightarrow A \oplus B$

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

| A | B | $A \oplus B$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Output

Input

## XNOR Gate

$\Rightarrow A \odot B$

$= \overline{A \oplus B}$

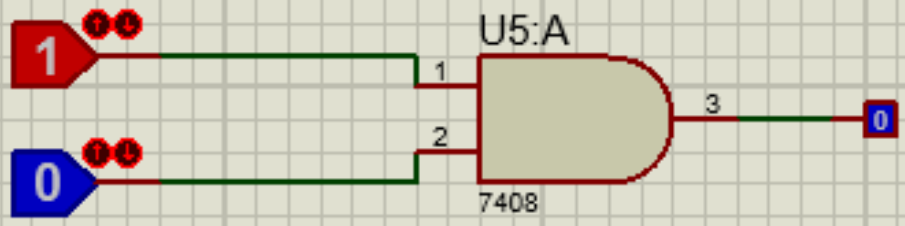
$= (\bar{A}+B).(A+\bar{B})$

| A | B | $A \oplus B$ | $\overline{A \oplus B}$ |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

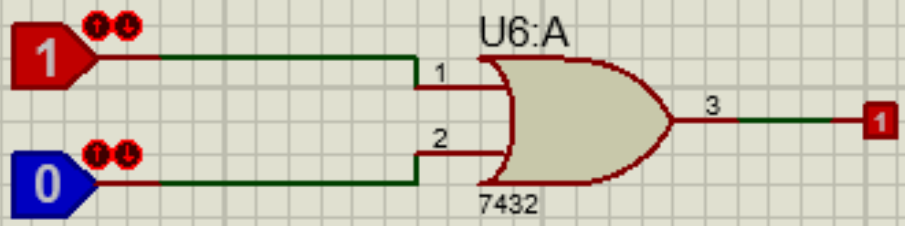
Output

Input

AND

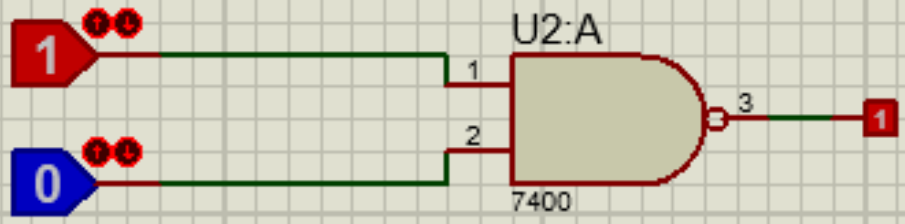



OR

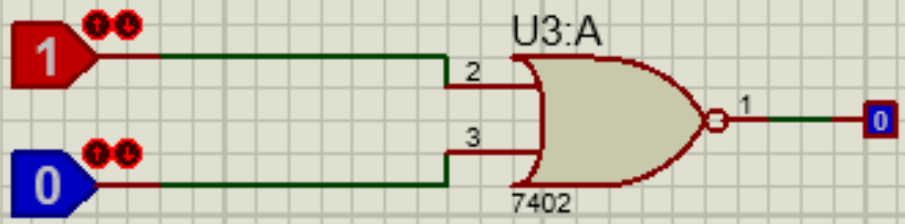



NOT

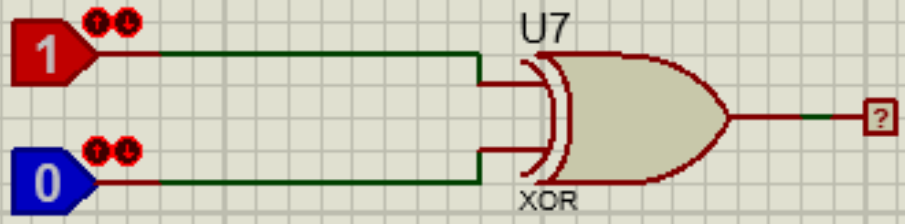

NAND



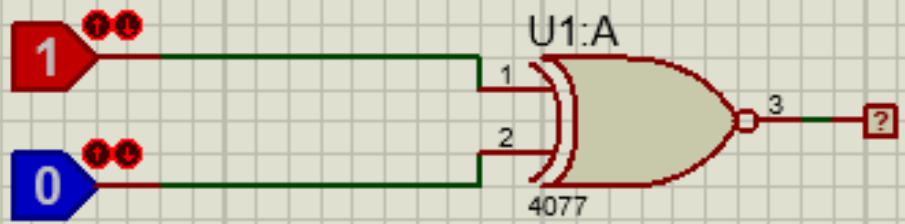

NOR




XOR




XNOR



## Conclusion:

(i) We have learnt how to implement circuits in Proteus Software.

(ii) We have understood the digit simulation of any circuit in the software.

(iii) we have verified the truth table for each input/output combination.

(iv) We repeated the process for all other logic gates.

# LAB - 02

Name of the experiment: Implementation of basic logic gates using universal gates.(NAND and NOR).

## NAND Gate:

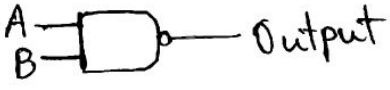
## Equipments:

(i) Proteus Software.
(ii) NAND Gate (7400).
(iii) LOGICPROBE (BIG).
(iv) LOGICSTATE.

## Description:

The NAND gate represents the complement of the AND operation. It's name is an abbreviation of NOT AND. It has the following truth table.

| A | B | Output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



To create other basic gates from NAND:

AND Gate = (A NAND B) NAND (A NAND B)

OR Gate = (A NAND A) NAND (B NAND B)

NOT Gate = A NAND A

X-OR Gate = [A NAND (A NAND B)] NAND
[B NAND (A NAND B)]

NOR Gate = [(A NAND A) NAND (B NAND B)] NAND
[(A NAND A) NAND (B NAND B)]

X-NOR Gate = {[A NAND (A NAND B)] NAND
[B NAND (A NAND B)]} NAND
{[A NAND (A NAND B)] NAND
[B NAND (A NAND B)]}

## NOR Gate:

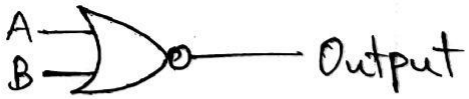
### Equipments:

(i) Proteus Software.
(ii) NOR Gate (NOR)
(iii) LOGICPROBE
(iv) LOGICSTATE

### Description:

The NOR gate represents the complement of the OR operation. Its ~~na~~name is an abbreviation of NOT OR. It has the following truth table:

| A | B | Output |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |




A NOR gate is a universal gate. So, other basic gates can be represented as a combination of NOR gates:

AND = (A NOR A) NOR (B NOR B)

OR = (A NOR B) NOR (A NOR B)

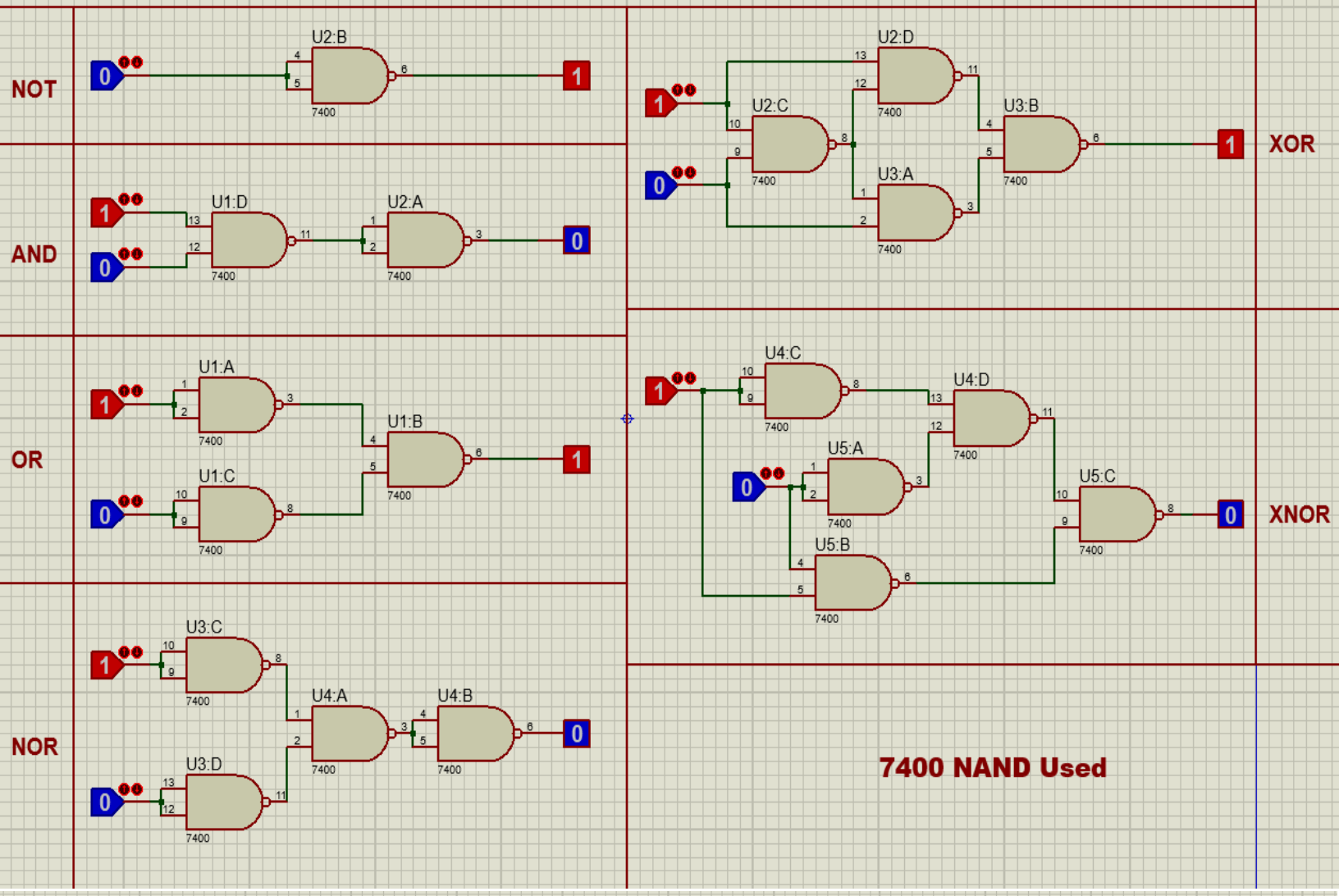
NOT = A NOR A

XOR = (A NOR A) NOR (B NOR B) NOR (A NOR B)

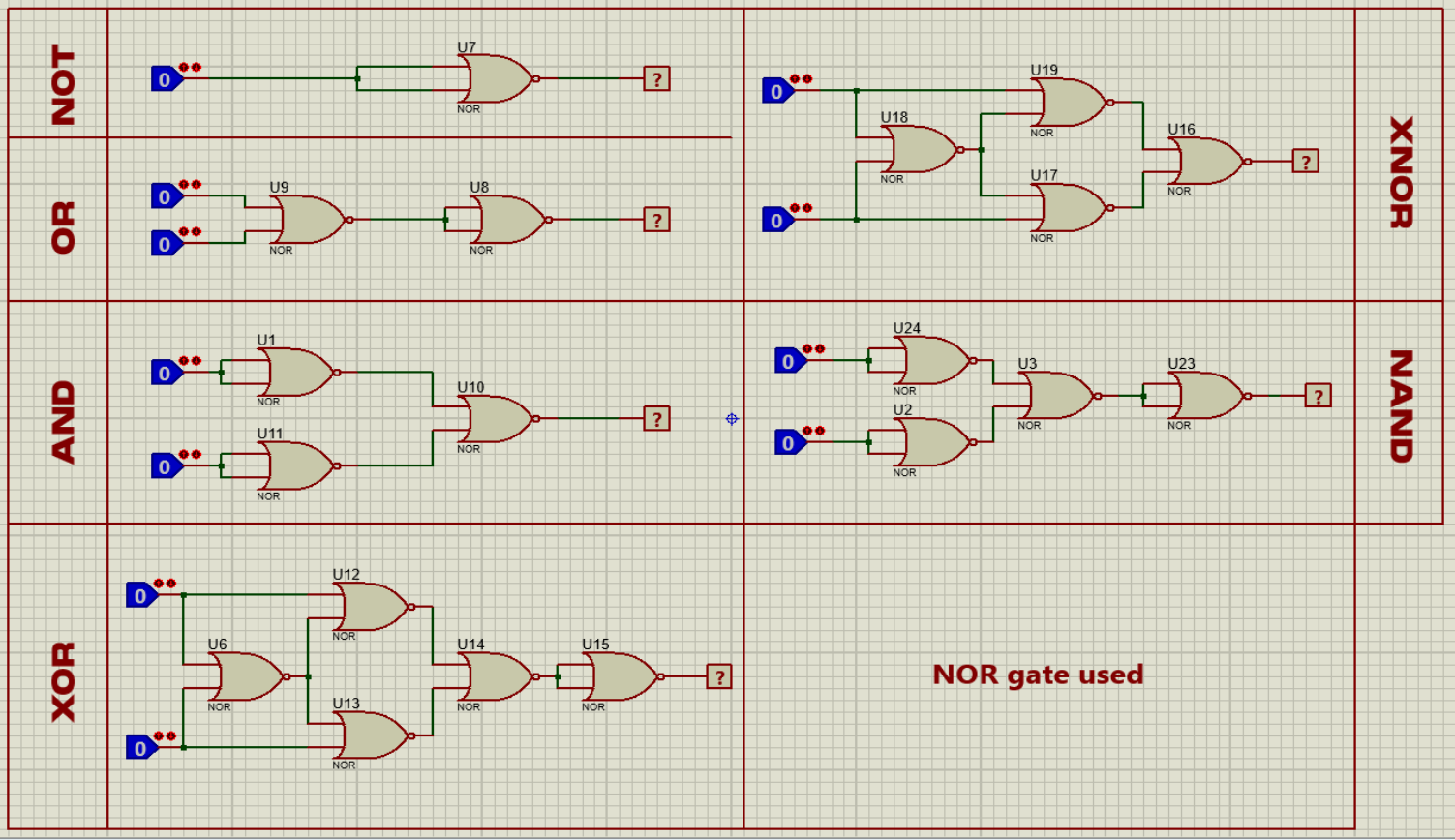
NAND = [(A NOR A) NOR (B NOR B)] NOR [(A NOR A) NOR (B NOR B)]

XNOR = [A NOR (A NOR B)] NOR [B NOR (A NOR B)]

# Universal NAND




# Universal NOR



## Conclusion:

(i) We have learnt how to implement basic gates from universal gates.

(ii) We have learnt what is NAND and what is NOR.

(iii) We understood the NAND and NOR gates to build up basic gates

(iv) We got two types of variant of each gates from NAND and NOR.

(v) We have also learnt how to implement circuits in Proteus Software.

# LAB - 03

Name of the experiment: Implementation of Boolean function.

Equipment:

Without Simplification:

(i) Not Gate
(ii) Logic State
(iii) Logic Probe
(iv) 4 input AND Gate (AND_4)
(v) x input OR Gate (OR_x)

With Simplification:

(i) Not Gate
(ii) Logic State
(iii) Logic Probe
(iv) 2 input AND Gate (AND)
(v) 3 input AND Gate (AND_3)
(vi) 3 input OR Gate (OR_3)

Description: Boolean functions are an important object in logic gate implementation. This function's implementation means logic gates involves connecting output of one logic gate to the input of another gates. Commonly used logic gates are AND, OR, NOT etc. These gates are easy to use and easy for explanation.

