Lab Manual

CS302 – Digital Logic Design

Prepared by

Waqar Ahmad, Lecturer
Waqas Ahmad, Lecturer

Table of Contents

Lab Experiment # 01

Familiarization; Playing with EWB 5.12

Introduction to Electronics Workbench

Electronics Workbench is an electronics and digital logic lab inside a computer, modeled after a real electronics workbench. It is a design tool that provides you with components & instruments to create "virtual" board-level designs:

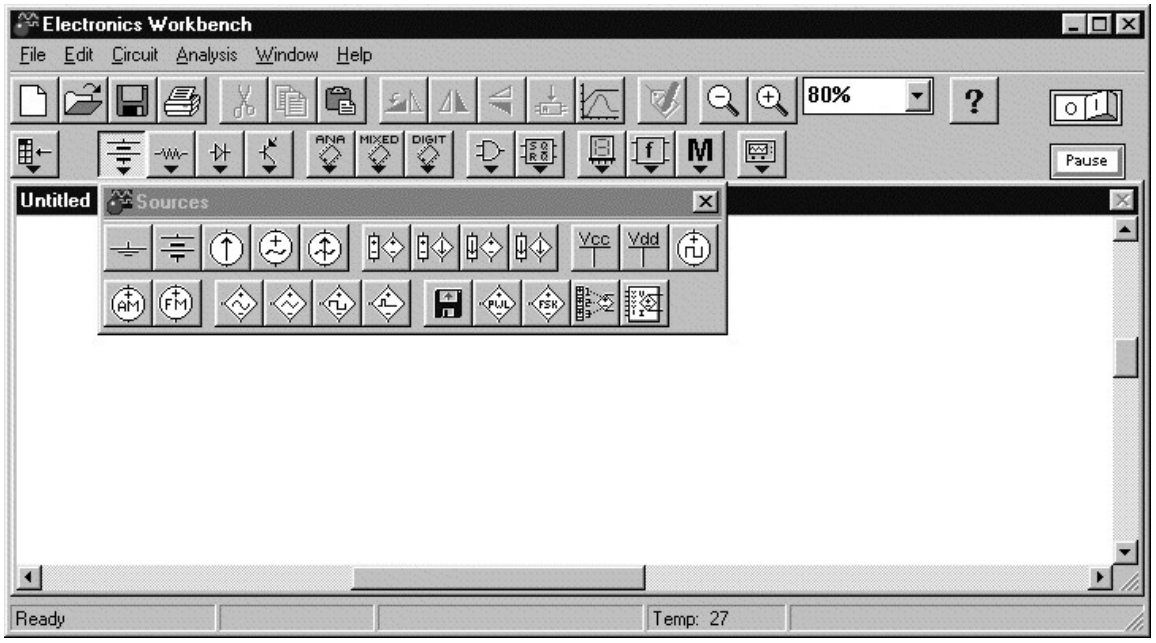
– No actual breadboards, components, or instruments needed.

– Click-and-drag schematic editing.

– It offers mixed analog & digital simulation and graphical waveform analysis.

– Circuit behavior simulated realistically.

– Results displayed on multimeter, oscilloscope, bode plotter, logic analyzer, etc.



The main GUI interface of EWB

Using Electronics Workbench for Design

You may use EWB to:

1- Explore ideas and test preliminary circuits.

2- Refine circuits to full layout (If circuit requires parts of a previous design)

3- Export files in format used by PCB (Printed Circuit Board) layout packages as move from design to production.

General EWB Functions

Selecting

– To move a component or instrument need to select it selected item highlights: components red, wires thicken

– *Clicking to Select*

To select single item, click on it.

To select additional items, press CTRL+ click.

– *Selecting All*

Choose Edit/Select All.

– *Dragging to Select*

Place pointer above & to side of group of items. Press & hold mouse button & drag downward diagonally. Release mouse button when rectangle encloses everything desired.

– *Deselecting*

To deselect single item, press CTRL+click.

To deselect all selected items, click on empty spot in window.

Setting Labels, Wiring

Setting Labels, Values, Models & Reference IDs,

– To set labels, values (for simple components) & models (for complex components), select component and choose Circuit/Component Properties, choose desired tab, make any changes, and click OK.

– Can also invoke Circuit/Component Properties box by double-clicking on component.

***** *Notes:*

The Circuit/Component Properties box contains a number of tabs; depending on which component is selected an analog component has either a value or a model, not both.

Wiring Components

– Point to a component's terminal so it highlights; press & hold mouse button, and drag so a wire appears drag wire to a terminal on another component or to an instrument connection, when terminal on second component or instrument highlights, release mouse button

Inserting, Connecting, Editing

Inserting Components

– To insert component into existing circuit, place it on top of wire; it will automatically be inserted if there is room.

Connecting Wires

– If drag a wire from a component's terminal to another wire, a **connector** is automatically created when

you release mouse button.

– Note: a connector button also appears in the Basic toolbar (to insert connectors into an existing circuit).

Deleting Wires

– To delete a wire, select it & choose Edit/Delete

– Alternatively, disconnect wire by selecting one end of it & moving it to an open spot on circuit window.

Changing Wire Color

– To change a wire's color, double-click it & choose Schematic Options tab; click the Color button & choose a new color.

Straightening a Wire

– move wire itself.

– move component to which wire is attached.

– press ALT and move component to which wire is attached.

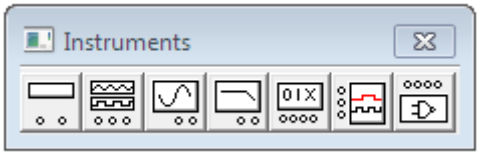
– select component and press appropriate arrow key to align it.

– If two wires cross in a way that makes them hard to follow, select one & drag it to new location

**Note*:

- the way a wire is routed sometimes depends on terminal from which wire was dragged; try disconnecting routed wire & then rewire from the opposite terminal.

Instruments



– *Using an Instrument Icon*

To display the Instruments toolbar, click the Instruments button on the Parts Bin toolbar.

To place an instrument on the circuit window, drag the desired button from the Instruments toolbar to the window. To attach an instrument to a circuit, point to a terminal on its icon so it highlights and drag a wire to a component. To remove an instrument icon, select it & choose Edit/Delete

– *Opening an Instrument*

Double-click the instrument's icon to see its controls

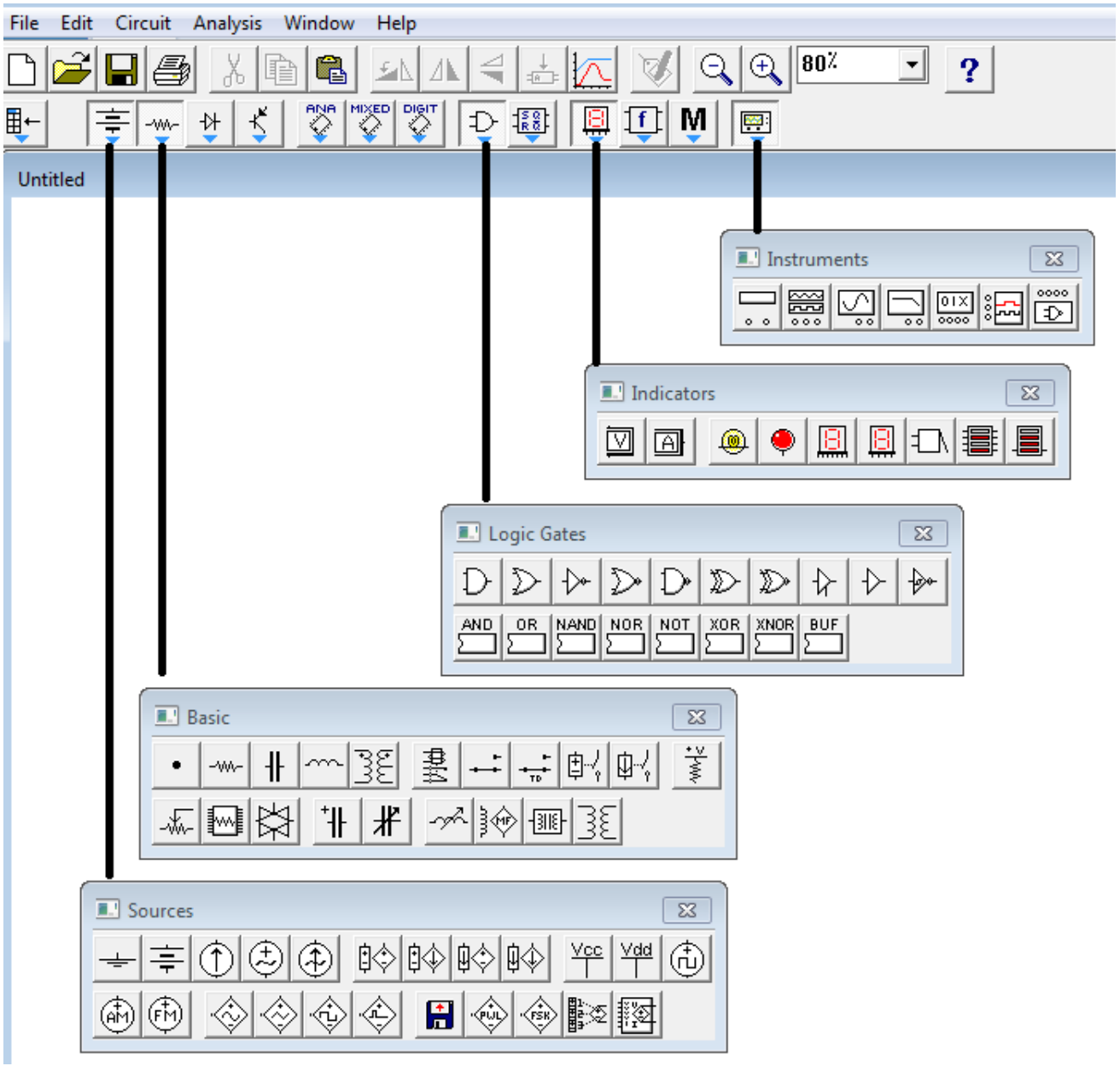
- To selection options, click buttons on the controls
- To change values or units, click the up/down arrows.

Simulation

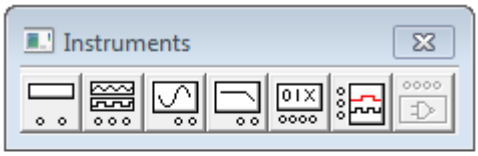
Click the power switch to turn power on. Click switch again to turn power off. (Note: Turning off power erases data & instrument traces.

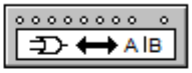
Lab Tasks

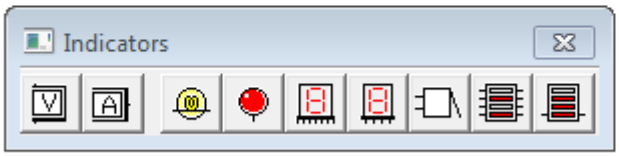
Task 1: Name the basic toolboxes of EWB

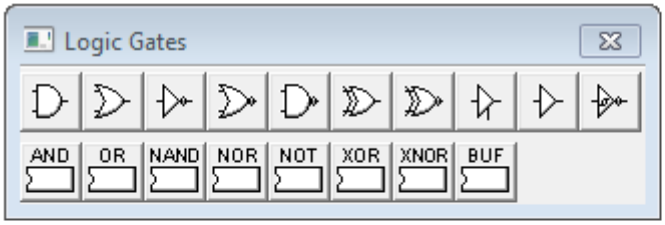


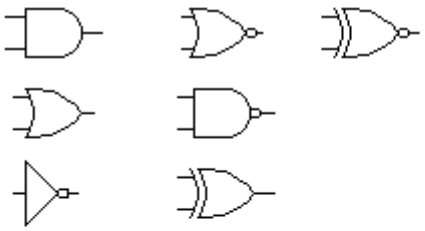
Task 2: Basic buttons in EWB toolboxes

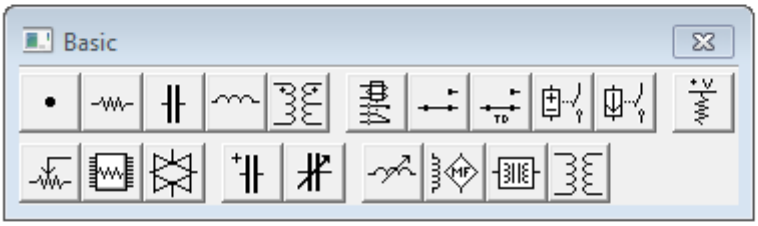


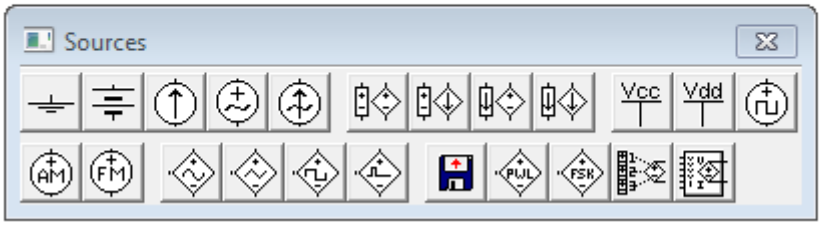




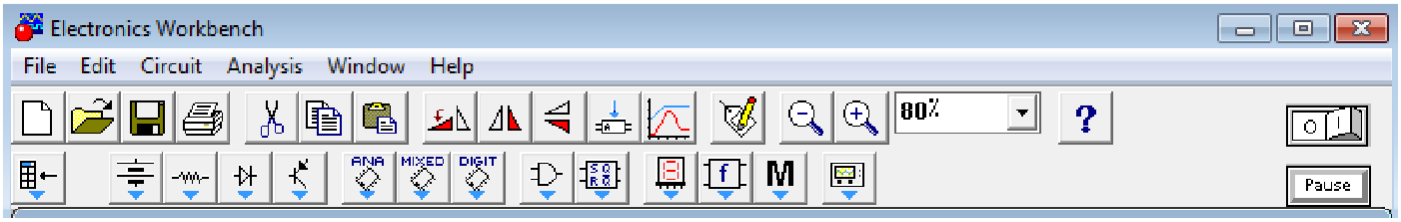






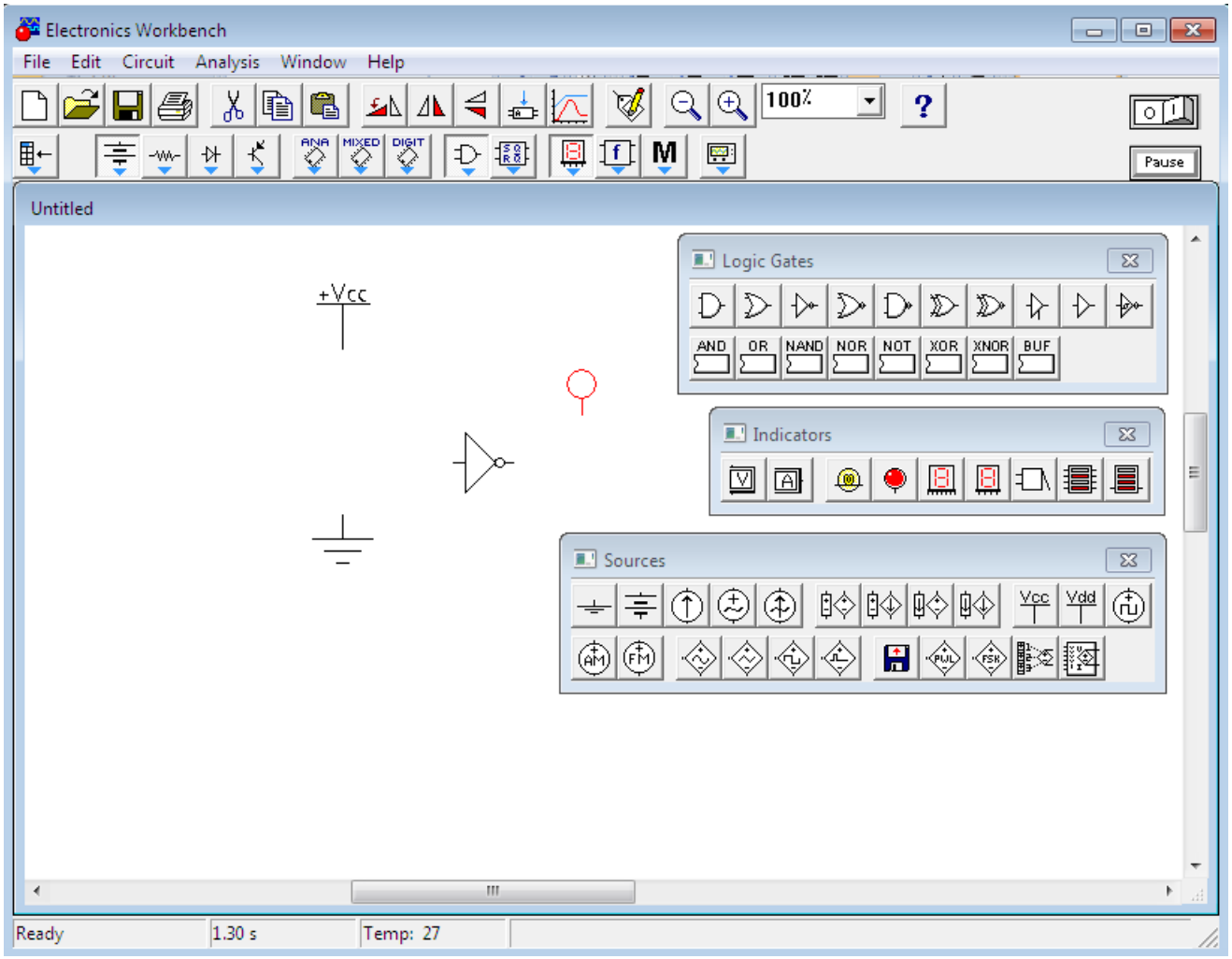


Task 3 EWB Toolbar



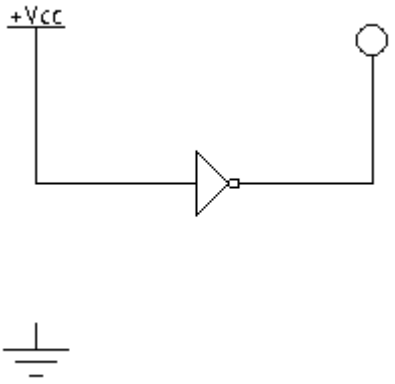
Task 4: Simple circuit; playing with EWB

In the following circuit

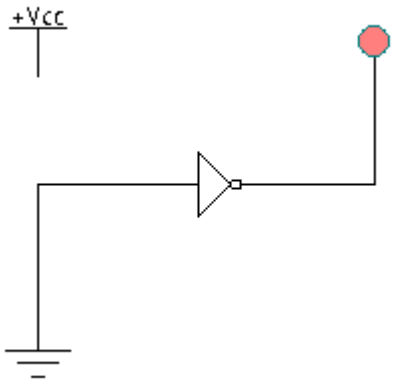


Draw the following circuit. After that make the following changes

- Connect the output of the converter to the red probe
- Connect the Vcc line to the input of the inverter
- Start simulating the circuit State your observation down: **Observation**:



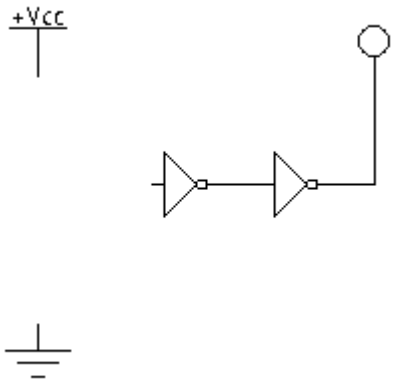
- In the same circuit above, stop the simulation and connect the ground to the input of the inverter. State your observation down:



Observation:

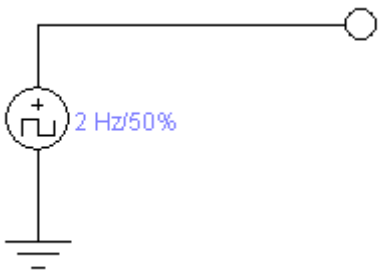
Task 5: Simple circuit; two inverters connected serially

Repeat Task 2 of this report and state down your observations.

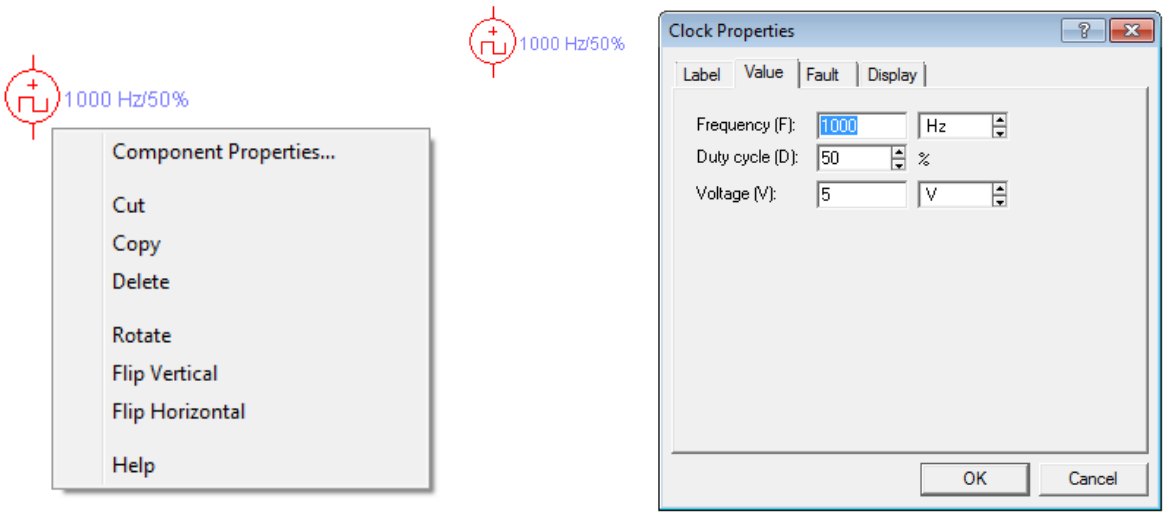


Task 6: Simple circuit; a clock source with a red probe

Draw the following circuit and simulate it. Write down your observations. Notice that the **clock** (from *Sources* toolbox) frequency is 2 Hz.

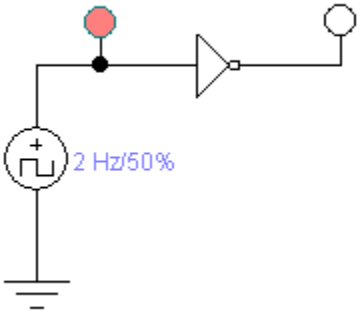


Note: You can change the default values of the clock by doing mouse right clicking on the clock and click on the "Component Properties ..." as shown below:



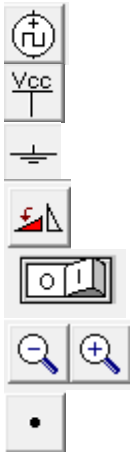
Task 7: Simple circuit; a clock source with two red probes

Draw the following circuit and simulate it. Write down your observations. Notice that the clock frequency is 2 Hz.



Task 8: EWB Menu

Name the following icons and state down their functions



Lab Experiment # 02

Basic logic Gates (AND, OR, and NOT gates)

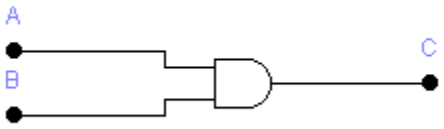
Objectives

1- To study and understand the 3 basic gates.
2- Implement the basic gate in EWB.
3- The study the specifications of every gate when connected it with one input constant and the other is variable.

AND and NAND gates

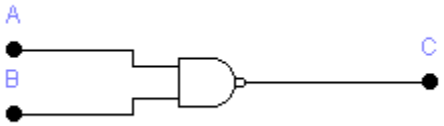
This gate gives high output (1) if all the inputs are 1's. otherwise the output will be low (0).

Its Boolean algebra representation is: C=A.B
And its truth table and schema as following:



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

The NAND gate works opposite to the AND gate. Its Boolean algebra representation is: C= (A.B)'
And it's truth table and schema as following:

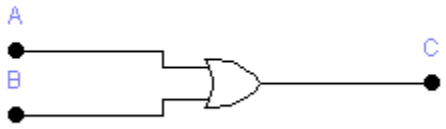


A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

OR and NOR gates

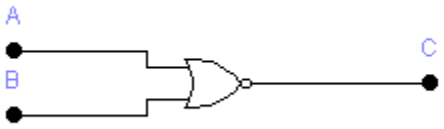
This circuit will give high output (1) if any input is high (1).

