VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Digital Logic Design
CSE1003 LAB

1.Digital Logic Gates
2.Boolean Algebra

Experiment – 1 & 2

Bimal Parajuli
(20BDS0405)

6/12/21

CSE 1003 - LAB

DIGITAL LOGIC DESIGN (CSE1003)

Exp # 1 - DIGITAL LOGIC GATES

OBJECTIVE:

- To study the basic logic gates: AND, OR, INVERT, NAND, and NOR.
- To study the representation of these functions by truth tables, logic diagrams and Boolean algebra.
- To observe the pulse response of logic gates.
- To measure the propagation delay of logic gates.

APPARATUS:

- IC Type 7400 Quadruple 2-input NAND gates
- IC Type 7402 Quadruple 2-input NOR gates
- IC Type 7404 Hex Inverters
- IC Type 7408 Quadruple 2-input AND gates
- IC Type 7432 Quadruple 2-input OR gates
- IC Type 7486 Quadruple 2-input XOR gate
- IC Type 7493 4-bit ripple counter
- Digi-Designer Logic Board
- Dual-trace oscilloscope

THEORY:

AND A multi-input circuit in which the output is 1 only if all inputs are 1.The symbolic representation of the AND gate is shown in Fig. 1a.

OR A multi-input circuit in which the output is 1 when any input is 1. The symbolic representation of the OR gate is shown in Fig. 1b.

INVERT The output is 0 when the input is 1, and the output is 1 when the input is 0. The symbolic representation of an inverter is shown in Fig. 1c.

NAND AND followed by INVERT. The symbolic representation of the NAND gate is shown in Fig 1d.

NOR OR followed by INVERT as shown in Fig 1e.

EX-OR The output of the Exclusive –OR gate, is 0 when it's two inputs are the same and it's output is 1 when its two inputs are different.

Truth Table Representation of the output logic levels of a logic circuit for every possible combination of levels of the inputs. This is best done by means of a systematic tabulation.

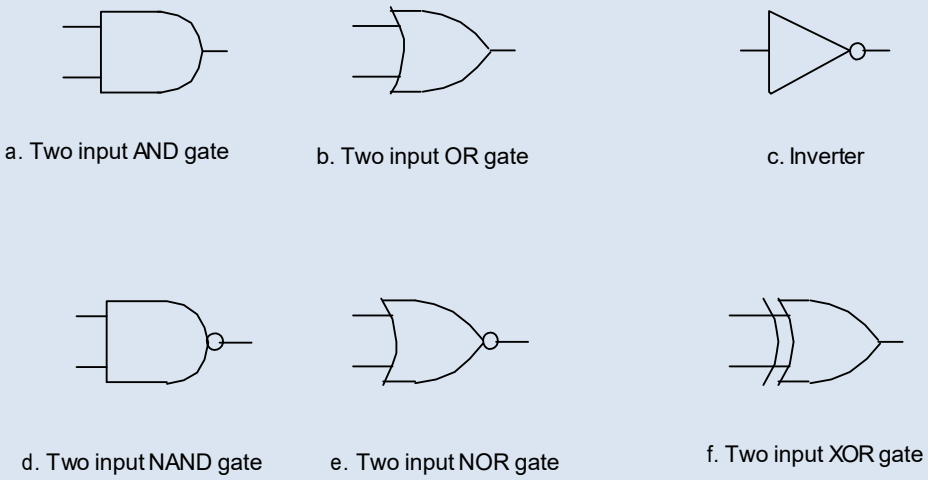
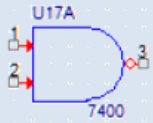


Fig.1 Symbols for digital logic gates

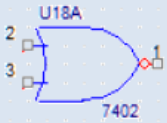
Part 1: Logic Functions

Exercises:

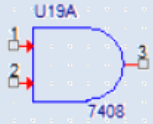
1. **Use one gate for each IC 7400 (NAND), 7402 (NOR), 7408 (AND), 7432 (OR), 7486 (XOR). Each has input pins, 1 and 2, and output pin 3.**



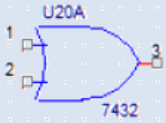
7400 (NAND gate)



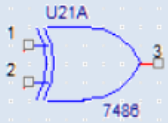
7402 (NOR gate)



7408 (AND gate)

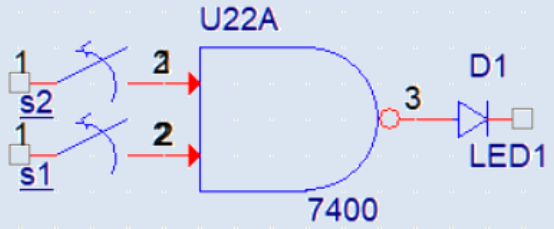


7432 (OR gate)



7486 (XOR gate)

2. **Connect pin 1 to switch S1-1, pin 2 to switch S1-2, and pin 3 to LED-1 for every gate as shown in Fig 2 as an example for the NAND gate.**



7400 (NAND gate) connected to switches and LED

3. Simulation and Truth Table of AND, OR, NAND, NOR and XOR gates:

- **AND gate**:

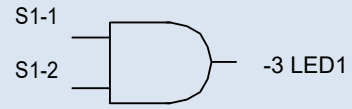
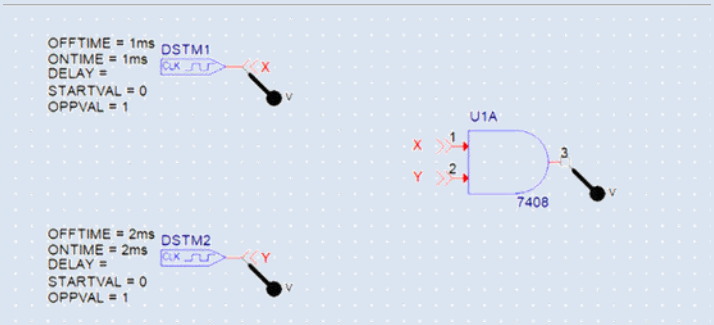
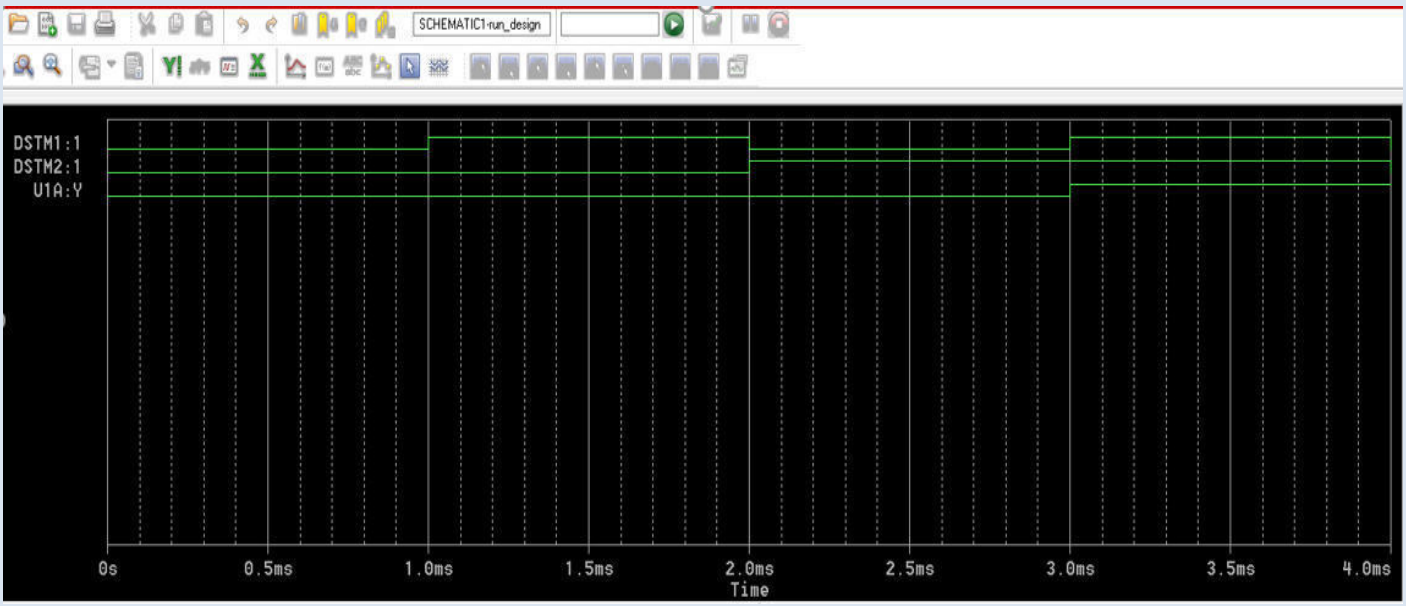


Fig.2 Two input AND gate





Observation:

Truth table of AND gate

Pin1(A)	Pin2 (B)	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

❖ **OR gate**:

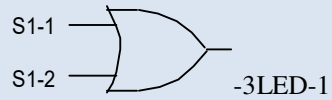
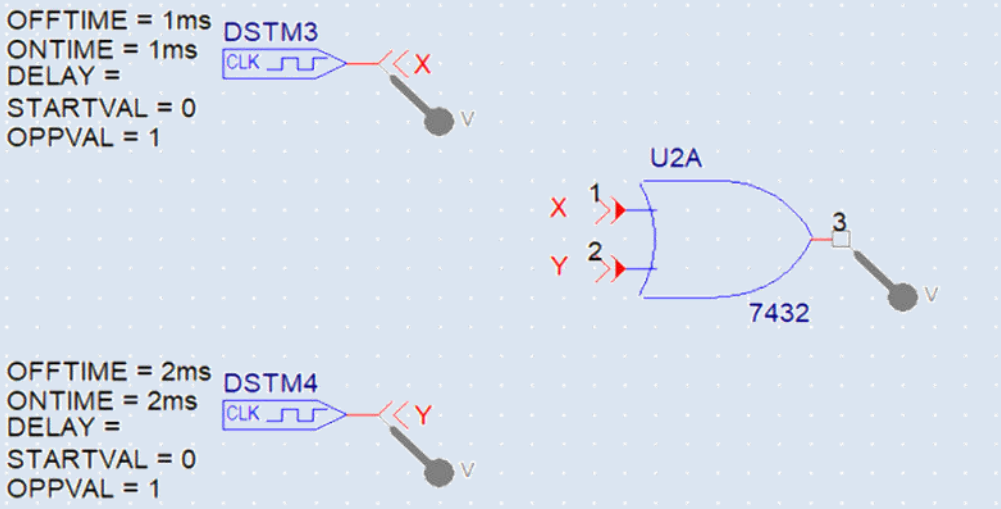
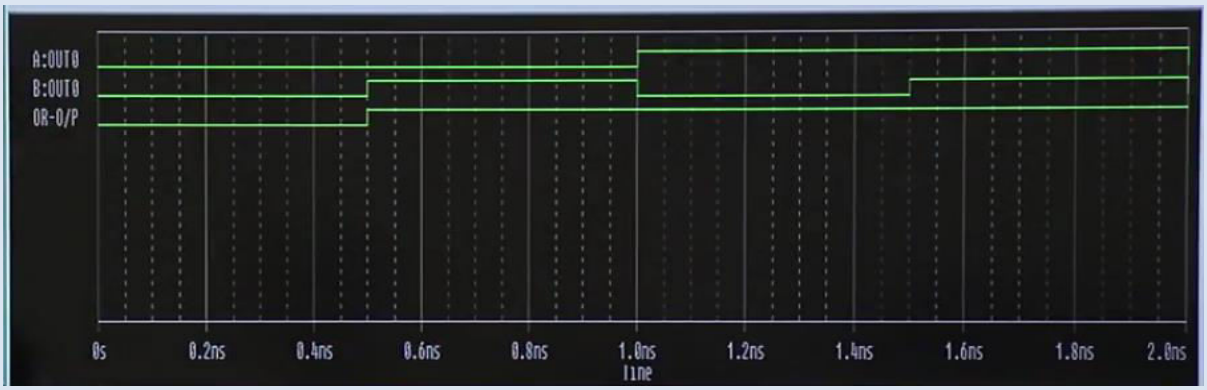


Fig.2 Two input OR gate





Observation:

Truth table of OR gate

Pin1(A)	Pin2 (B)	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

❖ **NAND gate**:

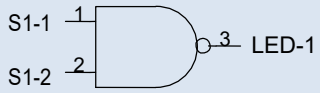
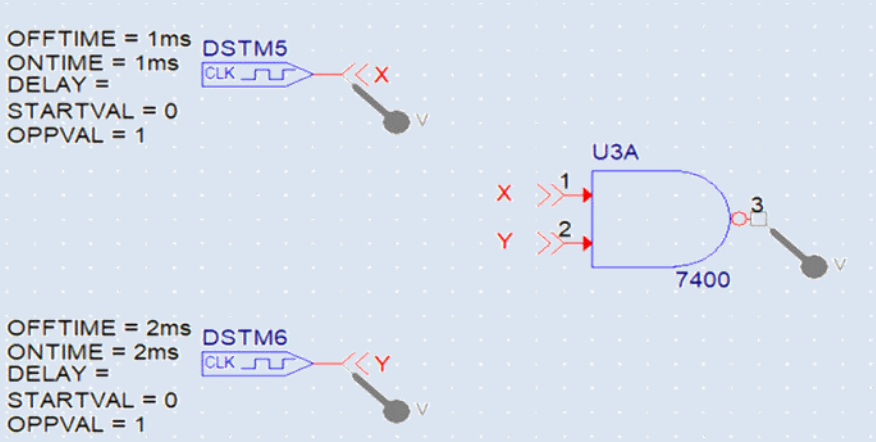
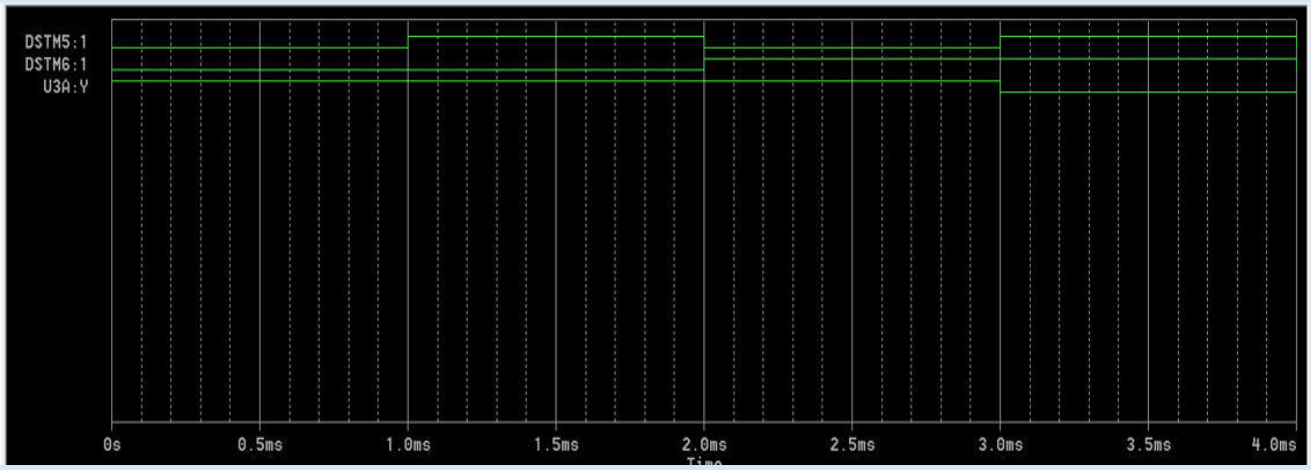


Fig.2 Two input NAND gate





Observation:

Truth table of NAND gate

Pin1(A)	Pin2 (B)	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

❖ **NOR gate:**

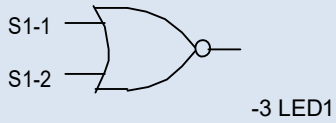
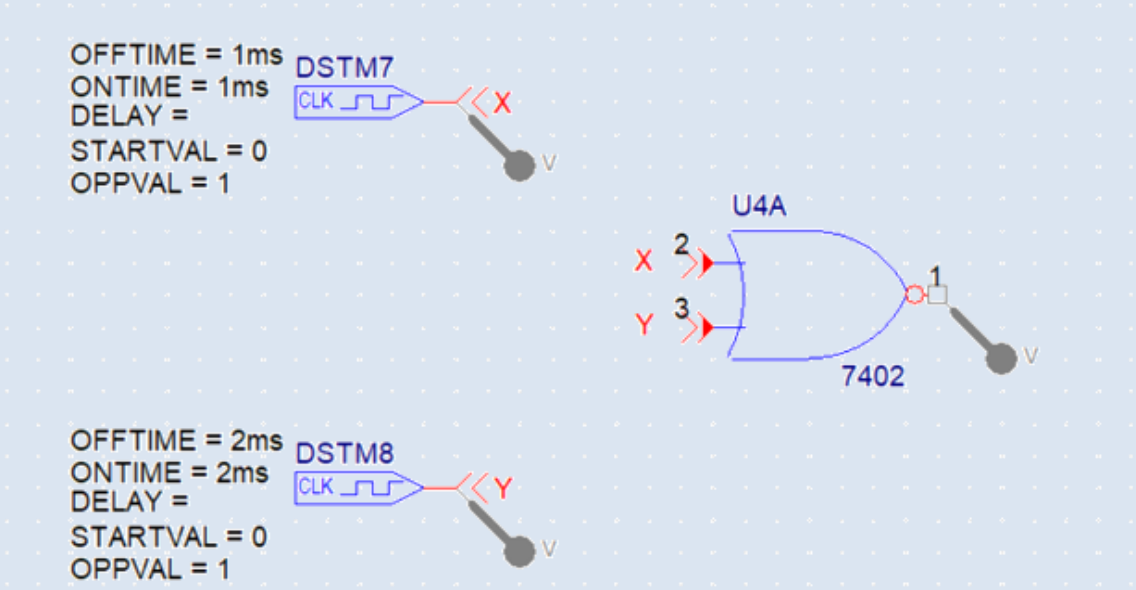


Fig.2 Two input NOR gate



Observation:

Truth table of NOR gate

Pin1(A)	Pin2 (B)	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

❖ **XOR gate:**

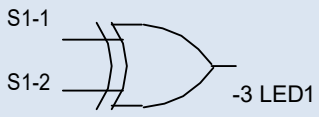
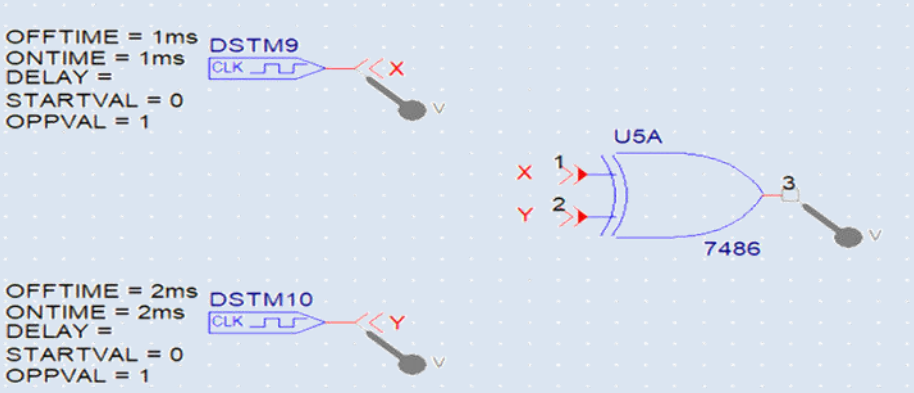
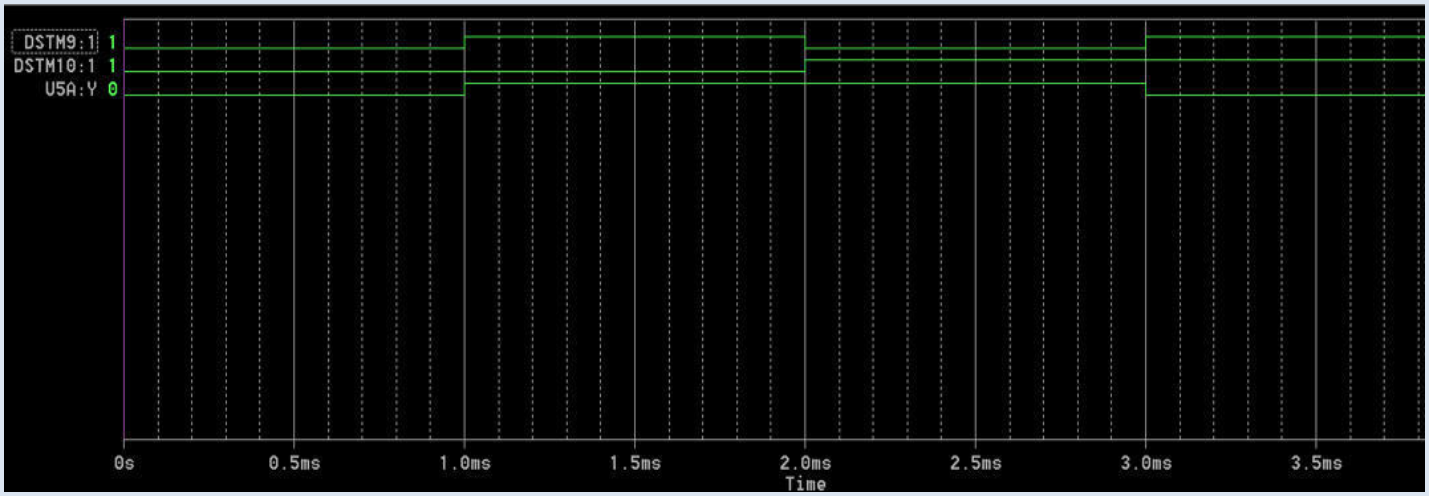


Fig.2 Two input XOR gate



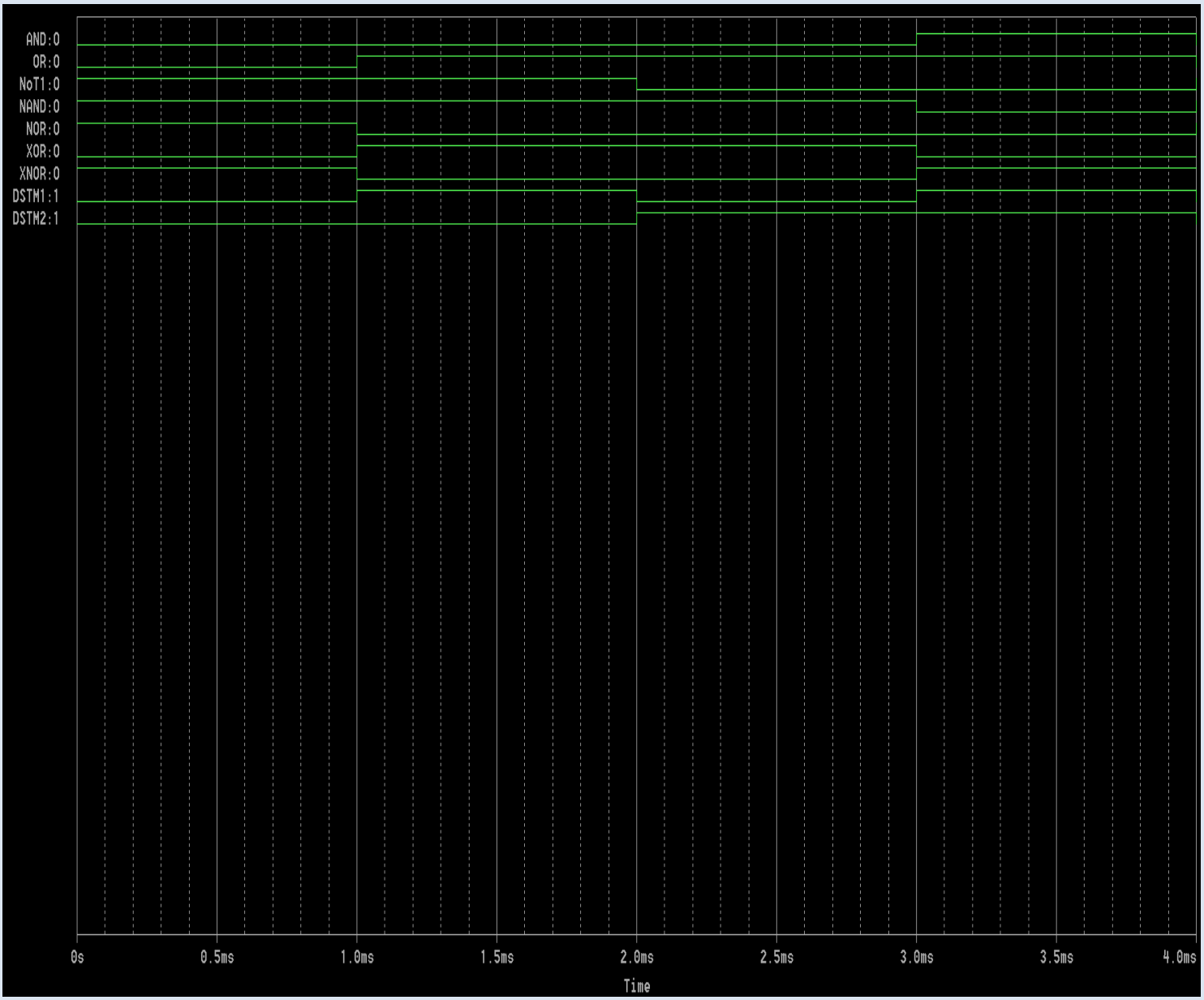


Observation:

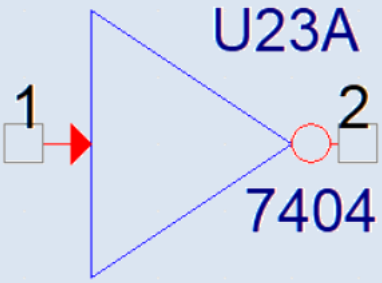
Truth table of XOR gate

Pin1(A)	Pin2 (B)	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

SIMULATION OF ALL GATES:



4. **Use an inverter gate from IC 7404 whose input pin is pin 1 and whose output pin is pin 2.**



IC7404 (NOT gate)

Observation:

Truth Table of NOT gate

Pin 1	Pin 2
0	1
1	2

Part-2: Response of Logic Gates:

Connect the circuits of figures 4 and 5 and write the corresponding truth tables 3 and 4, respectively.

A.

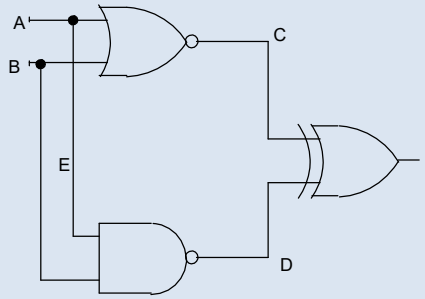
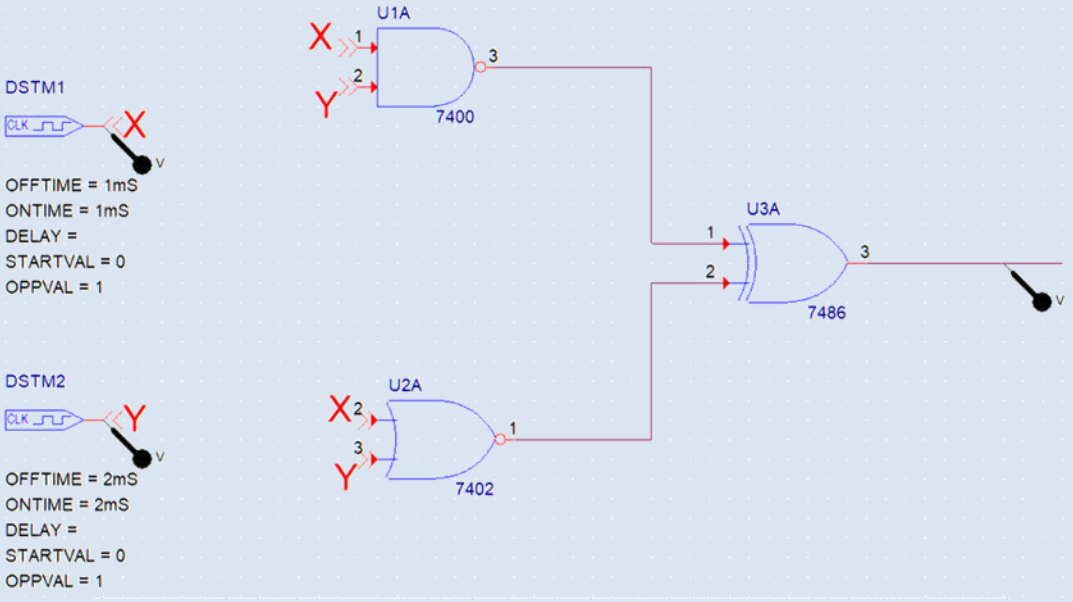
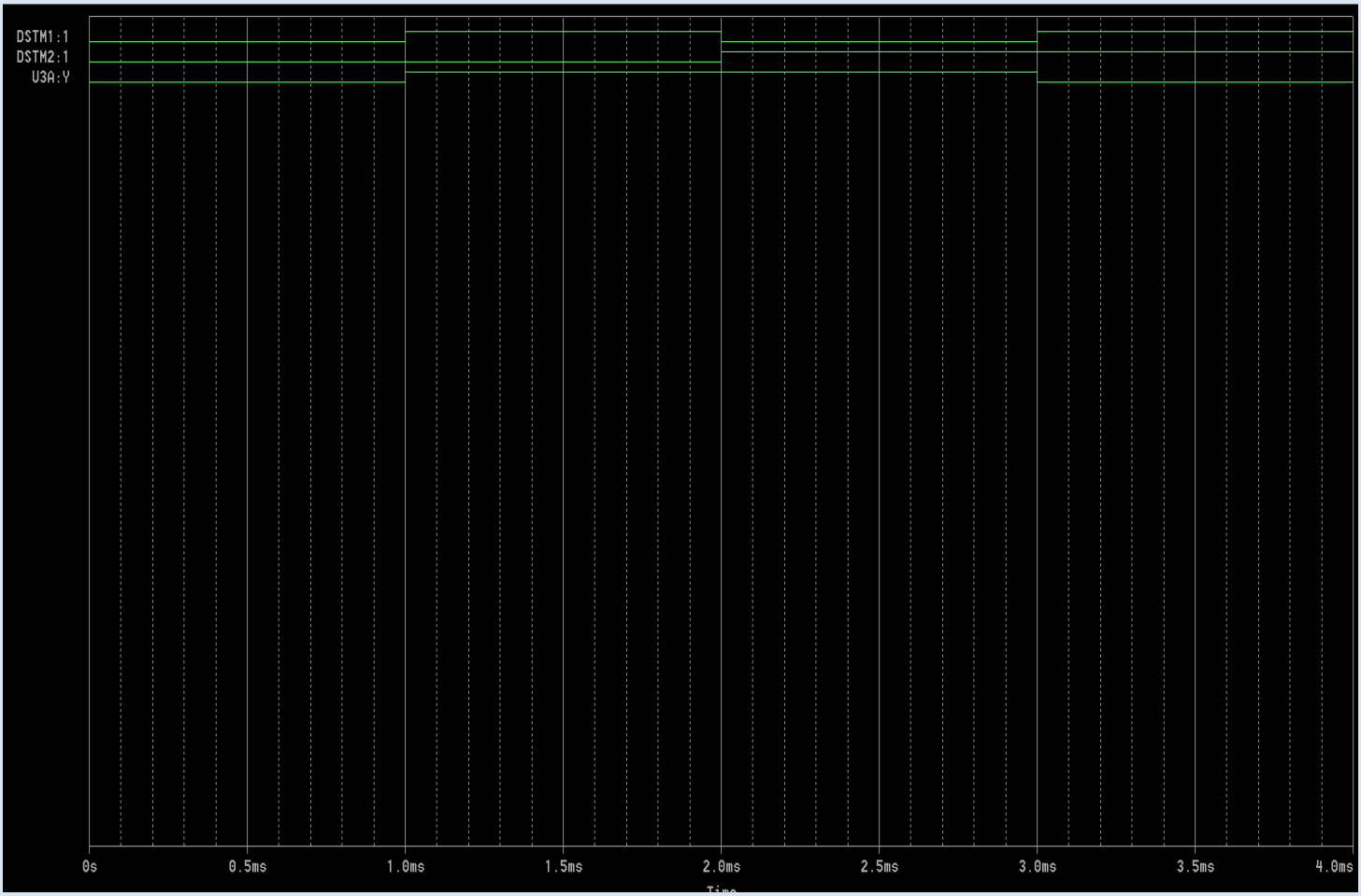


Fig. 4

Fig1:





A	B	C	D	E
0	0	1	1	0
0	1	0	1	1
1	0	0	1	1
1	1	0	0	0

B.

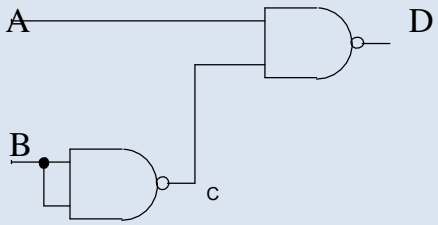
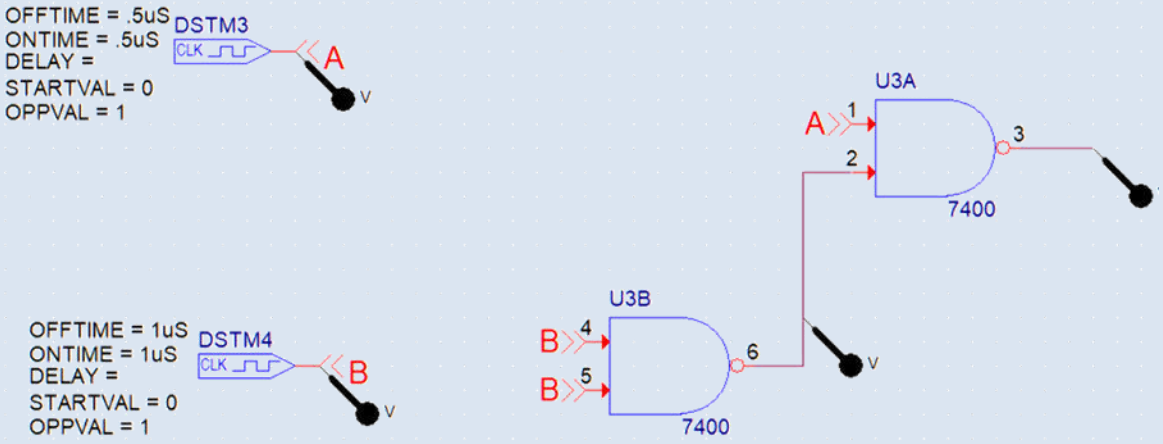
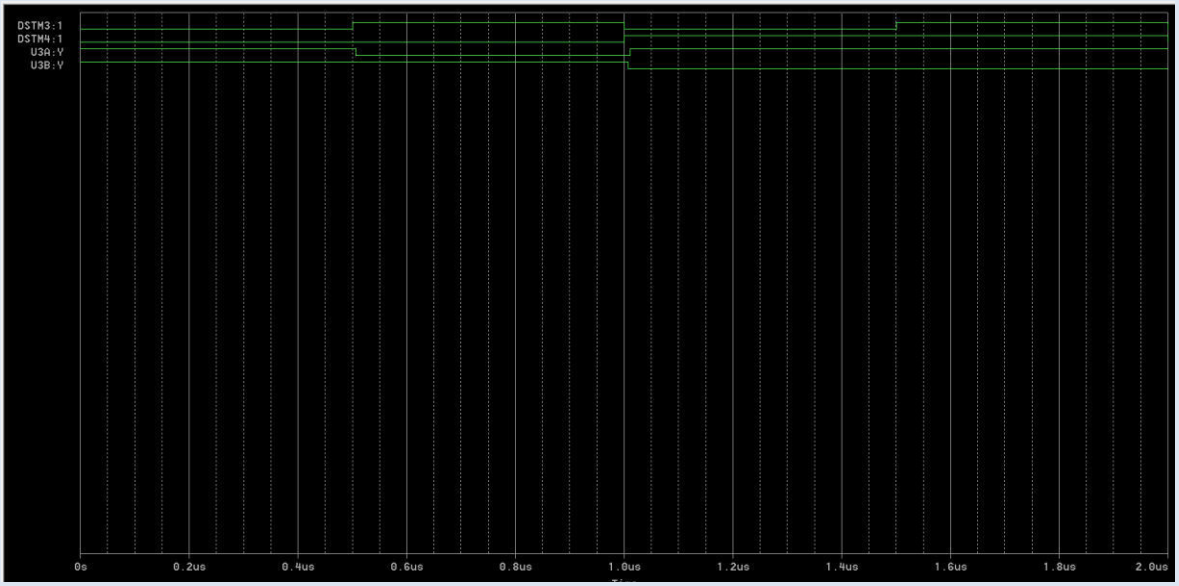


Fig: 5





Truth Table:

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

Part-3:

Propagation Delay in Logic Gates:

Connect all inverters inside two 7404 ICs in cascade. The output will be the same as the input except that it will be delayed by the time it takes the signal to propagate through all six inverters. Set S2 to 100 kHz and apply clock pulses to the input of the first inverter (connect pin 1 to j14) record the wave forms and determine the time delay from the input to the sixth inverter. This is done with a dual trace oscilloscope by applying the input clock pulses to one of the channels and the output of the sixth inverter to the second channel and measuring the delay between the two signals as shown in Fig 6. By using measured delay between two signals calculate the propagation delay for each inverter gate.

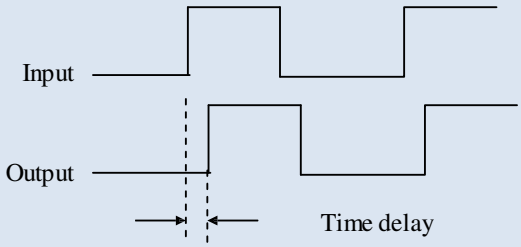
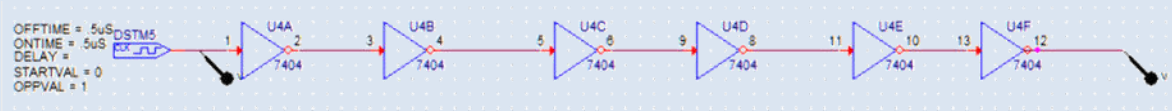
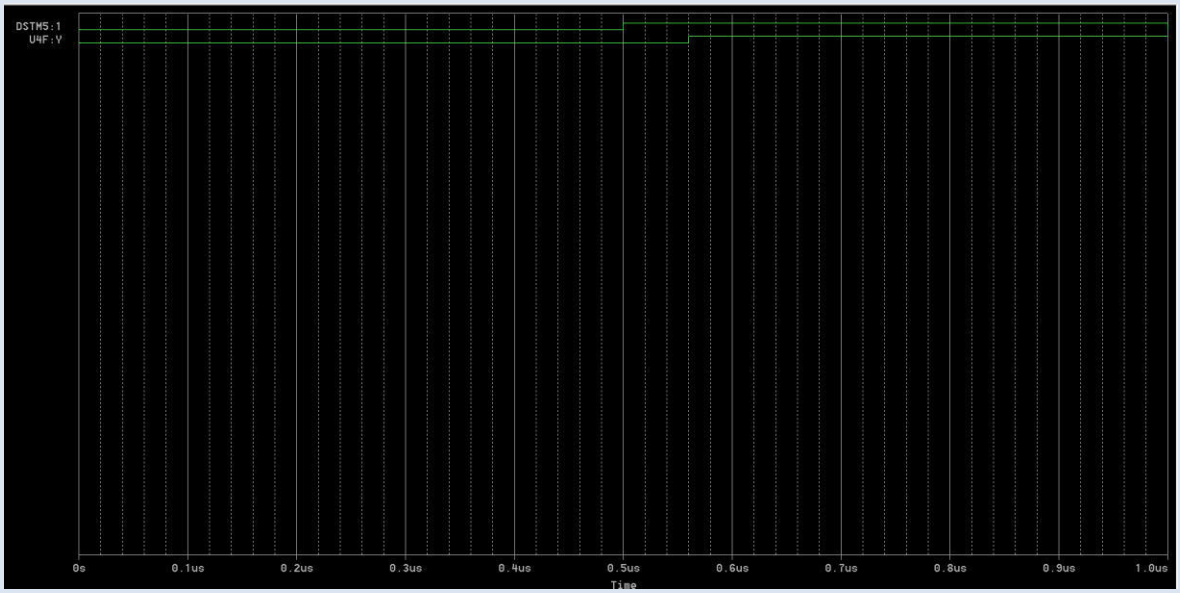


Fig. 6 Propagation delay





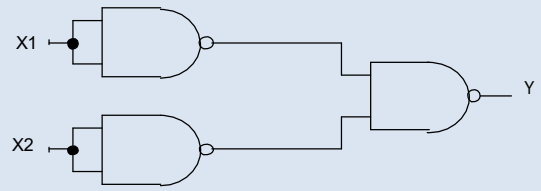
Inferences:

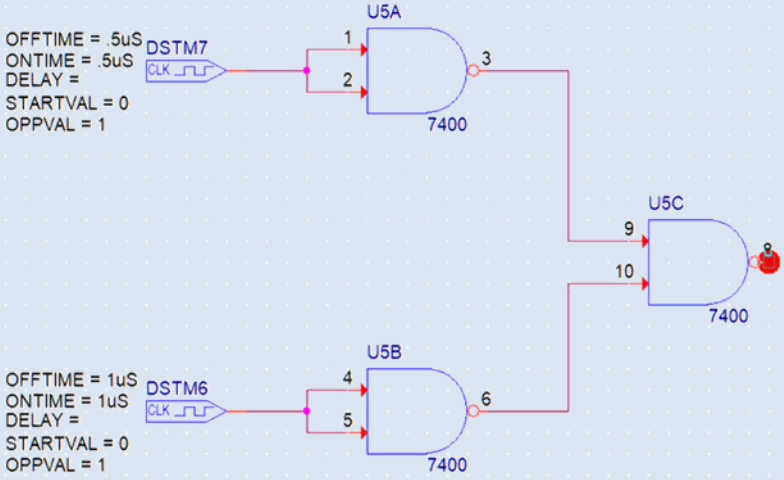
Here, we can see that the signal shifted causing a time delay of 0.06 microseconds between the input and the output signal through the series of NOT gates.

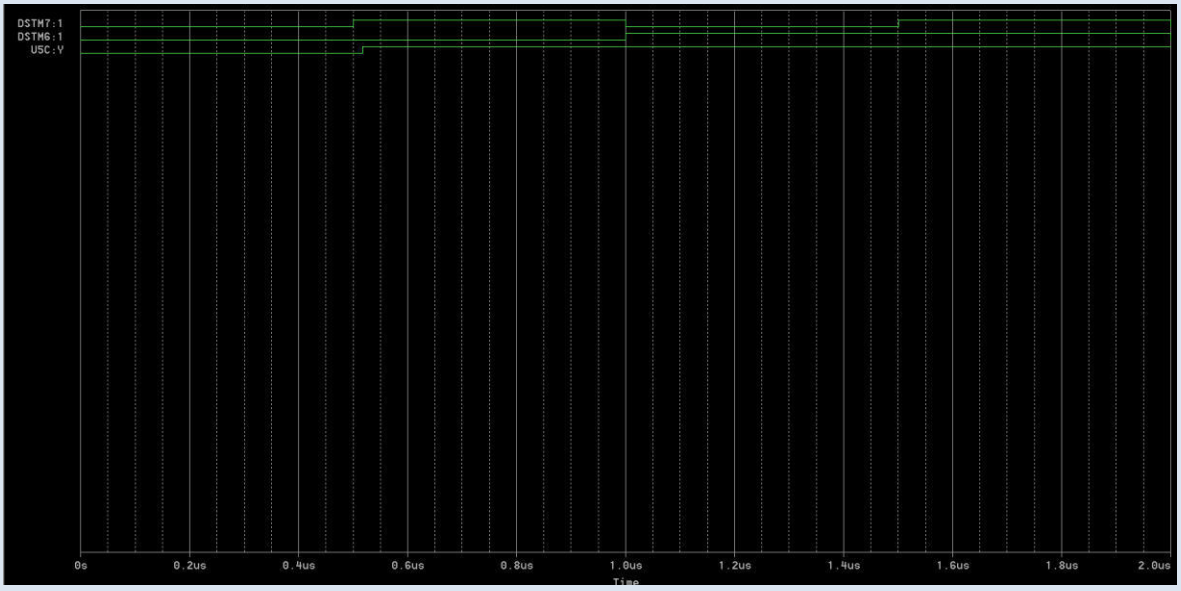
Part 4: Review Questions:

1. Write a truth table for each circuit. Derive Boolean expressions for all outputs.

A.





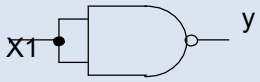


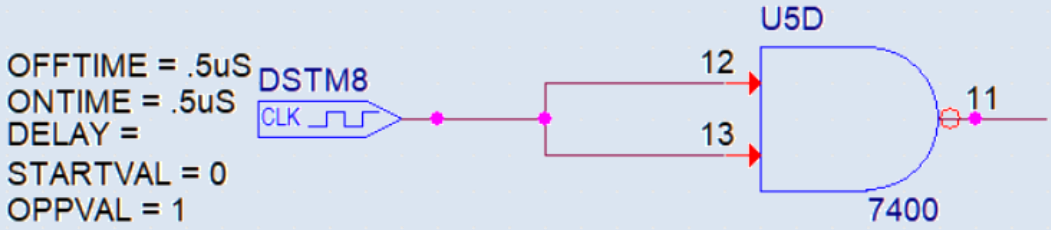
The truth table for above circuit is:

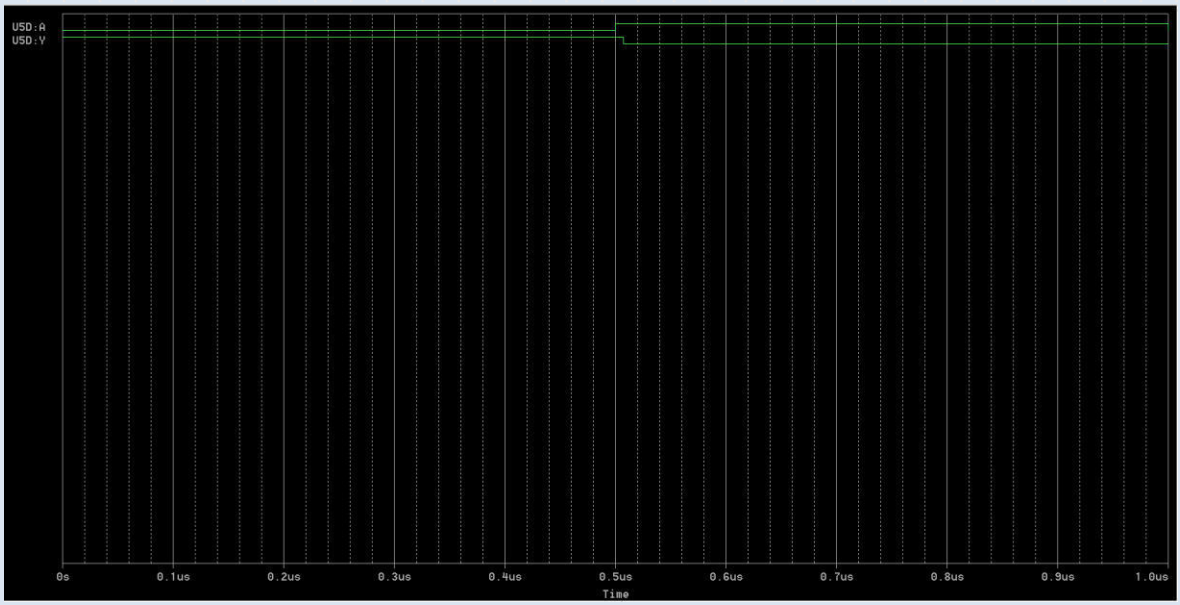
X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	1

The Boolean expression in POS (product of sums) form would be: Y = X1+ X2

B.

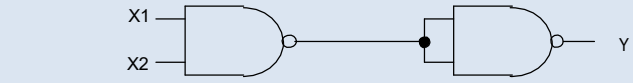


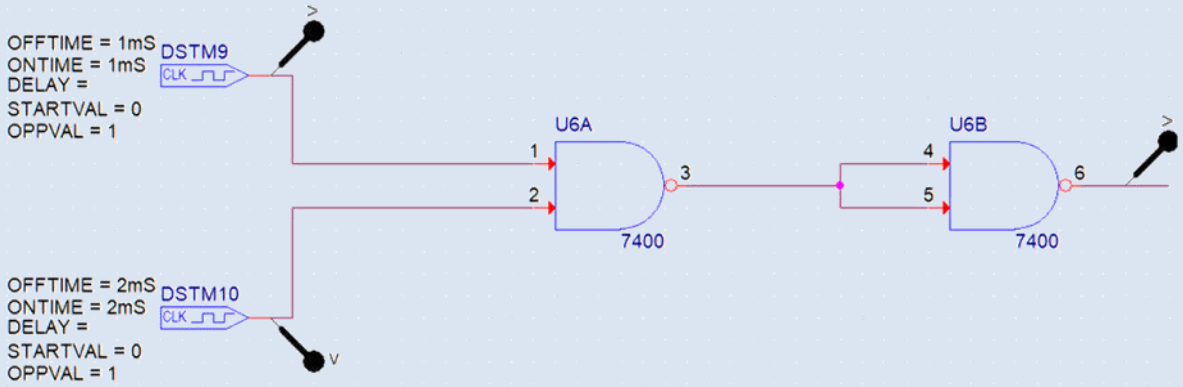


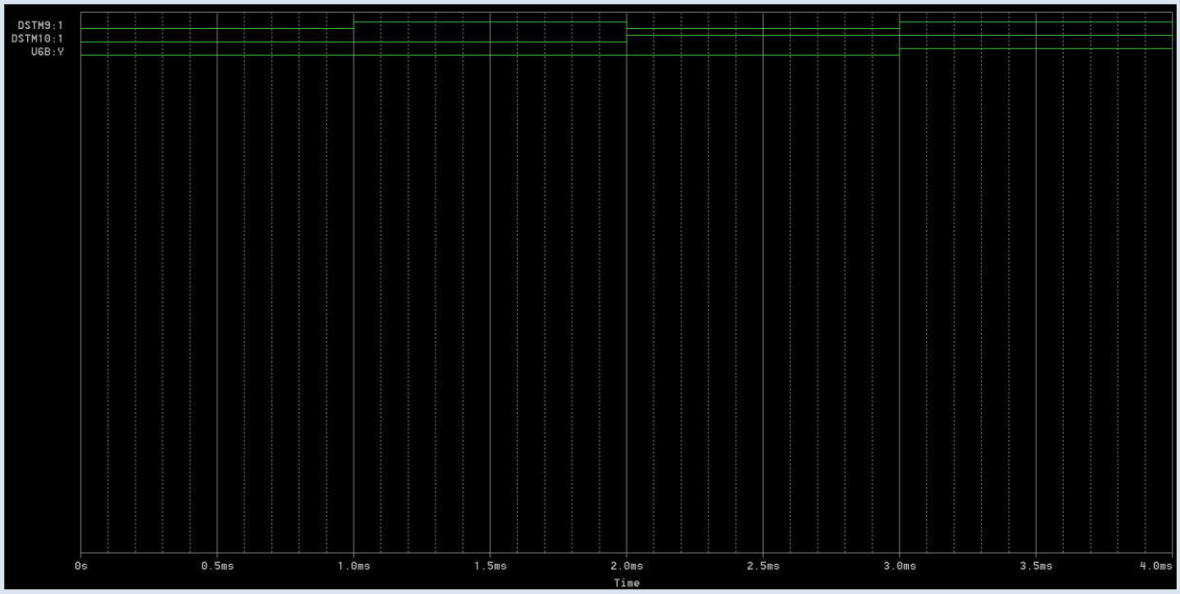


The Boolean expression for the above circuit will be:
Y = X1'

C.

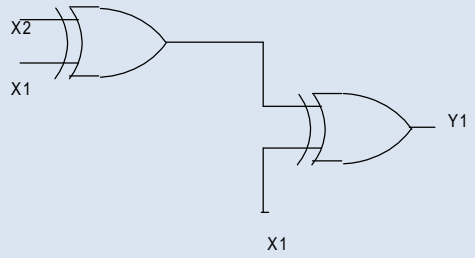


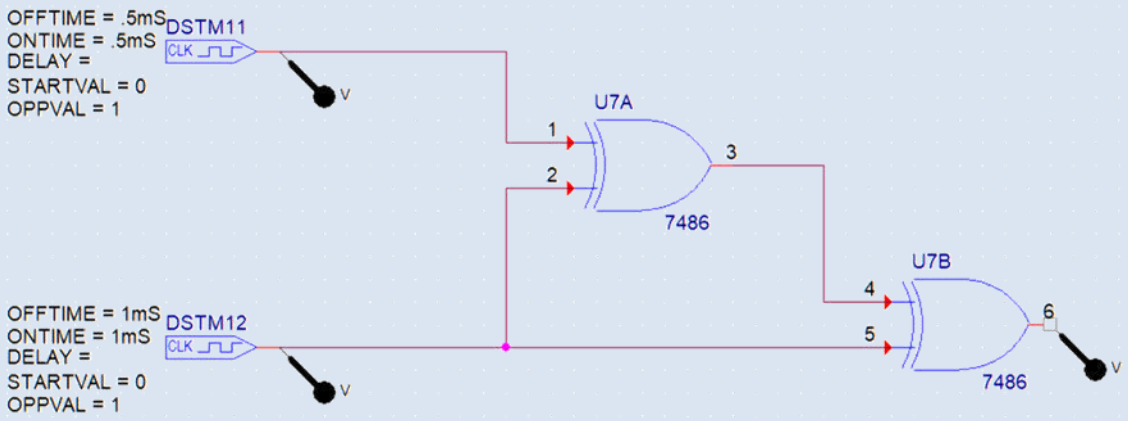


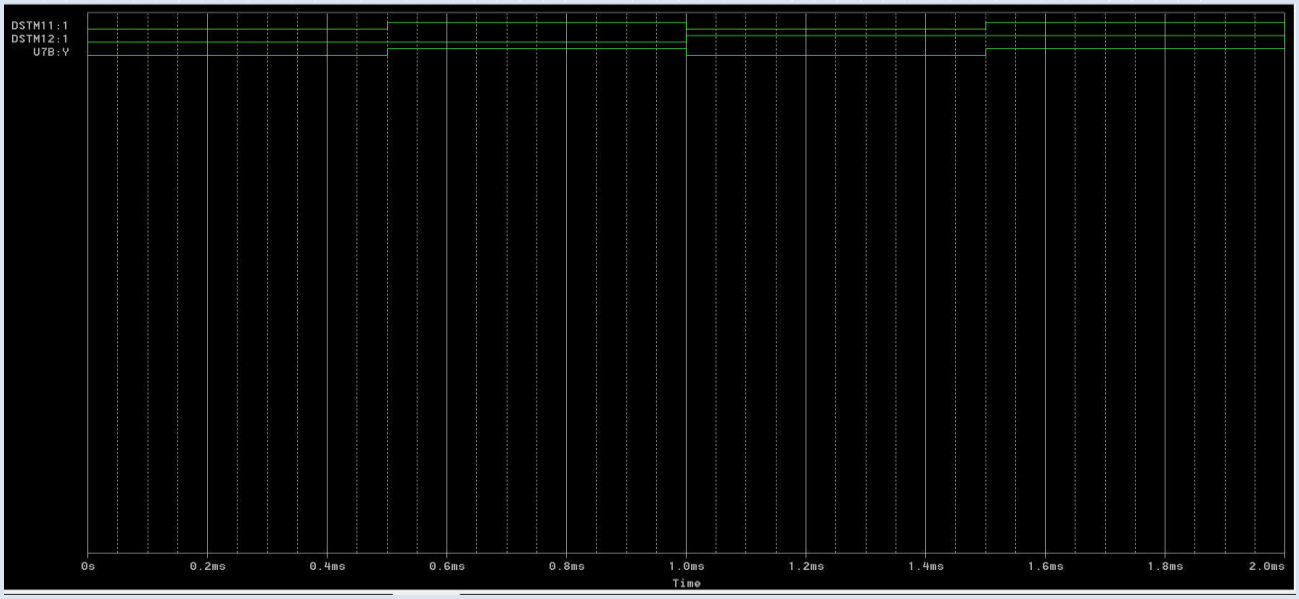


The Boolean expression of the given function is:
Y = X1 and X2

D.