Given equation:

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

[This form is without simplification]

Simplification:

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

[Distributive Law]

$\Rightarrow \bar{A}C\bar{D} \cdot 1 + ABC\bar{D} + A\bar{B}C\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D + A\bar{B}CD$

[Complement and Identity Law]

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

[Distributive Law]

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

[Absorption Law]

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

[Distributive Law]

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

[complement and Identity Law]

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

[Distributive Law]

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

[Absorption Law]

$\Rightarrow C\bar{D}B + C\bar{D}\bar{A} + A\bar{B}\bar{C} + A\bar{B}\bar{D} + A\bar{B}CD$ [Distribution]

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

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

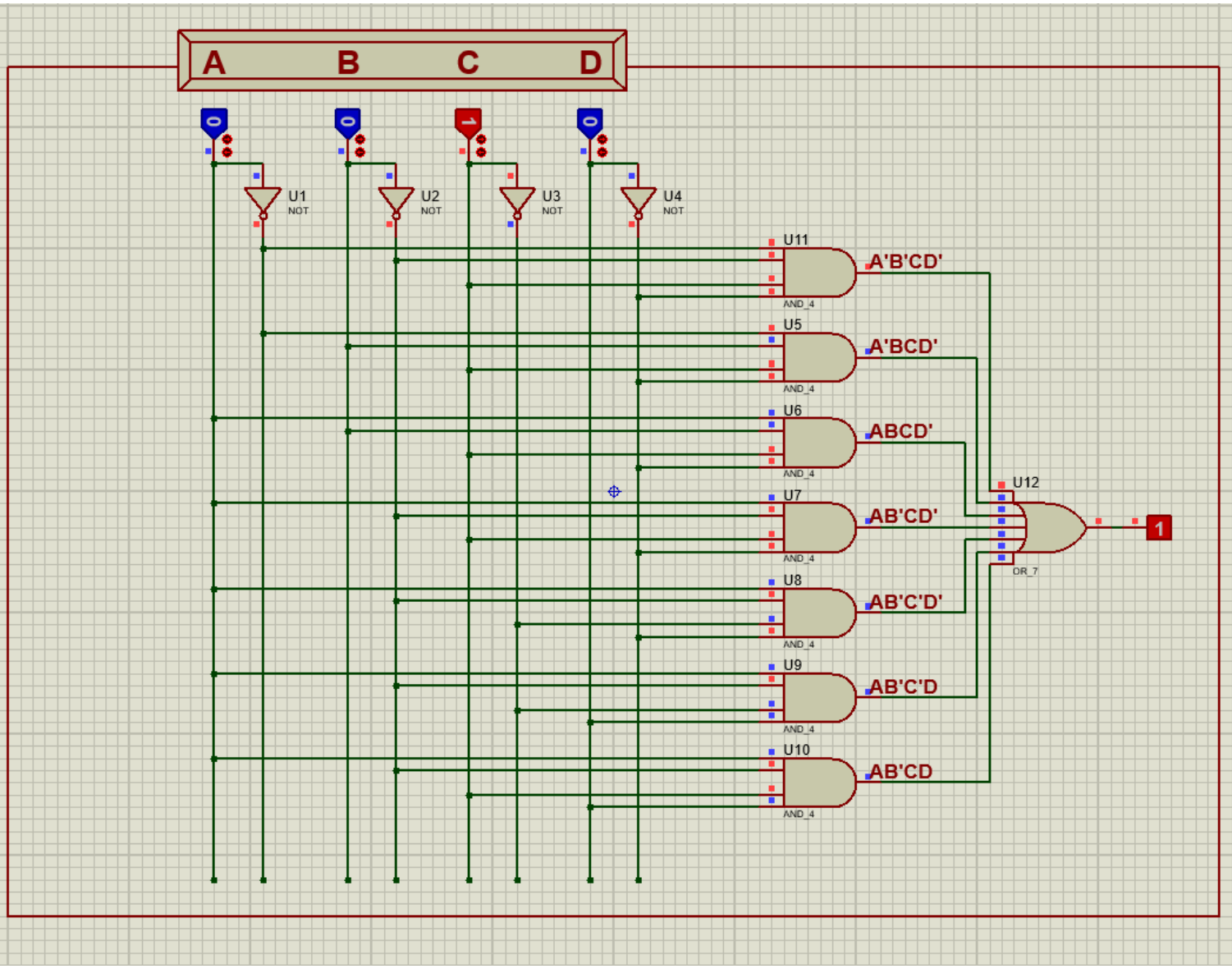
$\Rightarrow C\bar{D}B + C\bar{D}\bar{A} + A\bar{B}\bar{D} + A\bar{B}D + A\bar{B}\bar{C}$ [Distribution]

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

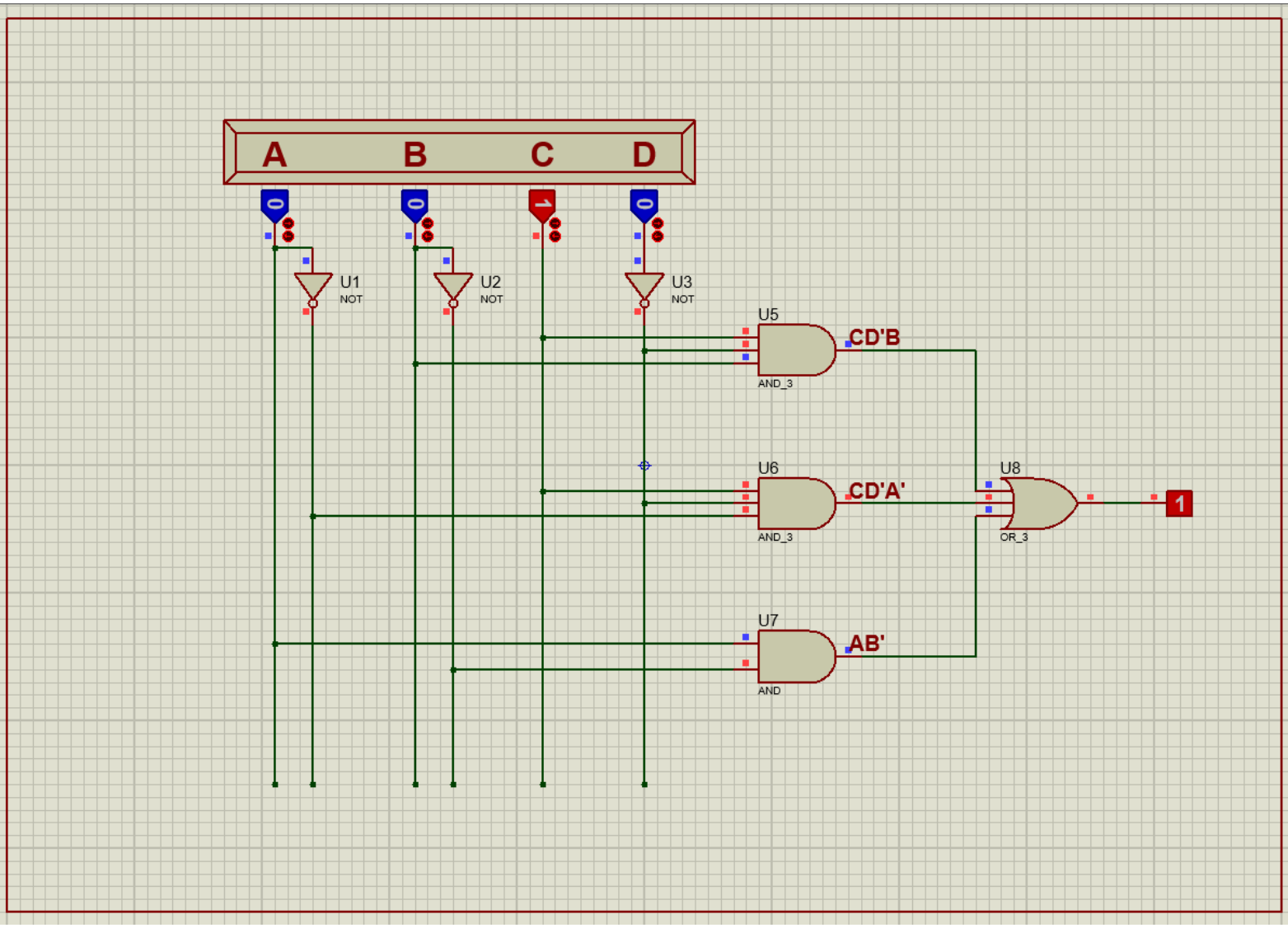
$\Rightarrow C\bar{D}B + C\bar{D}\bar{A} + A\bar{B}$ [Complement and Identity Law]

| A | B | C | D | Output |
|---|---|---|---|---|
| 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 |

Truth Table




**Without Simplification**




**With Simplification**

## Conclution:

(i) We learnt how to implement Boolean function using the basic gates.

(ii) We learnt how to implement Boolean function using truth table.

(iii) We have also learnt how to implement circuits in Proteus software.

# LAB - 04

Name of the experiment: Construct and test various adder and subtractor circuits.

Description:

Adder: An adder is a digital circuit that performs addition of numbers. Adders are two types : (i) Half adder
(ii) Full adder

(i) Half Adder: A half adder is a type of adder, That is able to add two binary digits and provide the output plus a carry value. It has to two ip inputs and two outputs. The common representation uses a XOR logic gate and an AND logic gate.

$\therefore$ Carry $(C) = XY$

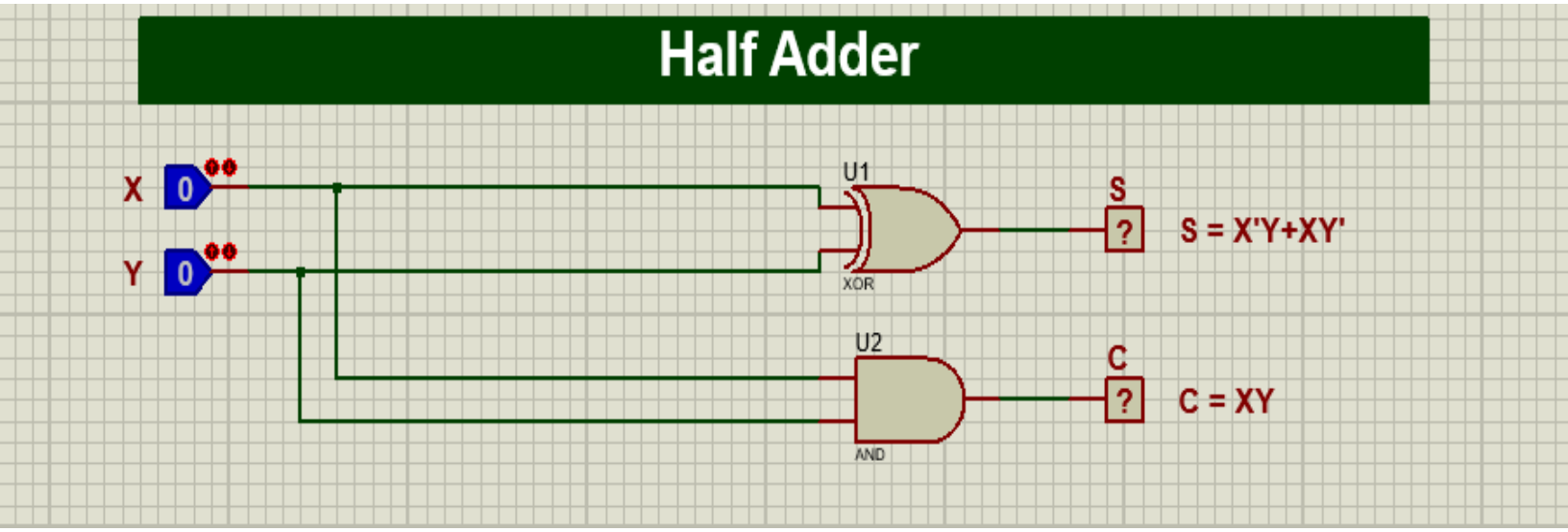
$\therefore$ Sum $(S) = X'Y + XY' = X \oplus Y$

Truth Table:

| X | Y | C | S |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Equipments:

(i) LOGICSTATE
(ii) LOGICPROBE
(iii) XOR Gate
(iv) AND Gate



**Full Adder:** A full adder circuit is central to most digital circuits that perform addition. It is so called because it adds together two binary digits, plus a carry in digit to produce a sum and carry out digit. It has three i/p inputs and two outputs.

$\therefore$ Carry $(C) = XY + XZ + YZ$

$\therefore$ Sum $(S) = X'Y'Z + X'YZ' + XY'Z' + XYZ$

It can also be written with two half adders

Then, Carry, $(C) = XY + Z(X+Y)$

Sum, $(S) = Z \oplus (X \oplus Y)$

Truth Table:

| 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 |

## Equipments:

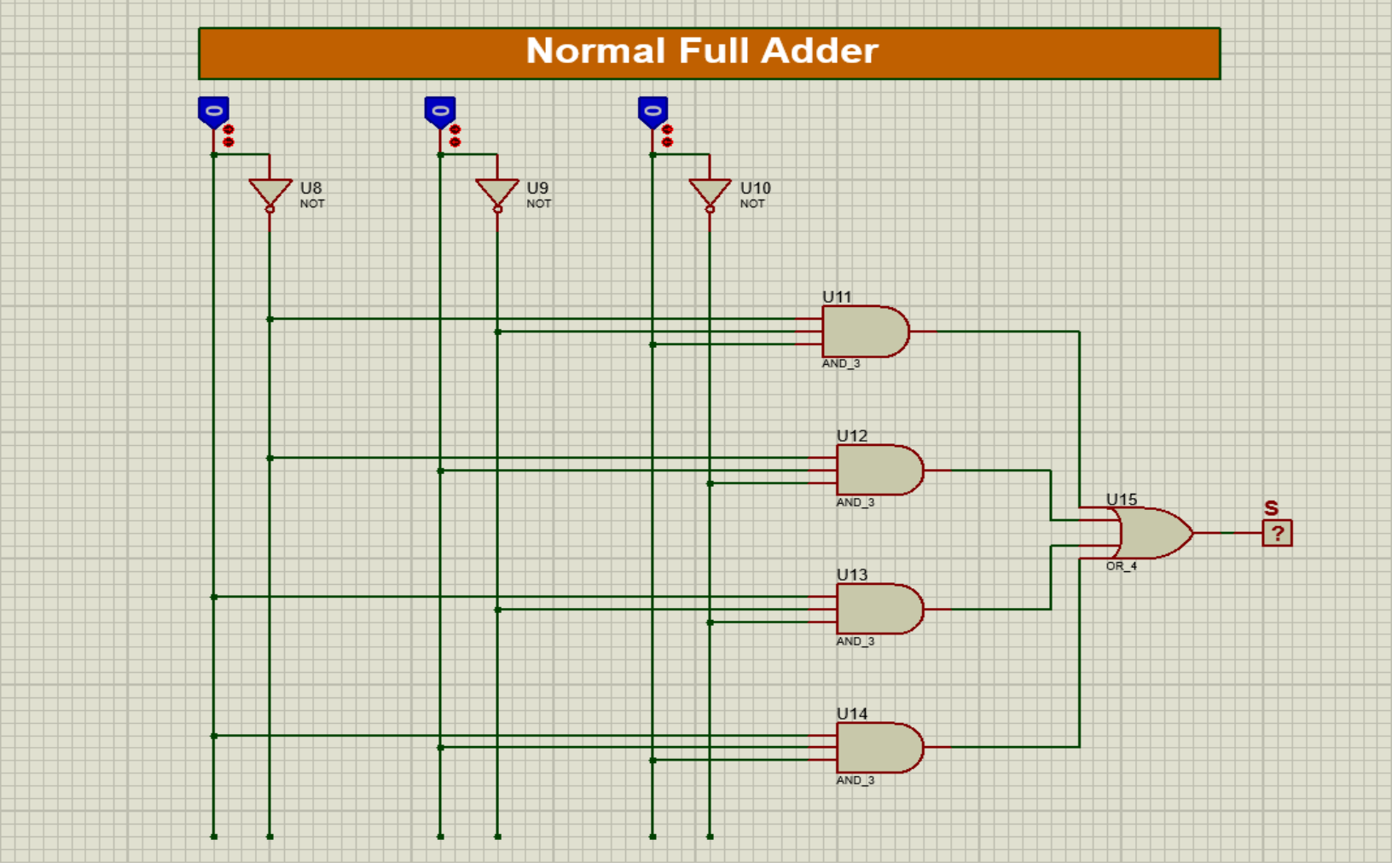
Normal Full Adder:

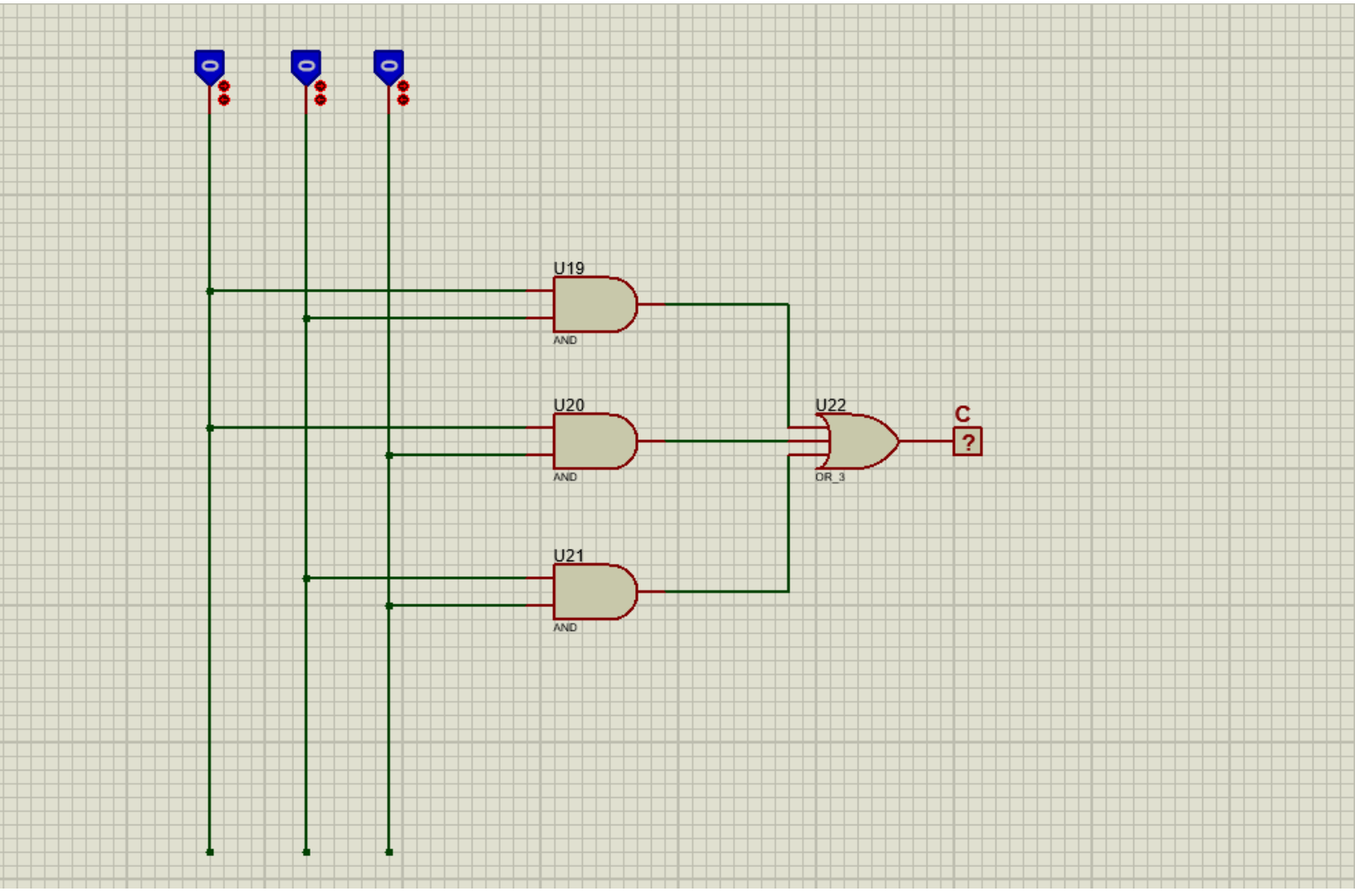
(i) LOGICPROBE
(ii) LOGICSTATE
(iii) 3 input AND Gate
(iv) 4 input OR Gate
(v) 3 input OR Gate
(vi) 2 input AND Gate