Its Boolean algebra representation is: C=A+B and its truth table and schema as following:



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

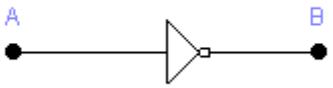
The NOR gate works opposite to the OR gate. Its Boolean algebra representation is: C=(A+B)'
And it's truth table and schema as following:



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

NOT gate

This is the simplest gate it just inverts the input, if the input is high the output will be low and conversely.
So, B =A'

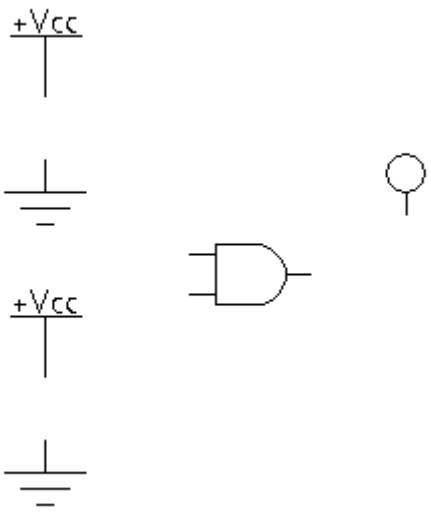


A	B
0	1
1	0

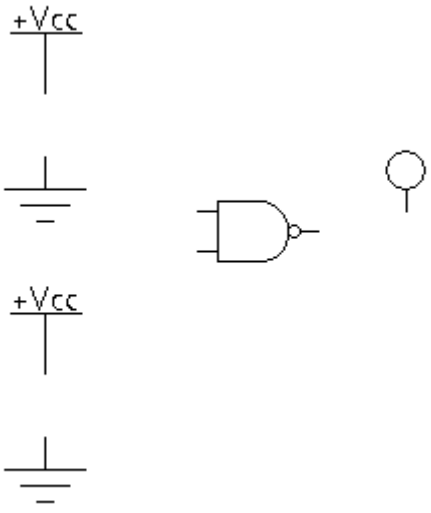
Lab Tasks

Task 1: The AND and NAND gates

In EWB, draw the following two circuits and fill the truth table below

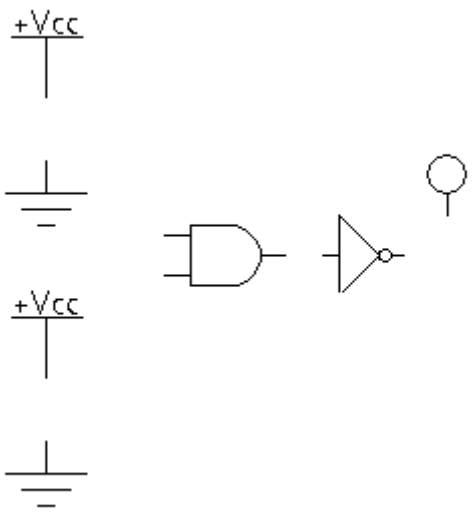


A	B	A.B	(A.B)’
0	0		
0	1		
1	0		
1	1		



Task 2: The AND-NOT combination

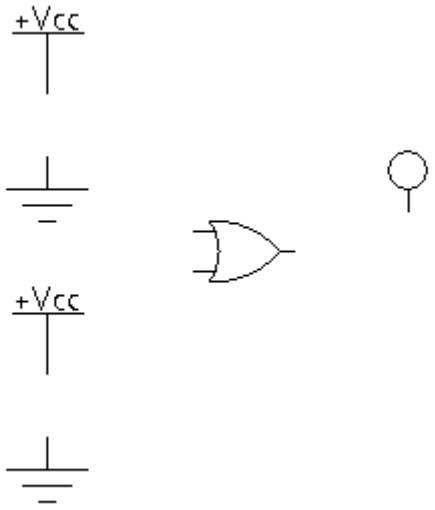
In EWB, draw the following circuit and fill the truth table

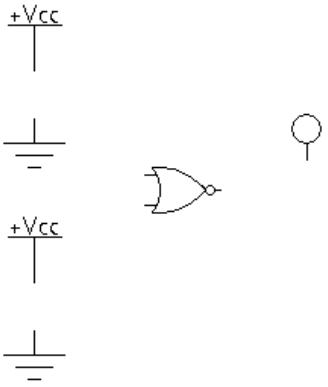


A	B	(A.B)’
0	0	
0	1	
1	0	
1	1	

Task 3: The OR and NOR gates

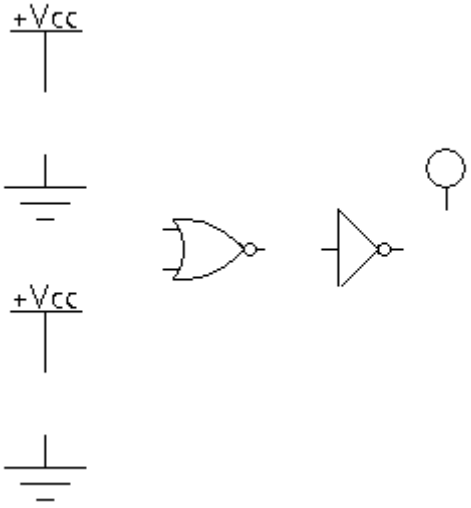
In EWB, draw the following two circuits and fill the truth table below





A	B	A+B	(A+B)’
0	0		
0	1		
1	0		
1	1		

Task 4: The NOR-NOT combination



A	B	((A+B)')'
0	0	
0	1	
1	0	
1	1	

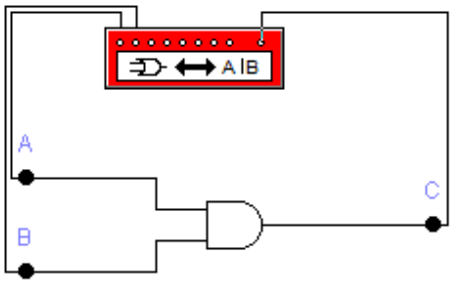
Task 5: Finding the truth table of a gate using the logic converter

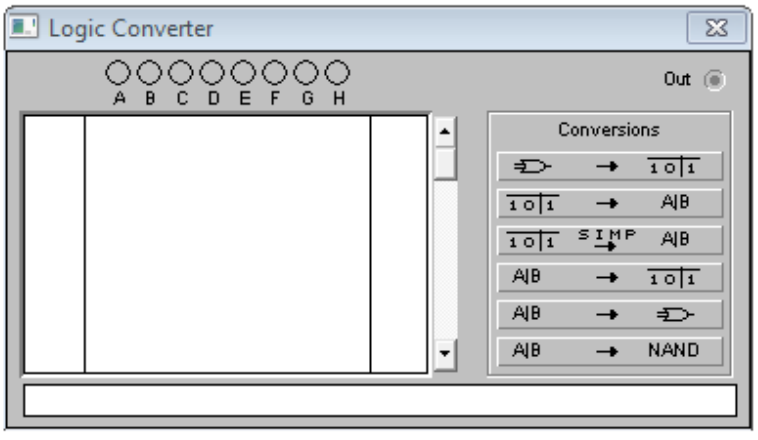
The logic converter can be found in the *Instruments* toolbox. It can be used to derive a truth table from a circuit schematic:

1. Attach the input terminals of the logic converter to up to eight input points in the circuit.
2. Connect the single output of the circuit to the output terminal on the logic converter icon.
3. Click the Circuit to Truth Table  button.

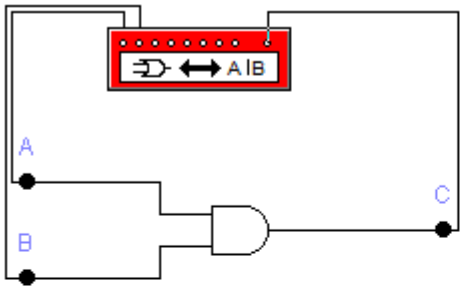
The truth table for the circuit appears in the logic converter's display.

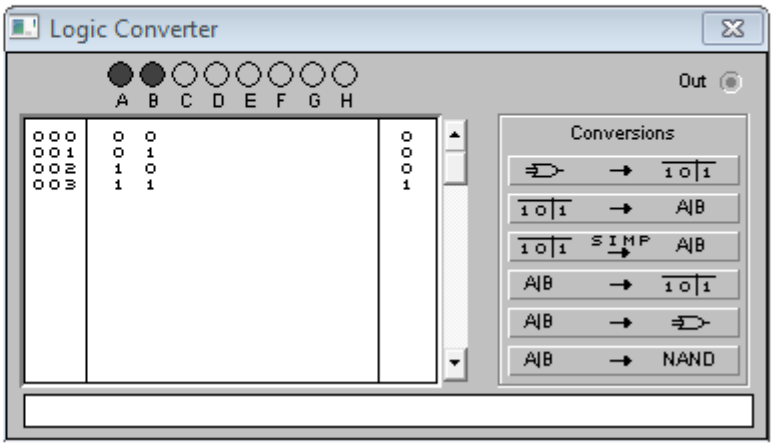
In the following circuit, we will be examining the AND gate. The two inputs of the gate are attached the A and B inputs of the logic converter. The circuit output C is connected to *Outl*ine of the logic converter.





After clicking on the Truth Table button of the logic converter, the logic converter tries all possible combinations of the circuit input and derives its truth table.

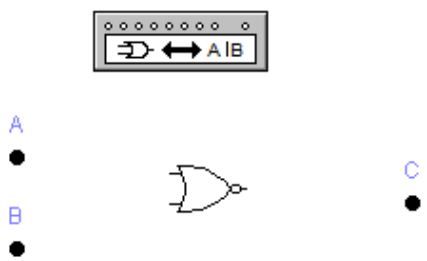


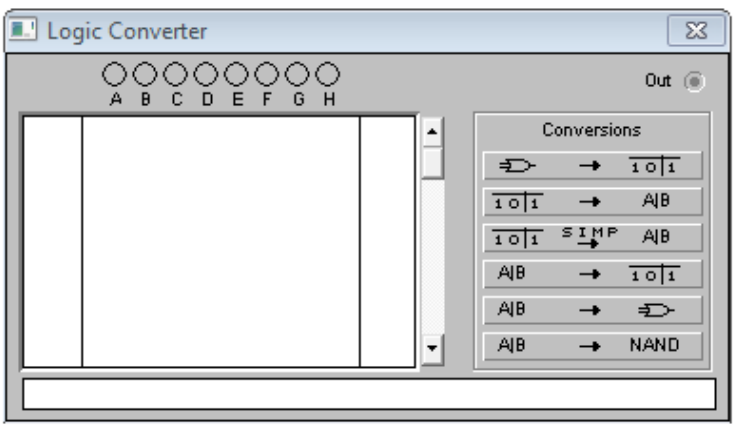


Task 6: Finding the truth table of a gate using the logic converter

Repeat what you did in task 5 for the NOR gate. Show your connections in the circuit below.

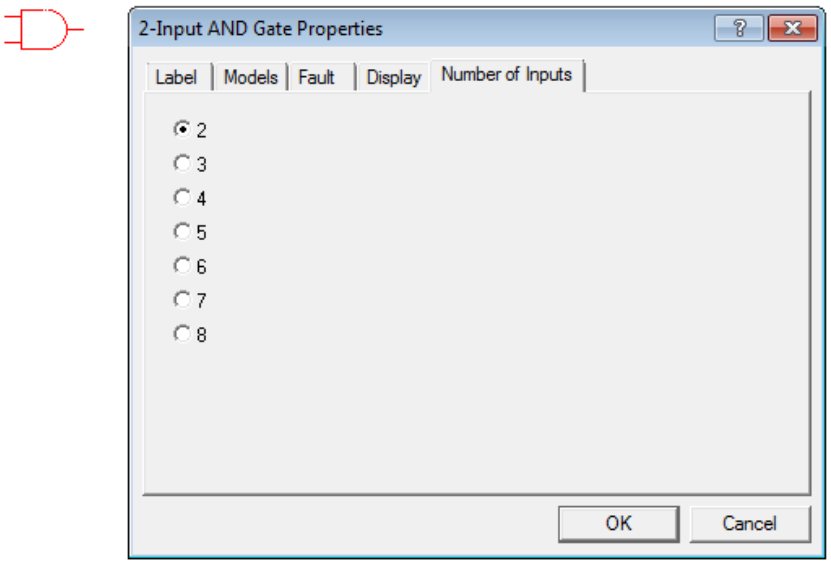
A	B	A+B	(A+B)'
0	0		
0	1		
1	0		
1	1		

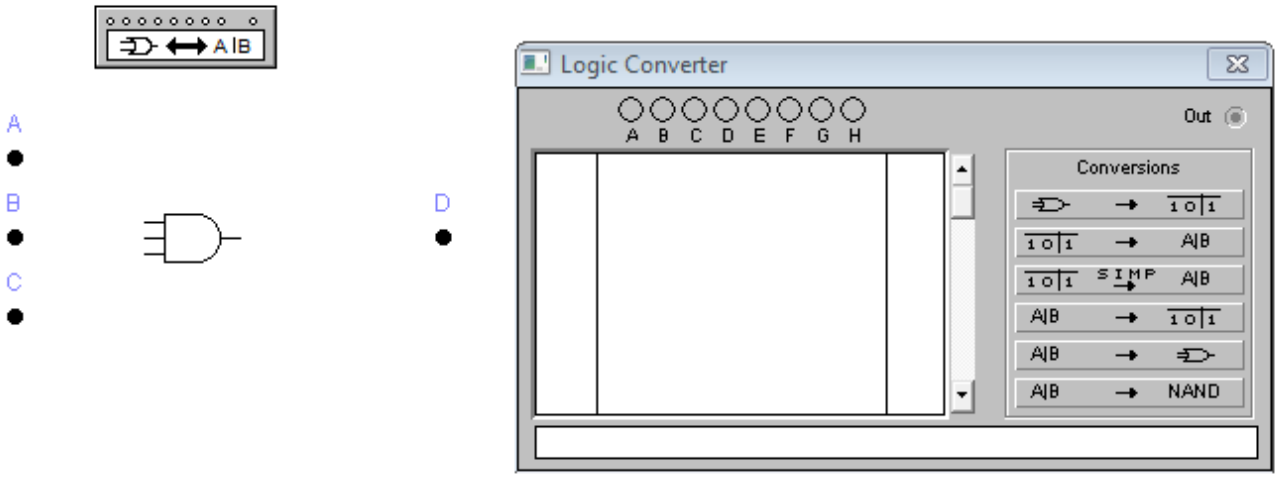




Task 7: Finding the truth table of a three-input gate using the logic converter

Repeat what you did in task 5 for a three-input AND gate. Show your connections in the circuit below.
Note: you can obtain a three-input AND gate by drawing a regular two-input AND gate and then changing its *Number of Inputs* property as shown next.

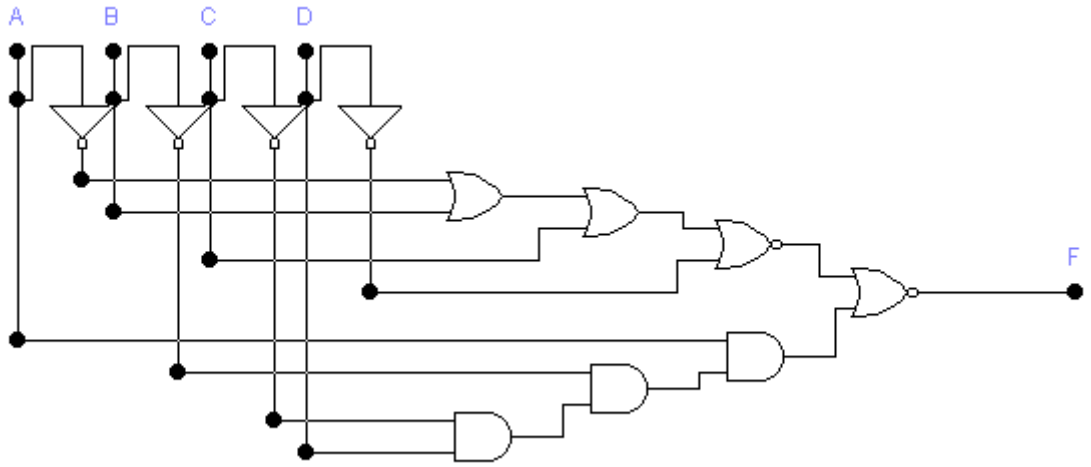




A	B	C	D
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Task 8: Finding the truth table of a given circuit using the logic converter

Find the truth table of the following circuit:



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

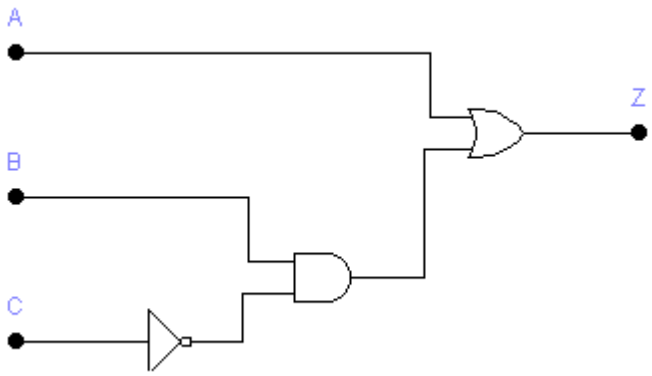
Lab Experiment # 03

Digital logic circuits analysis and converting Boolean expressions to digital circuits

Objectives

- To learn how to directly convert a Boolean expression to circuit.
- To learn how to analyze a given digital logic circuit by finding the Boolean expression that represents the circuit
- To learn how to analyze a given digital logic circuit by finding the truth table that represents the circuit.

Example:
Z = A + B . C'
The above function is implemented in the following digital logic Circuit



Now after drawing the circuit above using EWB we find that its truth table is as shown below (notice that logic 1 means connect the input to the Vcc line, and logic 0 means connecting the input to the ground)

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

Lab Tasks

Task 1: Converting Boolean expressions into circuits

Convert the following Boolean expression to a circuit, draw the circuit on EWB and simulate it to fill-in its truth table shown below.
X = Y + Z. Y'

Draw the circuit in the space below

Now, fill-in the truth table of the circuit you drawn

Y	Z	X
0	0	
0	1	
1	0	
1	1	

Task 2: Converting Boolean expressions into circuits

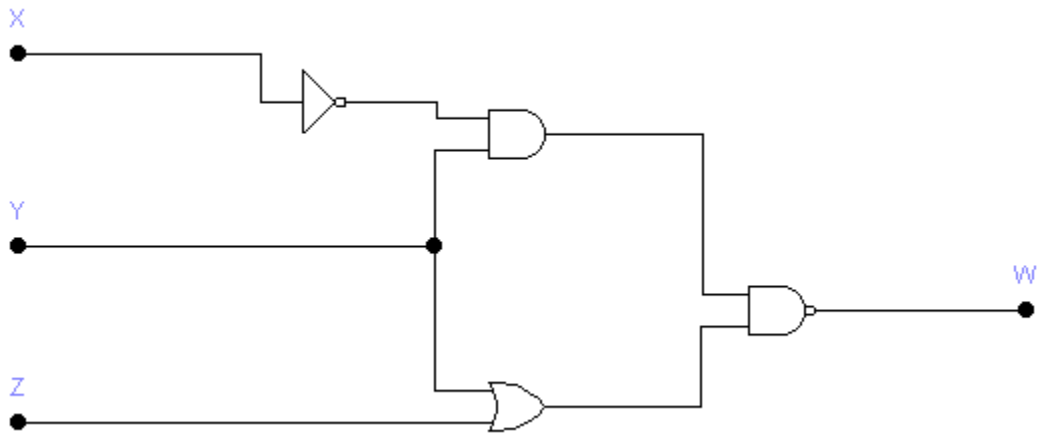
Convert the following Boolean expression to a circuit, draw the circuit on EWB and simulate it to fill-in its truth table shown below.
D = (A. B) + (C'. A)

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

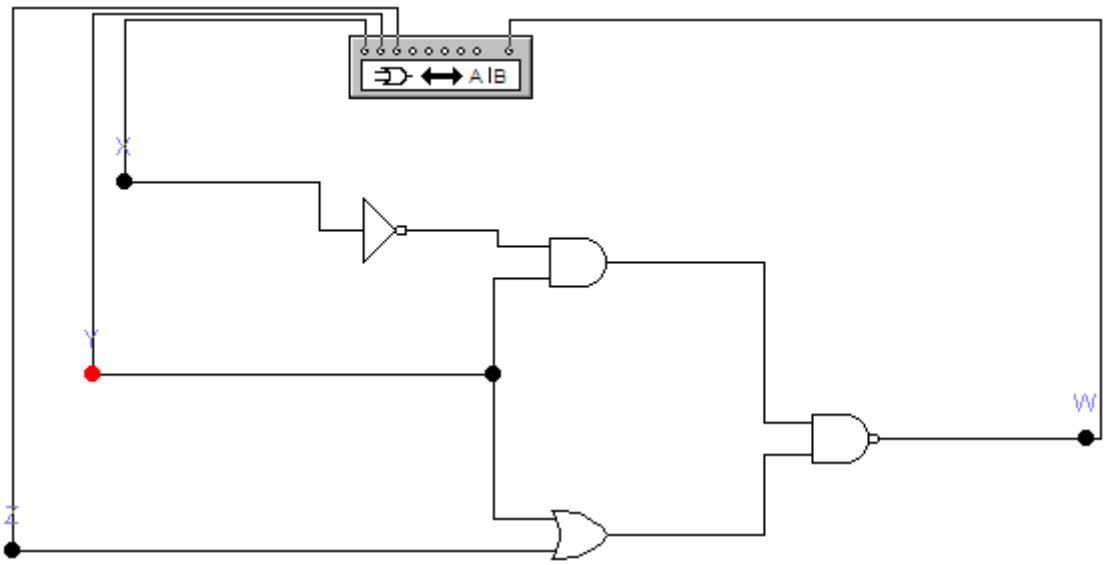
Task 3: Digital logic circuit analysis – Finding the Boolean expression of a given circuit

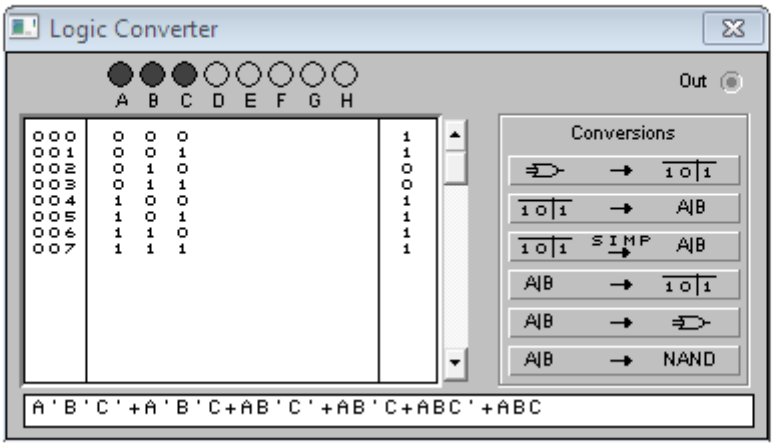
Find the Boolean expression of the following circuit, draw the circuit on EWB and simulate it to fill-in its truth table shown below.

W =



Note: the logic converter tool from EWB to fill-in the following table. For that, you need to connect the A, B and C inputs of the logic converter to X, Y and Z lines, respectively. Further, you need to connect the 'out' line of the logic converter to W. As shown in the following diagram



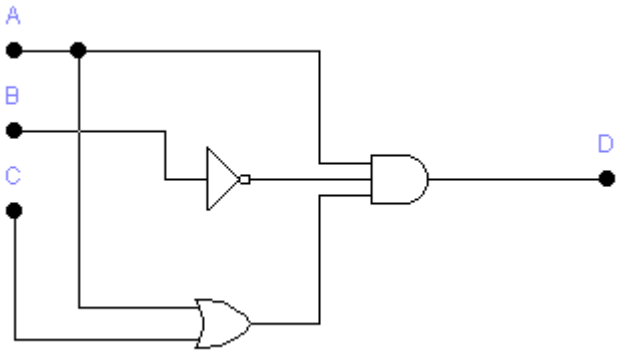


X	Y	Z	W
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Task 4: Digital logic circuit analysis – Finding the Boolean expression of a given circuit

Find the Boolean expression of the following circuit,

D =



Draw the circuit on EWB and simulate it to fill-in its truth table shown below (*use logic converter please*).

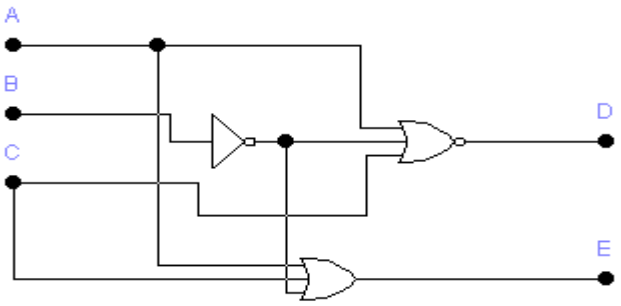
A	B	C	D
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Task 5: Logic circuits with multiple outputs

Find the Boolean expression of the outputs of the following circuit,

D =

E =



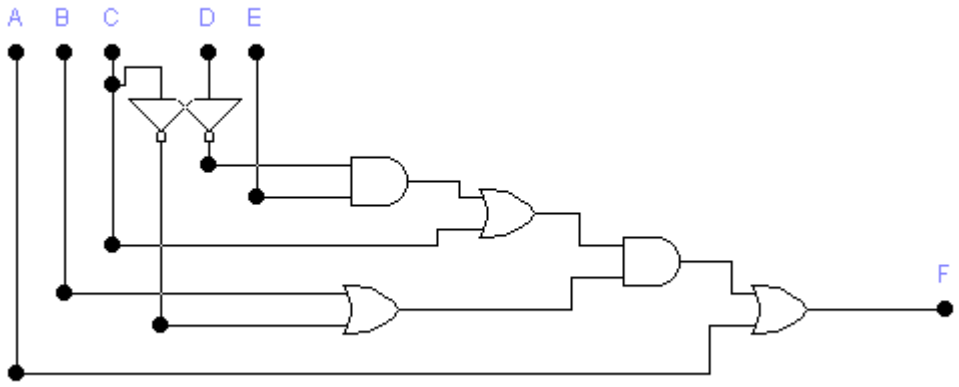
Draw the circuit on EWB and simulate it to fill-in its truth table shown below (*use logic converter please*).

Note: You need to use the logic converter two times, once for the output D, and another time for the second output E.

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

Task 6*: Finding the Boolean expression of a given circuit using the logic converter

Draw the following circuit on EWB and then find its Boolean expression using the logic converter.



Task 7*: Converting Boolean expressions to circuits using the logic converter

Use the logic converter to realize the following circuit using suitable logic gates:
AB'C (BD + CDE) + AC'

Lab Experiment # 04

Boolean algebra and Simplification of Boolean expressions -I

Objectives

Object

1- To study DE Morgan's theory and implemented it.

2- Learn how to simplify Boolean logic equations using DE Morgan's theory.

DE Morgan's Theory – Background

Augustus De Morgan (27 June 1806 – 18 March 1871) was a British mathematician and logician. He formulated De Morgan's laws.

