DIGITAL LOGIC AND DESIGN

LAB ASESSMENT – 4

Name: **VIBHU KUMAR SINGH**
Reg. No: **19BCE0215**
Teacher: **Sairabanu J.**

EXPERIMENT 5:

Q1) Design a 4-bit Binary parallel Adder.

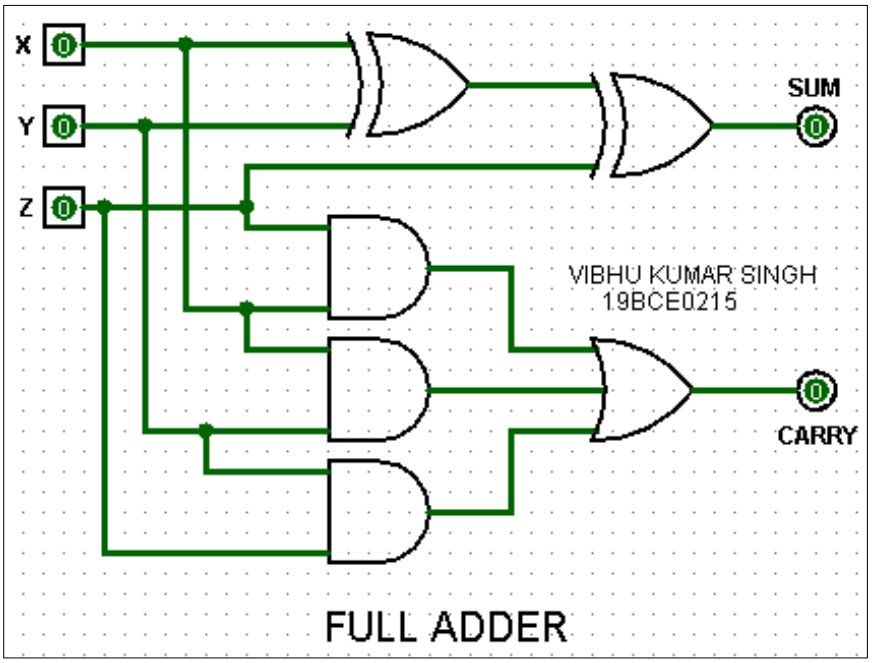
A1)

AIM:

To design a 4-bit Binary Parallel Adder.

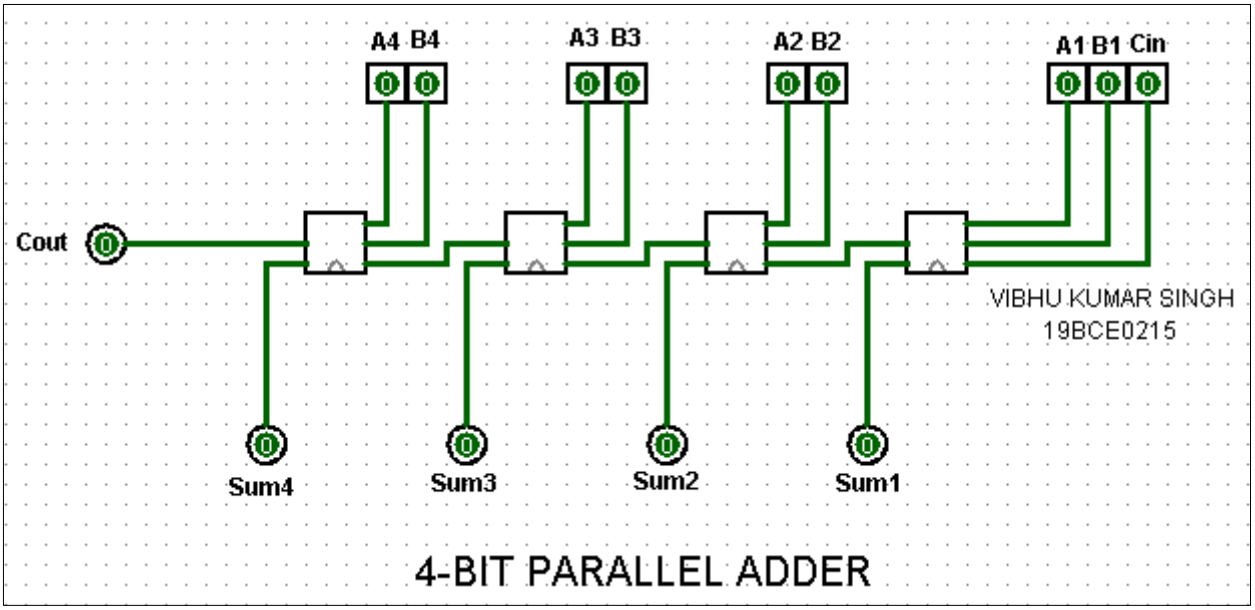
DESIGN:

In order to make 4-bit Binary Parallel Adder, a single Full Adder circuit is saved as user I.C. and imported in another Logisim file to make Binary Parallel Adder. The circuit for Full Adder is:



FULL ADDER				
INPUT			OUTPUT	
X	Y	Z	Sum	Carry
0	0	0	**0**	**0**
0	0	1	**1**	**0**
0	1	0	**1**	**0**
0	1	1	**1**	**1**
1	0	0	**1**	**0**
1	0	1	**0**	**1**
1	1	0	**0**	**1**
1	1	1	**1**	**1**

This Full Adder circuit is saved as user I.C. and imported in another file. The circuit for 4-bit Binary Parallel Adder is:



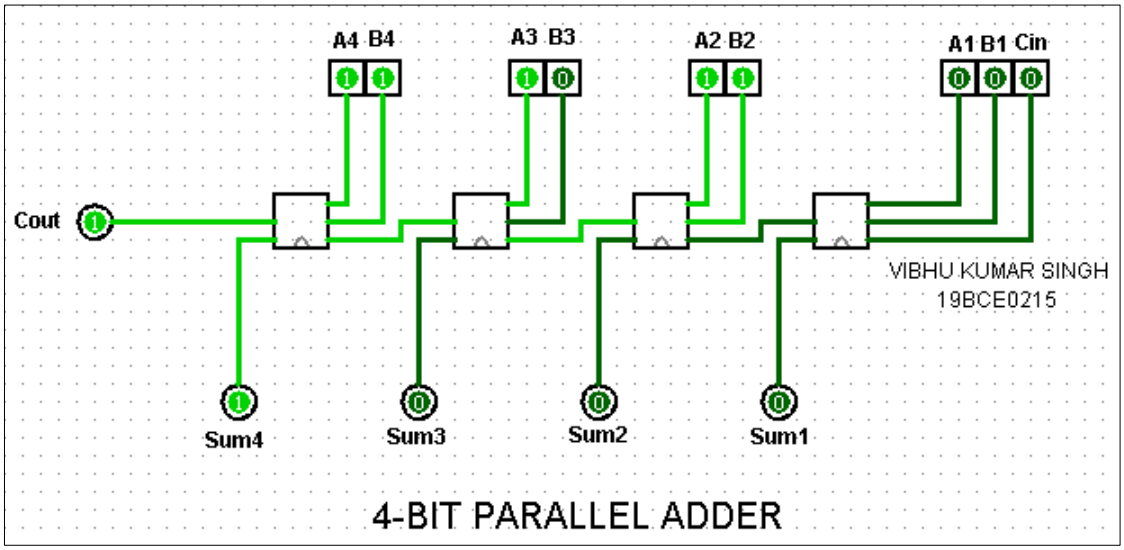
SCREENSHOTS:

EXAMPLE 1:

A=1110 B=1010

	A_4/B_4	A_3/B_3	A_2/B_2	A_1/B_1
	1	1	1	0
+	1	0	1	0
1	**1**	**0**	**0**	**0**

OUTPUT 1:

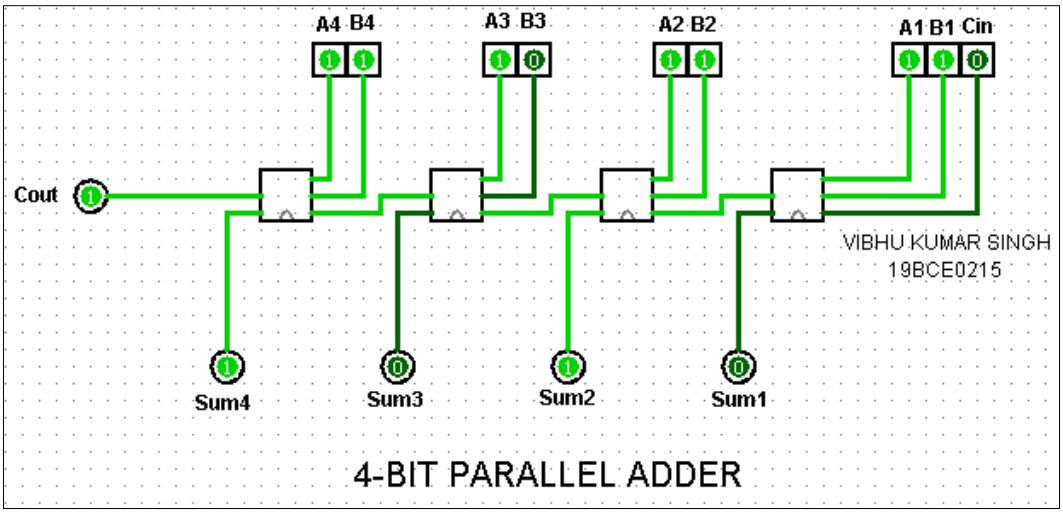


EXAMPLE 2:

A=1111 B=1011

	A_4/B_4	A_3/B_3	A_2/B_2	A_1/B_1
	1	1	1	1
+	1	0	1	1
1	**1**	**0**	**1**	**0**

OUTPUT 2:

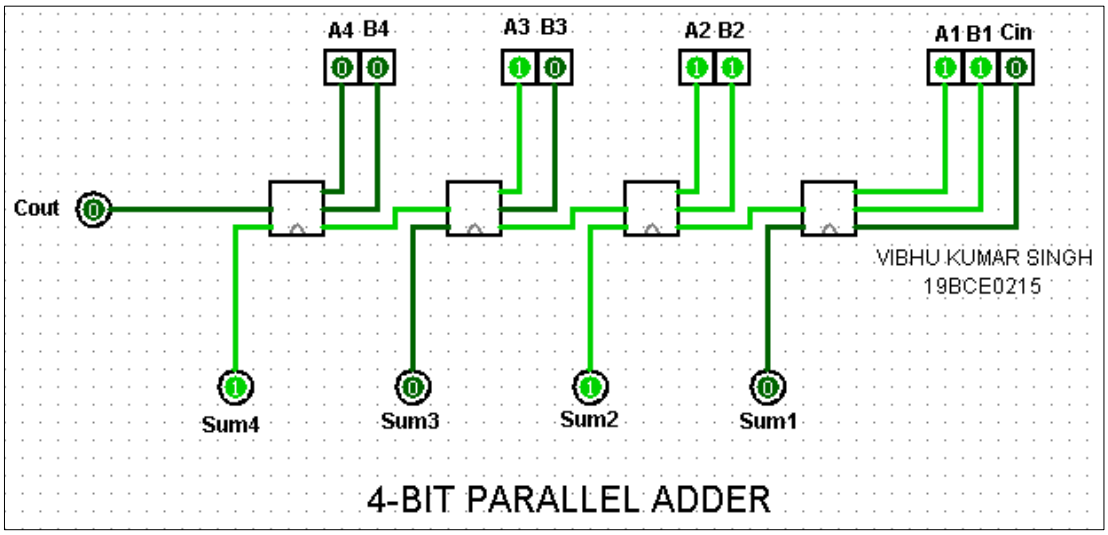


EXAMPLE 3:

A=0111 B=0011

	A_4/B_4	A_3/B_3	A_2/B_2	A_1/B_1
	0	1	1	1
+	0	0	1	1
0	**1**	**0**	**1**	**0**

OUTPUT 3:

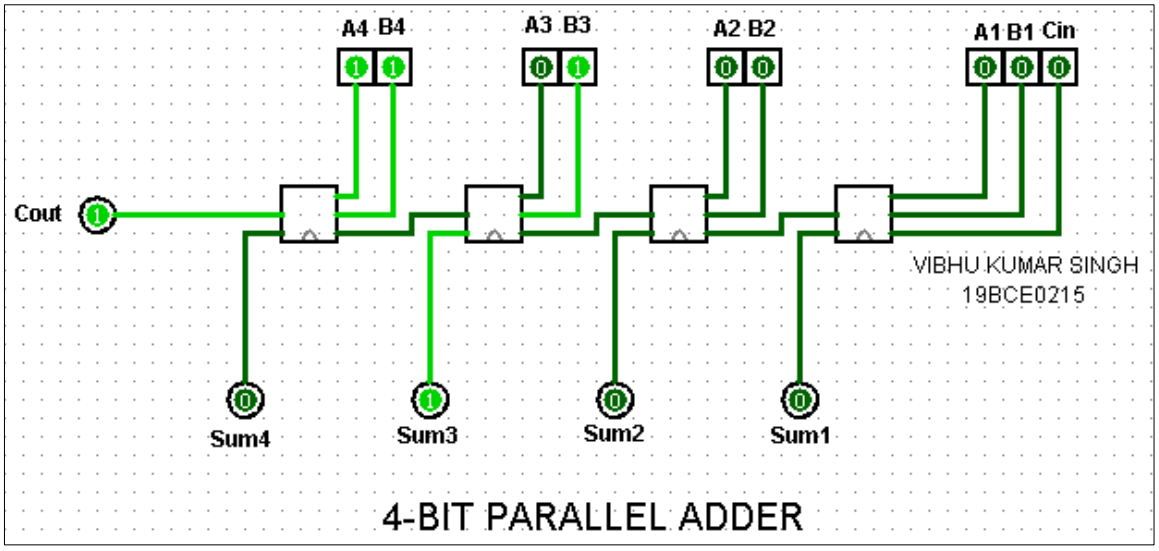


EXAMPLE 4:

A=1000 B=1100

	A_4/B_4	A_3/B_3	A_2/B_2	A_1/B_1
	1	0	0	0
+	1	1	0	0
1	**0**	**1**	**0**	**0**

OUTPUT 4:



Q2) Design Excess 5 to BCD using Binary parallel adder.

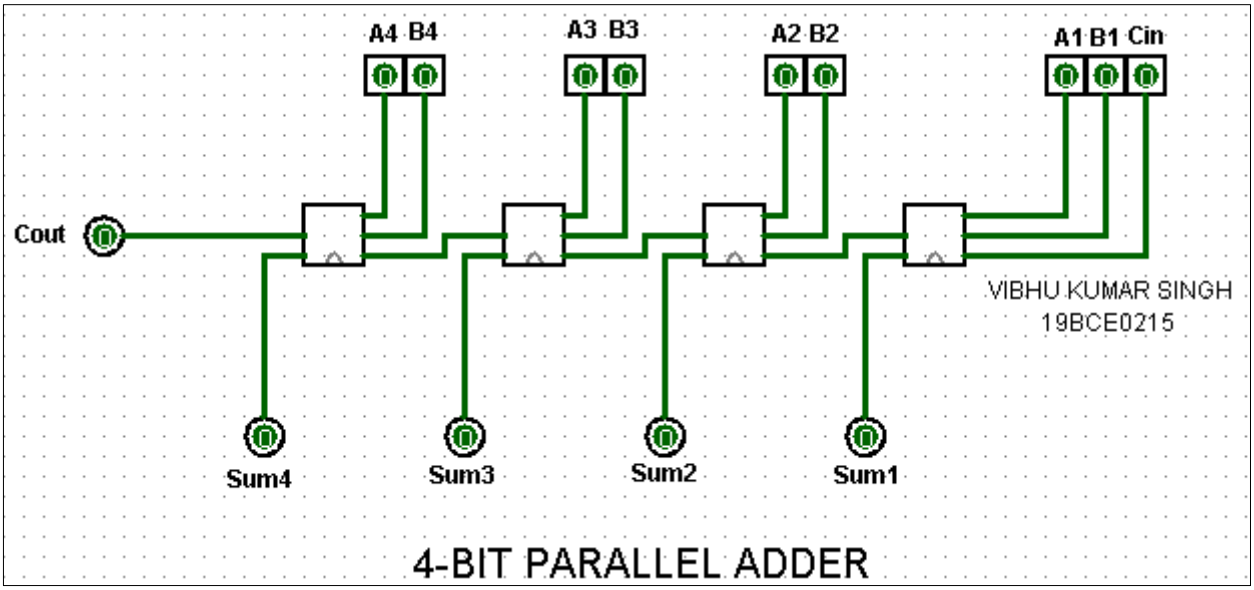
A2)

AIM:

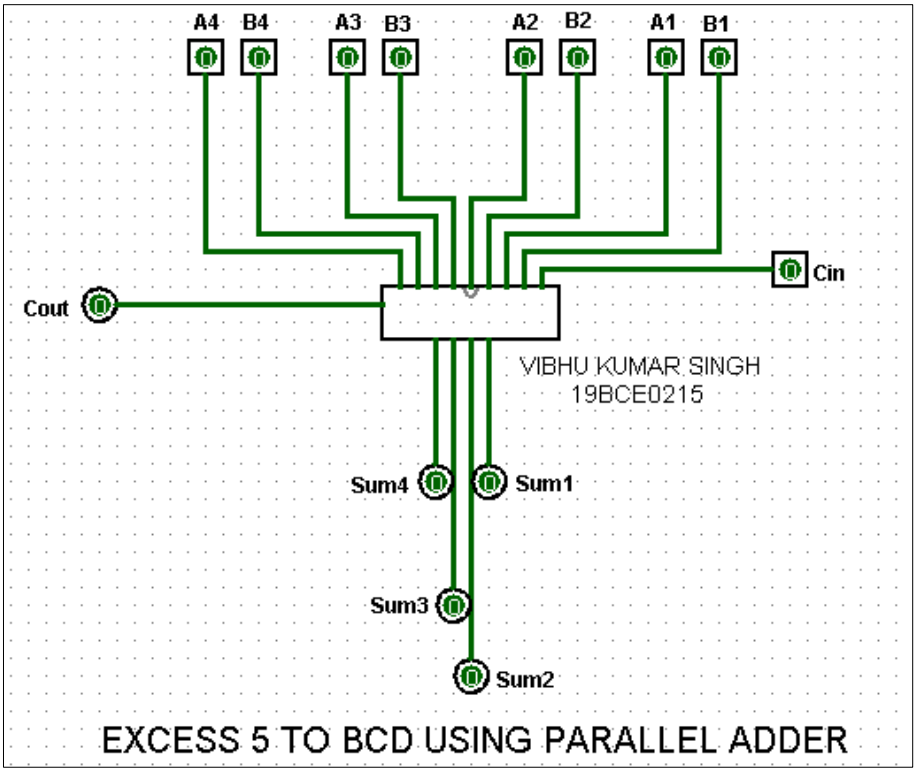
To design Excess 5 to BCD using Binary Parallel Adder.

DESIGN:

It has the same principle as a regular 4-bit Binary Parallel Adder except we will be subtracting 5(0101) from the given BCD code using 2's complement subtraction ie. add (1011) . The circuit for Binary Parallel Adder is saved as user I.C. and imported in a new Logisim file. The circuit for Binary Parallel Adder is:



This Binary Parallel Adder circuit is saved as user I.C. and imported in another file. The circuit for Excess 5 to BCD converter using Binary Parallel Adder is:



SCREENSHOTS:

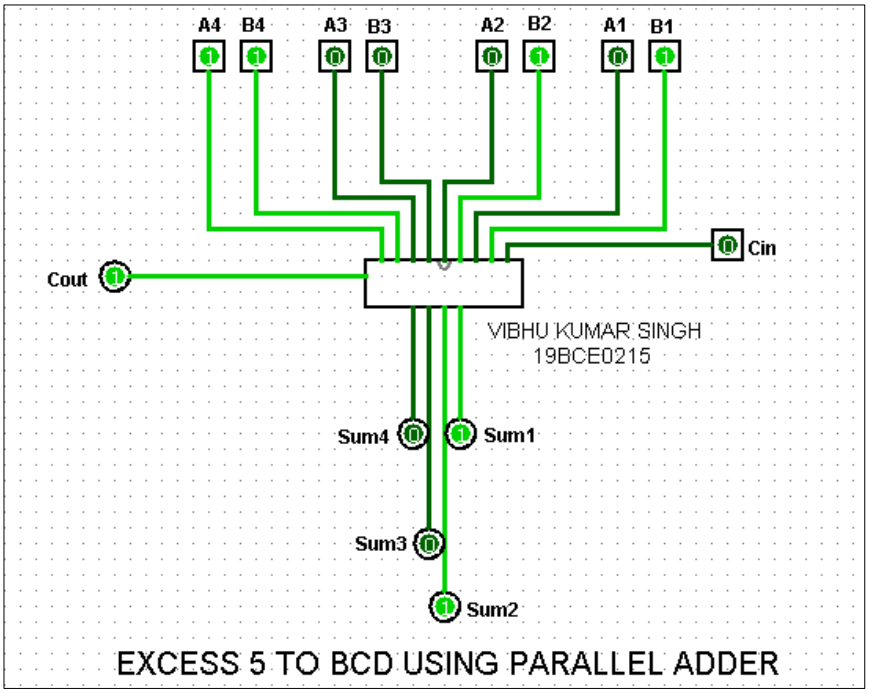
EXAMPLE 1:

Convert A=1000 into BCD code.

	A_4/B_4	A_3/B_3	A_2/B_2	A_1/B_1
	1	0	0	0
+	1	0	1	1
1	**0**	**0**	**1**	**1**

Since end carry is generated, it is discarded.
So, the BCD code is **0011**.

OUTPUT 1:



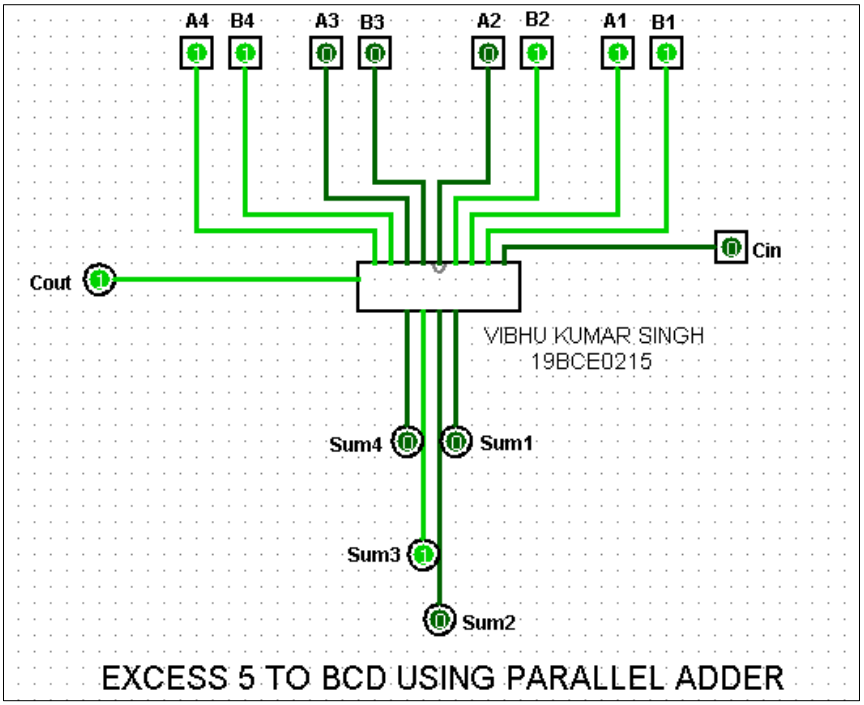
EXAMPLE 2:

Convert A=1001 into BCD code.

	A_4/B_4	A_3/B_3	A_2/B_2	A_1/B_1
	1	0	0	1
+	1	0	1	1
1	**0**	**1**	**0**	**0**

Since end carry is generated, it is discarded.
So, the BCD code is **0100**.

OUTPUT 2:



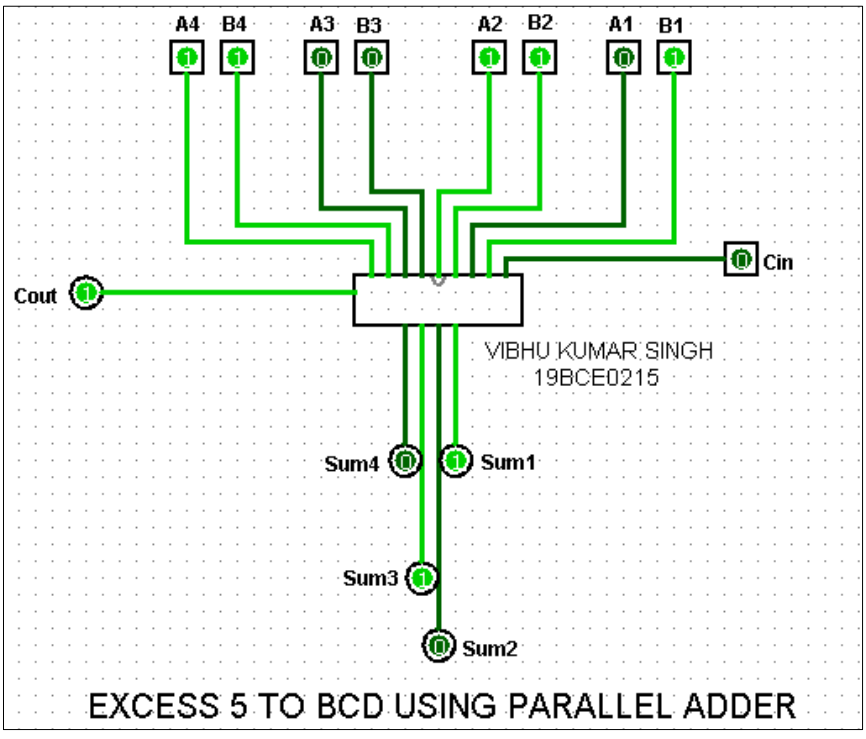
EXAMPLE 3:

Convert A=1010 into BCD code.