Full Adder with Half adders:

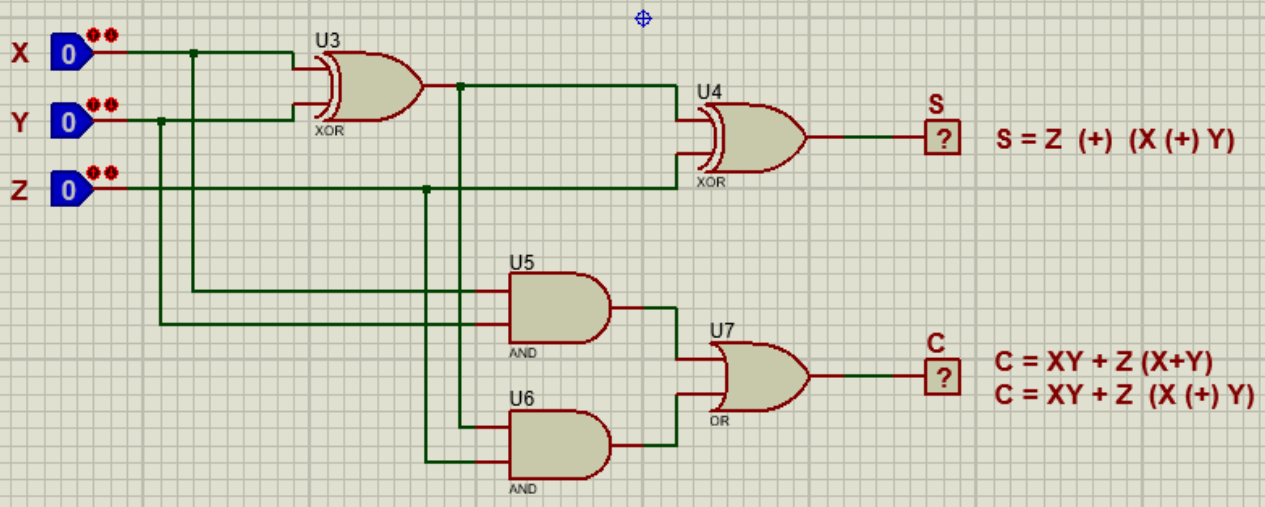
(i) LOGICPROBE
(ii) LOGICSTATE
(iii) XOR Gate
(iv) AND Gate
(v) OR Gate

# Normal Full Adder



# Full adder with two half adders



**Subtractor:** Subtractor circuits take two binary numbers as input and it subtract one number from another. It gives two outputs. (Difference and Borrow).

There are two types of subtractor:

(i) Half Subtractor

(ii) Full Subtractor

(i) **Half Subtractor:** The half subtractor is a combinational circuit which is used to perform subtraction of two bits. It has two inputs and two outputs. It doesn't consider the previous borrow.

$\therefore$ Borrow $(B) = X'Y$

$\therefore$ Difference $(D) = X'Y + XY'$

$= X \oplus Y$

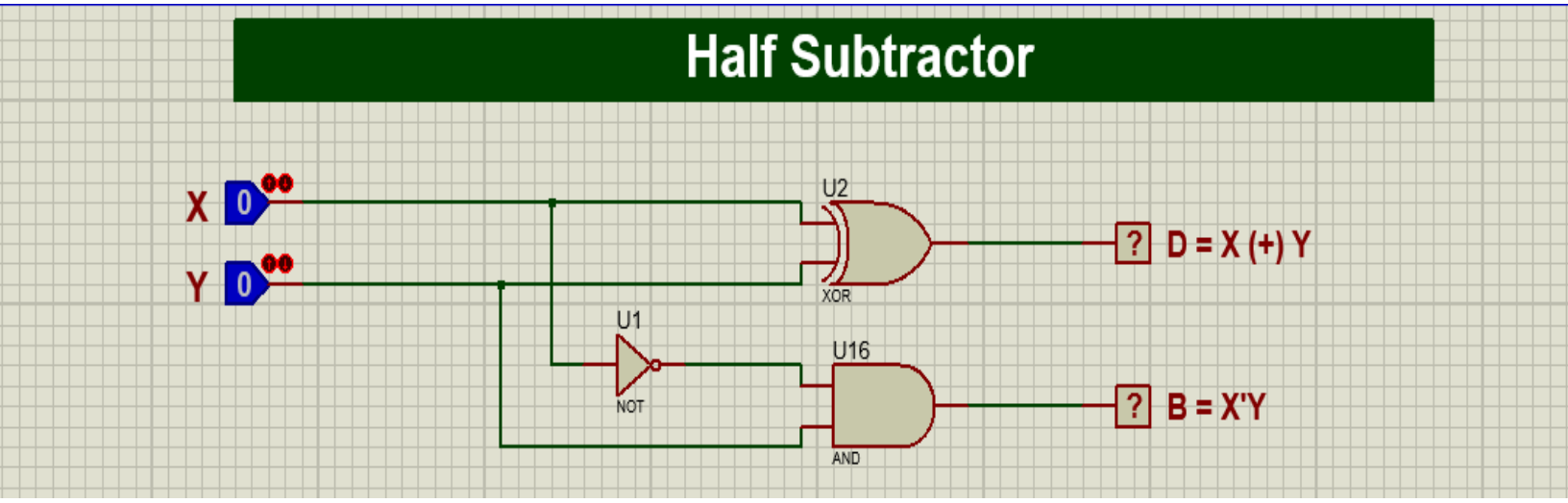
= Adder's Sum

Truth Table:

| X | Y | B | D |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

Equipments:

(i) LOGICSTATE
(ii) LOGICPROBE
(iii) XOR Gate
(iv) NOT Gate
(v) AND Gate.



**Full Subtractor:** A full subtractor performs subtraction involving three bits $(X, Y, Z)$. Here $(X, Y)$ are literal and $Z$ is previous borrow. It has three inputs and two outputs. It consider the previous borrow.

$\therefore$ Borrow, $(B) = X'Z + X'Y + YZ$

$\therefore$ Difference $(D) = X'Y'Z + X'YZ' + XY'Z' + XY'Z + XYZ$
$= $ Sum of full adder.

$\therefore$ It can also be written with two half subtractors.

Then,

Borrow, $(B) = \bar{X}Y + Z(\overline{X \oplus Y})$

Difference, $(D) = Z \oplus (X \oplus Y)$

Truth Table:

| 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 |

Equipments:

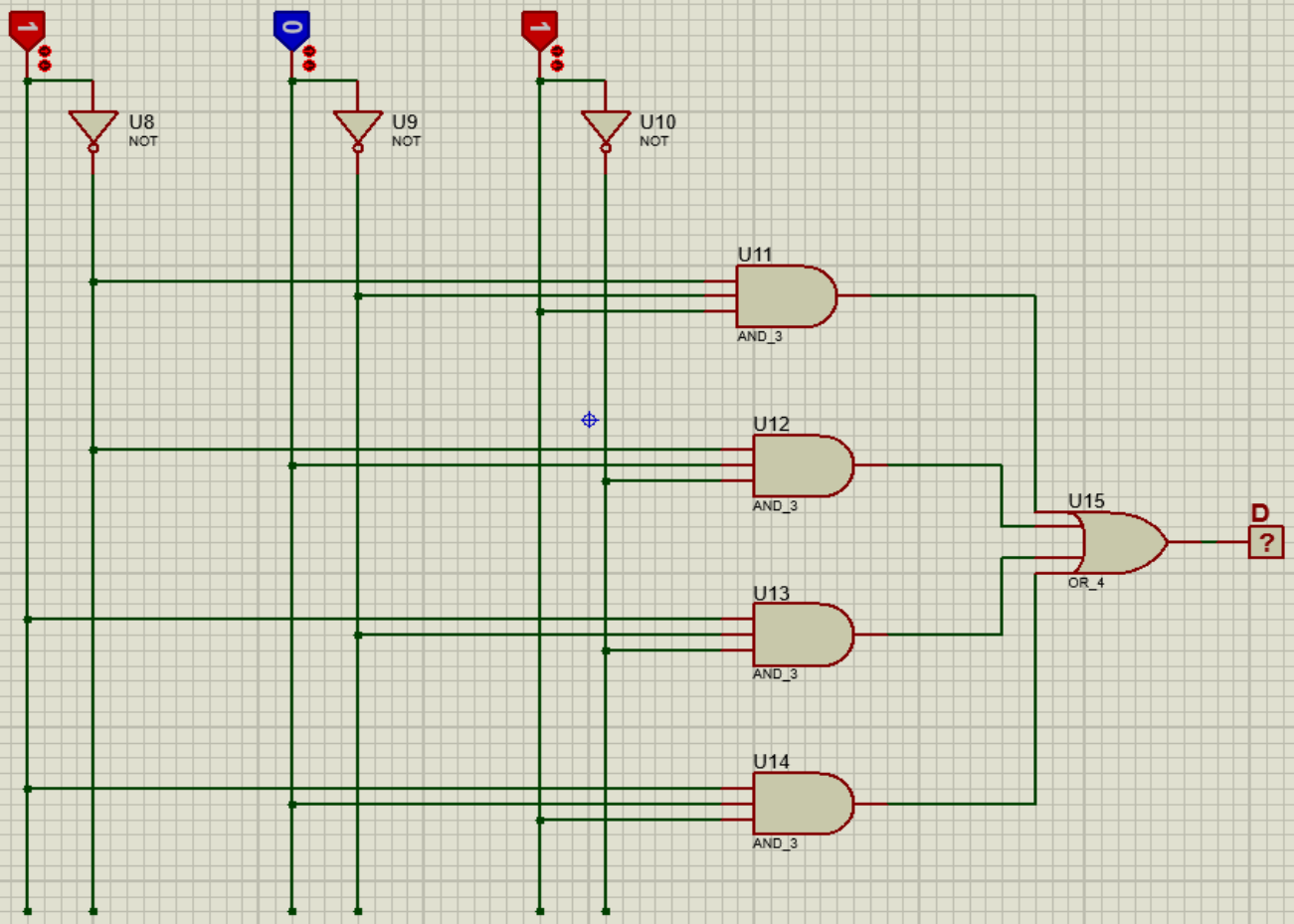
Normal Full Subtractor

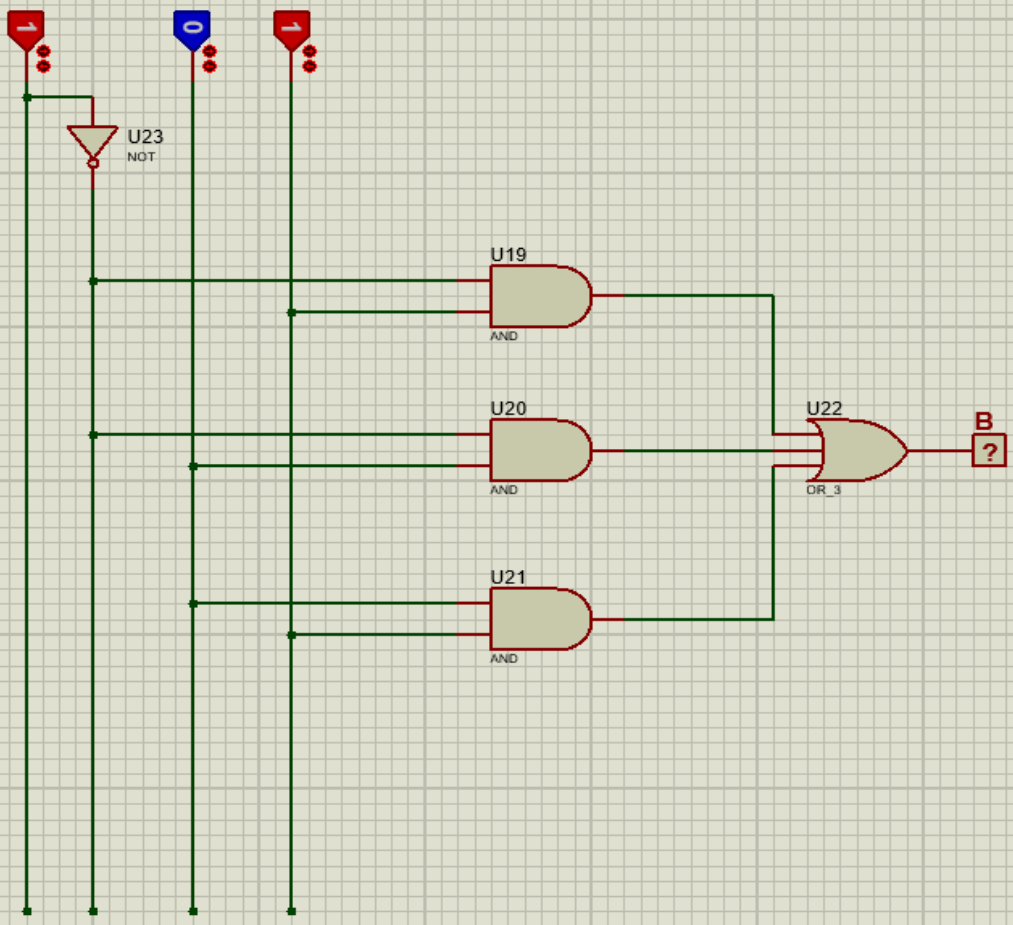
(i) LOGICPROBE
(ii) LOGICSTATE
(iii) AND Gate
(iv) NOT Gate
(v) 3 input OR Gate

Full Subtractor with two half subtractor

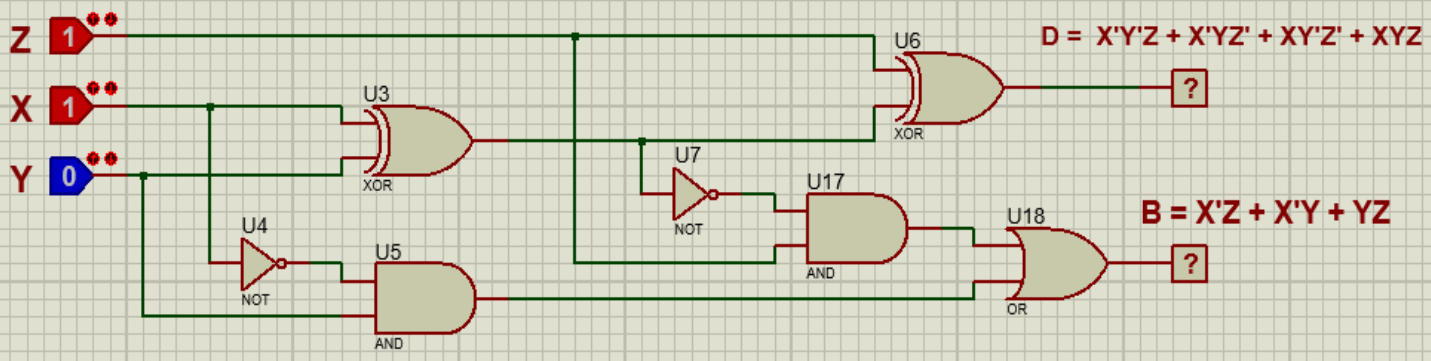
(i) LOGICSTATE
(ii) LOGICPROBE
(iii) XOR Gate
(iv) NOT Gate
(v) AND Gate
(vi) OR Gate

# Normal Full Subtractor



# Full subtractor with two half subtractors



## Conclusion:

(i) We learnt how to implement Adder circuit.

(ii) We learnt how to implement Subtractor circuit.

(iii) We learnt how to add two binary bits and find out carry and sum.

(iv) We learnt how to subtract two or three bits and find out Borrow and the Difference.

(v) We learnt implementation of full adder using half adders.

(vi) We learnt implementation of full subtractor using half subtractors.

# LAB - 05

Name of the experiment: Design, construct and test combination logic circuits like parity generator.

Explanation:

A parity bit, or check bit, is a bit added to a string of binary code. Parity bits are a simple form of error detecting code. Parity bits are generally applied to the smallest units of a communication protocol, typically 8-bit octets, although they can also be applied sparcately to an entire message string of bits. The parity bit ensures that the total number of 1-bits in the string is even or odd. There are two variants of parity bits: even parity bit, odd parity bit.

