Instruction manual

Experiment 1

Objective: To design a switch (NOT gate) using a transistor.

Apparatus Required:

1- Electronic circuit trainer

2- Dual channel Oscilloscope

3- Electronic components

Theory:

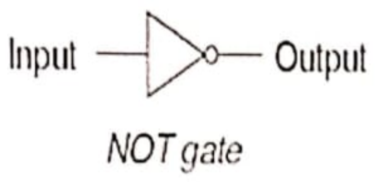
Definition: A logic gates is an electronic circuit which make decisions. It has one output and one or more inputs.

Formula and circuit diagram:

NOT-Gate

Its output is NOT the same as its input. It is also called inverter because it inverts the input. It has one input and one output as shown in figure below.

[1] NOT Gate : $Y = \overline{A}$



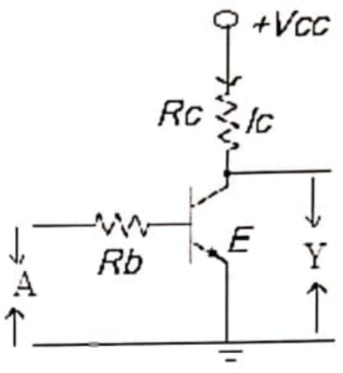


Figure 1: RTL as NOT gate

OR-gate

The output of this circuit is logic "1" when either one input is logic "1" as shown in figure below.

[2] AND Gate : **Y= A . B**

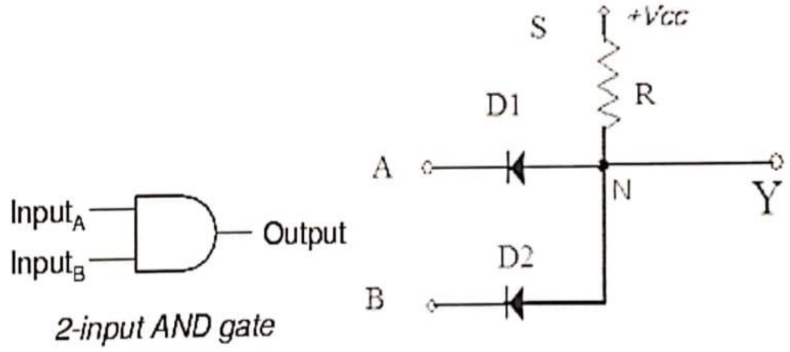


Figure 2: Diode circuit as AND gate

[3] OR Gate : **Y= A + B**

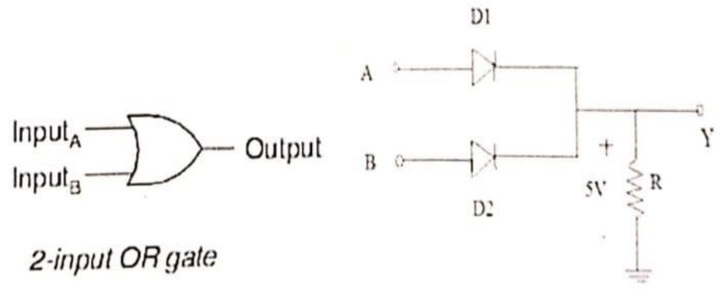
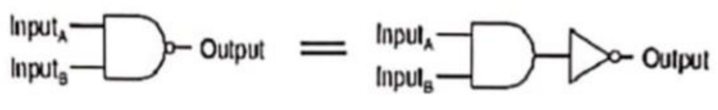


Figure 3: Diode circuit as OR gate

[4] NAND Gate : $Y = \overline{A.B}$



2-input NAND gate *Equivalent gate circuit*

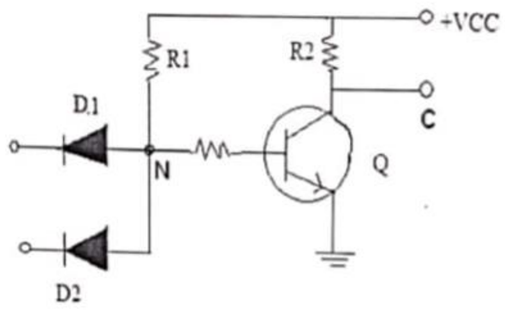
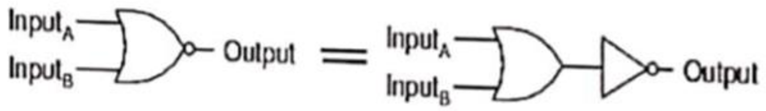