	A_4/B_4	A_3/B_3	A_2/B_2	A_1/B_1
	1	0	1	0
+	1	0	1	1
1	**0**	**1**	**0**	**1**

Since end carry is generated, it is discarded.
So, the BCD code is **0101**.

OUTPUT 3:



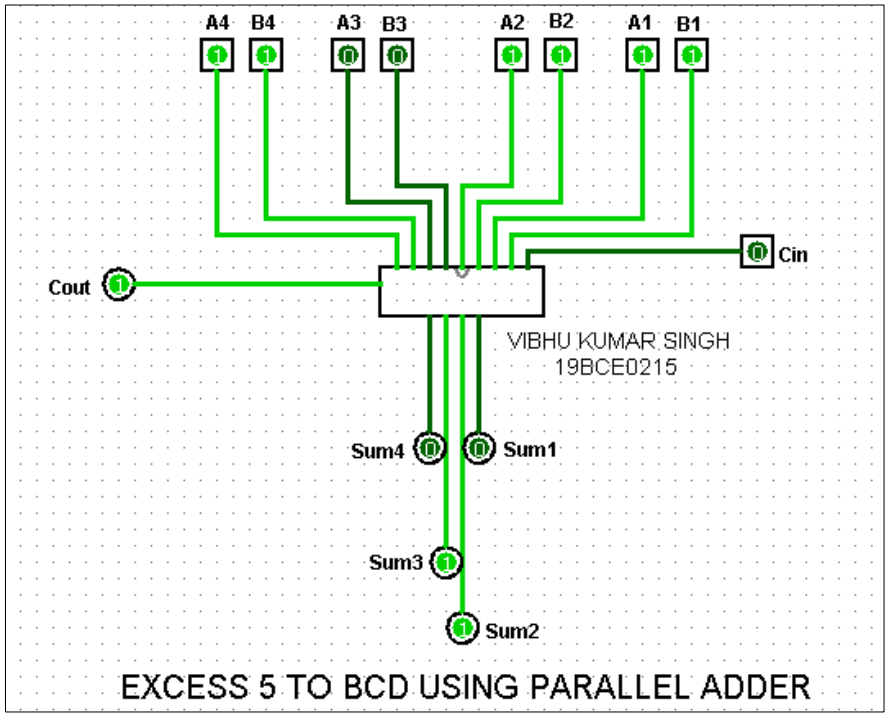
EXAMPLE 4:

Convert A=1011 into BCD code.

	A_4/B_4	A_3/B_3	A_2/B_2	A_1/B_1
	1	0	1	1
+	1	0	1	1
1	**0**	**1**	**1**	**0**

Since end carry is generated, it is discarded.
So, the BCD code is **0110**.

OUTPUT 4:



Q3) Design a 2-bit magnitude comparator.

A3)

AIM:

To design a 2-bit Magnitude Comparator.

TRUTH TABLE:

2-BIT MAGNITUDE COMPARATOR				
INPUT		OUTPUT		
A_2 A_1	B_2 B_1	A>B	A=B	A<B
0 0	0 0	**0**	**1**	**0**
0 0	0 1	**0**	**0**	**1**
0 0	1 0	**0**	**0**	**1**
0 0	1 1	**0**	**0**	**1**
0 1	0 0	**1**	**0**	**0**
0 1	0 1	**0**	**1**	**0**
0 1	1 0	**0**	**0**	**1**
0 1	1 1	**0**	**0**	**1**
1 0	0 0	**1**	**0**	**0**
1 0	0 1	**1**	**0**	**0**
1 0	1 0	**0**	**1**	**0**
1 0	1 1	**0**	**0**	**1**
1 1	0 0	**1**	**0**	**0**
1 1	0 1	**1**	**0**	**0**
1 1	1 0	**1**	**0**	**0**
1 1	1 1	**0**	**1**	**0**

BOOLEAN EXPRESSIONS:

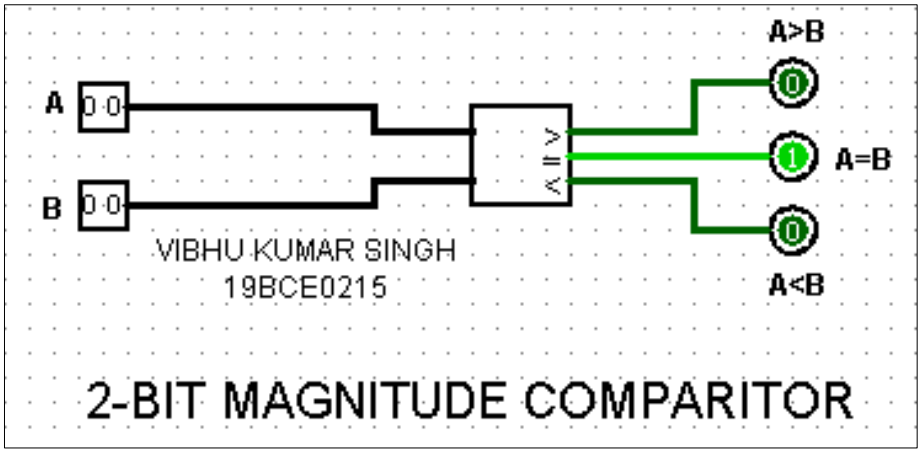
- FOR (A>B):
  =**$A_1.B_1$' + $X_1.A_0.B_0$'**

- FOR (A=B):
  =**X1.X0**

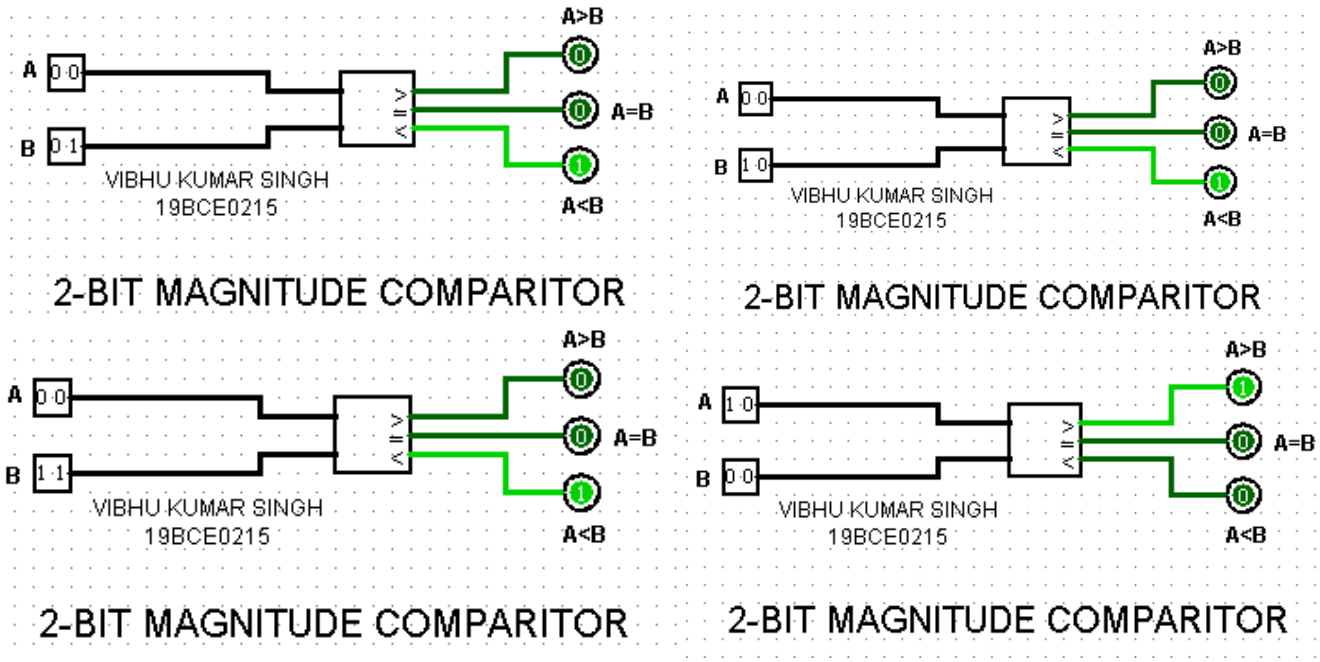
- FOR (A<B):
  =**A1'.B1 + X1.A0'.B0**

DESIGN:

The circuit of 2-bit Magnitude Comparator is made using a predefined I.C. in Logisim. The circuit for 2-bit Magnitude Comparator is:



SCREENSHOTS(4):



Q4) Design a circuit to determine the greatest of 3 2-bit numbers using a 2 bit magnitude comparator.