Even parity: Here all the total number of bits in the message is made even.

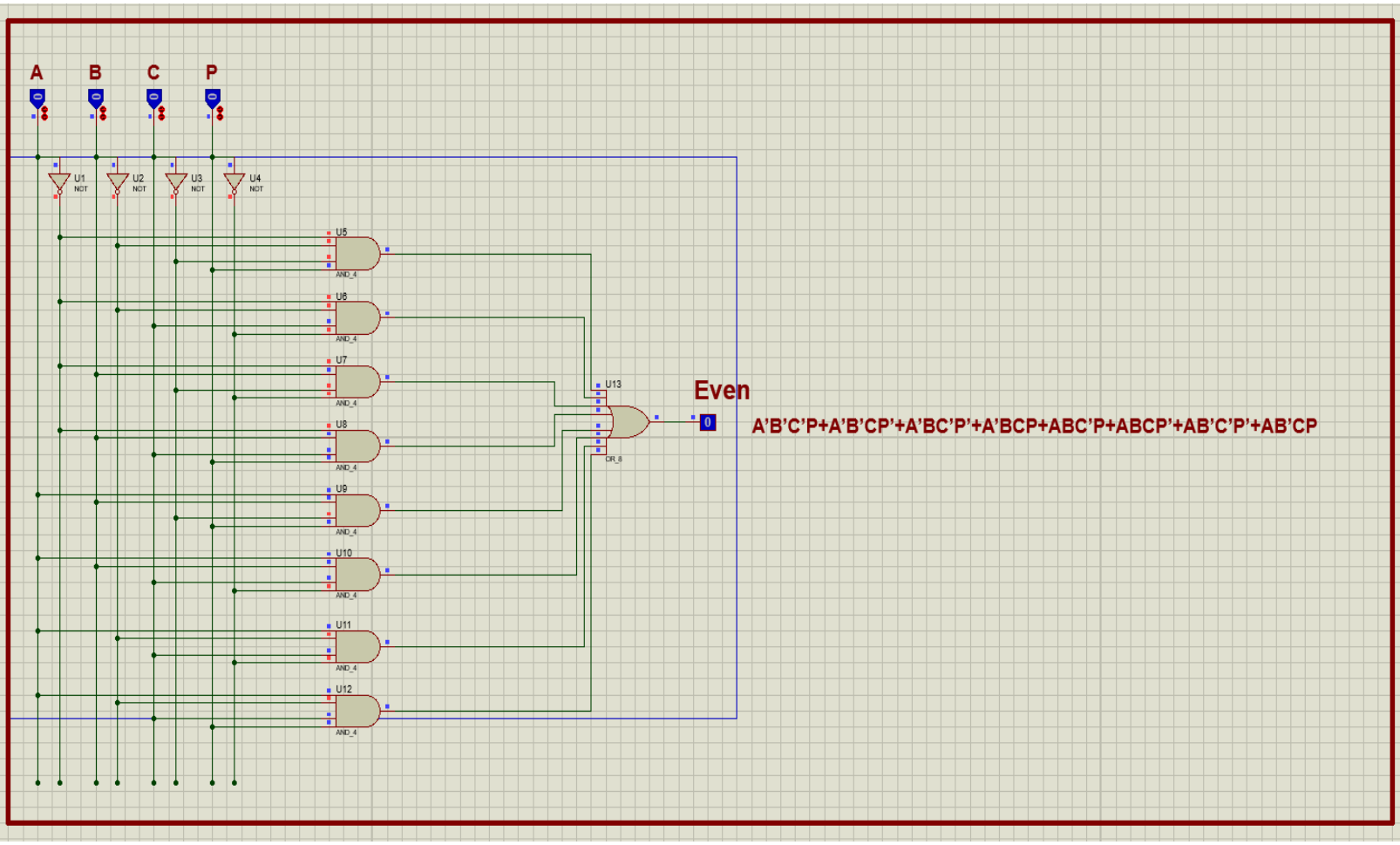
Truth table:

| A | B | C | P | Result |
|---|---|---|---|---|
| 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 |

K-Map:

| AB \ CP | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | (1) | 0 | (1) |
| 01 | (1) | 0 | (1) | 0 |
| 11 | 0 | (1) | 0 | (1) |
| 10 | (1) | 0 | (1) | 0 |

$F = A'B'C'P + A'B'CP' + A'BC'P' + A'BCP + ABC'P + ABCP' + AB'C'P' + AB'CP$


A
B
C
P
U1
NOT
U2
NOT
U3
NOT
U4
NOT
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
Even
0
A'B'C'P+A'B'CP'+A'BC'P'+A'BCP+ABC'P+ABCP'+AB'C'P'+AB'CP

Odd parity: Here the total number of bits in the message is made odd.

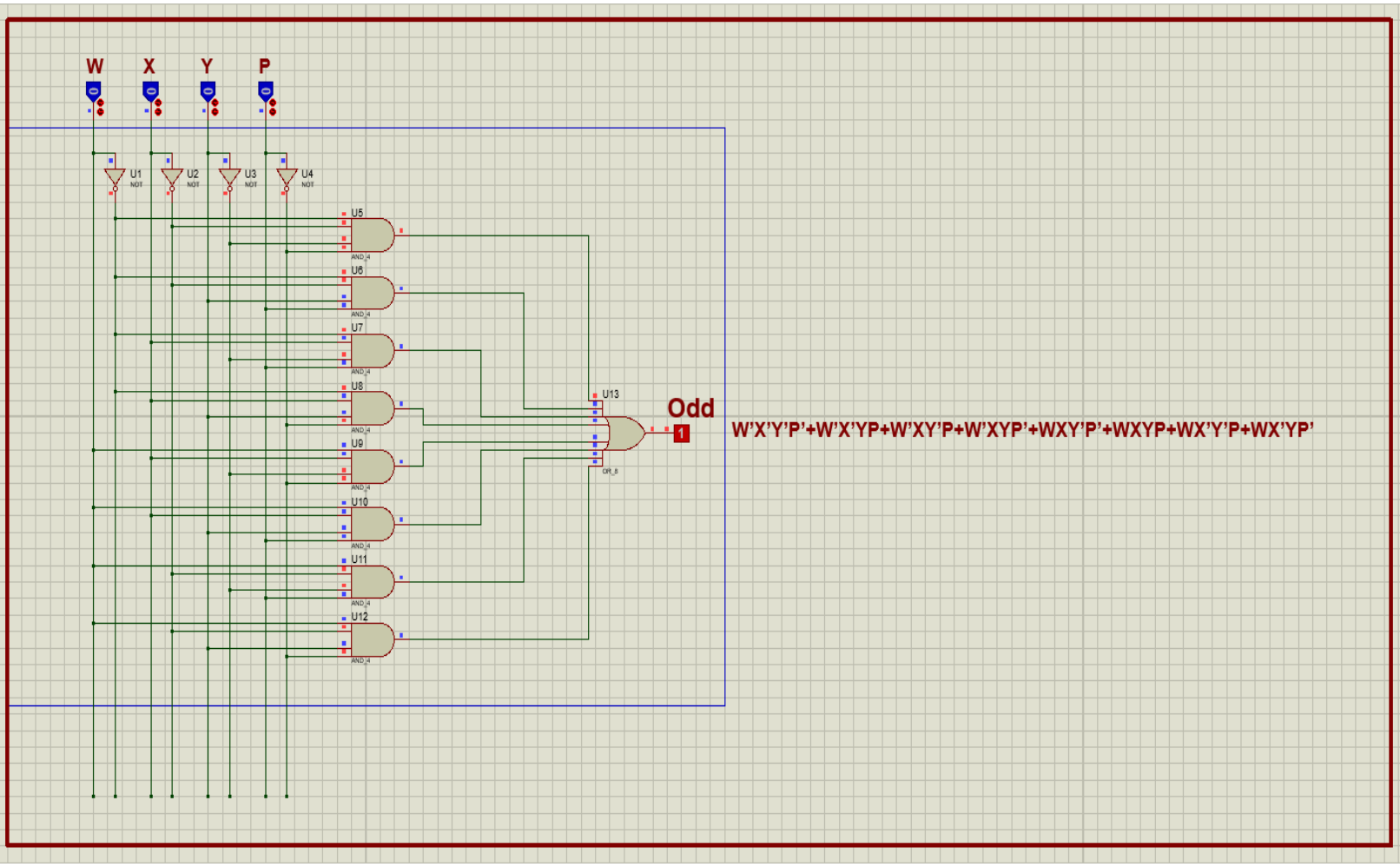
Truth table:

| W | X | Y | P | Result |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

K-Map:

| WX \ YP | 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
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
Odd
1
W'X'Y'P'+W'X'YP+W'XY'P+W'XYP'+WXY'P'+WXYP+WX'Y'P+WX'YP'

Name of the experiment: Design of code converters like BCD to Excess-3

Description: Excess-3 binary code is an unweighted self-complementary BCD code. Self-complementary property means that the 1's complement of an excess-3 number is the excess-3 code of the 9's complement of the corresponding decimal number. This property is used since a decimal number can be nines complement as easily as a binary number can be one's complemented; just by inverting all bits.

The process of converting BCD to excess-3 is quite simple from othe conversions. The Excess-3 code can be calculated by adding 3 to each four digit BCD code.

Truth table:

| Decimal | BCD | | | | Excess-3 | | | |
|---|---|---|---|---|---|---|---|---|
| | W | X | Y | P | A | B | C | D |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 5 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 7 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| 8 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 10 | 1 | 0 | 1 | 0 | X | X | X | X |
| 11 | 1 | 0 | 1 | 1 | X | X | X | X |
| 12 | 1 | 1 | 0 | 0 | X | X | X | X |
| 13 | 1 | 1 | 0 | 1 | X | X | X | X |
| 14 | 1 | 1 | 1 | 0 | X | X | X | X |
| 15 | 1 | 1 | 1 | 1 | X | X | X | X |

Now, we will use the K-map method to design the logical circuit for the conversion of BCD to Excess-3 code.

| WX \ YP | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 1 | 1 | 1 |
| 11 | X | X | X | X |
| 10 | 1 | 1 | X | X |

$A = W + XP + XY$

| WX \ YP | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 1 | 1 |
| 01 | 1 | 0 | 0 | 0 |
| 11 | X | X | X | X |
| 10 | 0 | 1 | X | X |

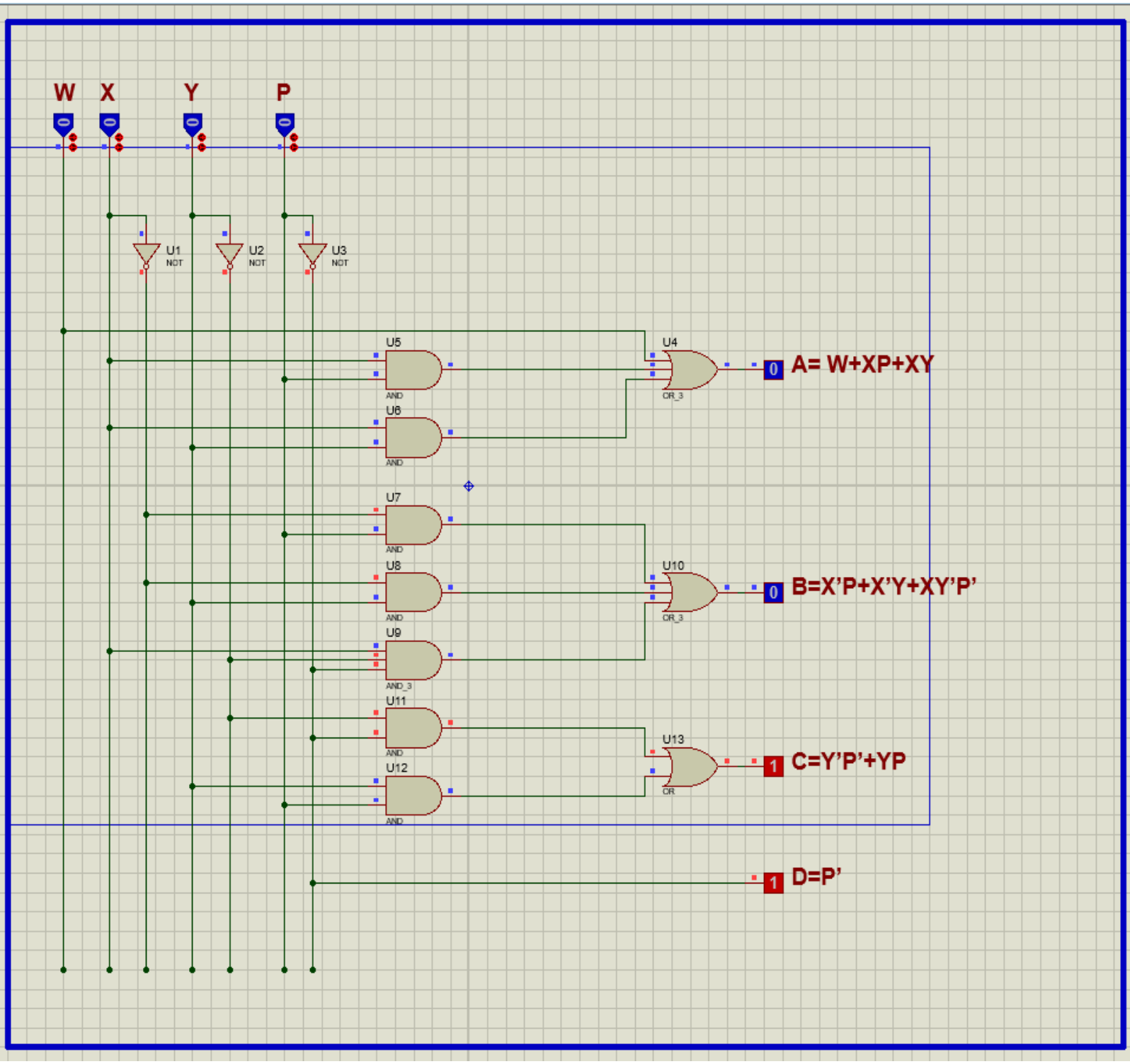
$B = X'P + X'Y + XY'P'$

| WX \ YP | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 0 | 1 | 0 |
| 01 | 1 | 0 | 1 | 0 |
| 11 | X | X | X | X |
| 10 | 1 | 0 | X | X |

$C = Y'P' + YP$

| | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 0 | 0 | 1 |
| 01 | 1 | 0 | 0 | 1 |
| 11 | X | X | X | X |
| 10 | 1 | 0 | X | X |

$D = P'$


W
X
Y
P
U1
NOT
U2
NOT
U3
NOT
U5
AND
U6
AND
U4
OR_3
0
A= W+XP+XY
U7
AND
U8
AND
U9
AND_3
U10
OR_3
0
B=X'P+X'Y+XY'P'
U11
AND
U12
AND
U13
OR
1
C=Y'P'+YP
1
D=P'

## Conclusion:

(i) We have learnt about parity bits.

(ii) We have learnt how to check error of a message using even or odd parity.

(iii) We learnt how to design a parity bit checker circuit.

(iv) We have also learnt how to convert BCD code to Excess-3 code.

(v) We have also learnt how to implement code converters using basic gates.

# LAB - 06

Name of the experiment: Check the operation of 2 to 4 line decoder and 3 to 8 line decoder.

Description:

A decoder is a circuit that changes a code into a set of signals. A common type of decoder is the line decoder which takes an n-digit binary number and decodes it into $2^n$ data lines.

2 to 4 line decoder: This 2 to 4 line decoder includes two inputs $(x, y)$ and four outputs $(D_0, D_1, D_2, D_3)$.