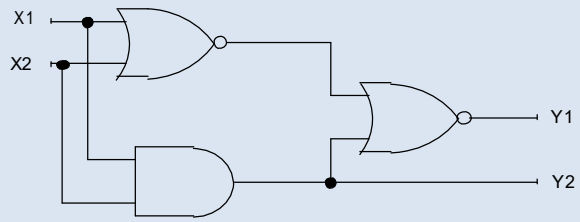


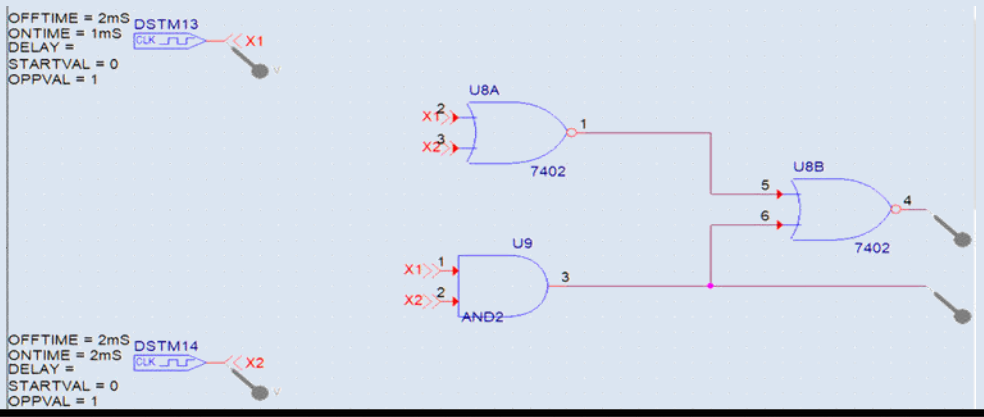


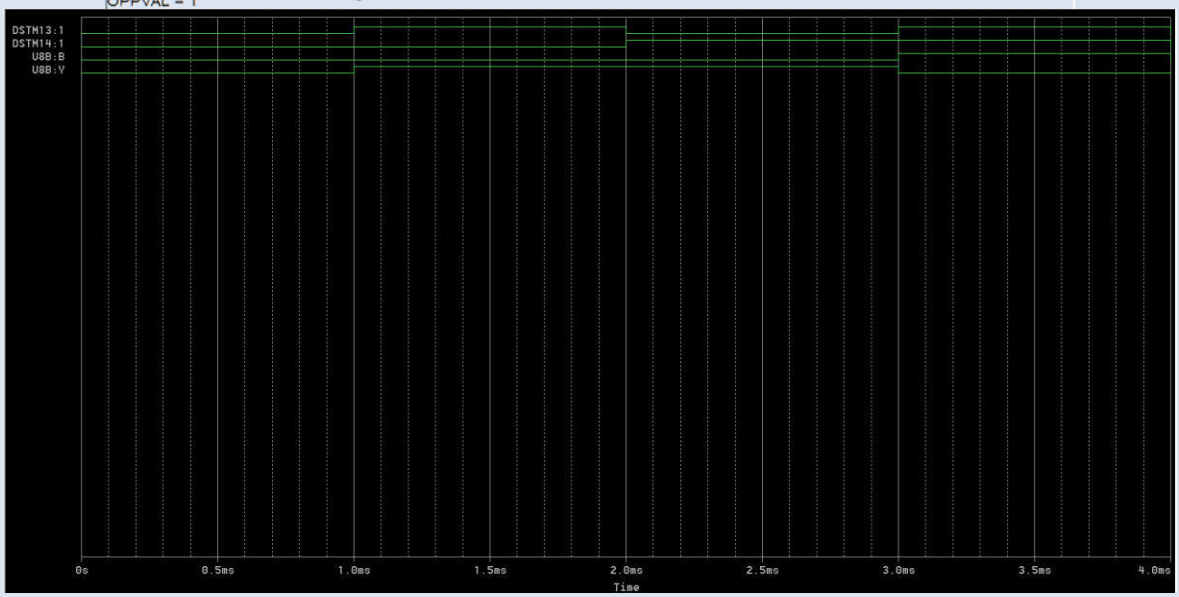
The Boolean expression is:

Y= X2

E.







The Boolean expressions of above circuit are:
Y1= X1 and X2
Y2= X1 XOR X2

2. A burglar alarm for a car has a normally low switch on each of four doors. If any door is opened the output of that switch goes HIGH. The alarm is set off with an active-LOW output signal. What type of gate will provide this logic? Support your answer with an explanation.

Answer:

For the above stated purpose of burglar's alarm, OR gate would best serve the purpose.

It is because in OR gate, the output will be high even if anyone among multiple inputs is set high. Seeing this in context of Car's Burglar alarm, the alarm should go high if at least one or many among the four doors are closed.

The truth table of OR gate for 4 inputs is:

A	B	C	D	A +B+C+D
0	0	0	0	0
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	1

Clearly, it can be seen that the output is high for all cases except when all inputs are low. Thus, the alarm will stay quiet when all doors are closed. For any other condition, it will make noise.

Hence, OR gate is best fit for the alarm.

DIGITAL LOGIC DESIGN

Exp #2 - BOOLEAN ALGEBRA

OBJECTIVE:

- To verify the rules and regulations of Boolean Algebra
- To simplify and modify Boolean logic functions by means of Demorgan's theorem.
- To design and implement a logic circuit.

APPARATUS:

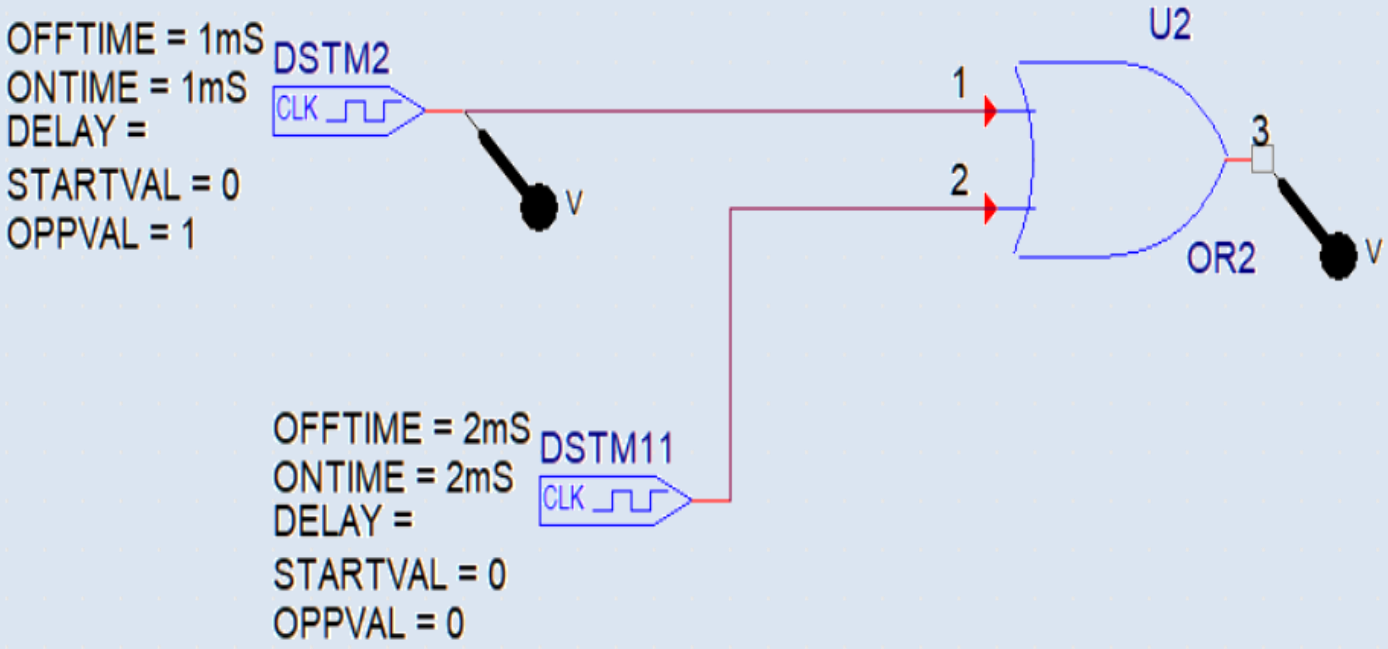
- PB-503
- 7400 Quadruple 2 input NAND gates.
- 7402 Quadruple 2 input NOR gates
- 7408 Quadruple 2 input AND gates
- 7432 Quadruple 2 input OR gates
- 7404 Hex inverters
- 7411 Triple 3-input AND gate

THEORY:

1. A+0 = A
2. A+1 = 1
3. A .0 = 0
4. A .1 = A
5. A+A = A
6. A+A' = 1
7. A.A = A
8. A.A' = 0
9. (A')' = A
10. A+AB = A
11. A+A'B = A+B
12. (A+B). (A+C) = A+BC
13. A'. B' = (A+B)'
14. A'+B' = (A.B)'

Procedure 1:

a. Prove rule 1 using OrCAD.



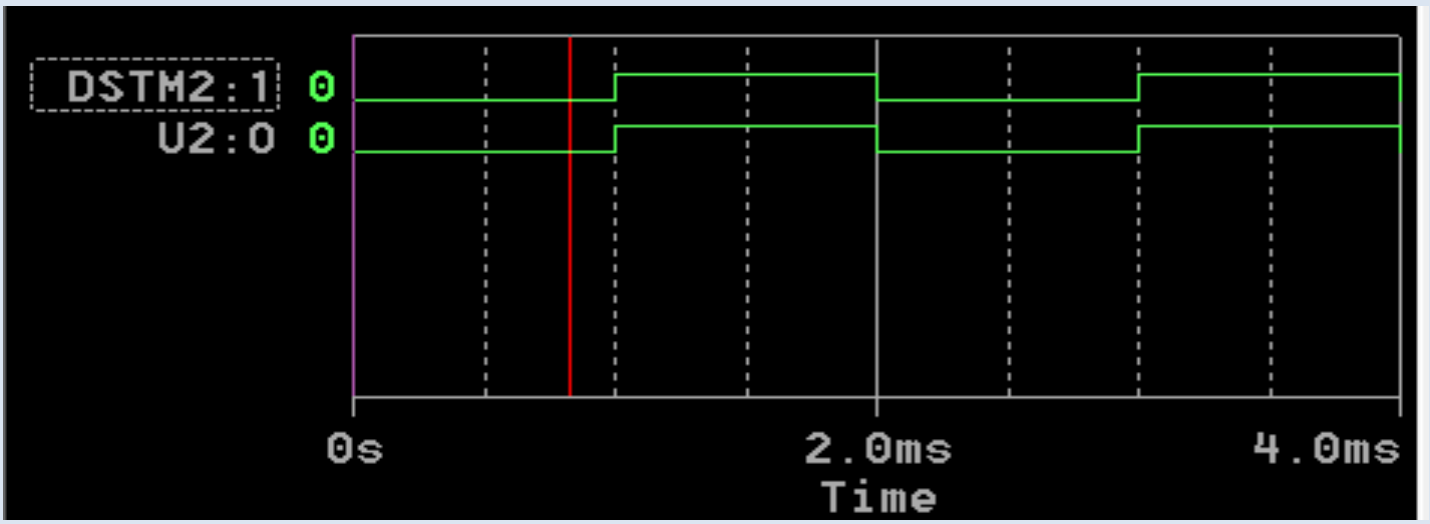


Fig.1 Verifying Rule 1

b. Connect the circuit of Fig.2 Using OrCAD. Which rule does this circuit illustrate?

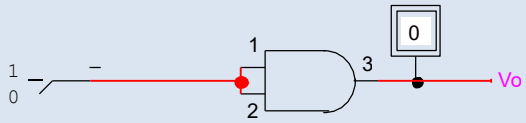
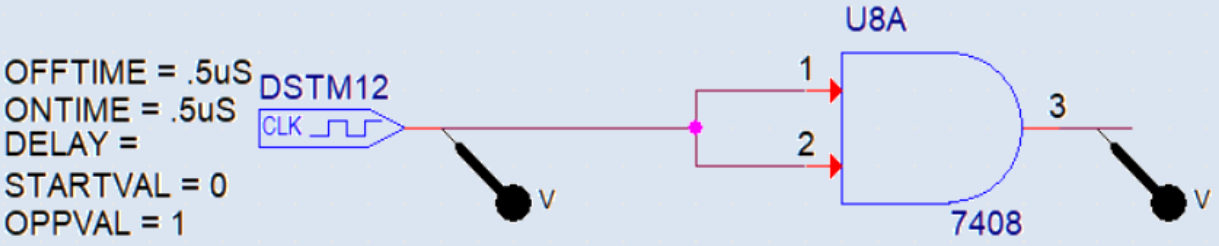
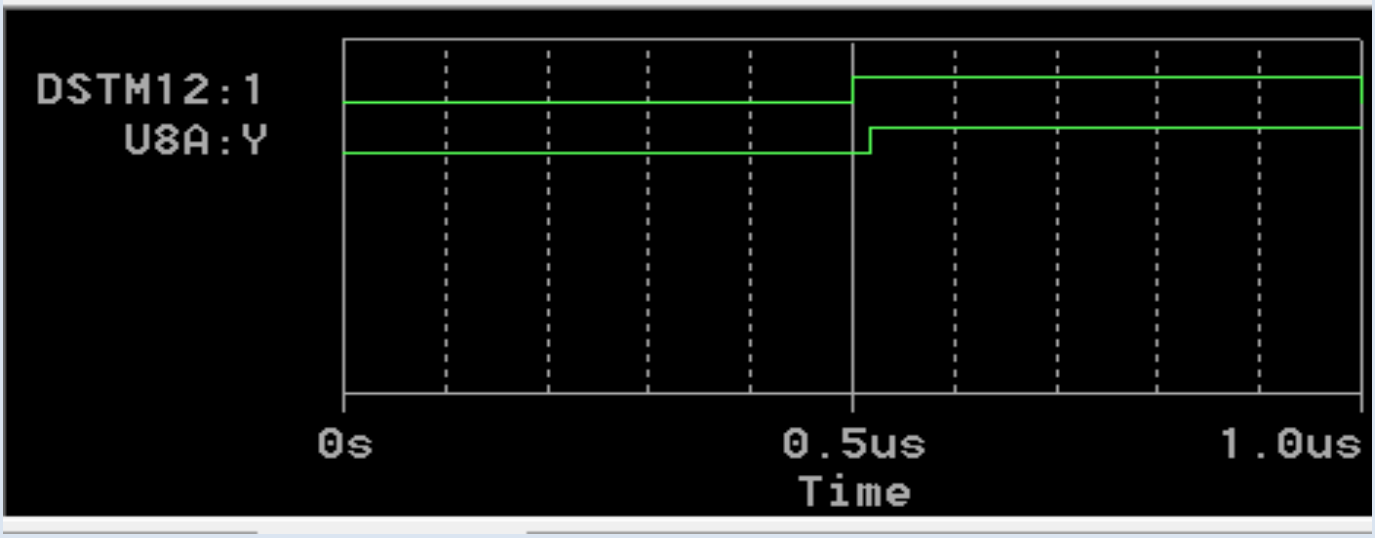


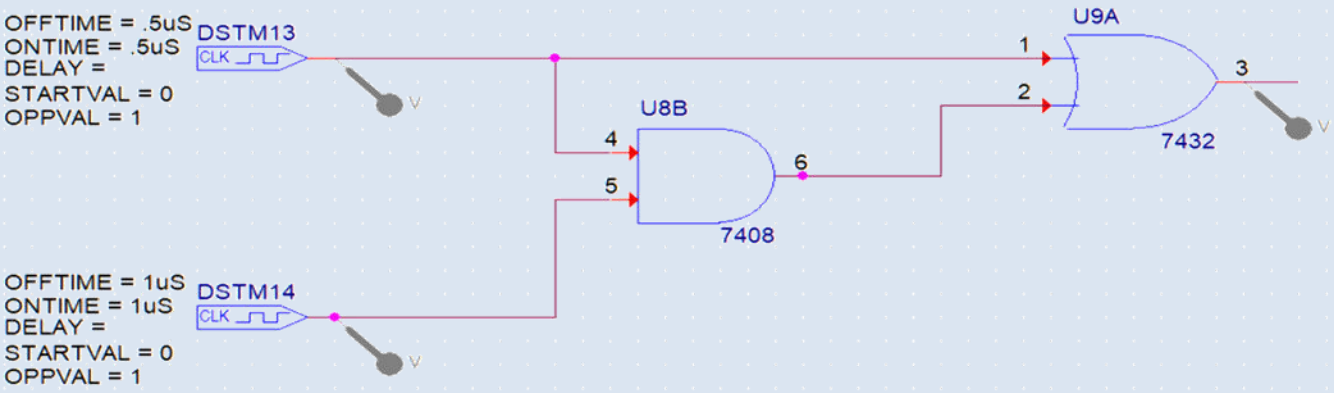
Fig.2

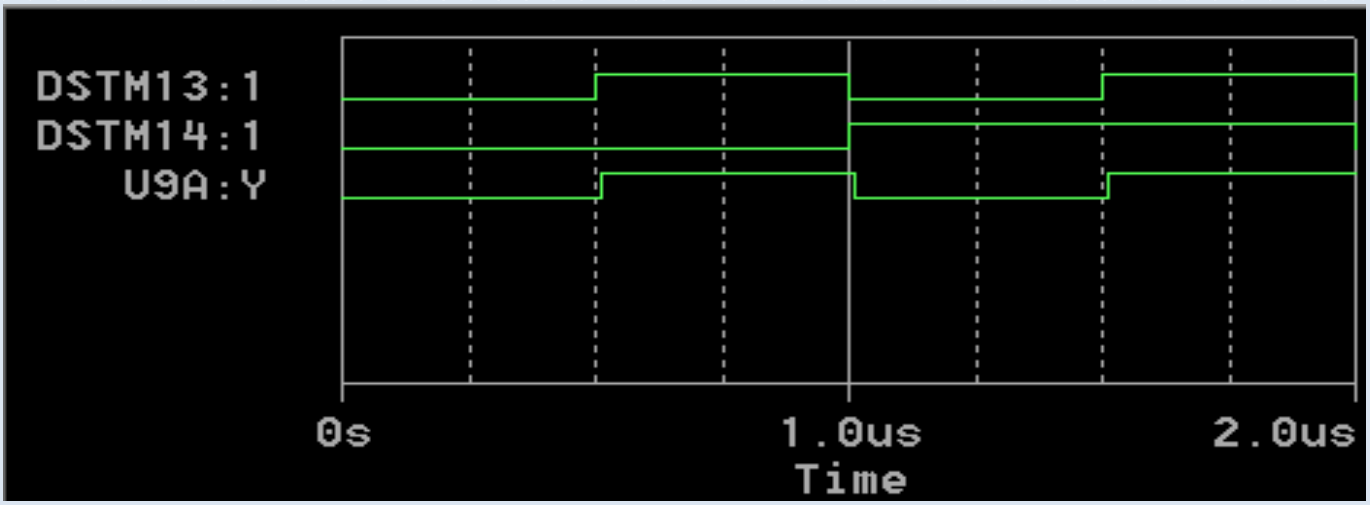




The above circuit illustrates **Idempotent Law** i.e. **A . A = A**

c. Design a circuit that illustrates rule 10. Copy the circuit from OrCAD and paste it in your lab report.





d. Rule 6 illustrates that A+A' could be replaced with a wire to Vcc. What does rule 8 illustrate?

Ans:

Rule 8 illustrates that **A.A'** can be represented with a break in the line or with a wire directly connected to the ground voltage (**GND**) because the output of **A.A'** is **Always Low.**

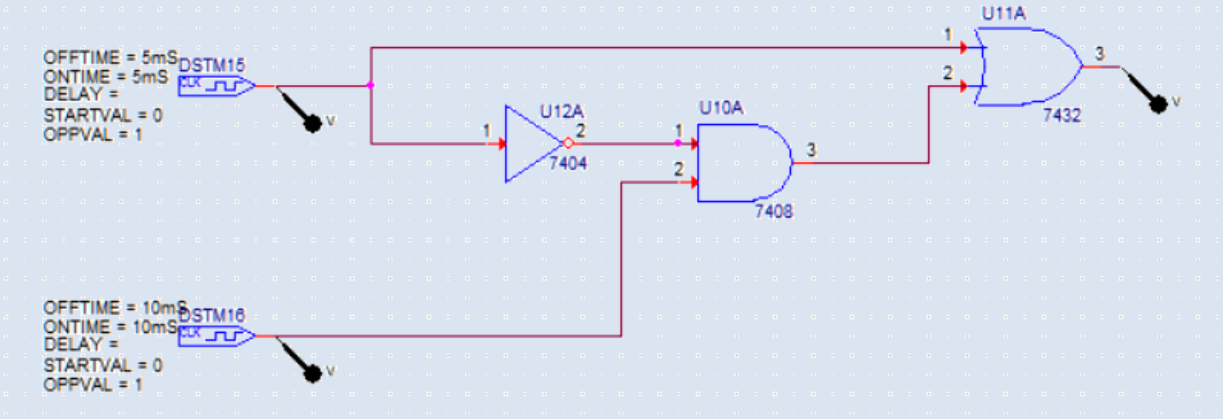
e. Rule 11 states that A+A'B = A+B. Using OrCAD design a circuit that illustrates each of these expressions.

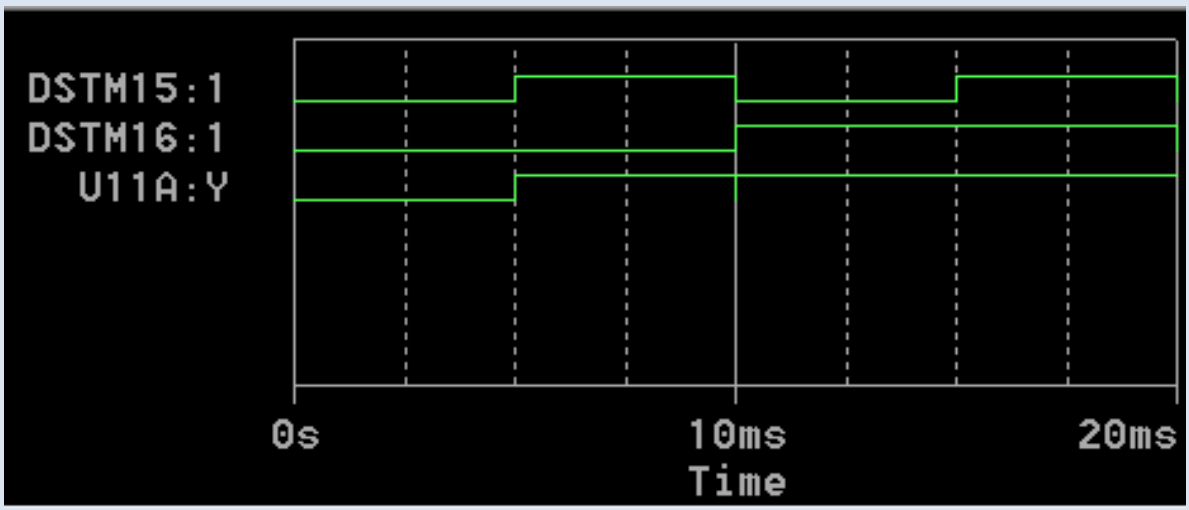
A+A'B

A+B

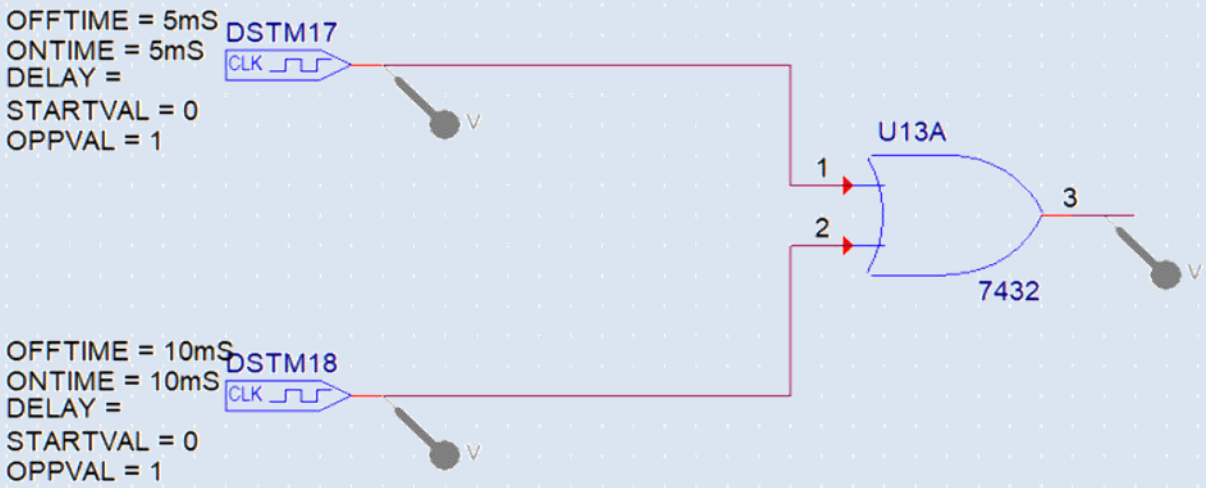
Prove that these two circuits perform equivalent logic. (Connect two circuits and show that their outputs are the same).

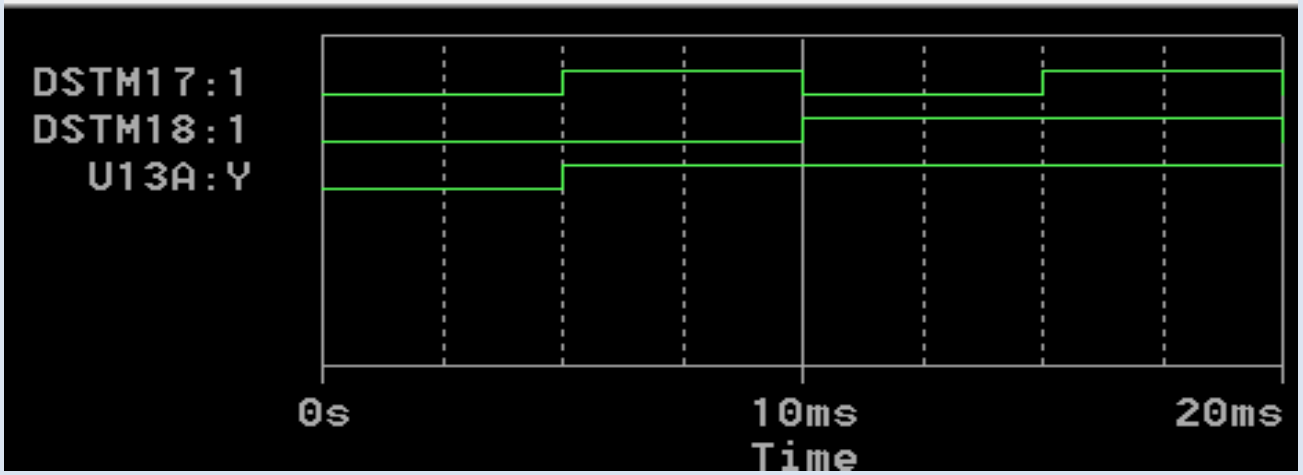
For the first circuit, **A + A'B** :





For the second circuit, **A + B**:





From the simulation, we can observe that the above circuits are logically equivalent as both of them give identical logical plots as output of simulation.

Hence, **A + A'B = A + B.**

Thus, Rule 11 is proved

Procedure 2: Demorgan's Theorem

Proof of equation (1)

Using OrCAD Cadence Capture construct the two circuits given in Figs.3 and 4 corresponding to the functions A'. B' and (A+B)' respectively.

Show that for all combinations of A and B, the two circuits give identical results.

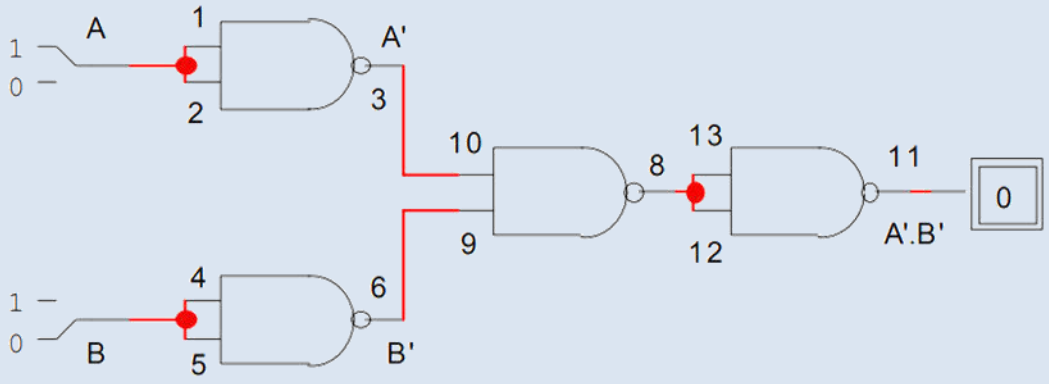


Fig.3

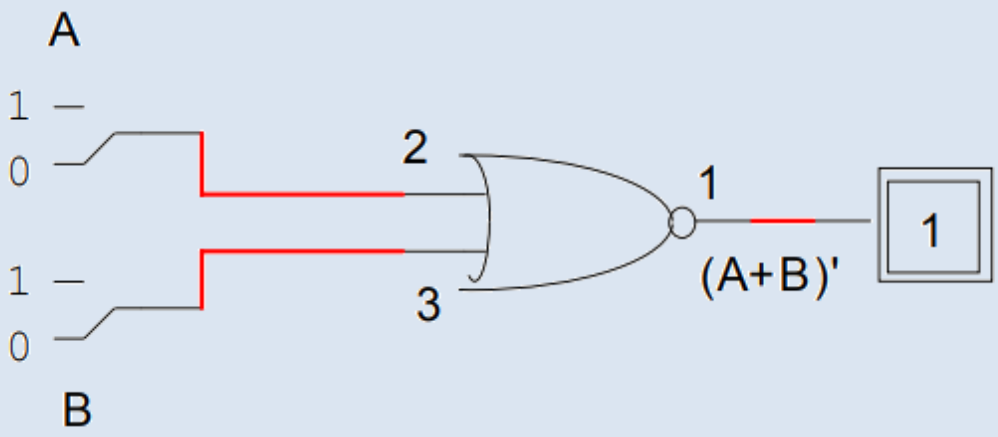
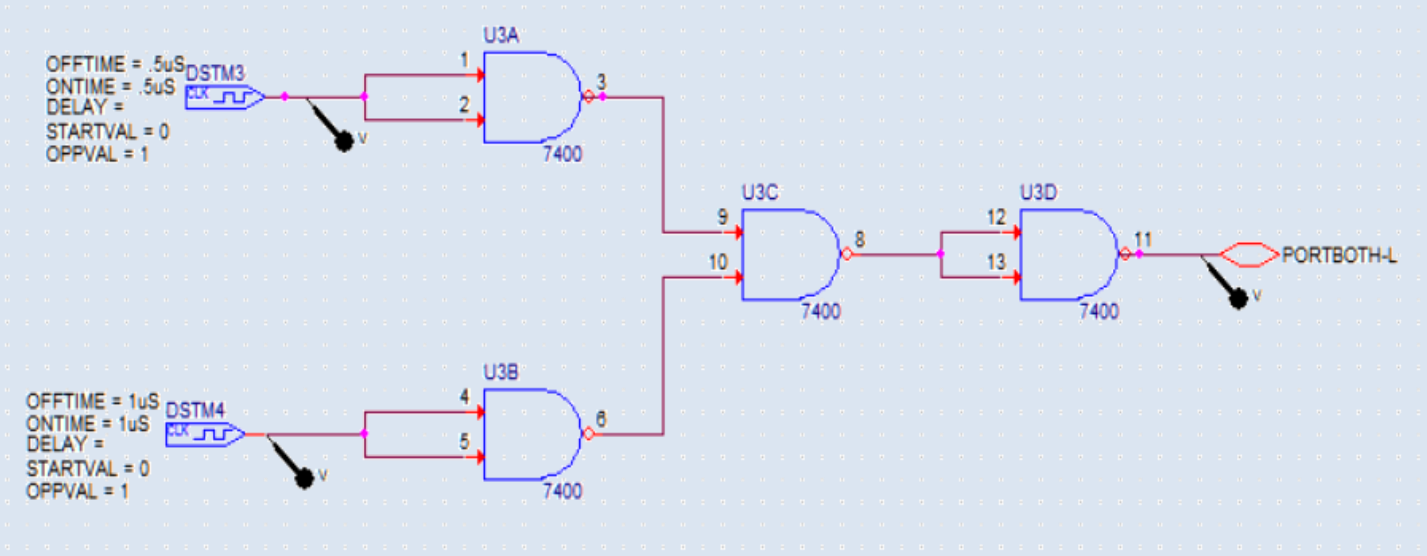
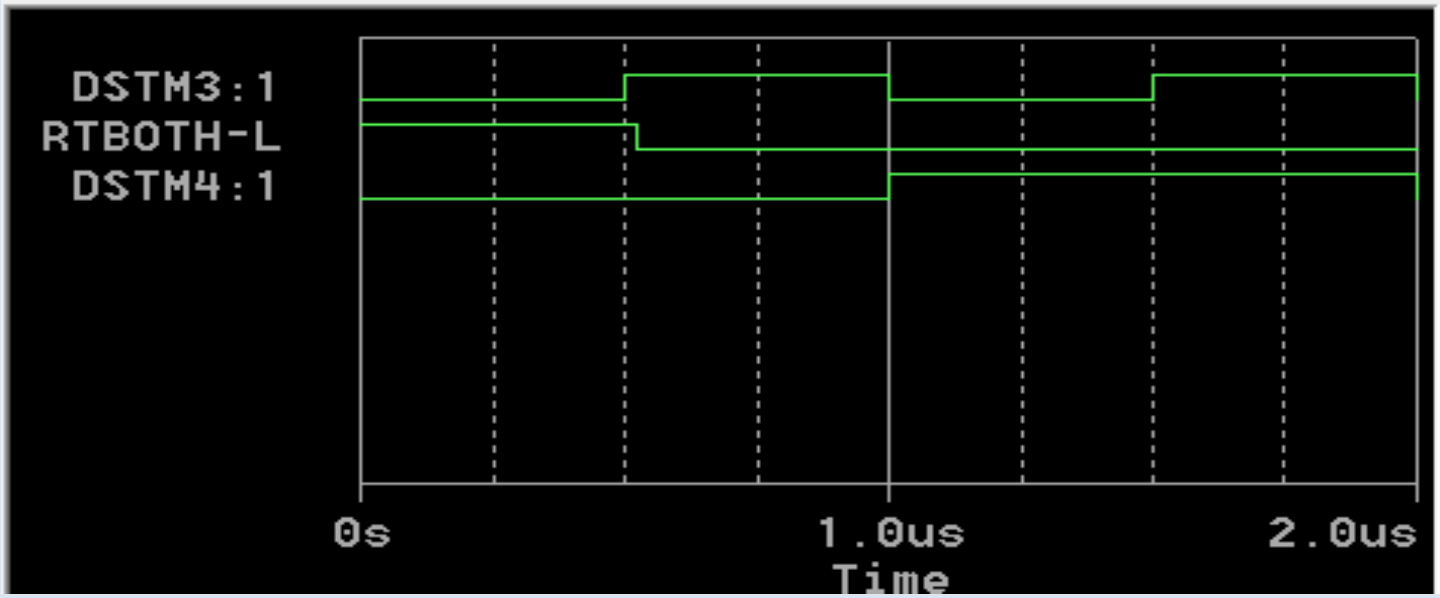


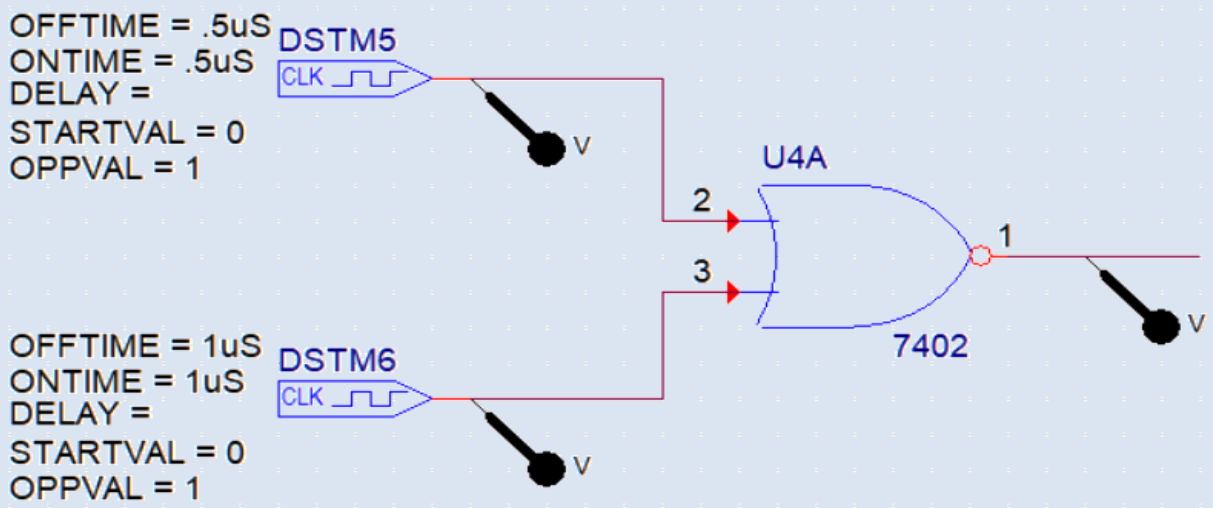
Fig.4

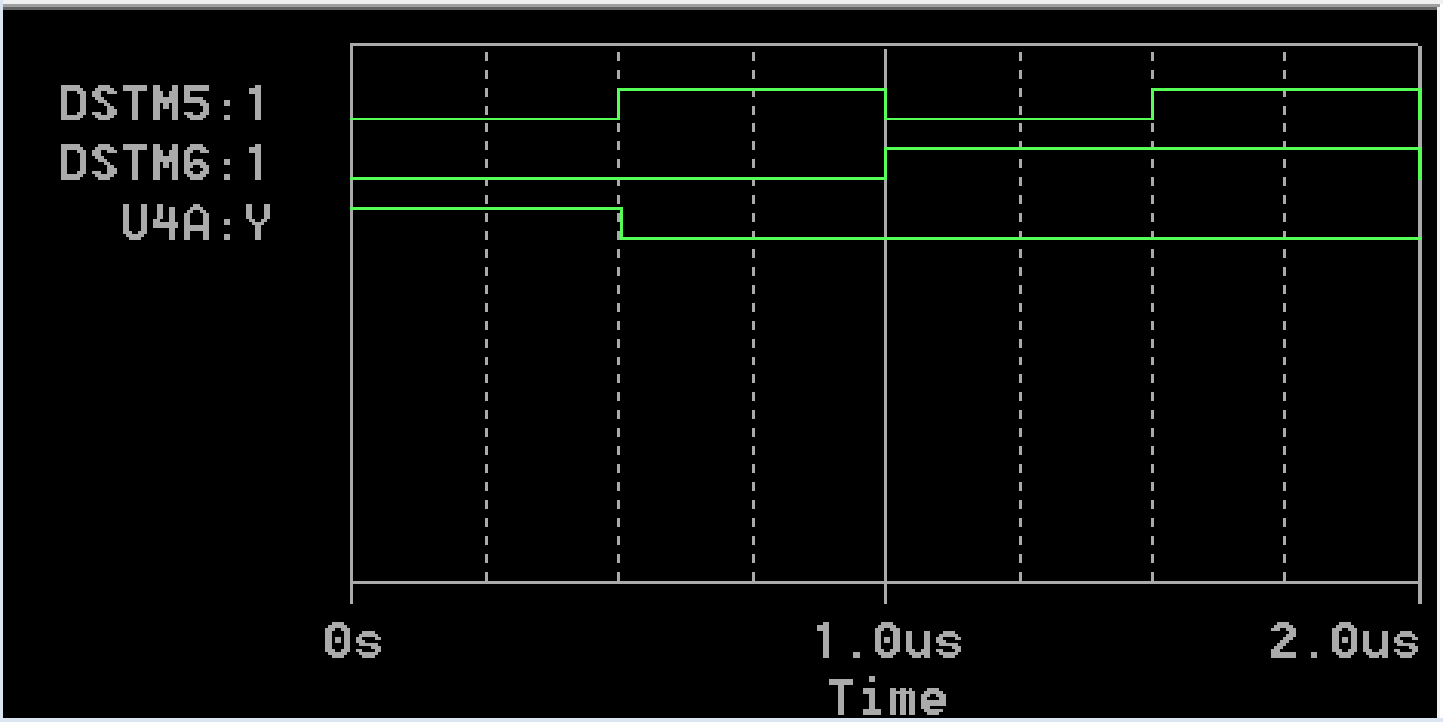
Circuit Analysis for fig 3:





Circuit Analysis for Fig 4:





Here, from the result obtained by simulation, it is clearly visible that both the gates are logically equivalent.
Thus, Demorgan's first law is verified.

Proof of equation (2)

Using OrCAD, construct two circuits given in Figs. 5 and 6, corresponding to

the functions A'+B' and (A.B)' A.B, respectively.

Show that, for all combinations of A and B, the two circuits give identical results. In the lab connect these circuits and verify their operations.

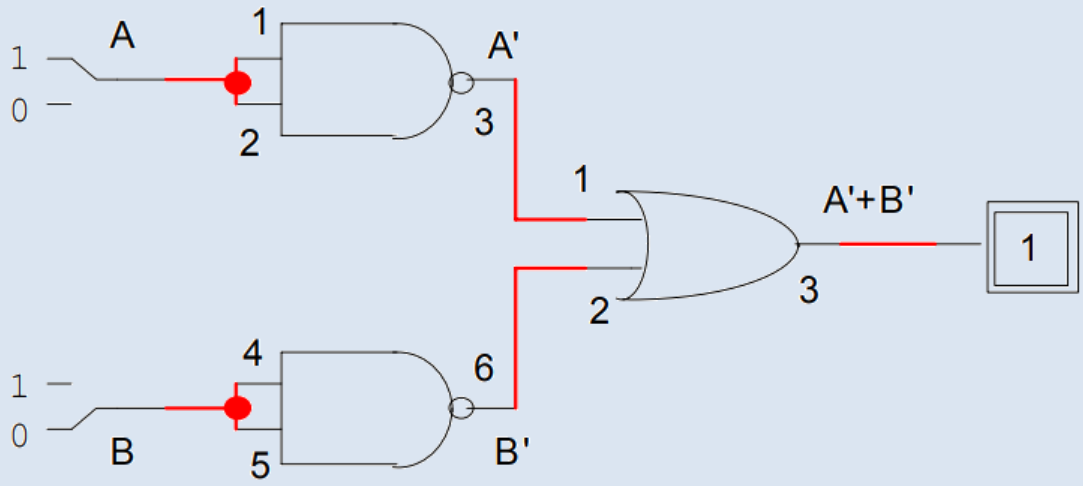


Fig. 5

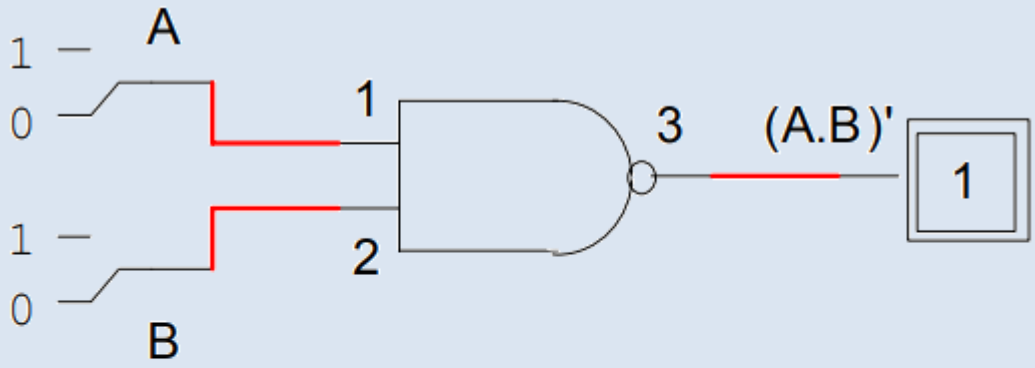
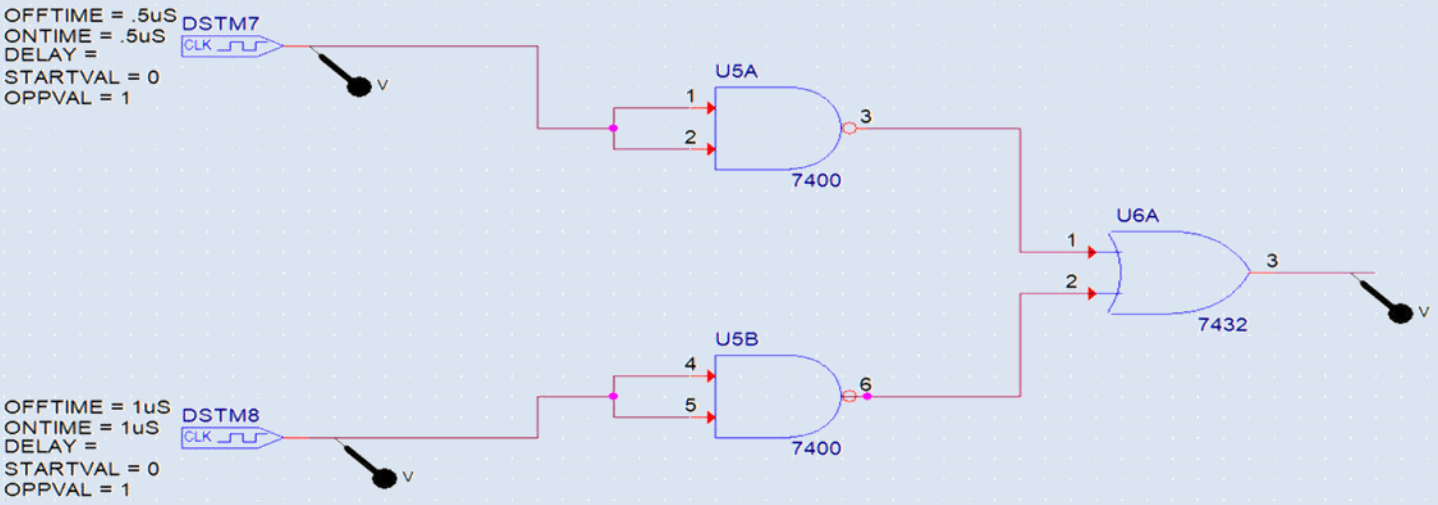
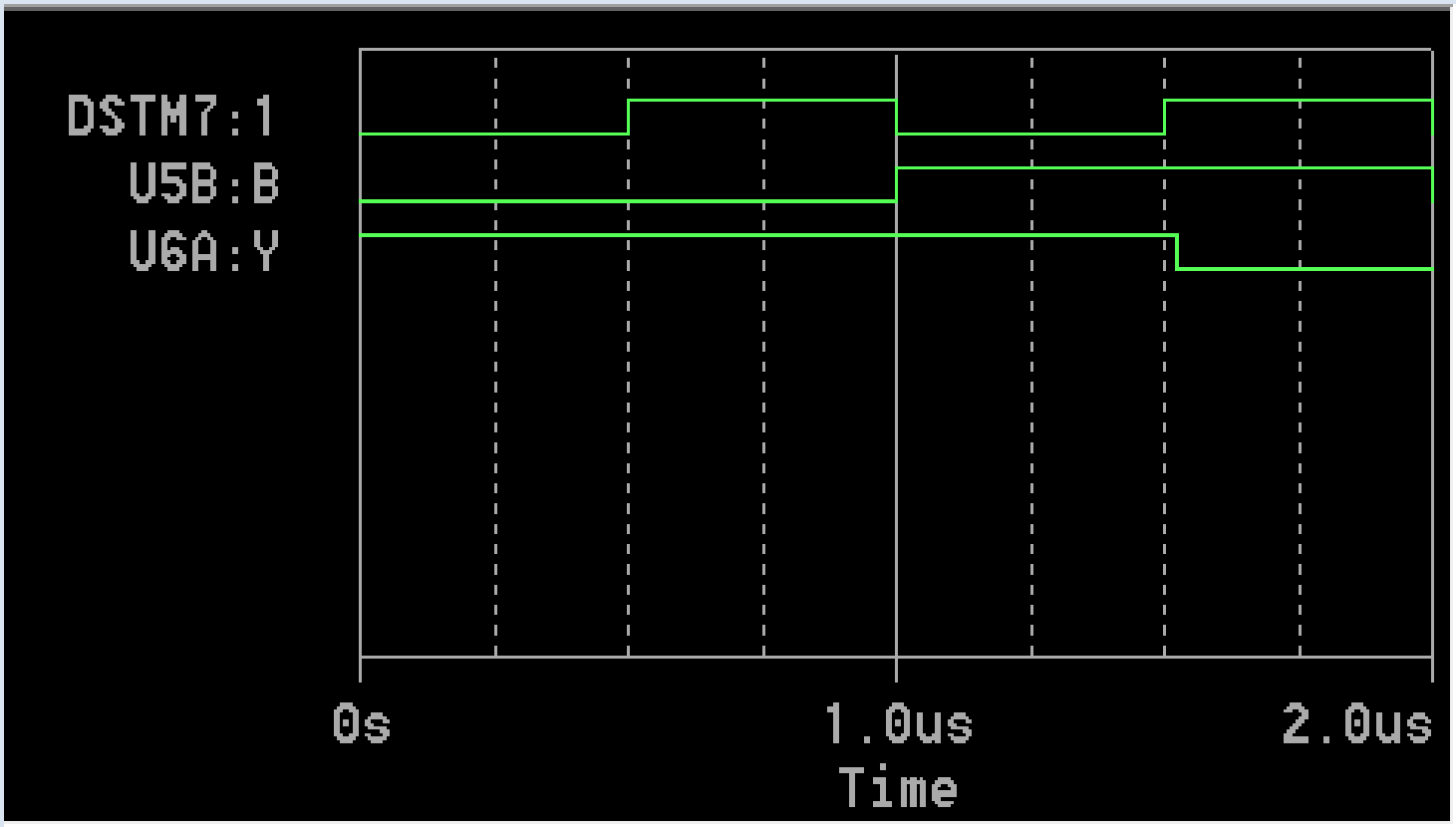


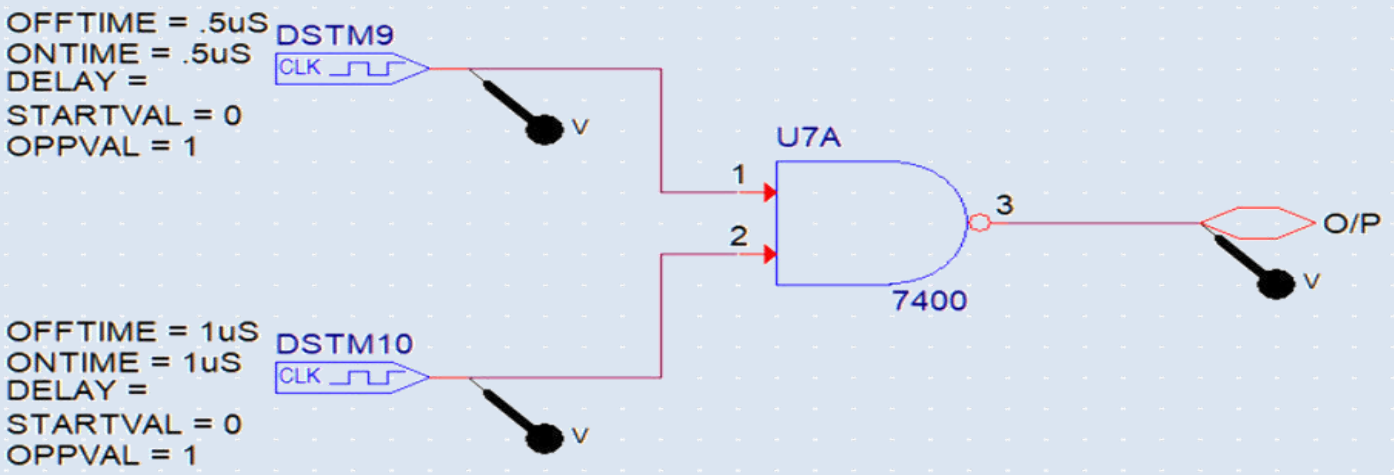
Fig. 6

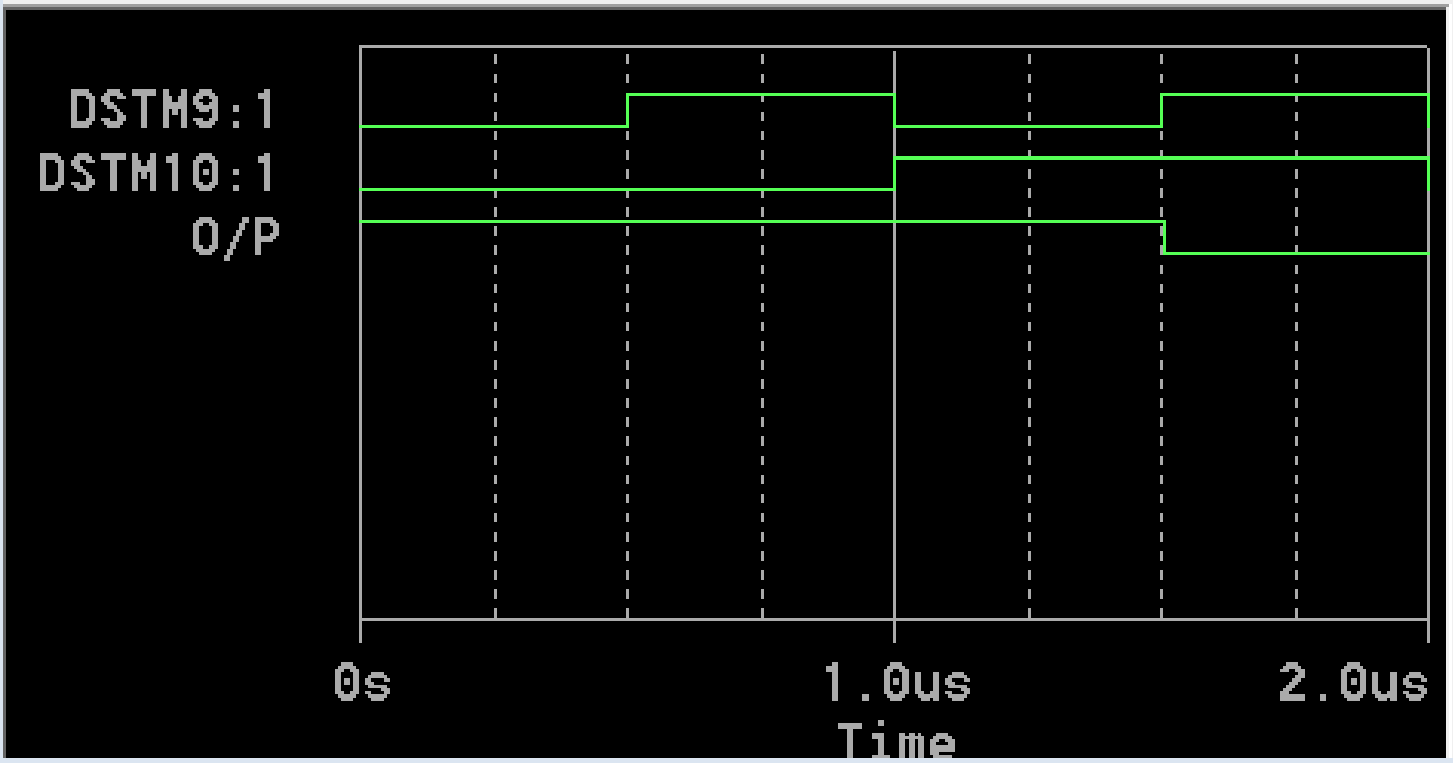
Circuit Analysis For Fig 5:





Circuit Analysis For Fig 6:





Here, from the result obtained by simulation, it is clearly visible that both the gates are logically equivalent.

Thus, Demorgan's second law is verified.