A4)

AIM:

Design a circuit to determine the greatest of 3 2-bit numbers using a 2-bit Magnitude Comparator.

TRUTH TABLE:

2-BIT MAGNITUDE COMPARATOR				
INPUT		OUTPUT		
A_2 A_1	B_2 B_1	A>B	A=B	A<B
0 0	0 0	**0**	**1**	**0**
0 0	0 1	**0**	**0**	**1**
0 0	1 0	**0**	**0**	**1**
0 0	1 1	**0**	**0**	**1**
0 1	0 0	**1**	**0**	**0**
0 1	0 1	**0**	**1**	**0**
0 1	1 0	**0**	**0**	**1**
0 1	1 1	**0**	**0**	**1**
1 0	0 0	**1**	**0**	**0**
1 0	0 1	**1**	**0**	**0**
1 0	1 0	**0**	**1**	**0**
1 0	1 1	**0**	**0**	**1**
1 1	0 0	**1**	**0**	**0**
1 1	0 1	**1**	**0**	**0**
1 1	1 0	**1**	**0**	**0**
1 1	1 1	**0**	**1**	**0**

BOOLEAN EXPRESSIONS:

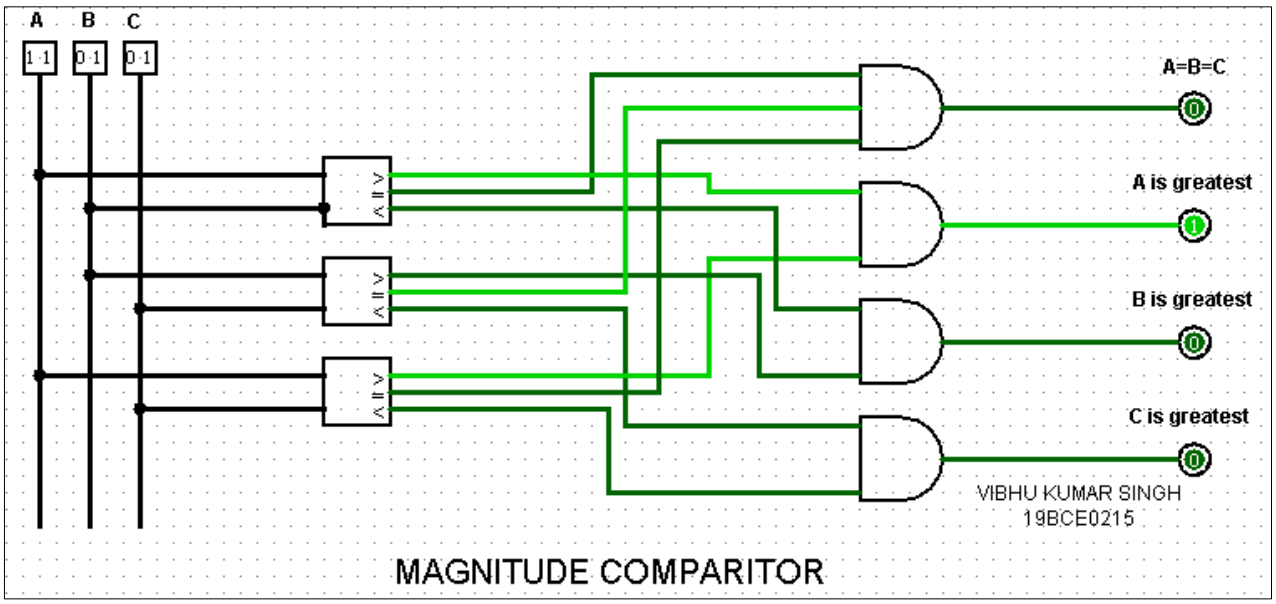
- FOR (A>B):
  $=A_1.B_1' + X_1.A_0.B_0'$

- FOR (A=B):
  $=X1.X0$

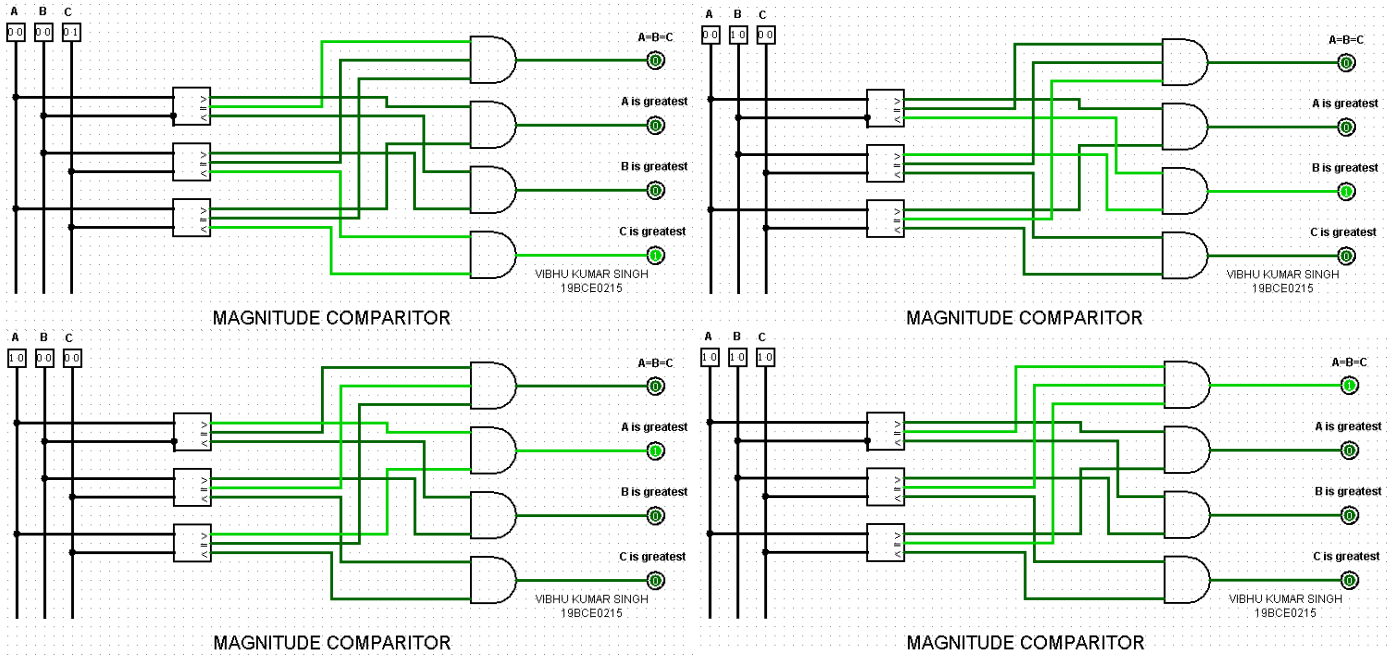
- FOR (A<B):
  $=A1'.B1 + X1.A0'.B0$

DESIGN:

Three 2-bit Magnitude Comparator I.C.(s) are used to determine the greatest of 3 2-bit numbers. The circuit is as follows:



SCREENSHOTS(4):



EXPERIMENT 6:

Q1) To design 2 to 4 line decoder using AND Gates.

A1)

AIM:

To design 2 to 4 line decoder using AND Gates.