Truth Table:

| X | Y | $D_0$ | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

$D_0 = X'Y'$ $D_2 = XY'$

$D_1 = X'Y$ $D_3 = XY$

3 to 8 line decoder: In this decoder there are three inputs (X, Y, Z) and eight outputs ($D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7$)

Truth Table:

| 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 |

$D_0 = X'Y'Z'$

$D_1 = X'Y'Z$

$D_2 = X'YZ'$

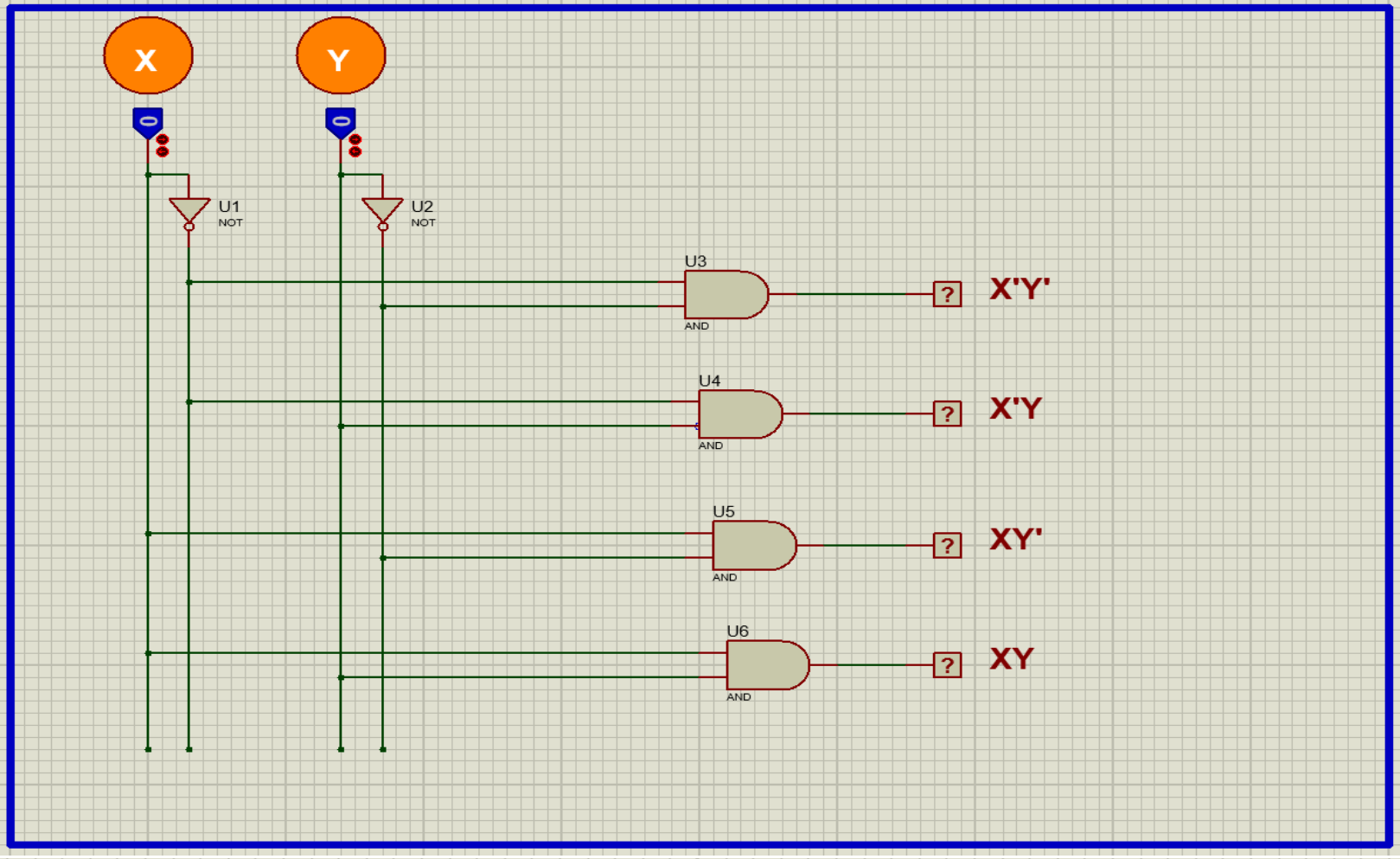
$D_3 = X'YZ$

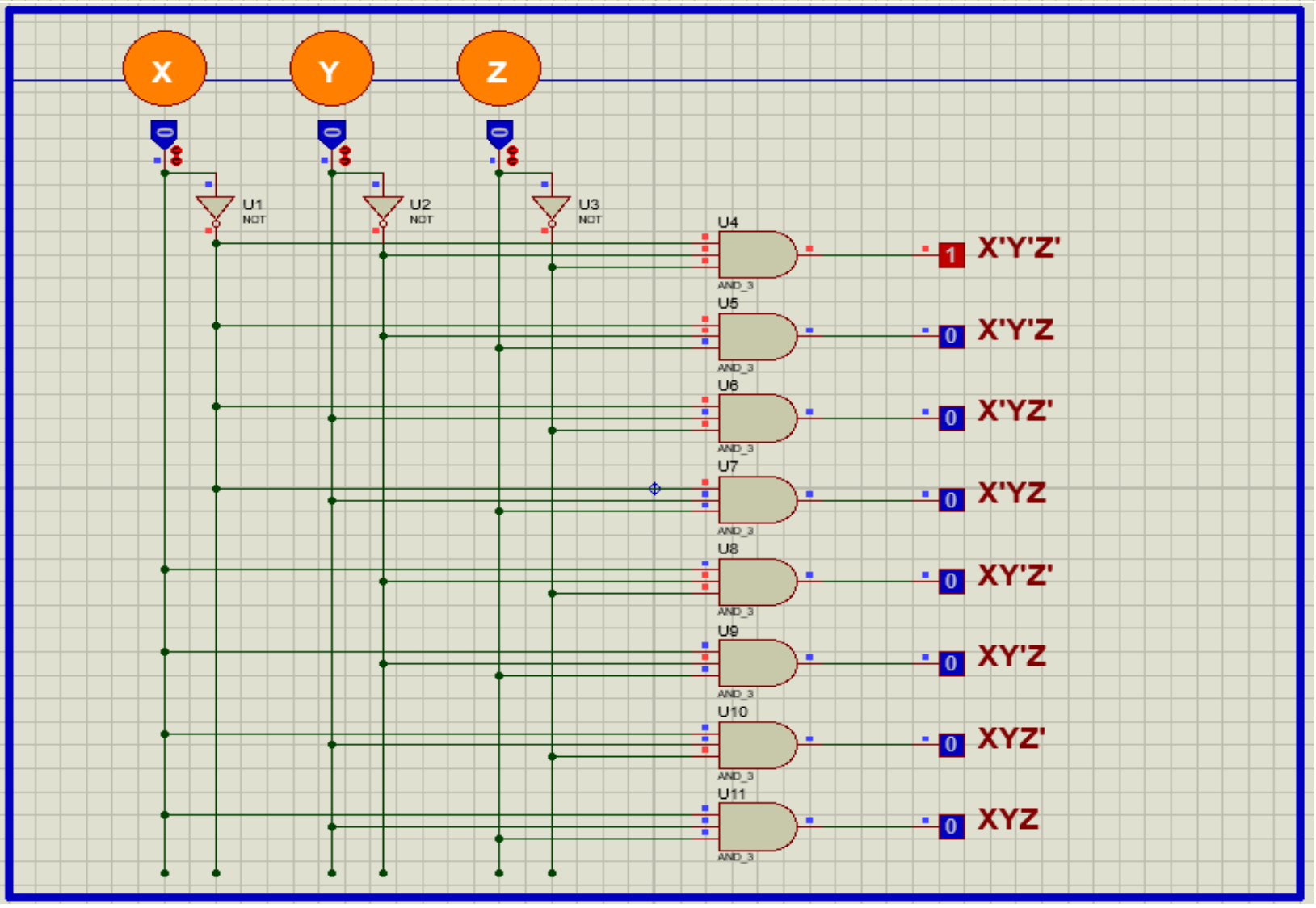
$D_4 = XY'Z'$

$D_5 = XY'Z$

$D_6 = XYZ'$

$D_7 = XYZ$


X
Y
U1
NOT
U2
NOT
U3
AND
?
X'Y'
U4
AND
?
X'Y
U5
AND
?
XY'
U6
AND
?
XY


X
Y
Z
U1
NOT
U2
NOT
U3
NOT
U4
AND_3
1
X'Y'Z'
U5
AND_3
0
X'Y'Z
U6
AND_3
0
X'YZ'
U7
AND_3
0
X'YZ
U8
AND_3
0
XY'Z'
U9
AND_3
0
XY'Z
U10
AND_3
0
XYZ'
U11
AND_3
0
XYZ

## Conclusion:

(i) We have learnt what is 2 to 4 line decoder and 3 to 8 line decoder.

(ii) We have learnt how to implement 2 to 4 line and 3 to 8 line decoder, via logic gates.

(iii) We have learnt how to find out the relation between input and output of a decoder.

# LAB - 07

Name of the experiment: To check the operation of active low demux, octal to binary encoder, Decimal to BCD encoder and Hexadecimal to binary encoder.

## Demux:

Description: The demux(demultiplexer) is a combinational ei logic circuit designed to switch one common input line to one of several seperate output line. The demux is also known as two to four demultiplexer. (two inputs, four outputs)

Truth Table:

| E | X | Y | $D_0$ | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|---|---|---|
| 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 |

$$\therefore D_0 = E + X + Y$$

$$= \overline{\overline{E + X + Y}}$$

$$= \overline{E' . X' . Y'}$$

$$\therefore D_1 = E + X + Y'$$

$$= \overline{\overline{E + X + Y'}}$$

$$= \overline{E' . X' . Y}$$

$$\therefore D_2 = E + X' + Y$$

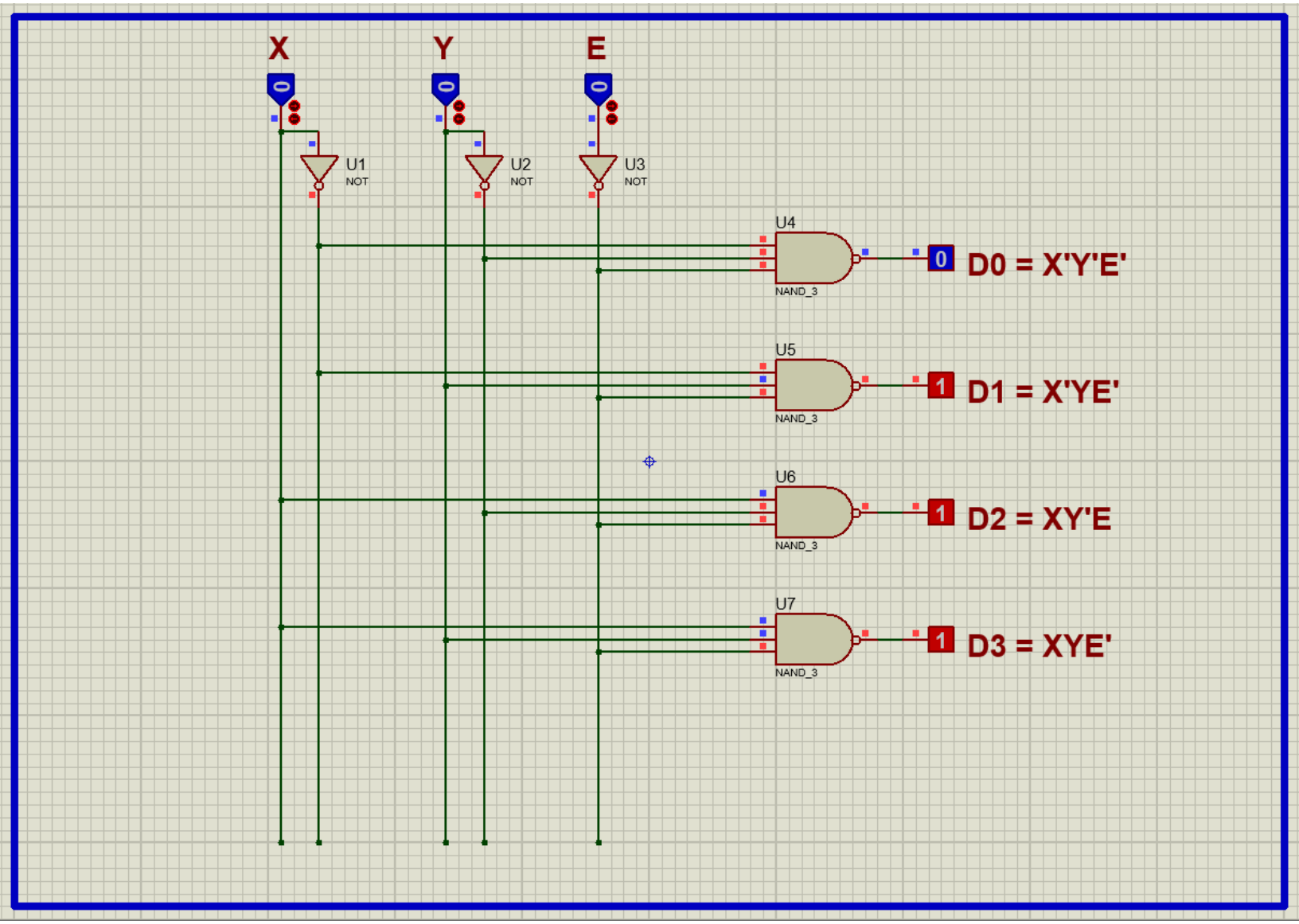
$$= \overline{\overline{E + X' + Y}}$$

$$= \overline{E' . X . Y'}$$

$$\therefore D_3 = E + X' + Y'$$

$$= \overline{\overline{E + X' + Y'}}$$

$$= \overline{E' . X . Y}$$


X
Y
E
U1
NOT
U2
NOT
U3
NOT
U4
NAND_3
0
D0 = X'Y'E'
U5
NAND_3
1
D1 = X'YE'
U6
NAND_3
1
D2 = XY'E
U7
NAND_3
1
D3 = XYE'

## Octal to Binary encoder:

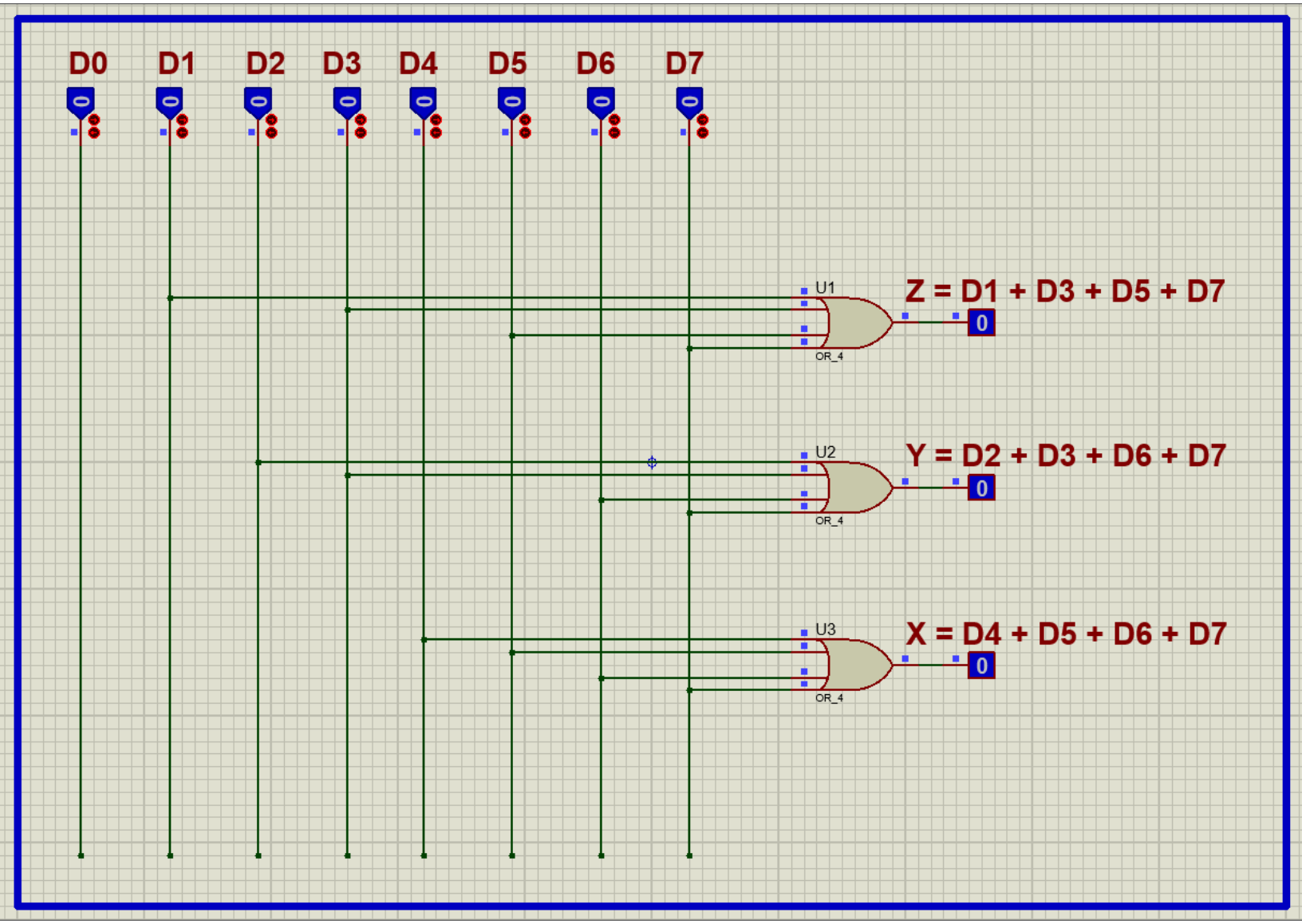
Octal to Binary encoder consists of eight inputs, one for each of eight digits and three outputs which generate the consequent binary numbers.

| $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ | 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 |

$$X = D_4 + D_5 + D_6 + D_7$$

$$Y = D_2 + D_3 + D_6 + D_7$$

$$Z = D_1 + D_3 + D_5 + D_7$$


D0
D1
D2
D3
D4
D5
D6
D7
U1
OR_4
Z = D1 + D3 + D5 + D7
0
U2
OR_4
Y = D2 + D3 + D6 + D7
0
U3
OR_4
X = D4 + D5 + D6 + D7
0

## Decimal to BCD encoder:

Decimal to BCD encoder consists of ten input and four output lines. Each input line corresponds to the each decimal digit and four output lines correspond to the BCD code.