II. Design of a Digital Circuit

Consider the following problem:

Four chairs A, B, C, and D are placed in a row. Each chair may be occupied ("l") or empty ("0"). A Boolean function F is "l" if and only if there are two or more adjacent chairs that are empty.

1. Give the truth table defining the Boolean function F

Ans:

The required truth table for this problem is:

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

`

2. Express F as a minterm expansion (standard sum of product)

Ans:

The minterm expansion of above Boolean function is:

F = A'.B'.C'.D' + A'.B'.C'.D + A'.B'.C.D' + A'.B'C.D + A'.B.C'.D' +

A’.B.C’.D + A’.B.C.D’ + A.B’.C’.D’ + A.B’.C’.D + A.B’.C.D’ + A.B.C’.D’

3. Express F as a maxterm expansion (standard product of sum).

Ans:

The maxterm expansion of above Boolean function is:

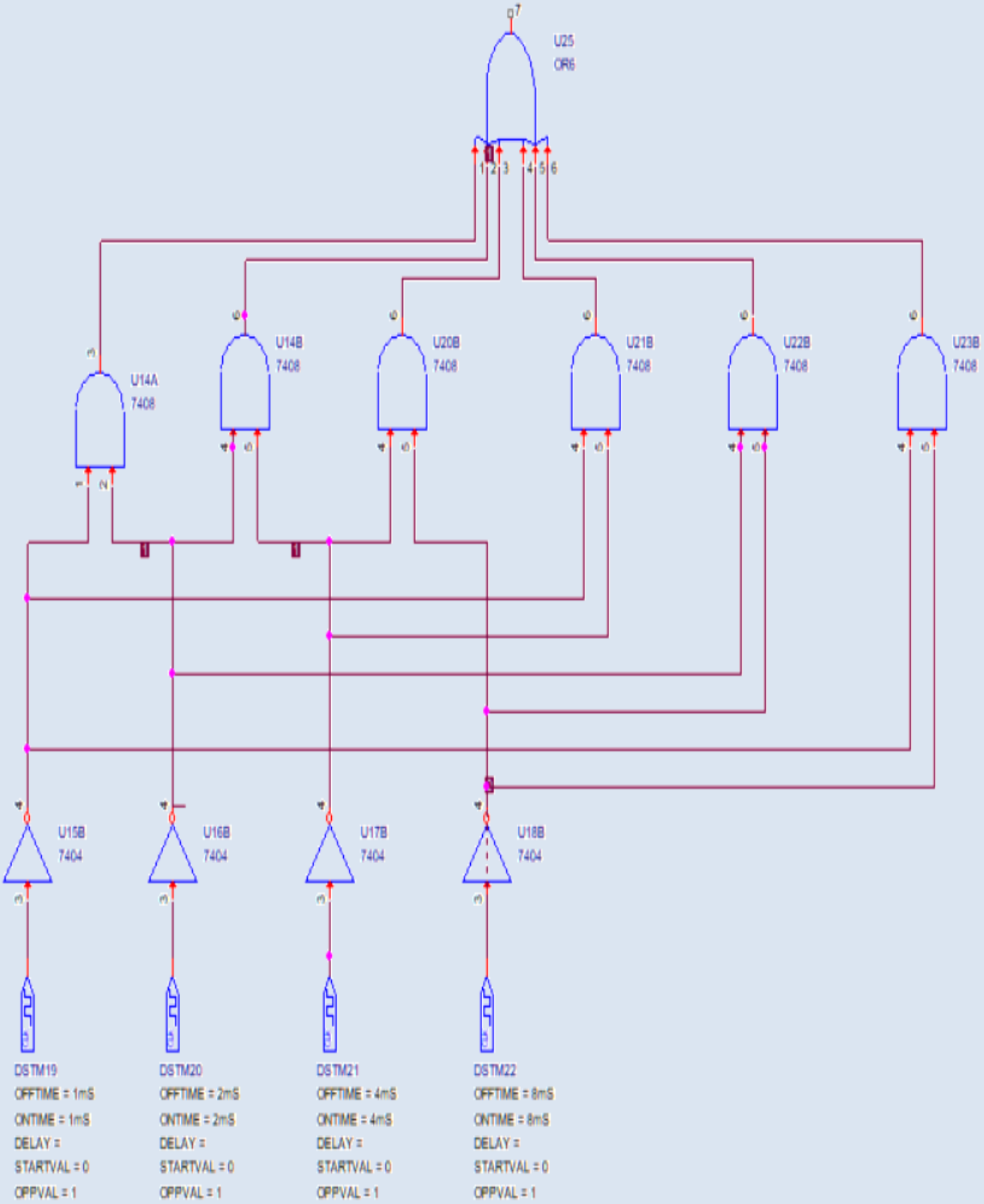
$$F = (A+B'+C'+D') \cdot (A'+B+C'+D') \cdot (A'+B'+C+D') \cdot (A'+B'+C'+D) \cdot (A'+B'+C'+D')$$

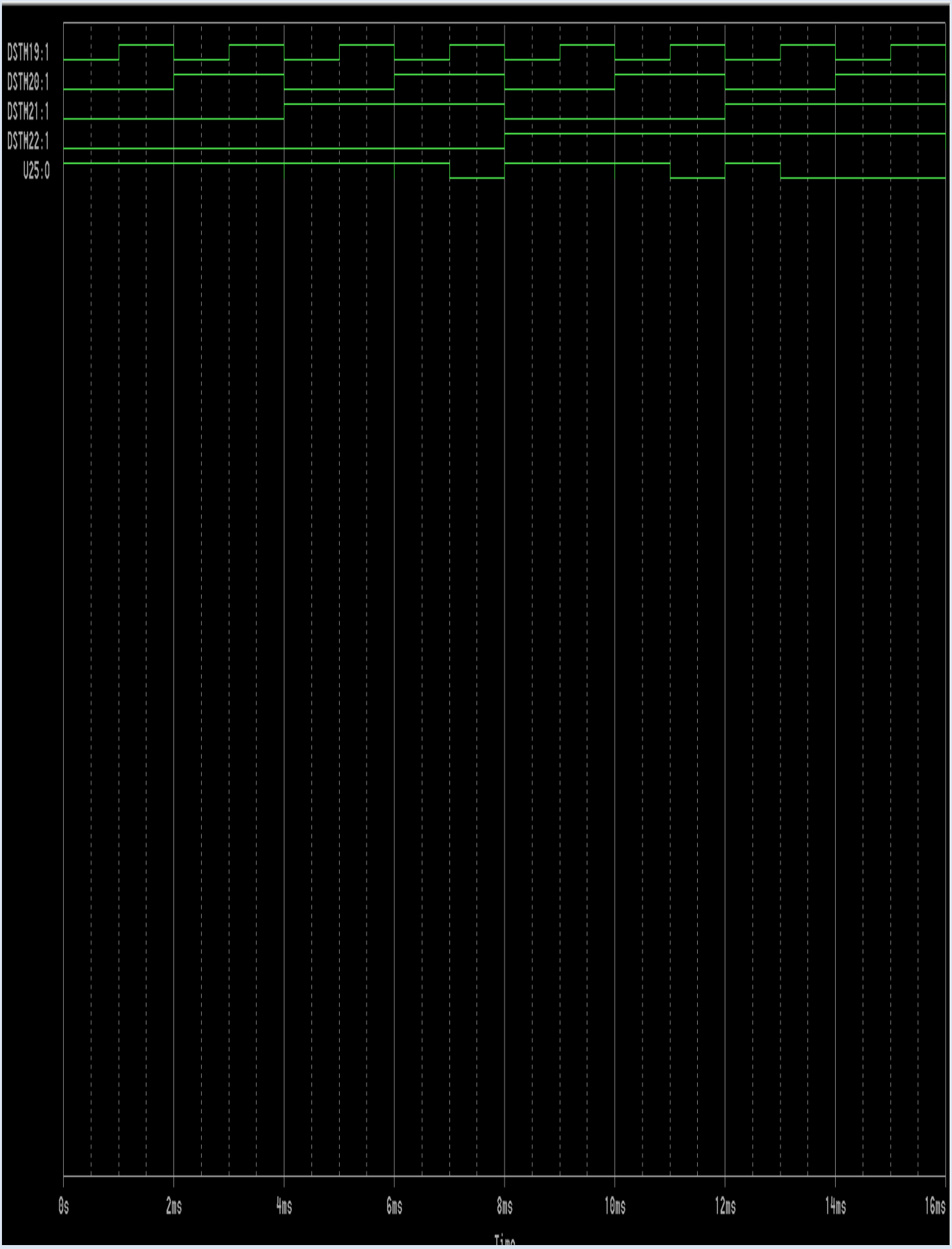
4. Using postulates and theorems Of Boolean algebra, simplify the minterm expansion of F to a form with as few occurrences of each as possible.

The above Boolean function can be reduced to the following form using the **Laws of Boolean Algebra:**

$$\mathbf{F = A'.B' + B'.C' + C'.D' + A'.C' + B'.D' + A'.D'}$$

5. Implement on ORCADCAPTURE CIS the simplified Boolean function with logic gates and check the operation of the circuit.


U25
OR6
U14A
7408
U14B
7408
U20B
7408
U21B
7408
U22B
7408
U23B
7408
U15B
7404
U16B
7404
U17B
7404
U18B
7404
DSTM19
OFFTIME = 1mS
ONTIME = 1mS
DELAY =
STARTVAL = 0
OPPVAL = 1
DSTM20
OFFTIME = 2mS
ONTIME = 2mS
DELAY =
STARTVAL = 0
OPPVAL = 1
DSTM21
OFFTIME = 4mS
ONTIME = 4mS
DELAY =
STARTVAL = 0
OPPVAL = 1
DSTM22
OFFTIME = 8mS
ONTIME = 8mS
DELAY =
STARTVAL = 0
OPPVAL = 1


DSTM19:1
DSTM20:1
DSTM21:1
DSTM22:1
U25:0
0s
2ms
4ms
6ms
8ms
10ms
12ms
14ms
16ms
Time

Result:

All truth tables, circuits (using OrCAD CAPTURE CIS), etc. used in completing this experiment has been shown above.

In this way, the laws of Boolean algebra can be observed, verified and studied practically using circuits and simulations

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Digital Logic Design
CSE1003 LAB

3.BOOLEAN SIMPLIFICATION
4.CODE CONVERSION

Experiment 3 & 4

DIGITAL LOGIC DESIGN

Exp #3- SIMPLIFICATION OF BOOLEAN FUNCTIONS USING K-MAP

OBJECTIVE:

- To develop the truth table for a combinational logic problem
- To use Karnaugh map to simplify Boolean expressions.
- To draw and simplify sum of products expressions.
- To draw logic diagrams using NAND gates.

APPARATUS:

- 7400 Quadruple 2 input NAND gates.
- 7404 Hex inverters
- 7410 Triple 3-input NAND gates
- 7420 Dual 4-input NAND gates
- 7432 Dual 2-input OR gates
- 7408 Dual 2-input AND gates

SOFTWARES USED:

- ORCAD CAPTURE CIS Lite

Part 1: BCD invalid code detector

BCD is a 4-bit binary code representing the decimal numbers 0

through 9. The binary numbers 1010 through 1111 are not used in

BCD.

a) Construct a truth table containing all possible inputs and desired output. Assume that the desired output for a valid code is a 1, and for an invalid code is 0. Complete the truth table as shown in Table 1. A is the most significant bit, and D is the least significant bit.

A	B	C	D	X
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	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Table: Truth Table of validity of BCD.

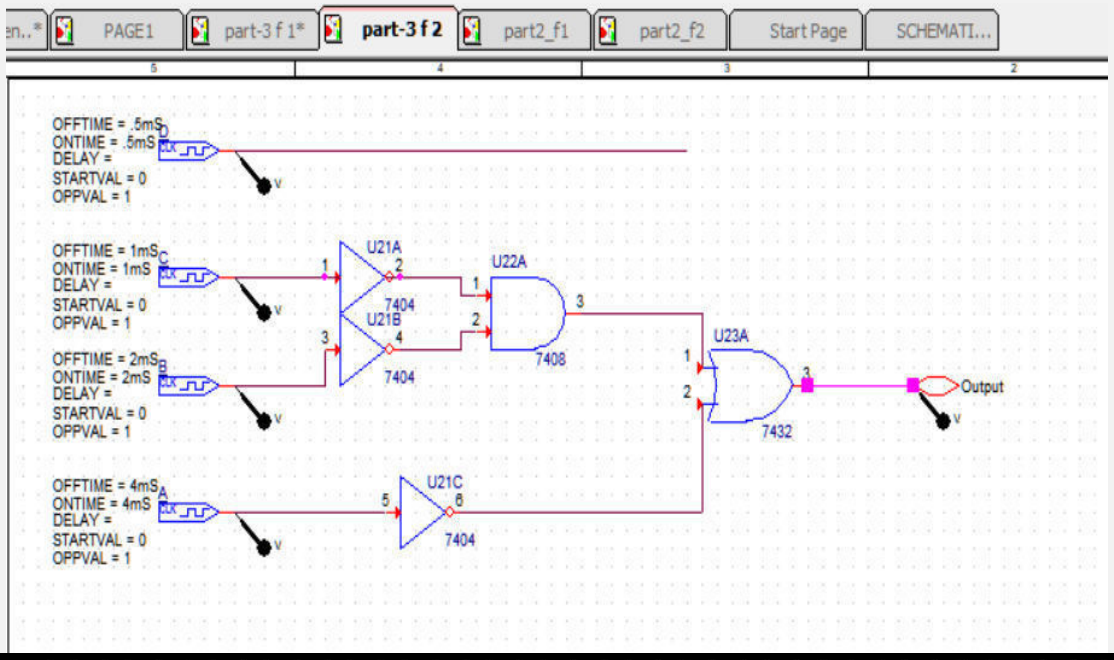
20BDS0405

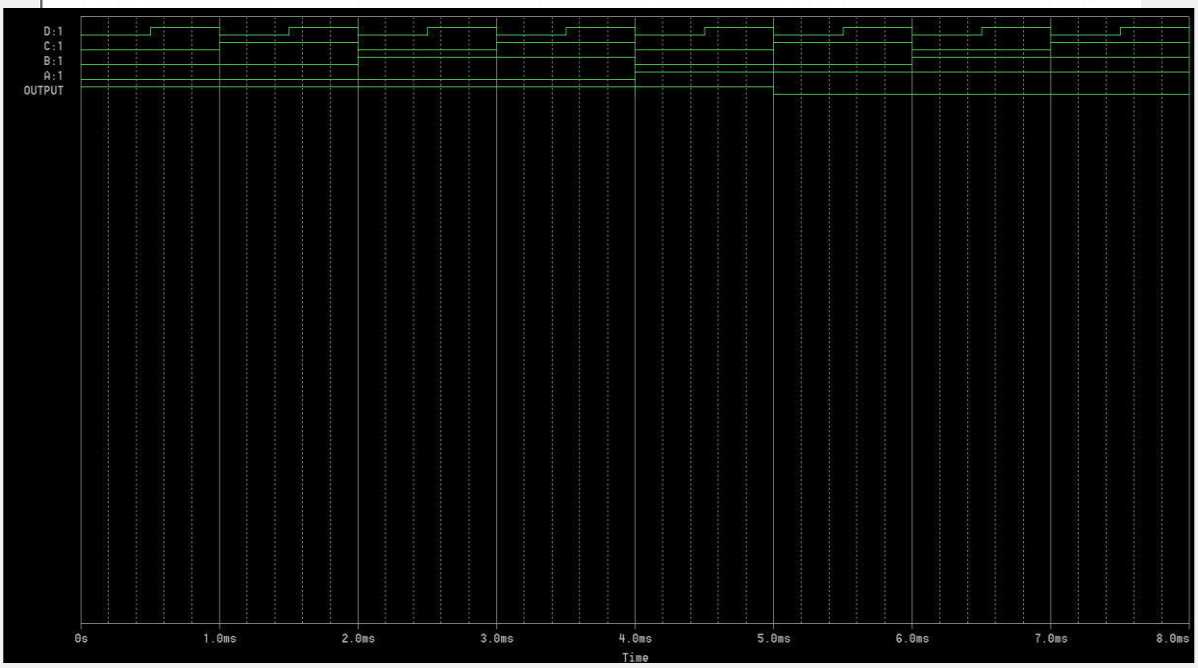
b) Draw the Karnaugh map, and write the simplified Boolean expression for the valid codes as sum of products.

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

$F = A' + B' C'$

a) Draw the circuit for the above simplified Boolean expression.

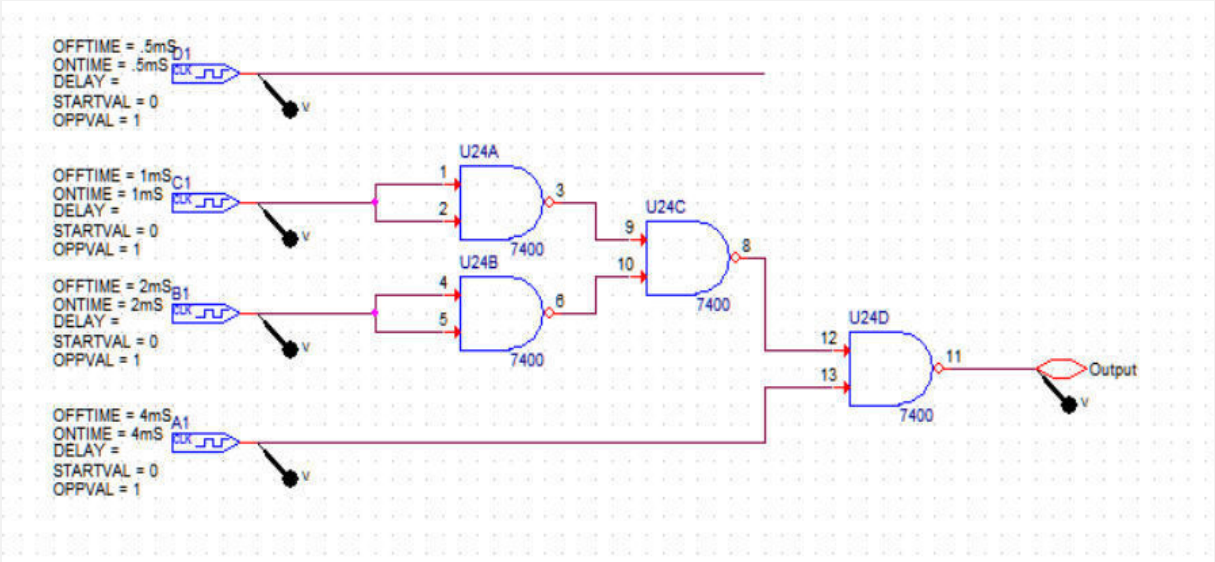


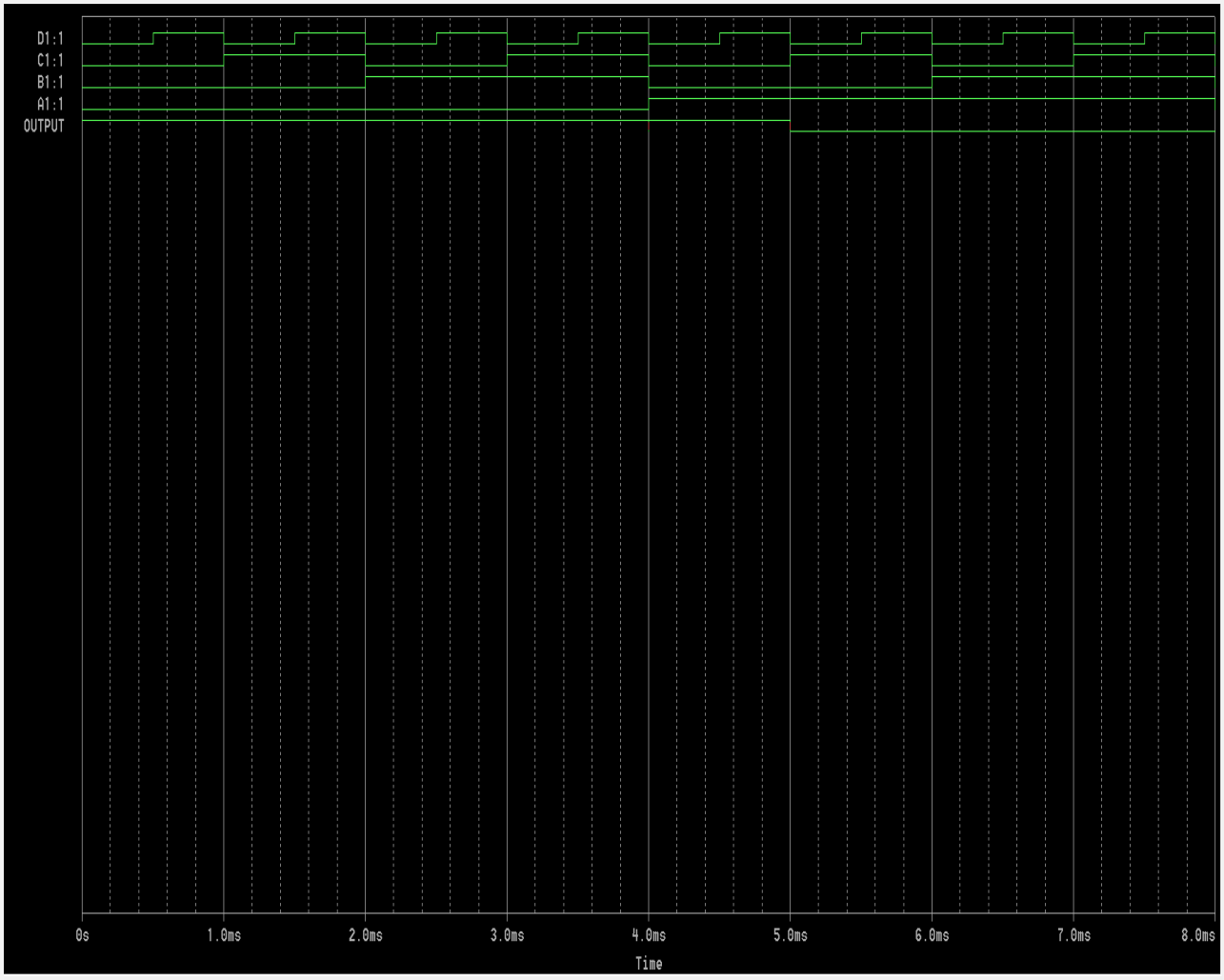


b) Using the universal property of the NAND gate

connect an equivalent circuit for these codes that uses only NAND gates.

Using NAND gates only:





Part 2: Boolean Functions (1)

1. Simplify the following two Boolean functions by means of Karnaugh maps.

2. Draw the logic diagrams for outputs F_1 and F_2 in terms of the inputs A, B, C, and D.

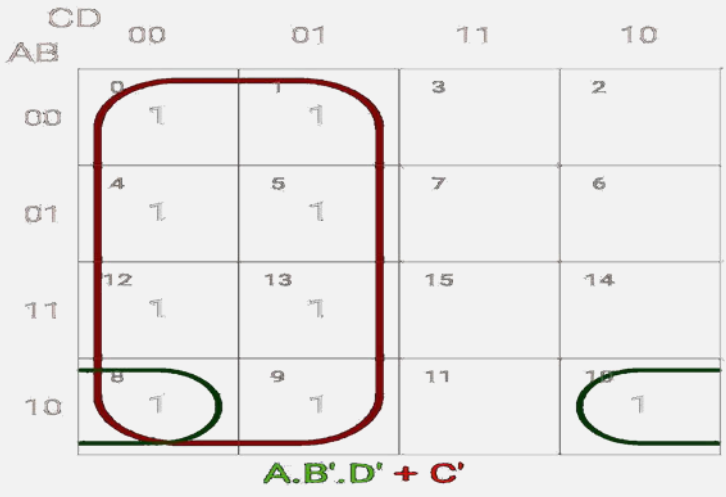
3. Implement and draw the two functions F_1 and F_2 together by using minimum number of NAND gates.

4. Connect the circuit and verify it's operation by preparing a truth table for F_1 and F_2 similar to Table 1.

For Boolean Function F1:

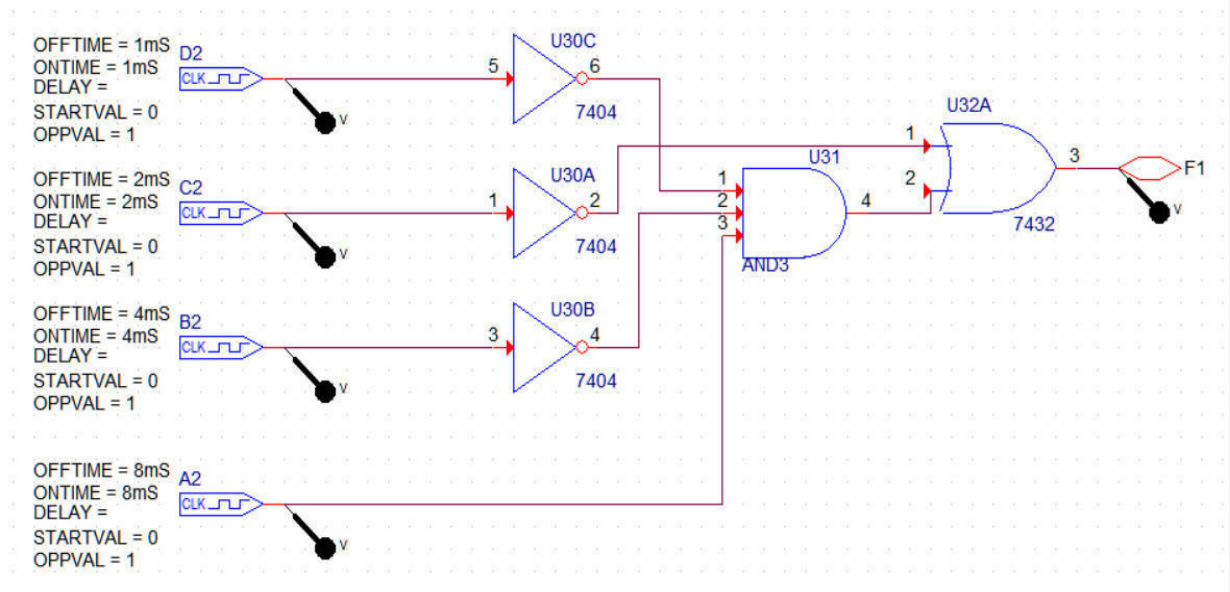
$$F_1 (A, B, C, D) = \Sigma m (0,1,4,5,8,9,10,12,13)$$

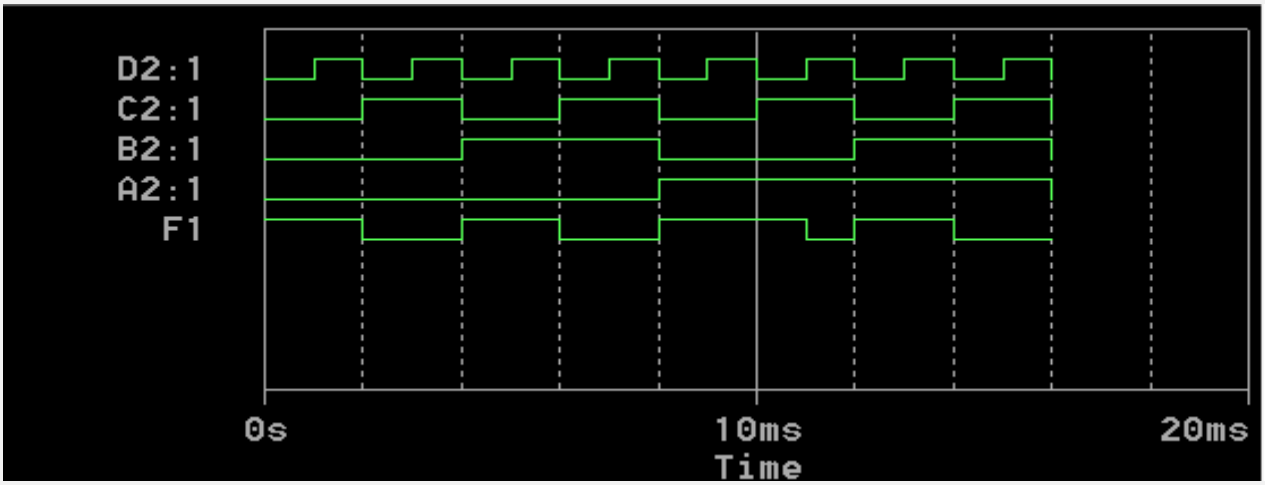
K- Map of F_1:



20BDS0405

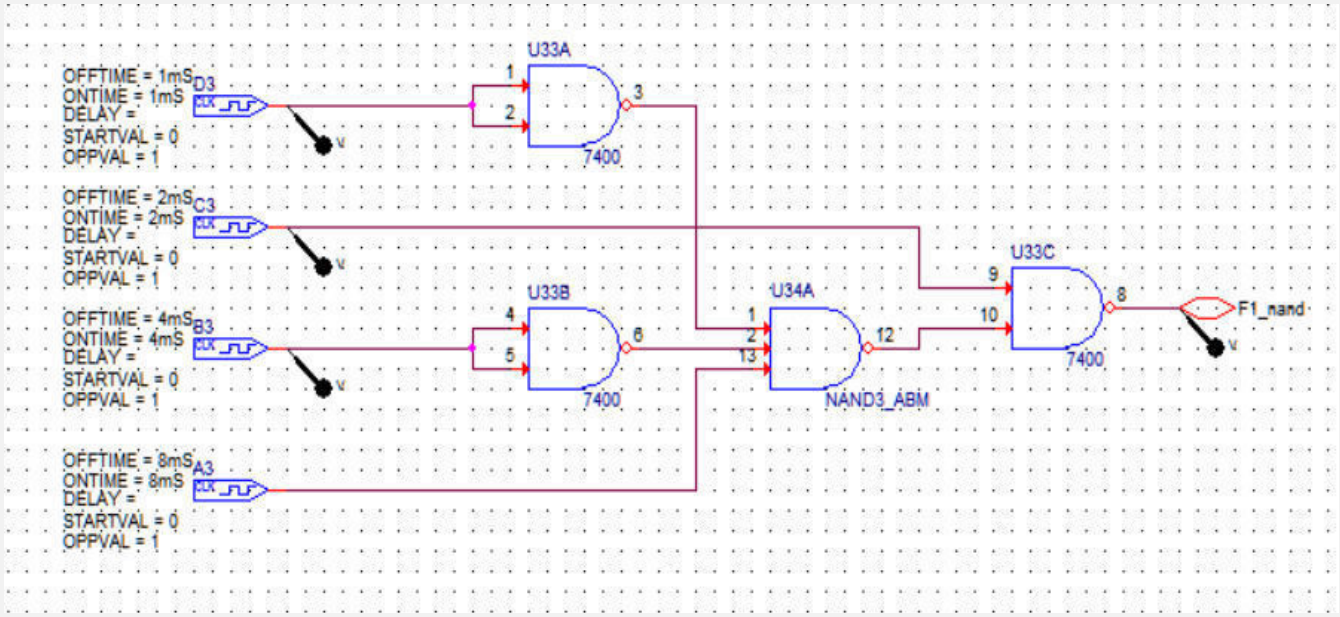
Logic Circuit of F1:

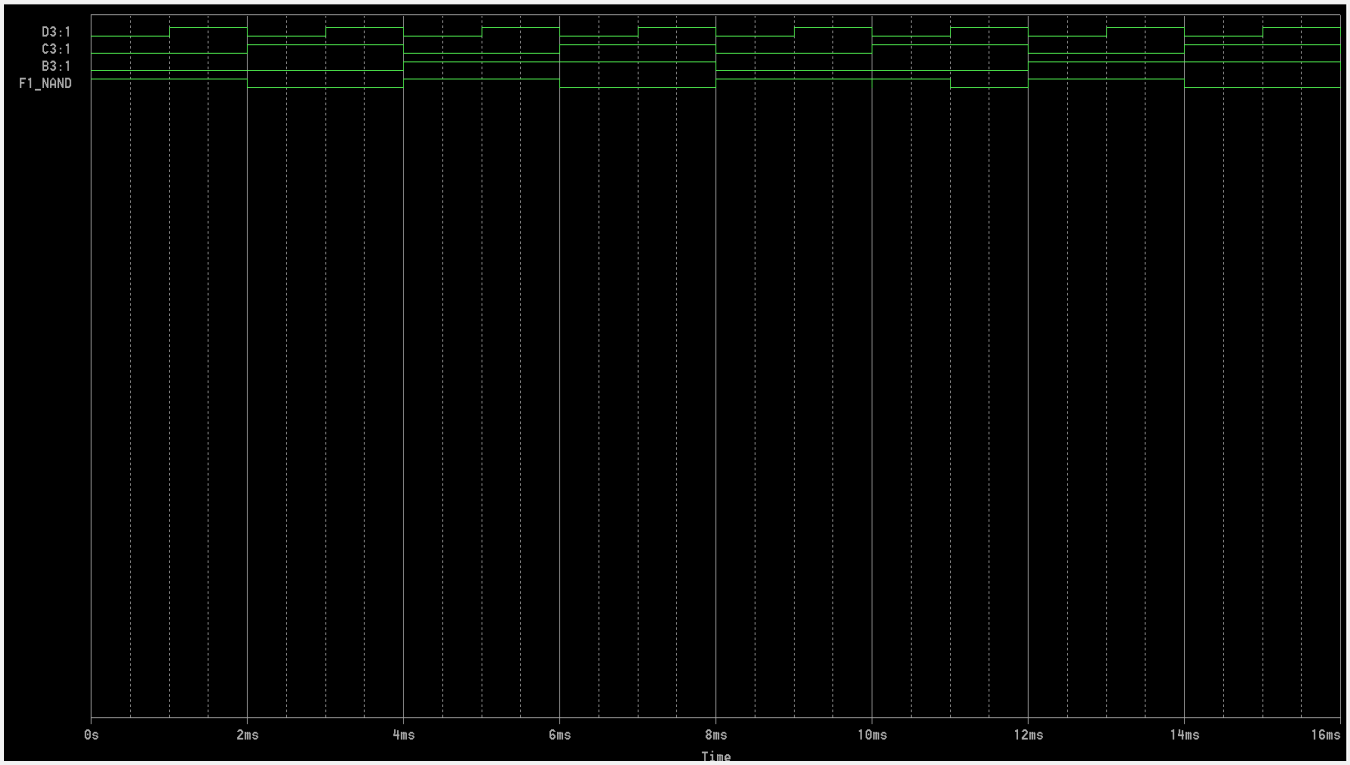




20BDS0405

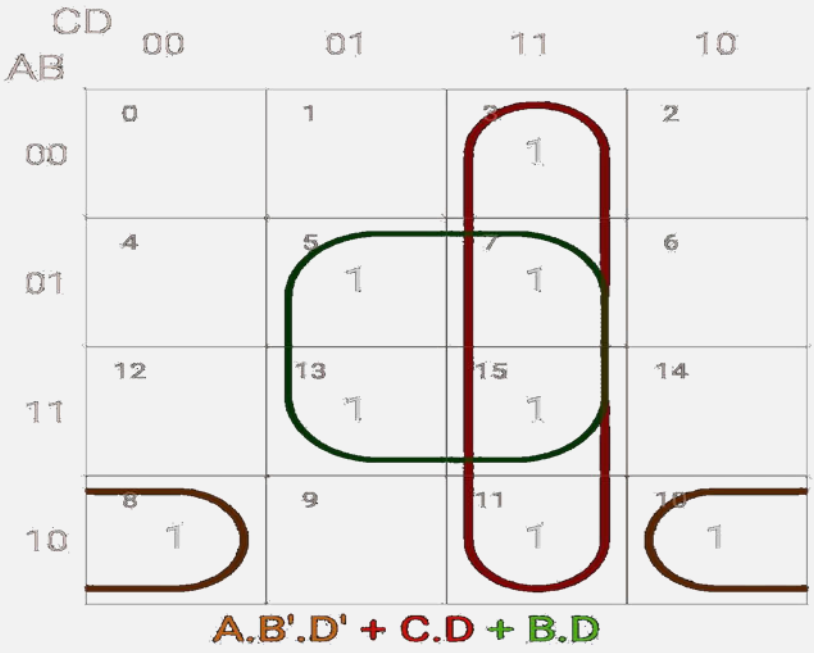
Logic Circuit of F1 using Only NAND gates:





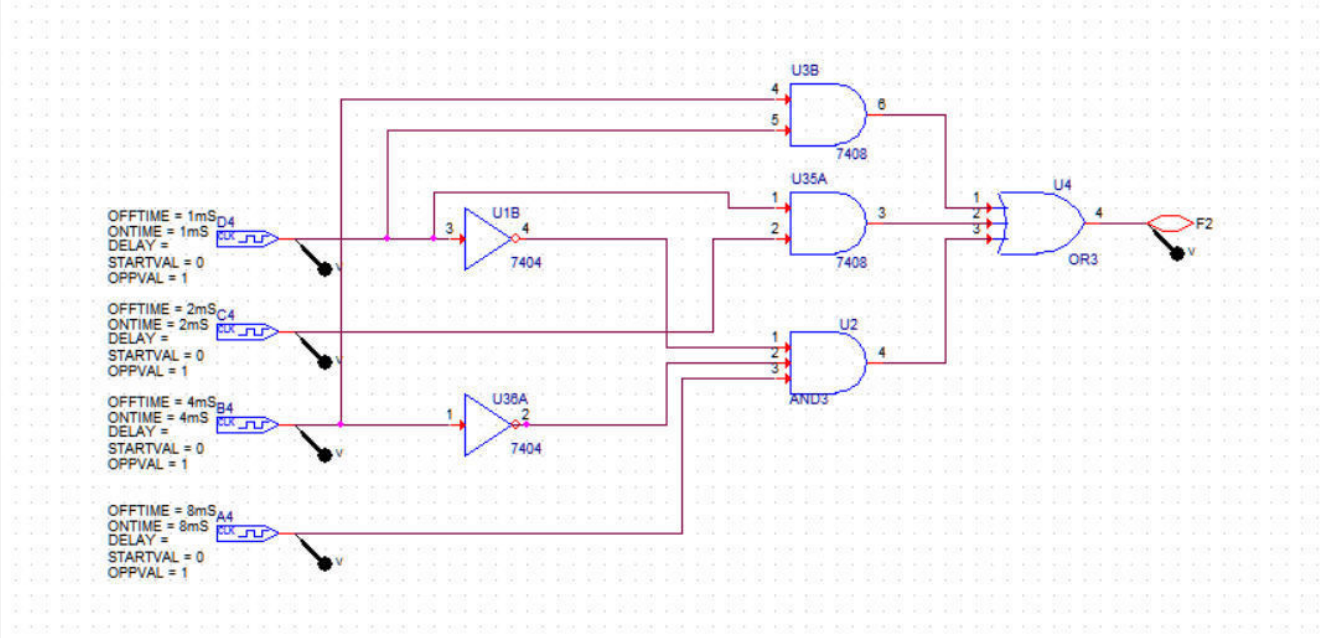
$$F_2 (A,B,C,D)= \sum m (3,5,7,8,10,11,13,15)$$

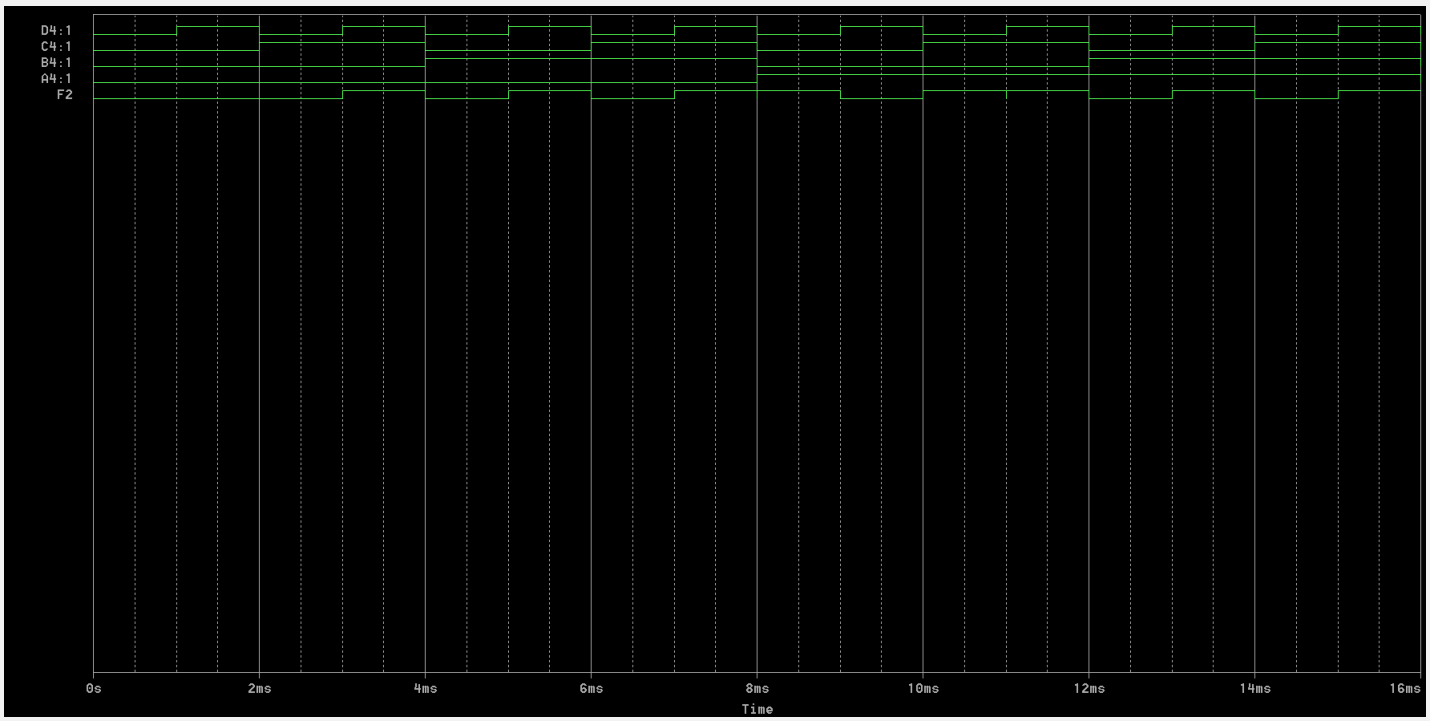
K- Map of F_2:



20BDS0405

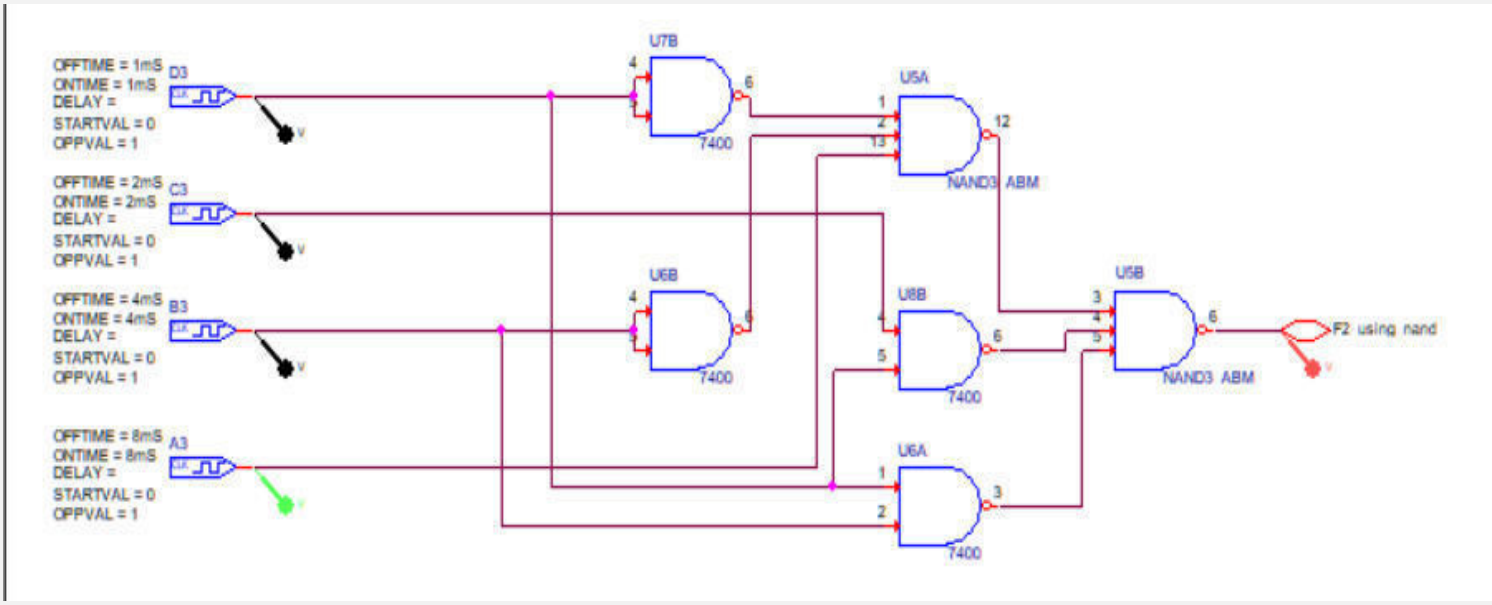
Logic Circuit of F_2:

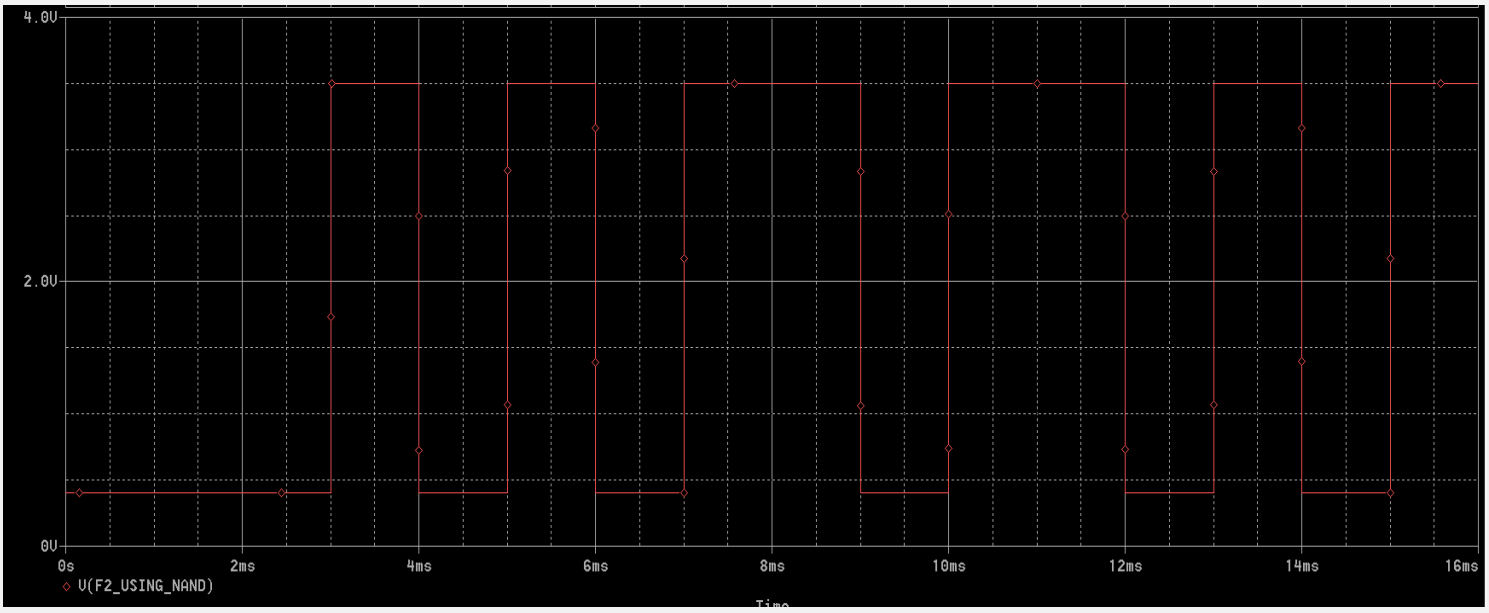




20BDS0405

Logic Circuit of F_2 using Only NAND gates:





Part 3: Boolean Functions (2)

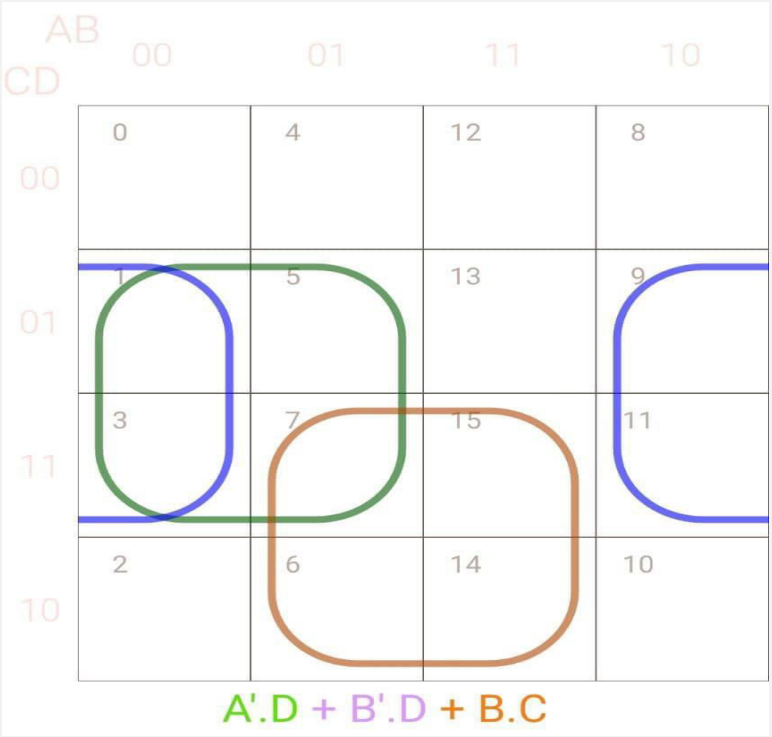
1. Derive a truth table for the following Boolean Functions.

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

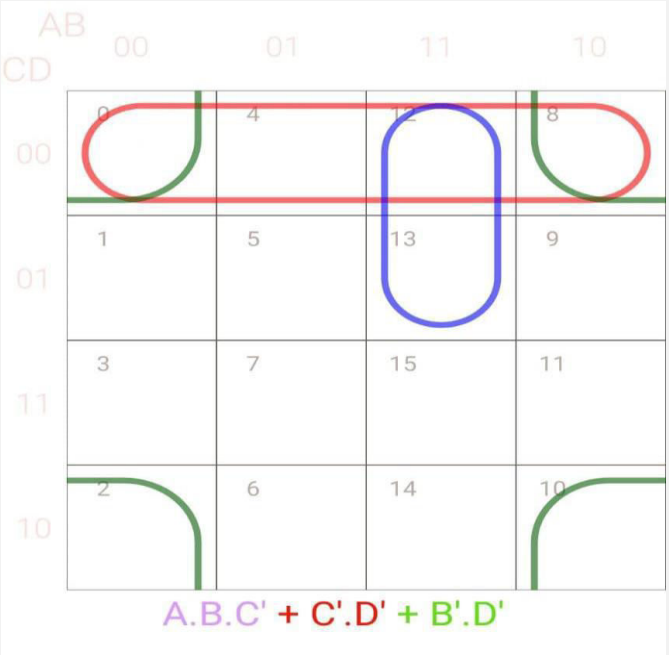
Fn = A'D+B'D+BC+AB'D				
A	B	C	D	Fn
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

2. Draw a Karnaugh map.

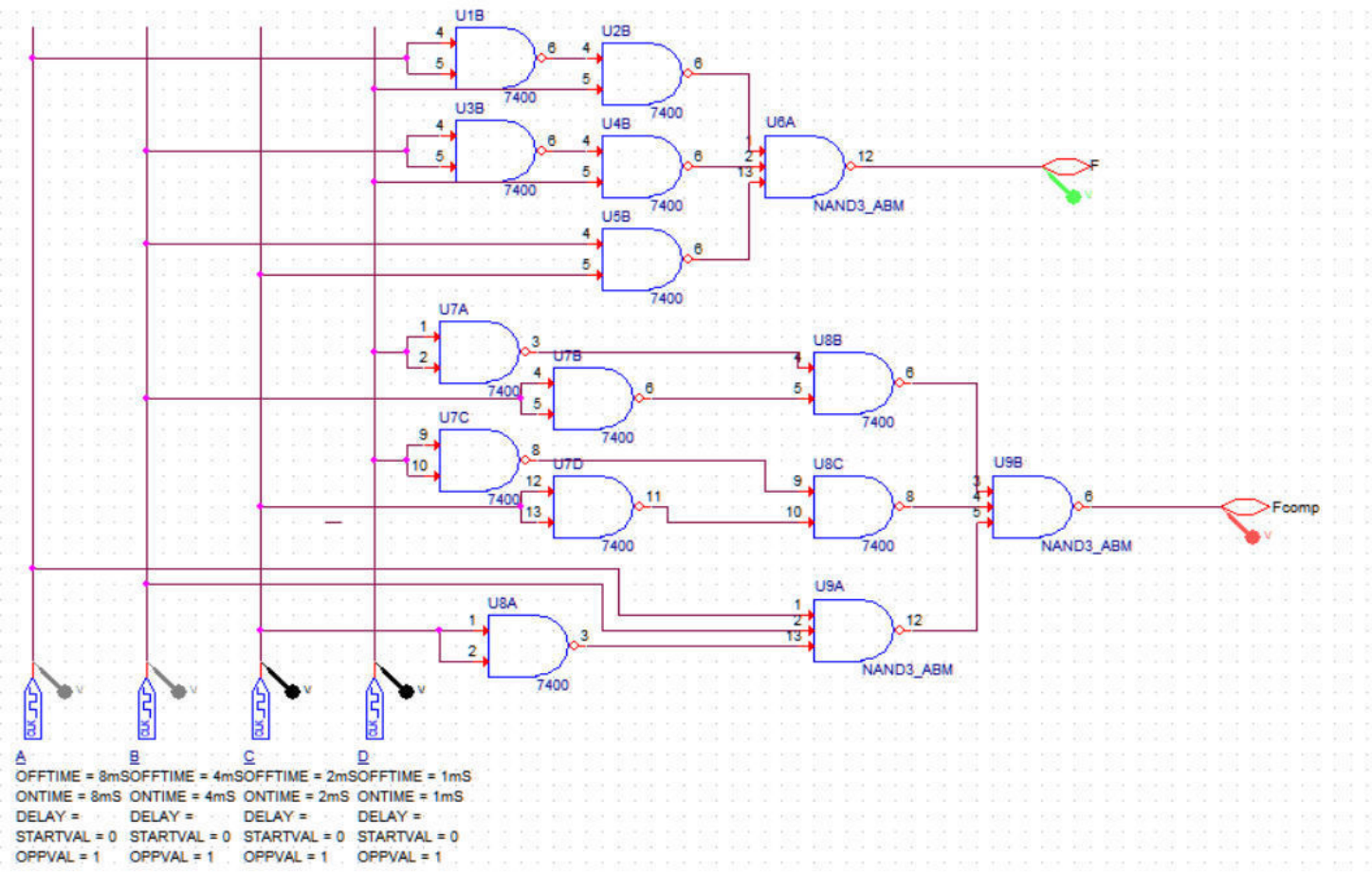
3. Combine all the 1's to obtain the simplified function for *F*.