In simple words, DE Morgan's Theory is used to convert AND/NAND gates to OR/NOR ones, and presented OR/NOR gates by AND/NAND gates by these 2-laws:

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

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

Basics of Boolean algebra

Boolean Postulates

P1: X = 0 or X = 1
P2: 0. 0 = 0

P3: 1 + 1 = 1

P4: 0 + 0 = 0

P5: 1. 1 = 1

P6: 1. 0 = 0. 1 = 0

P7: 1 + 0 = 0 + 1 = 1

Boolean Laws

T1: Commutative Law

(a) $A + B = B + A$
(b) $A\,B = B\,A$

T2: Associate Law

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

(b) $(A\,B)\,C = A\,(B\,C)$

T3: Distributive Law

(a) $A\,(B + C) = A\,B + A\,C$

(b) $A + (B\,C) = (A + B)\,(A + C)$

T4: Identity Law

(a) $A + A = A$
(b) $A\,A = A$

T5:

(a) $A\bar{B} + AB = A$

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

T6: Redundancy Law

(a) $A + A\,B = A$
(b) $A\,(A + B) = A$

T7:

(a) $0 + A = A$
(b) $0\,A = 0$

T8:

(a) $1 + A = 1$

(b) $1\,A = A$

T9:

(a) $\bar{A} + A = 1$

(b) $\bar{A}\,A = 0$

T10:

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

(b) $A\,(\overline{A} + B) = A\,B$

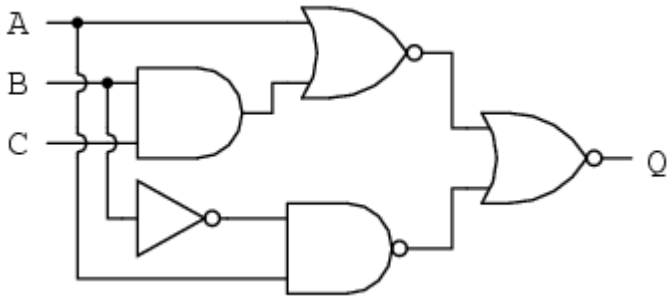
T11: De Morgan's Theorem

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

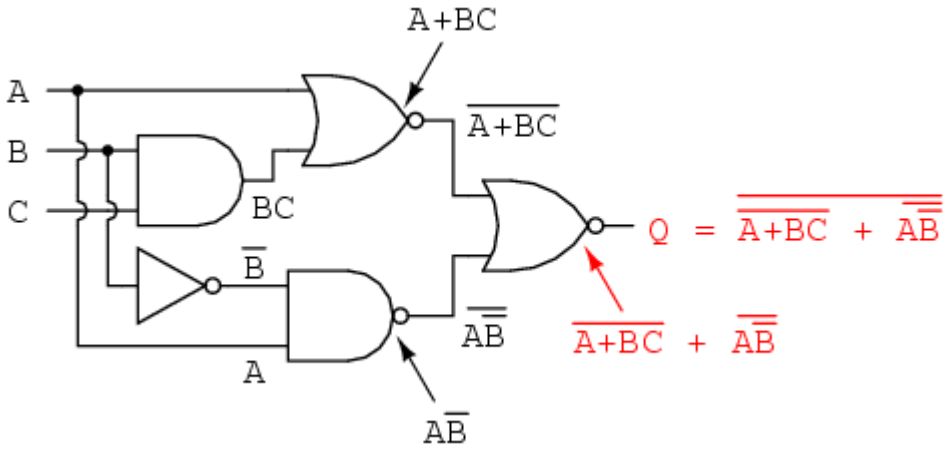
(b) $\overline{(A\,B)} = \ \overline{A} + \overline{B}$

Simplifying Boolean logic functions

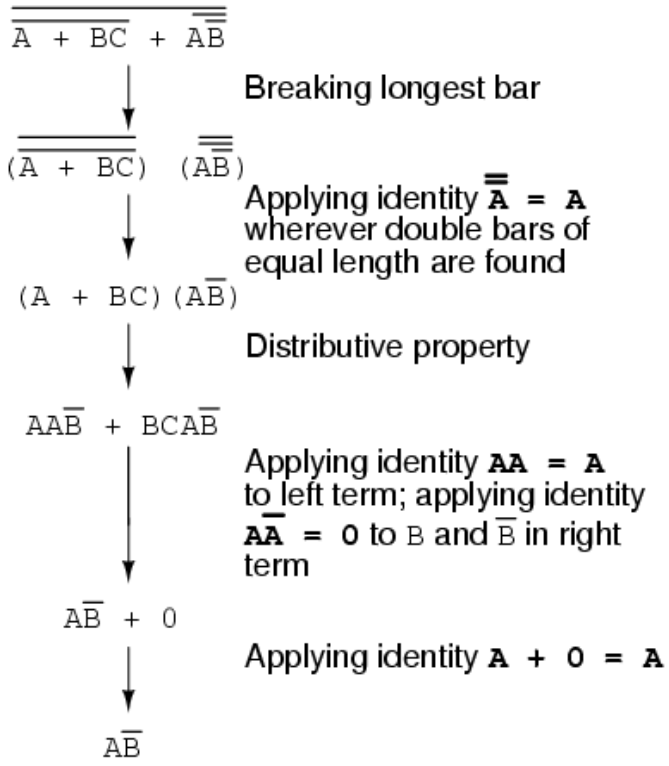
Given the following circuit



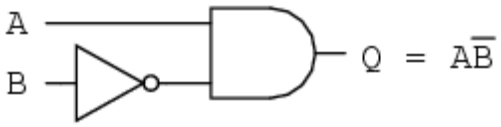
The Boolean expression that represents the above circuit is as follows



We can simplify the above Boolean expression as follows



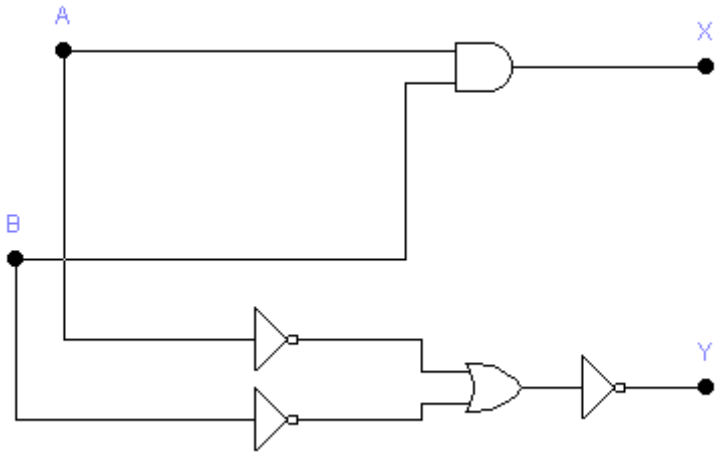
This means that the above circuit can be replaced by the following one



Lab Tasks

Task 1: Circuit analysis

Find the Boolean expression that represents the outputs x and y shown in the following circuit.



According to the circuit above find the equation of X and Y, then fill the truth table.

X =

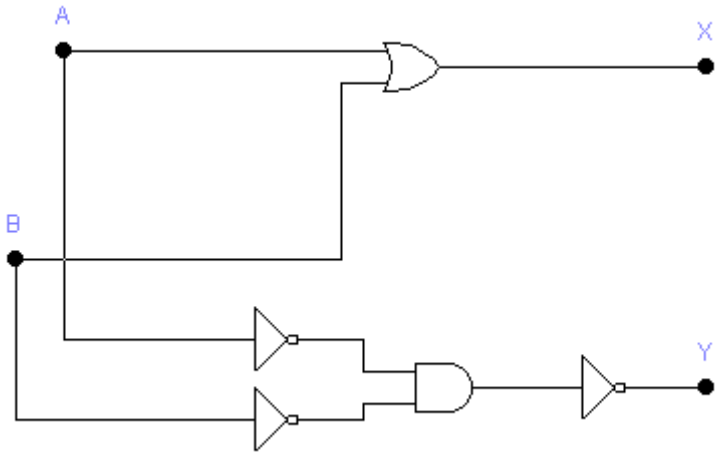
Y =

A	B	X	Y
0	0		
0	1		
1	0		
1	1		

What do you notice?

Task 2: Circuit analysis

Find the Boolean expression that represents the outputs x and y shown in the following circuit.



According to the circuit above find the equation for X and Y, then fill the truth table.

X =

Y =

A	B	X	Y
0	0		
0	1		
1	0		
1	1		

What do you notice?

Task 3: Simplifying Boolean functions

Simplify the following Boolean expression

F (A, B) = (A. B) + A' (A+B)

Draw the simplified and the original Boolean expression using EWB and make sure that they are booth equivalent by filling-in the following truth table.

A	B	F (A, B) (original)	Y (Simplified)
0	0		
0	1		
1	0		
1	1		

Task 4: Simplifying Boolean functions

Simplify the following Boolean expression

F (A, B, C) = (A+C') + C (C.A' + (B.A) +C

Draw the simplified Boolean expression using EWB. Find out the truth table of the circuit.

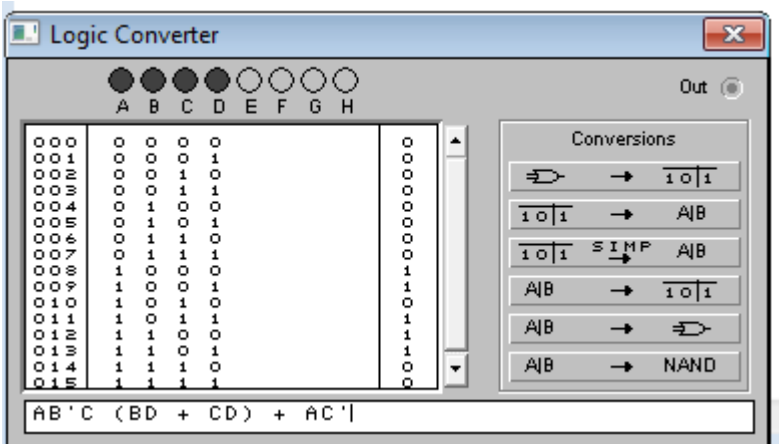
Task 5: Simplifying Boolean functions in EWB using the logic converter

Simplify the following Boolean expression in EWB using the logic converter

F (A, B, C) = AB'C (BD + CD) + AC'

To do so, you need to enter the expression as shown below, and then click on the following button  to extract the truth table of the expression. Finally, click on the following button  that will generate the simplified form of the equation.

To draw the circuit after simplification, you need to click on the following button  , this will realize the simplified expression using basic gates.



Task 6: Simplifying Boolean functions in EWB using the logic converter

Simplify the following Boolean expression in EWB using the logic converter

F (A, B, C) = AB'C'+ A'B'C'+ A'BC'+ A'B'C

Lab Experiment # 05

DE Morgan's Theory and the Universal Gates

Objectives

1- Practically show the correctness of DE Morgan's Theory.

2- Show how to represent any gate using NAND gates only or NOR gates only.

3- Universal gates - NAND and NOR.

4- How to implement NOT, AND, and OR gate using NAND gates only.

5- How to implement NOT, AND, and OR gate using NOR gates only.

6- Equivalent gates.

7- Two-level digital circuit implementations using universal gates only.

8- Two-level digital circuit implementations using other gates.

Background

The NAND gate represents the complement of the AND operation. Its name is an abbreviation of NOT AND. The graphic symbol for the NAND gate consists of an AND symbol with a bubble on the output, denoting that a complement operation is performed on the output of the AND gate.

The NOR gate represents the complement of the OR operation. Its name is an abbreviation of NOT OR. The graphic symbol for the NOR gate consists of an OR symbol with a bubble on the output, denoting that a complement operation is performed on the output of the OR gate.

A universal gate is a gate which can implement any Boolean function without need to use any other gate type. The NAND and NOR gates are universal gates.

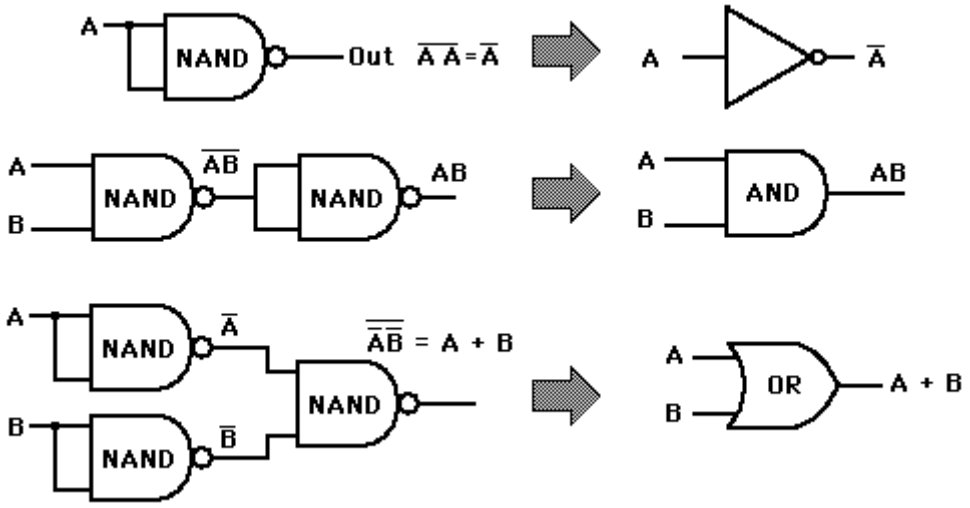
In practice, this is advantageous since NAND and NOR gates are economical and easier to fabricate and are the basic gates used in all IC digital logic families.

In fact, an AND gate is typically implemented as a NAND gate followed by an inverter not the other way around!! Likewise, an OR gate is typically implemented as a NOR gate followed by an inverter not the other way around!!

Implement any gate with NAND gates only

To build an inverter (**NOT** gate) using a NAND gate: All NAND input pins connect to the input signal A gives an output A'.
An **AND** gate can be replaced by NAND gates as shown in the figure (The AND is replaced by a NAND gate with its output complemented by a NAND gate inverter).
An **OR** gate can be replaced by NAND gates as shown in the figure (The OR gate is replaced by a NAND gate with all its inputs complemented by NAND gate inverters).

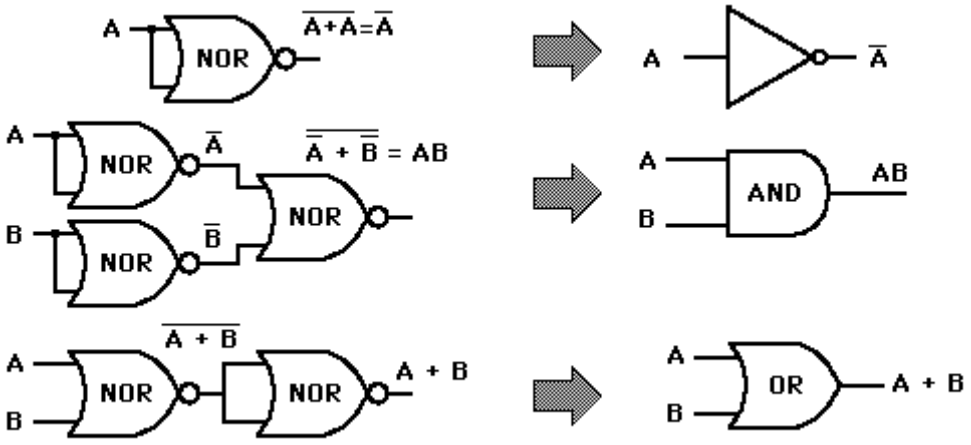
The following figure shows all cases presented above



Implement any gate with NOR gates only

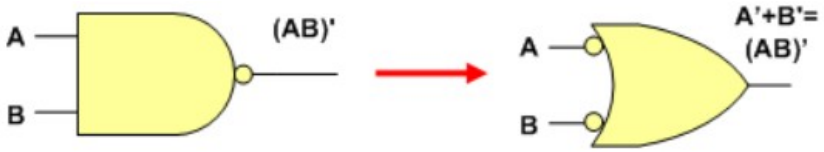
To build an inverter (**NOT** gate) using a NOR gate: All NOR input pins connect to the input signal A gives an output A'.
An OR gate can be replaced by NOR gates as shown in the figure (The OR is replaced by a NOR gate with its output complemented by a NOR gate inverter)
An AND gate can be replaced by NOR gates as shown in the figure (The AND gate is replaced by a NOR gate with all its inputs complemented by NOR gate inverters)

The following figure shows all cases presented above

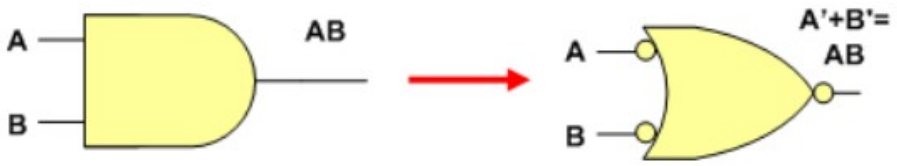


Equivalent Gates

A NAND gate is equivalent to an inverted-input OR gate.



An AND gate is equivalent to an inverted-input NOR gate.



A NOR gate is equivalent to an inverted-input AND gate.

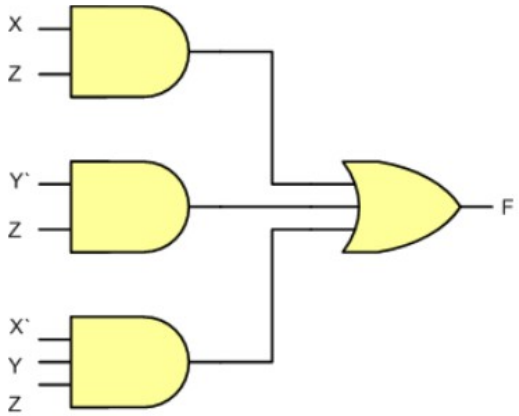
An OR gate is equivalent to an inverted-input NAND gate.

Building Circuits using NAND and NOR gates only

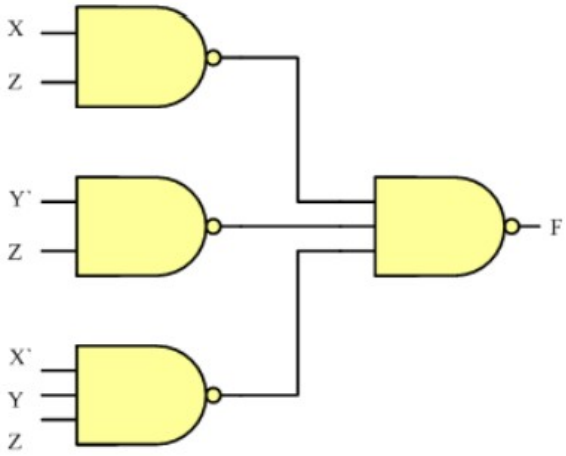
Example: Building Circuits using NAND gates only

Implement the following function using AND, OR gates

F = XZ + Y'Z + X'YZ



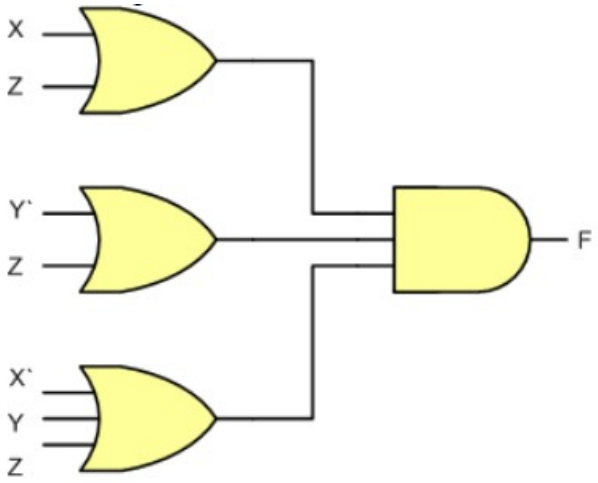
Re-implement the same function above using NAND gates only



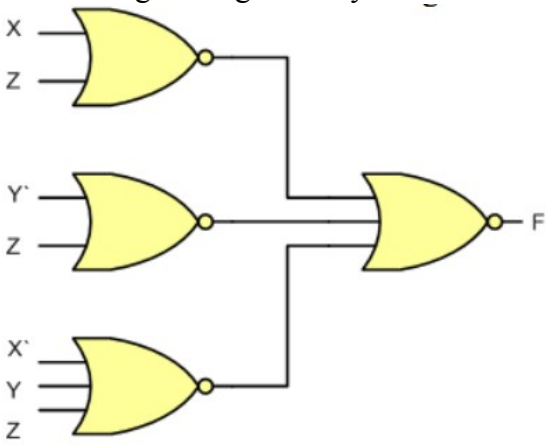
Example: Building Circuits using NOR gates only

Implement the following function using AND, OR gates

F = (X+Z) (Y'+Z) (X'+Y+Z)



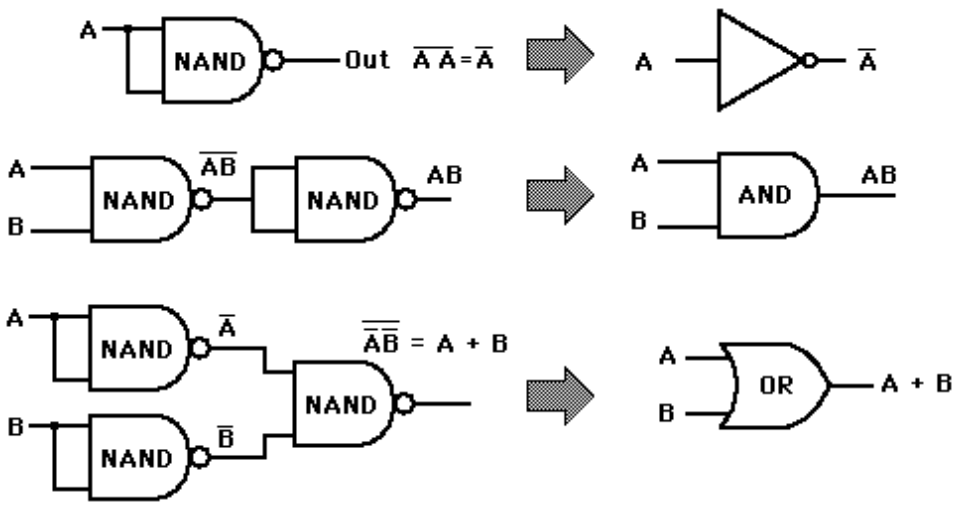
Re-implement the same function above using NOR gates only



Lab Tasks

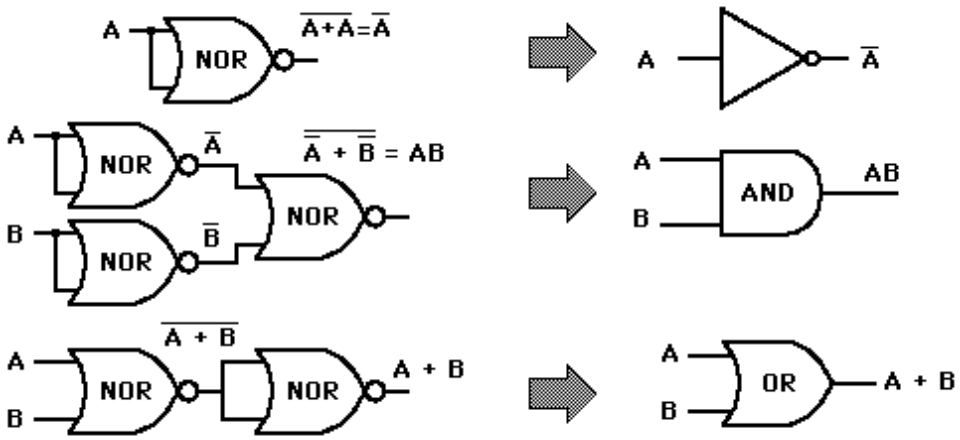
Task 1: The Universal NAND gate

Use EWB to show that the following gates are equivalent



Task 2: The Universal NOR gate

Use EWB to show that the following gates are equivalent



Task 3: Implementing circuits using NAND gates only

Implement the following function using AND, OR gates

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

Re-implement the same function above using NAND gates only

Show, using EWB, that both circuits are equivalent

Task 3: Implementing circuits using NOR gates only

Implement the following function using AND, OR gates

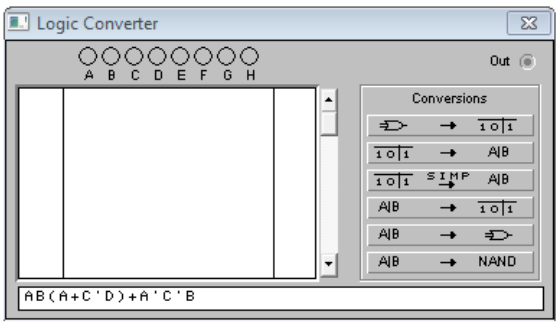
F=(A+B).C'+A'D

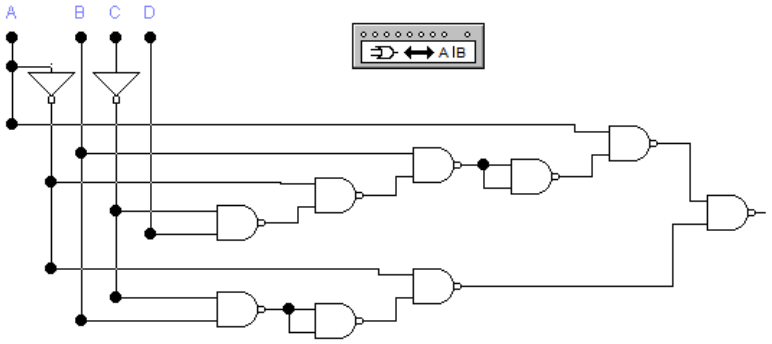
Re-implement the same function above using NOR gates only

Show, using EWB, that both circuits are equivalent

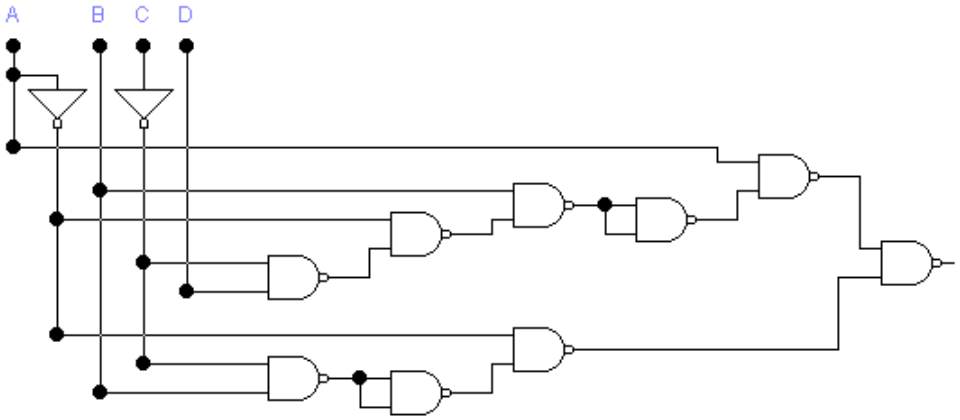
Task 4: Implementing circuits using NAND gates only

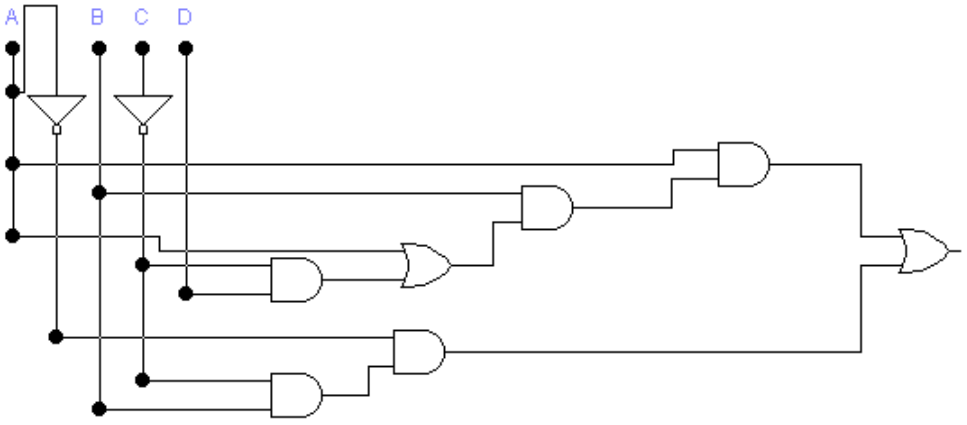
Implement the following function using NAND gates (Use the logic converter in EWB)
F= AB(A+C'D) +A'C'B
To do so, you need to write the Boolean algebra expression to implement and the press the A|B → NAND button in the logic converter as shown next





The solution should look like as follows





Task 5: Implementing circuits using NAND gates only

Implement the following function using NAND gates (Use the logic converter in EWB)
F= CA’+B(A’.C'+D) +A'CB’

Lab Experiment # 06

Simplification of Boolean expressions - II

Objectives

1- Study K-maps with 2, 3 and 4 inputs.

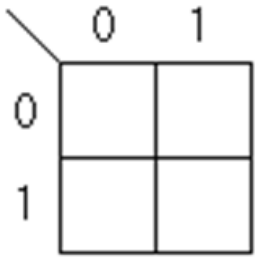
2- Simplify Boolean logic equations by using K-maps.

Lab Tasks

Task 1: Simplifying two-input Boolean functions

Simplify the following Boolean expression using a k-map of size 2x2.

F (A, B) = (A. B) + A' (A+B)



Draw the simplified and the original Boolean expression using EWB and make sure that they are booth equivalent by filling-in the following truth table.

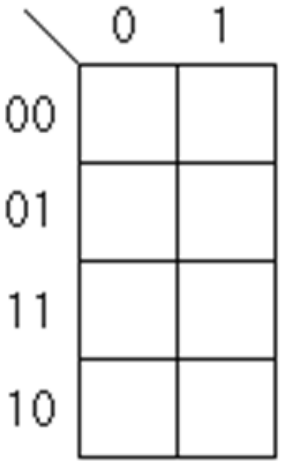
A	B	F (A, B) (original)	Y (Simplified)

0	0		
0	1		
1	0		
1	1		

Task 2: Simplifying three-input Boolean functions

Simplify the following Boolean expression F

(A, B, C) = (A+C') + C (C.A' + (B.A) +C)



Draw the simplified Boolean expression using EWB. Find out the truth table of the circuit.

	A	B	C	F
1	0	0	0	
2	0	0	1	
3	0	1	0	
4	0	1	1	
5	1	0	0	
6	1	0	1	
7	1	1	0	
8	1	1	1	

Task 3: Simplifying four-input Boolean functions

Simplify the following logic function using k-maps

F (A, B, C, D) = Σ (6, 8, 9, 10, 11, 12, 13, 14)

Then draw the logic circuit that represents this function.