TRUTH TABLE:

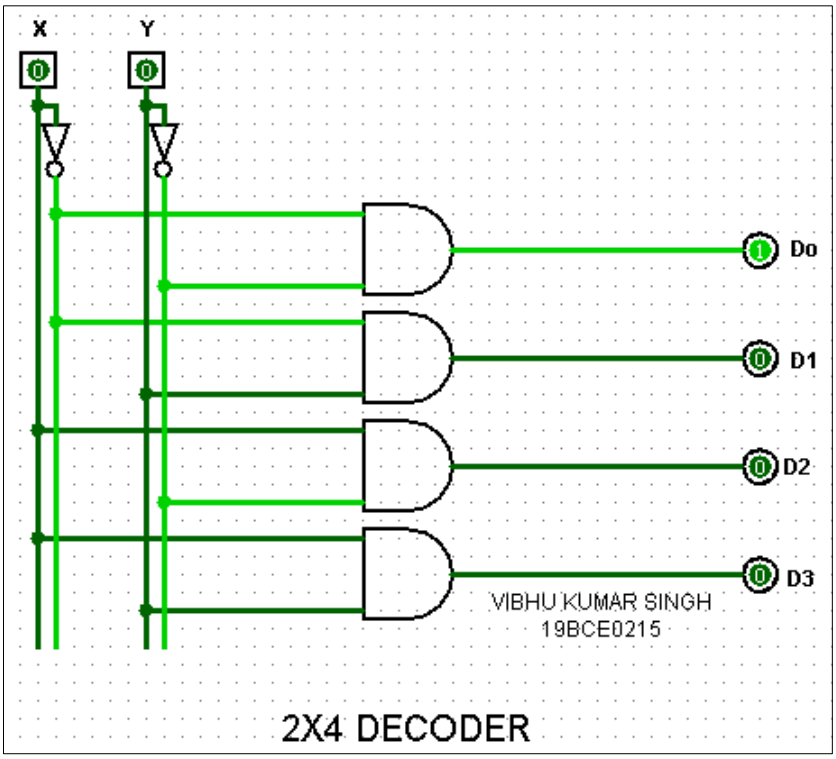
2-TO-4 DECODER					
INPUT		OUTPUT			
X	Y	D_0	D_1	D_2	D_3
0	0	**1**	**0**	**0**	**0**
0	1	**0**	**1**	**0**	**0**
1	0	**1**	**0**	**1**	**0**
1	1	**0**	**0**	**0**	**1**

BOOLEAN EXPRESSIONS:

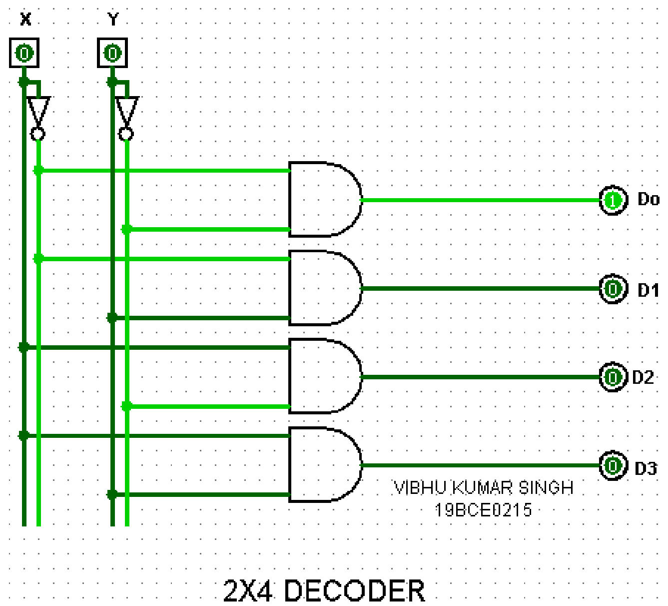
$$D_0 = X'Y'$$
$$D_1 = X'Y$$
$$D_2 = XY'$$
$$D_3 = XY$$

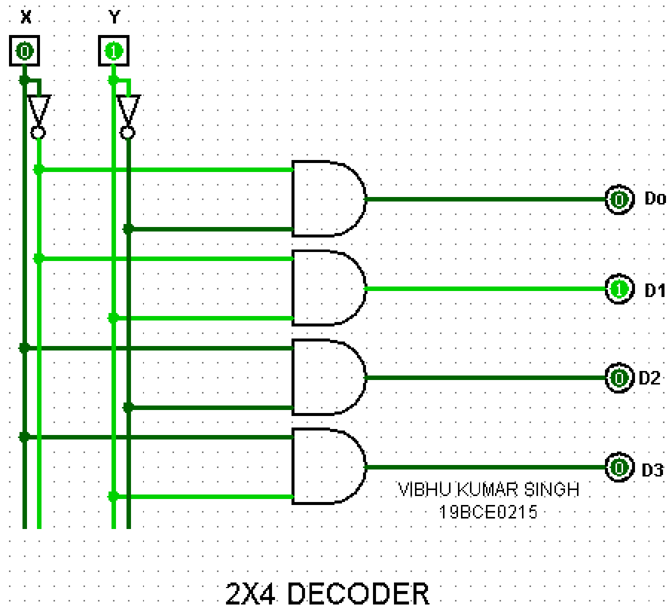
DESIGN:

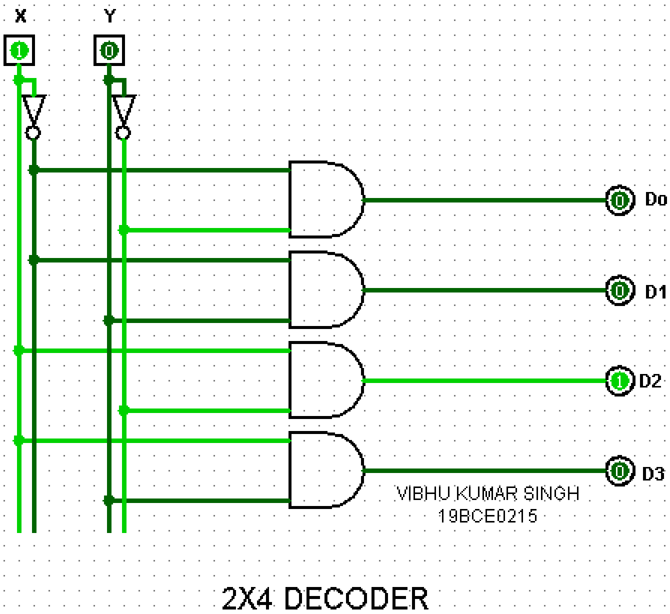
Using the above Boolean expressions, the Decoder can be made using basic AND Gates. The circuit for 2 to 4 line decoder is:

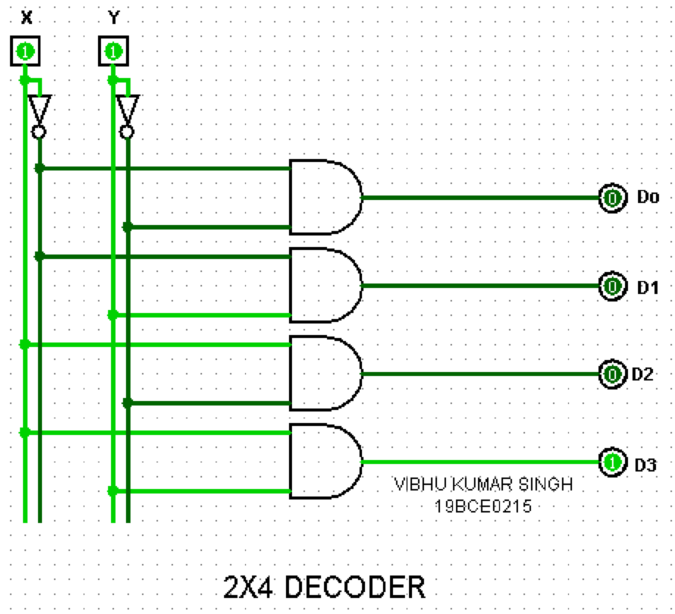


SCREENSHOTS(4):









Q2) Design a logic circuit to control the traffic light as per the given details. Vehicle detection sensors are placed along C and D (main road) and lanes A and B (access road). These sensor outputs are LOW (0) when no vehicle is present and HIGH (1) when a vehicle is present. The intersection traffic light is to be controlled according to the following logic:

a. **The east-west (E-W) traffic light will be green whenever both lanes C and D are occupied.**
b. **The E-W light will be green whenever either C or D is occupied but lanes A or B are not occupied.**
c. **The north-south (N-S) light will be green whenever both lanes A and B are occupied but C or D are not occupied.**
d. **The N-S light will also be green when either A or B is occupied while C and D are both vacant.**
e. **The E-W light will be green when no vehicles are present. Using the sensor outputs A, B, C and D as inputs, N-S and E-W be two outputs that go high when the corresponding light to be green. Simulate the above scenario using decoders.**