4. Combine all the 0's to obtain the simplified function for *F'*.

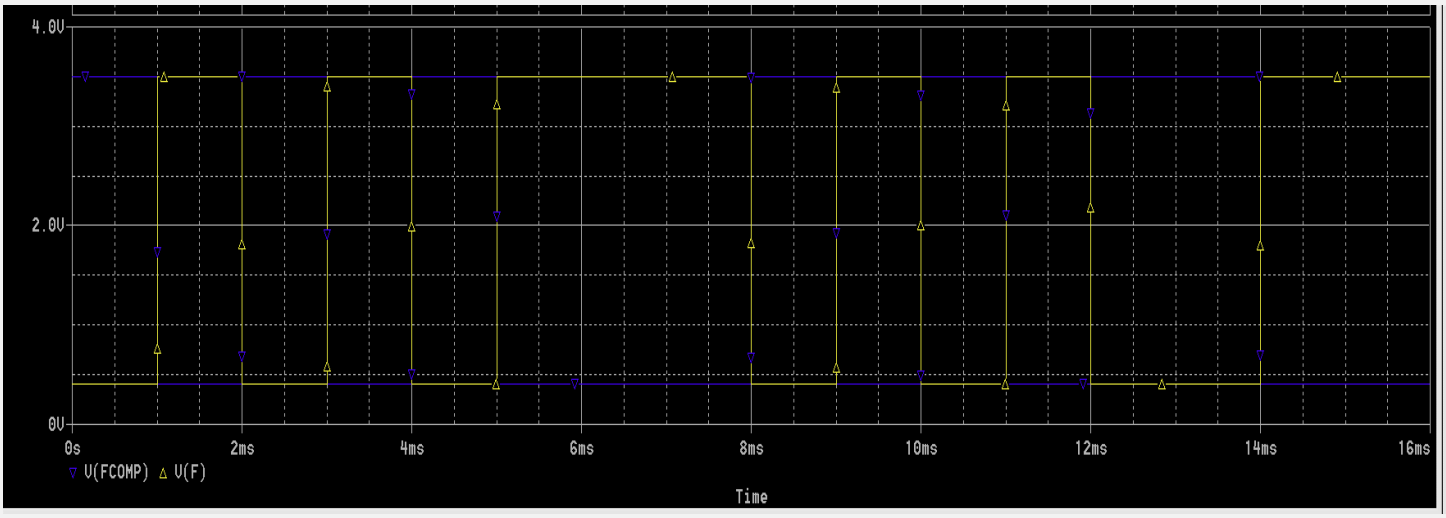


5. Draw both circuits.



6. Using OrCAD, implement both F and F' using NAND gates and connect two circuits to the same input switches but to separate output LED's. Prove that both circuits are complement of each other. In the lab implement and verify the operations of the circuit.

Logical Circuit and Output of Both above circuits are:



Part 4: A Majority

A nine member legislative committee requires a 2/3 vote to spend a billion dollars. The vote is tabulated and converted to BCD code. If 2/3 of the committee is in favor, the vote will be the BCD representation of 6, 7, 8, or 9.

1. Derive a truth table for the problem, Table 2.

A	B	C	D	X
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

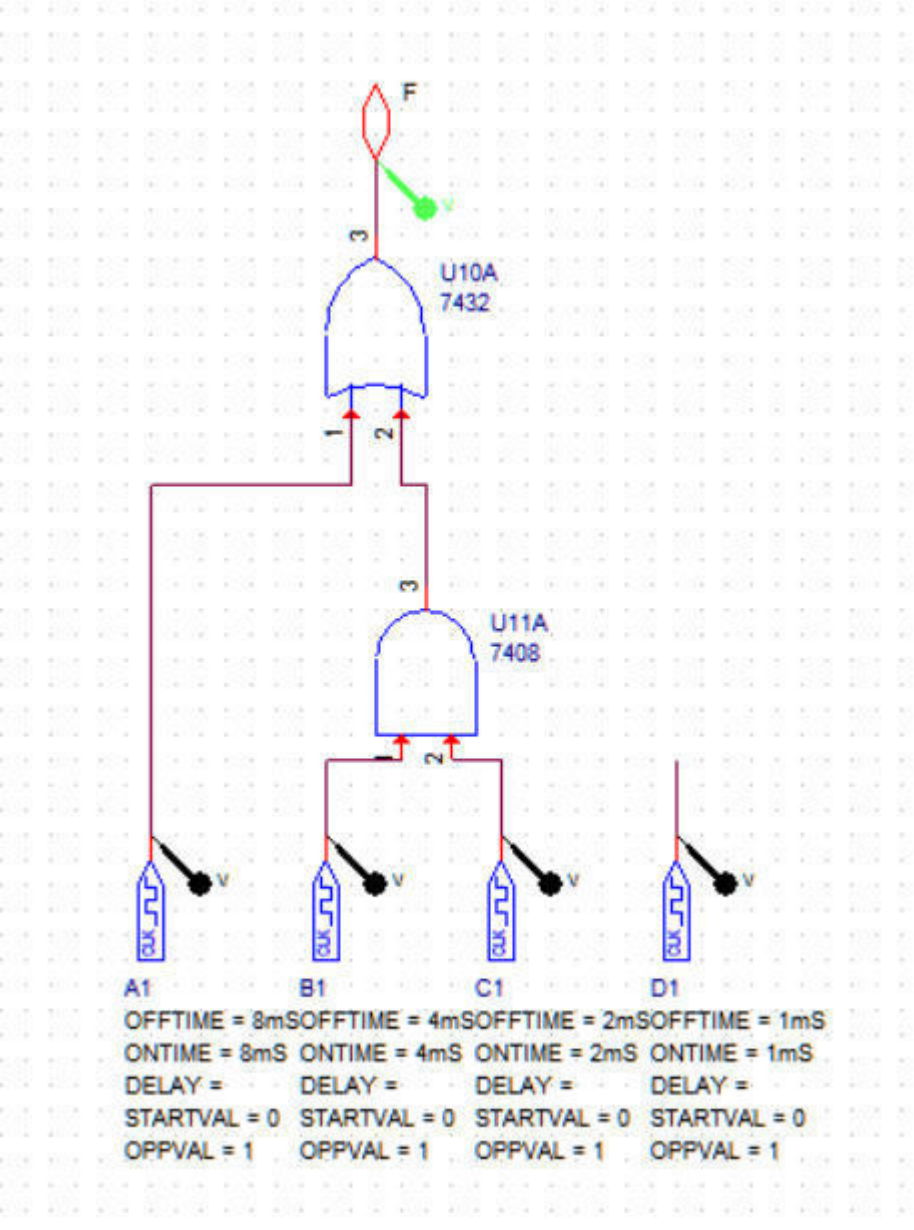
2. Derive a minimum sum of products expression from the map. {Enter the invalid BCD codes on the map as don't cares (x)}.

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

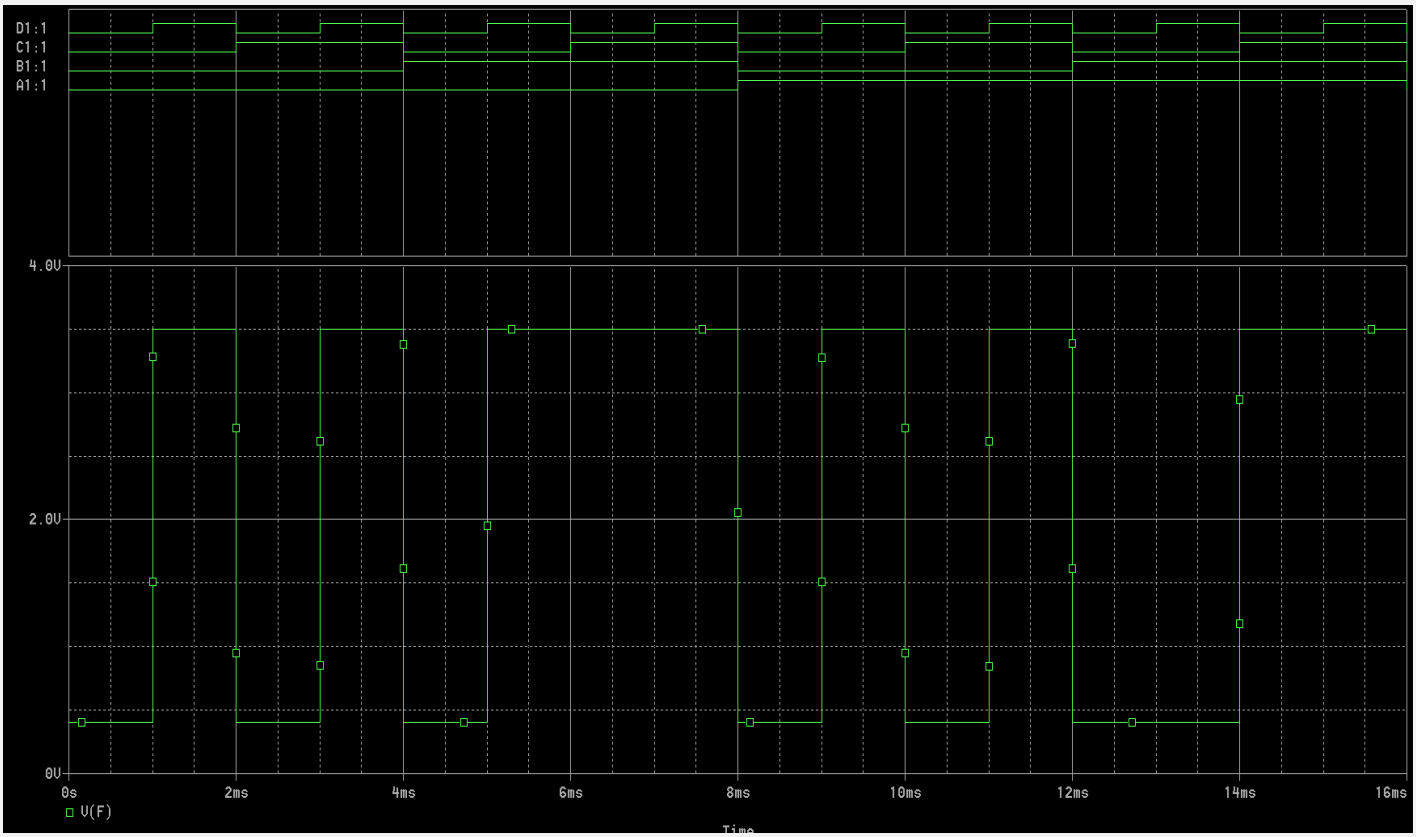
Hence, from the above Kmap, the minimal SOP form is:

F= A + BC

1. Using OrCAD Capture CIS, design a circuit that lights an LED if a majority has voted in favor of spending the billion dollars. Implement this circuit and verify its operation in the lab using hardware.



20BDS0405


D1:1
C1:1
B1:1
A1:1
4.0V
2.0V
0V
0s
2ms
4ms
6ms
8ms
10ms
12ms
14ms
16ms
V(F)
Time

20BDS0405

DIGITAL LOGIC CIRCUIT DESIGN CSE1003

Exp#4 - DESIGN OF CODE CONVERTERS

OBJECTIVE:

1. Design and build gray code to binary converter.
2. Design and build BCD-to-7 segment converter.

APPARATUS:

- Seven segment display.
- SN 7400 quad 2-input NAND gates (1)
- SN 7410 triple 3-input NAND gates (4)
- SN 7420 dual 4-input NAND gates (4)
- SN 7404 HEX inverter (1)
- SN 7446 BCD-to-seven segment decoder.

SOFTWARES USED:

- ORCAD CAPTURE CIS
- Proteus 8 pro

THEORY:

The conversion from one code to another is common in digital systems. Sometimes the output of a system is used as the input to the other system. A conversion circuit is necessary between 2 systems if each system uses different codes for the same information.

In this experiment we will design and construct 3-combinational circuit converters:

Procedure:

1. *Gray code to Binary converter:*

Gray code is one of the codes used in digital systems. It has the advantage over binary numbers that only one bit in the code word changes when going from one number to the next.

The following table illustrates the corresponding code representations of Decimal numbers in Binary Number System (Base 2 number system) and Gray Code Convention (Non-weighted)

Decimal	Binary	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Design a combinational circuit with 4 inputs and 4 outputs that converts a four- bit gray code number into an equivalent four-bit Binary number.

Let, the 4 Bit Gray Code number be ABCD and Binary number be WXYZ.

	Gray Code (ABCD)				Binary Code (WXYZ)			
Decimal	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	1	0	0	1	0
3	0	0	1	0	0	0	1	1
4	0	1	1	0	0	1	0	0
5	0	1	1	1	0	1	0	1
6	0	1	0	1	0	1	1	0
7	0	1	0	0	0	1	1	1
8	1	1	0	0	1	0	0	0
9	1	1	0	1	1	0	0	1
10	1	1	1	1	1	0	1	0
11	1	1	1	0	1	0	1	1
12	1	0	1	0	1	1	0	0
13	1	0	1	1	1	1	0	1
14	1	0	0	1	1	1	1	0
15	1	0	0	0	1	1	1	1

Table 1: Gray code with corresponding Binary and Decimal Equivalent values

- Use Karnaugh map technique for simplification.
- Use OrCAD for pre-lab demonstrations.
- Select the library "7400dev.clf " in the Parts Palette.
- Then select the XOR chip 74-86. This would give you a set of 4 XOR's as shown in Fig. 1, just like the hardware chip 74-86. You could use as many as needed from these XOR gates in your design.
- Get back to ALL LIBRARIES and select switches for the inputs and Binary Probes as indicators of the outputs.
- Verify your design in the pre-Lab. During the Lab construct the circuit and verify its operation

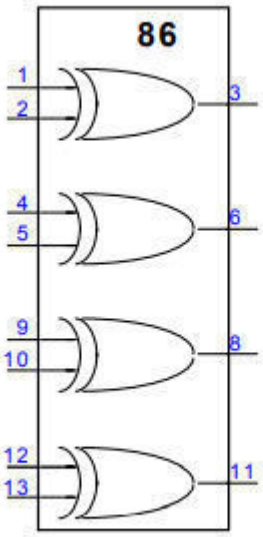
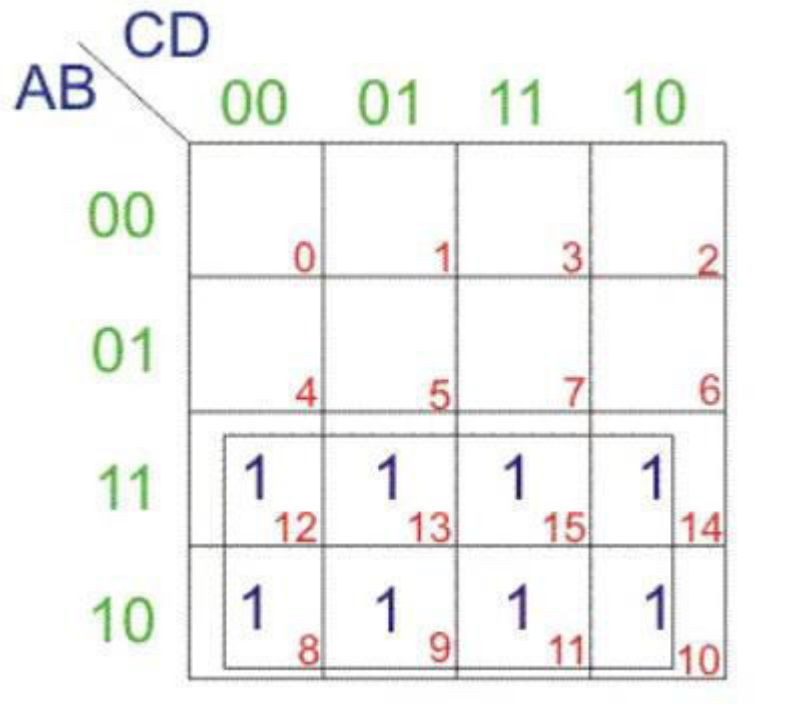


Figure 1 XOR chip74-86

20BDS0405

Following is the K Map simplification for Gray code to Binary Conversion.
Each digits of the Binary number form a separate Karnaugh Map where each digit is a direct function of all the gray code digits.

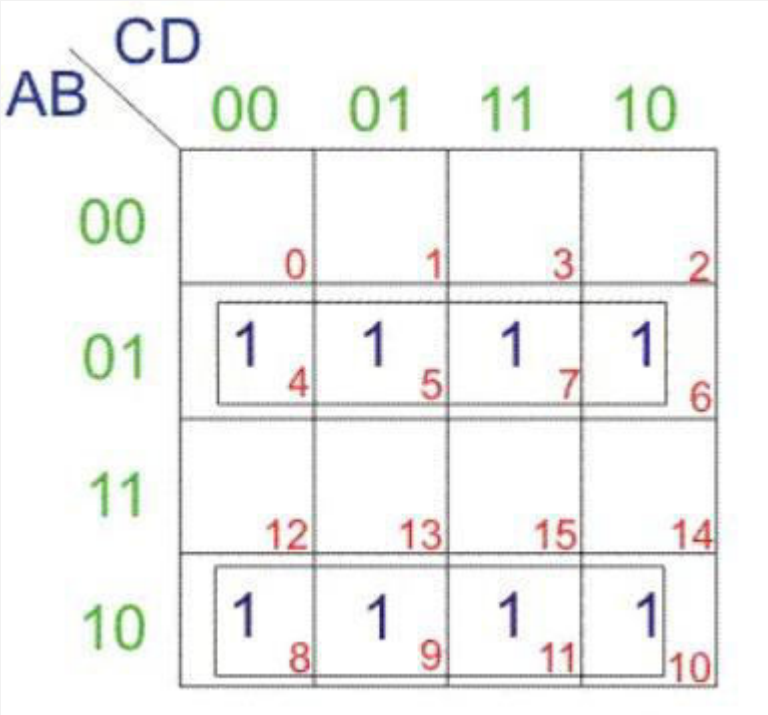
For W,



W=A

20BDS0405

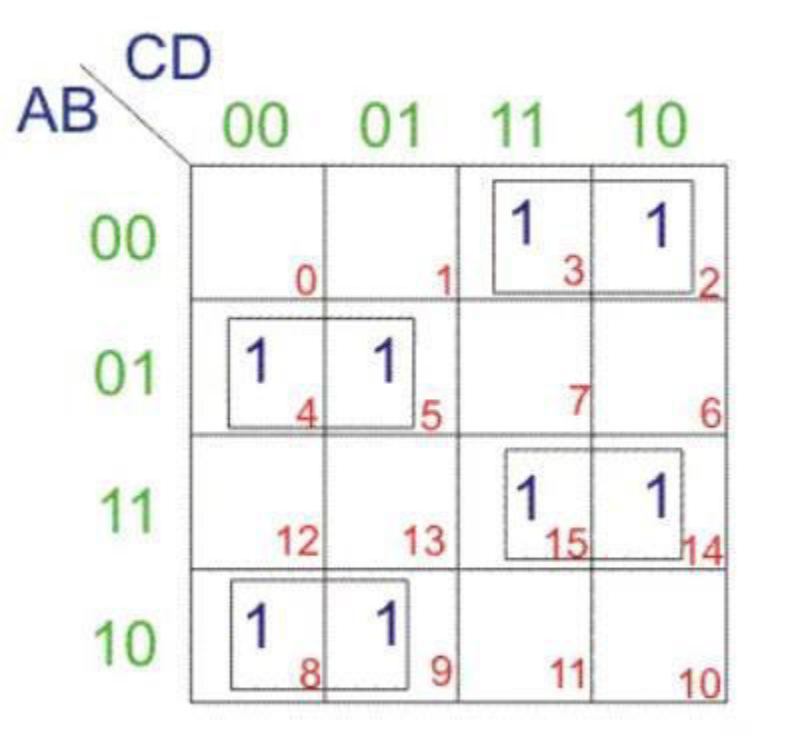
For X,



$$X=A'B+AB'$$
$$=A \oplus B$$

20BDS0405

For Y,



Y=A'BC' + AB'C' + A'B'C+ ABC

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

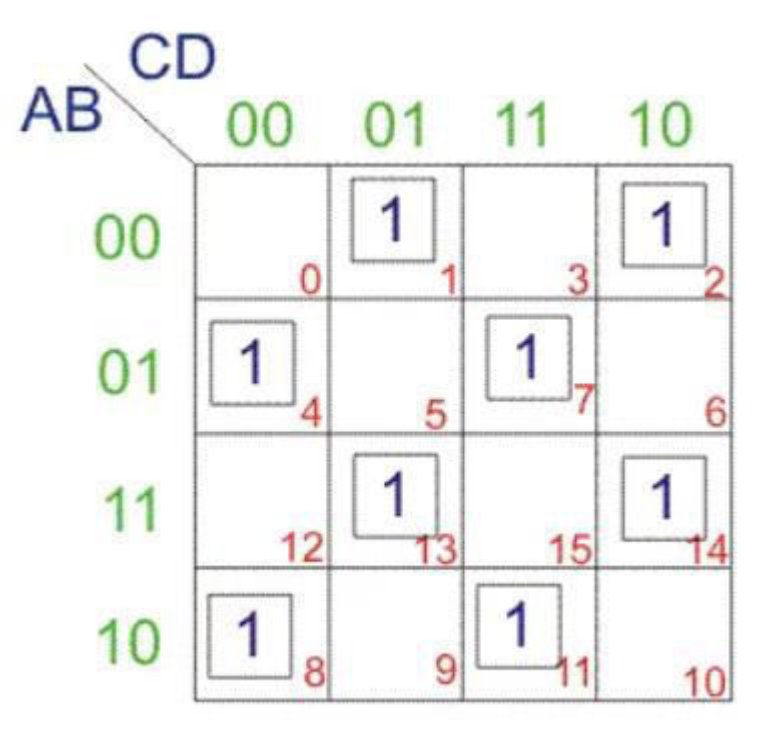
=A (BC'+B'C)'+A' (BC'+B'C)

=A ($B_{\oplus}$ C)' + A' ($B_{\oplus}$ C)

= $A_{\oplus}B_{\oplus}$ C

20BDS0405

For Z,

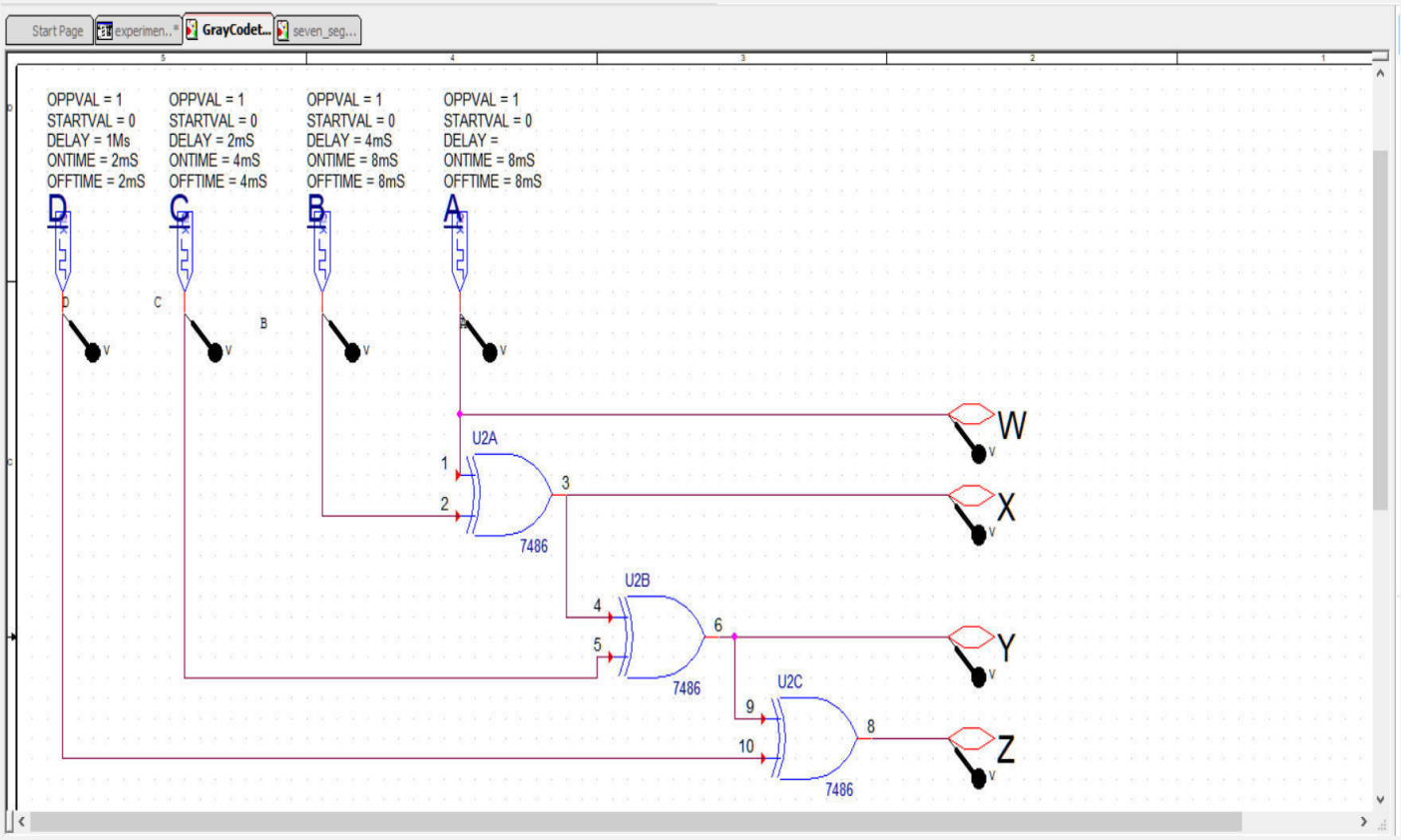


Z=A’B’C’D + A’B’CD’ + A’BC’D’ + AB’C’D’…
…+ A’BCD + AB’CD + ABC’D + ABCD’

Z= A⊕B⊕ C⊕D

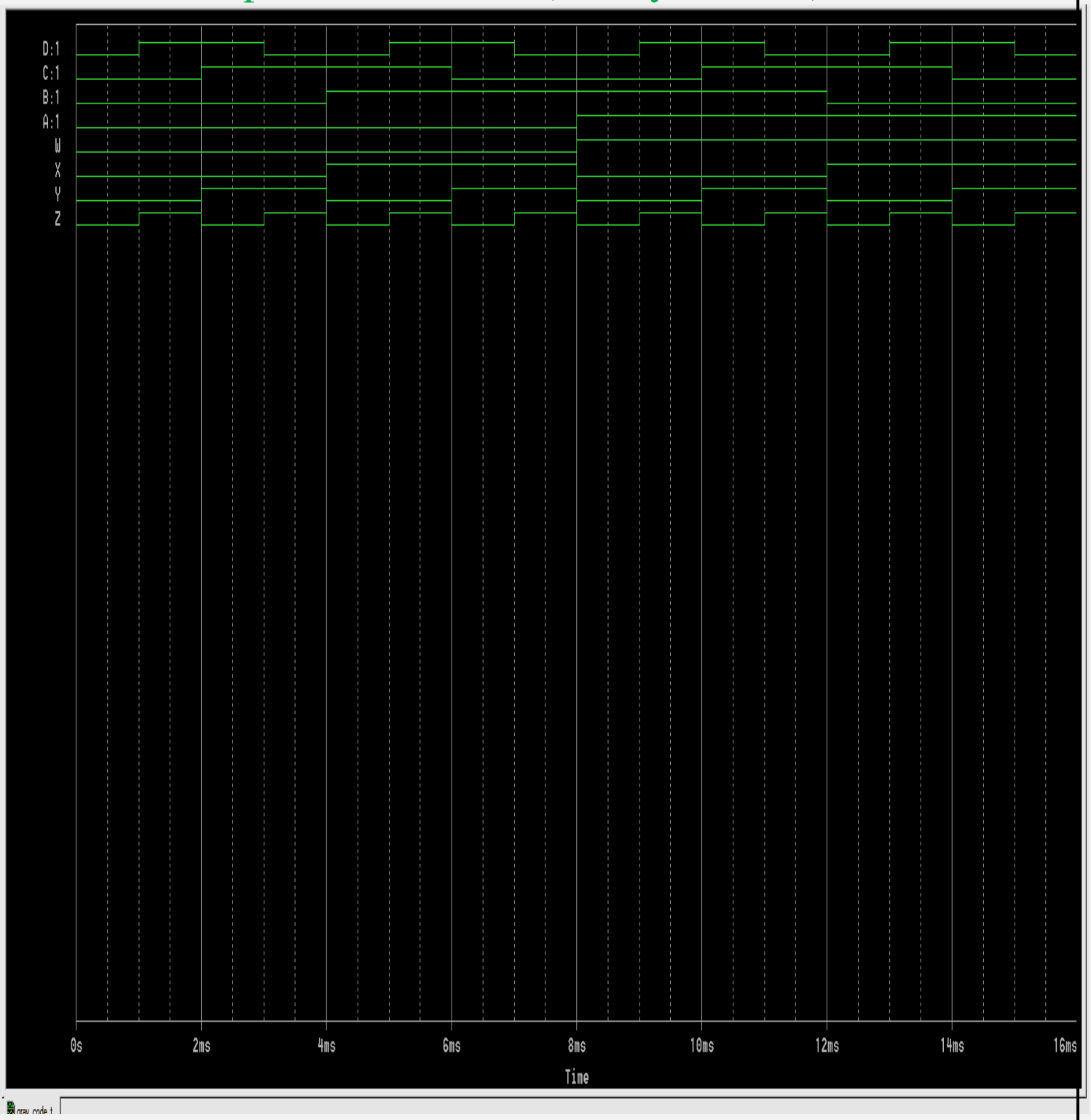
20BDS0405

Construction and simulation of Logic diagram to Convert Gray code Input to Binary Number.:



20BDS0405

Inputs: A, B, C, D (Gray Code)
Outputs: W, X, Y, Z (Binary number)



2. BCD-to-seven Segment converter:

A light emitting Diode (LED) is a PN junction diode. When the diode is forward biased, a current flows through the junction and the light is emitted.

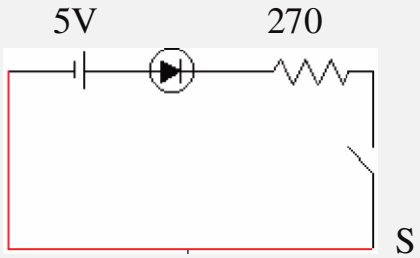


Figure.2

A seven segment LED display contains 7 LEDs. Each LED is called a segment and they are identified as (a, b, c, d, e, f, g) segments. Figure 3.

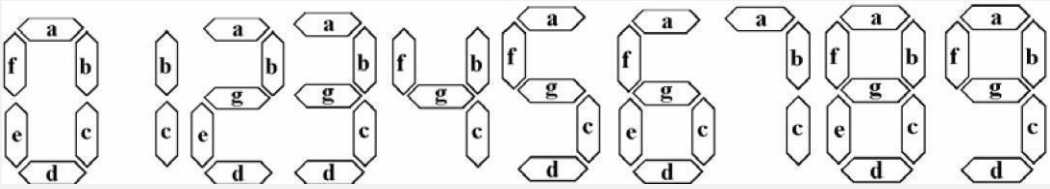


Figure 3. Digits represented by the 7 segments

The display has 7 inputs each connected to an LED segment. All anodes of LEDs are tied together and joined to 5 volts (this type is called common anode type). A limiting resistance network must be used at the inputs to protect the 7-segment from overloading.

BCD inputs are converted into 7 segment inputs (a, b, c, d, e, f, g) by using a decoder, as shown in Fig.4.

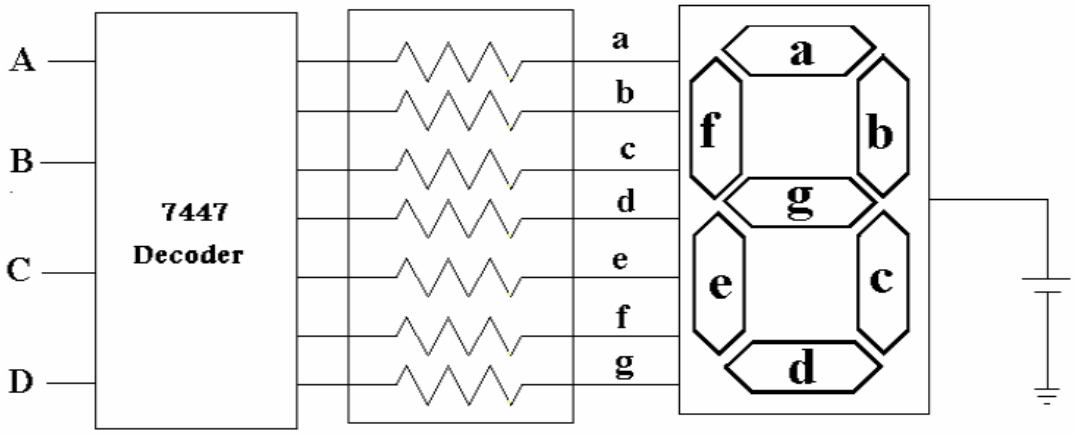


Figure. 4

A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2n output lines. The input to the decoder is a BCD code and the outputs of the systems are the seven segments a, b, c, d, e, f, and g. For further information and pin connections, consult the specification sheet for decoder and 7-segment units.

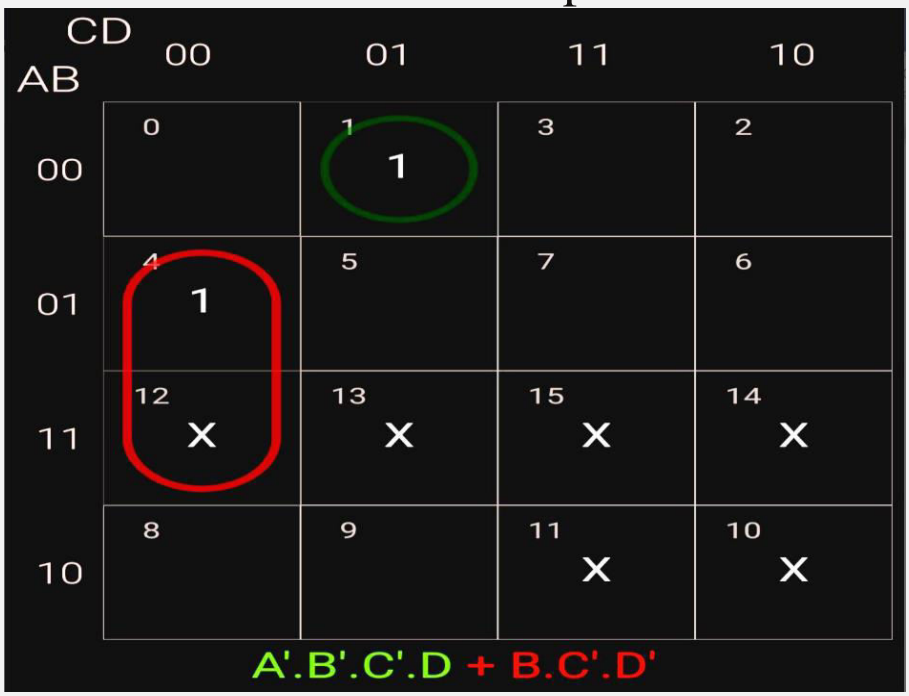
First design a combinational circuit which would simulate the decoder function for only the segment "a", of the display. This can be done in the following steps:

a) Write down the truth table with 4 inputs and 7 outputs (Table 2)

Dec	A	B	C	D		a	b	c	d	e	f	g
0	0	0	0	0		0	0	0	0	0	0	1
1	0	0	0	1		1	0	0	1	1	1	1
2	0	0	1	0		0	0	1	0	0	1	0
3	0	0	1	1		0	0	0	0	1	1	0
4	0	1	0	0		1	0	0	1	1	0	0
5	0	1	0	1		0	1	0	0	1	0	0
6	0	1	1	0		0	1	0	0	0	0	0
7	0	1	1	1		0	0	0	1	1	1	1
8	1	0	0	0		0	0	0	0	0	0	0
9	1	0	0	1		0	0	0	0	1	0	0

b) For only the output "a", obtain a minimum logic function. Realize this function using NAND gates and inverters only. For example if decimal 9 is to be displayed a, b, c, d, f, g must be 0 and the others must be 1 (For common anode type display units), if decimal 5 is to be displayed then a, f, g, c, d must be 0 and the others must be 1.

K Map for a

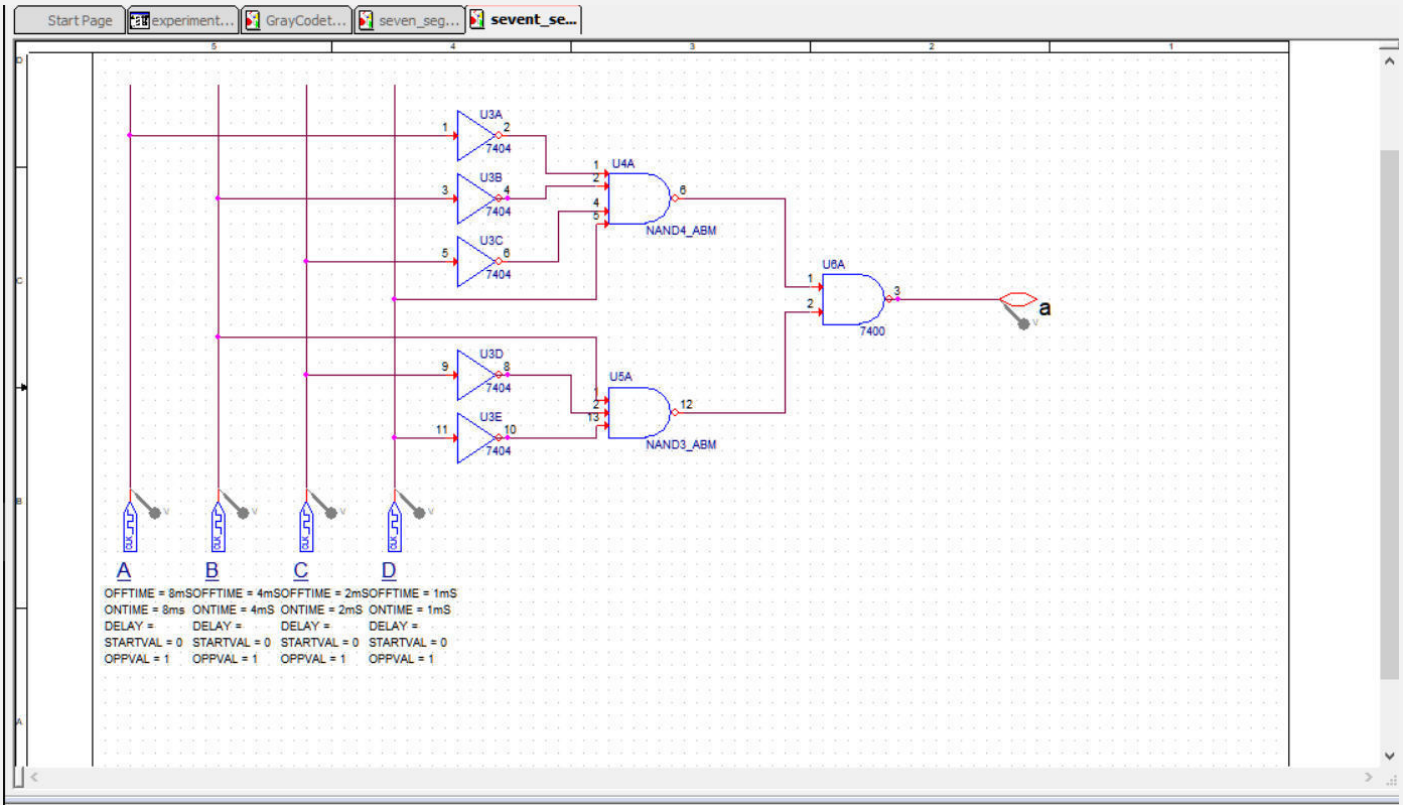


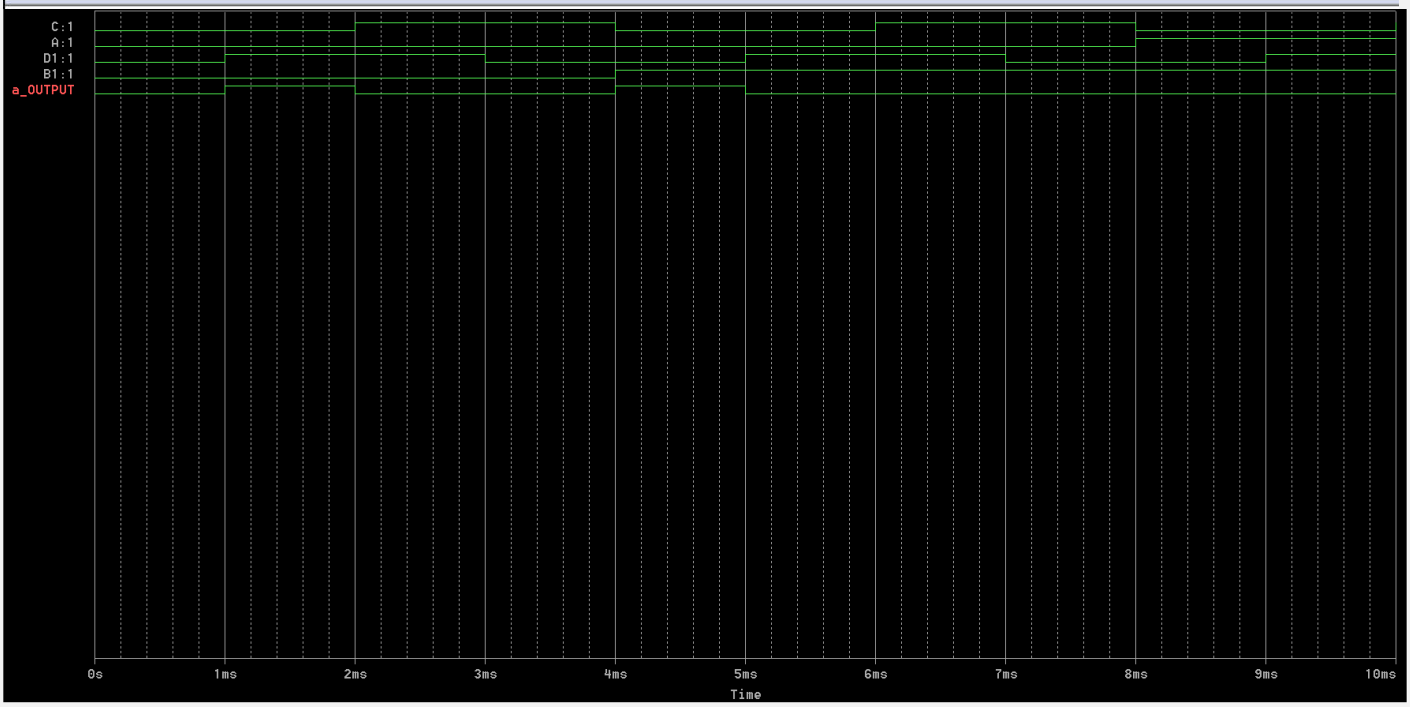
By Using De Morgan's Law,

a = A' B'C'D + BC'D'

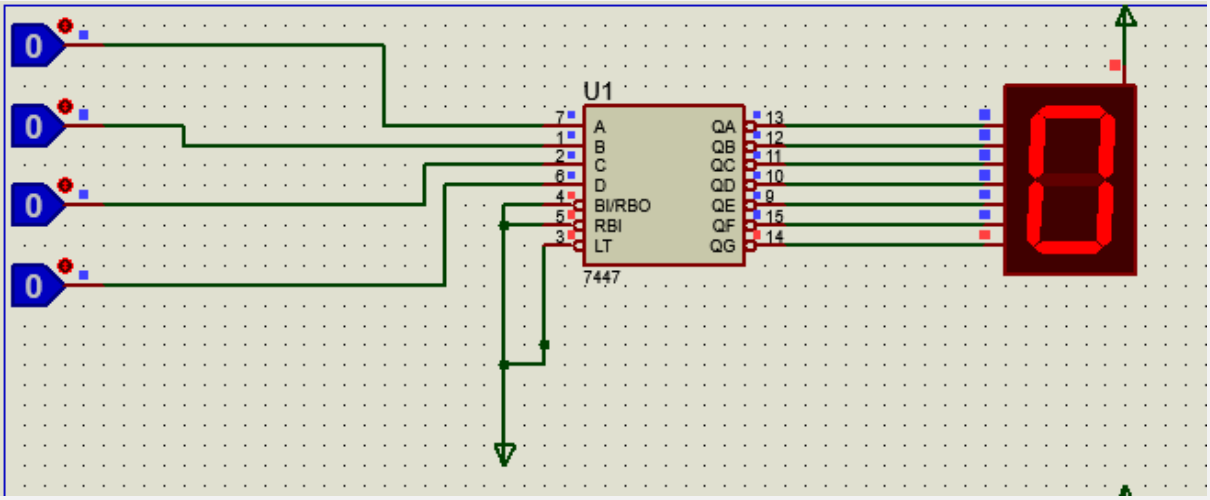
-= ((A'B'C'D)'. (BC'D')')'

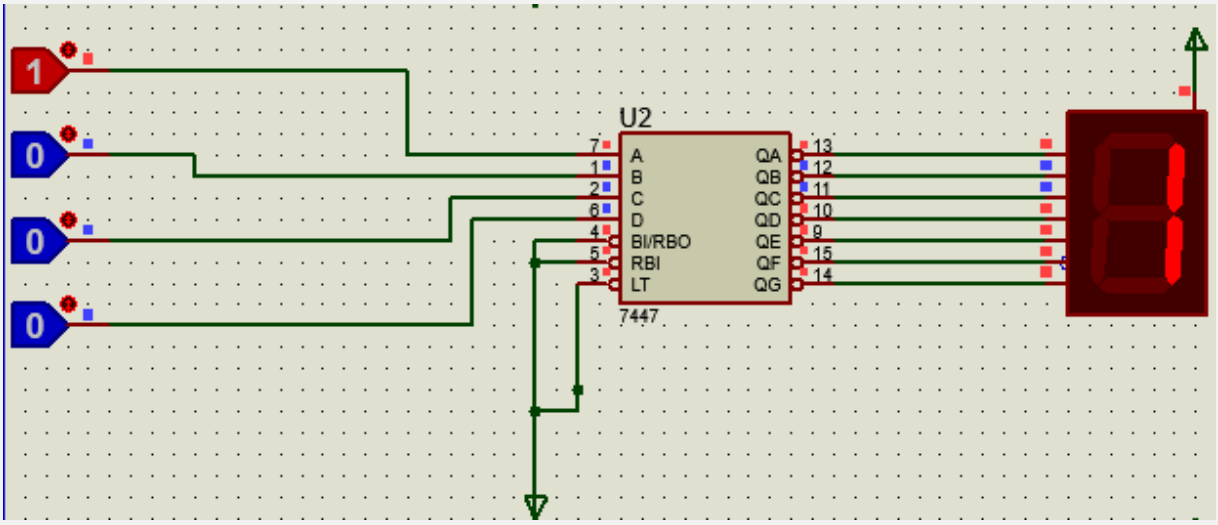
c) Connect the output "a" of your circuit to appropriate input of 7-segment display unit. By applying BCD codes verify the displayed decimal digits for that segment for "a" of the display.

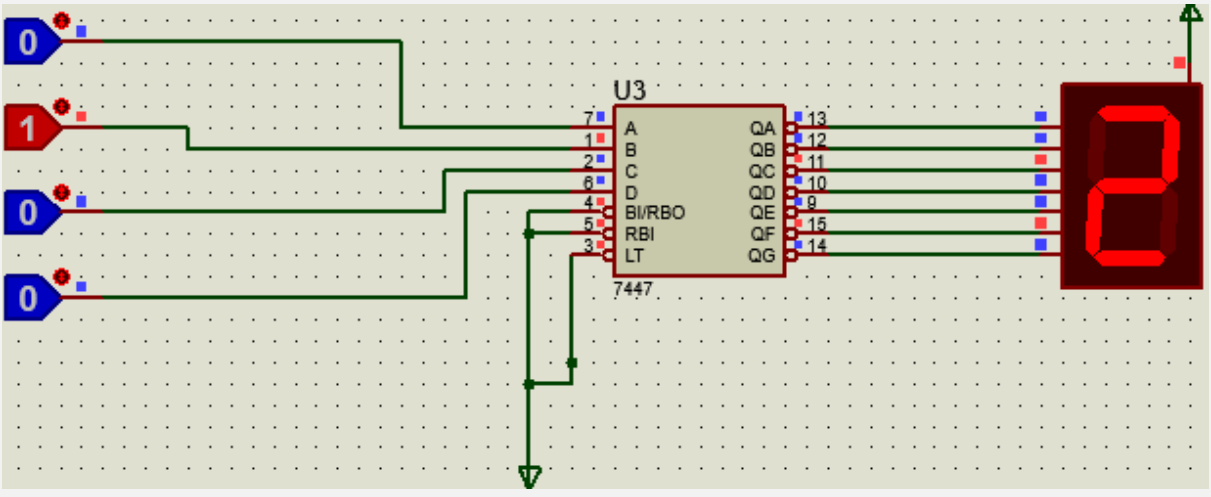




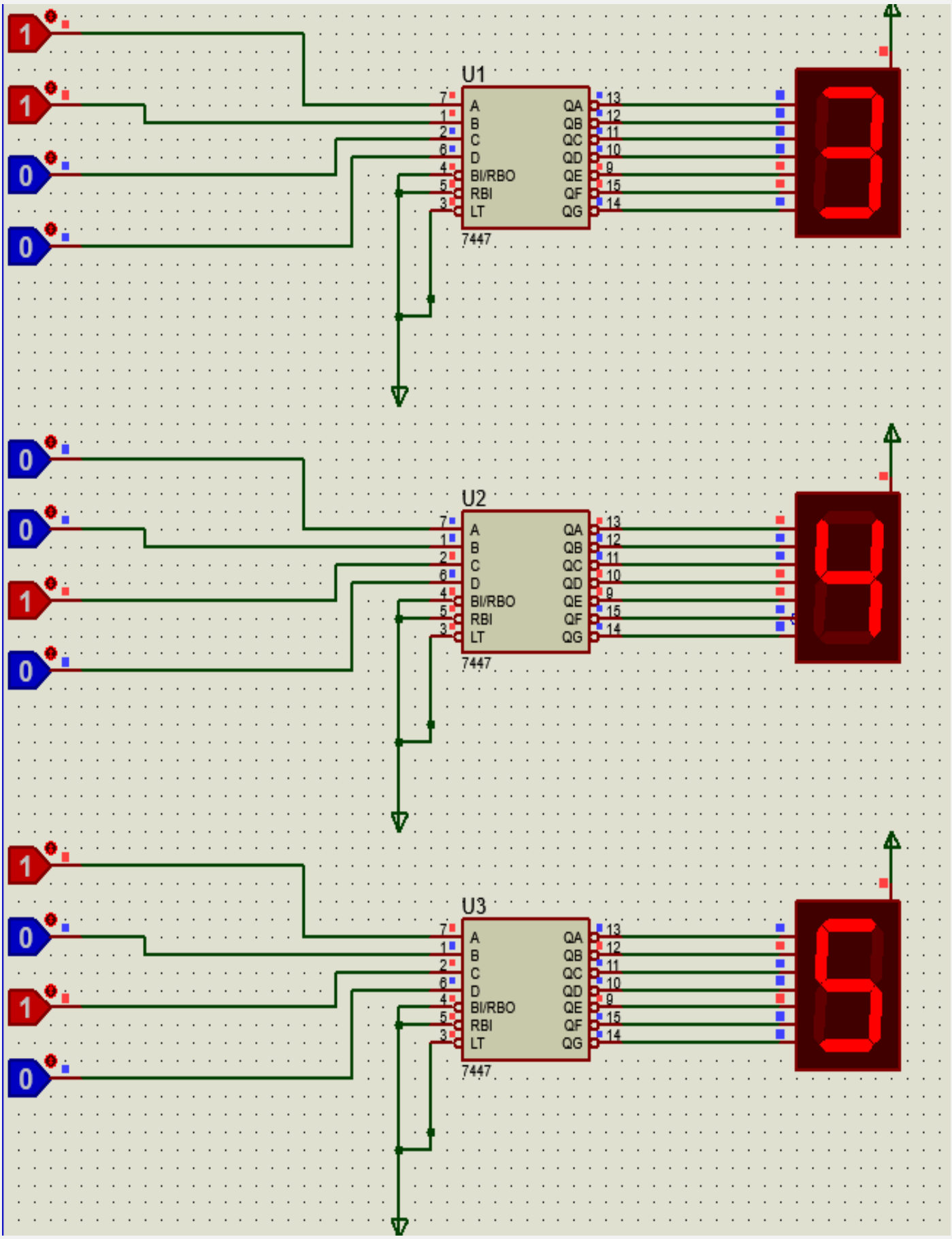
d) Replace your circuit by a decoder IC 7447 for all of the seven segments. Observe the display and record the segments that will light up for invalid inputs sequence.



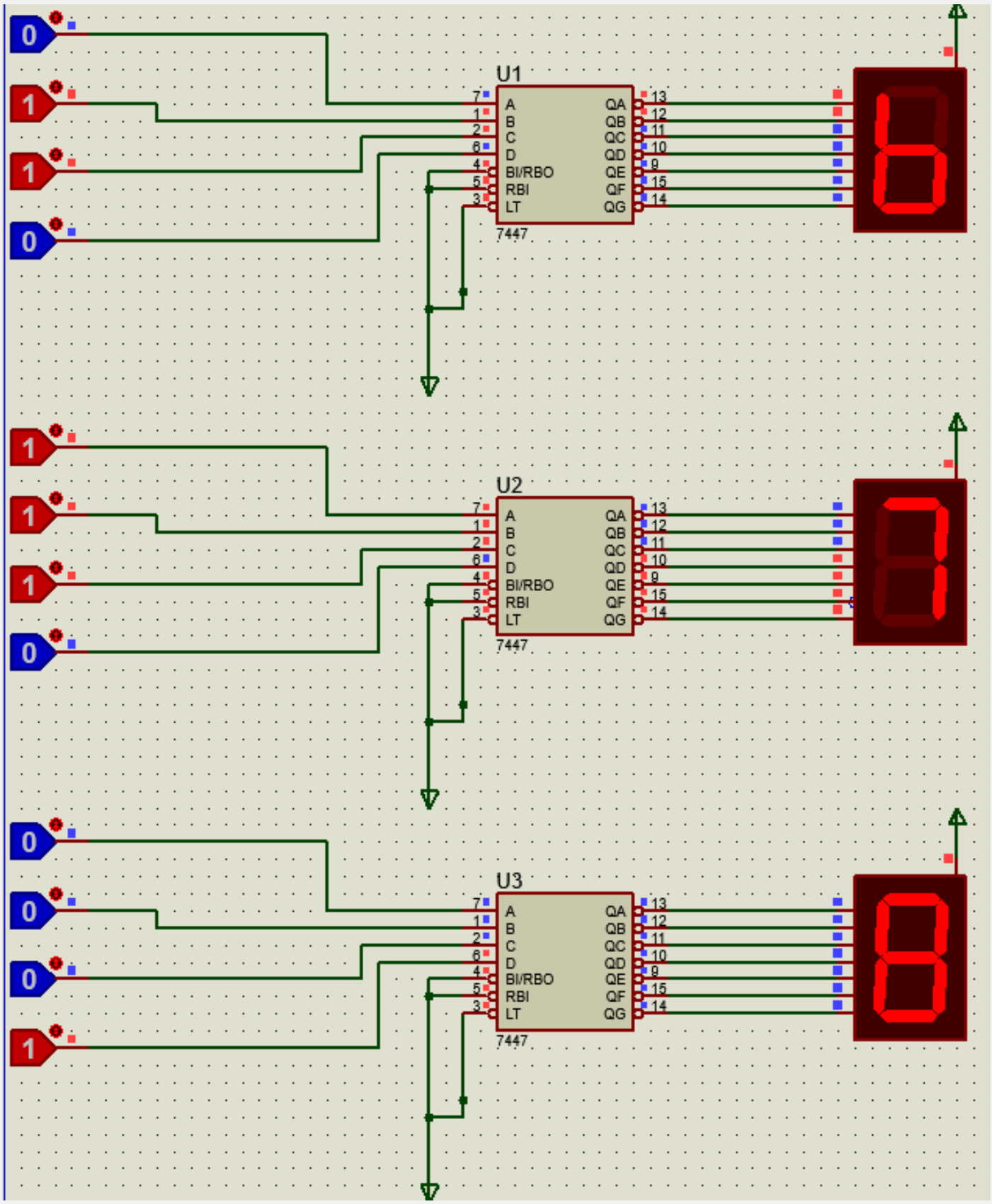


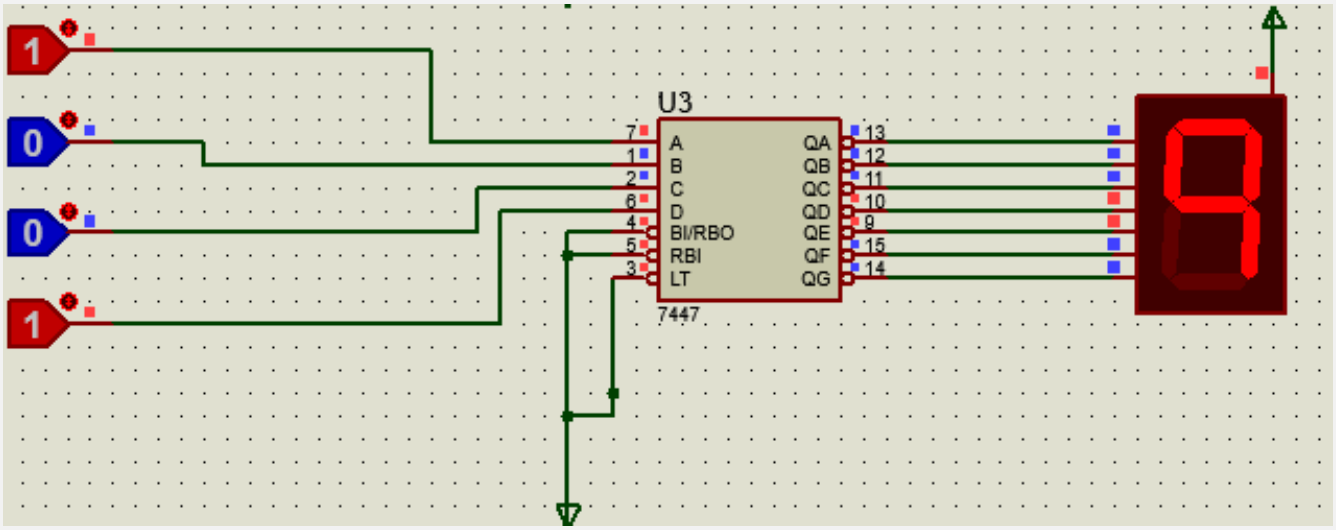


20BDS0405


U1
A
B
C
D
BI/RBO
RBI
LT
QA
QB
QC
QD
QE
QF
QG
7447
U2
A
B
C
D
BI/RBO
RBI
LT
QA
QB
QC
QD
QE
QF
QG
7447
U3
A
B
C
D
BI/RBO
RBI
LT
QA
QB
QC
QD
QE
QF
QG
7447

20BDS0405


U1
U2
U3
A
B
C
D
BI/RBO
RBI
LT
QA
QB
QC
QD
QE
QF
QG
7447
0
1
1
0
1
1
1
0
0
0
0
1



NOTE:

As observed from the simulation, the actual behavior of the 7447 IC circuit is slightly different from the expected ones.