Fill the truth table of the circuit above.

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

Lab Experiment # 07

The Story of Minterms and Maxterms

Objectives

Learn how implement logic functions using the standard forms: Sum of Products and Product of Sums.

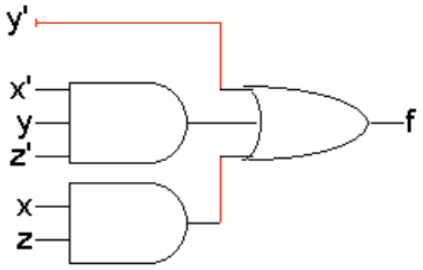
Background

We can write expressions in many ways, but some ways are more useful than others
A sum of products (SOP) expression contains: Only OR (sum) operations at the "outermost" level and each term that is summed must be a product of literals
The advantage is that any sum of products expression can be implemented using a three-level circuit

- literals and their complements at the first level
- AND gates at the second level
- a single OR gate at the third level

Example:

f(x,y,z) = y' + x'yz' + xz



Notice that the NOT gates are implicit and that literals are reused.

A minterm is a special product of literals, in which each input variable appears exactly once.
A function with n variables has 2^n minterms (since each variable can appear complemented or not)

Example:

A three-variable function, such as f(x,y,z), has $2^3 = 8$ minterms:
Each minterm is true for exactly one combination of inputs:
Those minterms are: x'y'z' x'y'z x'yz' x'yz xy'z' xy'z xyz' xyz
A Minterm is true when:

Minterm	When the minterm is True			Minterm ID
x'y'z'	x=0,	y=0,	z=0	m0
x'y'z	x=0,	y=0,	z=1	m1
x'yz'	x=0,	y=1,	z=0	m2
x'yz	x=0,	y=1,	z=1	m3
xy'z'	x=1,	y=0,	z=0	m4
xy'z	x=1,	y=0,	z=1	m5
xyz'	x=1,	y=1,	z=0	m6
xyz	x=1,	y=1,	z=1	m7

Sum of Minterms (or Sum of Products)

Every function can be written as a sum of minterms, which is a special kind of sum of products form
The sum of minterms form for any function is unique

If you have a truth table for a function, you can write a sum of minterms expression just by picking out the rows of the table where the function output is 1.

Example

f = x'y'z' + x'y'z + x'yz' + x'yz + xyz'
= m0 + m1 + m2 + m3 + m6
= Σm(0,1,2,3,6)

The dual idea: products of sums

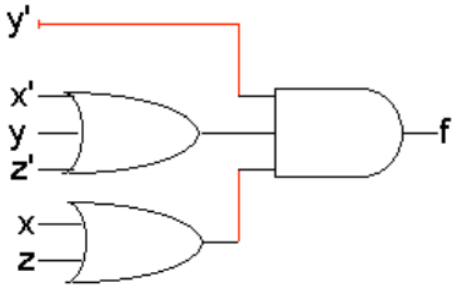
A product of sums (POS) expression contains: Only AND (product) operations at the "outermost" level,
Each term must be a sum of literals.
Product of sums expressions can be implemented with three-level circuits

- literals and their complements at the first level
- OR gates at the first level
- a single AND gate at the second level

• Compare this with sums of products

Example

f(x, y, z) = y' . (x'+y+z') . (x+z)



A maxterm is a sum of literals, in which each input variable appears exactly once.
A function with n variables has 2^n maxterms

Example

A three-variable function f(x,y,z) has 8 maxterms
Each maxterm is false for exactly one combination of inputs:
Those materms are: x'+y'+z' x'+y'+z x'+ y+z' x'+ y+z x+y'+z' x+y'+z x+y+z' x+y+z
Maxterm Is false when:

Maxterm	When the maxterm is false	Maxterm ID
x + y + z	x=0, y=0, z=0	M0
x + y + z'	x=0, y=0, z=1	M1
x + y' + z	x=0, y=1, z=0	M2
x + y' + z'	x=0, y=1, z=1	M3
x' + y + z	x=1, y=0, z=0	M4
x' + y + z'	x=1, y=0, z=1	M5
x' + y' + z	x=1, y=1, z=0	M6
x' + y' + z'	x=1, y=1, z=1	M7

Every function can be written as a unique product of maxterms
If you have a truth table for a function, you can write a product of maxterms expression by picking out the

rows of the table where the function output is 0. (Be careful if you're writing the actual literals!)
f = (x' + y + z).(x' + y + z').(x' + y' + z')
= M4. M5.M7 = ΠM(4,5,7)

f' = (x + y + z).(x + y + z').(x + y' + z).(x + y' + z').(x' + y' + z)
= M0. M1. M2. M3. M6 = ΠM(0,1,2,3,6)

Minterms and maxterms are related

Any minterm m_i is the complement of the corresponding maxterm M_i
For example, m4' = M4 because (xy'z')' = x' + y + z

Minterm	**Shorthand**	**Maxterm**	**Shorthand**
x'y'z'	m0	x + y + z	M0
x'y'z	m1	x + y + z'	M1
x'yz'	m2	x + y' + z	M2
x'yz	m3	x + y' + z'	M3
xy'z'	m4	x' + y + z	M4
xy'z	m5	x' + y + z'	M5
xyz'	m6	x' + y' + z	M6
xyz	m7	x' + y' + z'	M7

Converting between standard forms

We can convert a sum of minterms to a product of maxterms
• In general, just replace the minterms with maxterms, using maxterm numbers that don't appear in the sum of minterms:
• The same thing works for converting from a product of maxterms to a sum of minterms

Example

From before
f = Σm(0,1,2,3,6)
and f' = Σm(4,5,7)
= m4 + m5 + m7
complementing (f')' = (m4 + m5 + m7)'
so f = m4' . m5' . m7' [DeMorgan's law]
= M4 . M5 . M7
= ΠM(4,5,7)

Lab Tasks

Task 1: Three-input Boolean functions

Given the following truth table of a three-input logic circuit

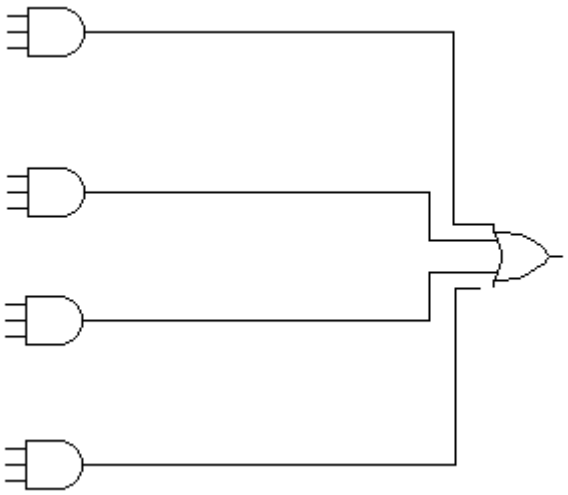
A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Write the above function in the two standard forms

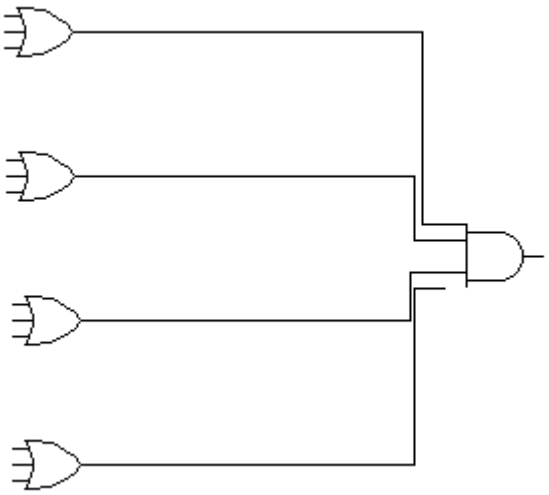
F(A, B, C)= Σ ( )

F(A, B, C) = Π( )

Draw a circuit that implements the above logic function (use minterms only)



Draw a circuit that implements the above logic function (use maxterms only)



Task 2: Three-input Boolean functions

Simplify (using k-maps) the function presented in Task 1 of this lab. Draw the simplified form of the function on EWB. Use the Logic Converter of EWB to generate the truth table of the simplified circuit.

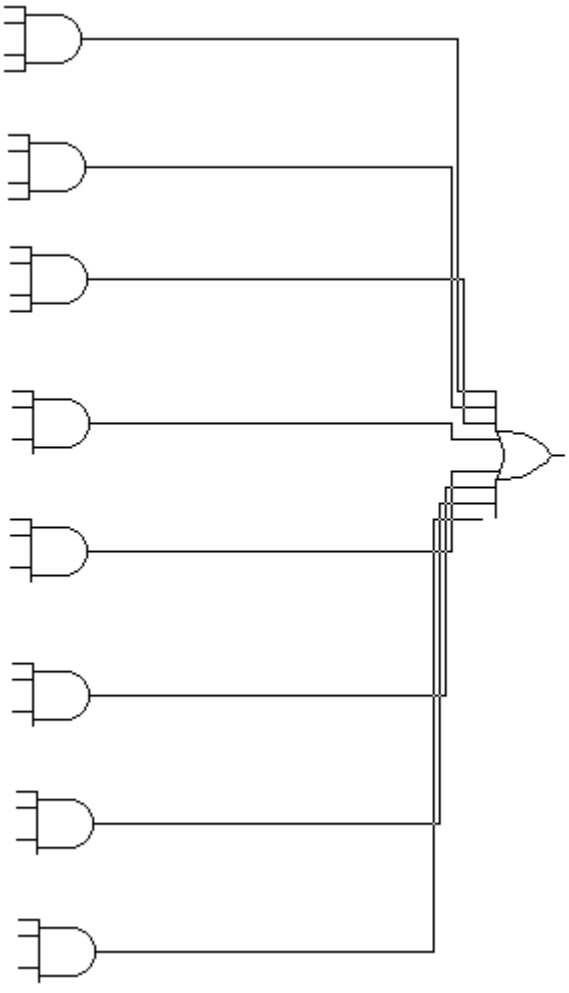
A	B	C	F (simplified)
0	0	0	
0	0	1	

0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Task 3: Four-input Boolean functions

Draw the following logic function using EWB

$F(A, B, C, D) = \Sigma(6, 8, 9, 10, 11, 12, 13, 14)$



Task 4: Four-input Boolean functions

Simplify (using k-maps) the function presented in Task 3 of this lab. Draw the simplified form of the function on EWB. Use the Logic Converter of EWB to generate the truth table of the simplified circuit.

	A	B	C	D	F
0	0	0	0	0	

1	0	0	0	1	
2	0	0	1	0	
3	0	0	1	1	
4	0	1	0	0	
5	0	1	0	1	
6	0	1	1	0	
7	0	1	1	1	
8	1	0	0	0	
9	1	0	0	1	
10	1	0	1	0	
11	1	0	1	1	
12	1	1	0	0	
13	1	1	0	1	
14	1	1	1	0	
15	1	1	1	1	

Task 5: Simplifying 4-variable functions

Simplify and implement (using EWB) the following function
F(a, b, c, d) = (a'+b'+d')(a+b'+c')(a'+b+d')(b+c'+d')

Draw you circuit below

Task 6: Simplifying 4-variable functions: SOP

Draw a NAND logic diagram that implements the **complement** of the following function
$F(A, B, C, D) = \Sigma(0, 1, 2, 3, 4, 8, 9, 12)$
Draw you circuit below

Task 7: Simplifying 4-variable functions: POS

Draw a logic diagram that implements the following function
$F(A, B, C, D) = \Pi(0, 1, 2, 3, 4, 8, 9, 12)$
Draw you circuit below

Lab Experiment # 08

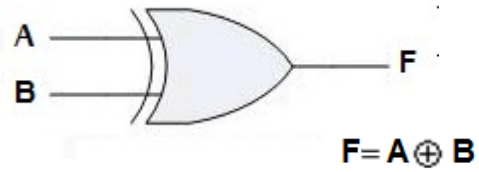
XOR and XNOR gates: Basics and Applications

Objectives

To learn how to build XOR gates from basic gates
To learn how to build a Half Adder and a Full Adder using XOR gates.
To learn how to build a parallel adder, subtracter, incrementer and decrementer using full adders.

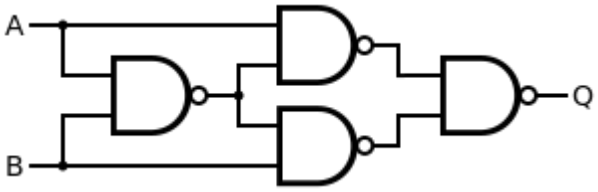
Background

The XOR gate (sometimes EOR gate, or EXOR gate) is a digital logic gate that implements an exclusive or; that is, a true output (1) results if one, and only one, of the inputs to the gate is true (1). If both inputs are false (0) or both are true (1), a false output (0) results. Next is the circuit representation of the XOR gate and its truth table.

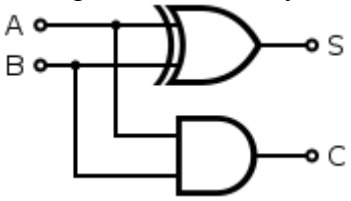


A	B	F1
0	0	0
0	1	1
1	0	1
1	1	0

Next is one way to build an XOR gate using NAND gates only

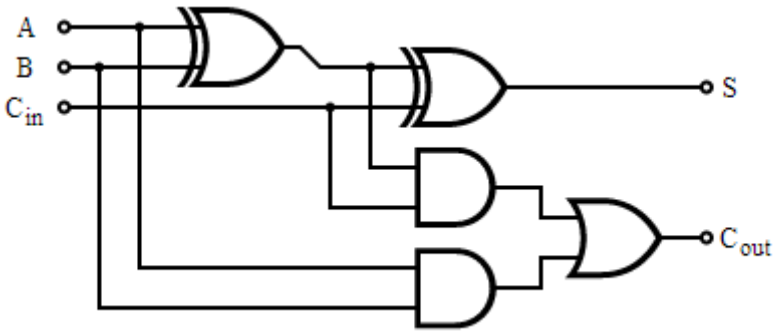


The XOR logic gate can be used as a one-bit adder (or a Half-Adder; HA)that adds any two bits together to output one bit (the sum) and another bit that represents the carry out. As shown below



The XOR logic gate can be used as a one-bit full adder that adds any three bits together to output one bit (the sum) and another bit that represents the carry out. As shown below

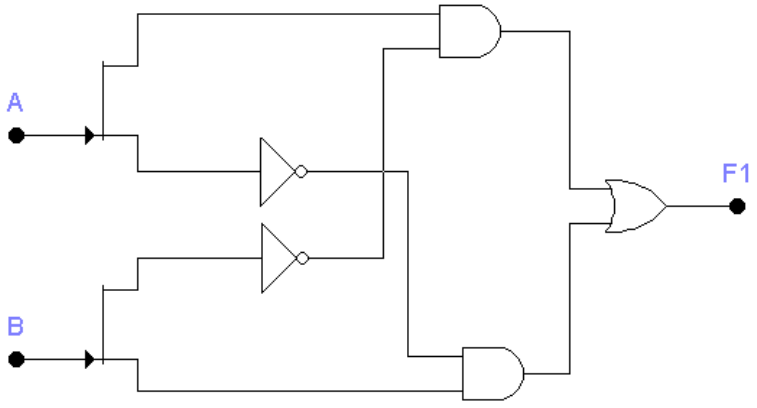
Inputs			Outputs	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	



Lab Tasks

Task 1: XOR built from basic gates

Draw using EWB the following circuits then fill their truth tables:

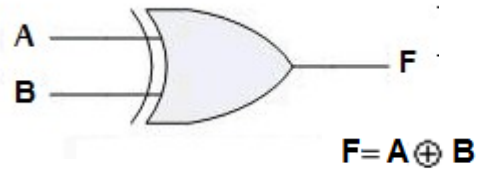


A	B	F1
0	0	
0	1	
1	0	

1	1	

What do you notice?

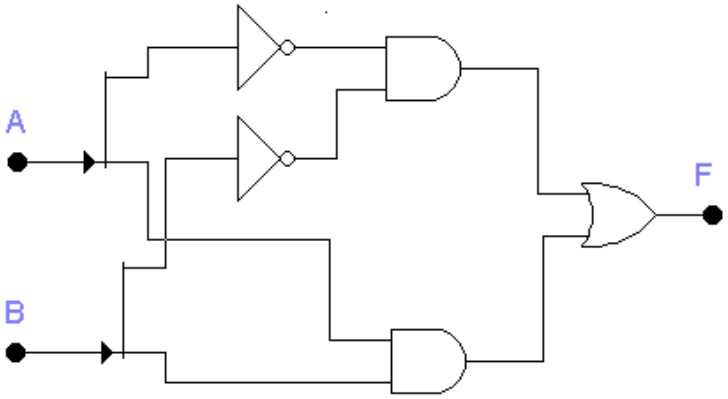
Each one of the above circuits can be replaced with one single logic gate that gives the same truth table, that's the Exclusive OR Gate or XOR.



A	B	F	No. of 1's
0	0	0	Even
0	1	1	Odd
1	0	1	Odd
1	1	0	Even

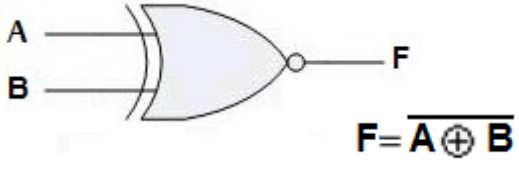
Task 2: XNOR Gate

Draw using EWB the following circuit then fill its truth table:



A	B	F
0	0	
0	1	
1	0	
1	1	

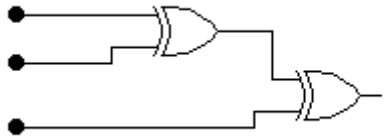
The above circuit can be replaced with one single logic gate that gives the same truth table, that's the Exclusive NOR Gate or XNOR.



A	B	F	No. of 1's
0	0	1	Odd
0	1	0	Even
1	0	0	Even
1	1	1	Odd

Task 3: 3-input XOR Gate

Draw using EWB a three-input XOR gate. Check the circuit using a Logic converter.



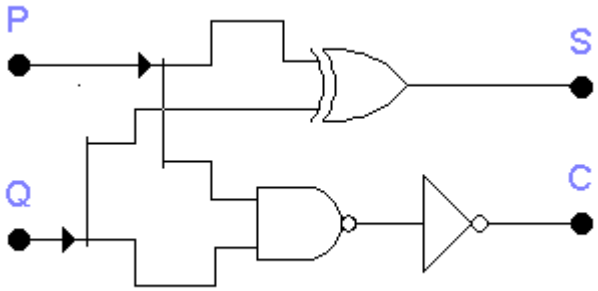
A	B	C	A B C
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Task 4: Half adder circuit

The following diagram represents the Half Adder (HA is a Logic Circuit that performs 1-bit binary addition). Given that P and Q are two 1-bit binary numbers, S is the 1-bit Sum of P and Q, and C is the CARRY bit.

(a) Find out the Boolean functions S and C, and write them in the corresponding blanks.

(b) Draw using EWB the HA circuit then find its truth table by using the logic converter.



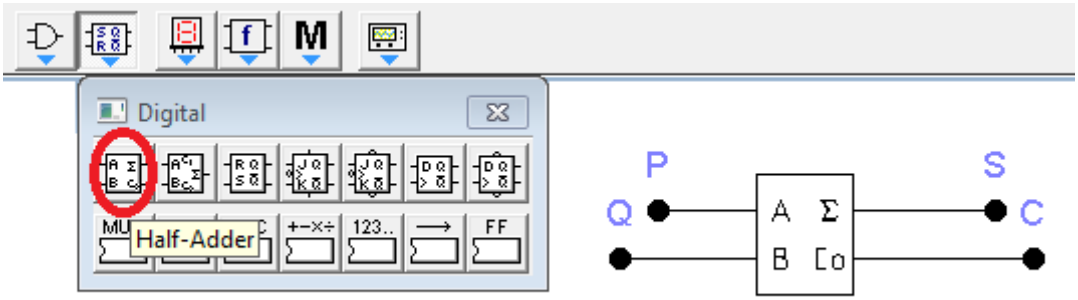
S =

C =

P	Q	S	C
0	0		
0	1		
1	0		
1	1		

Task 5: Implementing HA circuit using EWB

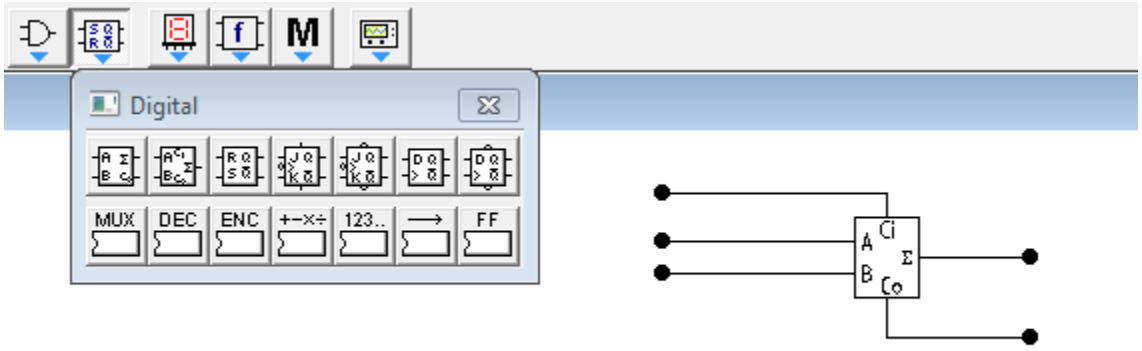
Draw using EWB the HA circuit shown in the figure below then find its truth table by using the logic converter, compare the truth table obtained with the one in Task5, what do you notice?



P	Q	S	C
0	0		
0	1		
1	0		
1	1		

Task 6: Implementing FA circuit using EWB

Draw using EWB a full adder circuit; find out its truth table and Boolean functions.

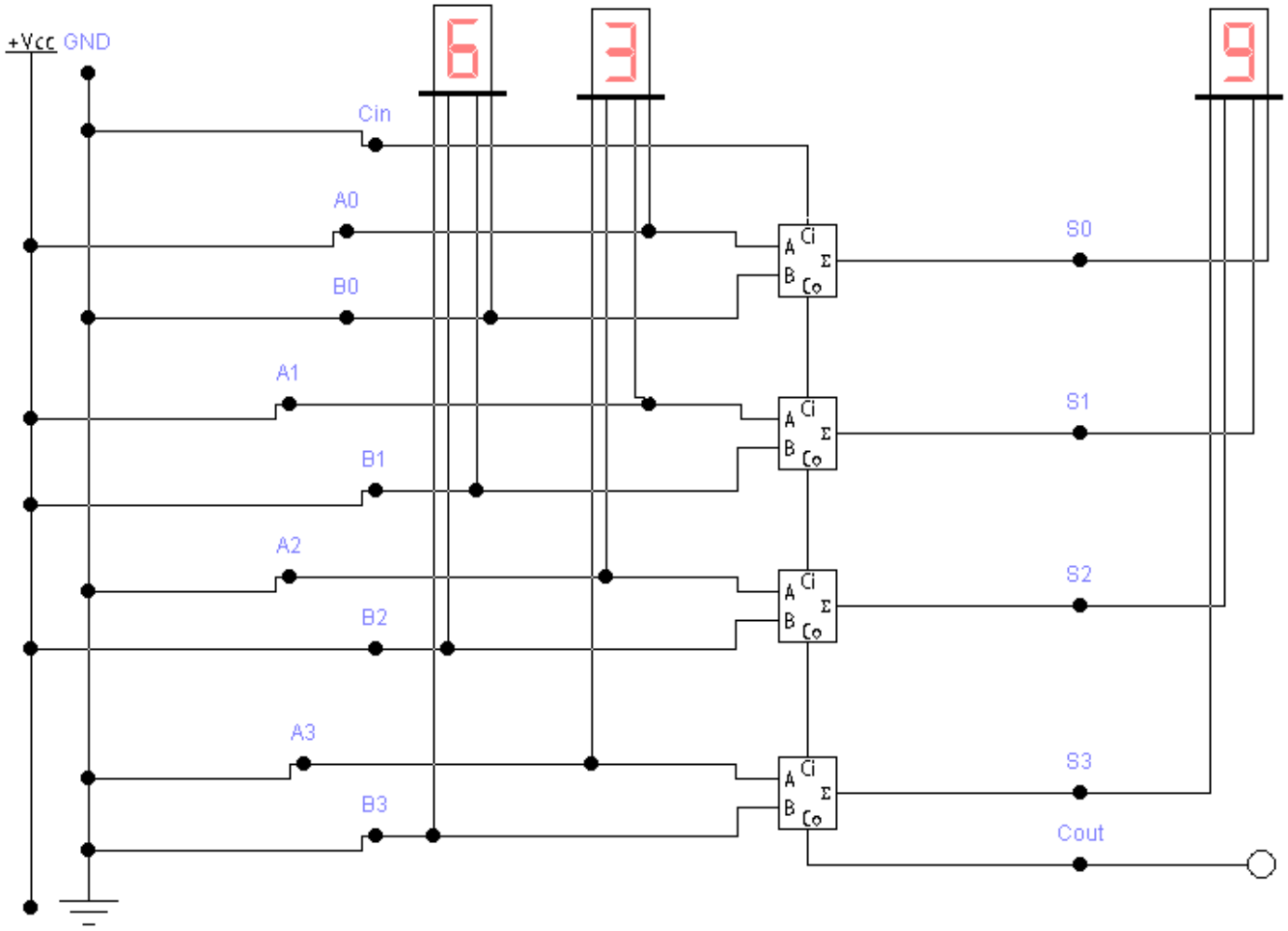


Cin	P	Q	Sum	Cout
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Task 7: Implementing a 4-bit parallel adder using 4 FA's

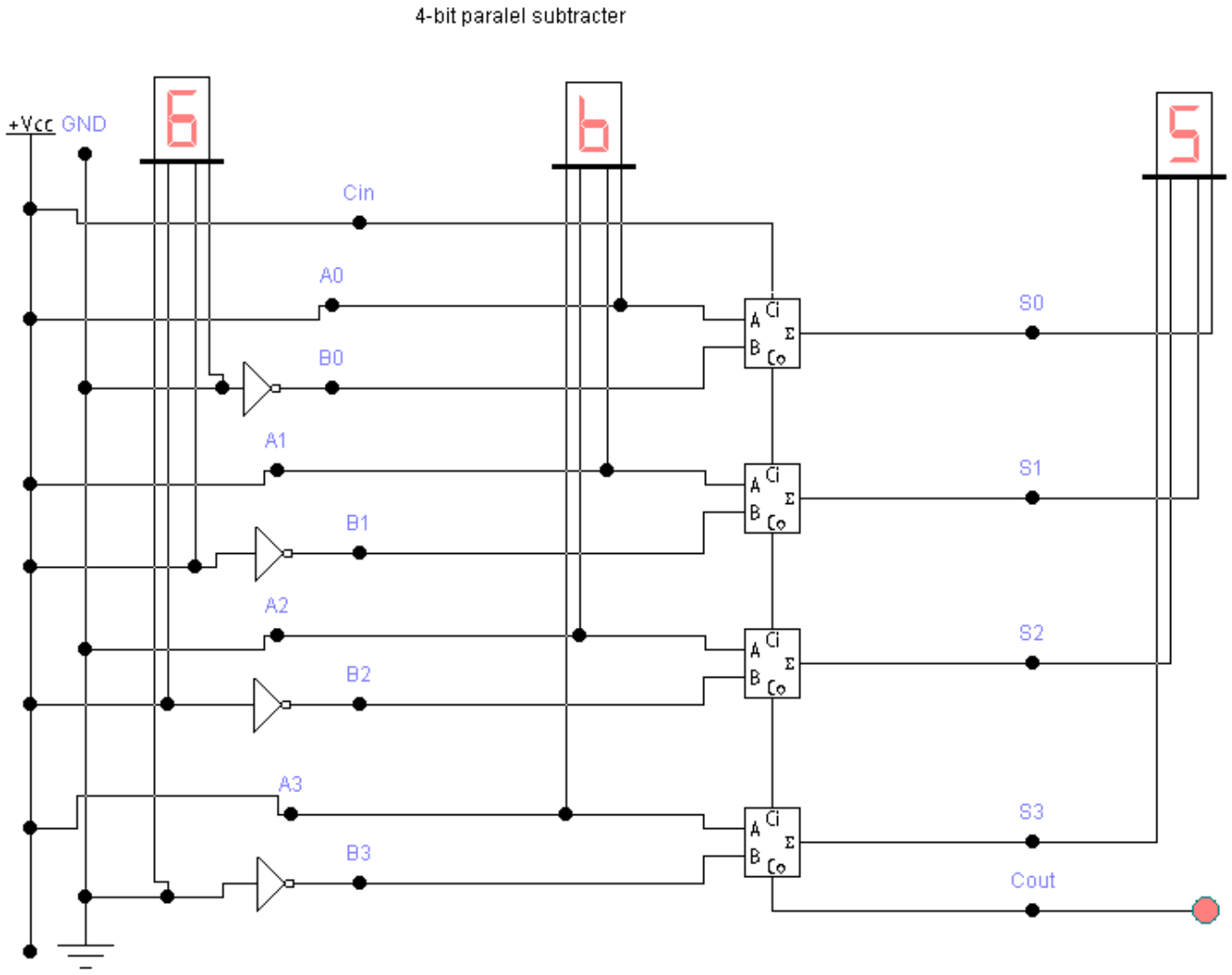
Draw using EWB a 4-bit parallel adder circuit (the circuit below shows (6+3=9))

Note: you can find the decoded 7-segment under the "indicators" toolbar.