A2)

AIM:

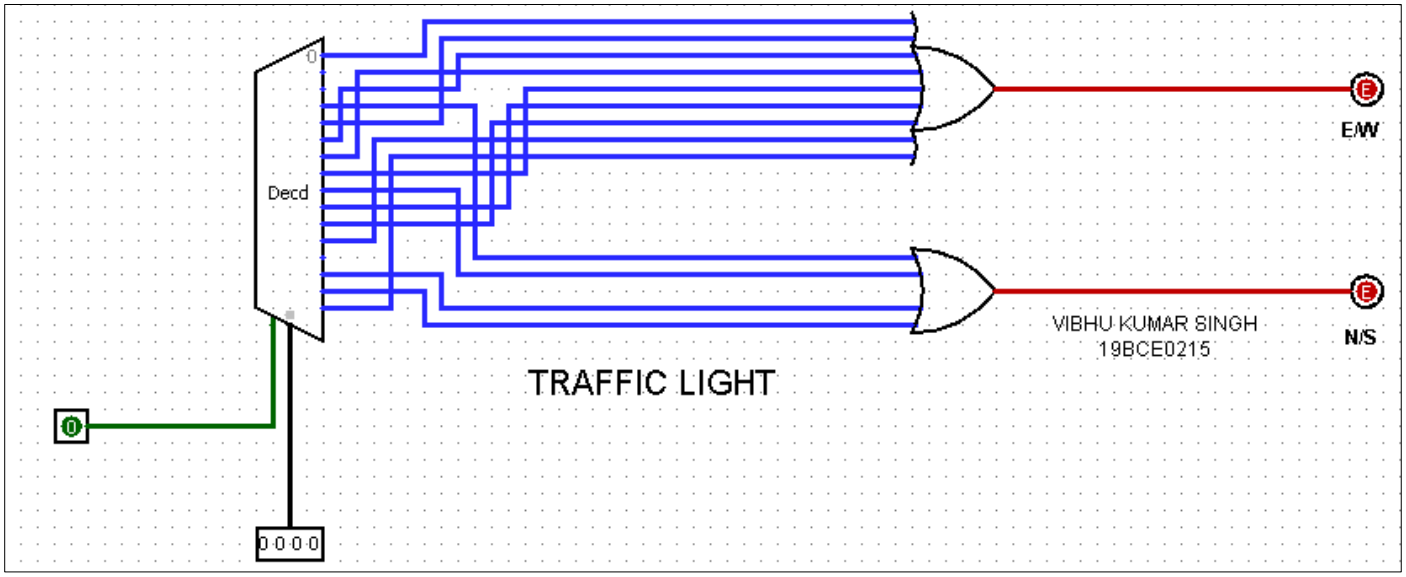
To obtain a circuit to solve the traffic light problem.

TRUTH TABLE:

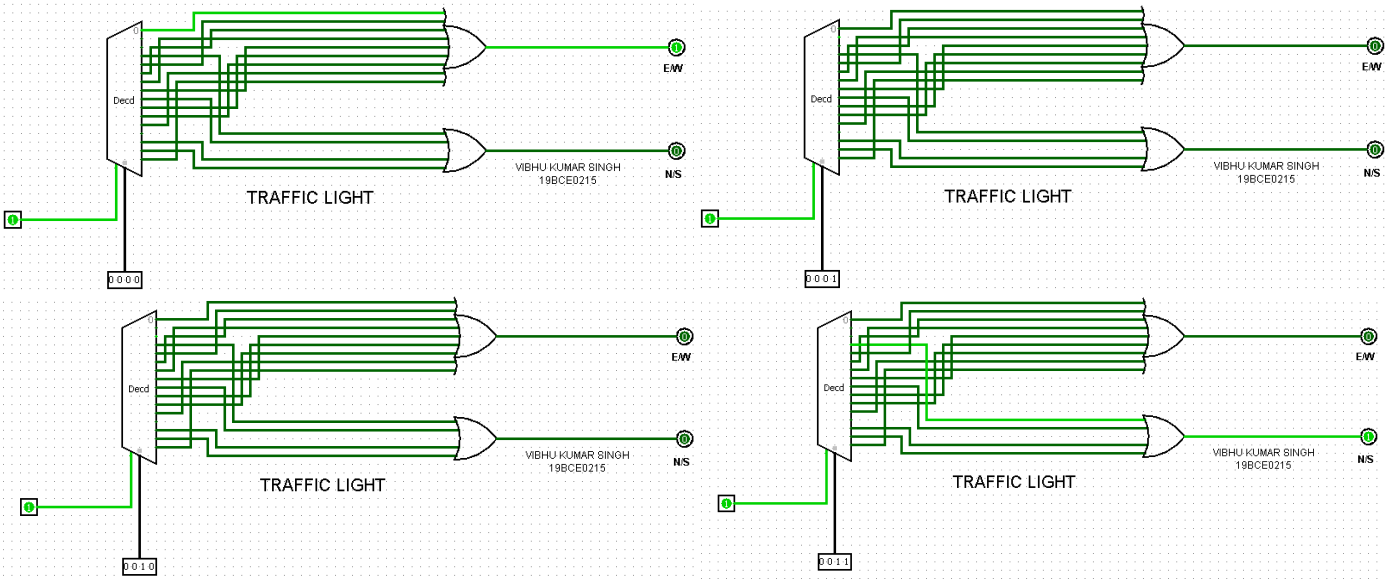
TRAFFIC LIGHT PROBLEM					
INPUT				OUTPUT	
A	B	C	D	E/W	N/S
0	0	0	0	**1**	**0**
0	0	0	1	**0**	**0**
0	0	1	0	**0**	**0**
0	0	1	1	**1**	**0**
0	1	0	0	**1**	**0**
0	1	0	1	**1**	**0**
0	1	1	0	**1**	**0**
0	1	1	1	**1**	**0**
1	0	0	0	**0**	**1**
1	0	0	1	**1**	**0**
1	0	1	0	**1**	**0**
1	0	1	1	**1**	**0**
1	1	0	0	**0**	**0**
1	1	0	1	**0**	**1**
1	1	1	0	**0**	**1**
1	1	1	1	**1**	**0**

DESIGN:

This can be designed using a 4-to-16 Decoder and simple OR Gates in the following manner:



SCREENSHOTS:



Q3) Show how two 4-to-1 and one 2-to-1 multiplexers could be connected to form an 8-to-1 MUX with three control inputs.

A3)

AIM:

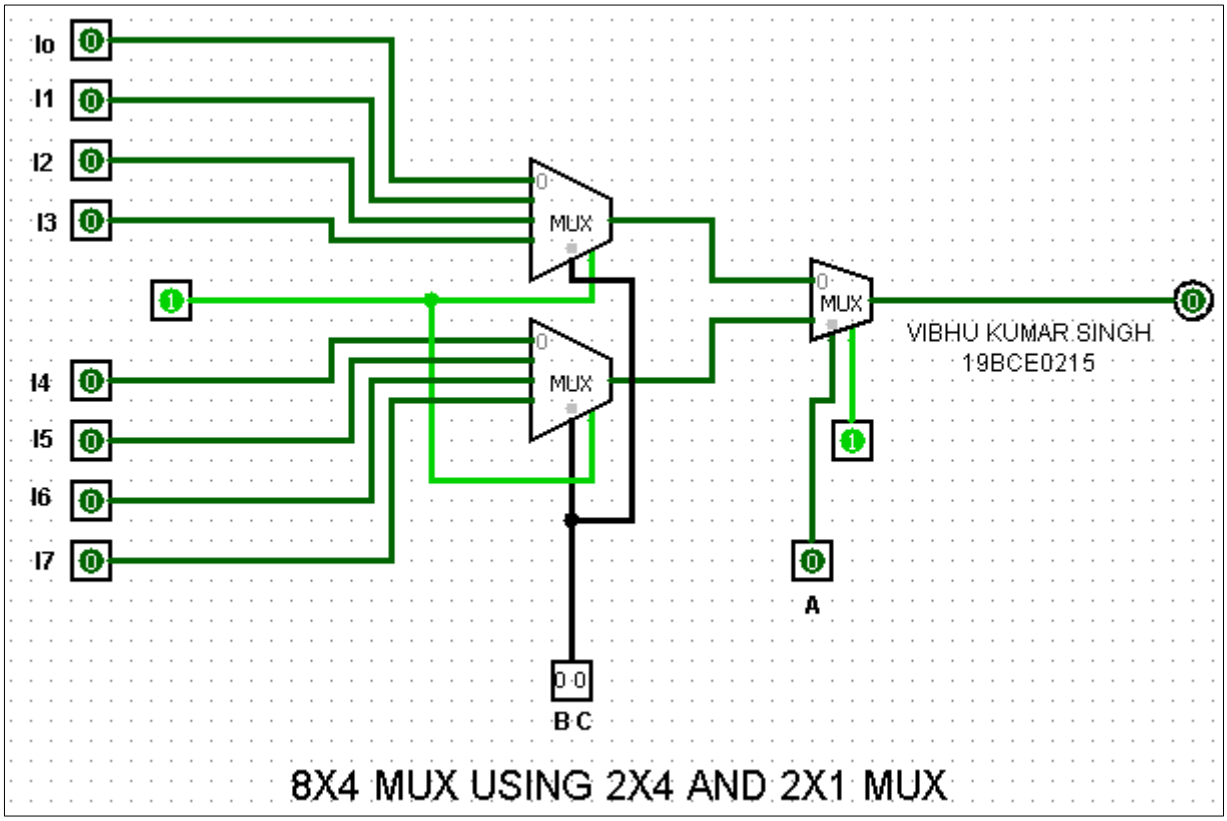
To design 8-to-1 MUX using two 4-to-1 and one 2-to-1 MUX.