In case of 6, which shows  instead of  and in case of 9, which shows  instead of  .

However, still they are sufficient to distinguish from the remaining numbers

e) Comment on the design if you don't want to see any digit for invalid input sequence.

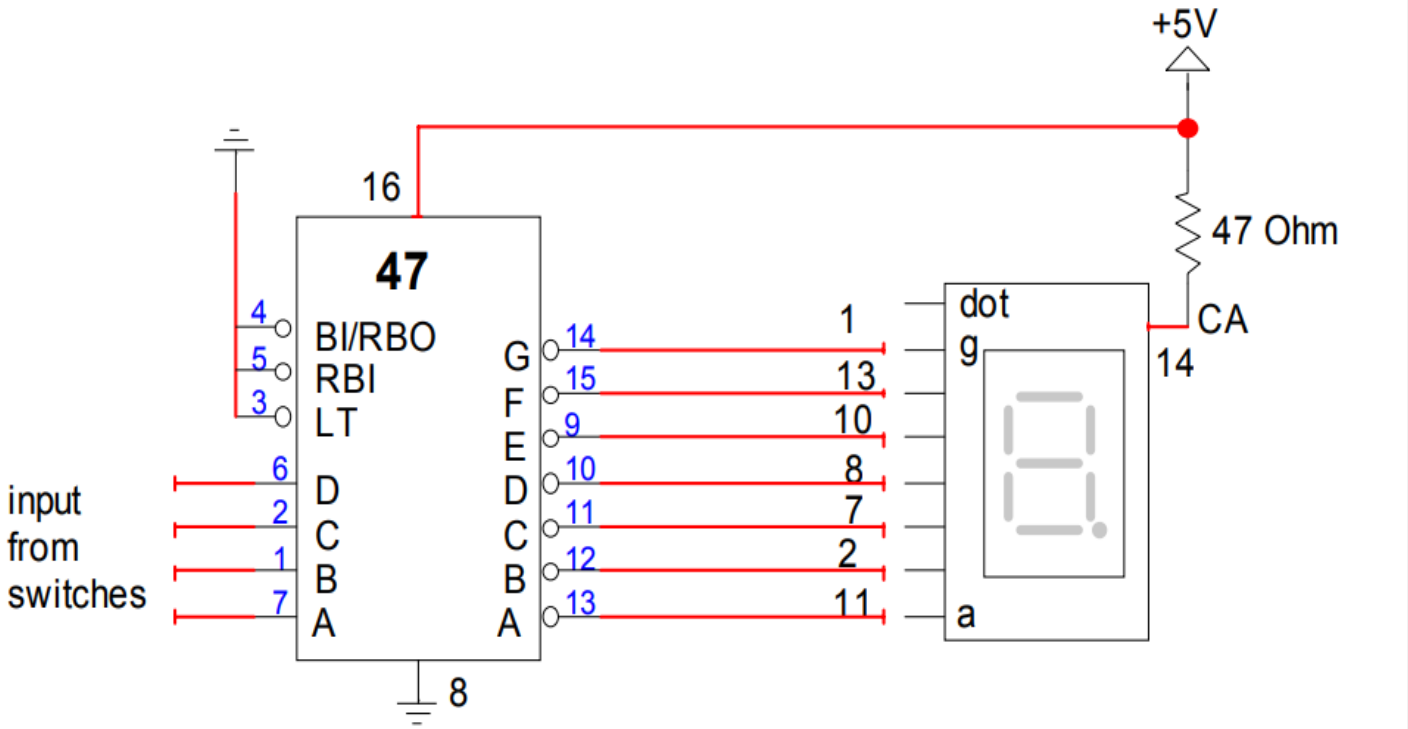
While implementing the 7 segment decoder circuit, the value above 9 are considered as don't care and hence can have really random output that we don't care. However, if we want to get rid of that output, we can have the following ways:

I. Limit the simulation time to only 10 units of time(0-9), instead of 16 units of time(0-16)

II. Add a combinational block before the decoder circuit such that it sets all the outputs from the clock to 0.

Table 2

Dec.	BCD				Outputs						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	0	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0
10	1	0	1	0	X	X	X	X	X	X	X
11	1	0	1	1	X	X	X	X	X	X	X
12	1	1	0	0	X	X	X	X	X	X	X
13	1	1	0	1	X	X	X	X	X	X	X
14	1	1	1	0	X	X	X	X	X	X	X
15	1	1	1	1	X	X	X	X	X	X	X



BCD-to-Seven Segment Decoder and 7-segment display

Note: In an actual 7-segment display the dot is on the left

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Digital Logic Design
CSE1003 LAB

5. Adder/Subtractor/Comparator

6. Design with Multiplexers

Experiment – 5 & 6

Bimal parajuli (20BDS0405)

DIGITAL LOGIC DESIGN CSE1003

Exp #5-ADDERS, SUBTRACTORS AND MAGNITUDE COMPARATORS

Objectives:

- To construct and test various adders and subtractor circuits.
- To construct and test a magnitude comparator circuit.

Apparatus:

- IC type 7486 quad 2-input XOR gates
- IC type 7408 quad 2-input AND gates
- IC type 7404 HEX inverter
- IC type 7483 4-bit binary adder
- IC type 7485 4-bit magnitude comparator.

Softwares Used:

- ORCAD CAPTURE CIS Lite
- Logic Works 5
- Proteus 8 professional

a) *Addition:*

IC type 7483 is a 4-bit binary adder with fast carry. The pin assignment is shown in Fig 1. The two 4-bit input binary numbers are A_1 through A_4 and B_1 through B_4. The 4-bit sum is obtained from S_1 through S_4. C_i is the input carry and C_o the out carry. This IC can be used as an adder-subtractor as a magnitude comparator.

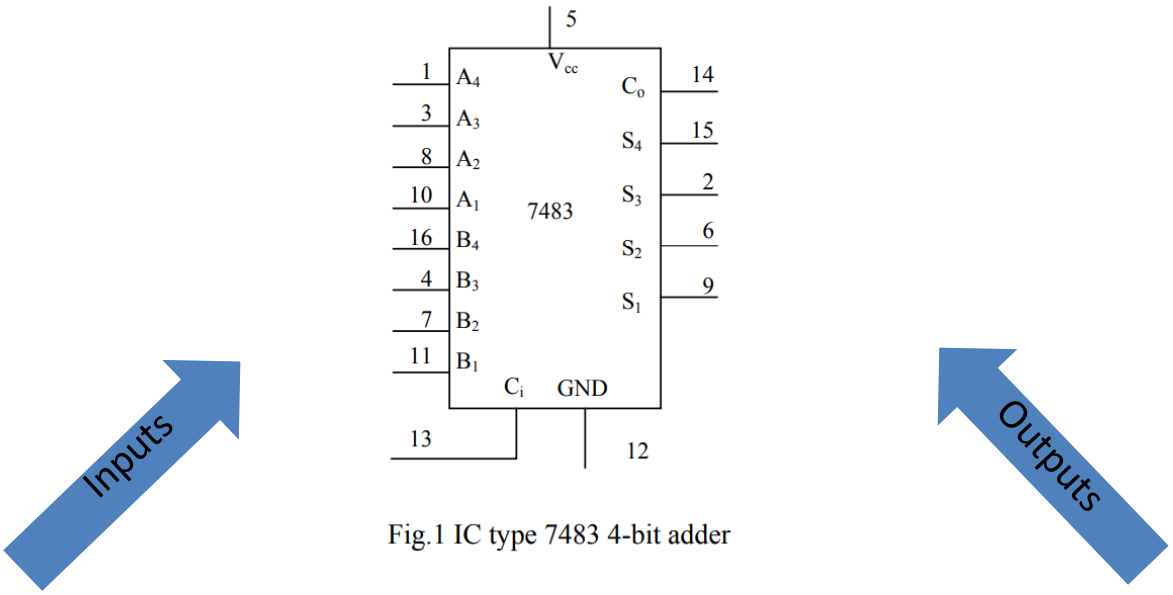
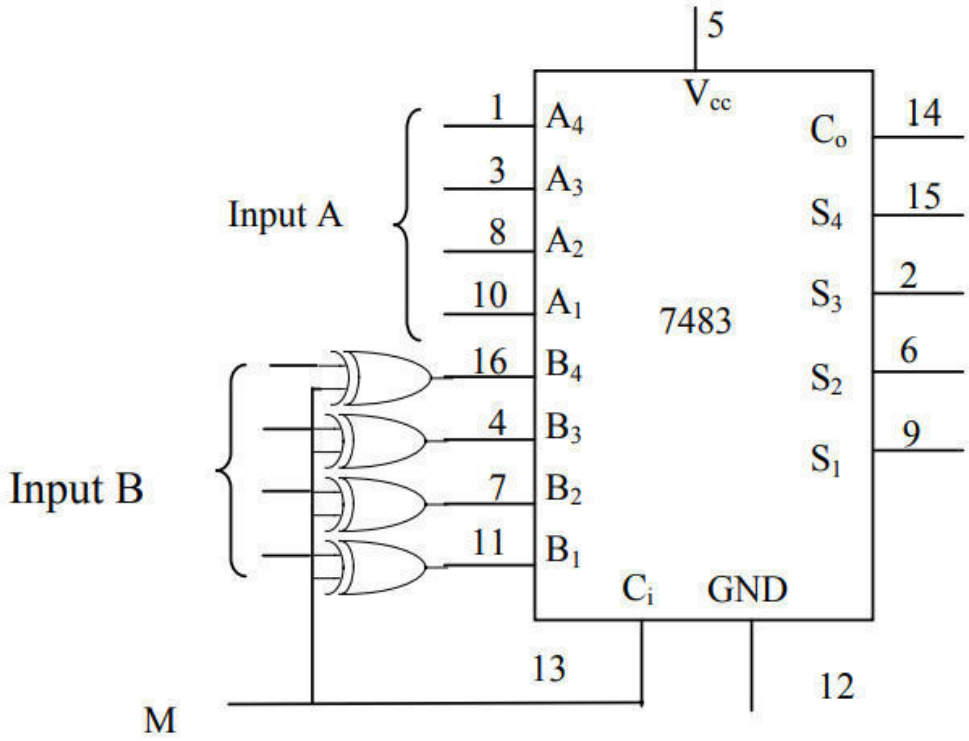


Fig.1 IC type 7483 4-bit adder

b) *Subtraction:*

The subtraction of two binary numbers can be done by taking the 2's complement of the subtrahend and adding it to the minued. The 2's complement can be obtained by taking the 1's complement and adding 1.

To perform **A - B**, we complement the four bits of B, add them to the four bits of A, and add 1 to the input carry. This is done as shown in **Fig 2.**



M = 0 for add and M = 1 for subtract

Fig. 2 4-bit adder/subtractor

Four **XOR** gates **complement the bits of B** when the mode select **M = 1** (because x ⊕ 1=x') and **leave the bits of B unchanged when M = 0** (because x ⊕ 0=x) thus, when the mode select M is equal to 1, the input carry C_i is equal to 1 and the sum output is A plus the 2's complement of B. When M is equal to 0, the input carry is equal to 0 and the sum generates A + B.

c) *Magnitude comparison*

The comparison of two numbers is an operation that determines whether one number is greater than, equal to, or less than the other number.

The **IC 7485** is a **4 bit magnitude comparator**. It compares two 4-Bit binary numbers (labeled as A&B) generates an output of 1 at one of three outputs labeled A > B, A < B, A = B. Three inputs are available for cascading comparators. See Fig.3.

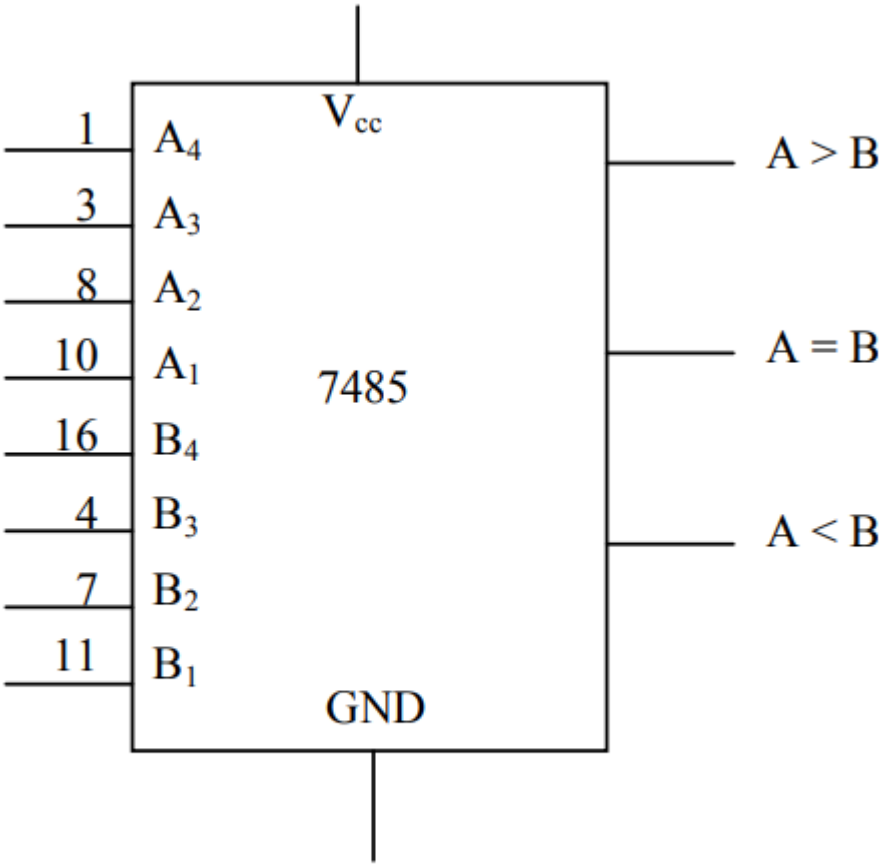


Fig. 3 4-bit magnitude comparator

Procedure:

a) ***Design using software a half adder circuit using only XOR gates and NAND gates. Then during the Lab construct the circuit and verify its operation.***

Following is the truth table for the Half Adder circuit:

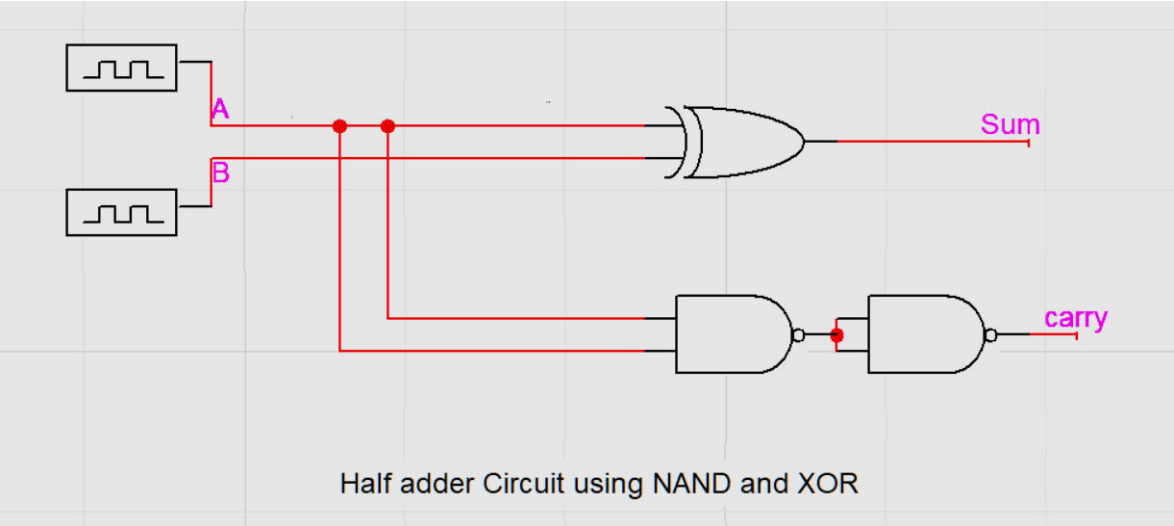
Input(A)	Input(B)	Carry (C)	Sum(S)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Here, C= A.B

$C = \overline{(\overline{A.B})}$ [Using Demorgan's law]

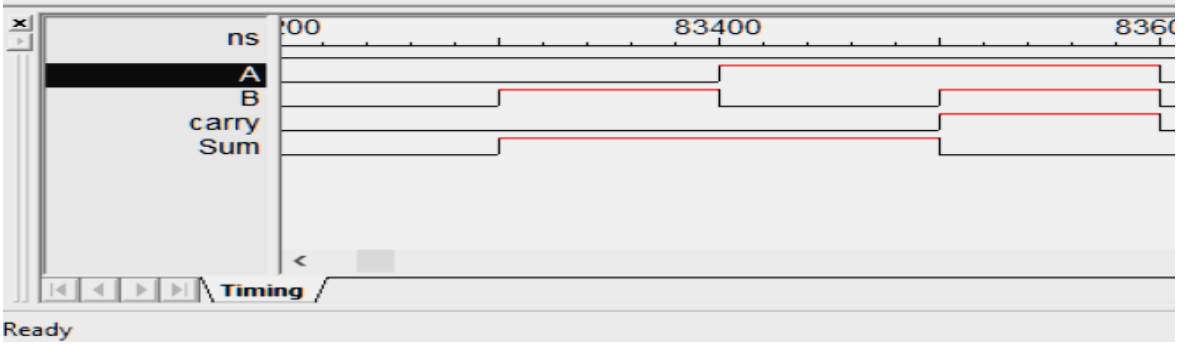
And, S=A XOR B

The corresponding logical circuit OF HALF ADDER is:



Half adder Circuit using NAND and XOR

The result of Simulation is:



b) ***Design using software a full adder circuit using only XOR gates and NAND gates. Then during the Lab construct the circuit and verify its operation.***

Following is the truth table for the Half Adder circuit:

Input(A)	Input(B)	Carry-In (C)	Carry Out (C_o)	Sum(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

Now,

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

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

From above Kmap:

$C_o = \bar{A}.B.C + A.\bar{B}.C + A.B.\bar{C} + A.B.C$

$= A.B + B.C + A.C$ [Solved from Karnaugh Map]

$= \overline{(\overline{A.B}.\overline{B.C}.\overline{C.A})}$ [De Morgan's Law]

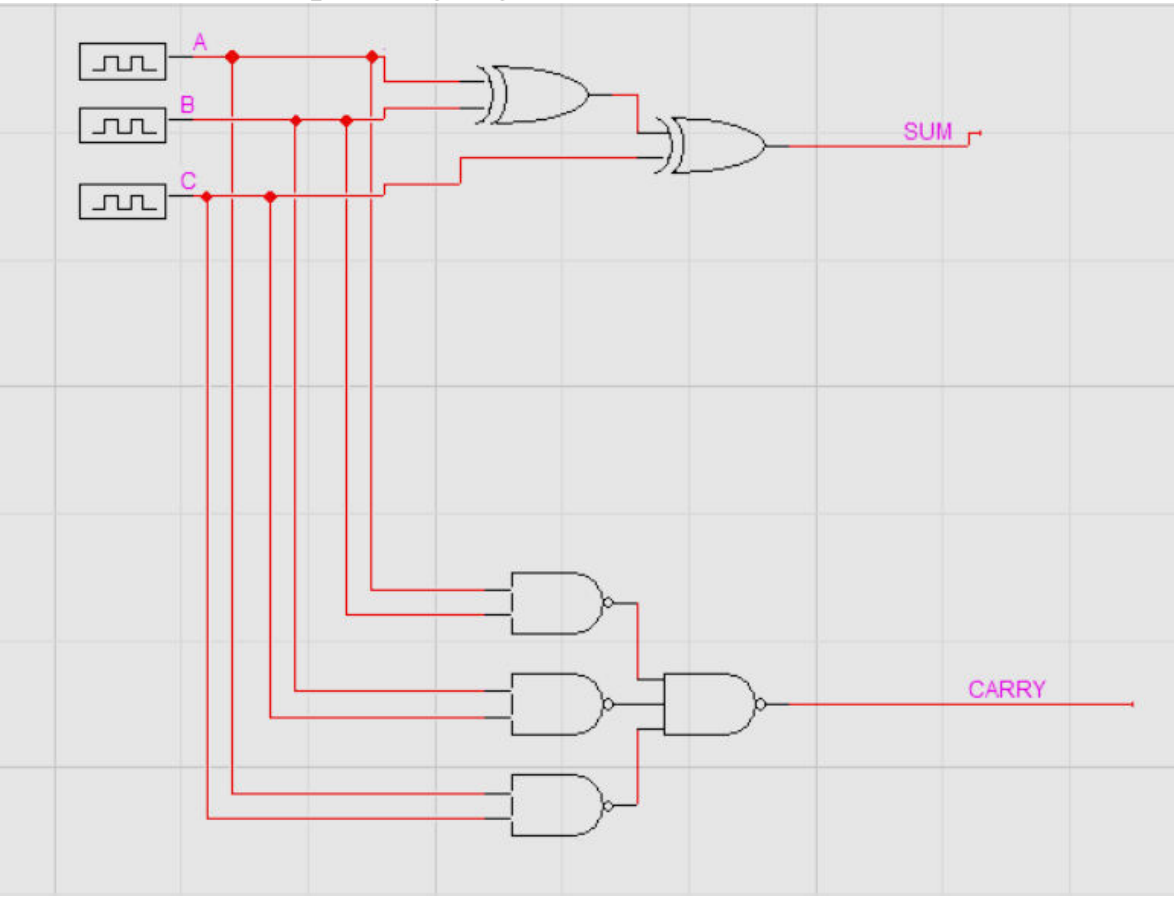
$S = \bar{A}.\bar{B}.C + \bar{A}.B.\bar{C} + A.\bar{B}.\bar{C} + A.B.C$

$= \bar{A}$ (B xor C) + A (B xnor C)

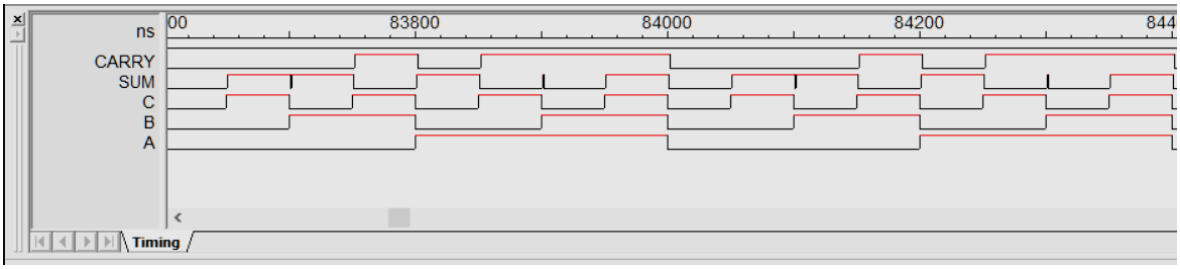
$= \bar{A}$ (B xor C) + $A\overline{(B \text{ xor } C)}$

= A xor B xor C

The corresponding logical circuit of FULL ADDER is:



The result of Simulation is:



c) ***Use IC 7483 to add the two 4-bit numbers A and B shown in Table1. In softwares, select the chip 74-83 and use Binary switches for the bits of the two numbers and the input carry and use Binary Probe for the sum and carry out.***

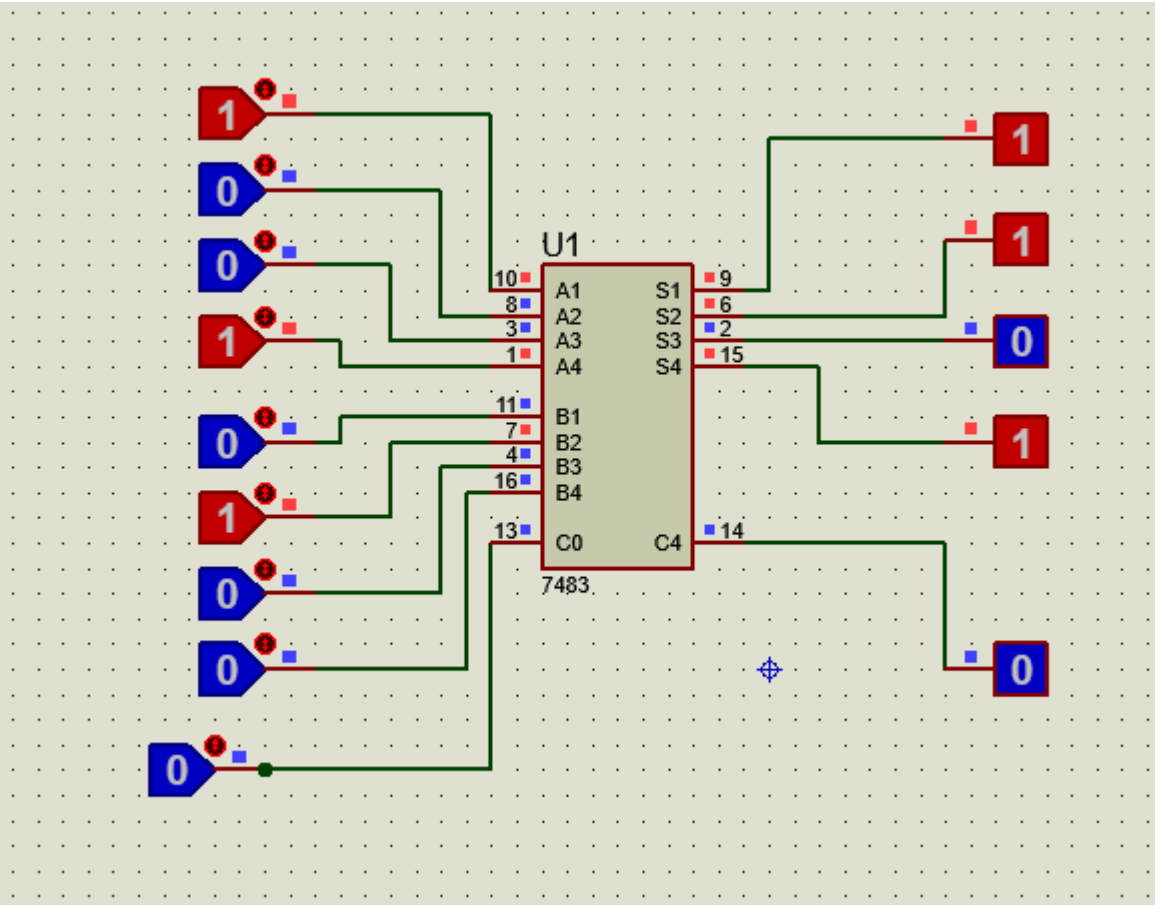
Input carry Ci is taken as logic 0. Show that if the input carry is 1, it adds 1 to the output sum.

In the Lab use switches S1-1 to S1-8 for the two numbers and use the SPDT S2 for the input carry Ci. For sum and carry out, use LED-1 to LED-5.

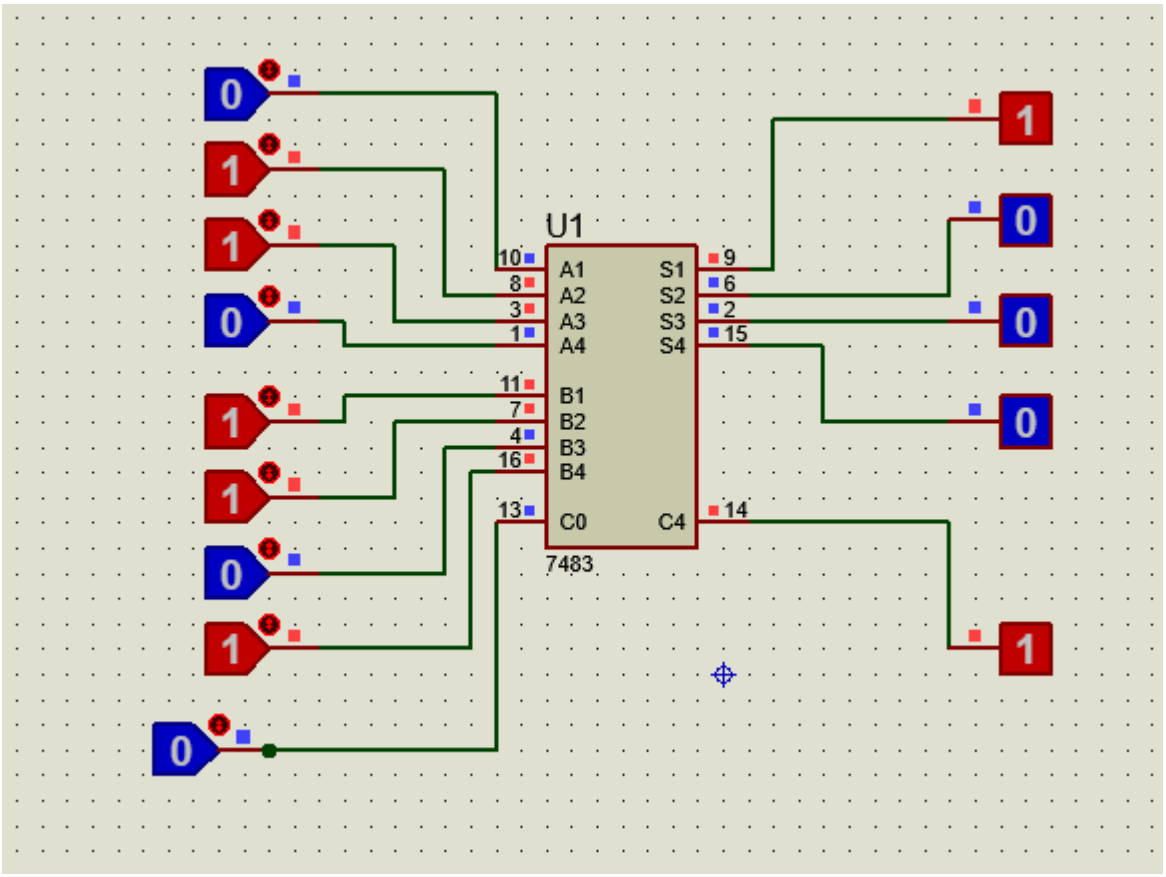
Table 1.

A4	A3	A2	A1	B4	B3	B2	B1	Sum				Carry out
1	0	0	1	0	0	1	0	1	0	1	1	0
0	1	1	0	1	0	1	1	0	0	0	1	1
1	1	0	0	1	0	1	0	0	1	1	0	1

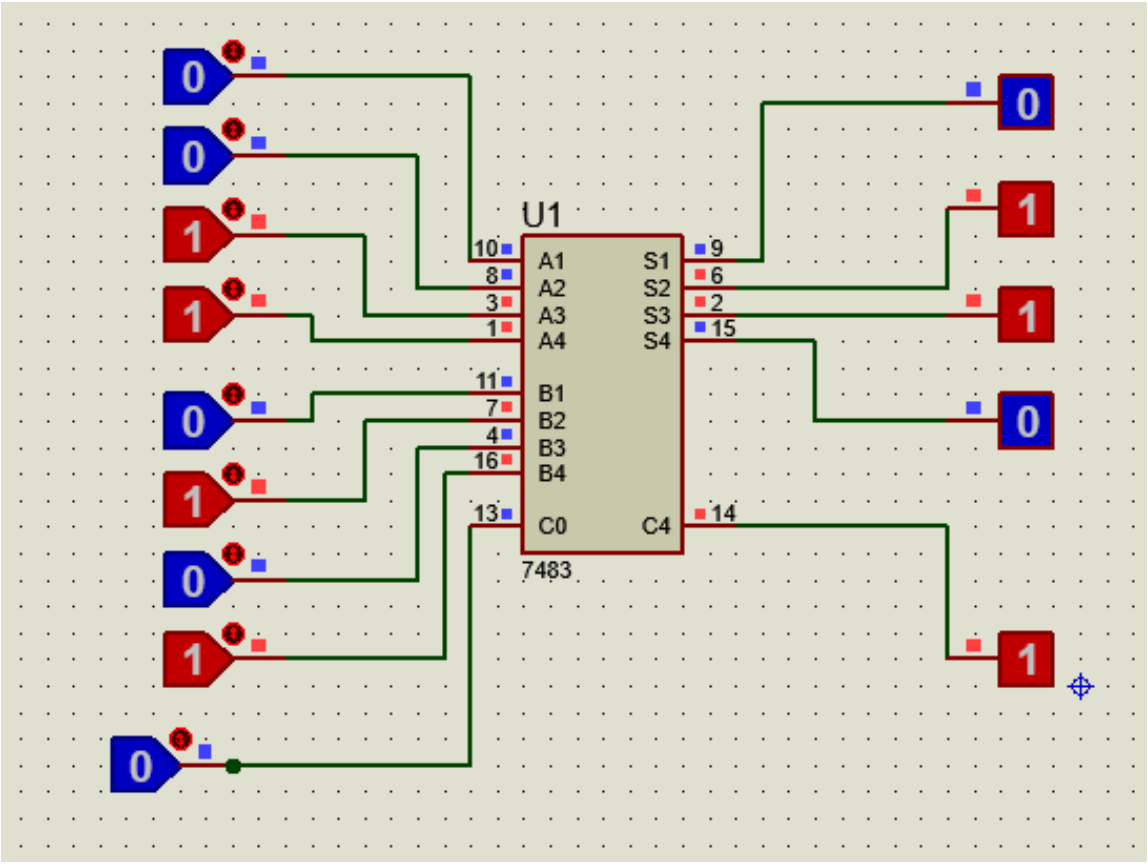
1 0 0 1 + 0 0 1 0



0 1 1 0 + 1 0 1 1



1 1 0 0 + 1 0 1 0

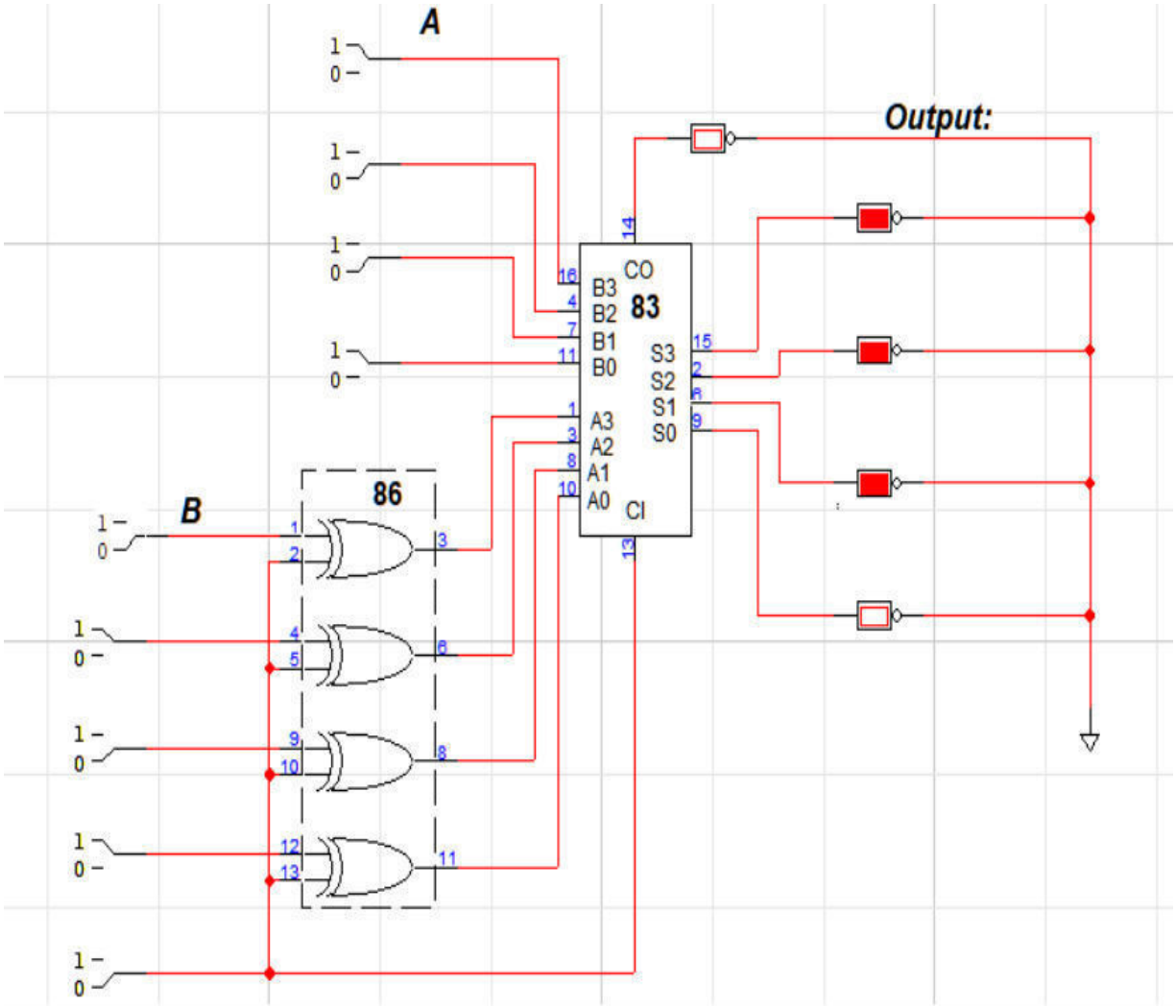


d) ***Connect the adder-subtractor circuit as shown in Fig 2. Perform the following operations and record the values of the output sum and the output carry Co.***

Table 2.

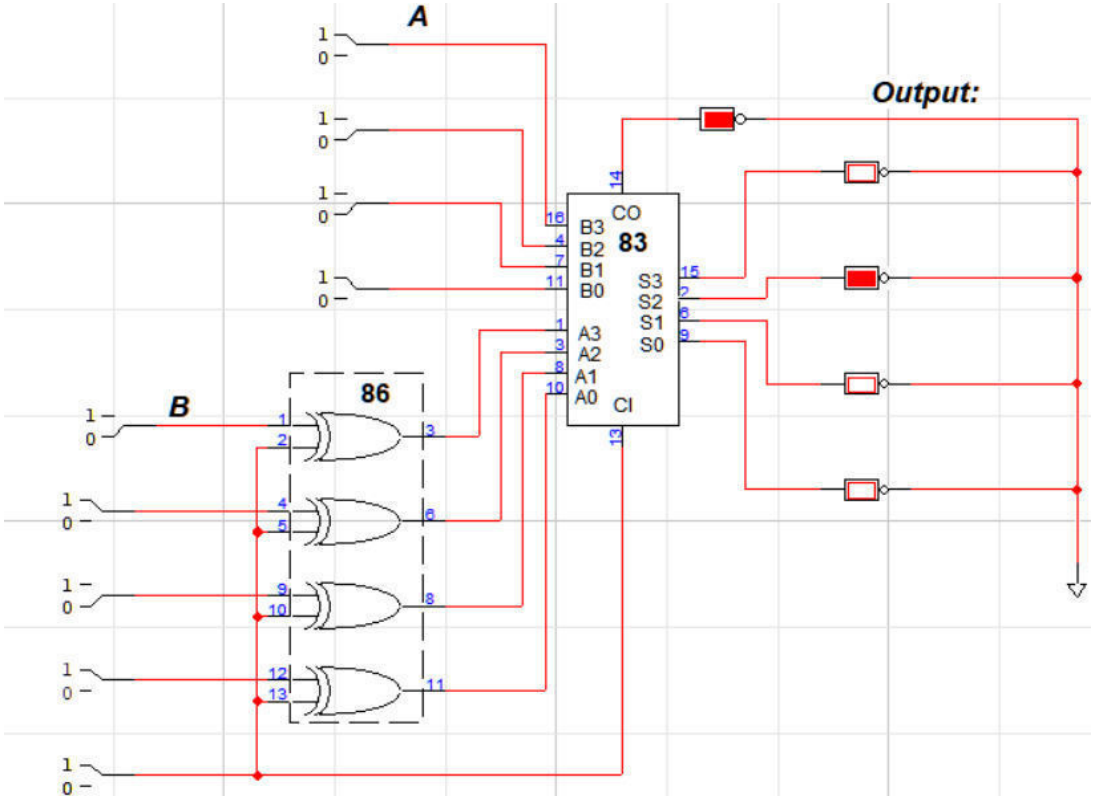
Decimal A B	Carry Out	Output sum			
9 + 5	0	1	1	1	0
9 - 5	0	0	1	0	0
9 + 13	1	0	1	1	0
9 - 9	0	0	0	0	0
10 + 6	1	0	0	0	0
6 - 10	0	1	1	0	0

9 + 5 = 14(01110):

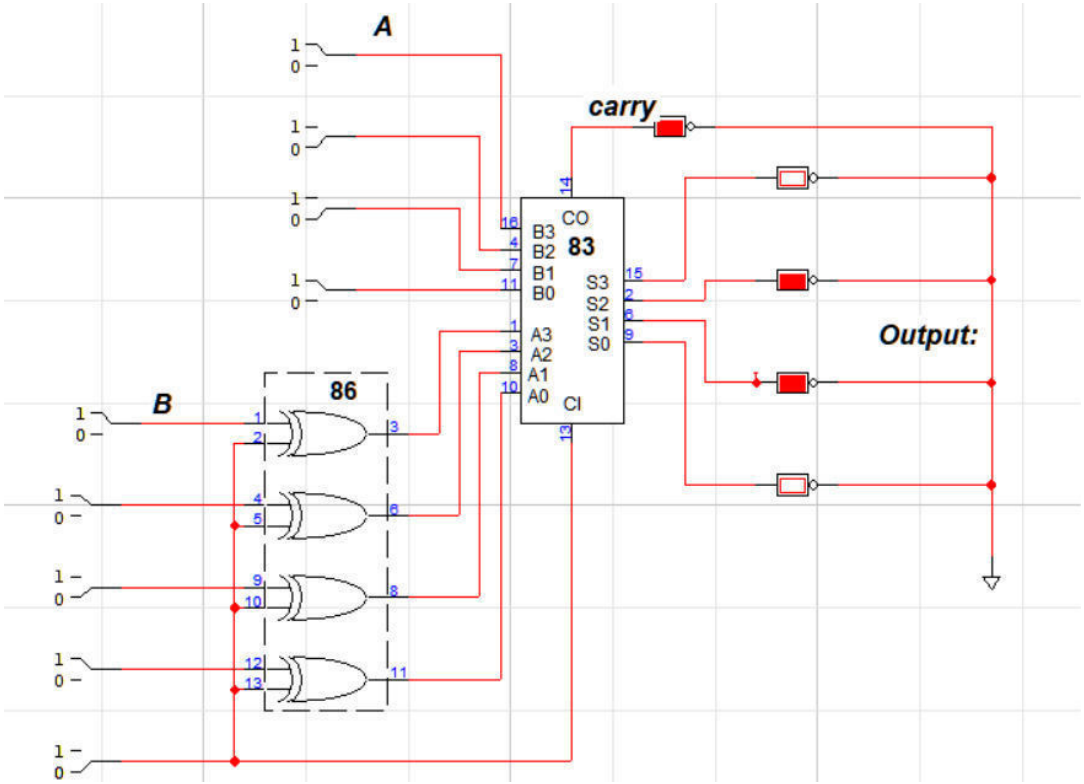


Bimal parajuli (20BDS0405)

9 - 5 = 4 (10100):

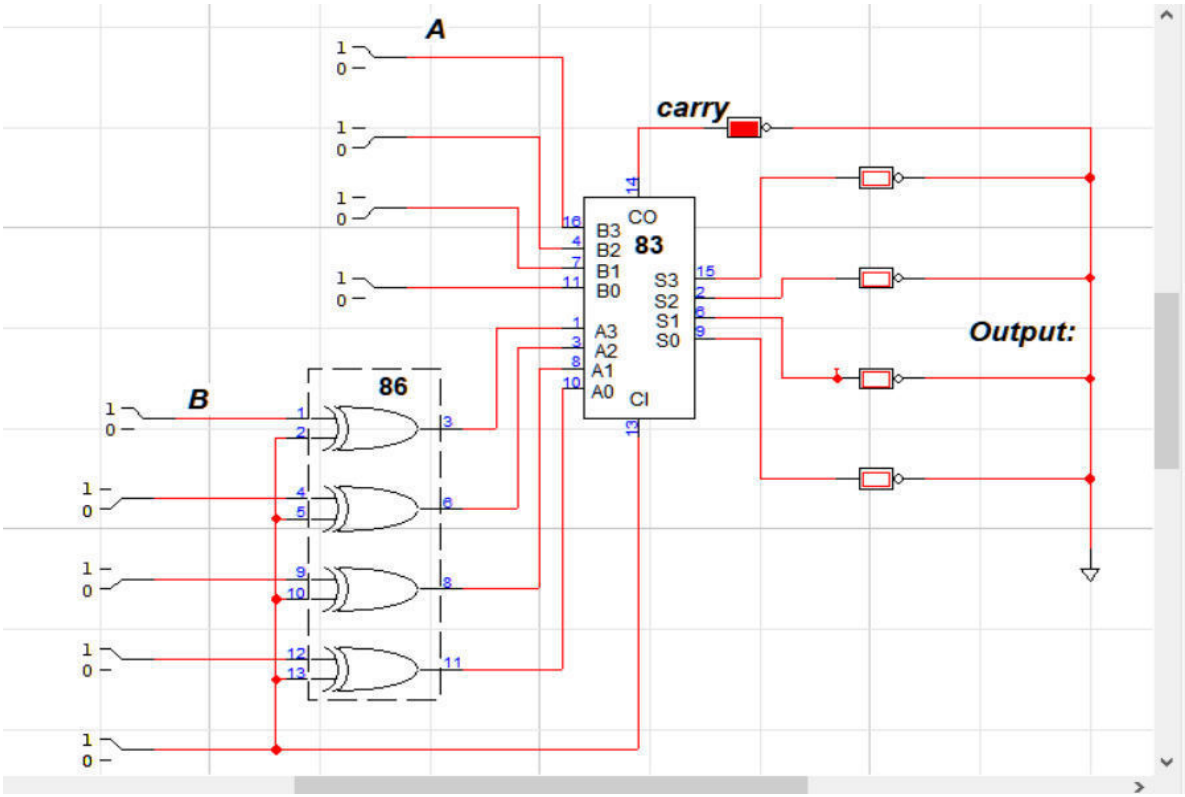


9 + 13 = 22 (10110):

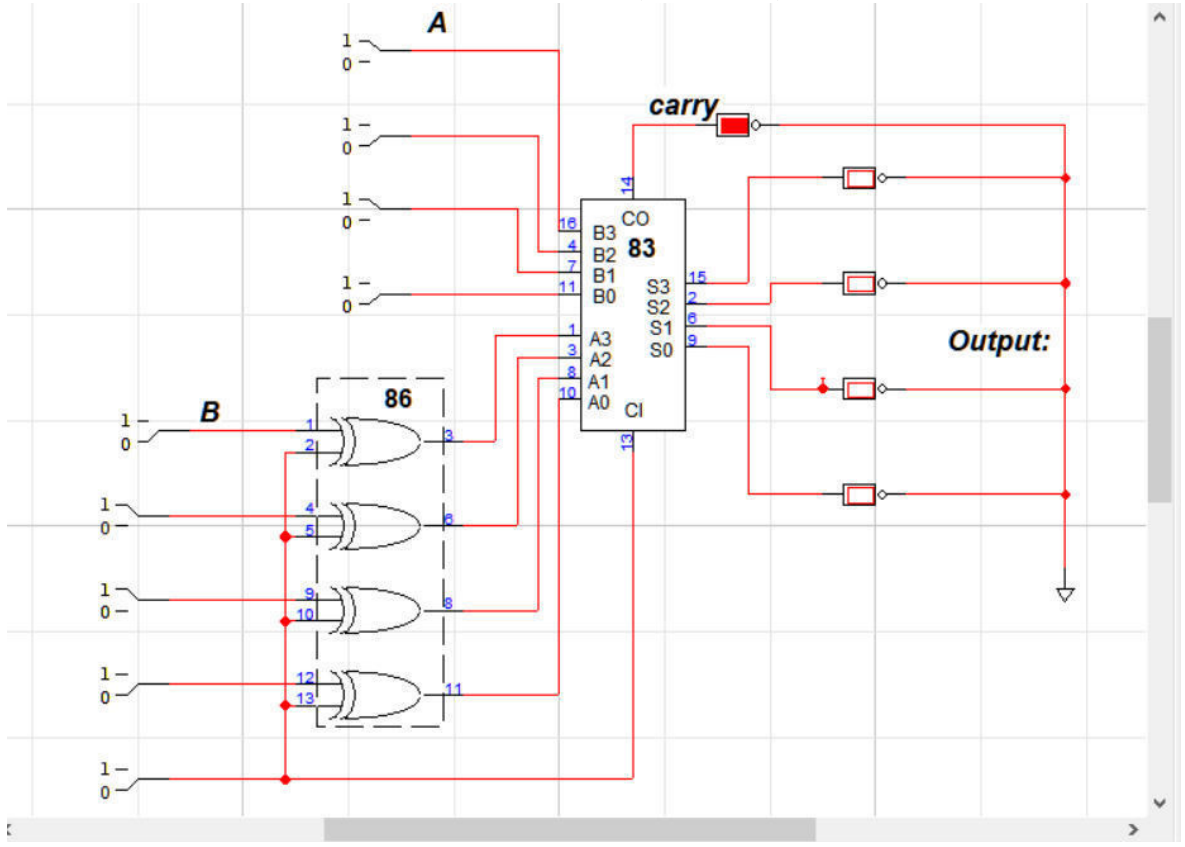


Bimal parajuli (20BDS0405)

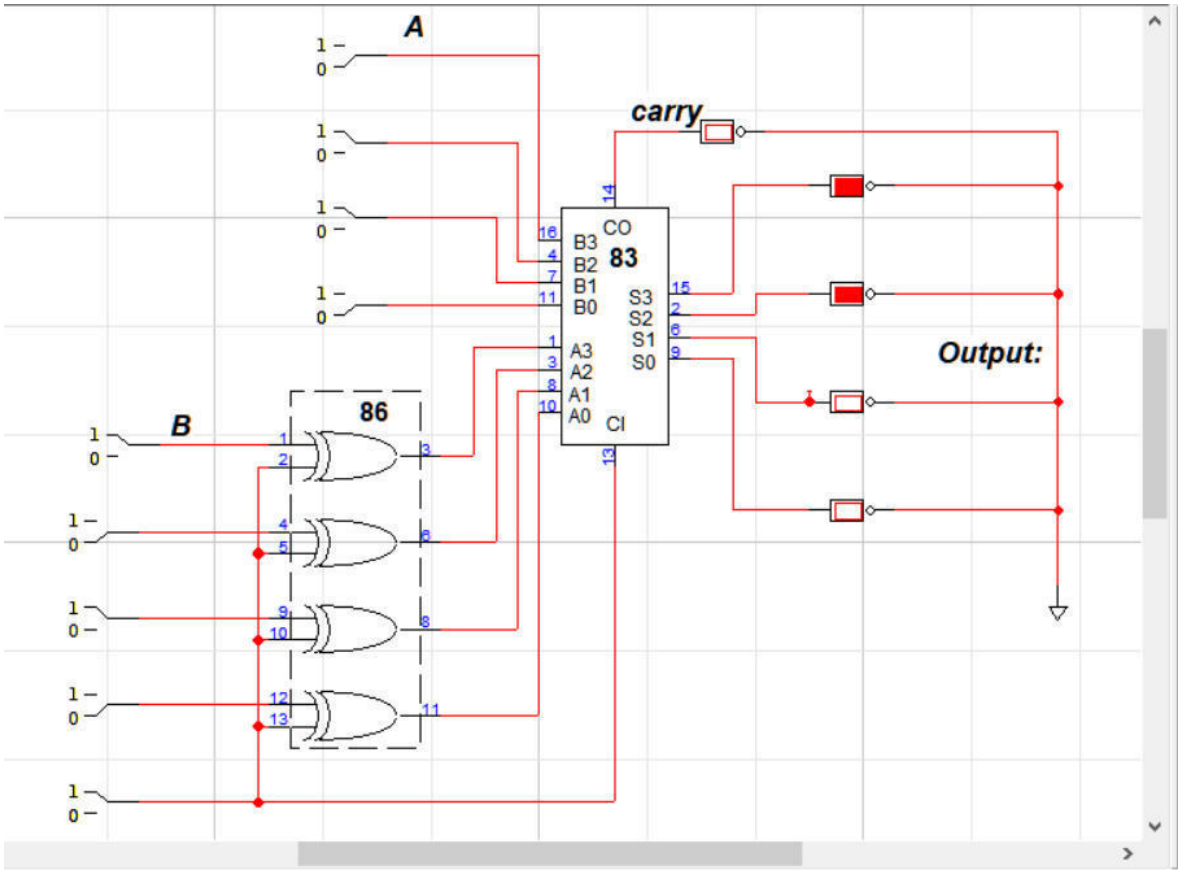
9 - 9 = 0 (10000):



10 + 6 = 16(10000):



6-10 = -4 (01100)

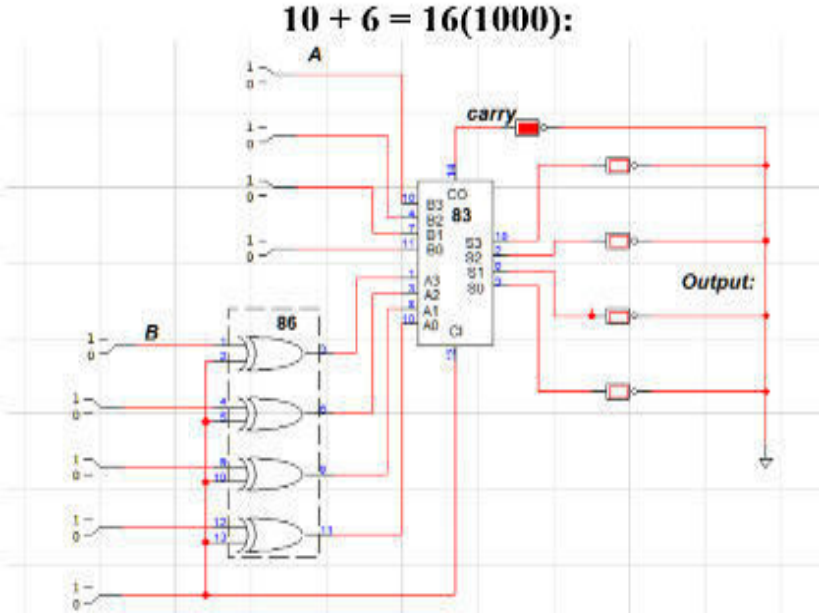


- ***Show that Co =1 when sum exceeds 15.***

In the demonstration of 6 + 10 = 16, we can see that the output carry out C_0 =1.