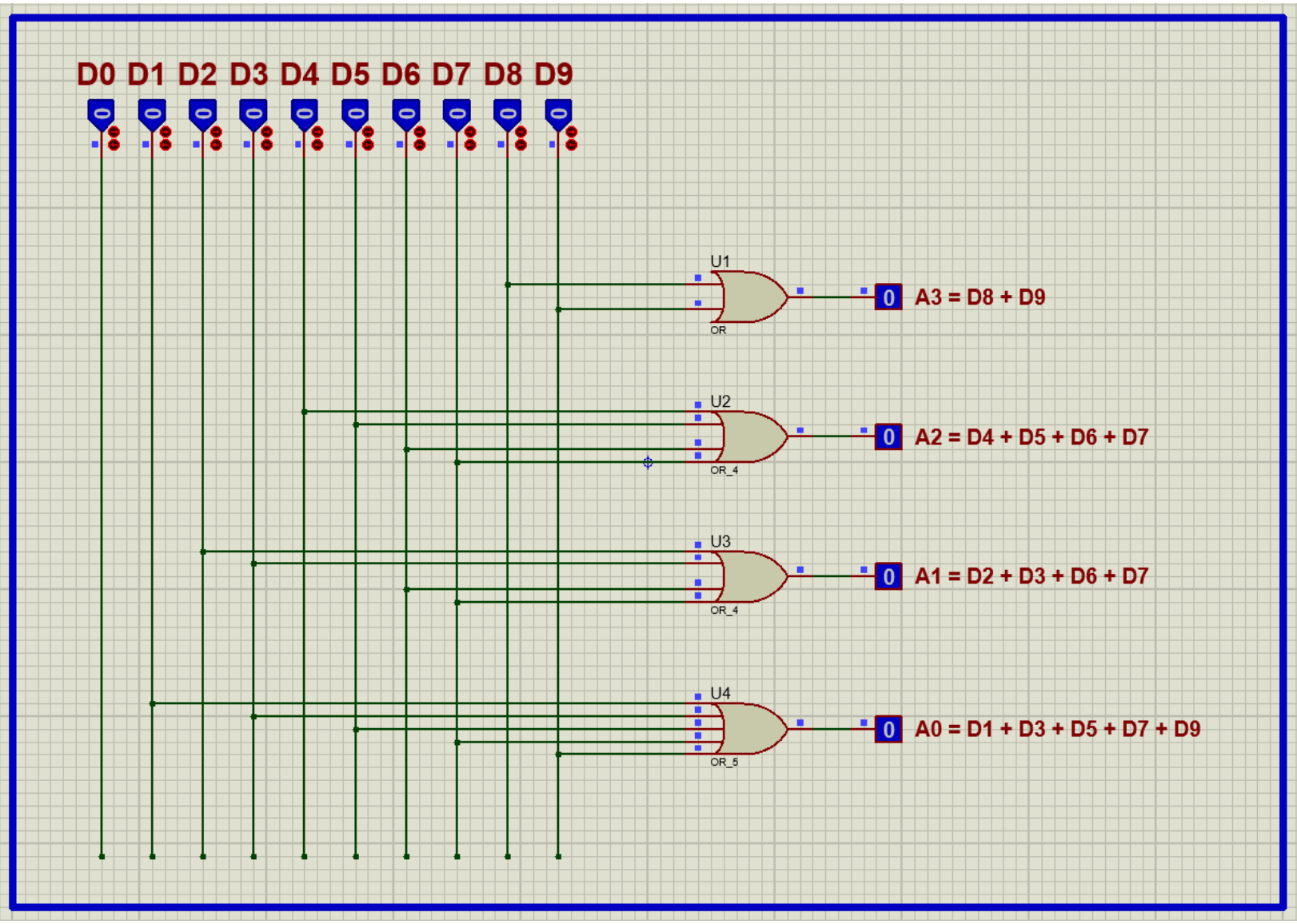
| $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ | $D_8$ | $D_9$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

$$A_3 = D_8 + D_9$$

$$A_2 = D_4 + D_5 + D_6 + D_7$$

$$A_1 = D_2 + D_3 + D_6 + D_7$$

$$A_0 = D_1 + D_3 + D_5 + D_7 + D_9$$


D0 D1 D2 D3 D4 D5 D6 D7 D8 D9
U1
OR
0
A3 = D8 + D9
U2
OR_4
0
A2 = D4 + D5 + D6 + D7
U3
OR_4
0
A1 = D2 + D3 + D6 + D7
U4
OR_5
0
A0 = D1 + D3 + D5 + D7 + D9

## Hexadecimal to Binary encoder:

Hexadecimal is used for larger calculation. But in a computer those hexadecimal numbers need to be converted in binary so that a computer can easily understand the comands. This type of encoders usually consists of 16 line input and 4 line output.

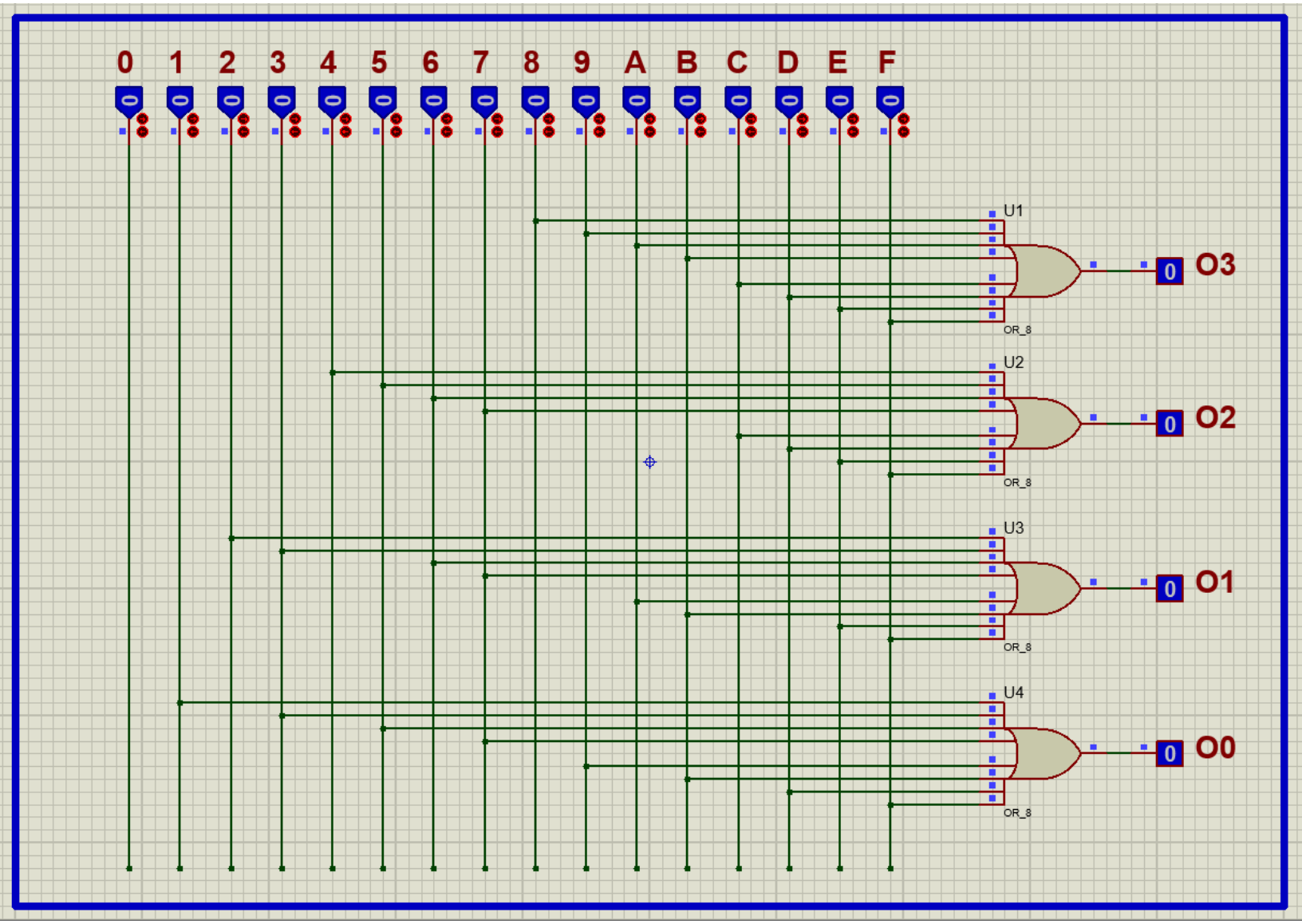
| Inputs | | | | | | | | | | | | | | | | Outputs | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| F | E | D | C | B | A | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | $O_3$ | $O_2$ | $O_1$ | $O_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

$O_3 = B+9+A+B+C+D+E+F$

$O_2 = 4+5+6+7+C+D+E+F$

$O_1 = 2+3+6+7+A+B+E+F$

$O_0 = 1+3+5+7+9+B+D+F$


0 1 2 3 4 5 6 7 8 9 A B C D E F
U1
OR_8
0
O3
U2
OR_8
0
O2
U3
OR_8
0
O1
U4
OR_8
0
O0

## Conclusion:

(i) We have learnt how to design an Octal to Binary encoder.

(ii) We have learnt how to design a Hexadecimal to Binary encoder.

(iii) We have learnt how to design a Decimal to BCD encoder.

(iv) We have learnt how to implement demultiplexer.

(v) We have also learnt many encoding techniques with their circuit diagram.

# LAB - 08

Name of the experiment: Designing 2-to-1, 4-to-1 and 2-to-1 quadruple Multiplexer (Mux). Also implement 1-to-4 line de-multiplexer.

Description:

Multiplexer: Multiplexer means many into one. A multiplexer is a circuit used to select and route any one of the several input signals to a single output. Multiplexers can handle two types of data. For digital application, they are built from standard logic gates.

Truth Table:

2-to-1 mux:

| $S_0$ | Output |
|---|---|
| 0 | $I_0$ |
| 1 | $I_1$ |

4-to-1 mux:

| $S_0$ | $S_1$ | Output |
|---|---|---|
| 0 | 0 | $I_0$ |
| 0 | 1 | $I_1$ |
| 1 | 0 | $I_2$ |
| 1 | 1 | $I_3$ |

2-to-1 quadruple mux:

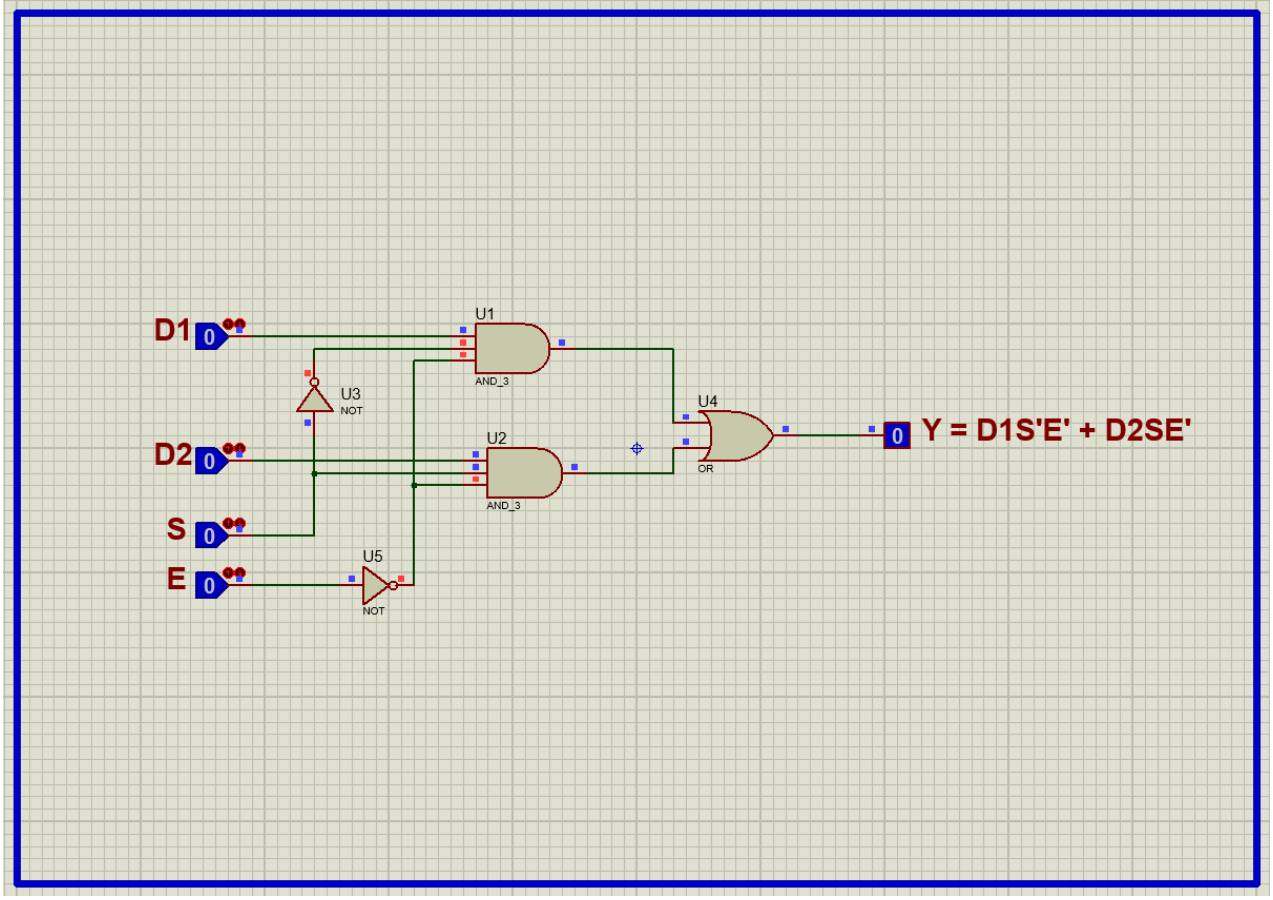
| S | Output |
|---|---|
| 0 | A |
| 1 | B |

## Demultiplexer:

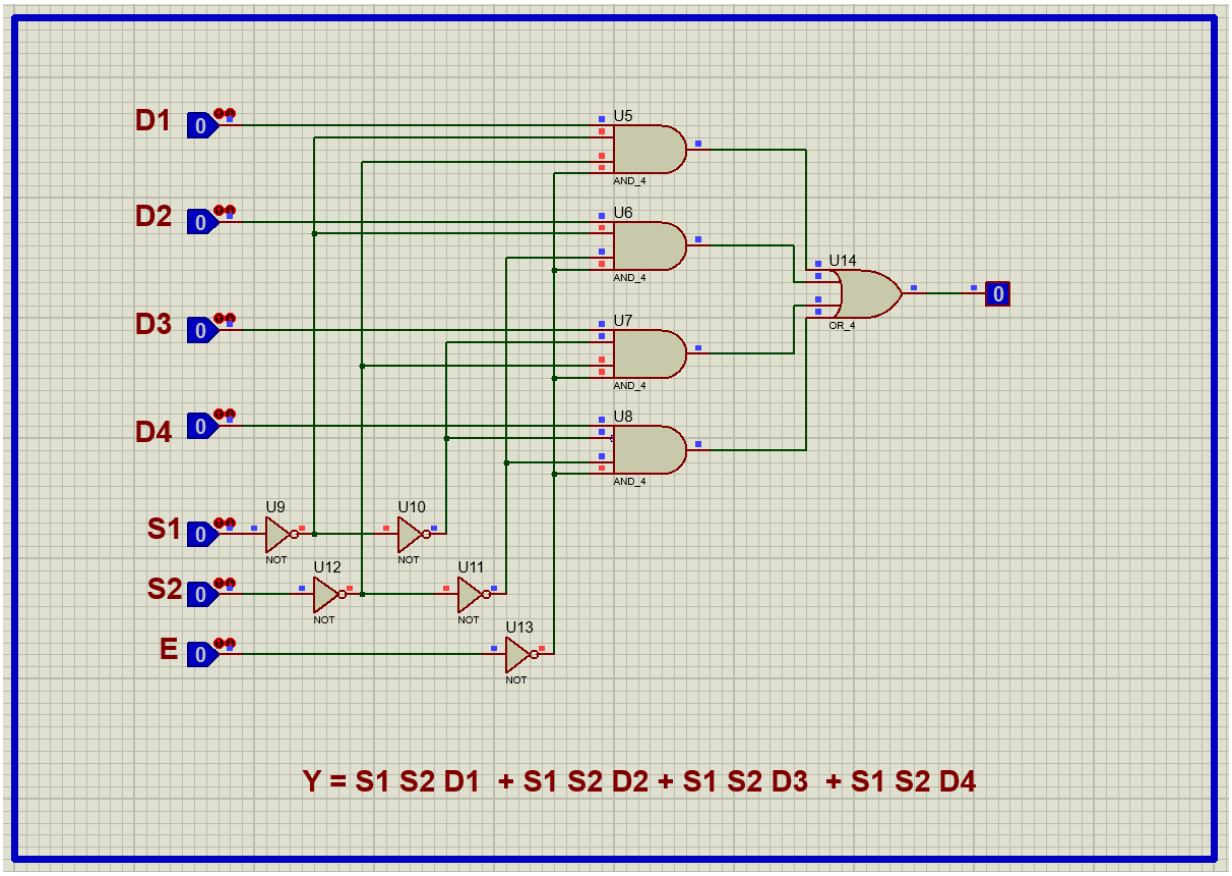
Demultiplexer means one to many. A demultiplexer is a circuit with one input and many outputs. By applying control signals, we can steer any input to the output. Input = n then output will be $2^n$.