Figure 4: DTL as NAND Gate

[5] NOR Gate: $Y = \overline{A + B}$



2-input NOR gate *Equivalent gate circuit*

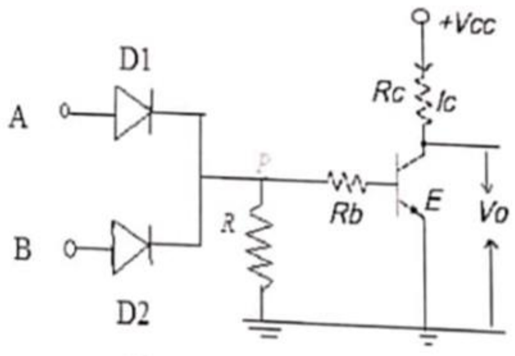


Figure 5: DTL as NOR Gate

Observation :

1. **Truth Table for NOT gate**

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

2. **Truth Table for AND gate**

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

3. **Truth Table for OR gate**

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

4. **Truth Table for NAND gate**

A	B	A.B	$Y = \overline{A.B}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

5. **Truth Table for NOR gate**

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

RESULT: The truth table for basic logic gates are verified.

Precaution:

1. **Input must be off 5 volts.**
2. **Connections should be jointed correctly.**
3. **Connections must be tight.**

Experiment No.2

Objective: - To verify and design AND, OR, NOT, EXOR logic gates using NAND gate.

Apparatus: - Digital trainer kit wires, probes, etc.

Theory:-

Universal gates: -

The Nand and Nor gates are called as universal gates, because it is possible to implement anyBoolean expression with the help of only Nand or only Nor gates.

Hence a user can build any combinational circuit with the help of only Nand gates or only Norgates.

The NAND & NOR gates are called as „Universal gates". Because it is possible to implement any Boolean expression with the help of only NAND or only NOR gate. We canconstruct AND, OR, NOT, X-OR & X-NOR gates.

The Boolean expression for NAND gate is,

$$X = \overline{AB}$$

The Boolean expression for NOR gate is,

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

This is a great advantage because a user will have to make a stock of only Nand

or Nor gates.All gates using Nand Gate:-

1) Not using Nand:-

   The Boolean expression for NOT gate $\overline{\text{is}}$ A = A Fig. shows the realization of a NOT gate using a two i/p NAND gate. As both i/p"s are connected together we can write i/p A=B=A

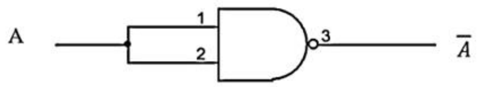
   So o/p is given as

$$Y = \overline{A.B}$$

$$Y = \overline{A.A} \qquad \because A = B$$

But A.A=A ∵ by AND law

$$Y = \overline{A}$$



2) AND using NAND:-

The Boolean expression for an AND gate is Y=A.B

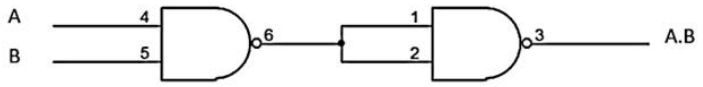
Taking double inversion,

$$Y = \overline{\overline{A.B}}$$

$$\text{But } \overline{\overline{A}} = A$$

Y=A.B

This equation can be realized using only NAND gate as shown in fig.