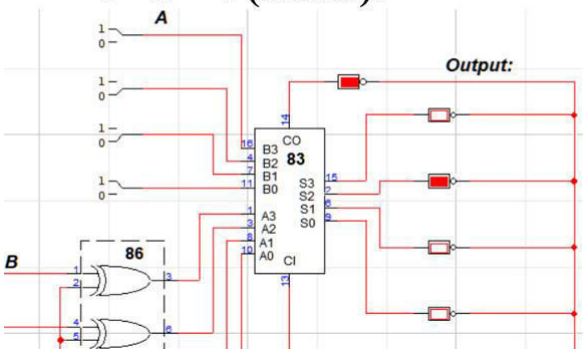
Here, since, the output of the sum is greater than 15, it needs 5 digits to be represented in equivalent binary notation and hence the carry out needs to be set high as overflow.

10 + 6 = 16(1000):



- ***Comment on sum and Co for the subtraction operations when A > B and A < B.***

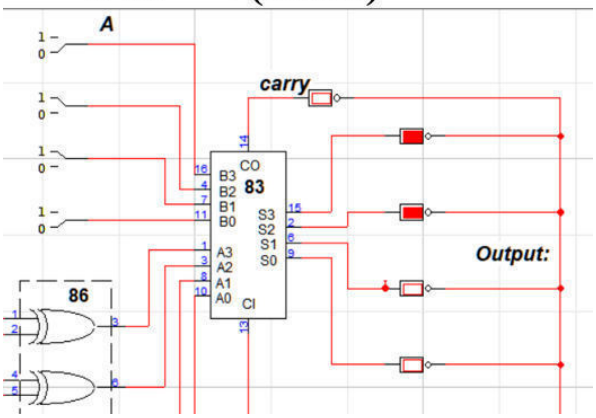
9 - 5 = 4 (10100):



When A> B, the Carry out is set high and the sum is exactly the expected output and can be accepted as the difference of two supplied numbers.

Carry => 1
Sum => can be accepted as it is

6-10 = -4 (01100)



However,
When A < B, the Carry out is set low and the sum is the twos complement of the exact expected difference. For example in 6-10, the carry out is 0, so the difference must be negative, the sum is 1100. By taking it's 2's complement, we get the required answer i.e. 100 or 4

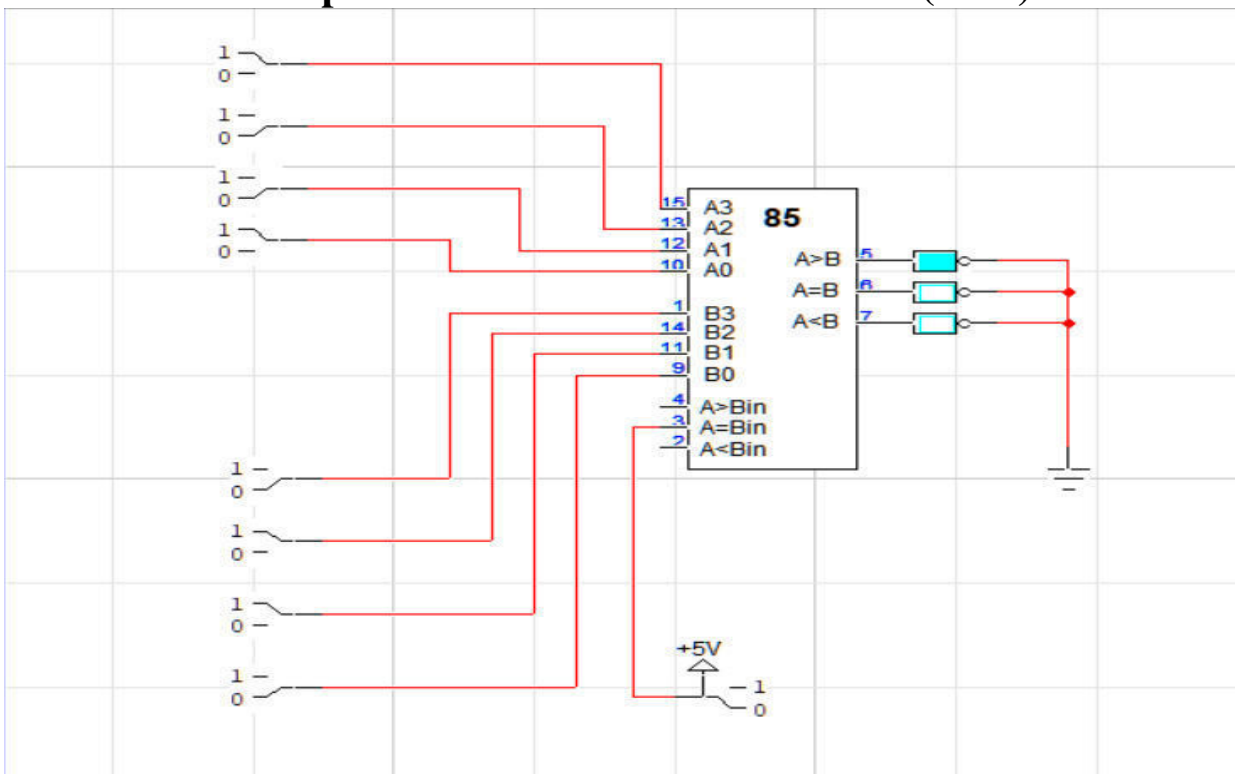
Carry => 1
Sum => must be calculated 2's complement.

e) ***Use IC7485 to compare the following two 4 bit numbers A and B. Record the outputs in table 3. Note that in softwares, you need to connect (A = B) input to logic 1 (as an indication that previous stages are equal in multi-digit numbers) for correct results while this is not necessary for the hardware.***

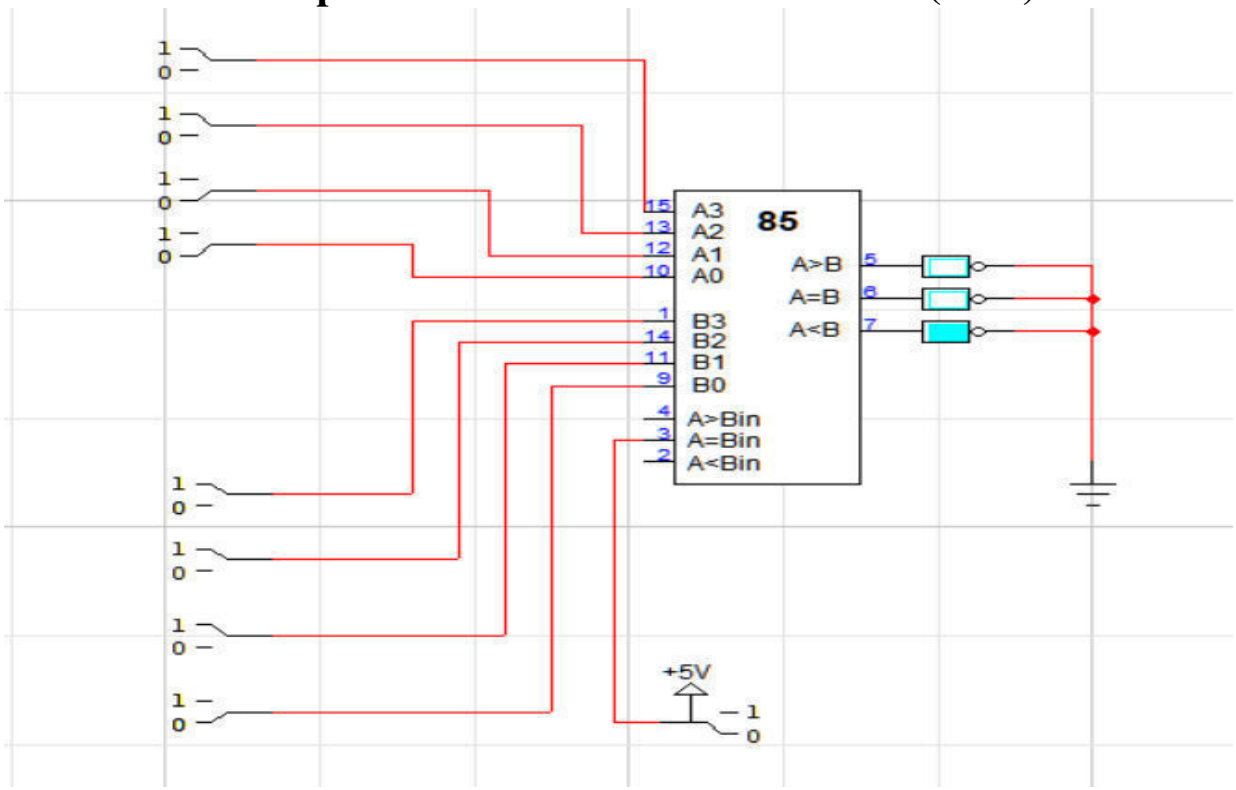
Table 3.

A	B	Outputs
1001	0110	A>B
1100	1110	A<B
0011	0101	A<B
0101	0101	A=B

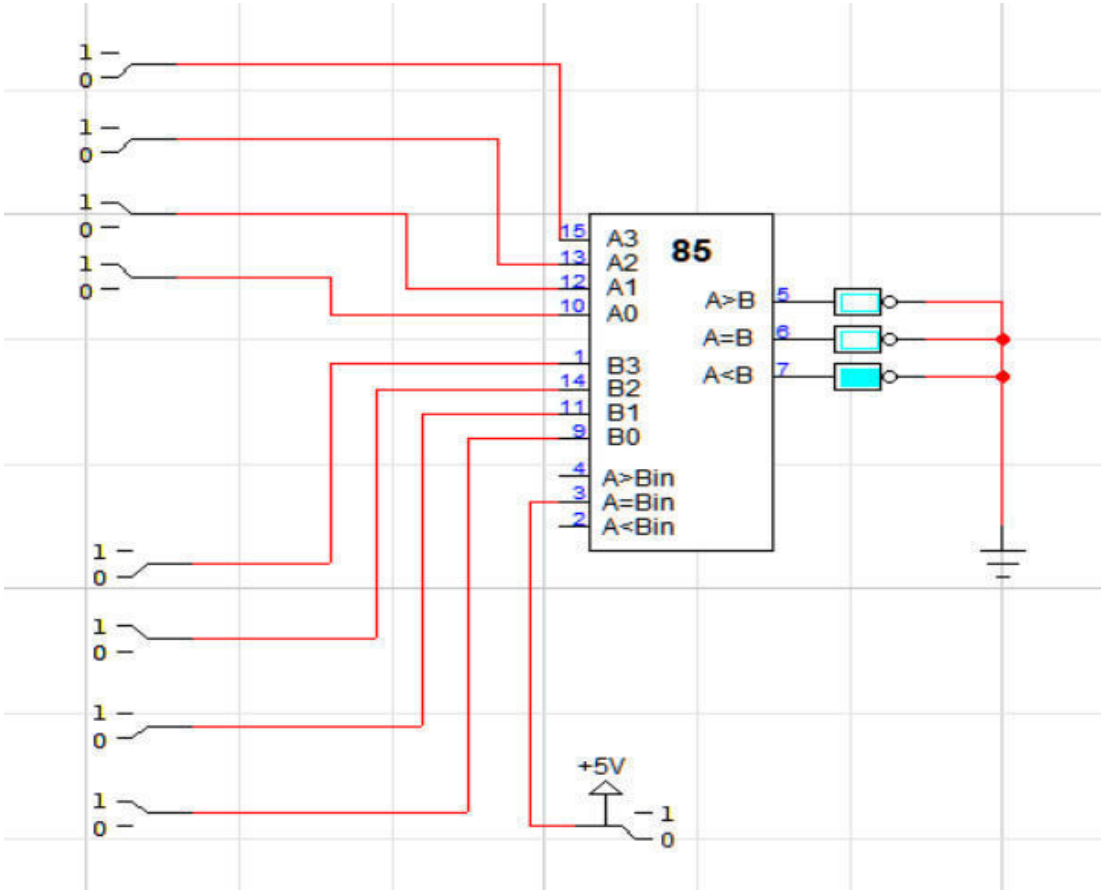
Comparison of A=1001 and B= 0110: (A>B)



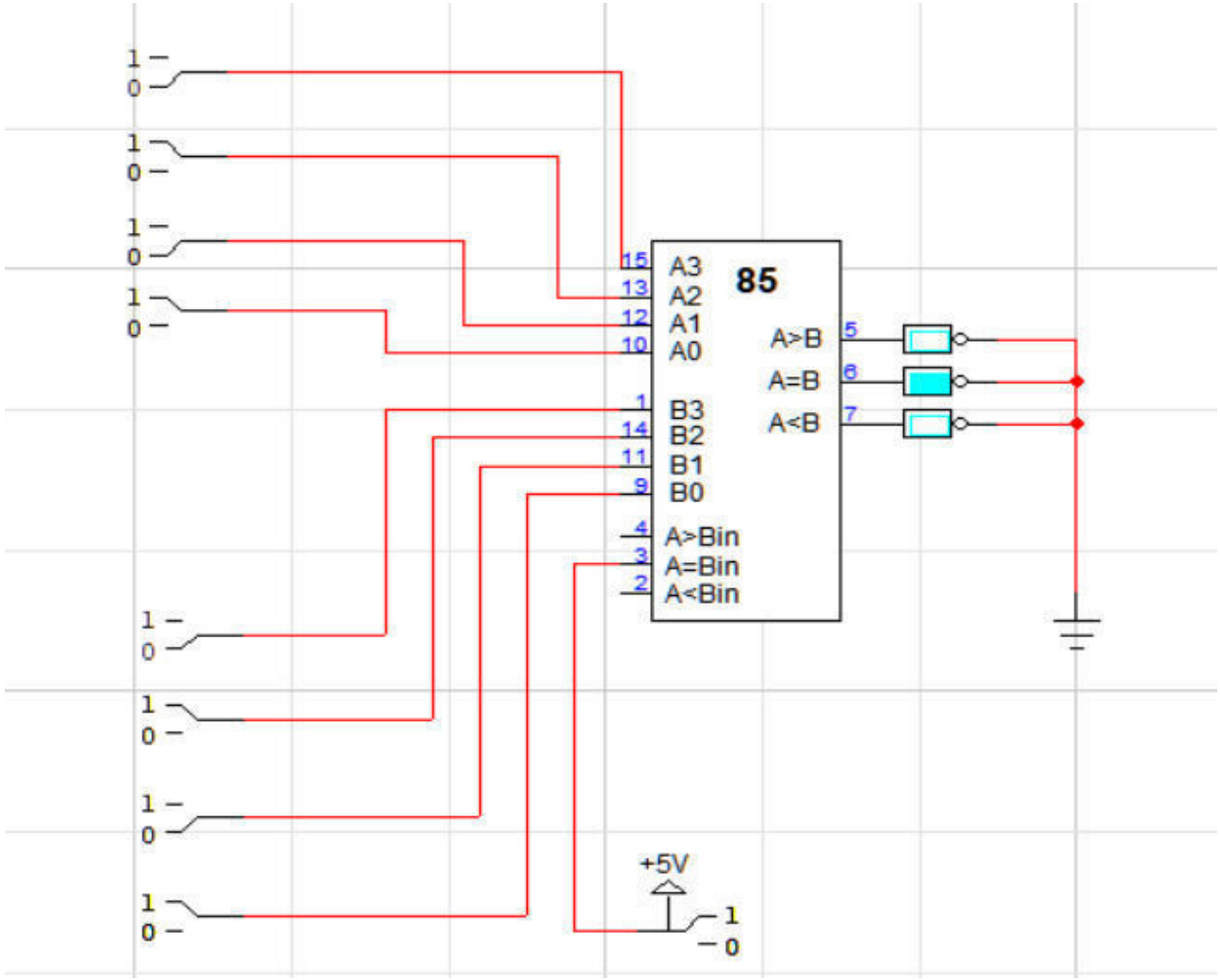
Comparison of A= 1100 and B= 1110: (A<B)



Comparison of A= 0011 and B = 0101: (A<B)



Comparison of A=0101 and B= 0101: (A=B)



f) ***A magnitude comparator can be constructed by using a subtractor as in Fig 2. And an additional combinational circuit. This is done with a combinational circuit which has 5 inputs S1, S2, S3, S4, and Co, and three outputs X, Y, Z see Fig.4.***

X = 1 if A = B Where Co = and S = 0000
Y = 1 if A < B Where Co = 0 and S ≠ 0000
Z = 1 if A > B Where Co = 1 and S ≠ 0000

Design and construct this logic circuit with minimum number of gates. Check the comparator action using Part (e). In the Lab verify your simulation.

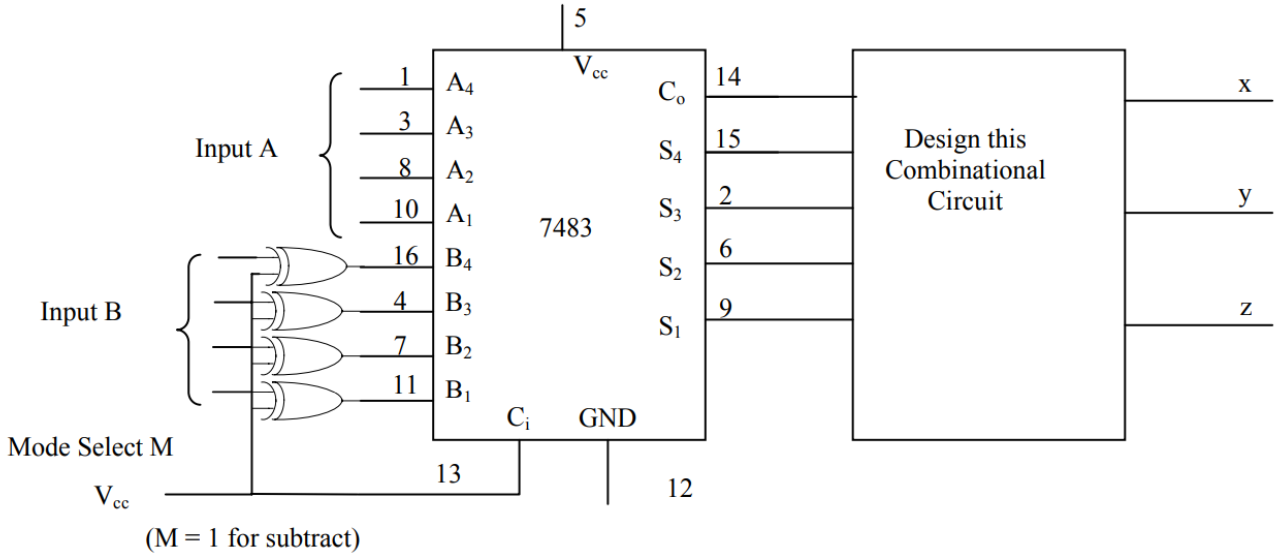


Fig.4 A magnitude comparator using a subtractor

Based on the above conditions,

X will be high if the numbers are equal i.e. S=0000 and Co=1.

So, **X** = **C**o. $(\overline{\boldsymbol{So} + \boldsymbol{S1} + \boldsymbol{S2} + \boldsymbol{S3}})$

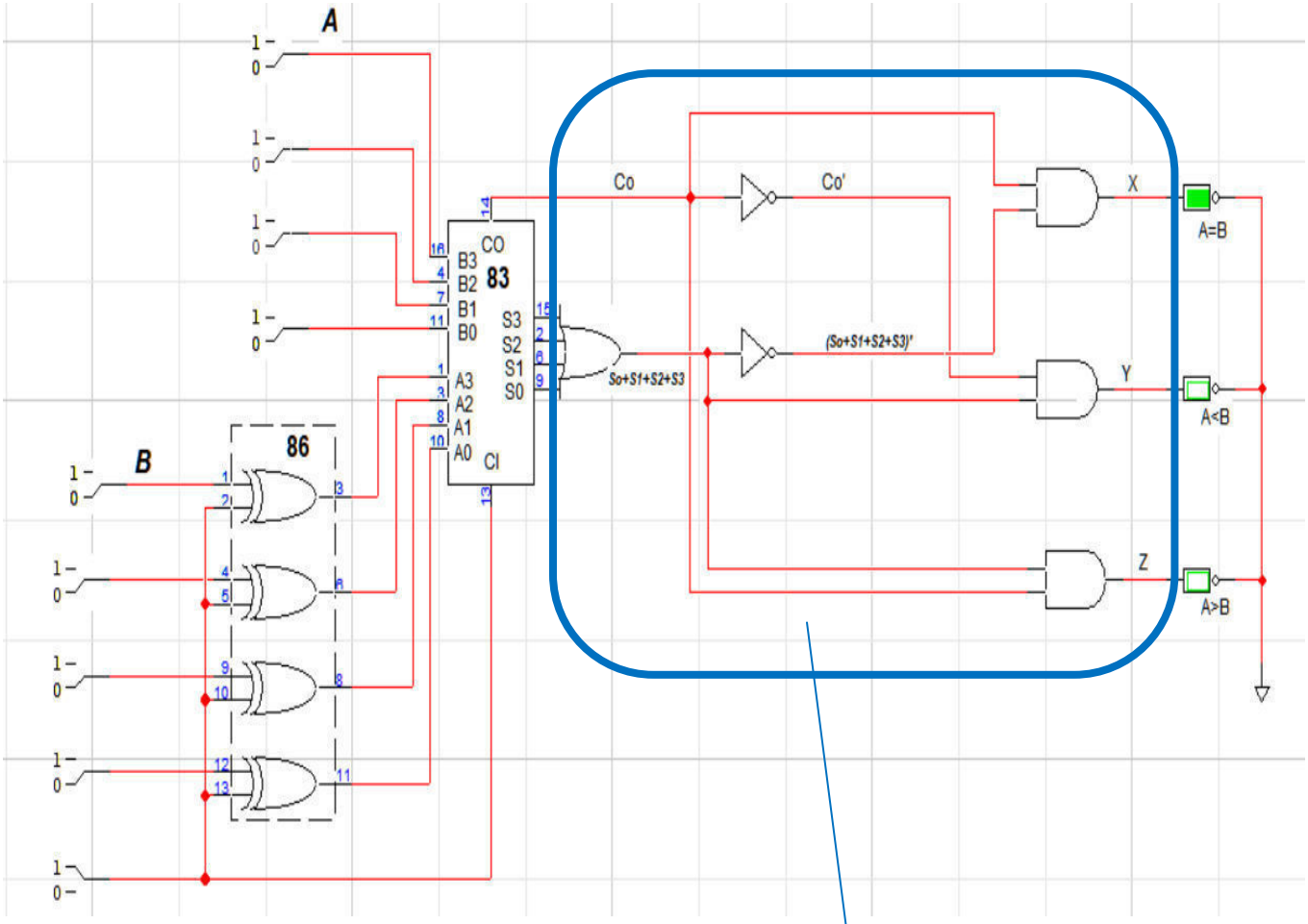
Y will be high if A < B. i.e. S ≠ 0000 and Co = 0

So, **Y** = $\overline{\mathbf{C}}$o. $(\boldsymbol{So} + \boldsymbol{S1} + \boldsymbol{S2} + \boldsymbol{S3})$

Y will be high if A > B. i.e. S ≠ 0000 and Co = 1

So, **Z** = **C**o. $(\boldsymbol{So} + \boldsymbol{S1} + \boldsymbol{S2} + \boldsymbol{S3})$

Bimal parajuli (20BDS0405)



Combinational Block

Bimal parajuli (20BDS0405)

DIGITAL LOGIC DESIGN CSE1003

Exp#6 - DESIGN WITH MULTIPLEXERS

Objectives:

To design a combinational circuit and implement it with multiplexers. To use a demultiplexer to implement a multiple output combinational circuit from the same input variables.

Apparatus:

- IC type 7404 HEX inverter
- IC type 7408 quad 2-input AND gate
- IC type 74151 8x1 multiplexer (1)
- IC type 74153 dual 4x1 multiplexer (2)
- IC type 7446 BCD-to-Seven-Segment decoder (1)
- Resistance network (1)
- Seven-Segment Display (1)

Softwares Used:

- ORCAD CAPTURE CIS
- Logic Works 5
- Proteus 8 pro

IC Description:

74151 is an 8 line-to-1 line multiplexer. It has the schematic representation shown in Fig 1. Selection lines S_2, S_1 and S_0 select the particular input to be multiplexed and applied to the output.

Strobe S acts as an enable signal. If strobe =1, the chip 74151 is disabled and output y = 0. If strobe = 0 then the chip 74151 is enabled and functions as a multiplexer. Table 1 shows the multiplex function of 74151 in terms of select lines.

Table 1.

Strobe	Select Lines			Output
S	S_2	S_1	S_0	Y
1	X	X	X	0
0	0	0	0	D0
0	0	0	1	D1
0	0	1	0	D2
0	0	1	1	D3
0	1	0	0	D4
0	1	0	1	D5
0	1	1	0	D6
0	1	1	1	D7

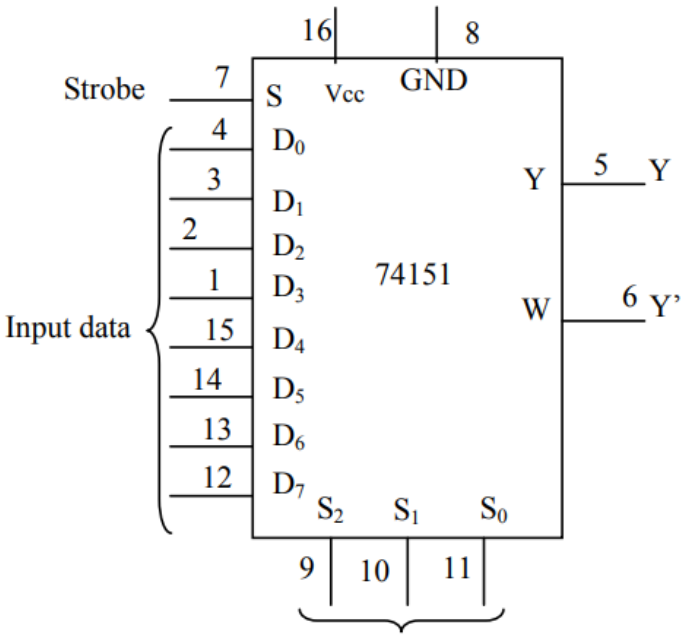


Fig.1 IC type 74151 Multiplexer 8×1

74153 is a dual 4 line-to-1 line multiplexer. It has the schematic representation shown in Fig 2. Selection lines S1 and S0 select the particular input to be multiplexed and applied to the output IY {1 = 1, 2}.
Each of the strobe signals *IG* {I = 1, 2} acts as an enable signal for the corresponding multiplexer.
Table 2. Shows the multiplex function of 74153 in terms of select lines.

Note that each of the on-chip multiplexers act independently from the other,

while sharing the same select lines S_1 and S_0.

Table 2

	Multiplexer 1		
Strobe	Select lines		Output
1G	S_1	S_0	1Y
1	X	X	0
0	0	0	$1D_0$
0	0	1	$1D_1$
0	1	0	$1D_2$
0	1	1	$1D_3$

	Multiplexer 2		
Strobe	Select lines		Output
2G	S_1	S_0	2Y
1	X	X	0
0	0	0	$2D_0$
0	0	1	$2D_1$
0	1	0	$2D_2$
0	1	1	$2D_3$

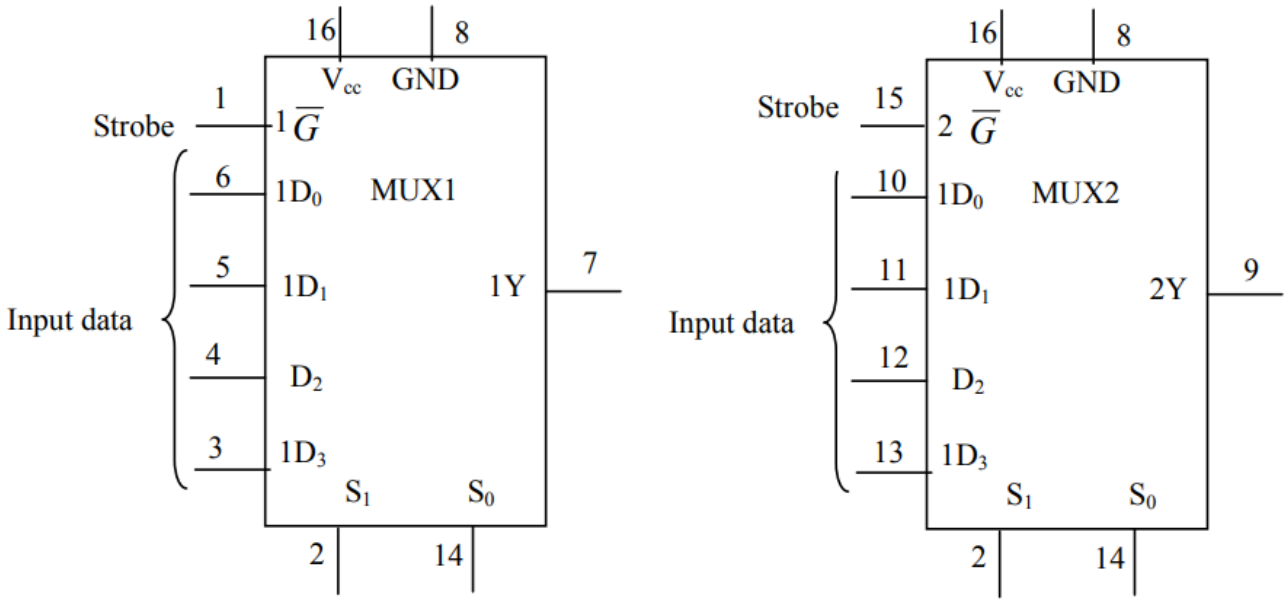


Fig.2 Chip 74153

IC 7446 is a BCD to seven segment decoder driver. It is used to convert the combinational circuit outputs in BCD forms into 7 segment digits for the 7 segment LED display units. Just like earlier experiment.

Procedure:

Part I: Parity Generator:

a) Design a parity generator by using a 74151 multiplexer. Parity is an extra bit attached to a code to check that the code has been received correctly.

Odd parity bit means that the number of 1's in the code including the parity bit is an odd number. Fill the output column of the truth table in Table 2 for a 5-bit code in which four of the bits (A, B, C, D) represents the information to be sent and fifth bit (x), represents the parity bit. The required parity is an odd parity.

The inputs B, C and D correspond to the select inputs of 74151. Complete the truth table in Table 3 by filling in the last column with 0, 1, A or A'.

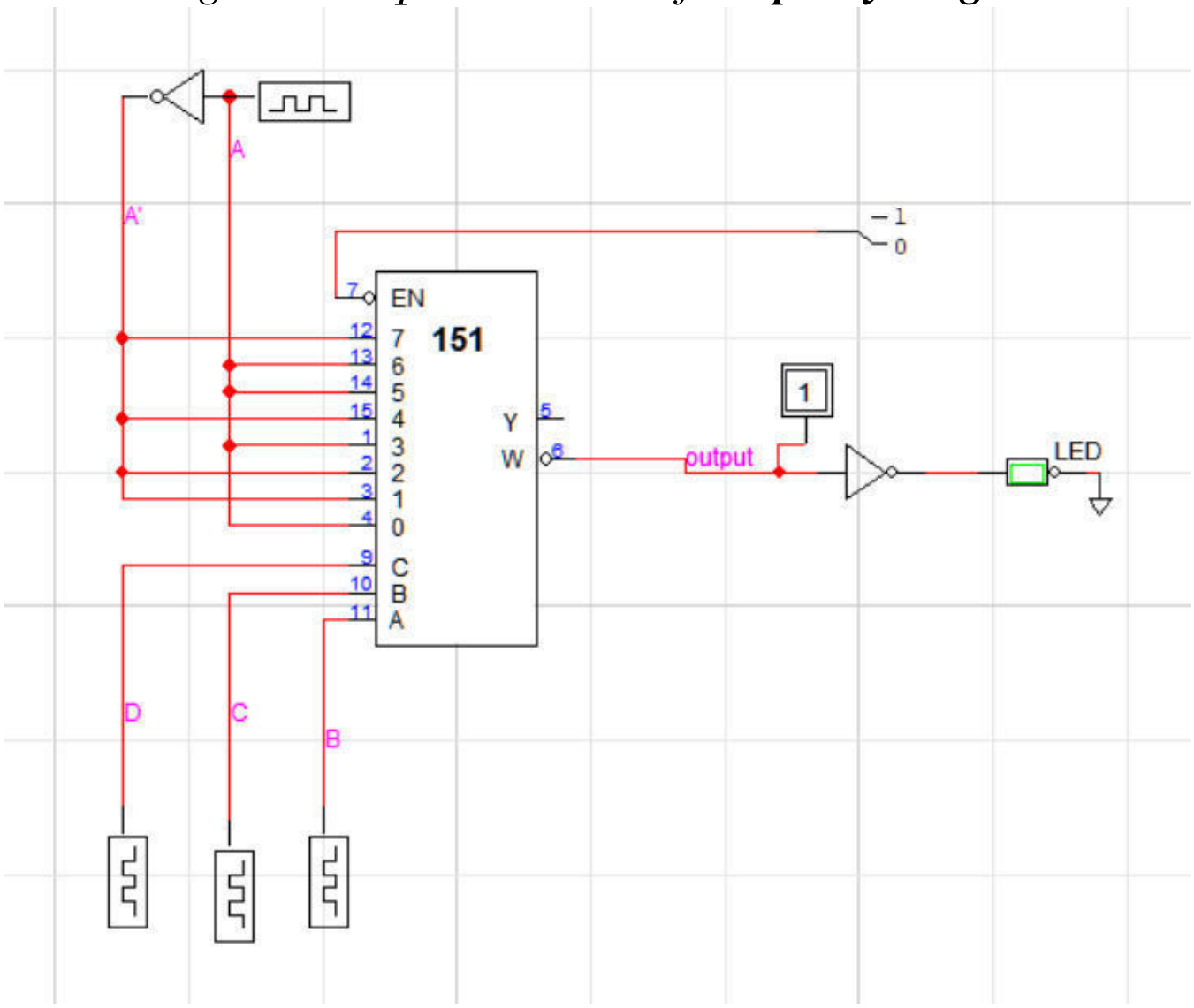
Ans:

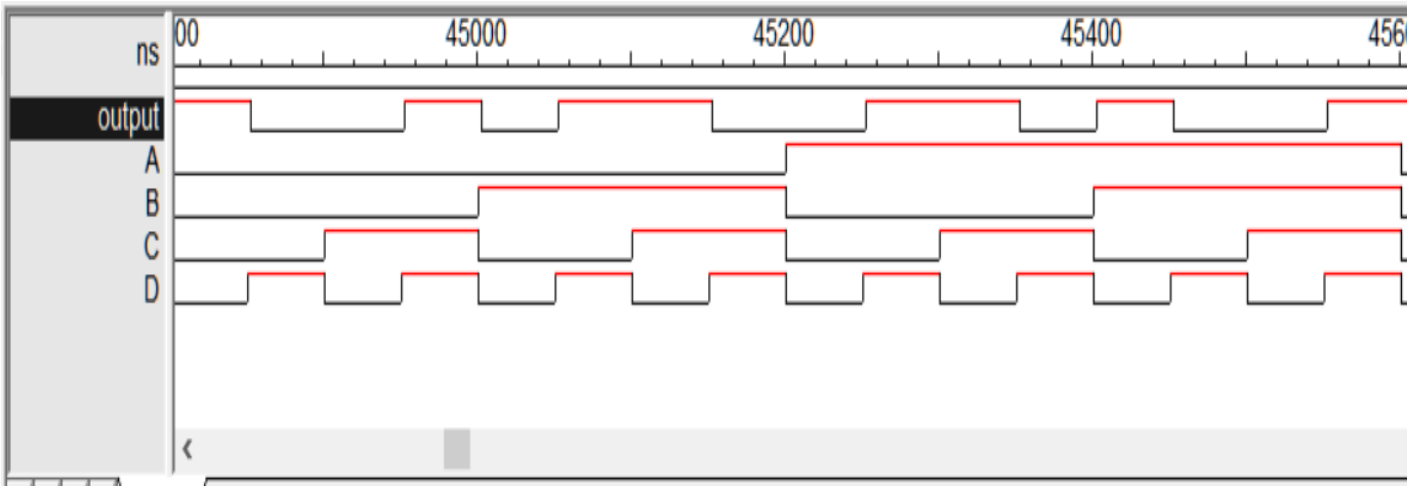
Following is the table that demonstrates the corresponding parity bit along with the 4 original bits for **odd parity check**

Inputs				Output	Connects Data to
A	B	C	D	x	
0	0	0	0	1	A’
0	0	0	1	0	A
0	0	1	0	0	A
0	0	1	1	1	A’
0	1	0	0	0	A
0	1	0	1	1	A’
0	1	1	0	1	A’
0	1	1	1	0	A
1	0	0	0	0	A’
1	0	0	1	1	A
1	0	1	0	1	A
1	0	1	1	0	A’
1	1	0	0	1	A
1	1	0	1	0	A’
1	1	1	0	0	A’
1	1	1	1	1	A

b) Simulate the circuit using Software, use 74-151 multiplexer and Binary switches for inputs and Binary Probes for outputs. The 74151 has one output for Y and another inverted output W. Use A and A' for providing values for inputs 0-7. The internal values "A, B, C" are used for selection inputs B, C, and D. Simulate the circuit and test each input combination filling in the table shown below. In the Lab connect the circuit and verify the operations. Connect an LED to the multiplexer output so that it represents the parity bit which lights any time when the four bits input have even parity.

Following is the required circuit of the ***parity bit generator:***




ns
00
45000
45200
45400
output
A
B
C
D

Part 2: Vote Counter:

A committee is composed of a chairman (C), a senior member (S), and a member (M). The rules of the committee state that:

- The vote of the member (M) will be counted as 2 votes
- The vote of the senior member (S) will be counted as 3 votes.
- The vote of the chairman (C) will be counted as 5 votes.

Each of these persons has a switch to close ("l") when voting yes and to open ("0") when voting no.

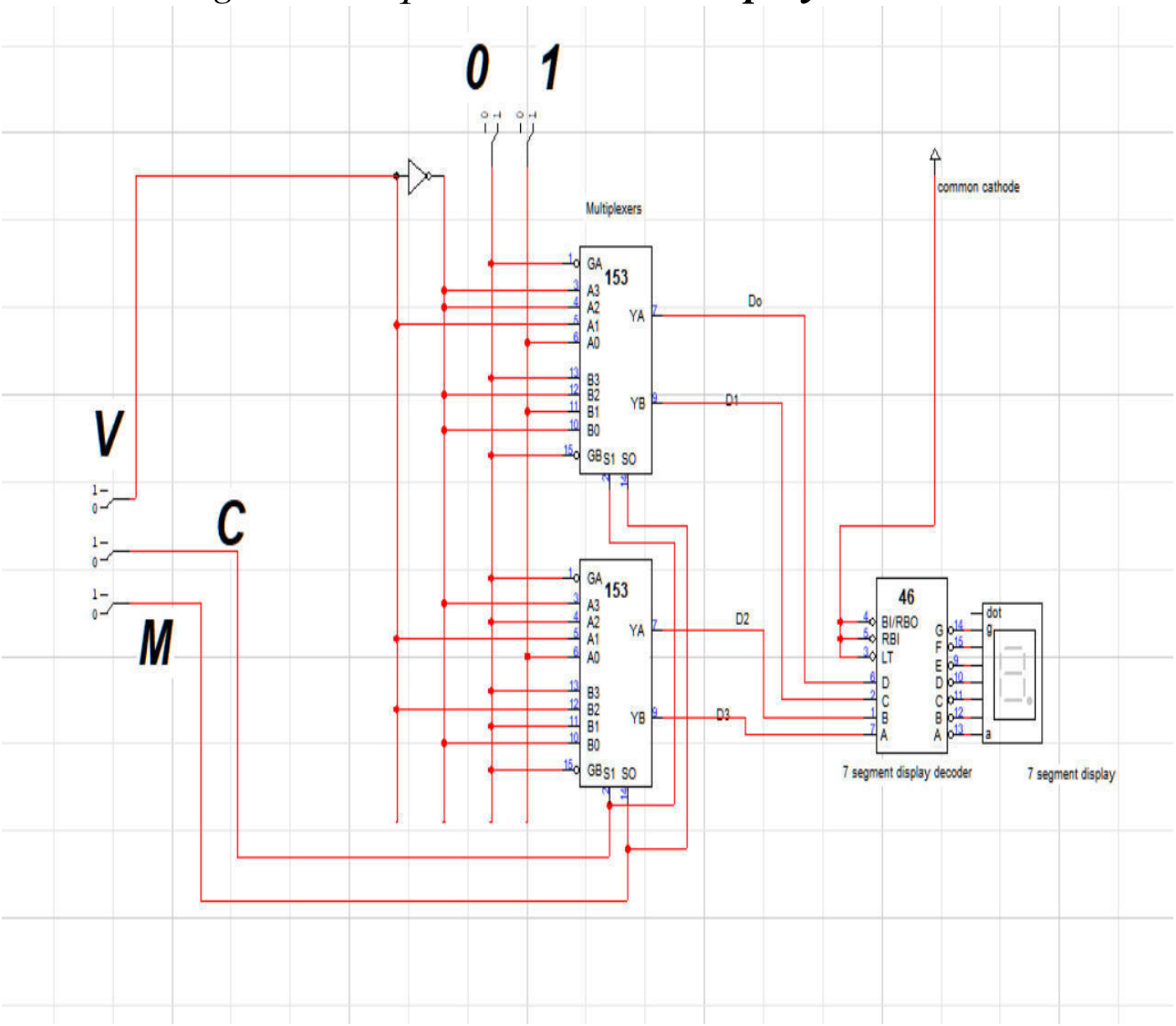
It is necessary to design a circuit that displays the total number of votes for each issue. Use a seven segment display and a decoder to display the required number.

If all members vote no for an issue the display should be blank. (Recall from Experiment #5, that a binary input 15 into the 7446 blanks all seven segments).

If all members vote yes for an issue, the display should be 0. Otherwise the display shows a decimal number equal to the number of 'yes' votes. Use two 74153 units, which include four multiplexers to design the combinational circuit that converts the inputs from the members' switch to the BCD digit for the 7446.

In Software, use +5V for Logic 1 and ground for Logic 0 and use switches for C, S, and M. Use two chips 74153 and one decoder 7446 verify your design and get a copy of your circuit with the pin numbers to Lab so that you could connect the hardware in exactly the same way.

Following is the required circuit to ***display the vote count:***:



Here,

Let vote of Chairperson --> C (weightage, C = 5)

Let vote of senior member --> S (weightage, S = 3)

Let vote of general member --> M (weightage, M=2)