Task 8: Implementing a 4-bit parallel subtracter using 4 FA's

Draw using EWB a 4-bit parallel adder circuit (the circuit below shows (b-6=5))



Task 9: Implementing a 4-bit incrementer using 4 FA's

Draw using EWB a 4-bit incrementer circuit.

Task 10: Implementing a 4-bit decrementer using 4 FA's

Draw using EWB a 4-bit decrementer circuit.

Lab Experiment # 09

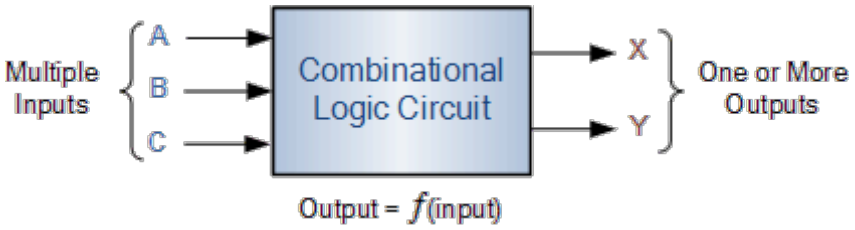
Building logic circuits using Multiplexers

Objectives

- **To learn how to build combinational logic circuits using multiplexers.**

Background

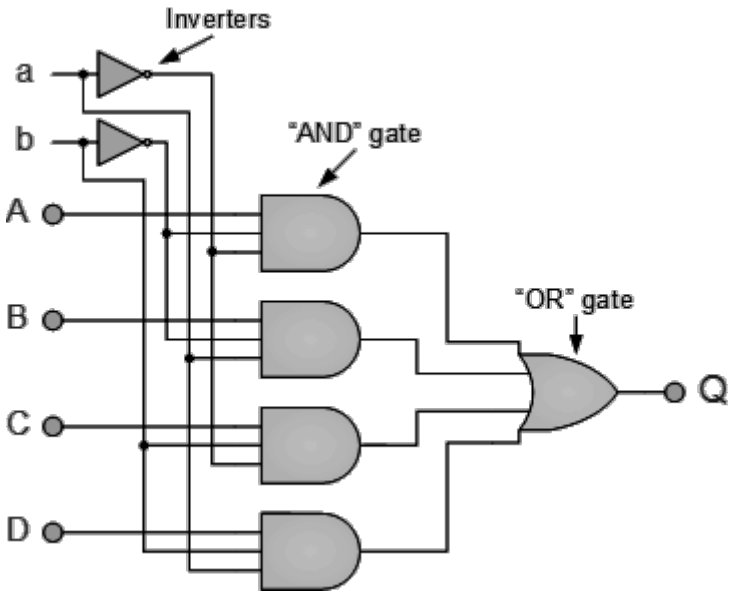
In a **Combinational Logic Circuit**, the output is dependant at all times on the combination of its inputs. Some examples of a combinational circuit include **Multiplexers**, **De-multiplexers**, **Encoders**, **Decoders**, **Full** and **Half Adders** etc.



A Multiplexer is a combination of logic gates resulting into circuits with two or more inputs (data inputs) and one output.

4 Channel Multiplexer using Logic Gates

The following circuit shows a 4x1 mux. Based on the binary value placed at the inputs "a" and "b", what will appear at the circuit output Q is one of the following values: A, B, C, or D.

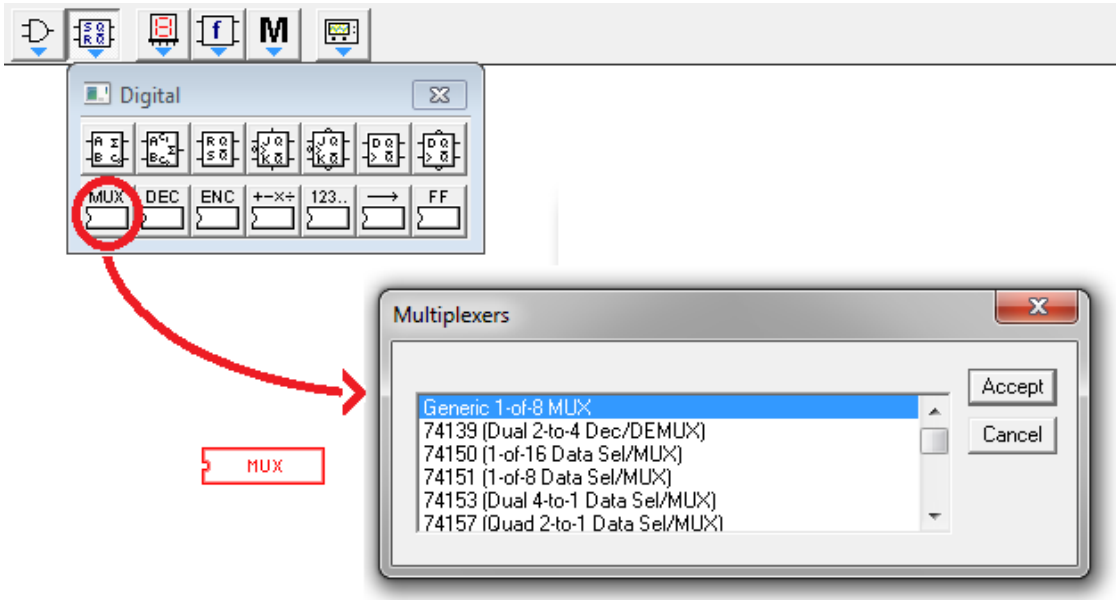


The circuit above is implemented based on the following truth table.

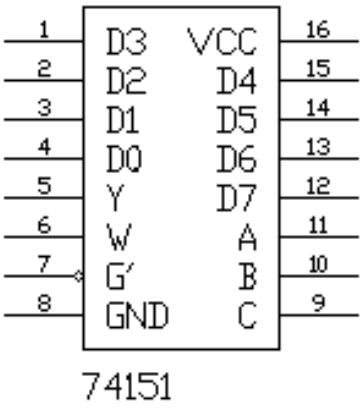
a	b	Q
0	0	A
0	1	B
1	0	C
1	1	D

Drawing Multiplexers in EWB:

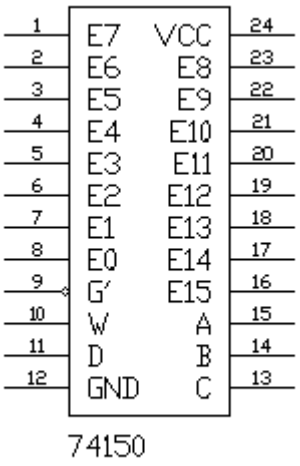
Task: Draw the previous lab examples using EWB, follow the steps below to implement Multiplexers and Decoders.



Then choose 74151 (1-of-8 Data Sel/MUX)from the list:



You may also choose 74150 (1-of-16 Data Sel/Mux) as follows



NOTE: the "A" line in the multiplexer is the least significant bit, while "C" is the most significant bit.

Data selector/multiplexer truth table:

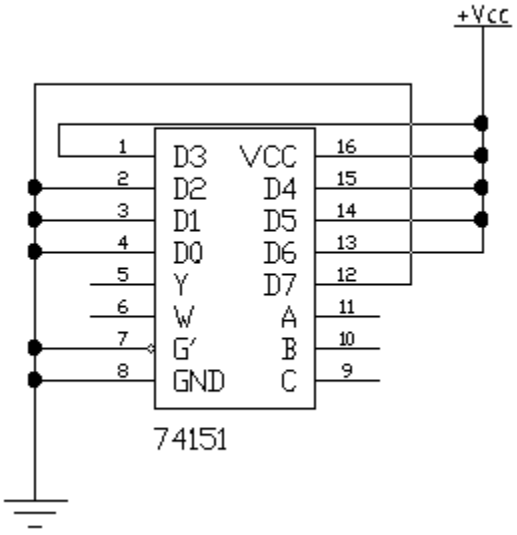
Select			**Strobe**	**Outputs**	
C	**B**	**A**	**G'**	**W**	**Y**
x	x	x	1	1	0
0	0	0	0	D0'	D0
0	0	1	0	D1'	D1
0	1	0	0	D2'	D2
0	1	1	0	D3'	D3
1	0	0	0	D4'	D4
1	0	1	0	D5'	D5
1	1	0	0	D6'	D6
1	1	1	0	D7'	D7

Multiplexers can be used to synthesize logic functions

4-to-1 MUX can realize any 3-variable function, 8-to-1 MUX can realize a 3-variable or 4-variable function, in general 2^n-to-1 MUX can realize an (n +1)-variable and n-variable function.

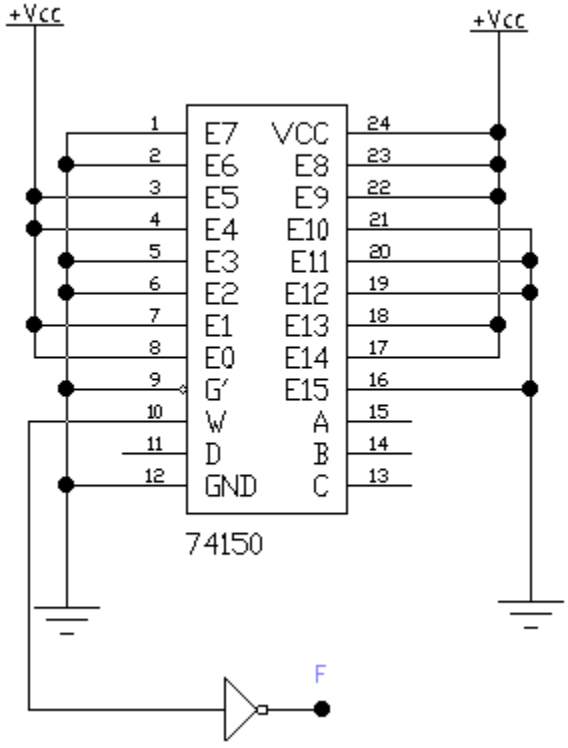
Example: realizing functions using Multiplexers

The function
F=A'BC+AB'+AC'
Can be implemented using an 8-1 mux as follows



Example: realizing functions using Multiplexers

The function
F= A'C'+B'C'+C'D+ABCD'
Can be implemented using an 16-1 mux as follows



Example: realizing a 4-variable function using 8-to-1Multiplexer

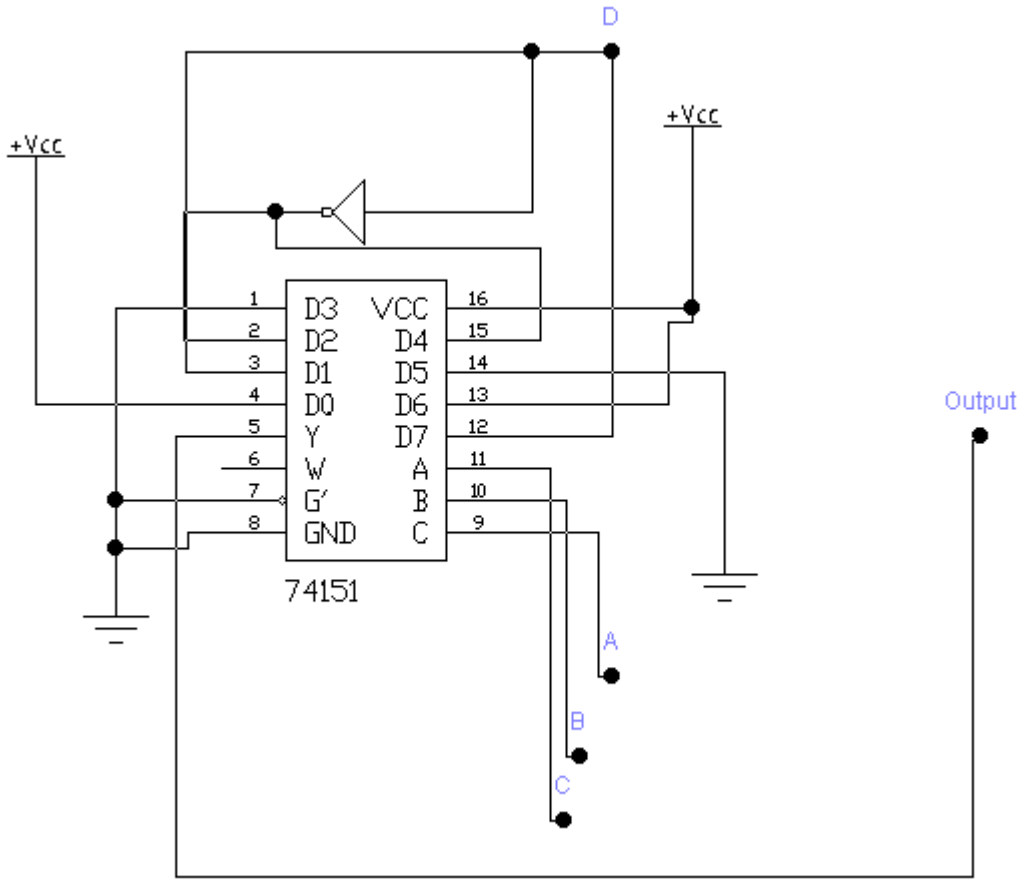
F(A, B, C, D) = A'B'C'D'+A'B'C'D+A'B'CD+A'BC'D'+AB'C'D'+ABC'D'+ABC'D+ABCD

Truth table:

	A	**B**	**C**	**D**	**F**	
0	0	0	0	0	1	F=1
1	0	0	0	1	1	

2	0	0	1	0	0	F=D
3	0	0	1	1	1	
4	0	1	0	0	1	F=D’
5	0	1	0	1	0	
6	0	1	1	0	0	F=0
7	0	1	1	1	0	
8	1	0	0	0	1	F=D’
9	1	0	0	1	0	
10	1	0	1	0	0	F=0
11	1	0	1	1	0	
12	1	1	0	0	1	F=1
13	1	1	0	1	1	
14	1	1	1	0	0	F=D
15	1	1	1	1	1	

To implement this function using EWB, you draw the following circuit:



Lab Tasks

Task 1: Implementing single-output circuits using muxes

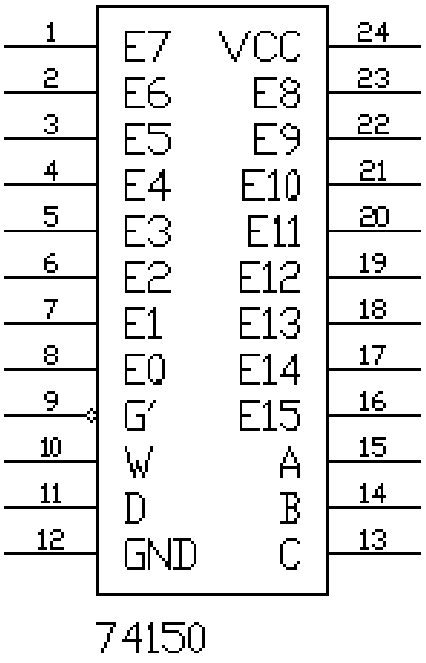
Implement the following function using one 8x1 multiplexer
F(A, B, C, D) = A'B'C'D'+A'B'C'D+A'B'CD+A'BC'D'+AB'C'D'+ABC'D'+ABC'D+ABCD

Note: this example has already been solved above. Just draw the circuit using EWB.

Task 2: Implementing single-output circuits using muxes

Implement the following function using one 16x1 multiplexer

F(A, B, C, D) = A'C'B+AB'C'+B’C'D+ABCD'

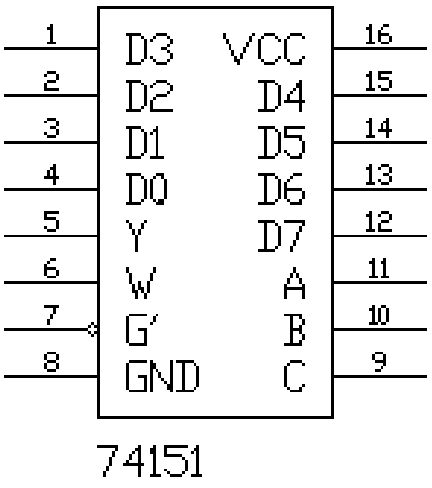


Task 3: Implementing single-output circuits using muxes

Implement the following function using one 8x1 multiplexer

F(A, B, C, D) = A'C'B+AB'C'+B’C'D+ABCD'

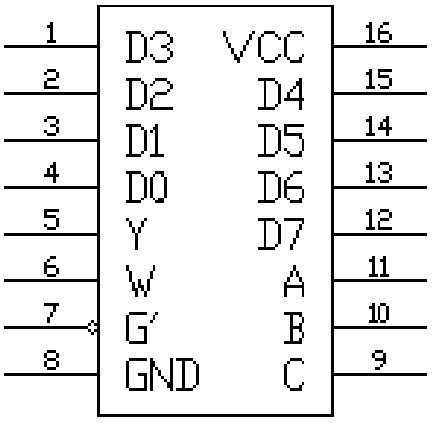
	A	B	C	D	F	
0	0	0	0	0		
1	0	0	0	1		
2	0	0	1	0		
3	0	0	1	1		
4	0	1	0	0		
5	0	1	0	1		
6	0	1	1	0		
7	0	1	1	1		
8	1	0	0	0		
9	1	0	0	1		
10	1	0	1	0		
11	1	0	1	1		
12	1	1	0	0		
13	1	1	0	1		
14	1	1	1	0		
15	1	1	1	1		



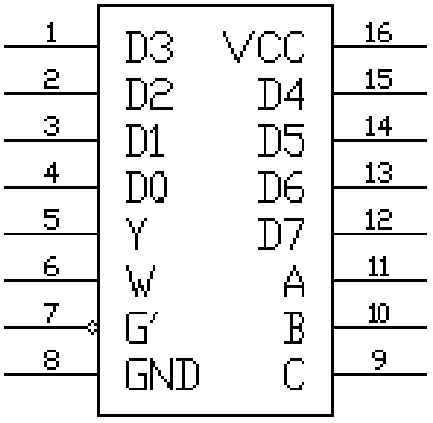
Task 4: Problems with verbal description

Design a combinational circuit (using two 8-to-1 multiplexers) with three inputs, and one output to implement the following function.

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


1
2
3
4
5
6
7
8
D3
D2
D1
D0
Y
W
G'
GND
VCC
D4
D5
D6
D7
A
B
C
16
15
14
13
12
11
10
9

74151


1
2
3
4
5
6
7
8
D3
D2
D1
D0
Y
W
G'
GND
VCC
D4
D5
D6
D7
A
B
C
16
15
14
13
12
11
10
9

74151

Lab Experiment # 10

Building digital logic circuits using Decoders

Objectives

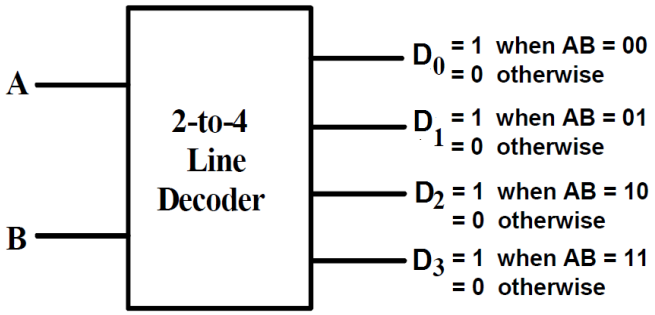
- **To learn how to build combinational logic circuits using decoders.**

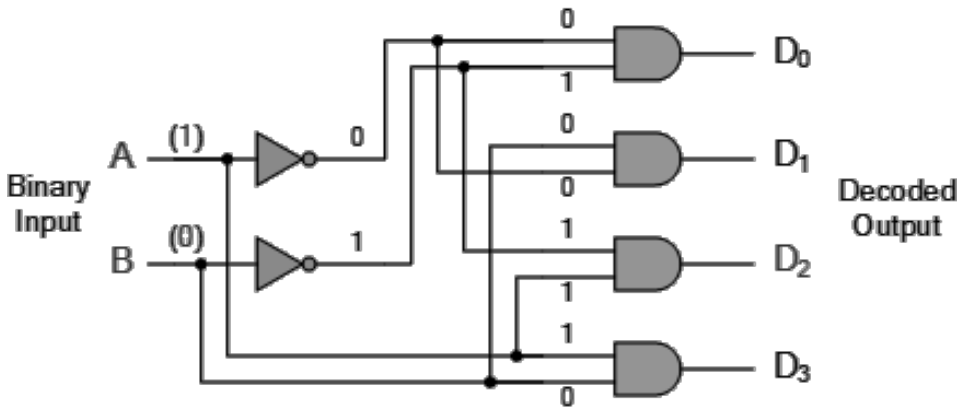
Background

In a **Combinational Logic Circuit**, the output is dependant at all times on the combination of its inputs. Some examples of a combinational circuit include **Multiplexers**, **De-multiplexers**, **Encoders**, **Decoders**, **Full** and **Half Adders** etc.

A Decoder is a circuit with two or more inputs and one or more outputs. Its basic function is to accept a binary word (code) as an input and create a different binary word as an output.

A	B	D1	D2	D3	D4
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



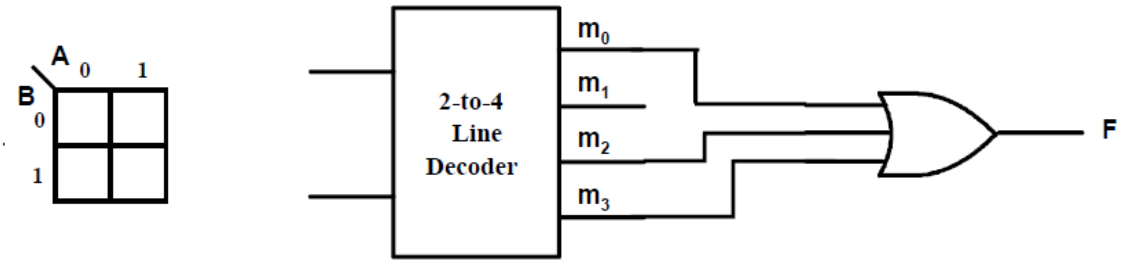


Logic Functions Realized with Decoders:

Example: F = A + B'

= A (B + B') + B' (A + A')

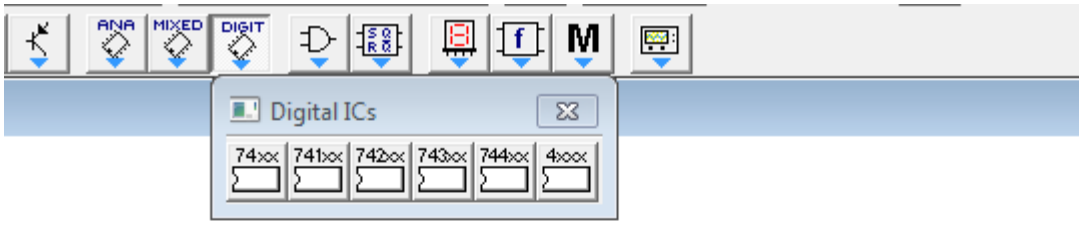
= AB + AB' + A'B'

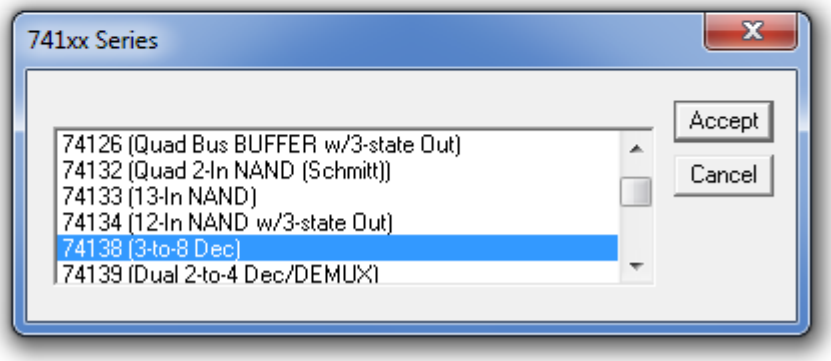


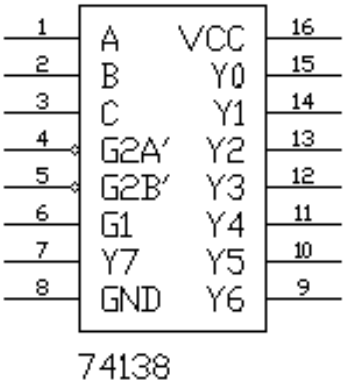
Drawing Decoders using EWB:

Click on the 

button on the toolbar, then drag a 741xx digital IC into your workspace. From the list, select either 74138 (3-8 decoder) or 74154 (4-16 decoder) as shown next.