3) OR using NAND:-

The Boolean expression for an OR gate is Y=A + B

Taking double inversion,

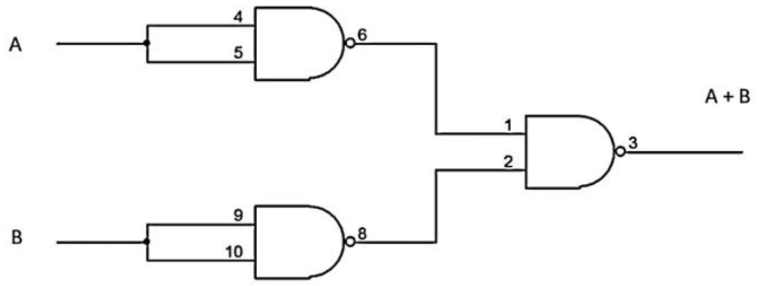
$$Y = \overline{\overline{A+B}}$$

But by DE-Morgan's theorem

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

$$Y = \overline{\overline{A}.\overline{B}}$$

This is required expression for OR gate



4) NOR using NAND:-

The Boolean expression for an NOR gate is $Y = \overline{A + B}$

But by DE-Morgan's theorem

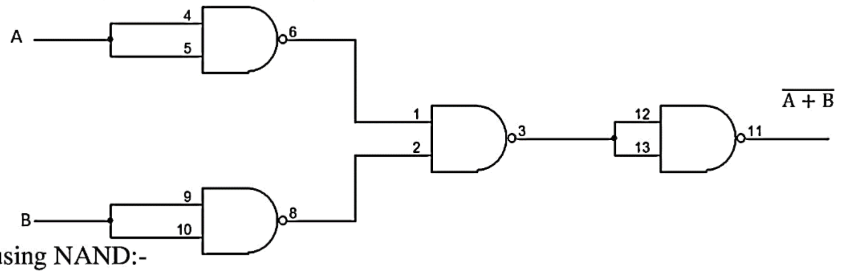
$$\overline{A + B} = \overline{A}.\overline{B}$$

$$Y = \overline{A}.\overline{B}$$

Taking double inversion,

$$Y = \overline{\overline{\overline{A}.\overline{B}}}$$

This is required expression for NOR gate



5) Ex-OR using NAND:-

The expression for Ex- OR gate is

$$Y = A \oplus B$$

$$Y = \overline{A}\,.\,B + A.\overline{B}$$

Taking double inversion

$$Y = \overline{\overline{\overline{A}\,.\,B + A.\overline{B}}}$$

Let $\overline{A}\,.\,B$ =X and $A.\overline{B}$ =Z

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

$$\text{using De} - \text{Morgan'stheorem}$$

$$\overline{X + Z} = \overline{X}.\overline{Z}$$

$$Y = \overline{\overline{X}\,.\,\overline{Z}}$$

$$Y = \overline{\overline{(\overline{A}B)}\,.\,\overline{(A.\overline{B})}}$$

Instruction manual

Experiment 3

Objective: To design a combinational logic system for specified truth table using NAND and NOR Gates.

Apparatus Required: Logic gate kit and wires etc.

Theory:

NAND GATE

NAND gate is universal gate. It can perform all the basic logic function. NAND means NOT AND that is, AND output is NOTed.so NAND gate is combination of an AND gate and a NOT gate. The output is logic 0 level, only when each of its inputs assumes a logic 1 level. For any other combination of inputs, the output is logic 1 level. NAND gate is equivalent to a

bubbled OR gate.

The logic symbol & truth table of two input NAND gate are shown in figure 1.g & 1.h respectively.

With input variables A & B the Boolean expression for output can be written as;

$X = \overline{A.B}$

Logic symbol:

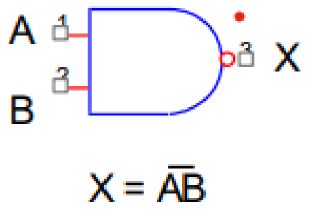


Fig. 1

Truth table:

Input		Output
A	**B**	**X**
0	0	1
0	1	1
1	0	1
1	1	0

Fig. 2

NOR GATE

NOR gate is universal gate. It can perform all the basic logic function. NOR means NOT OR that is, OR output is NOTed.so NOR gate is combination of an OR gate and a NOT gate. The output is logic 1 level, only when each of its inputs assumes a logic 0 level. For any other combination of inputs, the output is logic 0 level. NOR gate is equivalent to a bubbled AND gate. The logic symbol & truth table of two inputs NOR gate are shown in figure 1.i& 1.j respectively. With input variables A & B the Boolean expression for output can be written as;