TRUTH TABLE:

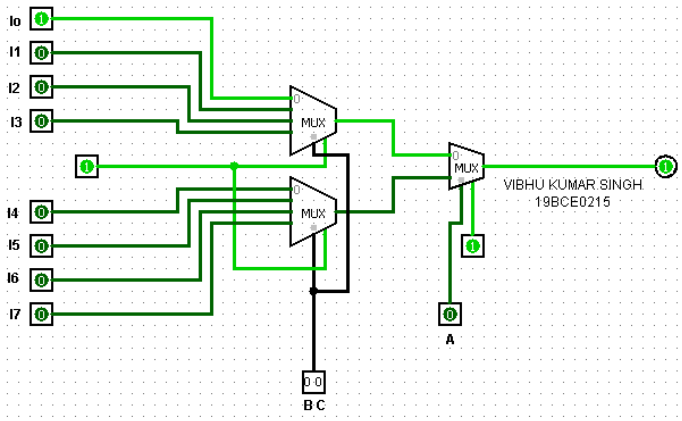
8-TO-1 MUX			
INPUT			OUTPUT
A	B	C	
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

DESIGN:

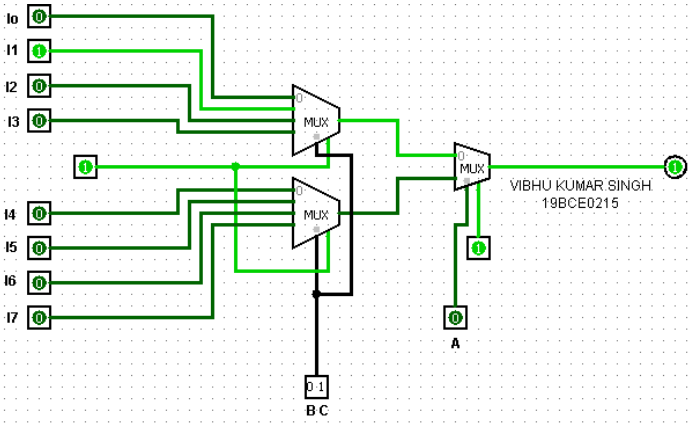
The 8-to-1 MUX using two 4-to-1 and one 2-to-1 MUX can be made like this:



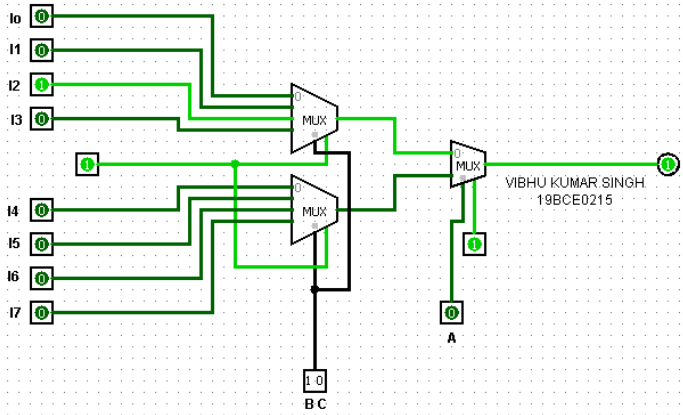
SCREENSHOTS(4):



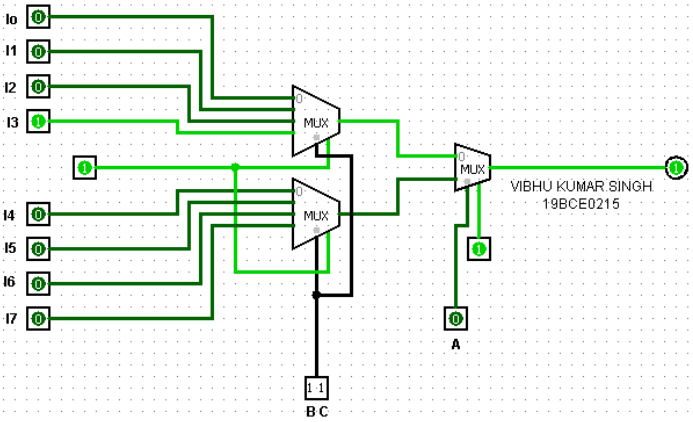
8X4 MUX USING 2X4 AND 2X1 MUX



8X4 MUX USING 2X4 AND 2X1 MUX



8X4 MUX USING 2X4 AND 2X1 MUX



8X4 MUX USING 2X4 AND 2X1 MUX

Q4) Design a 4-bit Even parity Checker using a multiplexer.

A4)

AIM:

To design a 4-bit Even Parity Checker using a multiplexer.

TRUTH TABLE:

EVEN PARITY CHECKER				
INPUT				OUTPUT
A	B	C	D	EVEN PARITY CHECKER
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**

DESIGN:

We will use reduced MUX to obtain the minimum number of input variables.

MINTERMS:

Select lines

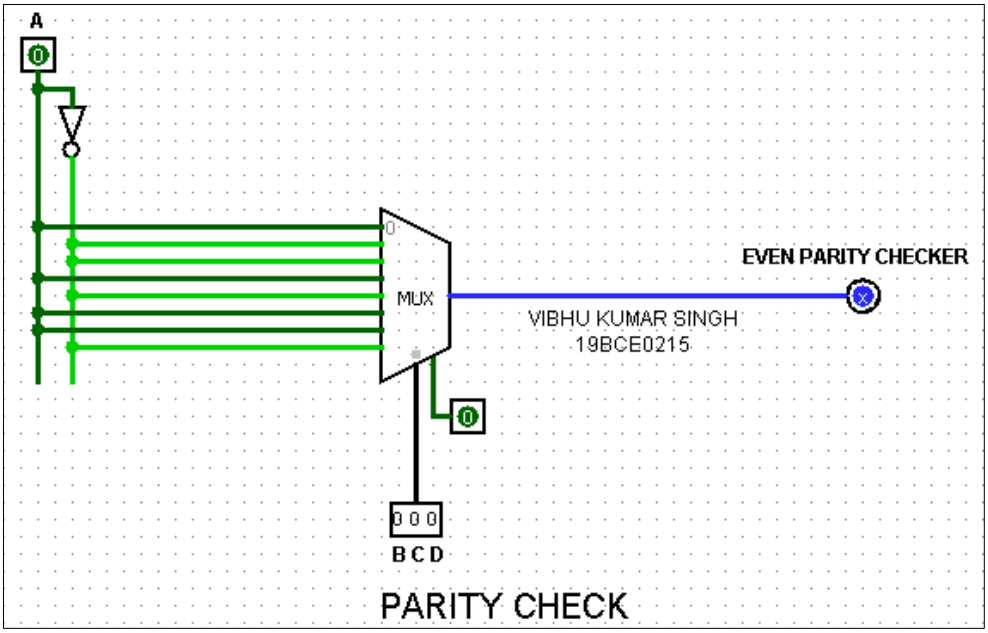
$$\text{PARITY CHECKER}(A,B,C,D)=\sum m(1,2,4,7,8,11,13,14)$$

Input variable

MAPPING MINTERMS:

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

The circuit for a 4-bit Even Parity Checker using a multiplexer is:



SCREENSHOTS(4):