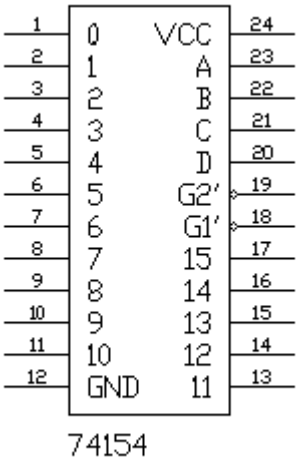




74138 (3-8 decoder)

The 3-to-8 decoder truth table is shown next:

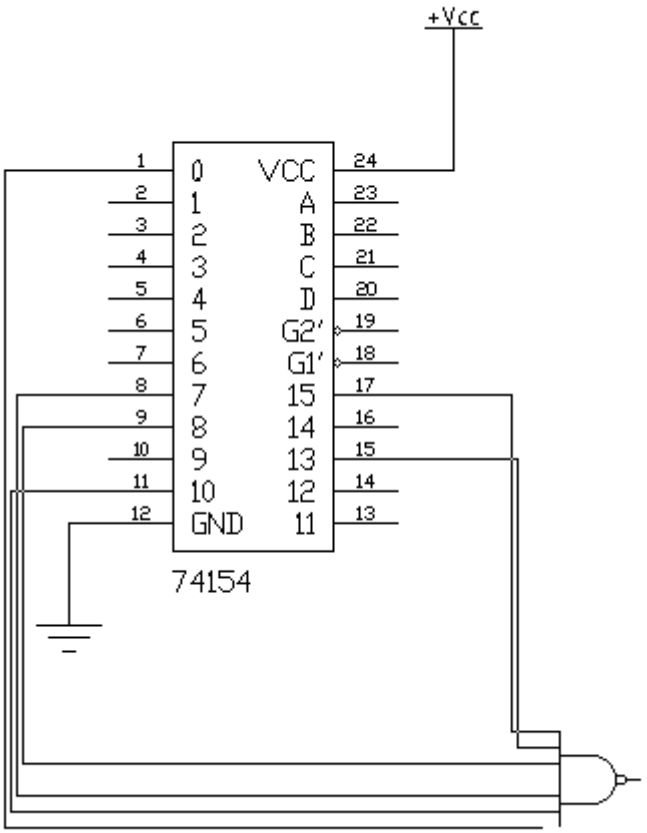
			Select										
G2A’	G1	G2B’	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
x	x	1	x	x	x	1	1	1	1	1	1	1	1
x	0	X	x	x	x	1	1	1	1	1	1	1	1
0	1	0	0	0	0	0	1	1	1	1	1	1	1
0	1	0	0	0	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	0	1	1	1	1	1	0	1	1	1	1
0	1	0	1	0	0	1	1	1	1	0	1	1	1
0	1	0	1	0	1	1	1	1	1	1	0	1	1
0	1	0	1	1	0	1	1	1	1	1	1	0	1
0	1	0	1	1	1	1	1	1	1	1	1	1	0
1	1	0	x	x	x	Output corresponding to stored address 0; all others 1							



74154 (4-16 decoder)

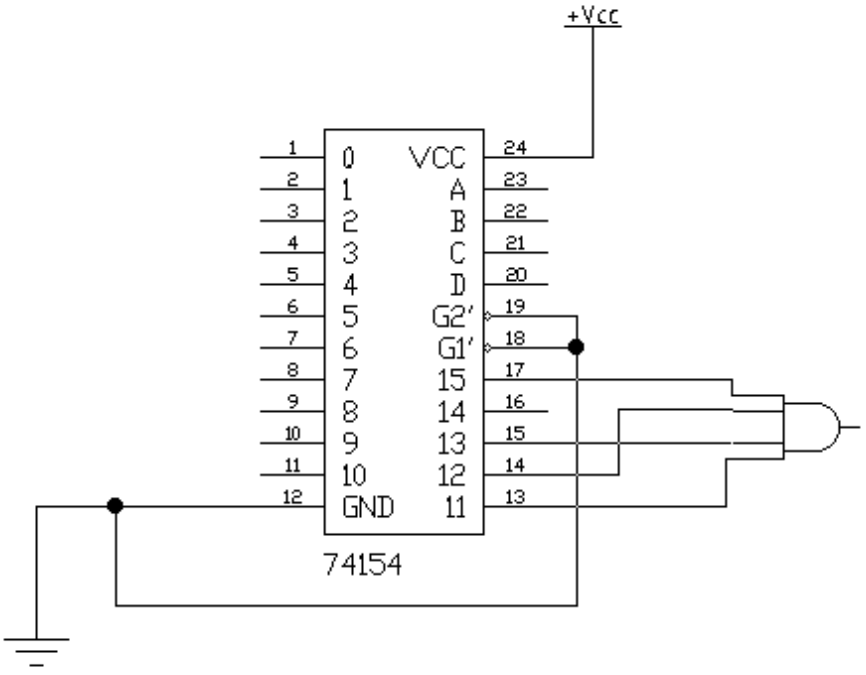
Example: drawing a 4-input function using 74154 (Sum of minterms)

Next is the function F(D, C, B, A)=Σ 0 , 7, 8, 10, 13, 15



Example: drawing a 4-input function using 74154 (Product of maxterms)

Next is the function F(D, C, B, A)= Π 11 , 12, 13, 15



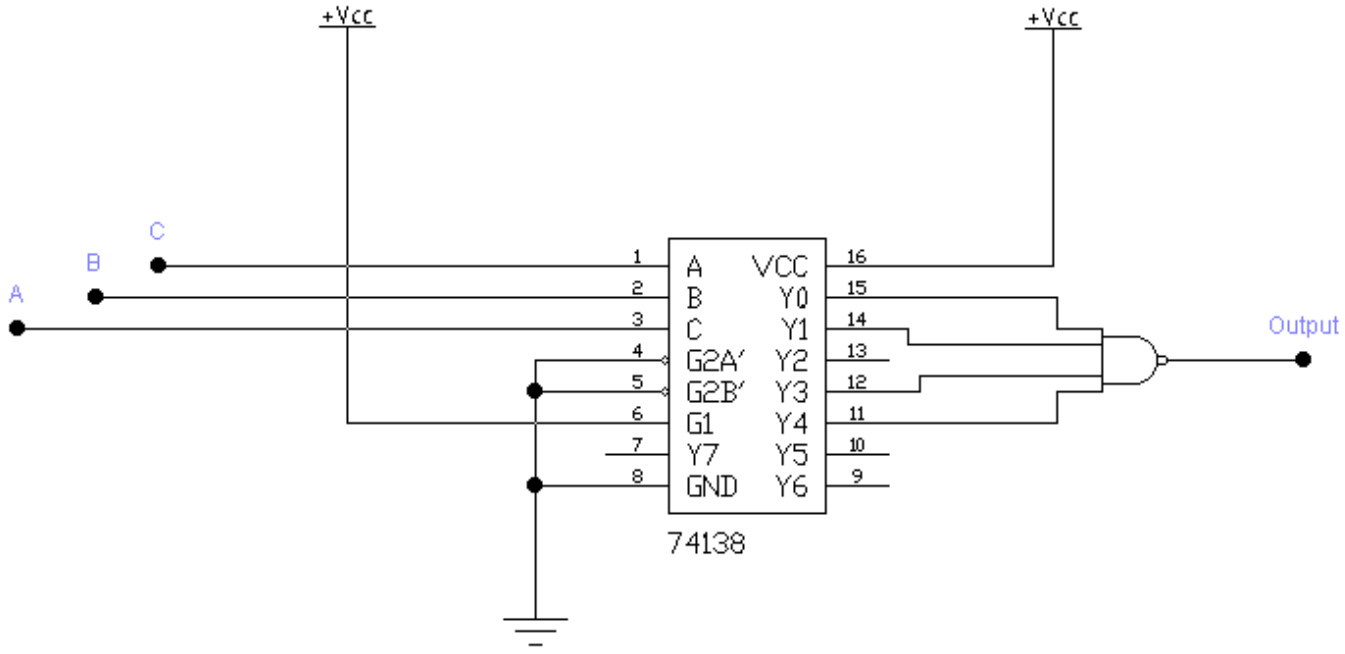
Lab Tasks

Task 1: Implementing 3-variable Boolean expressions using 3-8 decoder

Implement the following function using 3-8 decoders.

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

The above function can be implemented as shown next. Redraw this circuit using EWB.

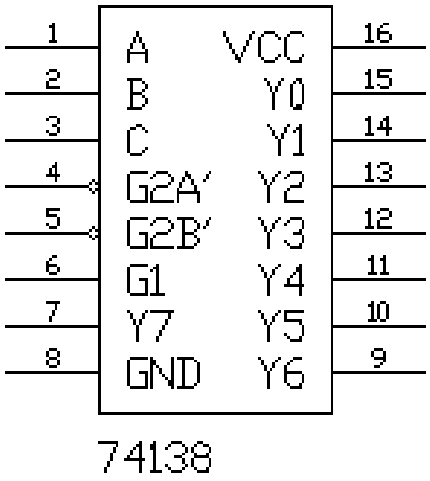


Task 2: Implementing multiple 3-variable Boolean expressions using 3-8 decoder

Implement the following three functions using 3-8 decoders.

	A	B	C	F1	F2	F3
0	0	0	0	0	1	1
1	0	0	1	1	1	0
2	0	1	0	0	0	1
3	0	1	1	1	1	0

4	1	0	0	0	1	1
5	1	0	1	1	0	0
6	1	1	0	1	0	0
7	1	1	1	0	0	1

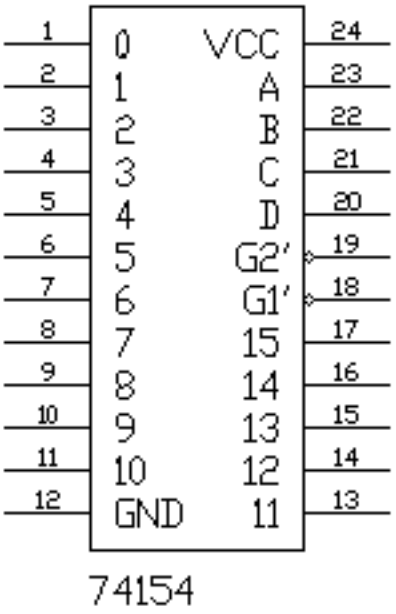


Task 3: Problems with verbal description

Design a combinational circuit (using **one 4-16 decoder**) with four inputs, and one output to implement the following function.

	A	**B**	**C**	**D**	**F**
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	0

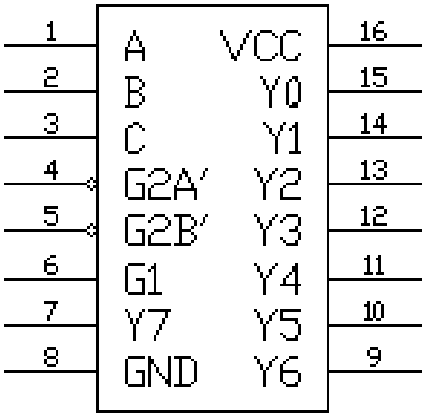
15	1	1	1	1	1



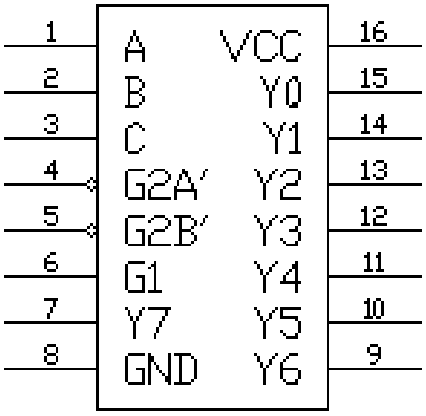
Task 4: Problems with verbal description

Design a combinational circuit (using **two 3-8 decoder**) with four inputs, and one output to implement the following function.

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


1
2
3
4
5
6
7
8
A
B
C
G2A′
G2B′
G1
Y7
GND
VCC
Y0
Y1
Y2
Y3
Y4
Y5
Y6
16
15
14
13
12
11
10
9

74138


1
2
3
4
5
6
7
8
A
B
C
G2A′
G2B′
G1
Y7
GND
VCC
Y0
Y1
Y2
Y3
Y4
Y5
Y6
16
15
14
13
12
11
10
9

74138

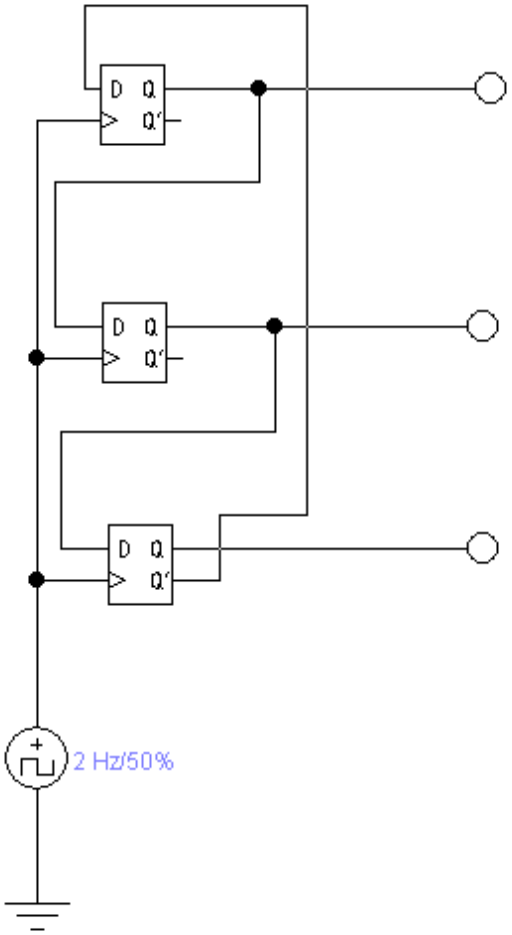
Lab Experiment # 11

Sequential Circuits??

Objectives

- **To learn how to build combinational logic circuits using decoders.**

Background



Lab Experiment # 12

Design and Implementation of Code Convertor

Objective

To design and implement 4-bit

(i) Binary to gray code converter

(ii) Gray to binary code converter

(iii) BCD to excess-3 code converter

(iv) Excess-3 to BCD code converter

Parts required:-

Sl.No.	COMPONENT	SPECIFICATION	QTY.
1.	X-OR GATE	IC 7486	1
2.	AND GATE	IC 7408	1
3.	OR GATE	IC 7432	1
4.	NOT GATE	IC 7404	1

Equipment required:-

- Trainer/ proto board
- Wire cutter
- Patch Cords
- Voltmeter

Theory:

The availability of large variety of codes for the same discrete elements of information results in the use of different codes by different systems. A conversion circuit must be inserted between the two systems if each uses different codes for same information. Thus, code converter is a circuit that makes the two systems compatible even though each uses different binary code.

The bit combination assigned to binary code to gray code. Since each code uses four bits to represent a decimal digit. There are four inputs and four outputs. Gray code is a non-weighted code.

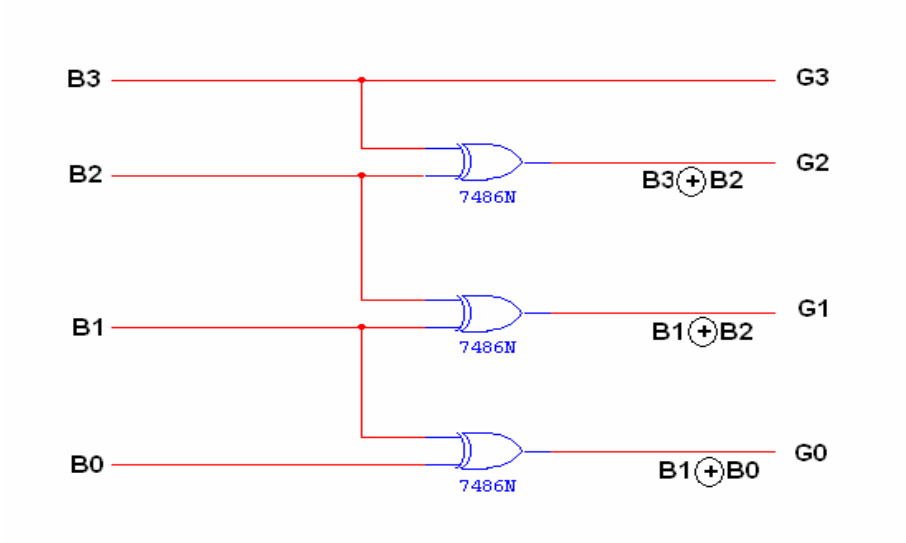
The input variable are designated as B3, B2, B1, B0 and the output variables are designated as C3, C2, C1, Co. from the truth table, combinational circuit is designed. The Boolean functions are obtained from K-Map for each output variable.

A code converter is a circuit that makes the two systems compatible even though each uses a different binary code. To convert from binary code to Excess-3 code, the input lines must supply the bit combination of elements as specified by code and the output lines generate the corresponding bit combination of code. Each one of the four maps represents one of the four outputs of the circuit as a

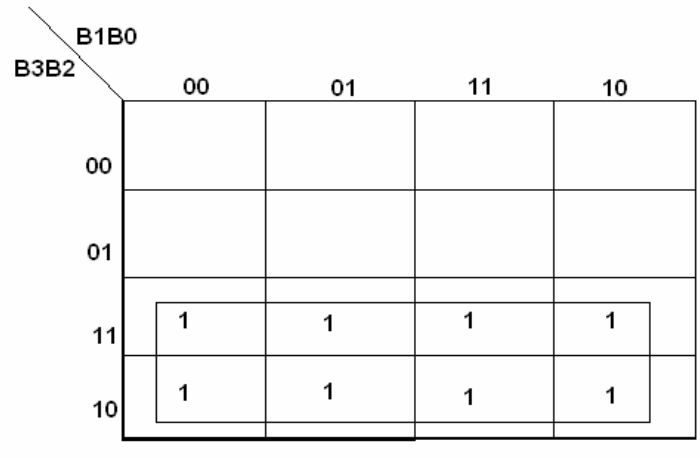
function of the four input variables.

A two-level logic diagram may be obtained directly from the Boolean expressions derived by the maps. These are various other possibilities for a logic diagram that implements this circuit. Now the OR gate whose output is C+D has been used to implement partially each of three outputs.

Logic Diagram:
Binary To Gray Code Convertor

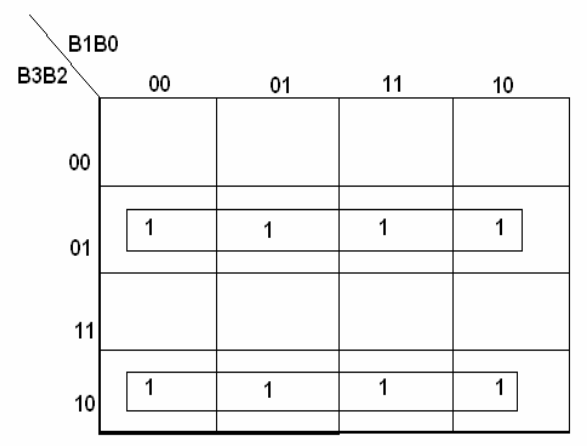


K-Map for G3:



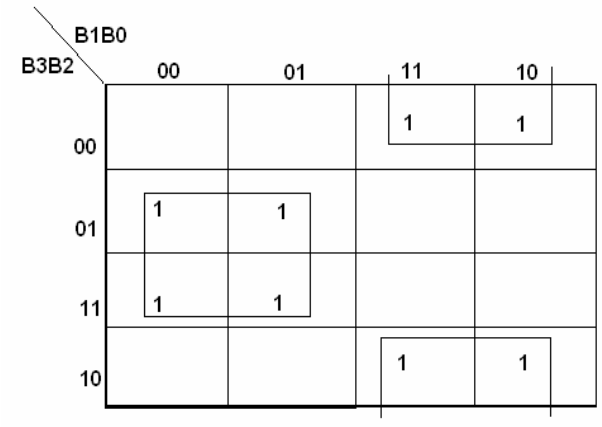
$G_3 = B_3$

K-Map for G2:



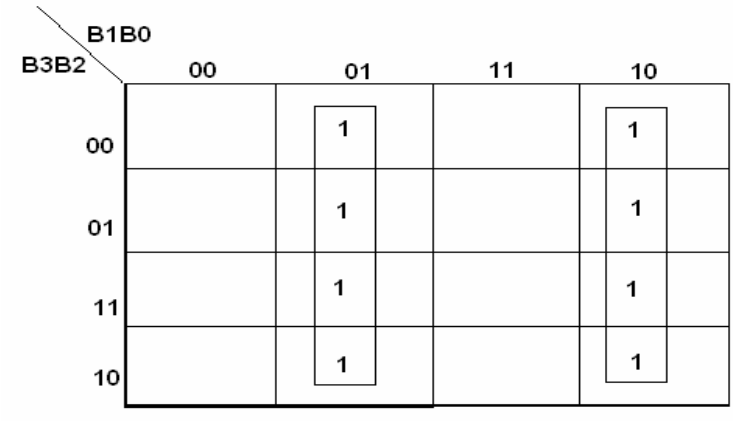
G2 = B3⊕B2

K-Map for G_1:



$G1 = B1 \oplus B2$

K-Map for G_0:



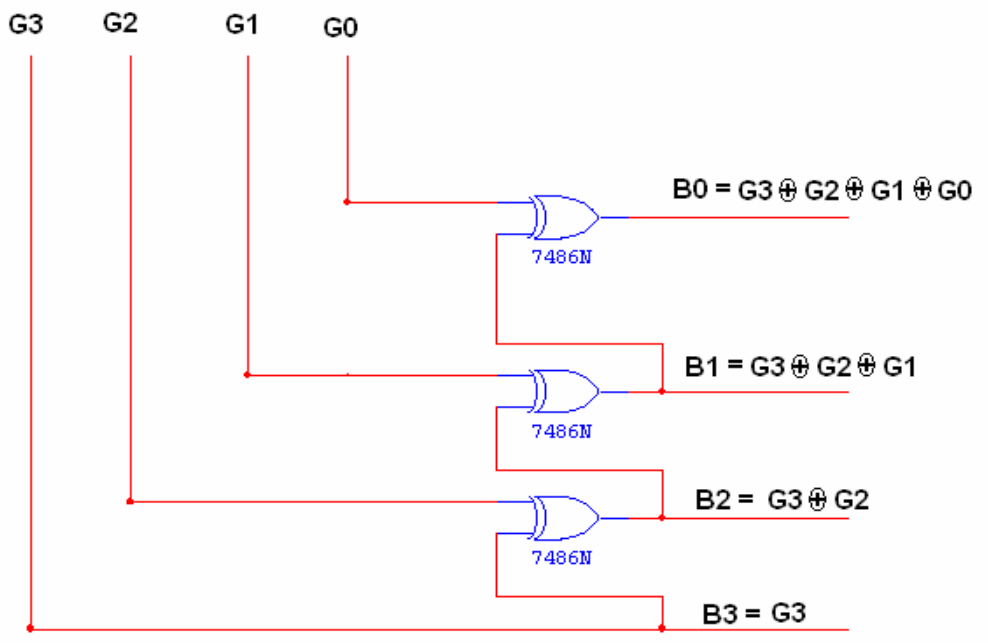
$G0 = B1 \oplus B0$

Truth Table:

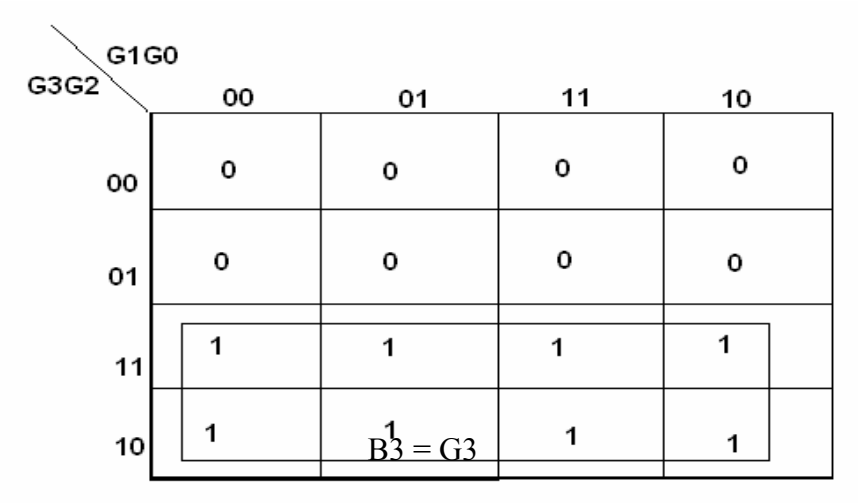
Binary input				Gray code output			
B3	B2	B1	B0	G3	G2	G1	G0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Logic Diagram:

Gray Code To Binary Convertor

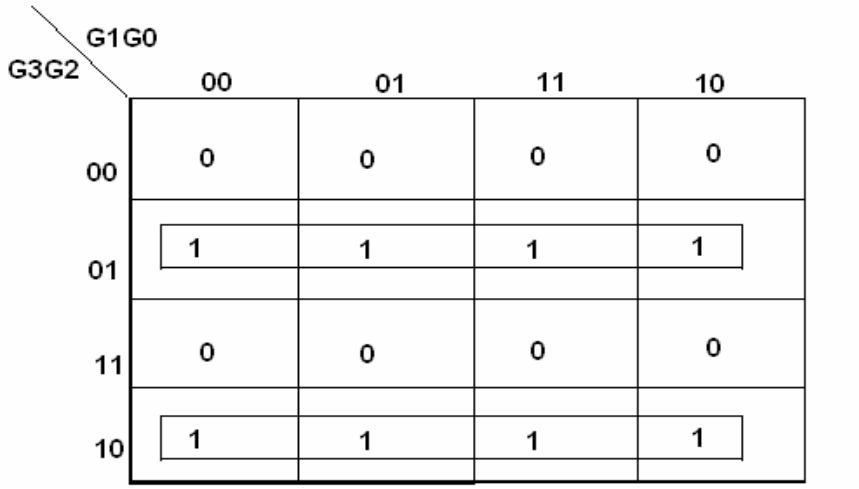


K-Map for B_3:



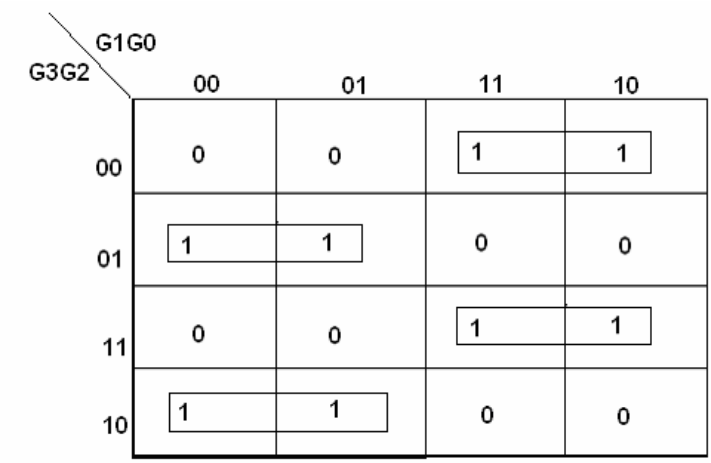
$B3 = G3$

K-Map for B_2:



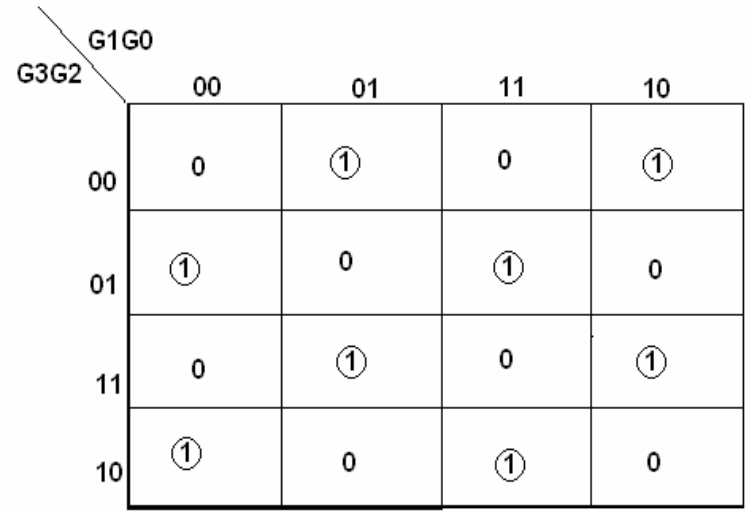
$B2 = G3 \oplus G2$

K-Map for B_1:



$B1 = G3 \oplus G2 \oplus G1$

K-Map for B_0:



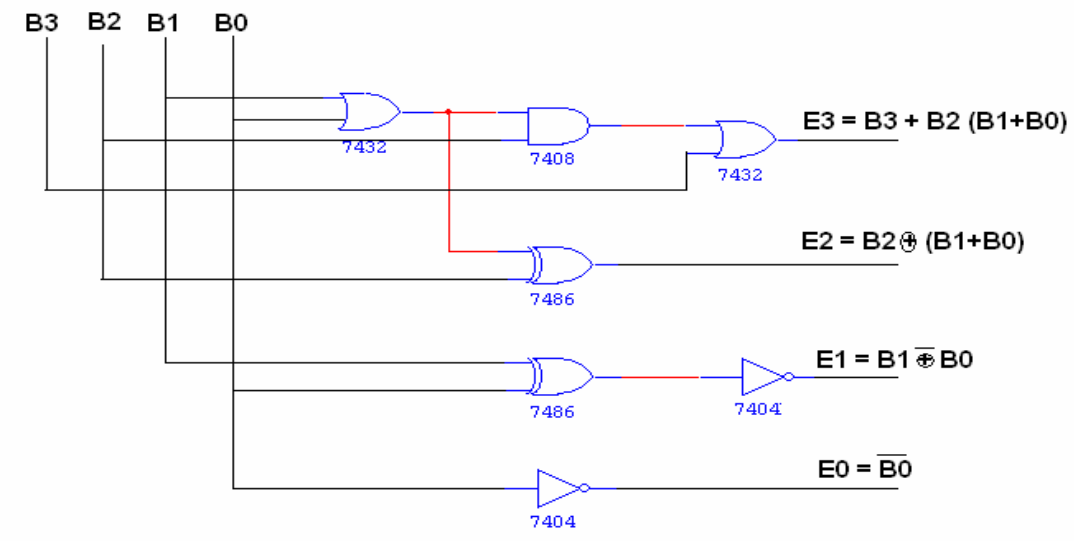
$B0 = G3 \oplus G2 \oplus G1 \oplus G0$

Truth Table:

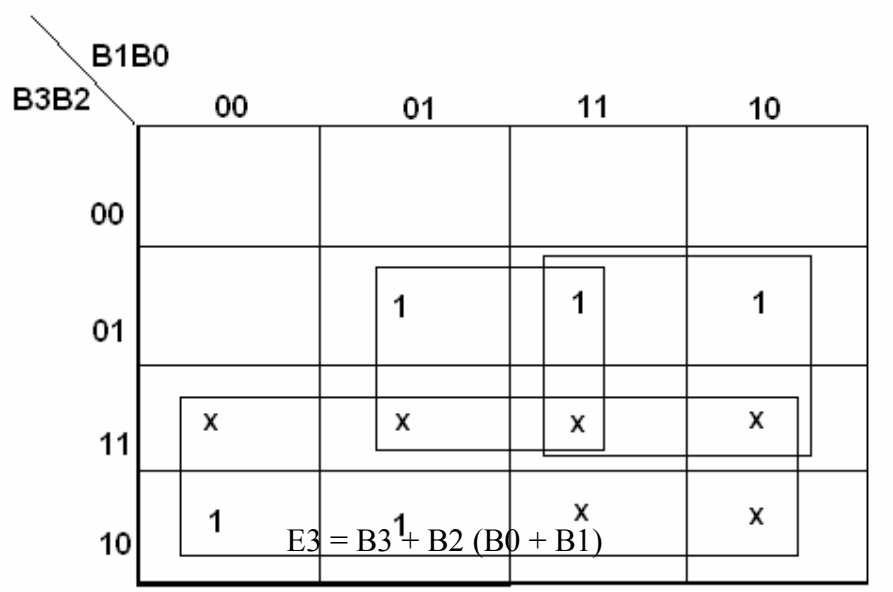
Gray Code				Binary Code			
G3	G2	G1	G0	B3	B2	B1	B0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

Logic Diagram:

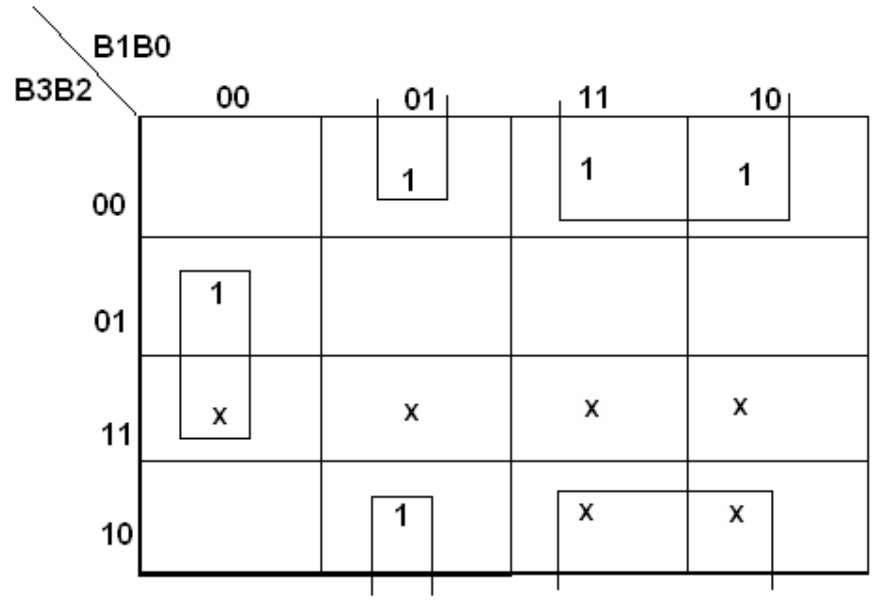
Bcd To Excess-3 Convertor



K-Map for E3:



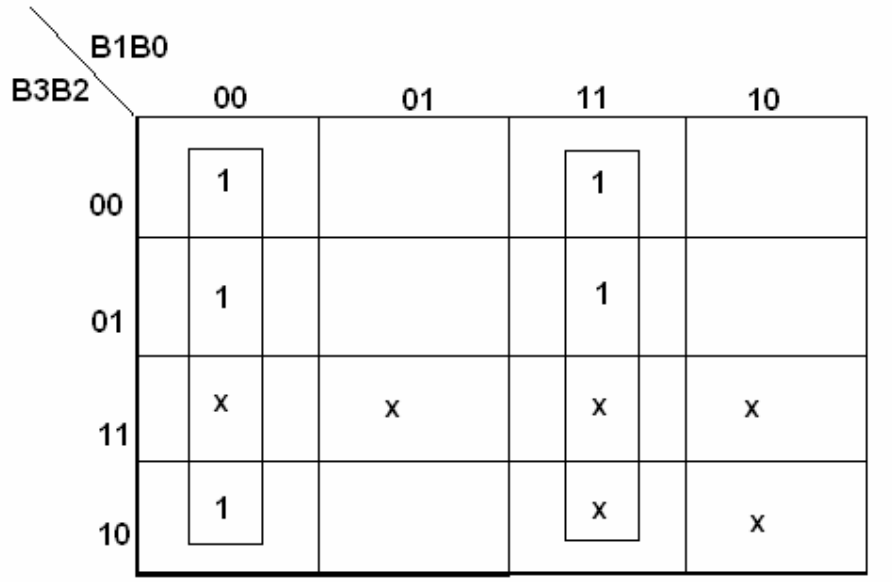
E3 = B3 + B2 (B0 + B1)



E2 = B2 ⊕ (B1 + B0)

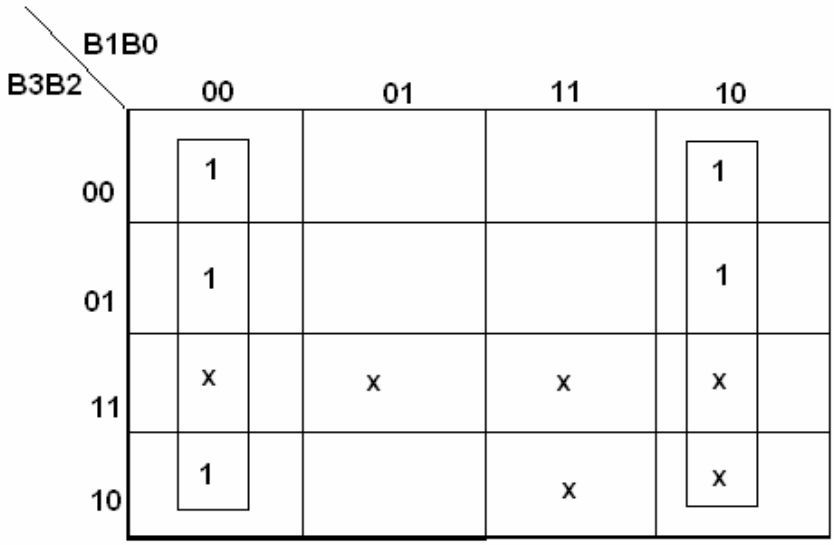
K-Map for E2:

K-Map for E_1:



$E1 = B1 \bar{\oplus} B0$

K-Map for E_0:



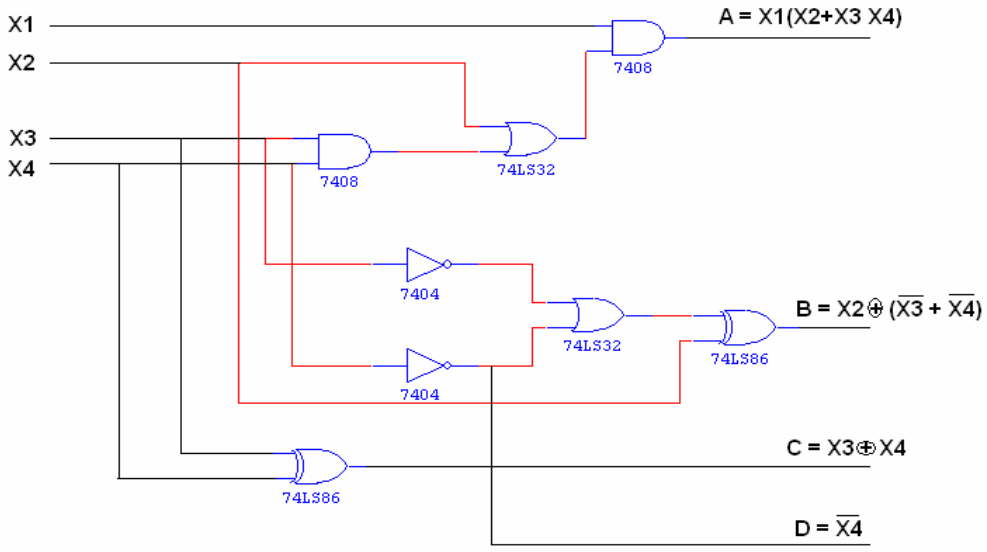
$E0 = \overline{B0}$

Truth Table:

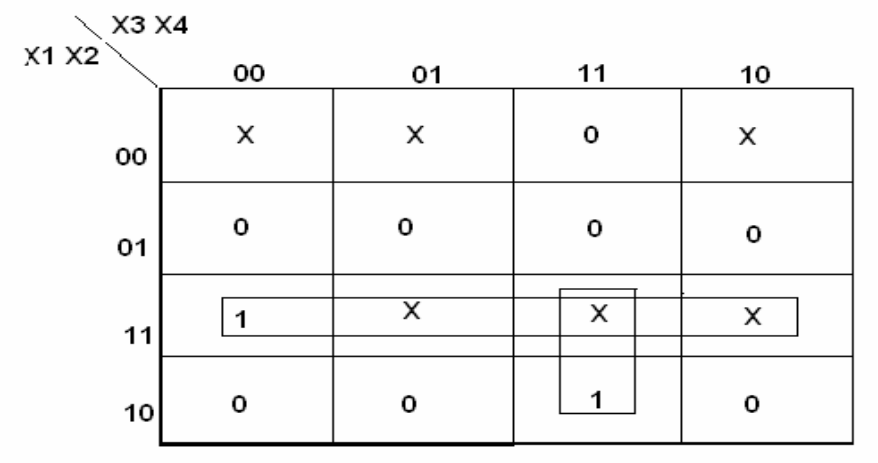
BCD input				Excess – 3 output			
B3	B2	B1	B0	G3	G2	G1	G0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

Logic Diagram:

Excess-3 To Bcd Convertor

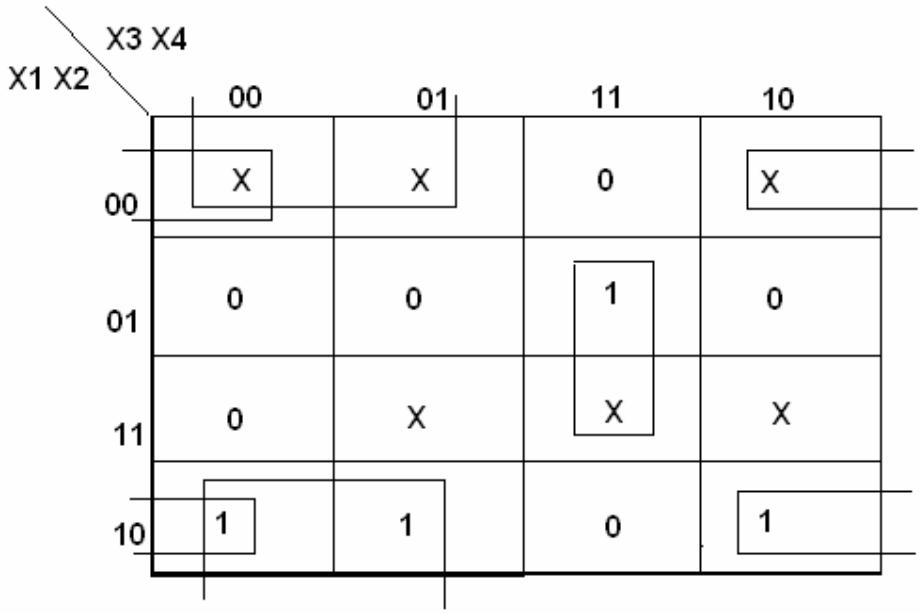


K-Map for A:



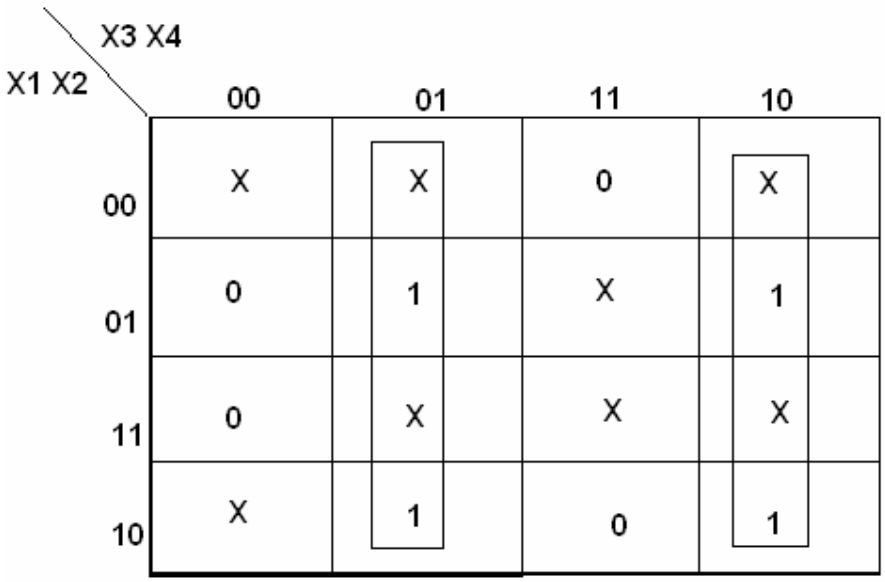
A = X1 X2 + X3 X4 X

K-Map for B:



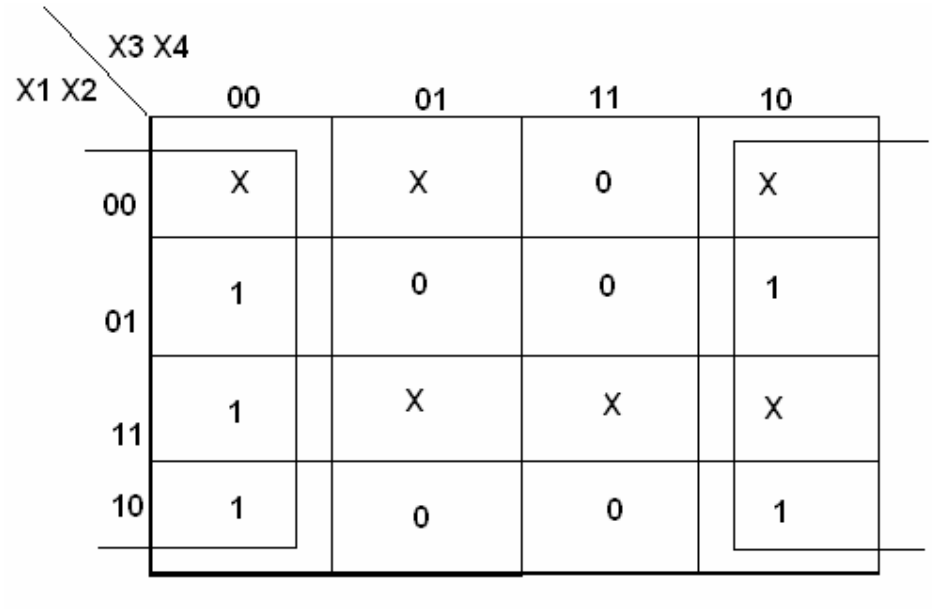
$B = X2 \oplus (\overline{X3} + \overline{X4})$

K-Map for C:



$C = X3 \oplus X4$

K-Map for D:



$D = \overline{X4}$

Truth Table:

Excess – 3 Input				BCD Output			
B3	B2	B1	B0	G3	G2	G1	G0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	0
1	0	0	0	0	1	0	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	1	1
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	1

Procedure:

(i) verify the gates
(ii) Connect the proper power supply
(iii) Connections were given as per circuit diagram.
(iv) Logical inputs were given as per truth table
(v) Observe the logical output and verify with the truth tables.

Lab Experiment # 13

Design and Implementation of Magnitude Comparator

Objective

To design and implement

(i) 2 – Bit magnitude comparator using basic gates.

(ii) 8 – Bit magnitude comparator using IC 7485.

Parts Required

Sl.No.	COMPONENT	SPECIFICATION	QTY.
1.	AND GATE	IC 7408	2
2.	X-OR GATE	IC 7486	1
3.	OR GATE	IC 7432	1
4.	NOT GATE	IC 7404	1
5.	4-BIT MAGNITUDE COMPARATOR	IC 7485	2

Theory:

The comparison of two numbers is an operator that determines one number is greater than, less than (or) equal to the other number. A magnitude comparator is combinational circuits that compares two numbers A and B and determine their relative magnitude. The outcome of the comparator is specified by three binary variables that indicate whether A>B, A=B or A<B.

A = A3 A2 A1 A0 B = B3 B2 B1 B0

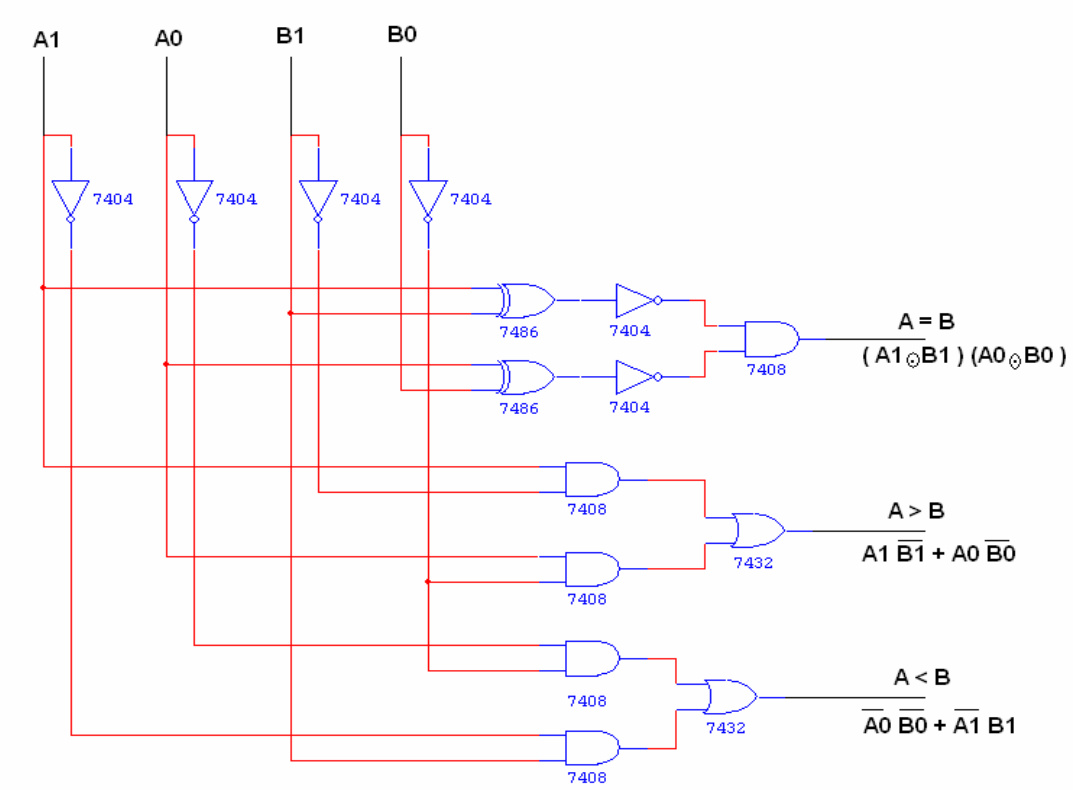
The equality of the two numbers and B is displayed in a combinational circuit designated by the symbol (A=B).

This indicates A greater than B, then inspect the relative magnitude of pairs of significant digits starting from most significant position. A is 0 and that of B is 0.

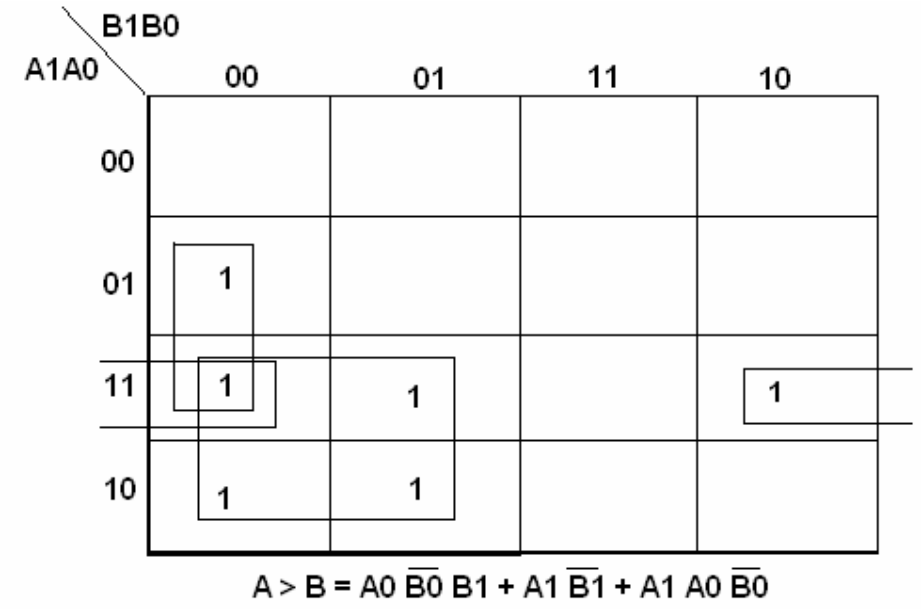
We have A<B, the sequential comparison can be expanded as

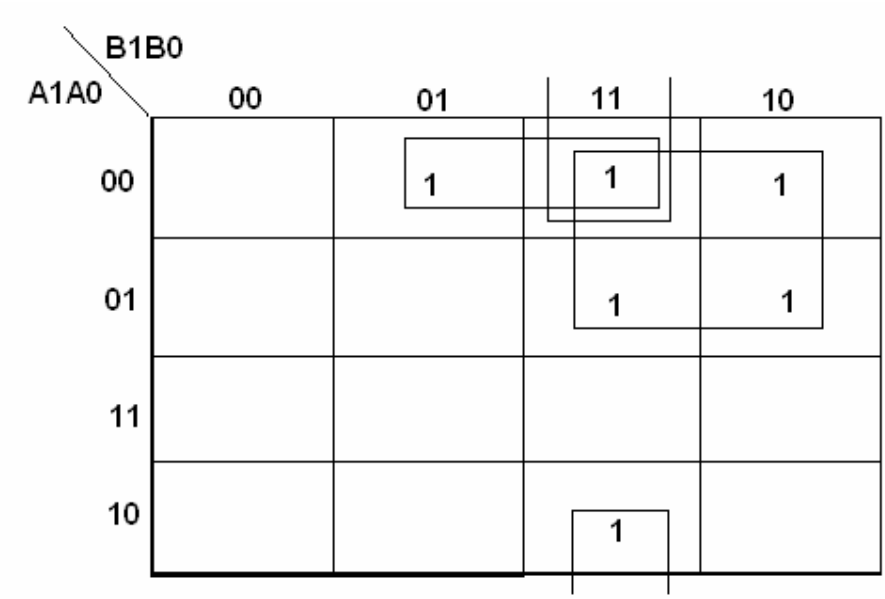
A>B = A3B31 + X3A2B2 + X3X2A1B1 + X3X2X1A0B0 A_1<B = A31B3 + X3$A_1$21B2 + X3X2A11B1 + X3X2X1A01B0

Logic Diagram:
2 Bit Magnitude Comparator

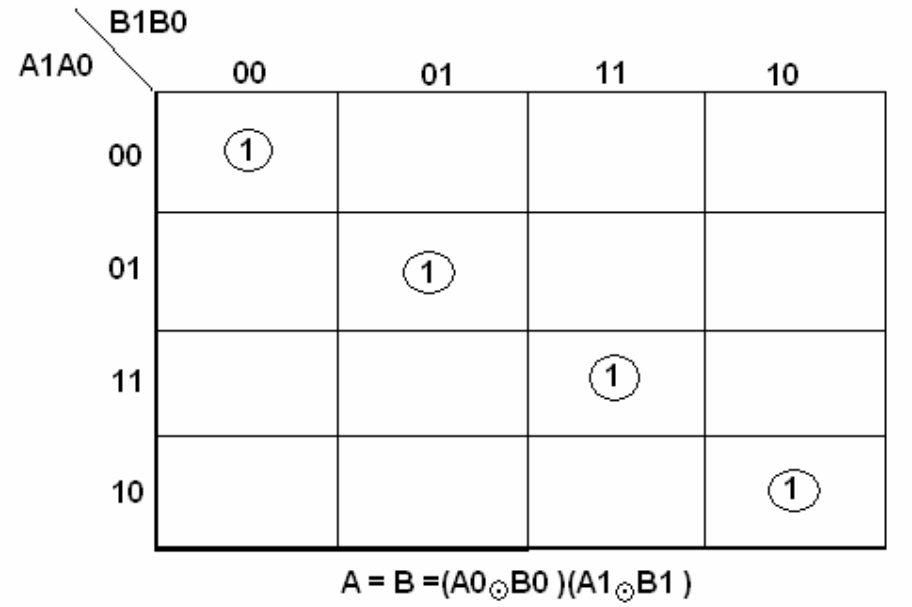


K MAP


B1B0
A1A0
00
01
11
10
00
01
11
10
1
1
1
1
1
1
A > B = A0 $\overline{B0}$ B1 + A1 $\overline{B1}$ + A1 A0 $\overline{B0}$



$A < B = \overline{A1}\ \overline{A0}\ B0 + \overline{A0}\ B0\ B1 + \overline{A1}\ B1$

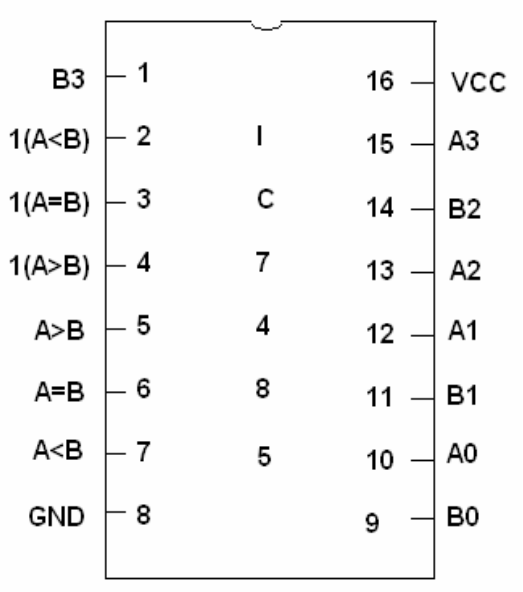


$A = B = (A0 \odot B0)(A1 \odot B1)$

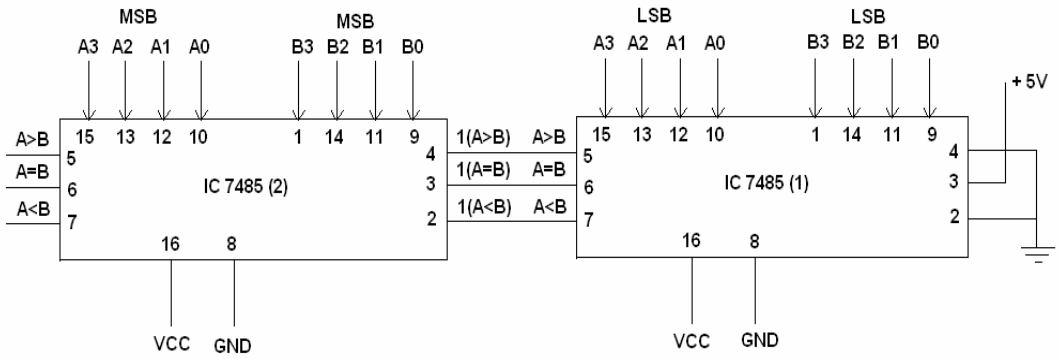
Truth Table

A	A	B	B	A > B	A = B	A < B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

Pin Diagram For IC 7485



Logic Diagram
8 Bit Magnitude Comparator



Truth Table

A	B	A>B	A=B	A<B
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0	1	0
0 0 0 1 0 0 0 1	0 0 0 0 0 0 0 0	1	0	0
0 0 0 0 0 0 0 0	0 0 0 1 0 0 0 1	0	0	1

Procedure

(i) Verify the gates
(ii) Connections are given as per circuit diagram.
(iii) Logical inputs are given as per circuit diagram.
(iv) Observe the output and verify the truth table.