$X = \overline{A + B}$

Logic symbol: Truth table:

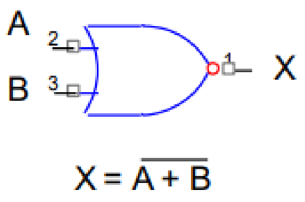


Fig. 3

Input		Output
A	**B**	**X**
0	0	1
0	1	0
1	0	0
1	1	0

Fig. 4

(1) $A\overline{B} + \overline{A}B$

Logical expression circuit using NAND and NOR Gate is shown in fig. (5)

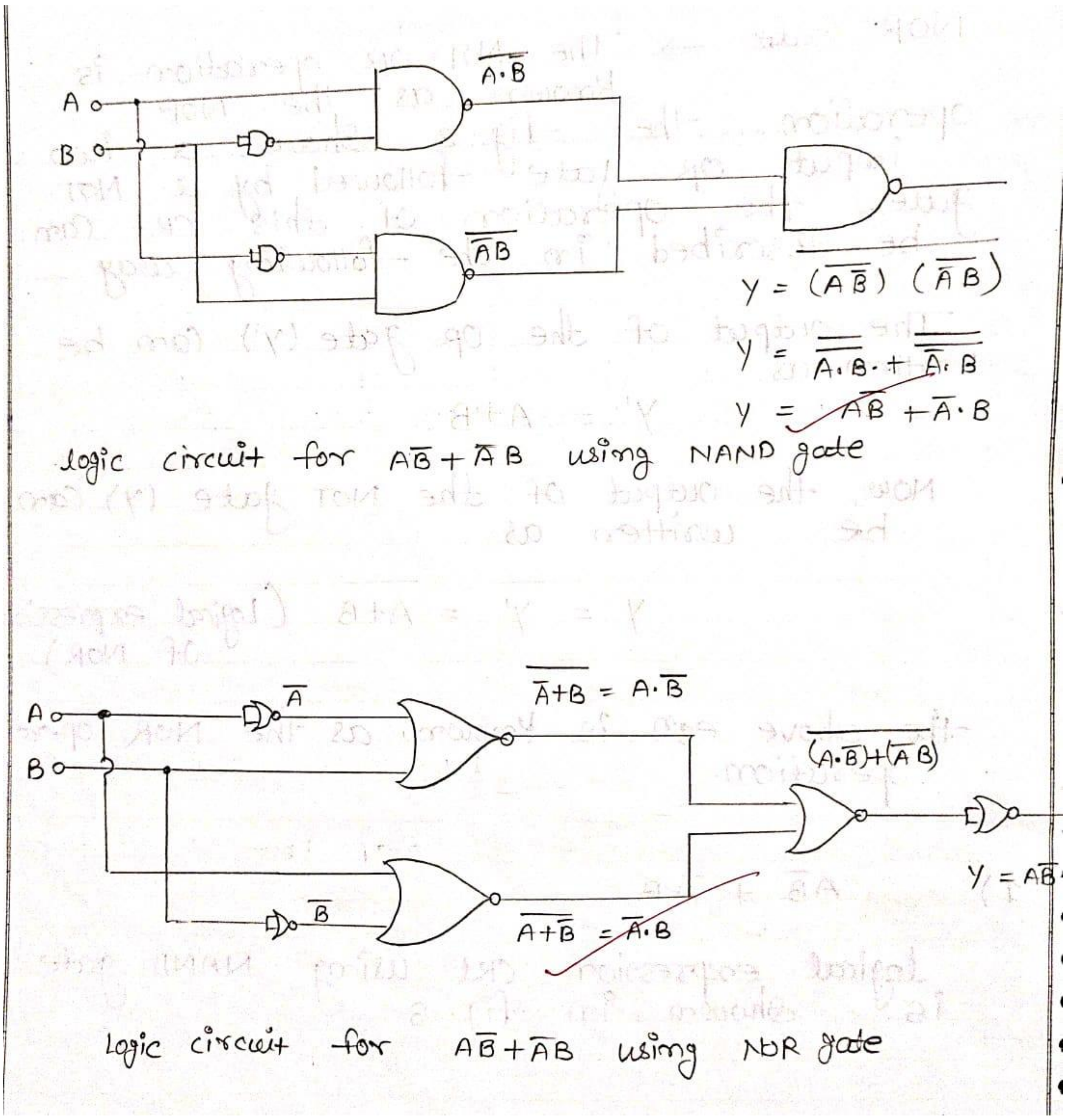
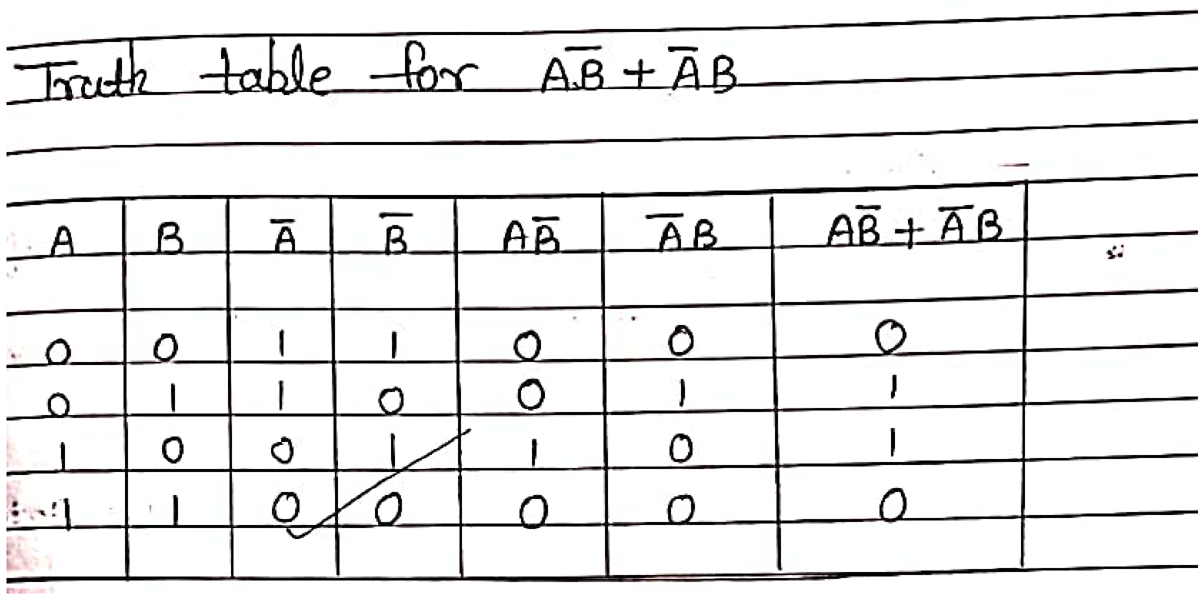