Truth Table:

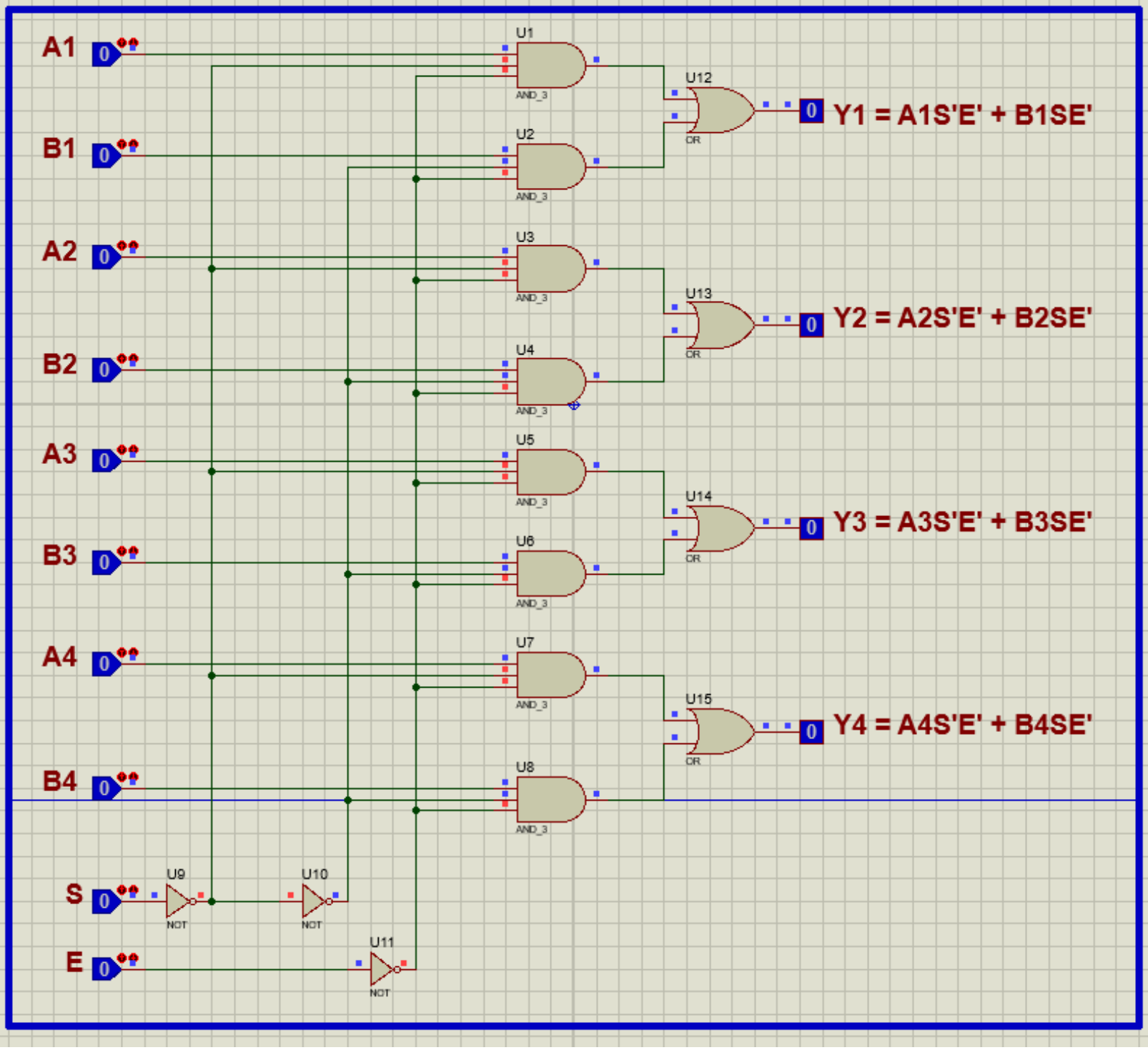
| $S_1$ | $S_0$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |



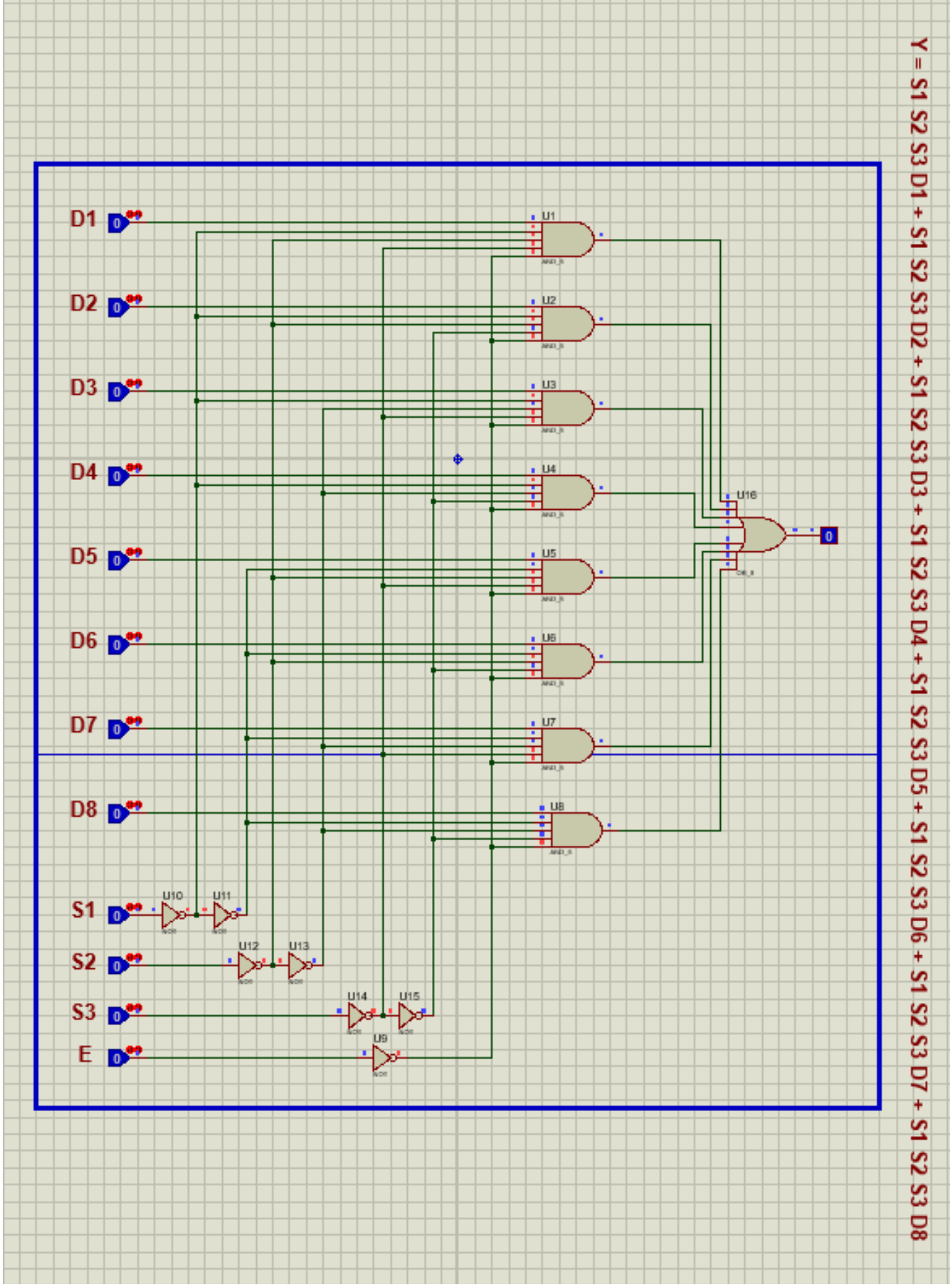

## 2 to 1 Line MUX




## 4 to 1 Line MUX




**Quadruple 2 to 1 Line MUX**


Y = S1 S2 S3 D1 + S1 S2 S3 D2 + S1 S2 S3 D3 + S1 S2 S3 D4 + S1 S2 S3 D5 + S1 S2 S3 D6 + S1 S2 S3 D7 + S1 S2 S3 D8
D1
D2
D3
D4
D5
D6
D7
D8
S1
S2
S3
E
U1
U2
U3
U4
U5
U6
U7
U8
U9
U10
U11
U12
U13
U14
U15
U16

## Function

## Conclusion:

(i) We have learnt about Mux and Dmux.

(ii) We have learnt how to design a circuit of a mux and dmux.

(iii) We have learnt the differences between Mux and Dmux.

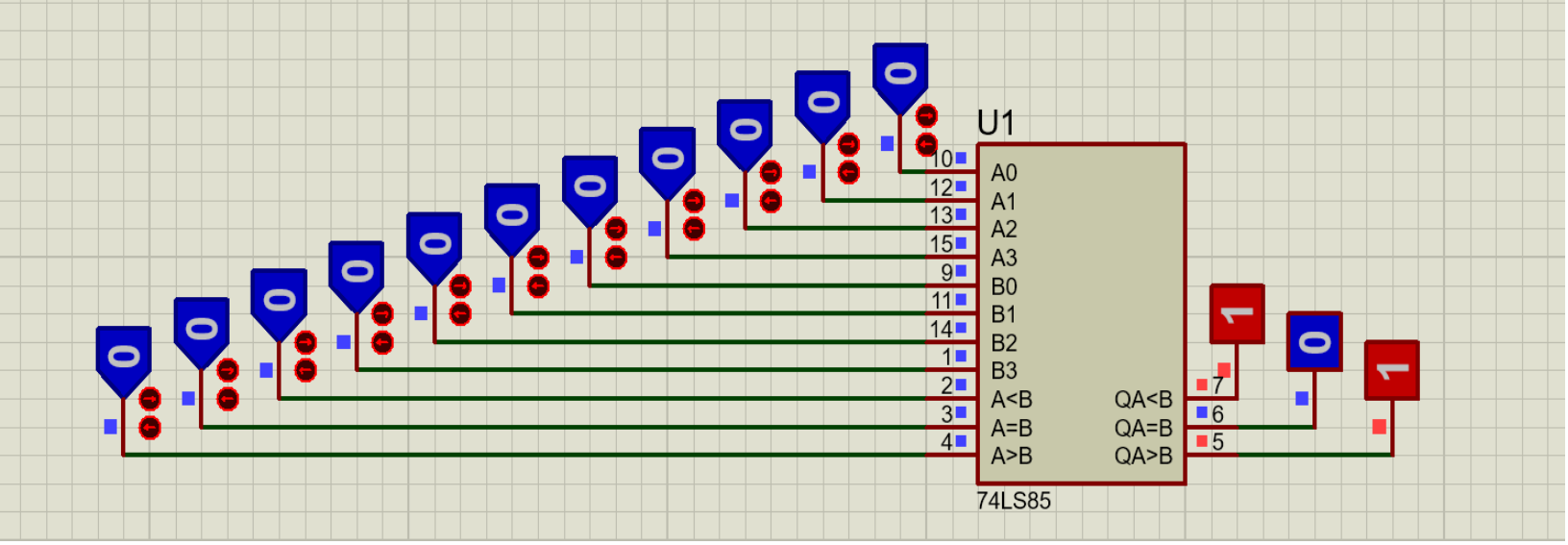
(iv) We have learnt how to implement functions using mux and dmux.

# LAB - 09

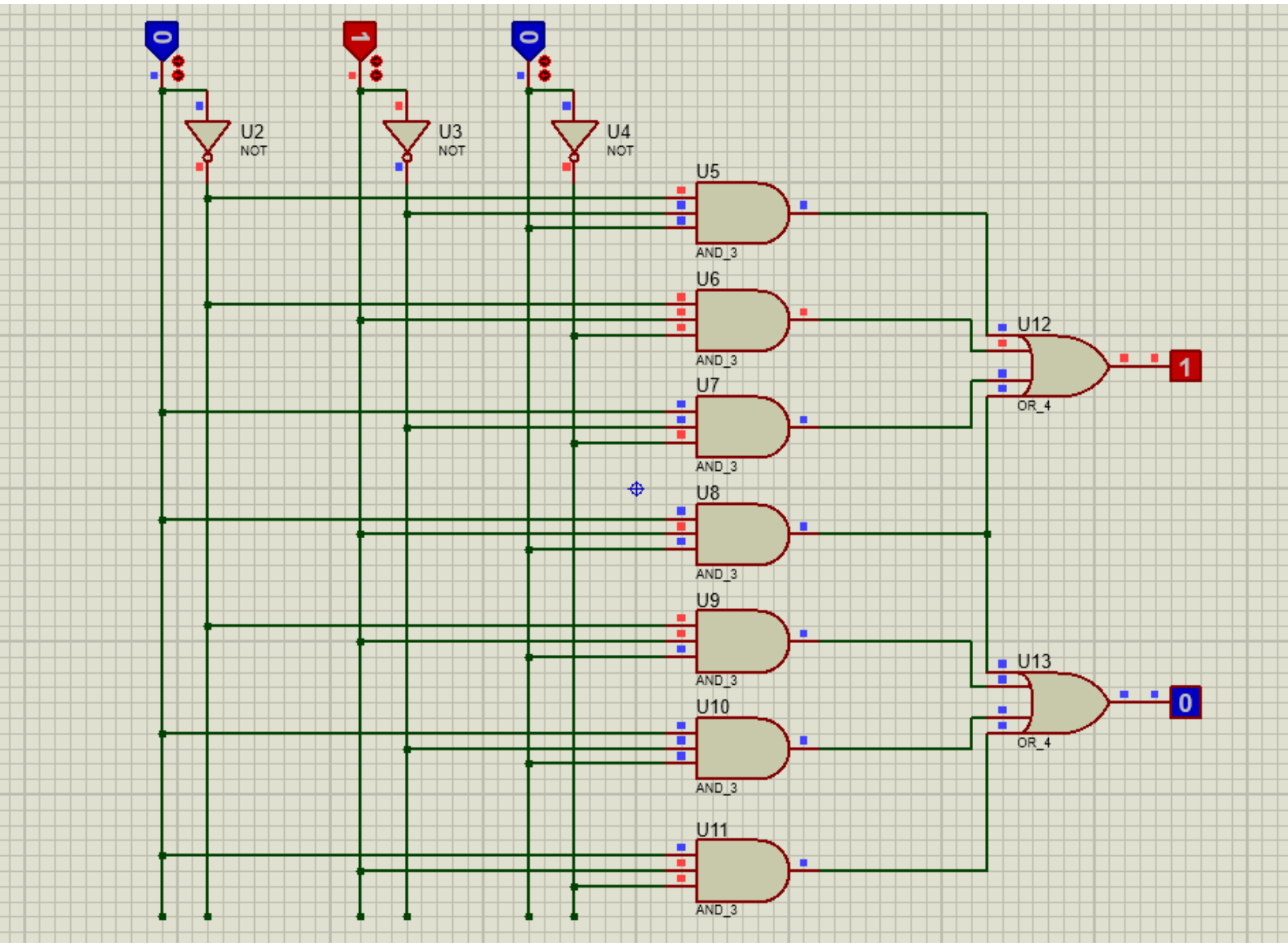
Name of the experiment: Designing of 4-bit magnitude comparator using 74LS85 Ic and full adder using decoder.

Description:

A comparator that is used for comparing two binary numbers each of four bits are called a 4-bit magnitude comparator. It compares whether a binary number is equal, less or greater than the other binary numbers. For building this or with logic gates we will have two inputs (A and B) and have three outputs: one for A>B, another one for A<B and lastly for A=B condition.




## 4–bit Magnitude Comparator Using 74LS85 IC




## Full Adder Using Decoder

## Conclution:

(i) We have learnt about magnitude comparator.

(ii) We have learnt about full adder and how to implement it with decoder.

(iii) We have learnt how to implement a 4-bit magnitude comparator using 74LS85 IC.

# LAB - 10

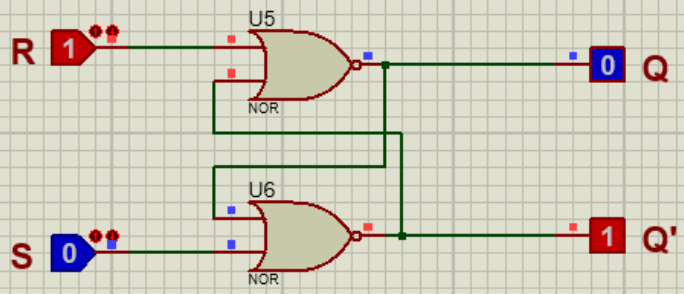
Name of the experiment: Construct, test and investigate the operation of various flipflops and latches.

Description:

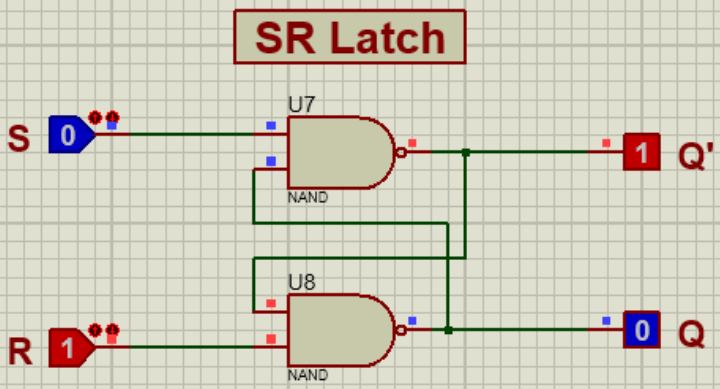
Latch: A latch is a storage device that holds the data using the feedback lane. The latch stores 1-bit until the devedevice set to 1. The latch changes the stored data and constantly trials the inputs when the enable input set to 1. Latches are (SR, JK, T, D, R) types.