Lab Experiment # 14

Construction and verification of 4 bit ripple counter and mod 10/mod 12 ripple counter

Objective

To design and verify 4-bit ripple counter mod 10/ mod 12 ripple counter.

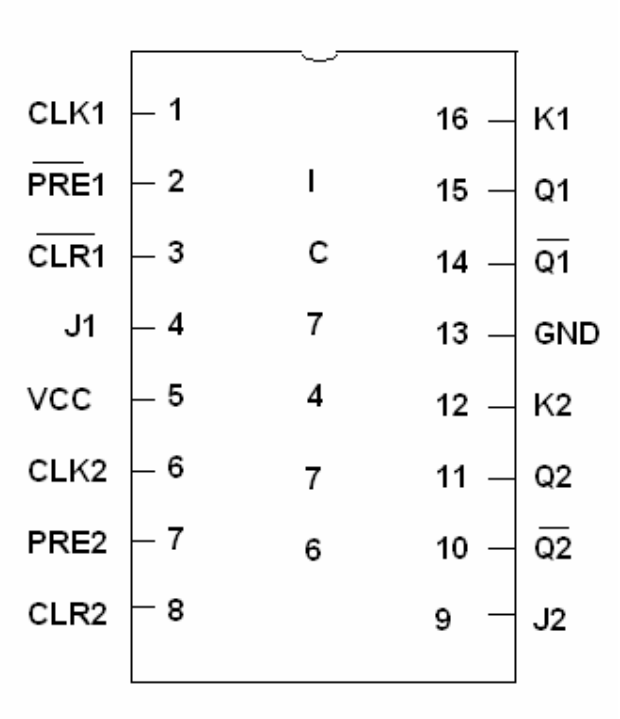
Parts required

S No.	COMPONENT	SPECIFICATION	QTY.
1.	JK FLIP FLOP	IC 7476	2
2.	NAND GATE	IC 7400	1

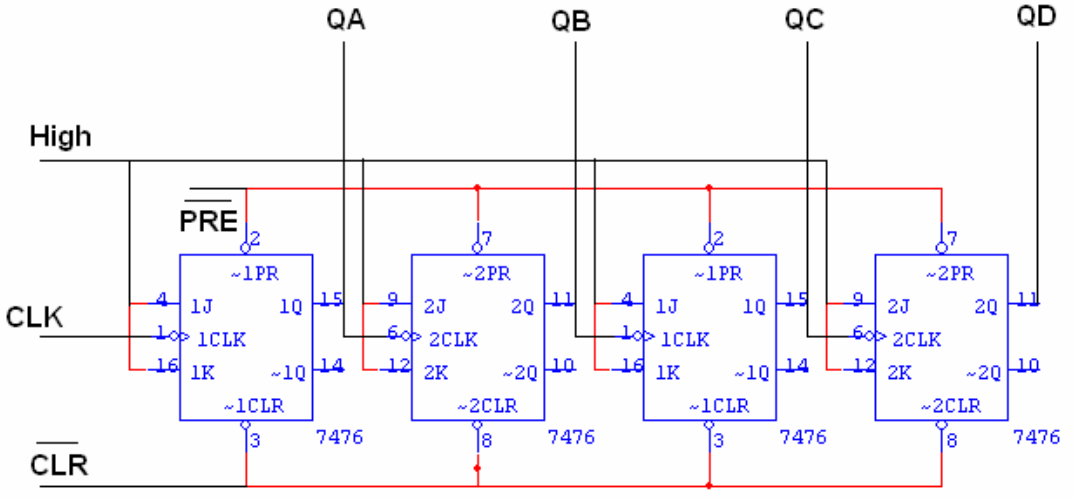
Theory

A counter is a register capable of counting number of clock pulse arriving at its clock input. Counter represents the number of clock pulses arrived. A specified sequence of states appears as counter output. This is the main difference between a register and a counter. There are two types of counter, synchronous and asynchronous. In synchronous common clock is given to all flip flop and in asynchronous first flip flop is clocked by external pulse and then each successive flip flop is clocked by Q or Q output of previous stage. A soon the clock of second stage is triggered by output of first stage. Because of inherent propagation delay time all flip flops are not activated at same time which results in asynchronous operation.

Pin Diagram for IC 7476:



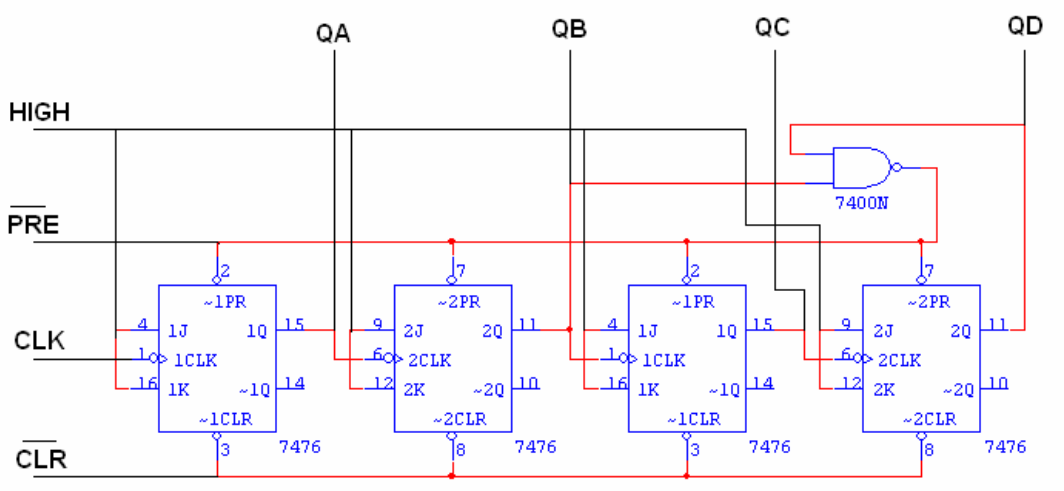
Logic Diagram For 4 Bit Ripple Counter:



Truth Table

CLK	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	1	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1

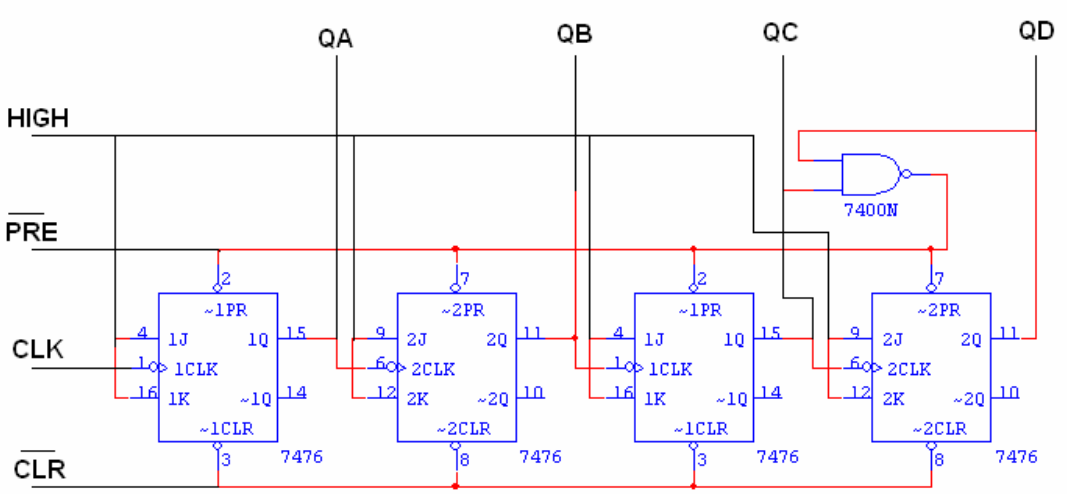
Logic Diagram for Mod - 10 Ripples Counter



Truth Table

CLK	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	0	0	0

Logic Diagram for Mod - 12 Ripples Counter



Truth Table

CLK	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	0	0

Procedure

(i) Connections are given as per circuit diagram.
(ii) Logical inputs are given as per circuit diagram.
(iii) Observe the output and verify the truth table.

Lab Experiment # 15

Design and Implementation Of 3 Bit Synchronous Up/Down Counter

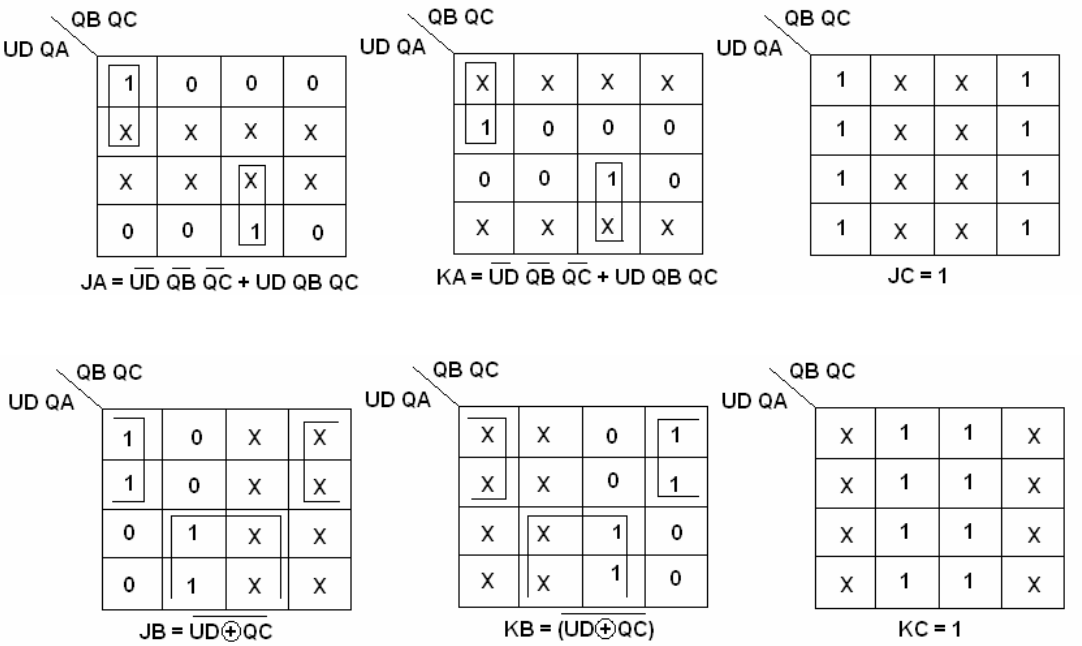
Apparatus Required

S No.	COMPONENT	SPECIFICATION	QTY.
1.	JK FLIP FLOP	IC 7476	2
2.	3 I/P AND GATE	IC 7411	1
3.	OR GATE	IC 7432	1
4.	XOR GATE	IC 7486	1
5.	NOT GATE	IC 7404	1
6.	IC TRAINER KIT	-	1
7.	PATCH CORDS	-	35

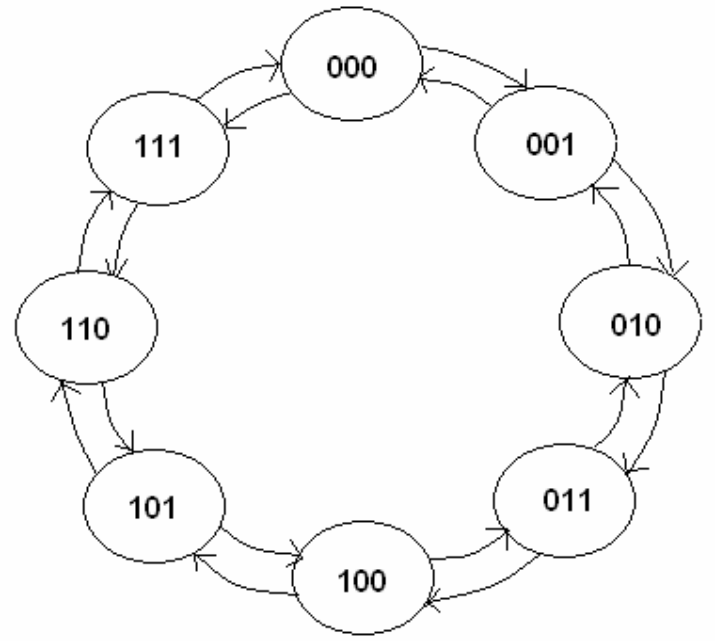
Theory

A counter is a register capable of counting number of clock pulse arriving at its clock input. Counter represents the number of clock pulses arrived. An up/down counter is one that is capable of progressing in increasing order or decreasing order through a certain sequence. An up/down counter is also called bidirectional counter. Usually up/down operation of the counter is controlled by up/down signal. When this signal is high counter goes through up sequence and when up/down signal is low counter follows reverse sequence.

K Map



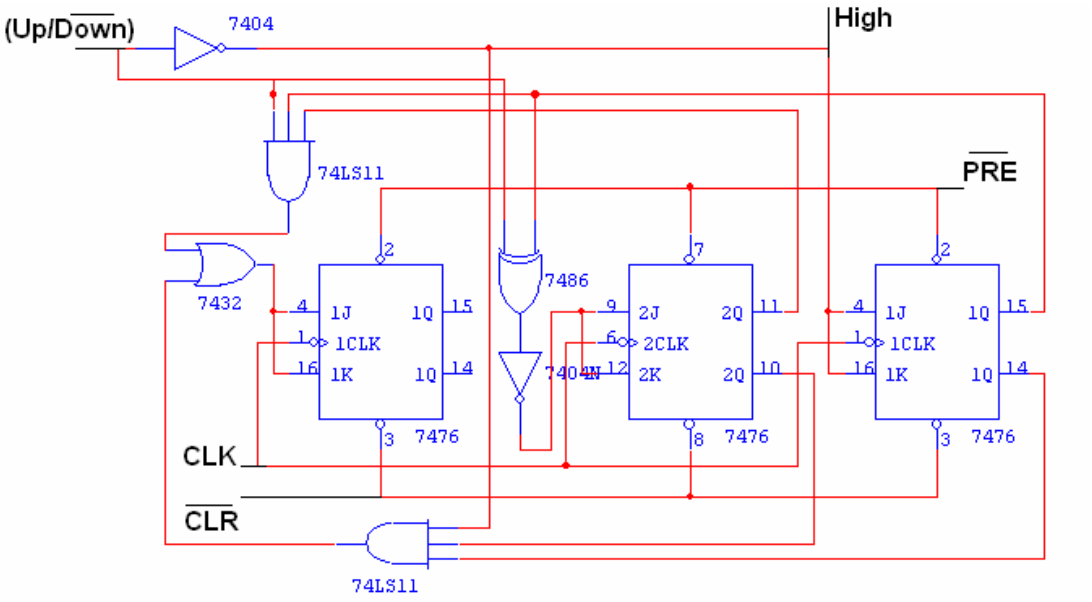
State Diagram



Characteristics Table

Q	Qt+1	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Logic Diagram



Truth Table

Input Up/Down	Present State			Next State			A		B		C	
	Q_A	Q_B	Q_C	Q_{A+1}	Q_{B+1}		J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	1	1	1	1	X	1	X	1	X
0	1	1	1	1	1	0	X	0	X	0	X	1
0	1	1	0	1	0	1	X	0	X	1	1	X
0	1	0	1	1	0	0	X	0	0	X	X	1
0	1	0	0	0	1	1	X	1	1	X	1	X
0	0	1	1	0	1	0	0	X	X	0	X	1
0	0	1	0	0	0	1	0	X	X	1	1	X
0	0	0	1	0	0	0	0	X	0	X	X	1
1	0	0	0	0	0	1	0	X	0	X	1	X
1	0	0	1	0	1	0	0	X	1	X	X	1
1	0	1	0	0	1	1	0	X	X	0	1	X
1	0	1	1	1	0	0	1	X	X	1	X	1
1	1	0	0	1	0	1	X	0	0	X	1	X
1	1	0	1	1	1	0	X	0	1	X	X	1
1	1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	1	0	0	0	X	1	X	1	X	1

Procedure:

(i) Connections are given as per circuit diagram.
(ii) Logical inputs are given as per circuit diagram.
(iii) Observe the output and verify the truth table.

Lab Experiment # 16

Design and Implementation of Shift Register

AIM

To design and implement

(i) Serial in serial out

(ii) Serial in parallel out

(iii) Parallel in serial out

(iv) Parallel in parallel out

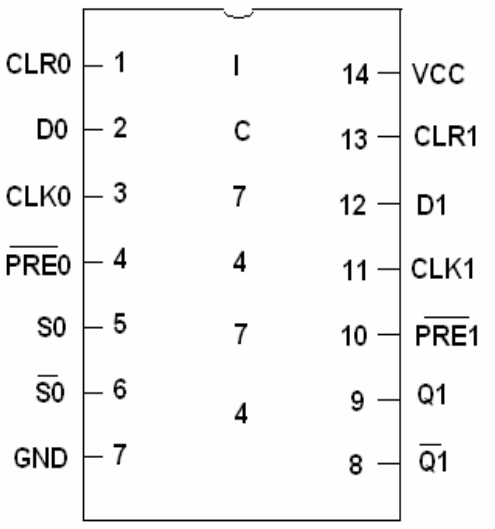
Apparatus Required

S No.	COMPONENT	SPECIFICATION	QTY.
1.	D FLIP FLOP	IC 7474	2
2.	OR GATE	IC 7432	1
3.	IC TRAINER KIT	-	1
4.	PATCH CORDS	-	35

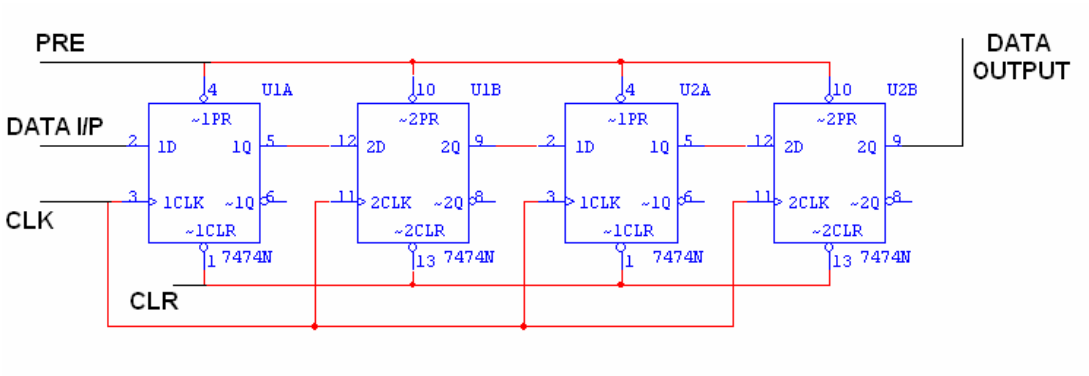
Theory

A register can shift its binary information in one or both directions is known as shift register. The logical configuration of shift register consists of a D-Flip flop cascaded with output of one flip flop connected to input of next flip flop. All flip flops receive common clock pulses which causes the shift in the output of the flip flop. The simplest possible shift register is one that uses only flip flop. The output of a given flip flop is connected to the input of next flip flop of the register. Each clock pulse shifts the content of register one-bit position to right.

Pin Diagram



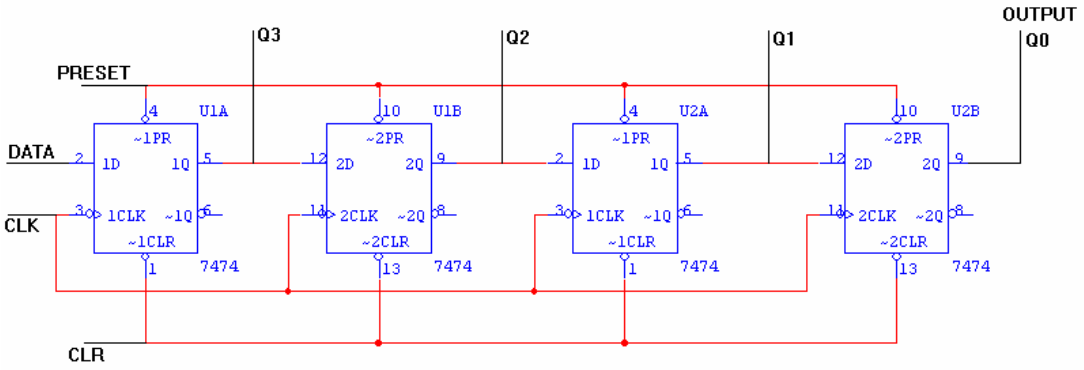
Logic Diagram
Serial in Serial Out



Truth Table

CLK	Serial in	Serial out
1	1	0
2	0	0
3	0	0
4	1	1
5	X	0
6	X	0
7	X	1

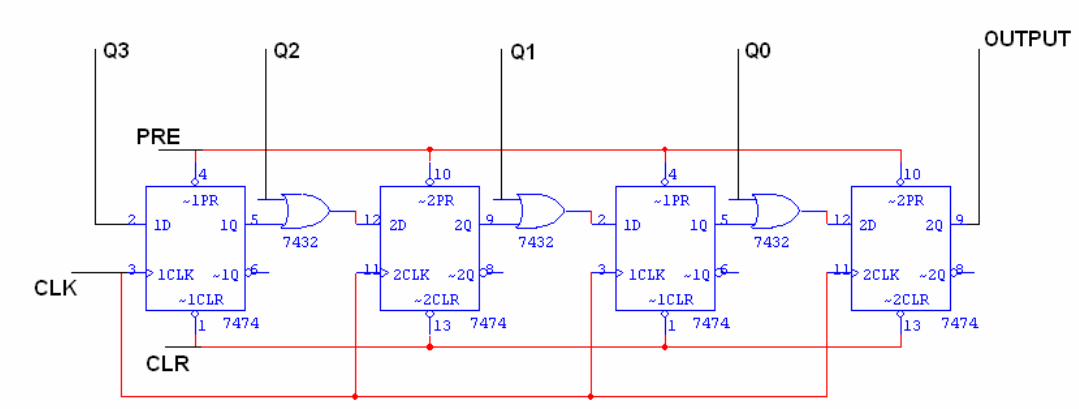
Logic Diagram
Serial in Parallel Out



Truth Table

CLK	DATA	OUTPUT			
		QA	QB	QC	QD
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	0	0	1

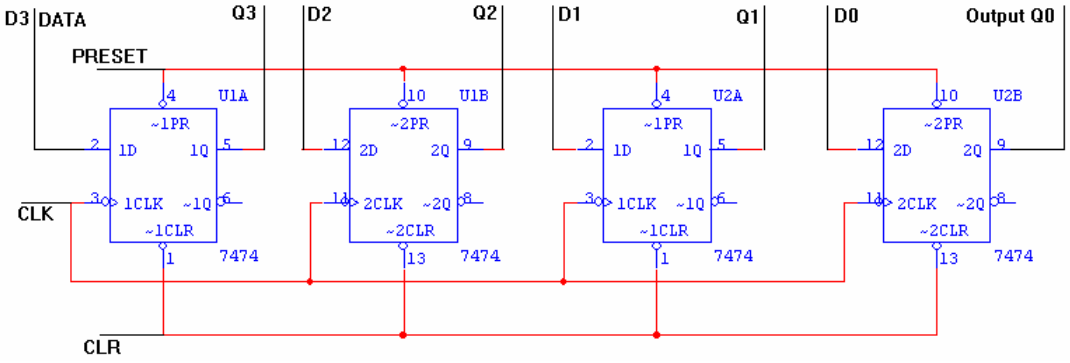
Logic Diagram: Parallel in Serial Out



Truth Table

CLK	Q3	Q2	Q1	Q0	O/P
0	1	0	0	1	1
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	1

Logic Diagram
Parallel in Parallel Out



Truth Table

CLK	DATA INPUT				OUTPUT			
	DA	DB	DC	DD	QA	QB	QC	QD

1	1	0	0	1	1	0	0	1
2	1	0	1	0	1	0	1	0

Procedure

(i) Connections are given as per circuit diagram.
(ii) Logical inputs are given as per circuit diagram.
(iii) Observe the output and verify the truth table.