A
B
$\overline{A\cdot\overline{B}}$
$\overline{\overline{A}B}$
$Y = \overline{(A\overline{B})\,(\overline{A}B)}$
$Y = \overline{\overline{A\cdot\overline{B}}} + \overline{\overline{\overline{A}\cdot B}}$
$Y = A\overline{B} + \overline{A}\cdot B$
logic circuit for $A\overline{B}+\overline{A}B$ using NAND gate
A
B
$\overline{A}$
$\overline{\overline{A}+B} = A\cdot\overline{B}$
$\overline{(A\cdot\overline{B})+(\overline{A}B)}$
$Y = A\overline{B}$
$\overline{B}$
$\overline{A+\overline{B}} = \overline{A}\cdot B$
logic circuit for $A\overline{B}+\overline{A}B$ using NOR gate

Fig. (5) Logical expression circuit using NAND and NOR Gate.



Truth table for $A.\bar{B} + \bar{A}B$

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

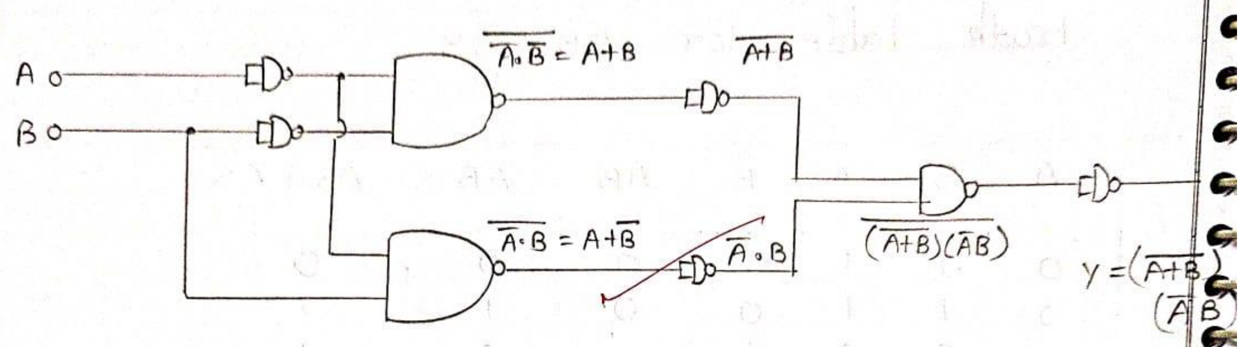
1). when provide both the input as low (or ‘o’) Voltage we get low (or 'o') output.

2). when we provide input. A as low (or ‘o’) and input we B as high (or '1') then we gate high output.

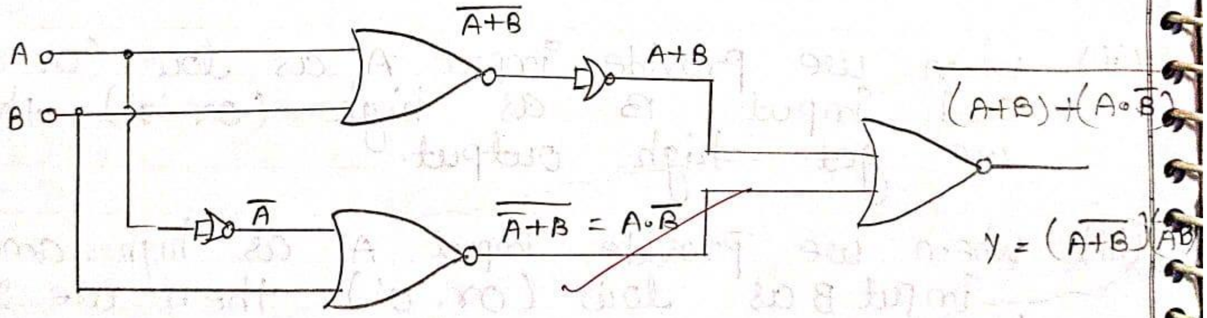
3). when we provide input A as high and input B as low (or 'o') then we get high or ‘1’ output.

4). when we provide input A and B both as low (or ‘o’) then we get low or o' output.

(2) $(\overline{A + B})\ (\bar{A}B)$ Logical expression circuit using NAND and NOR Gate is shown in fig. (6)