Flip-flop: A flip-flop is an electronic circuit with two stable states that can be used to store binary data. The stored data can be changed by applying varying inputs. Flipflops are also having different types like- D-flip-flop, JK flipflop etc.

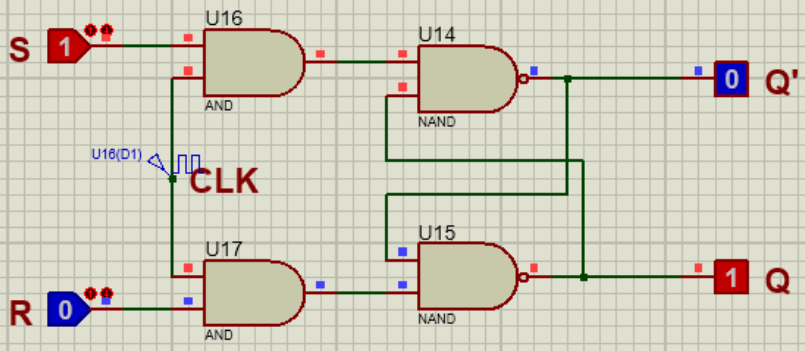
## RS Latch



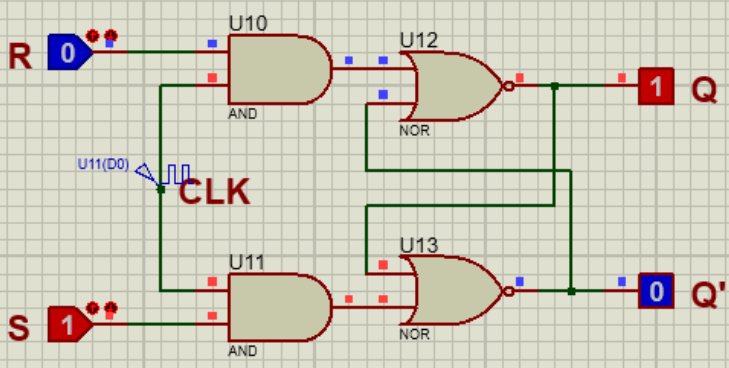

## SR Latch



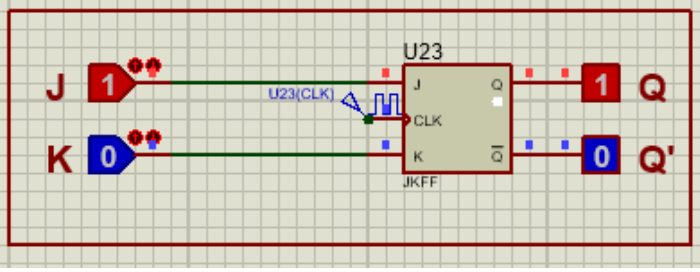

## SR Flip-Flop



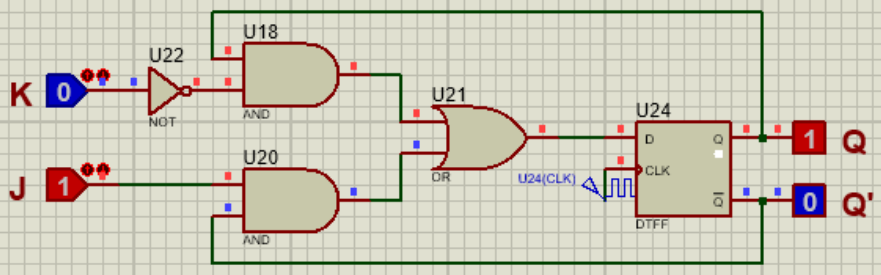

## RS Flip-Flop



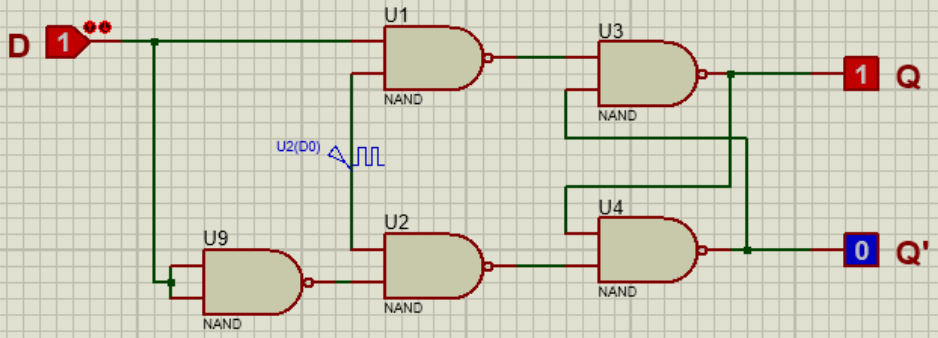

## JK Flip-Flop

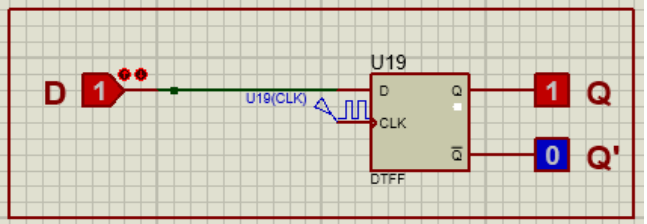



## JK Flip-Flop Using D Flip-Flop

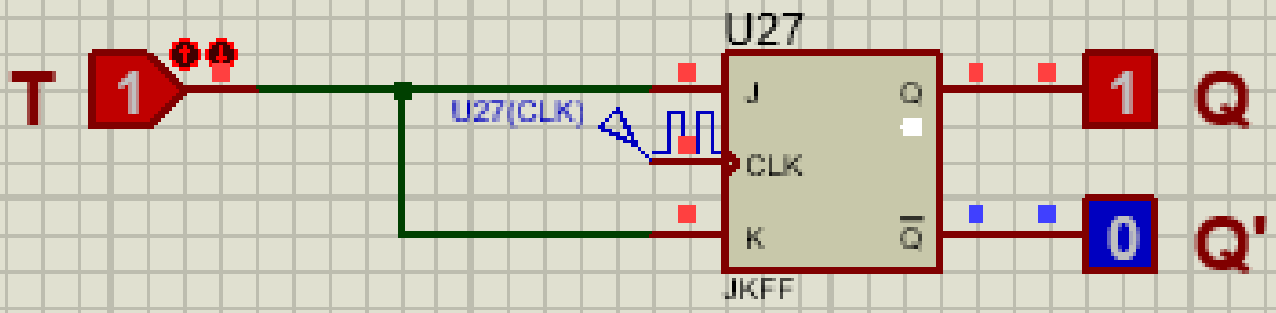



## D Flip-Flop

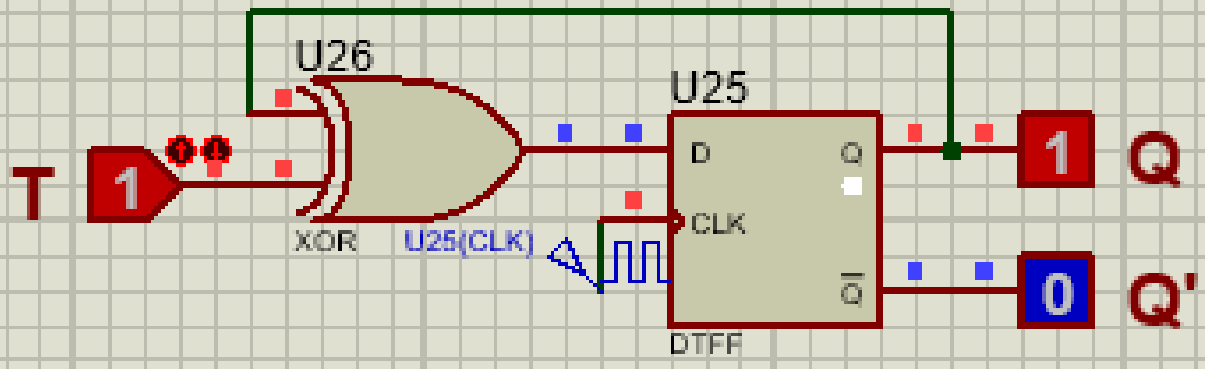


# T Flip-Flop Using JK Flip-Flop




# T Flip-Flop Using D Flip-Flop



## Conclusion:

(i) We have learnt about the topic of latches and flip-flops.

(ii) We have learnt how to implement circuits of latches and flipflops.

(iii) We learnt how we can make flipflops with latches.

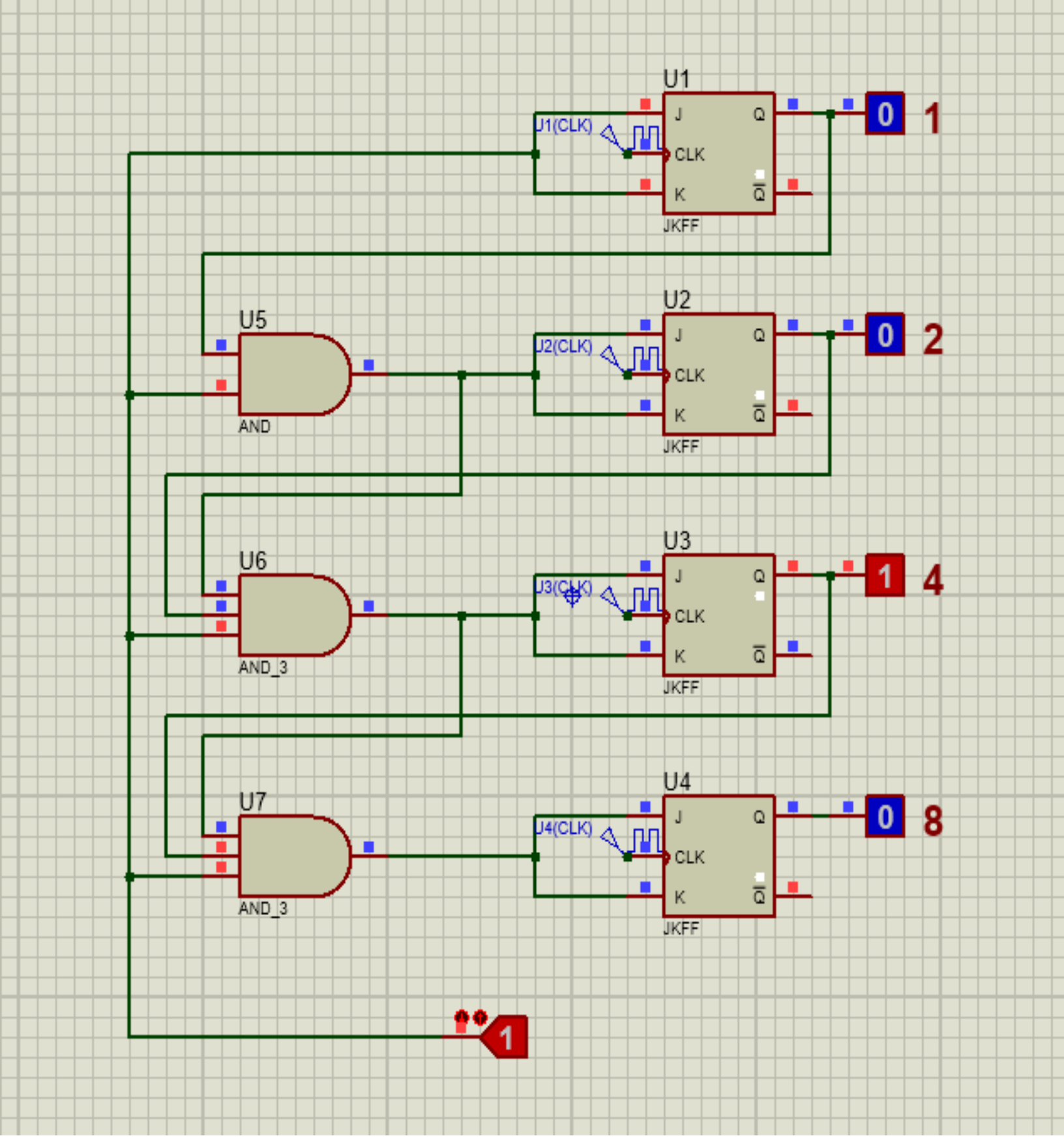
# LAB - 11

Name of the experiment: Construction of counters and Registers.

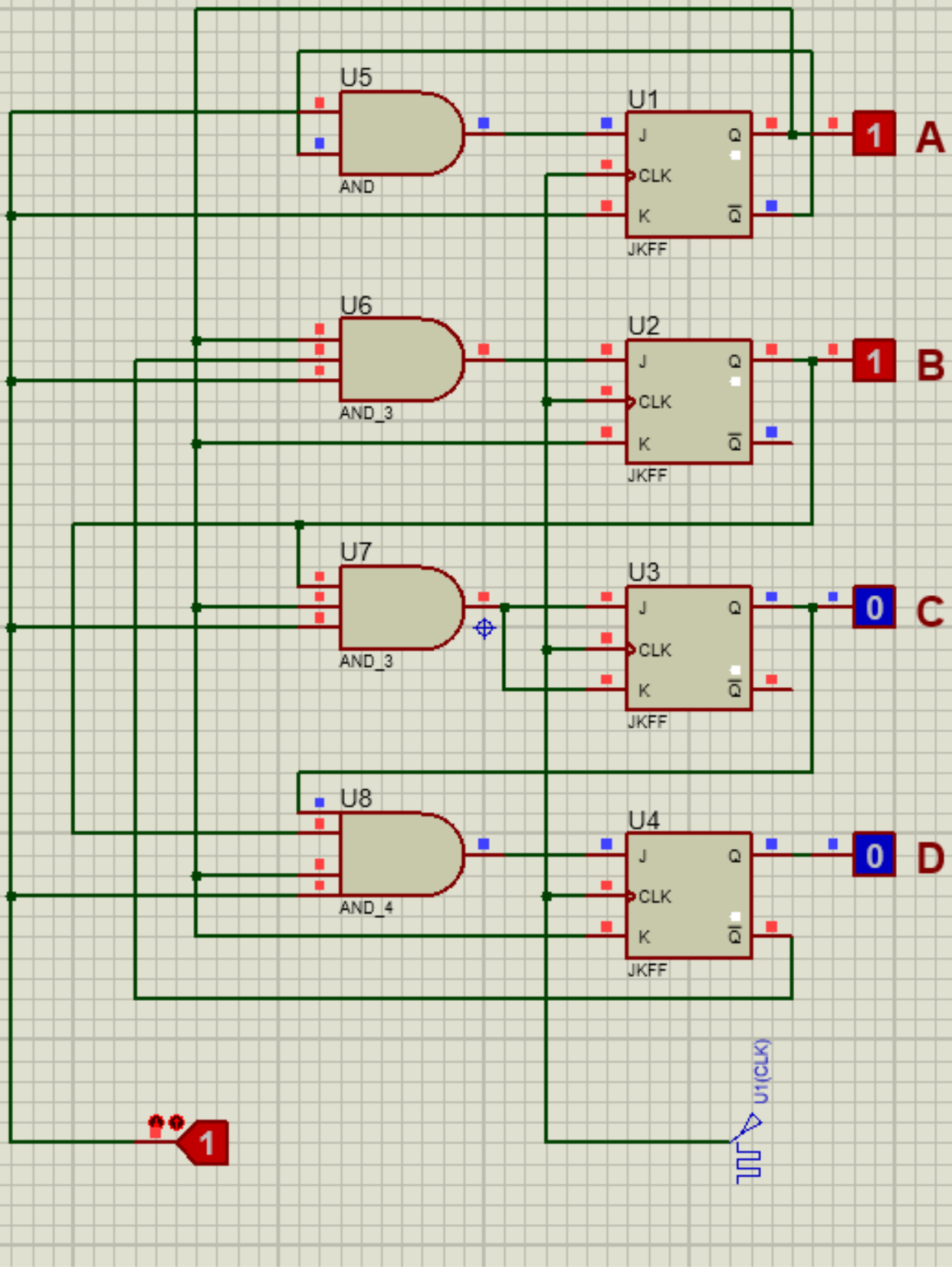
Description:

BCD counter: A BCD counter is one of the types of most widely used digital counters, which counts up to 10 with an applied clock signal. It is a 4-bit ~~to~~ binary digital counter, counts from 1(0001) to 10(1010).

Binary counter: A binary counter is a hardware circuit that is made out of a series of flip-flops. The output of one flip-flop is sent to the input of the next flip-flop in the series. A binary counter can be either asynchronous or synchronous, depending on how the flipflops are connected.




**Binary Counter**


U5
AND
U1
J
CLK
K
Q
JKFF
1
A
U6
AND_3
U2
1
B
U7
AND_3
U3
0
C
U8
AND_4
U4
0
D
1
U1(CLK)

**BCD Counter**

## Conclusion:

(i) we have learnt about counter.

(ii) we have learnt about BCD counter and it's implementation.

(iii) we have learnt about Binary counter and it's implementation.

# THE END