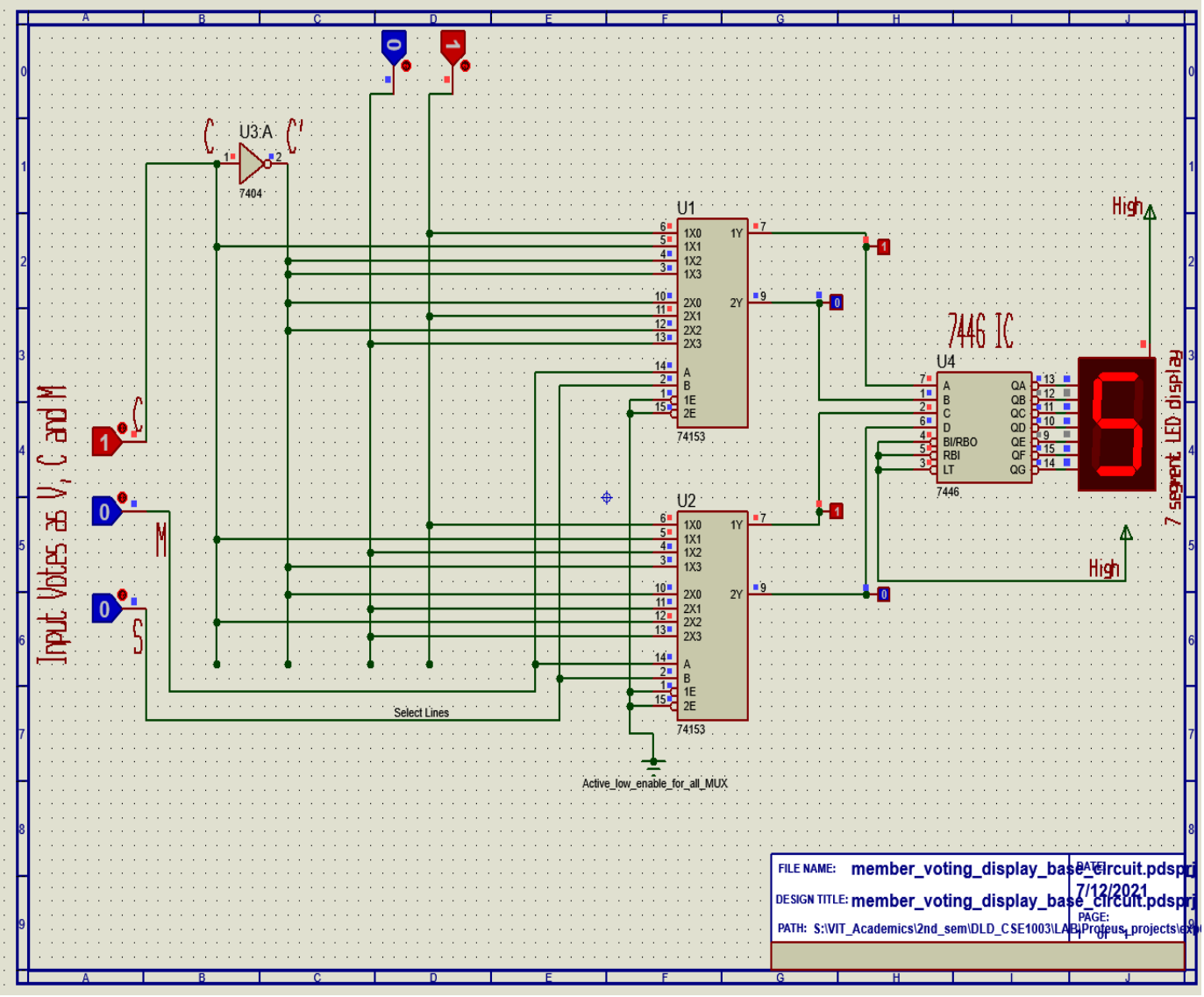
When only C votes in favor,

C=1,

S=0,

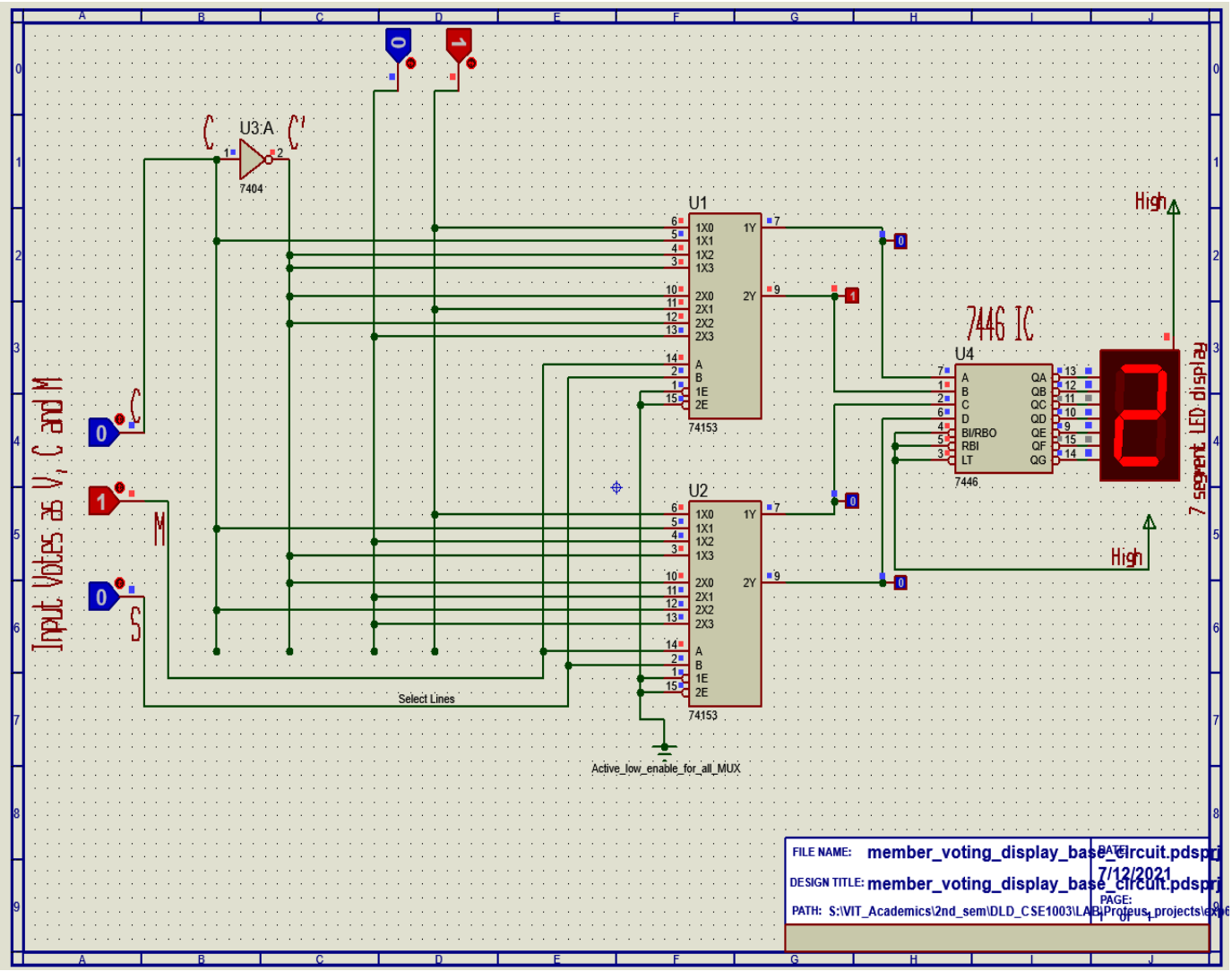
M=0

Display out =5



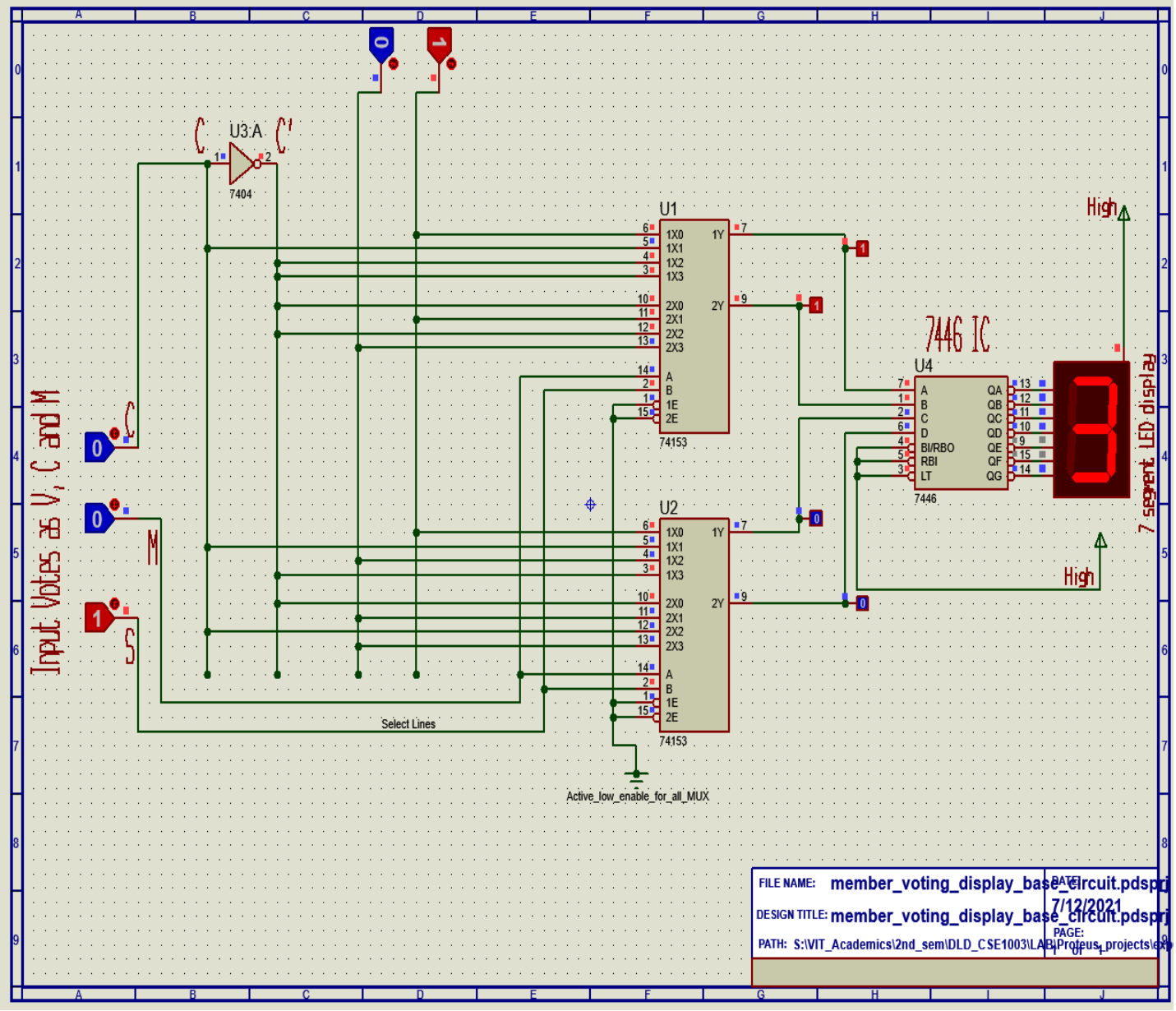
Simulated in Proteus 8 pro

When only M votes in favor,
C=0,
M=1,
S=0
Display out =2



Simulated in Proteus 8 pro

When only S votes in favor,
C=0,
M=0,
S=1
Display out = 3



Simulated in Proteus 8 pro

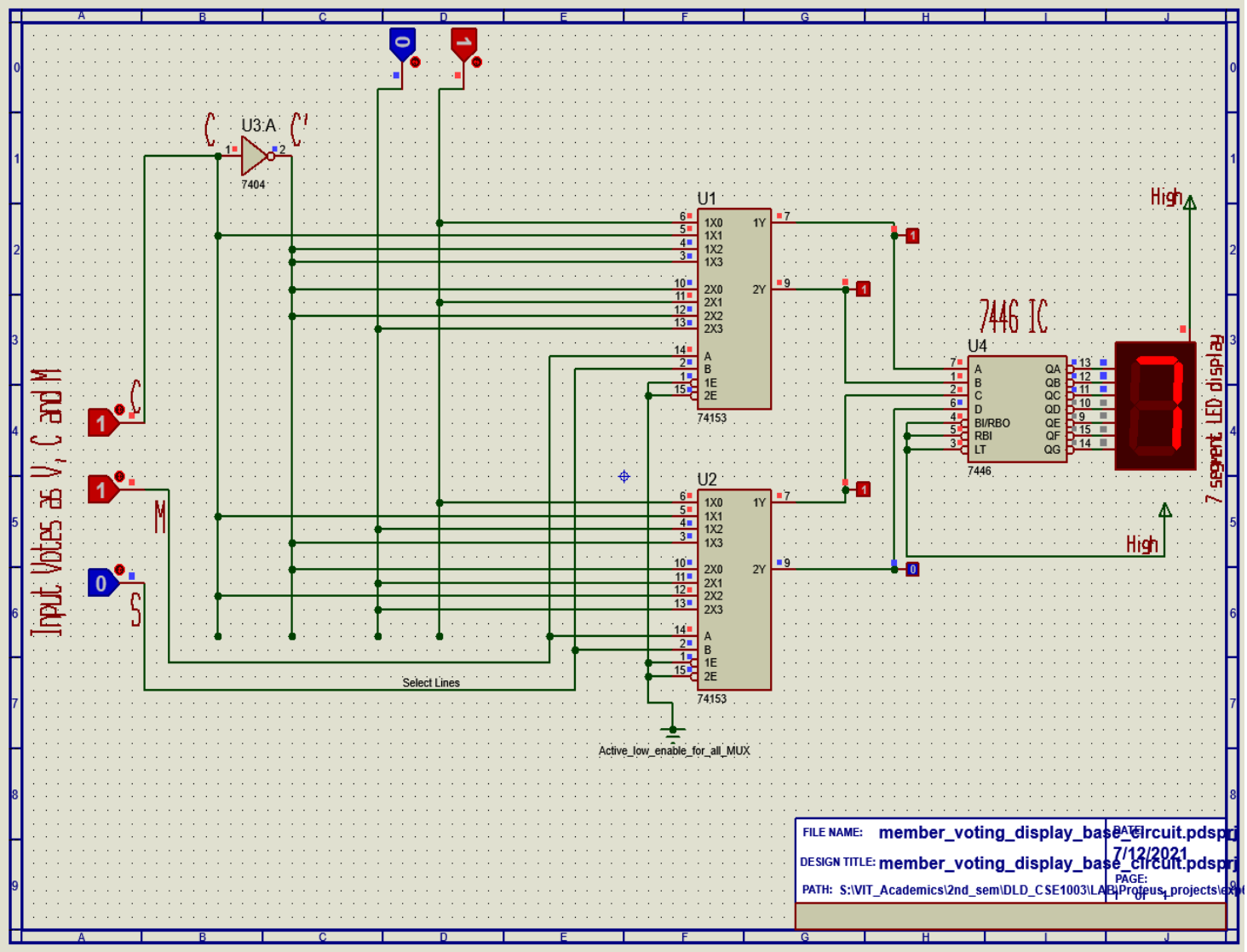
When C and M votes in favor,

C=1,

M=1,

S=0

Display out = 7



Simulated in Proteus 8 pro

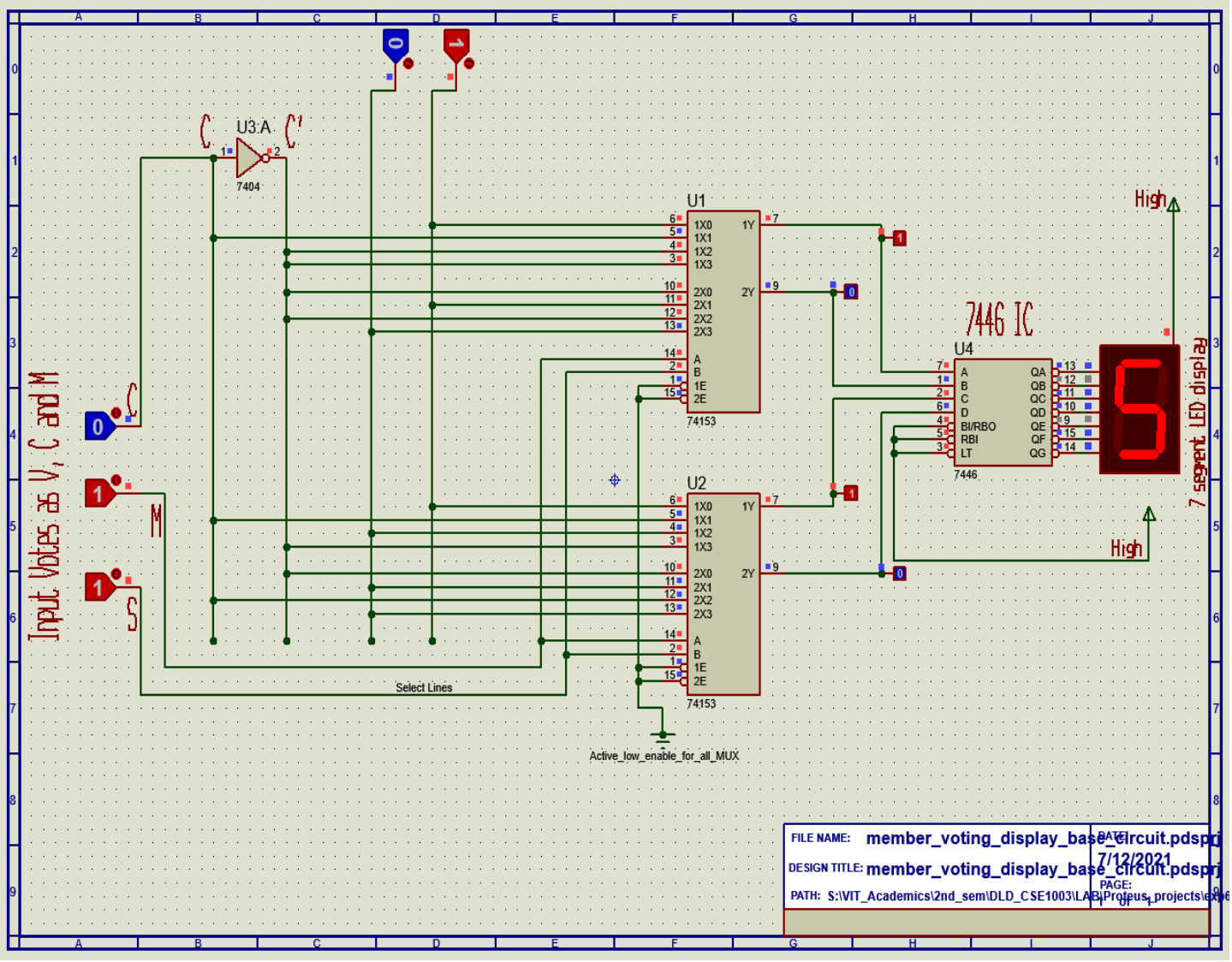
When M and S votes in favor,

C=0,

M=1,

S=1

Display out = 5



Simulated in Proteus 8 pro

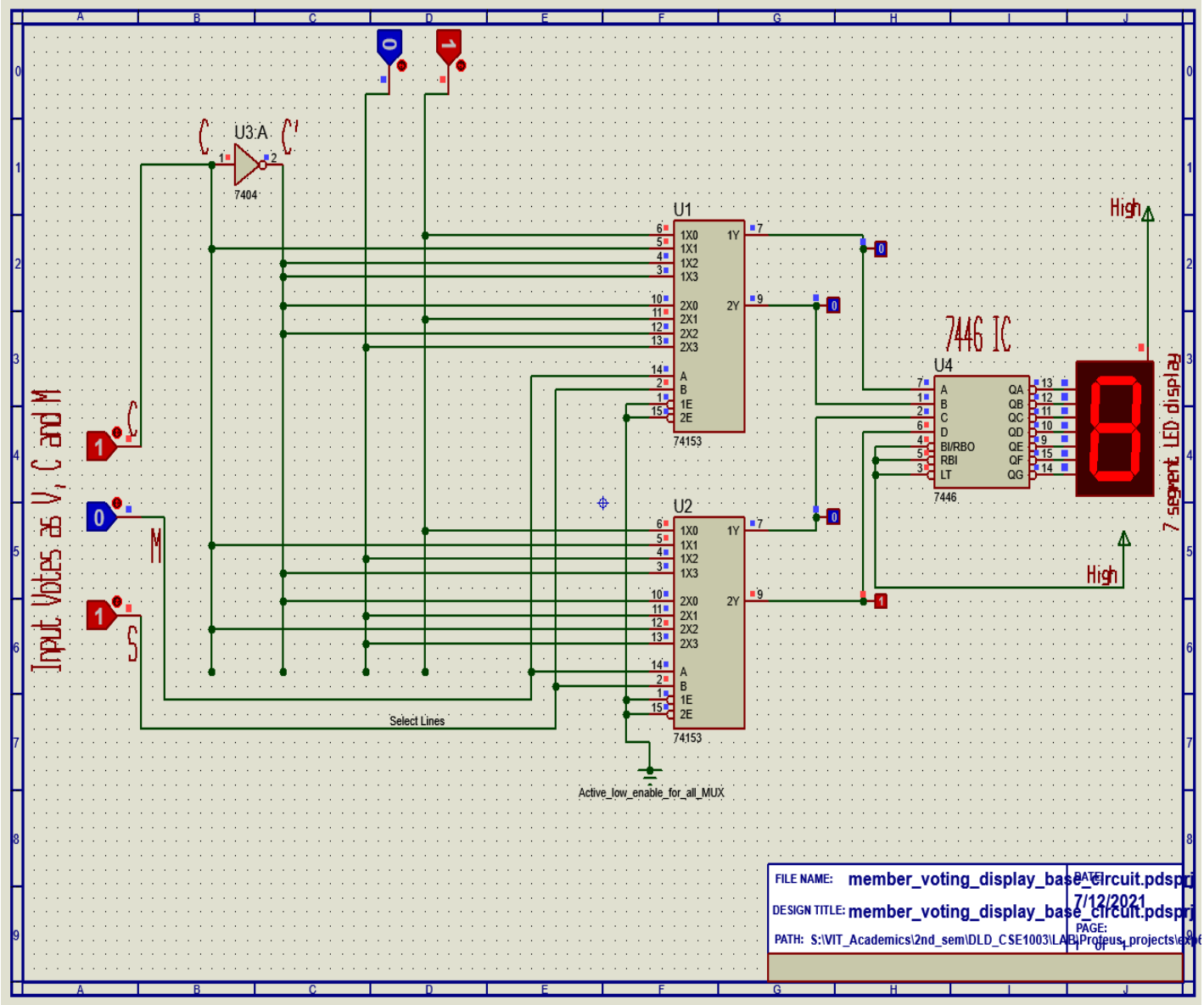
When C and S votes in favor,

C=1,

M=0,

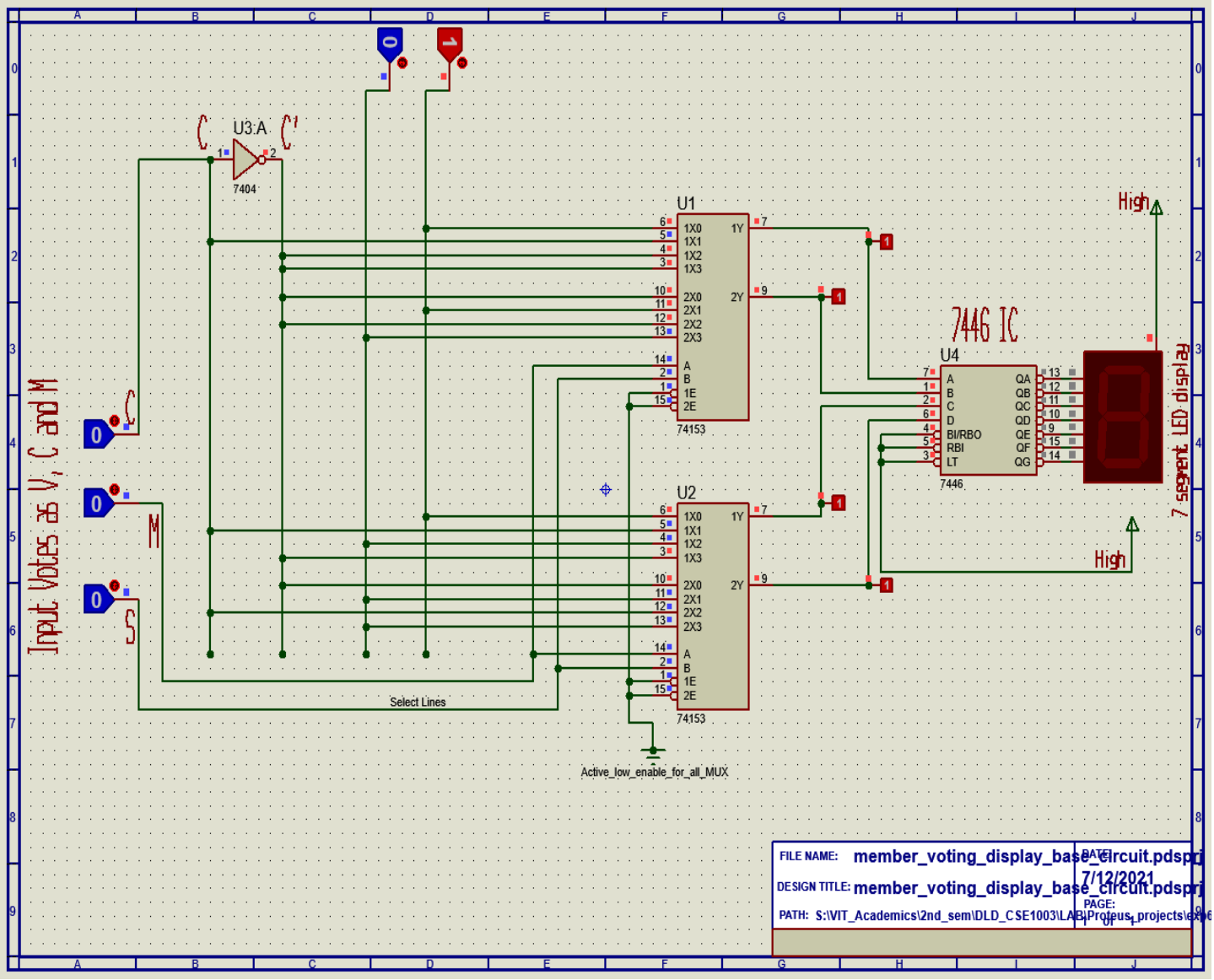
S=1

Display out = 8



Simulated in Proteus 8 pro

When no one votes in favor,
C=0,
M=0,
S=0
Display out = null.



Simulated in Proteus 8 pro

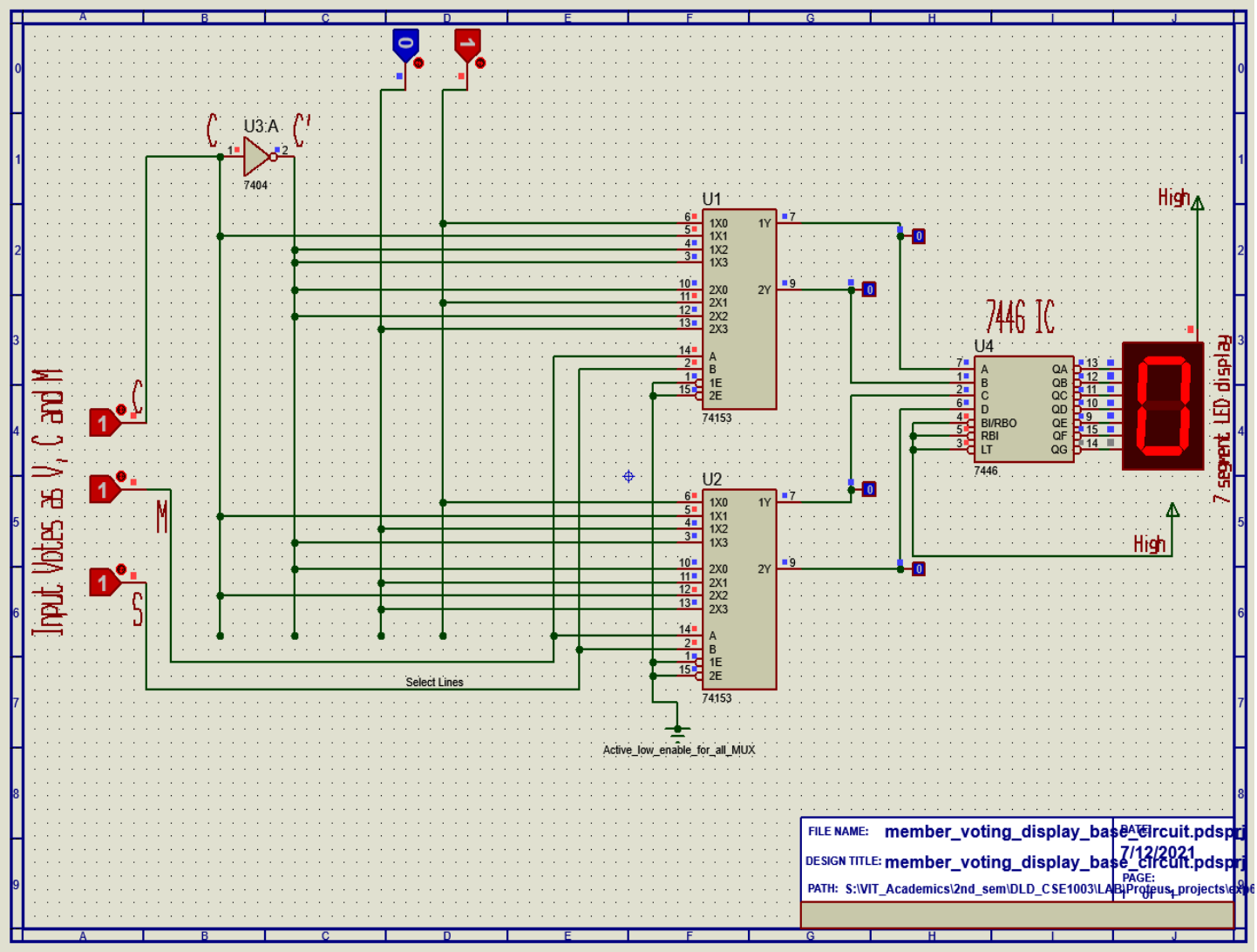
When all votes in favor,

C=1,

M=1,

S=1

Display out = 0 (Although it's 10, 1 as carry can't be displayed).



Simulated in Proteus 8 pro

--*THE END*--

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Digital Logic Design
CSE1003 LAB

7. Flip Flops
8.Sequential circuits and Counters

Experiment – 7 & 8

BIMAL PARAJULI
(20BDS0405)

6/14/21

CSE1003 (LAB)

Bimal Parajuli (20BDS0405)

DIGITAL LOGIC DESIGN (CSE1003)

Exp#7- FLIP-FLOPS

Objectives:

1. To become familiar with flip-flops.

2. To implement and observe the operation of different flip-flops.

Apparatus:

- IC type 7400 quad 2-input NAND gate
- IC type 7410 triple 3- input NAND gate
- IC type 7476 dual JK master-slave flip-flops.
- IC type 7474 dual D positive-edge-trigged flip-flops.
- Dual trace oscilloscope.

Softwares Used

- Cadence Capture CIS lite

Bimal Parajuli (20BDS0405)

Procedure:

1. ***In the pre-lab using software construct the circuit shown in Fig. 1***

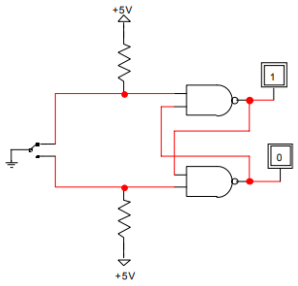
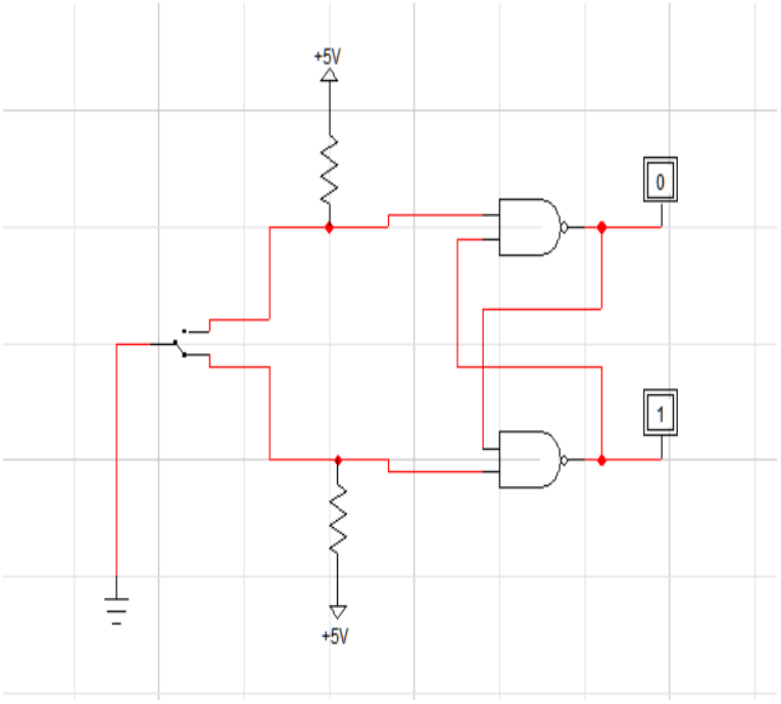
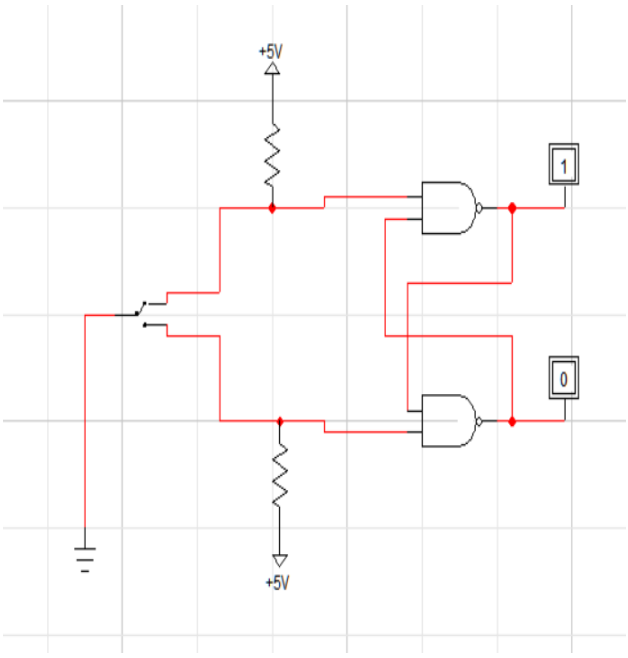


Fig :1

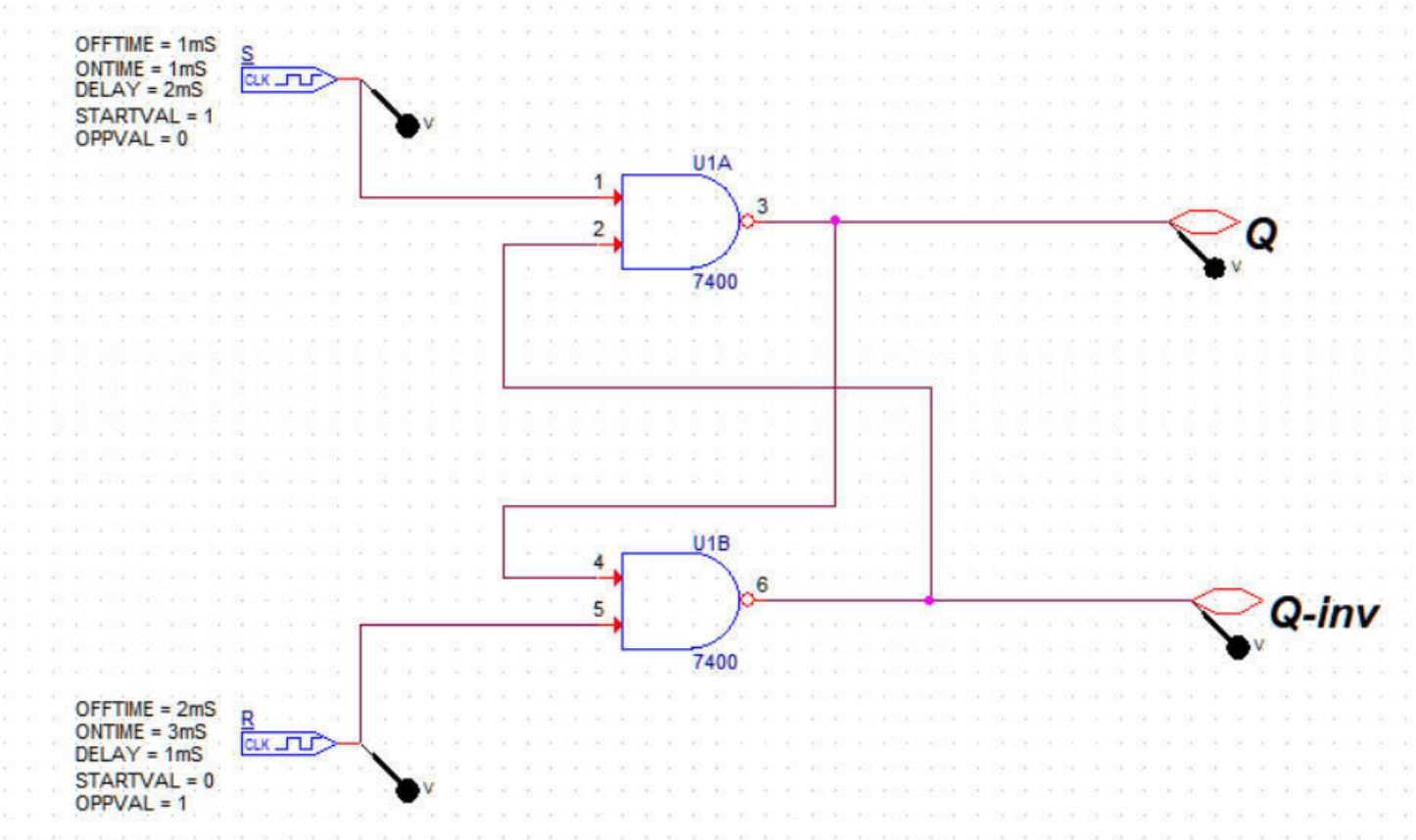


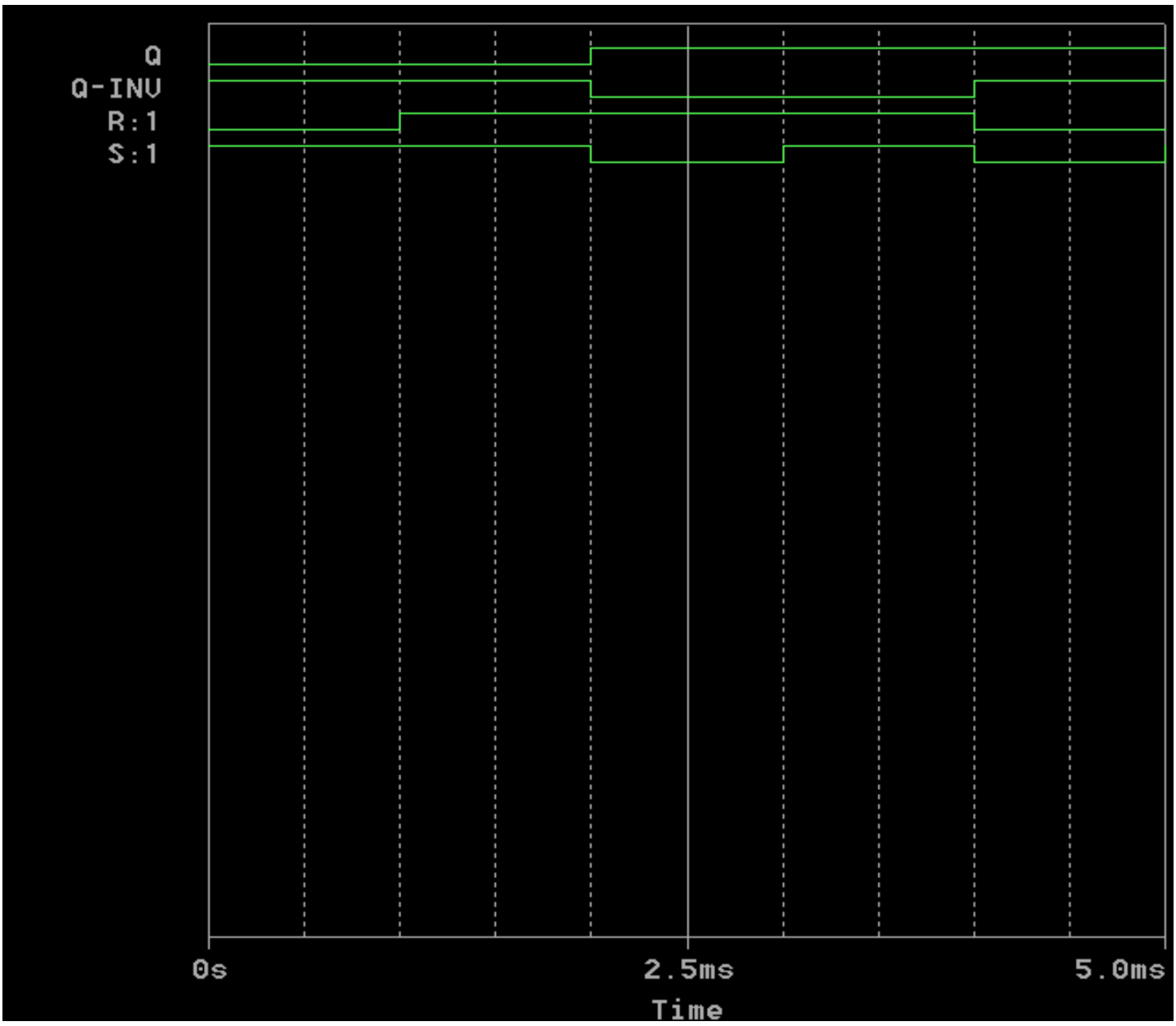


Where we could use generic NAND gates or 74-00 and Binary Probes to simulate LEDs. Finally, we use SPDT for the bouncing switch. Using the simulated circuit fill in the truth table.

S	R	Q	Q′	Cases
1	0	0	1	Reset
1	1	0	1	Memory
0	1	1	0	Set
1	1	1	0	Memory
0	0	1	1	Invalid

On doing laboratory simulation using OrCAD PsPice, we obtain the following circuit which is tabulated in the above table




Q
Q-INV
R:1
S:1
0s
2.5ms
5.0ms
Time

In the Lab, Build the RS latch shown in fig.2. Use SPDT switch S2 as a bouncing switch. Q and Q' Outputs are connected to LED 'S. Verify the truth table experimentally.

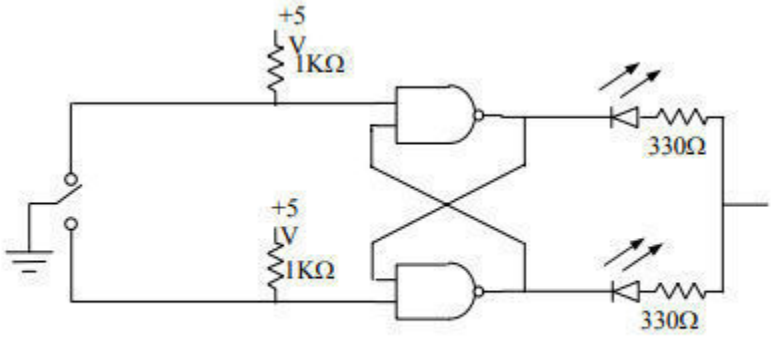
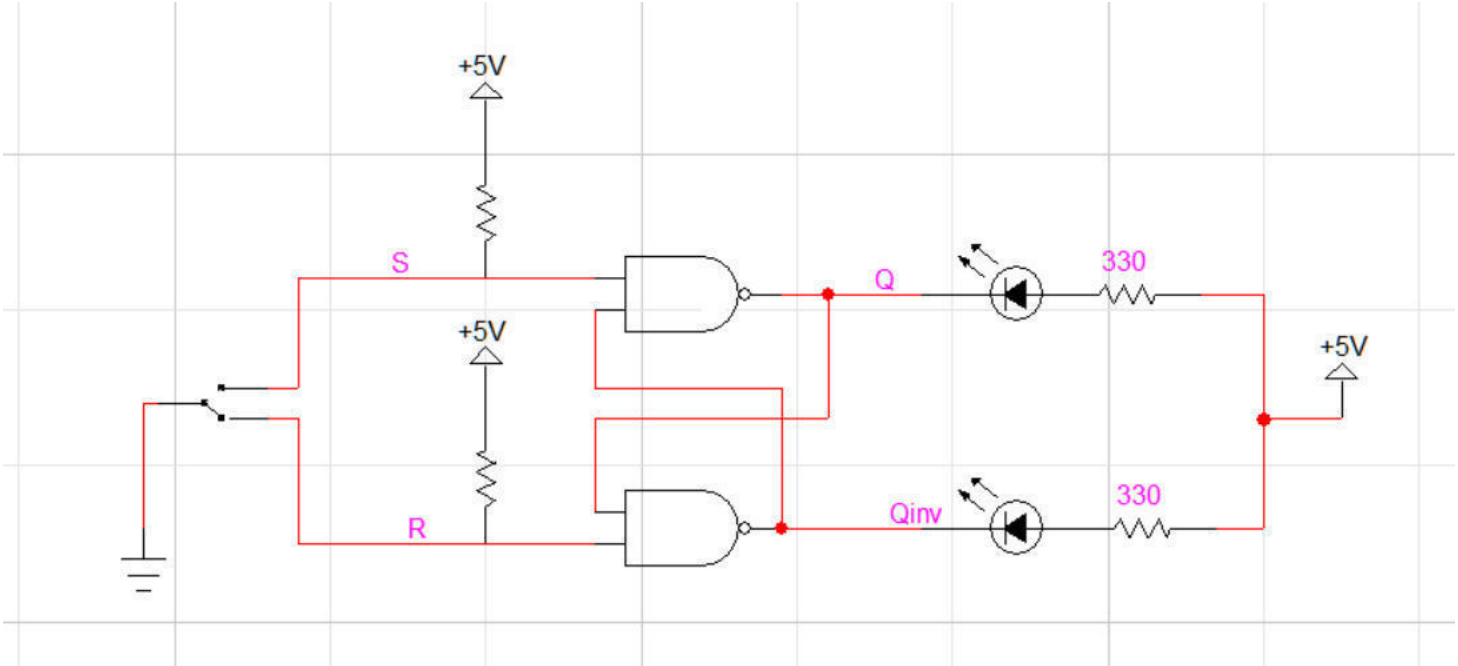
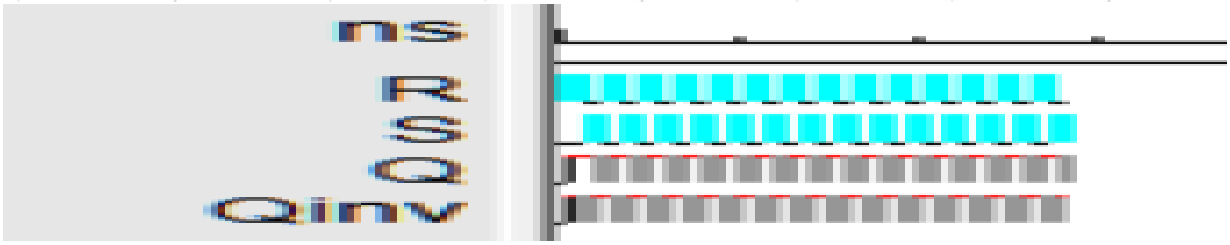


Fig. 2

Following is the SR latch made in laboratory using SPDT switch input in Software





2. ***Modify the basic R-S into a D latch by adding the steering gates and the inverter shown in Fig 3.***
   Connect the D input to the pulse generator of the Digi designer and set it at 1 Hz.
   Connect the enable input to a high through 1k resistor. Observe the output; obtain the truth table experimentally then change the enable to a low.
   Is the enable an active high or an active low? Leave the enable low and place a momentary short to ground first on one output and then on the other. What happens?

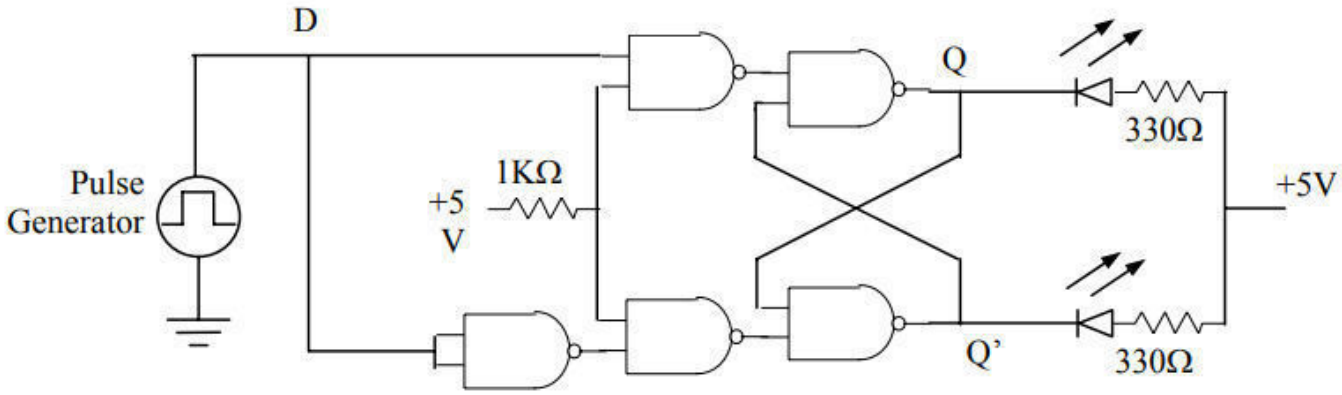
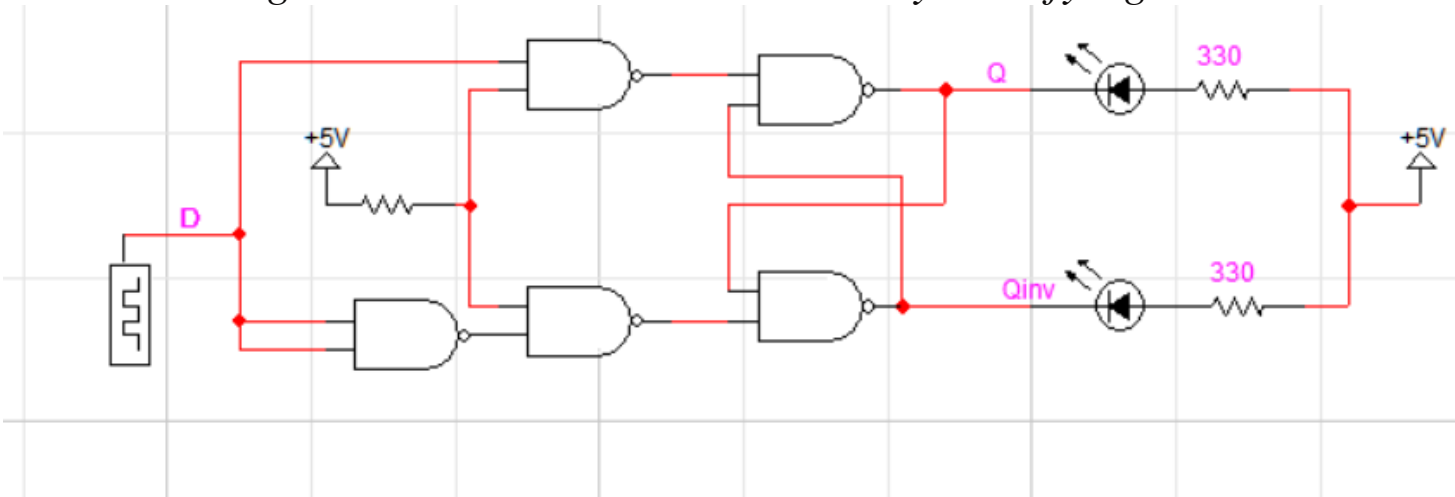
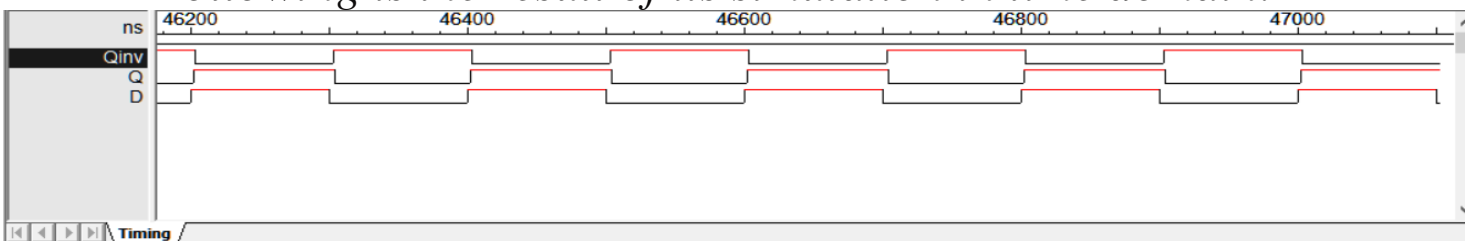


Fig: 3

Following is the Basic D Latch created by modifying SR latch.



Following is the result of its simulation in time domain:

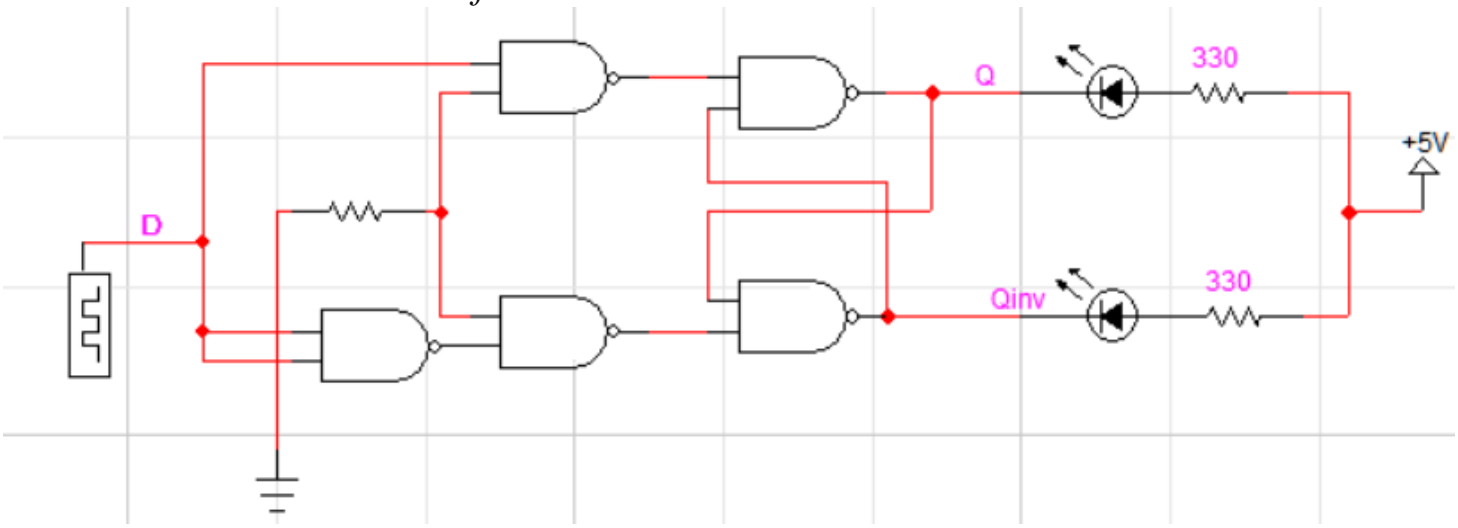


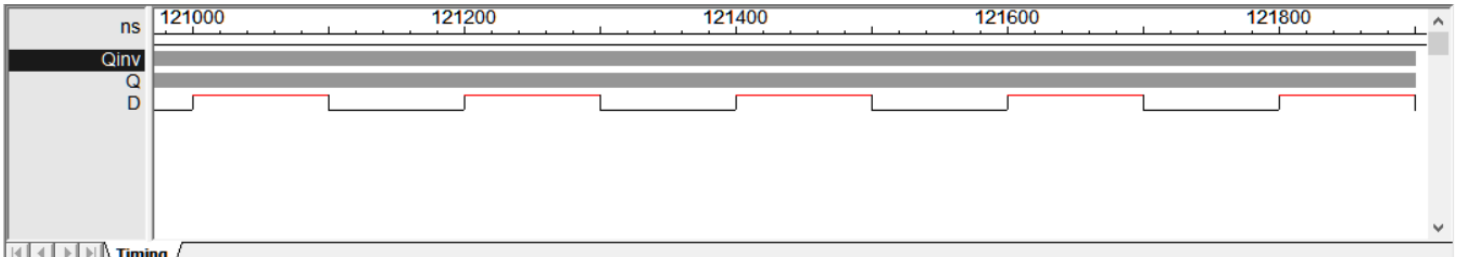
Thus the following behavior is obtained:

Enable	*D*	*Q*	*Q'*	*Observation*
0	*X*	*Retained*	*Retained*	*No change*
1	*0*	*0*	*1*	*Reset*
1	*1*	*1*	*0*	*Set*

The enable of D latch is active high one. If we set enable as low, the output of the flip-flop stays the same and independent to the input value D.

Which is demonstrated as follows:





2. ***The 7476 is a dual JK master-slave flip-flop with preset and clear inputs. The function table given in table 1 defines the operation of the flip-flop. The +ve transition of the CLOCK (CP) pulse changes the master flip-flop, and the (-ve) transition changes the slave flip-flop as well as the output of the circuit. In Software the chip 7476 is not available, however, the generic JK flip-flop behave in exactly the same way as the 7476. The "S" represents the Preset, the "R" represents the Clear, and C represents the clock pulse (CP). Verify the table by connecting Binary switches to R, S, J, K, and C. Notice that only the negative edge of the clock affects the outputs (Q, and Q').***

Table 1

Input					Output	
Preset	Clear	Clock	J	K	Q	Q'
0	1	X	X	X	1	0
1	0	X	X	X	0	1
0	0	X	X	X	1	1
1	1	↓	0	0	No change	
1	1	↓	0	1	0	1
1	1	↓	1	0	1	0
1	1	↓	1	1	Toggle	

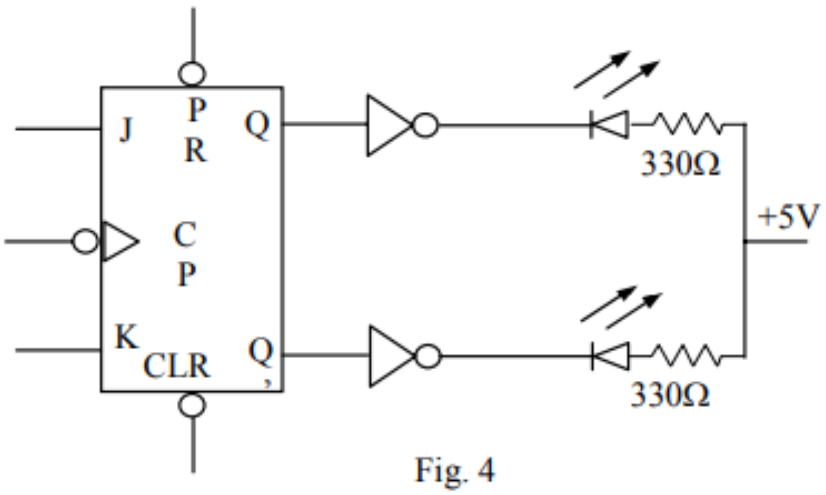


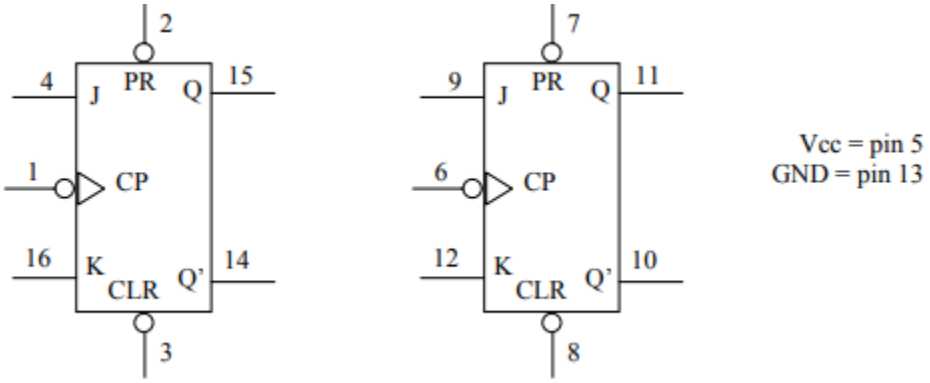
Fig. 4

In the Lab, Construct the circuit of Fig 4. Look at the data sheet for the 7476 and determine the inactive logic required at the PRE and CLR inputs.

Connect the 7476 for the SET mode by connecting J = 1, K = 0. With CLOCK (CP) = O; test the effect of PRE, CLR by putting a 0 on each, one at a time.

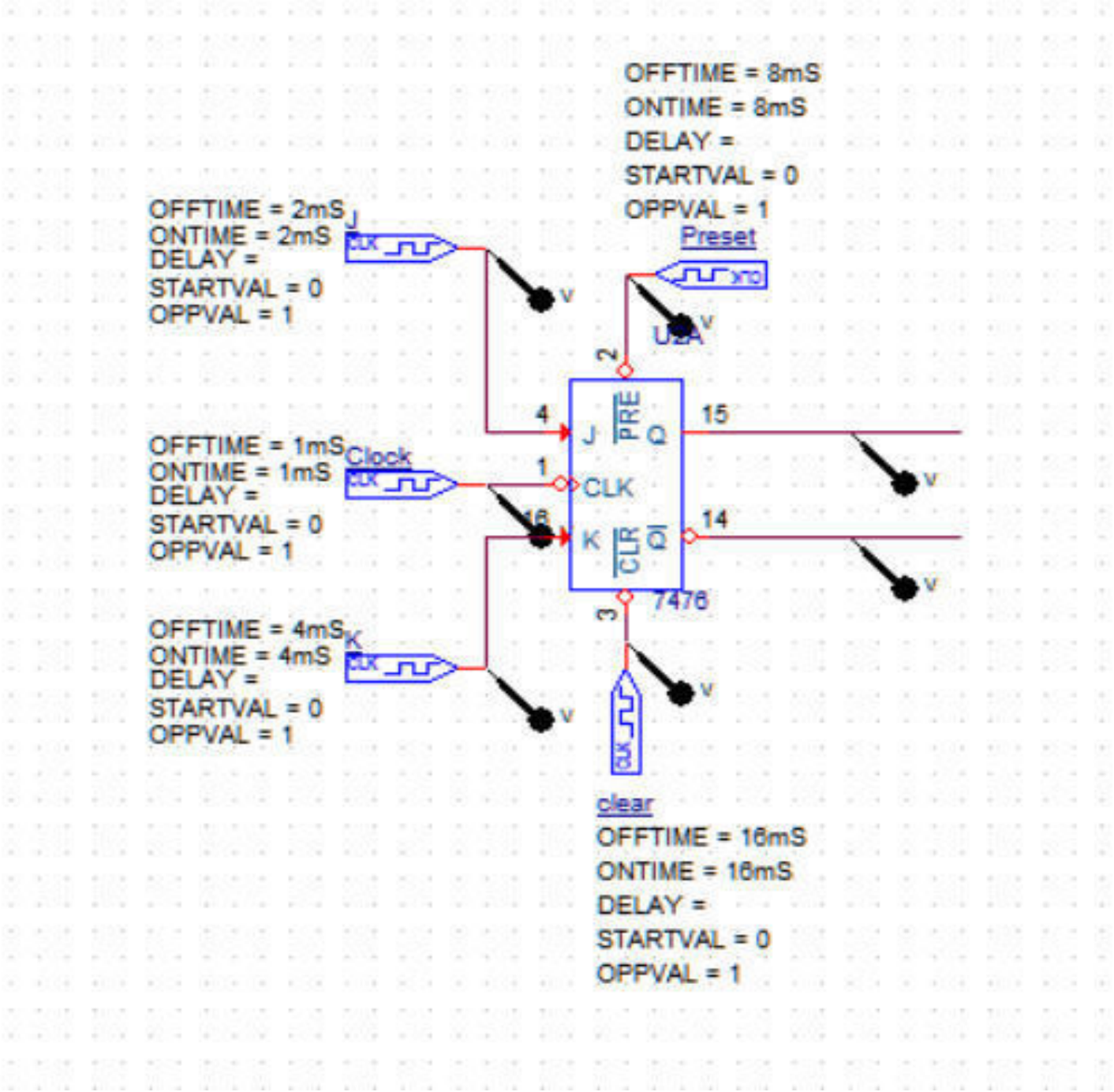
Put CLR = 0, then pulse the clock (CP) by putting a HIGH then a LOW, on the clock. Does the CLR input override J input?

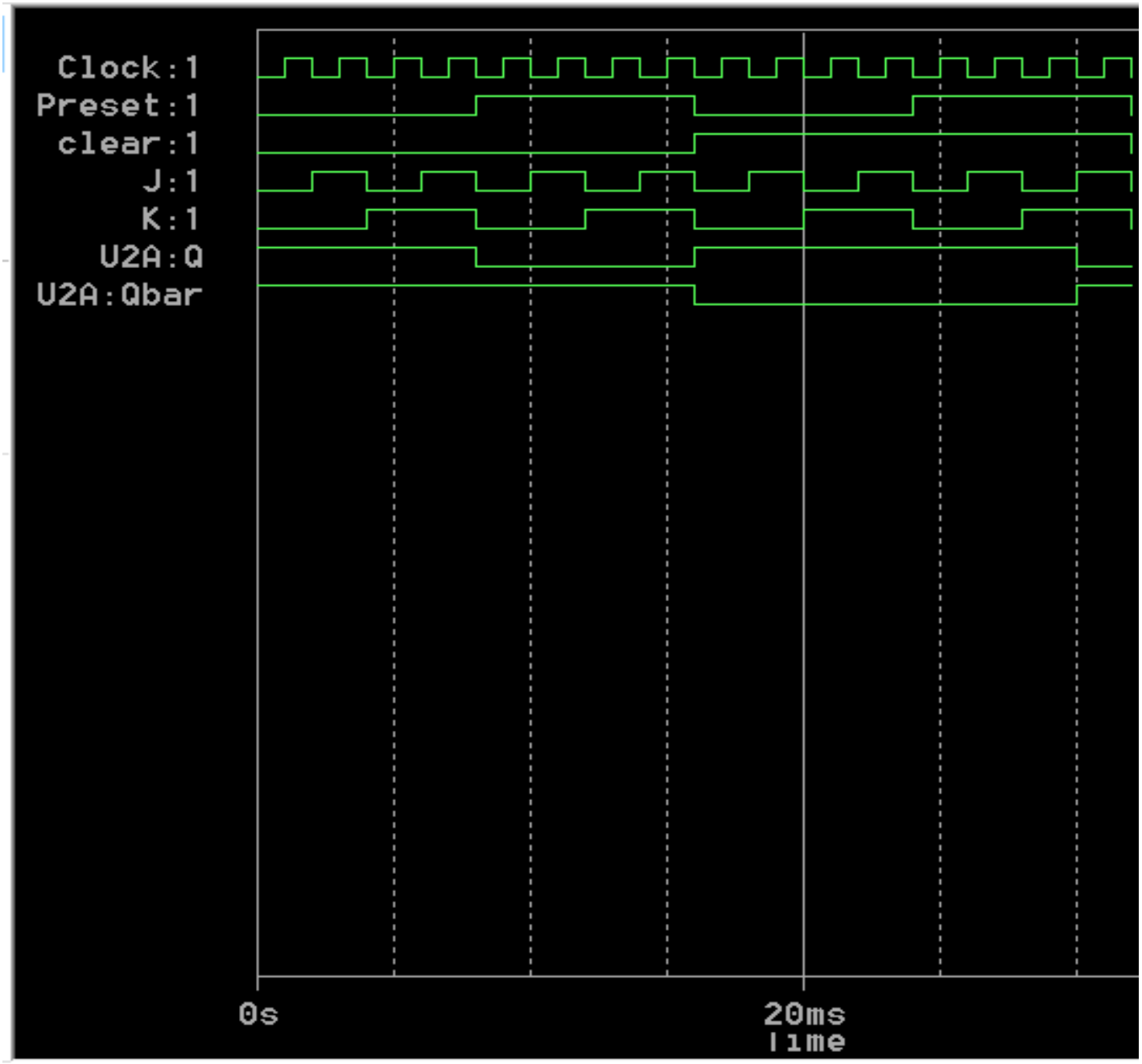
Verify the operation of the JK flip flop by experimentally obtaining the characteristics Table.



Following is the arrangement to observe and study the behavior of JK flip flop with preset and clear inputs.

*In the transient simulation alongside, we can observe how **Q** and **Q'** change with the change in clock, J, K, Preset and Clear.*




Clock:1
Preset:1
clear:1
J:1
K:1
U2A:Q
U2A:Qbar
0s
20ms
Time

Observations:

When Preset =0 and Clear =0, always Q=1, Q'=1

When Preset =0 and Clear =1, always Q=1, Q'=0

When Preset =1 and Clear =0, always Q=0, Q'=1

When Preset =1 and Clear =1, Q depends on J and K at negative edge of the clock.

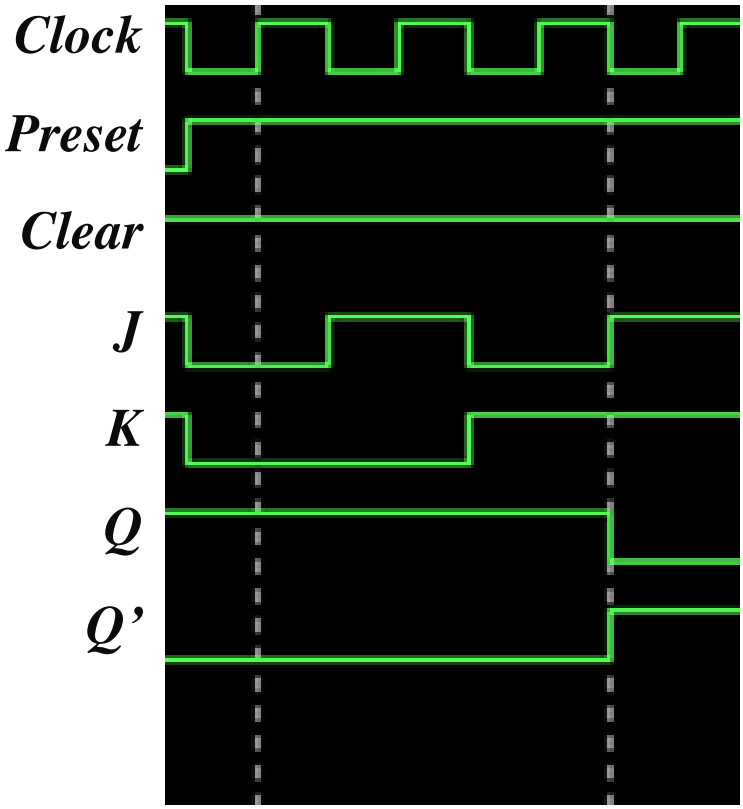
When J=0, K=0: Q and Q' remains same

When J=0, K=1: Q=0 and Q'=1

When J=1, K=0: Q=1 and Q'=0

When J=1, K=1: Q and Q' toggles.

This is the small part of simulation when both Preset, Clear are 1.



Following characteristics table is obtained for JK flip flop:

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Does the CLR input override the J input?

- *Yes, CLR and preset can override J, K and Clock inputs. J, K and Clock are significant only when preset and clear inputs are set to 1, 1*

DIGITAL LOGIC DESIGN (CSE1003)

Exp #8 - CLOCKED SEQUENTIAL CIRCUITS AND COUNTERS

OBJECTIVE:

- To design, build and test synchronous sequential circuits.
- To design, build, and test synchronous counters
- To design, build and test asynchronous counters

APPARATUS:

- IC type 7476 dual JK master-slave flip-flops
- IC type 7400 quad 2-input NAND gates

Softwares Used:

- Cadence Capture CIS Lite

PROCEDURE:

1. *SYNCHRONOUS SEQUENTIAL CIRCUITS:*

a) Design, construct and test a sequential circuit whose state is shown in Fig.1. Use JK flip-flops in the design.

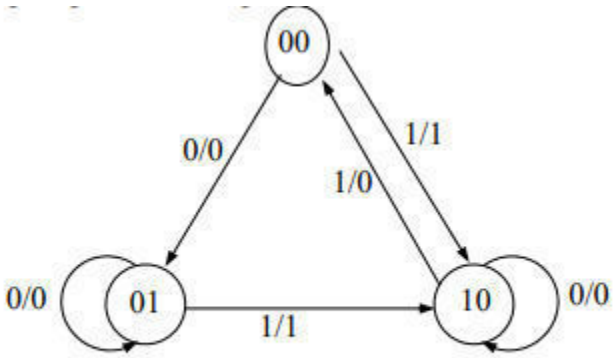


Fig. 1

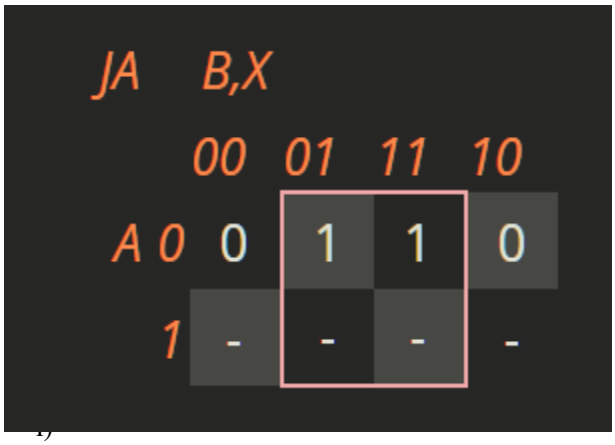
The circuit has two flip-flops A, B, one input x and one output y. The circuit is to be designed by treating the unused states as don't care conditions. The final circuit must be analyzed to ensure that it is self-correcting. If not suggest a solution.

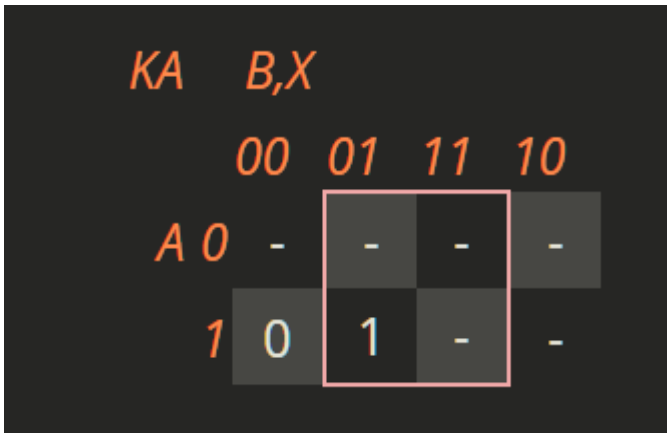
b) *Complete the excitation table shown in Table 1*

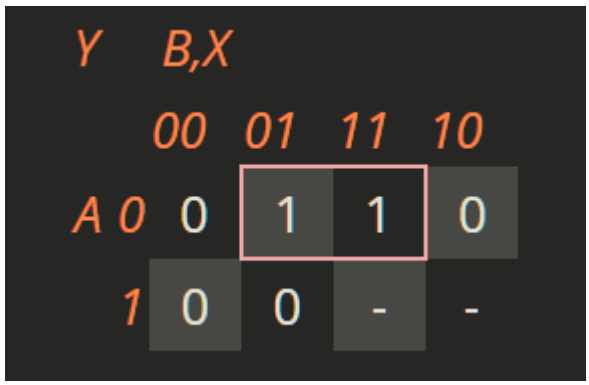
Table 1.

Present state		Input	Next state		Output	Flip-flop input function			
A	B	X	A	B	Y	JA	KA	JB	KB
0	0	0	0	1	0	0	X	1	X
0	0	1	1	0	1	1	X	0	X
0	1	0	0	1	0	0	X	X	0
0	1	1	1	0	1	1	X	X	1
1	0	0	1	0	0	X	0	0	X
1	0	1	0	0	0	X	1	0	X
1	1	0	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X

c) *Using Karnaugh maps obtain minimal expressions for the flip-flop input functions JA, KA, JB, KB.*







From the above Kmap, Input functions are:

JA = X

KA = X

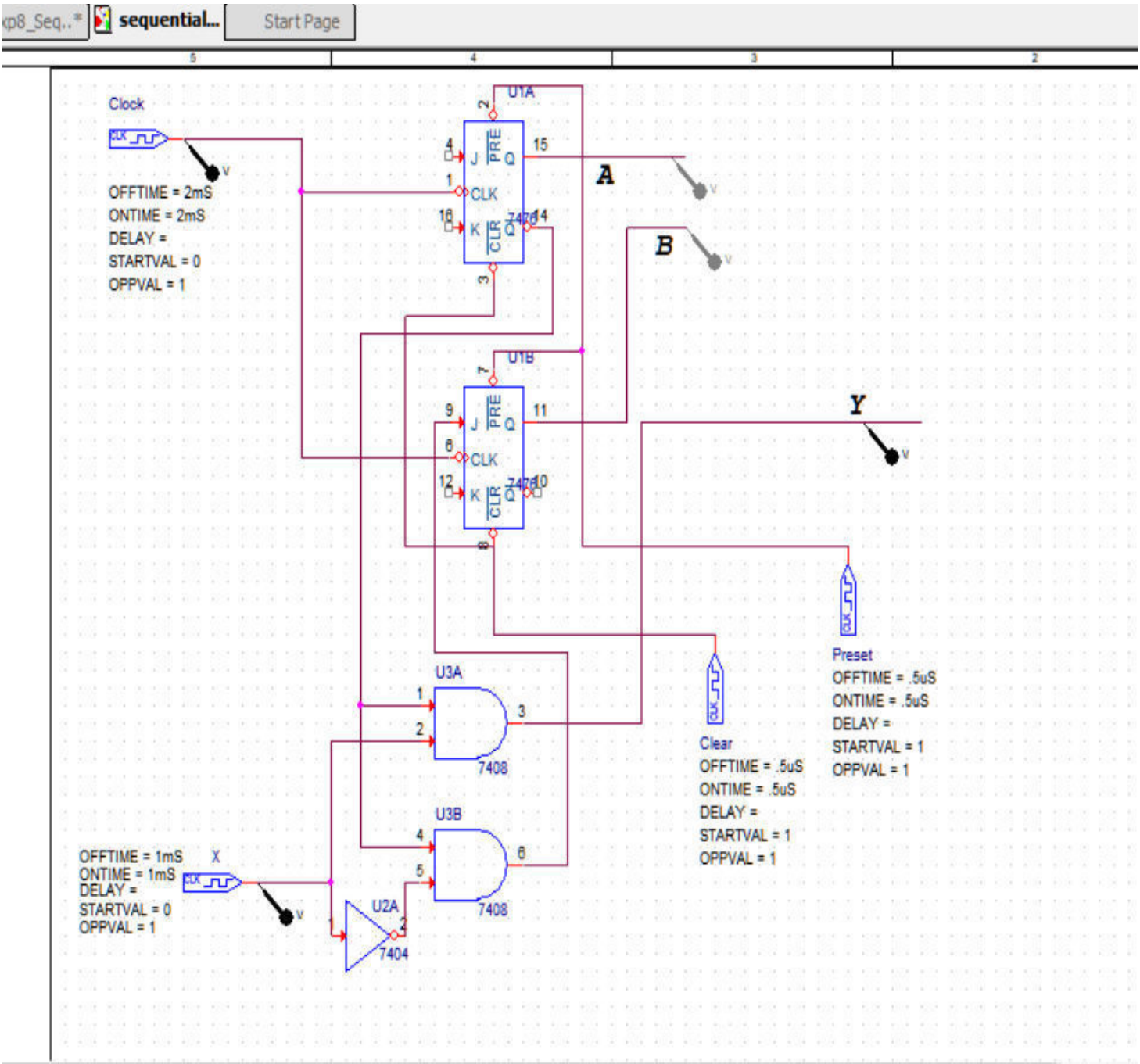
JB = A'X'

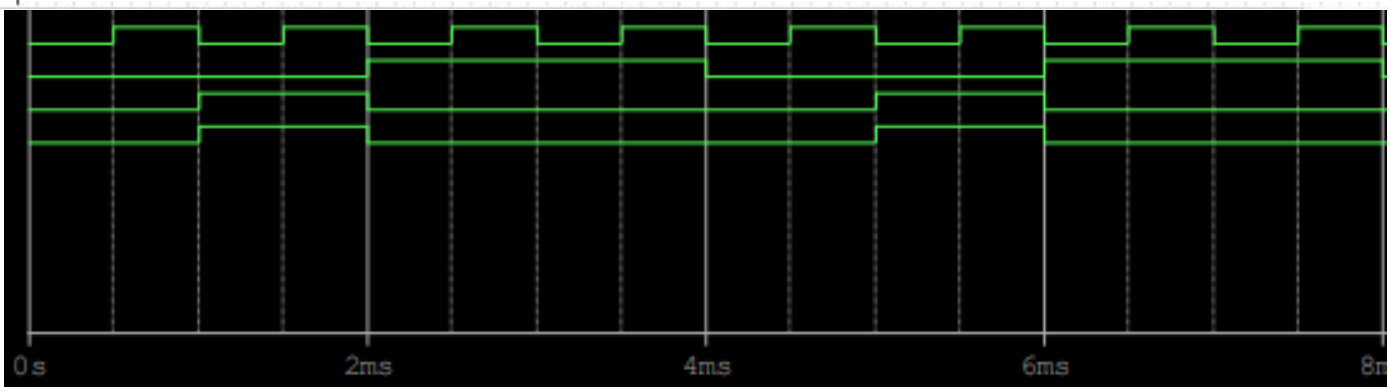
KB = X

Output function is:

Y=A'X

d. Simulate the circuit using software. Softwares may not have the JK master-slave flip-flop IC 7476. Use instead the generic JK flip-flop as you did in experiment 9. In the Lab, build the circuit and check the output to verify the state table values.


sequential...
Start Page
Clock
OFFTIME = 2mS
ONTIME = 2mS
DELAY =
STARTVAL = 0
OPPVAL = 1
U1A
U1B
A
B
Y
U3A
U3B
U2A
7408
7404
Preset
OFFTIME = .5uS
ONTIME = .5uS
DELAY =
STARTVAL = 1
OPPVAL = 1
Clear
OFFTIME = .5uS
ONTIME = .5uS
DELAY =
STARTVAL = 1
OPPVAL = 1
OFFTIME = 1mS
ONTIME = 1mS
DELAY =
STARTVAL = 0
OPPVAL = 1
X


0 s
2ms
4ms
6ms

2. *Synchronous Counters*

Synchronous counters have all clock lines tied to a common clock causing all flip-flops to change at the same time. The count sequence of a counter can be analyzed by placing the counter into every possible number in the sequence and determining the next number in the sequence state diagram is developed as the analysis proceeds. (A state diagram is an illustration of the transitions that occur after each clock pulse).

a) *In the pre-lab using software and then in the lab using hardware chips, design a 2-bit gray code counter using JK flip-flops. The required sequence is the binary equivalent of (0-1-3-2-0). A state diagram for this counter is given in Fig. 2.*

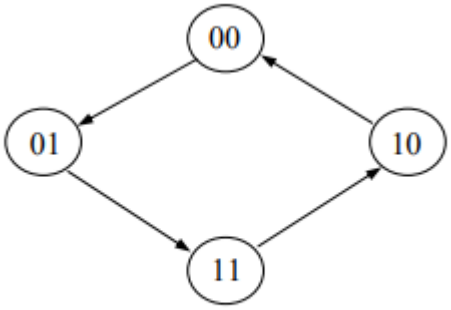
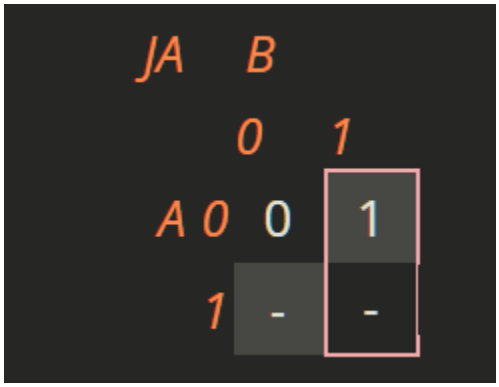


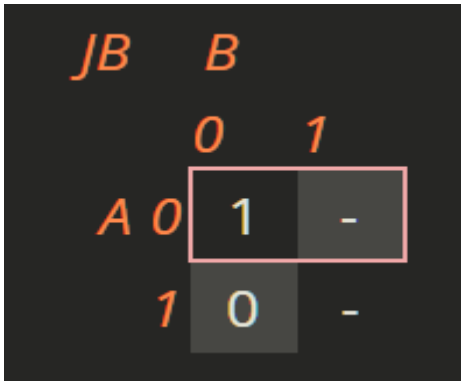
Fig. 2

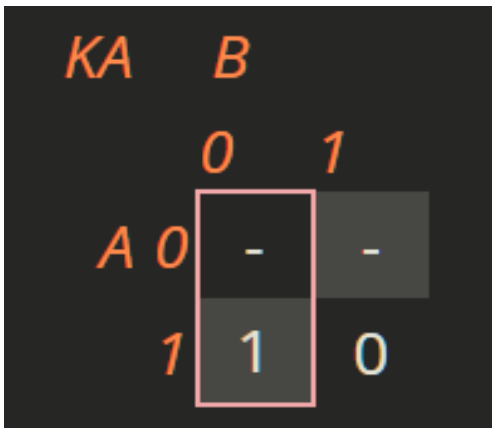
b) *Complete the excitation table (Table 2) for the counter and obtain logic expression for the JK flip-flop input functions.*

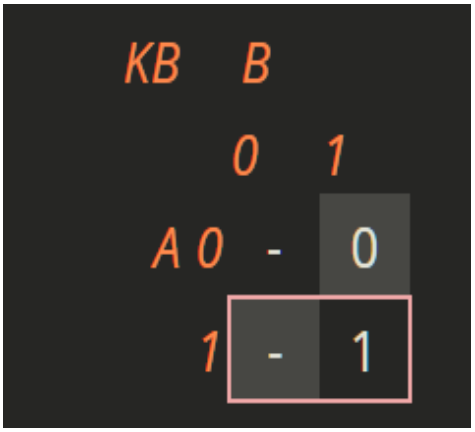
Present_State		Next state		Flip Flop input functions			
A	B	A	B	JA	JB	KA	KB
0	0	0	1	0	X	X	X
0	1	1	1	X	X	X	0
1	0	0	0	0	1	1	X
1	1	1	0	X	0	0	1

The respective KMaps of JA, JB, KA, KB are:







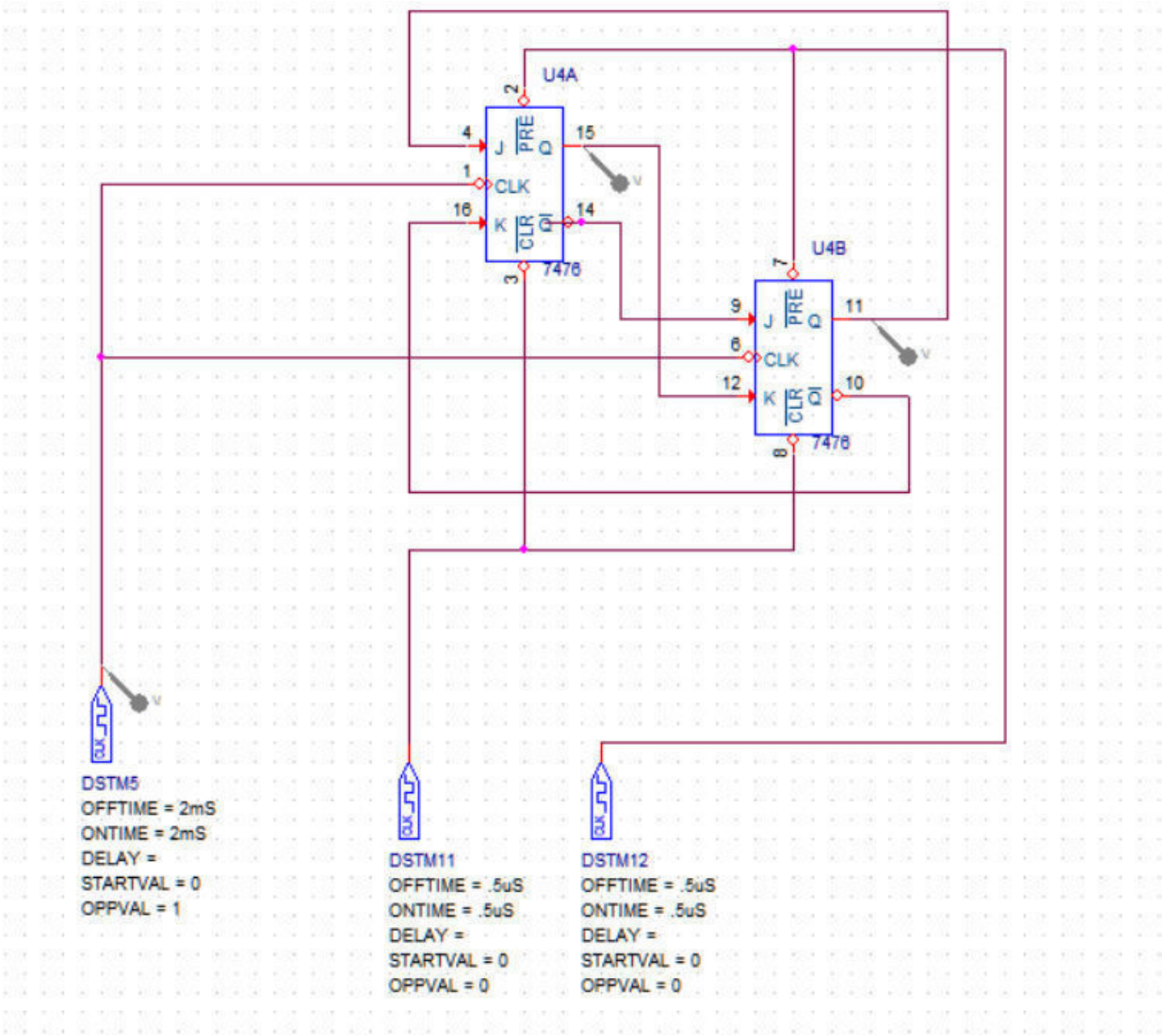


Flip-flop input functions are:

JA=B **KA=B'**

JB=A' **KB=A**

c) *In the lab, build the circuit and test it by pulsing it. Check that the output is the designed sequence.*



3. *Asynchronous Counters*

Asynchronous counters are a series of flip-flops each clocked by the previous state, one after the other. Since all the stages of the counter are not clocked together, a ripple effect propagates as various flip-flops are clocked. For this reason they are called ripple counters. The modulus of a counter is the number of different output states the counter may take (i.e. Mod 4 means the counter has four output states).

a) *In the pre-lab construct a 4-bit asynchronous counter shown in Fig.3. (It is also called binary ripple counter). Use four generic JK flip-flops. Connect four Binary Probes to Q outputs. Connect all R and S inputs to Logic 1 and connect a switch to the CP input.*

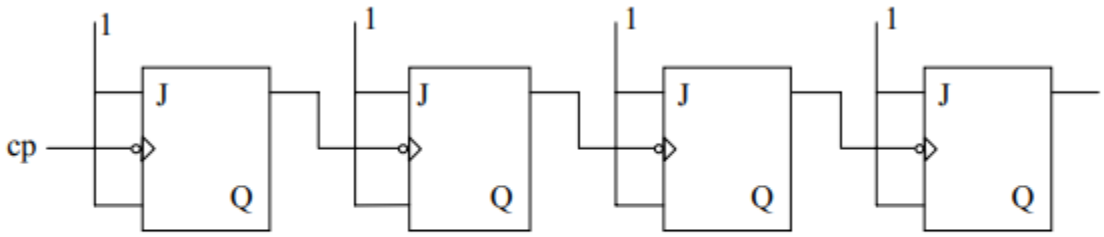
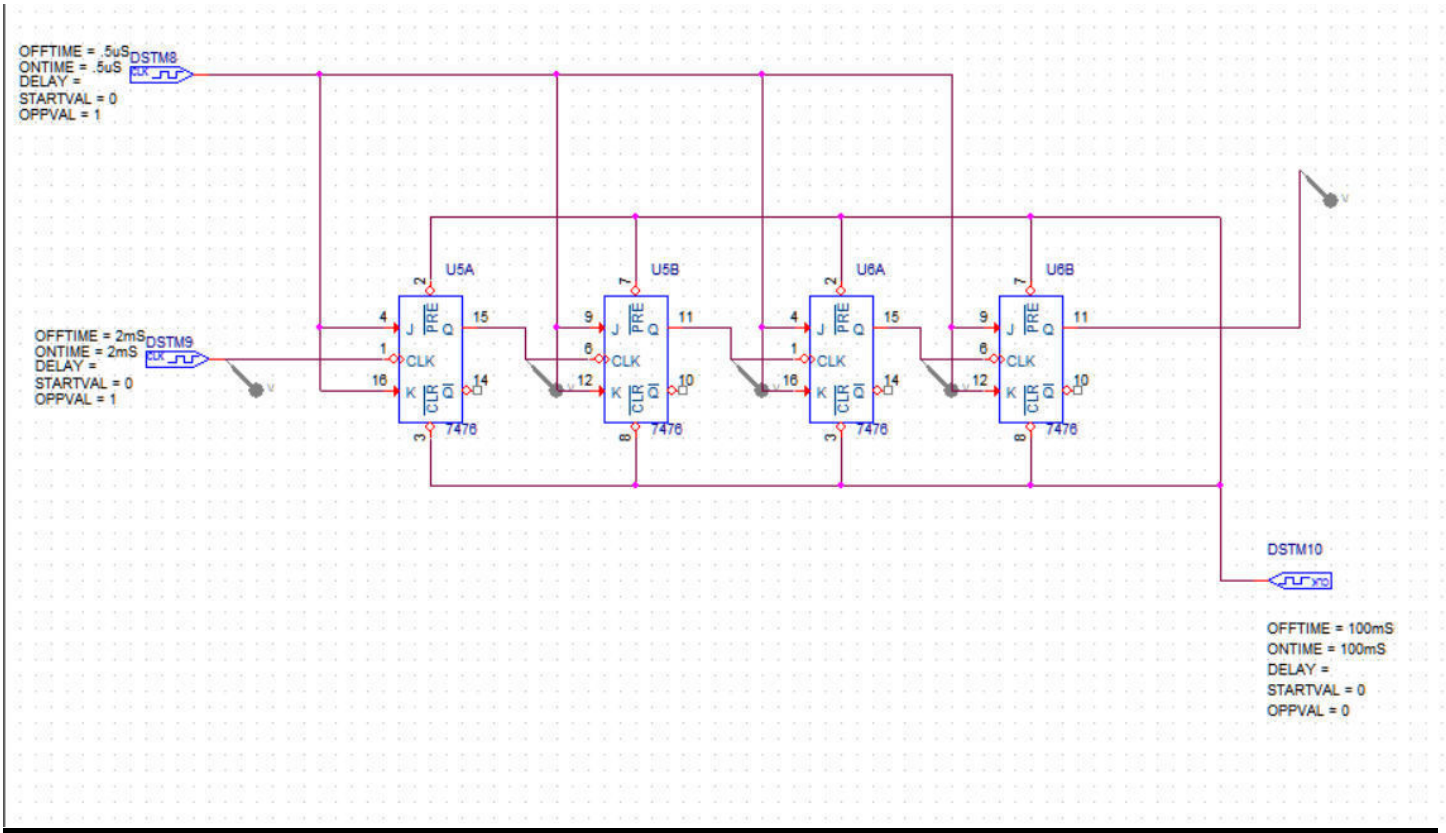
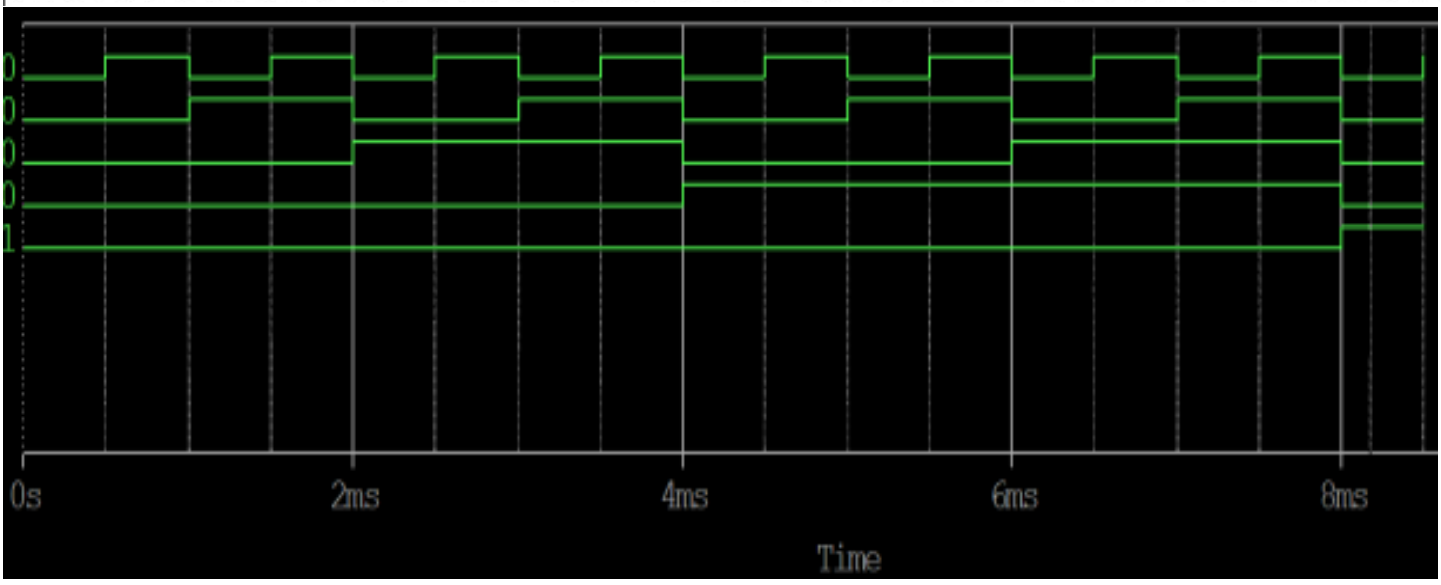


Fig. 3 4-bit ripple counter

b) In the Lab use two 7476 ICs to implement the design. Connect Q outputs of flip-flops to indicator lamps. Connect all clear (CLR) and preset (PRE) inputs to logic 1. Connect the CP input to the pulse output and check the counter for proper operation.





c) Write down the count sequence in Table 3. Identify this count sequence (up or down). Comment on what happens after the application of 15 pulses to CP input.

Table 3. Count sequence for the 4-bit ripple counter.

A	B	C	D
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	0
0	0	1	0
1	0	`1	0
0	1	1	0
1	1	1	0
0	0	0	1
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	1
1	0	1	1
0	1	1	1
1	1	1	1

The above given is up counter.

After the counter reaches upto 16, it resets to 0000 and the cycle again continues.

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Digital Logic Design
CSE1003 LAB

Digital Simulation Using Verilog in Modelsim

Experiment 9

BIMAL PARAJULI
(20BDS0405)

7/18/21

CSE – 1003 (LAB)

Bimal parajuli (20BDS0405)

DIGITAL LOGIC DESIGN (CSE1003)

Exp#9 – Digital Simulation using ModelSim

Objectives:

- To write the Verilog HDL code for various digital circuits.
- Simulate them using the ModelSim Altera Software
- Test them using appropriate test benches.

Softwares Used:

- ModelSim Altera 10.1d (Quartus II 13.0sp1)

Theory:

Verilog is a hardware description language (HDL) used to model electronic systems. It is most commonly used in the design and verification of digital circuits at the register-transfer level of abstraction. It is also used in the design and verification of analog, digital and mixed-signal circuits.

ModelSim is a Software environment developed by Mentor Graphics for the simulation of hardware description languages such as VHDL, Verilog, etc.

In this experiment, we use ModelSim Software Environment to design, model and test various digital Circuits.

Bimal parajuli (20BDS0405)

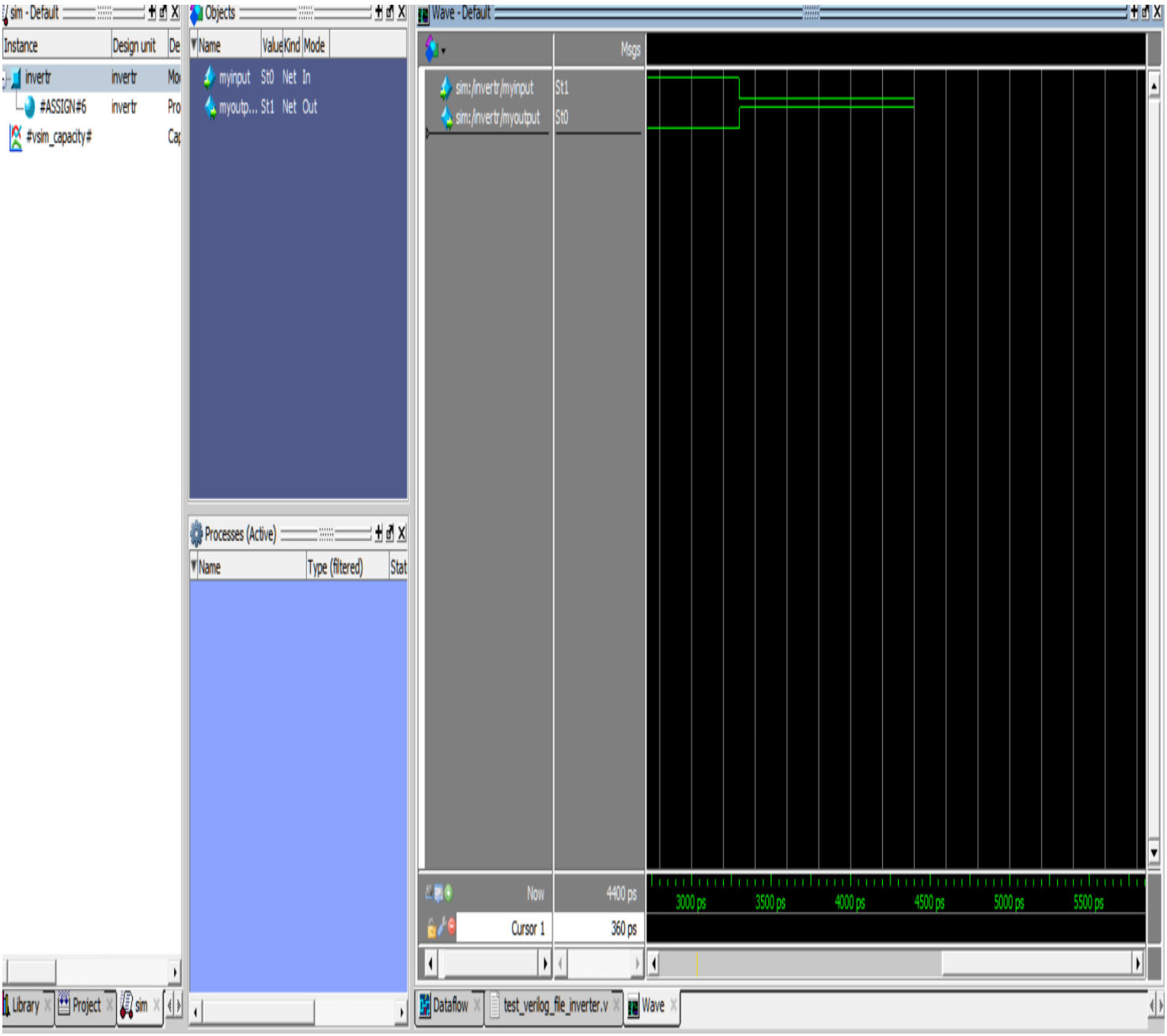
1. Verilog Simulation of a basic Inverter:

C:\altera\13.0sp1\test_verilog_file_inverter.v (/invertr) - Default

```
1   //model an inverter logic gate in modelsim
2
3   module invertr(myinput, myoutput);
4     input myinput;
5     output myoutput;
6     assign myoutput = ~myinput;
7     // if input is 1, output is 0
8     // if input is 0, output is 1
9
10  endmodule
11
```

Dataflow | test_verilog_file_inverter.v | Wave

Bimal parajuli (20BDS0405)



Bimal parajuli (20BDS0405)

2. Verilog Simulation of a half Adder:

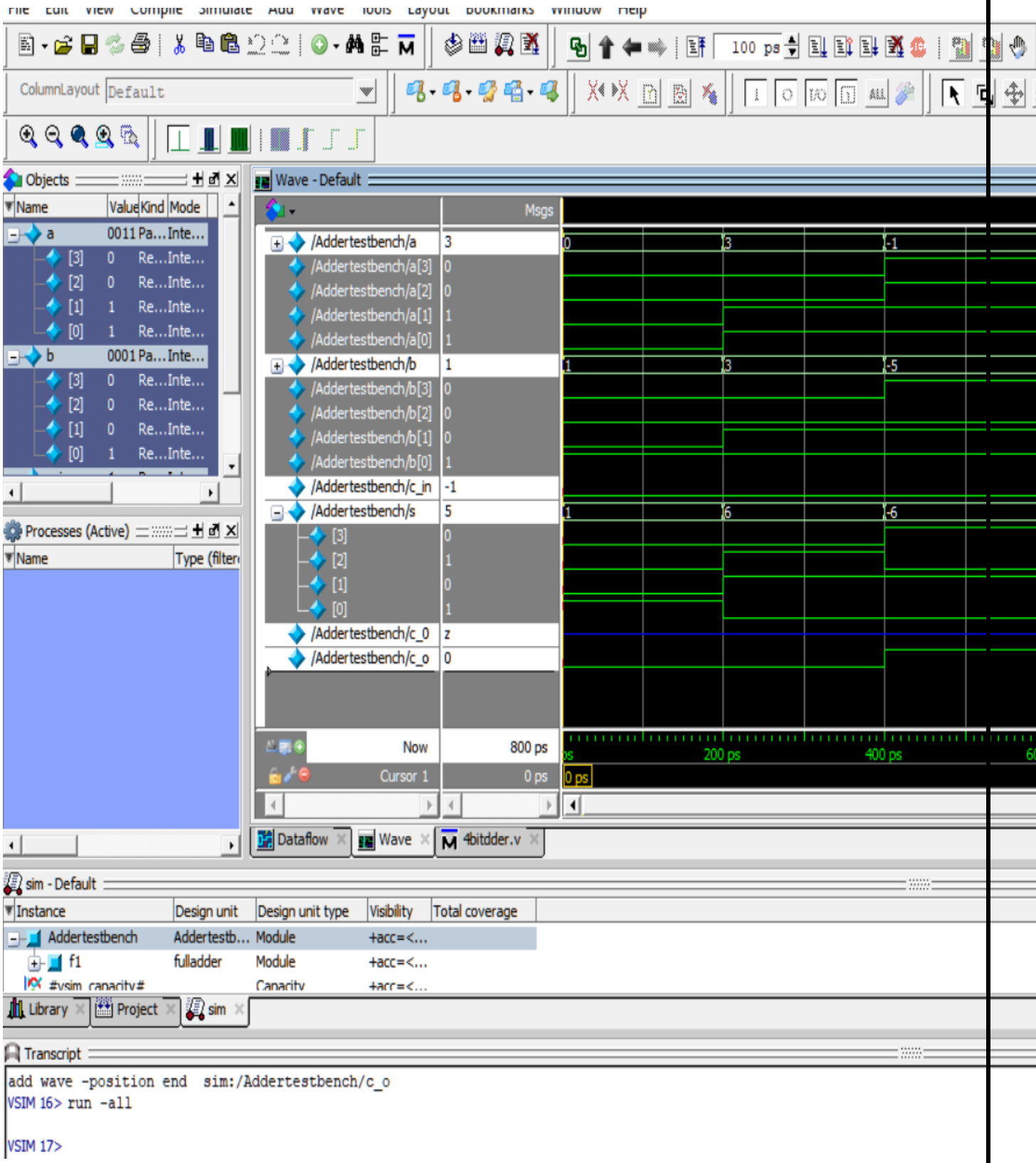
C:/altera/13.0sp1/half_adder.v (/ha_tb)

File Edit View Tools Bookmarks Window Help

C:/altera/13.0sp1/half_adder.v (/ha_tb) - Default

```
Ln#
 1
 2  ////////////////////////////////////////
 3  //////Half adder Verilog Code////////////
 4
 5
 6  module half_adder(A,B,Sum,Cout);
 7
 8    input A,B;
 9    output Sum, Cout;
10    xor(Sum,A,B);
11    and(Cout,A,B);
12
13  endmodule
14
15
16
17  ////////////////////////////////////////////
18  //////Test bench for half adder Circuit//////
19
20  module ha_tb();
21    reg a,b;
22    wire s,c;
23
24    half_adder a1(a,b,s,cout);
25  initial
26      begin
27        a=1'b0; b=1'b0;
28        #100
29
30        a=1'b1; b=1'b0;
31        #100
32
33        a=1'b0; b=1'b1;
34        #100
35
36        a=1'b1; b=1'b1;
37
38      end
39  endmodule
40
```

Bimal parajuli (20BDS0405)



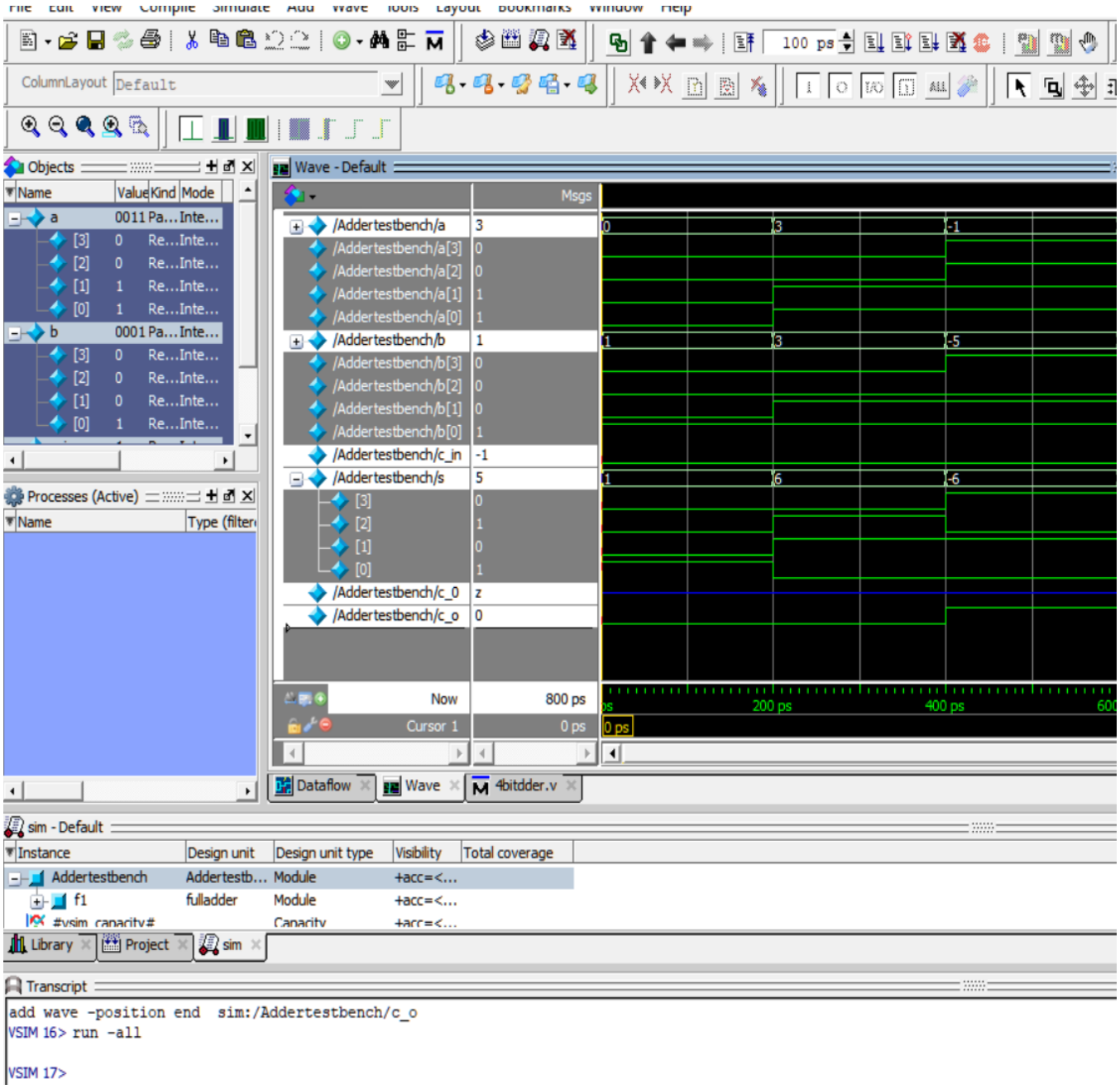
Bimal parajuli (20BDS0405)

3. Verilog Simulation of a Full Adder:

C:/altera/13.0sp1/4bitdder.v (/Addertestbench) - Default

```
Ln#
 1  ////////////////////////////////
 2  module fulladder(A,B,Cin,Sum,Cout);
 3
 4    input [3:0] A, B;
 5    input Cin;
 6
 7    output [3:0] Sum;
 8    output Cout;
 9
10    //Specifying the function of a full adder
11    assign {Cout,Sum} = A + B + Cin;
12  endmodule
13
14  ////////////////////////////////////
15
16  ////////////////////////////////////
17
18  ///////Test bench for above Module///
19  module Addertestbench();
20    reg[3:0] a,b;
21    reg c_in;
22    wire [3:0] s;
23    wire c_0;
24
25    fulladder f1(.A(a),.B(b),.Cin(c_in),.Sum(s),.Cout(c_o));
26
27  initial
28    begin
29         a= 4'b0000; b=4'b0001; c_in =1'b0;
30         #200;
31
32         a= 4'b0011; b=4'b0011; c_in =1'b0;
33         #200;
34
35         a= 4'b1111; b=4'b1011; c_in =1'b0;
36         #200;
37
38         a= 4'b0011; b=4'b0001; c_in =1'b1;
39         #200;
40       end
41  endmodule
42
```

Bimal parajuli (20BDS0405)



Bimal parajuli (20BDS0405)

4. Verilog Simulation of a 2 to 4 decoder:

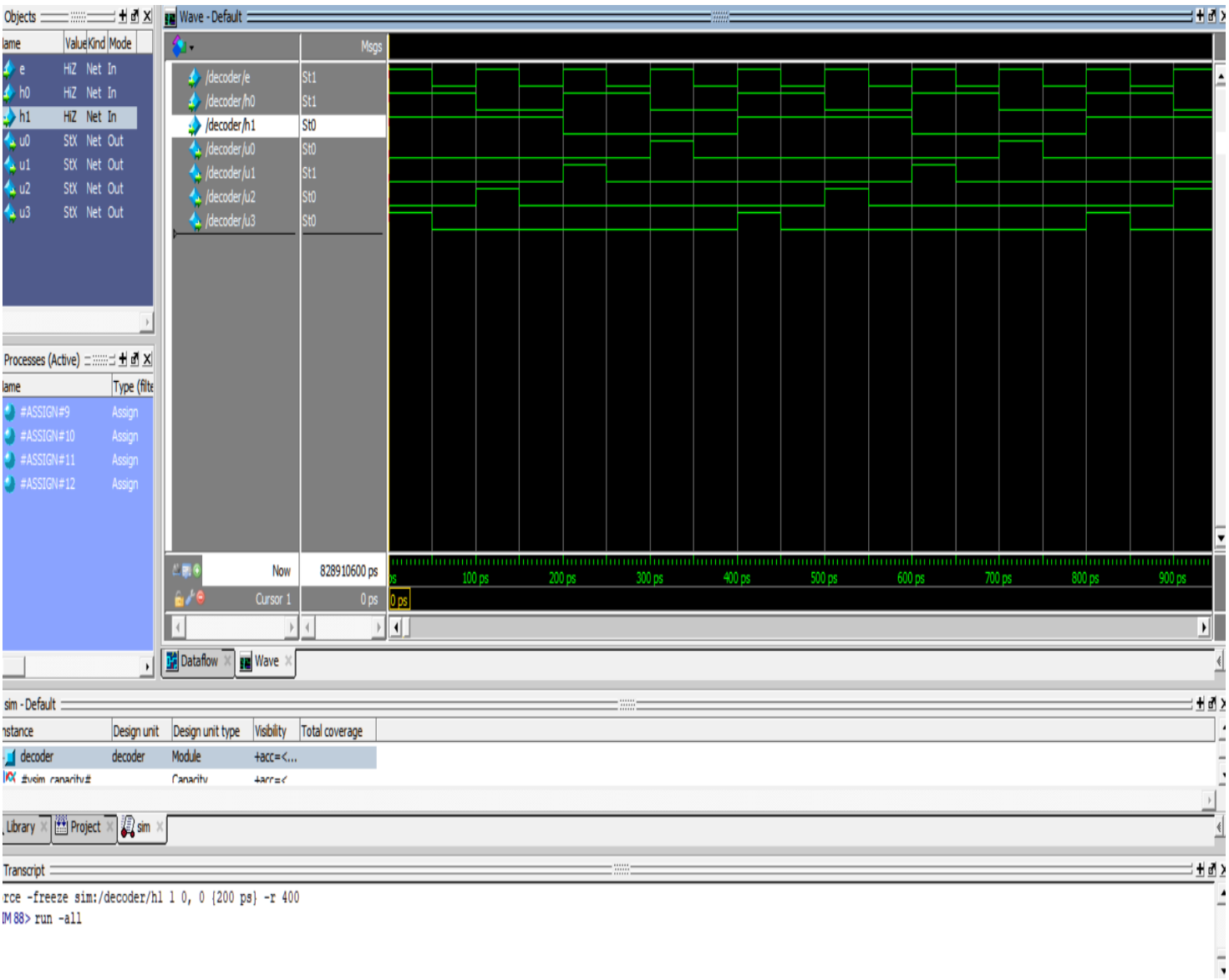
C:/altera/13.0sp1/2_to_4_decoder.v (/decoder)

File Edit View Tools Bookmarks Window Help

C:/altera/13.0sp1/2_to_4_decoder.v (/decoder) - Default

```
Ln#
 1  ////////////////////////////////////////////////////////////////////
 2  ////////////////////2 to 4 decoder in verilog ///////////////////////
 3
 4  module decoder(u0,u1,u2,u3,e,h0,h1);
 5
 6      input e,h0,h1;
 7      output u0, u1, u2, u3;
 8
 9      assign u0= (e & ~h1 & ~h0);
10      assign u1= (e & ~h1 & h0);
11      assign u2= (e & h1 & ~h0);
12      assign u3= (e & h1 & h0);
13
14    endmodule
15
16    ////////////////////////////////////////////////////////////////////
17    /////////////Test bench for 2 to 4 decoder ///////////////////////
18
19    module decoder_tb();
20
21      reg e,h0,h1;
22      wire u0,u1,u2,u3;
23
24      decoder d1(u0, u1, u2, u3, e, h0, h1);
25
26      initial
27        begin
28          e=0; h0=1; h1=0;
29          #100;
30          e=1; h0=0; h1=0;
31          #100;
32          e=1; h0=0; h1=1;
33          #100;
34          e=1; h0=1; h1=0;
35          #100;
36          e=1; h0=1; h1=1;
37        end
38      endmodule
39
```

Bimal parajuli (20BDS0405)



Bimal parajuli (20BDS0405)

5. Verilog Simulation of 2421 to 53-1-1 Code Converter (DA_2).

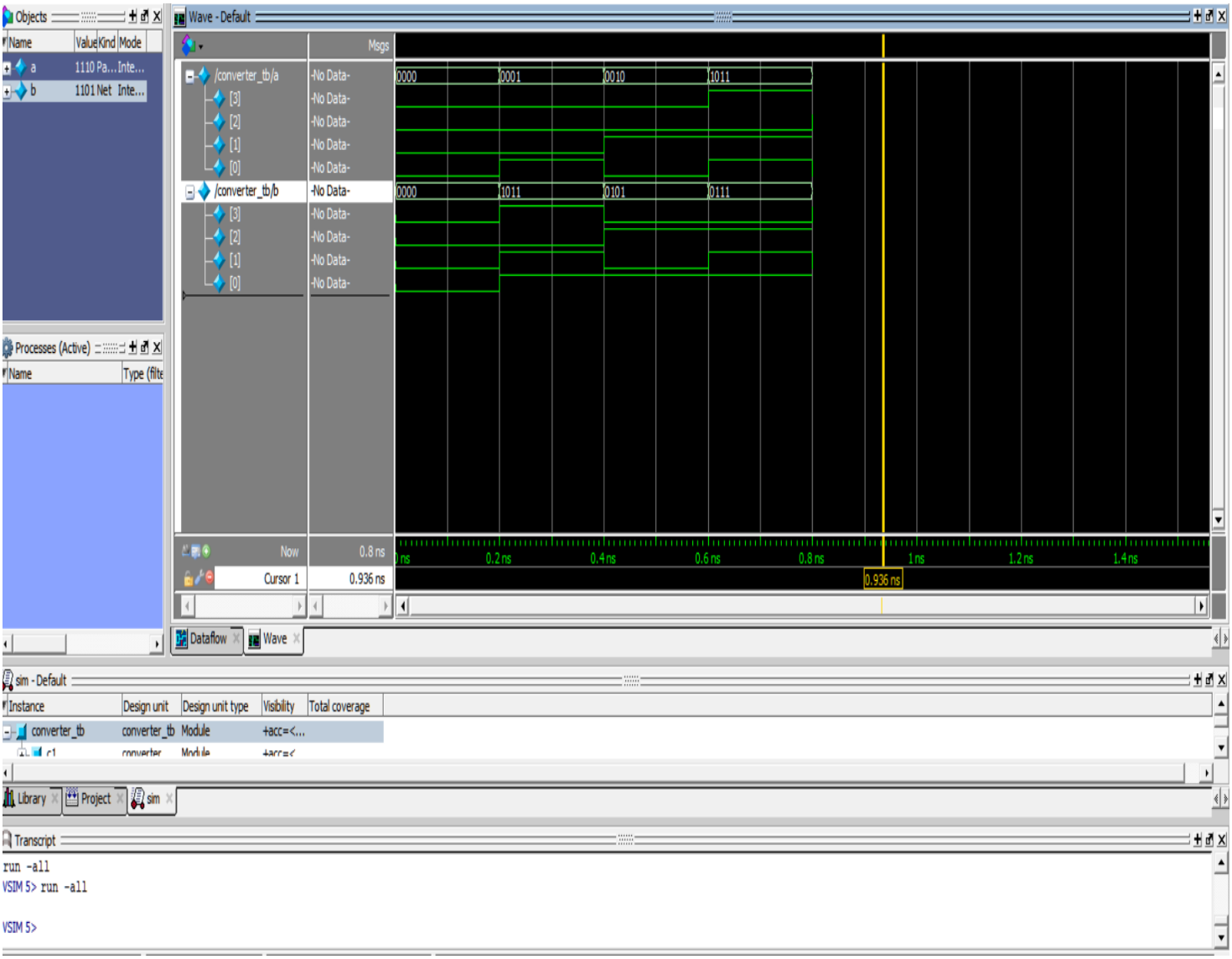
C:/altera/13.0sp1/code_Converter.v (/converter_tb)

File Edit View Tools Bookmarks Window Help

C:/altera/13.0sp1/code_Converter.v (/converter_tb) - Default

Ln#

```
1  ////////////////////////////////////////////////////////////////////////////////
2  ///Verilog Code to design a module for converting 4 bit 2421 to 53-1-1 code///////
3
4  module converter(A,B);
5
6    input [3:0] A;
7    output [3:0] B;
8
9        assign B[0] = A[1] | A[0] | (A[2] & A[3]);
10       assign B[1] = (~A[2] & A[3]) | (A[2] & ~A[3]) | (~A[2] & A[0]);
11       assign B[2] = (A[1] & ~A[2]) | (~A[2] & A[3] ) | (~A[0] & A[2]);
12       assign B[3] = (A[0] & ~A[2] & ~A[3]) | ( ~A[0] & A[3] );
13 endmodule
14
15
16 ////////////////////////////////////////////////
17 ///////Test bench for above code converte/////////
18
19 module converter_tb();
20   reg[3:0] a;
21   wire [3:0] b;
22
23   converter c1(.A(a), .B(b));
24
25   initial
26       begin
27           a= 4'b0000;
28           #200;
29           a= 4'b0001;
30           #200;
31           a= 4'b0010;
32           #200;
33           a= 4'b1011;
34           #200;
35           a= 4'b1110;
36       end
37 endmodule
38
```



CONCLUSION/ INFERENCES:

From the above experiments, we can observe that any digital circuits can be simulated by writing a proper Verilog(HDL) code of the corresponding circuit. HDL compilers like ModelSim help to convert the code into it's circuit equivalent which can further be tested with a test bench and the design can be further implemented in FPGAs if required.