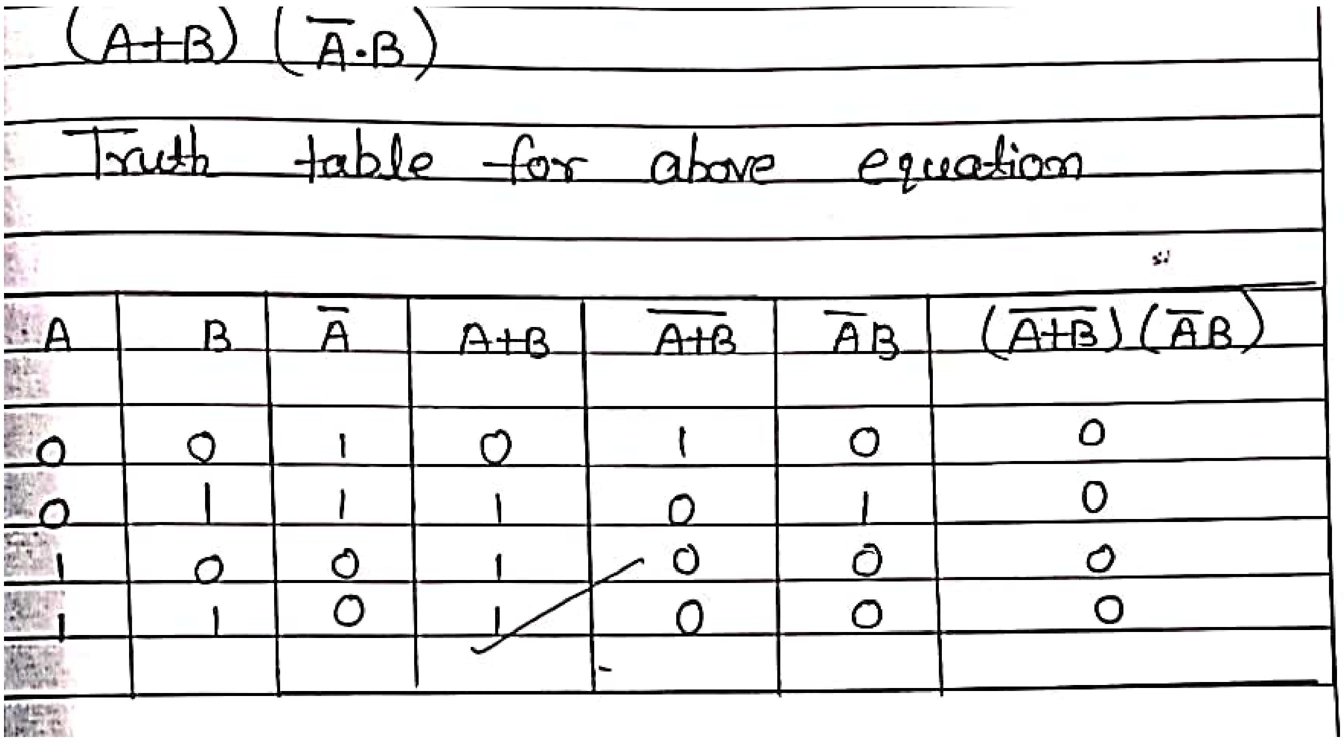

A
B
$\overline{\overline{A}\cdot\overline{B}} = A+B$
$\overline{A+B}$
$\overline{\overline{A}\cdot B} = A+\overline{B}$
$\overline{A}\cdot B$
$\overline{(\overline{A+B})(\overline{A}B)}$
$y = (\overline{A+B})(\overline{A}B)$

Logic circuit $(\overline{A+B})$ $(\overline{A}B)$ using NAND gate


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

Logic ckt $(\overline{A+B})$ $(\overline{A}B)$ using NOR gate

Fig (6) Logical expression circuit using NAND and NOR Gate.



$(A+B)(\bar{A}\cdot B)$

Truth table for above equation

A	B	$\bar{A}$	$A+B$	$\overline{A+B}$	$\bar{A}B$	$(\overline{A+B})(\bar{A}B)$
0	0	1	0	1	0	0
0	1	1	1	0	1	0
1	0	0	1	0	0	0
1	1	0	1	0	0	0

1). When we provide both input A+B as low (or ‘o’) we get output as low or ‘o'.

2). When we provide input A as low and B as high ‘1’ we get output as low.

3). When we provide imput A as high and B law as low (or 'o') we get output as low (or 'o').

4). When we provide both input A+B as high (or '1') we get output as low (or ‘o’).

Result: We designed some combinational circuit using NAND and NOR gate and verified it’s.

Precautions:

1. Make the connection properly.

2. Switch off when changing the input.

3. Take all reading carefuly.

Instruction manual

Experiment- 04

Objective: Introduction to Digital Laboratory Equipments & IC's

Theory:

The Breadboard

The breadboard consists of two terminal strips and two bus strips (often broken in the center). Each bus strip has two rows of contacts. Each of the two rows of contacts are a node. That is, each contact along a row on a bus strip is connected together (inside the breadboard). Bus strips are used primarily for power supply connections, but are also used for any node requiring a large number of connections. Each terminal strip has 60 rows and 5 columns of contacts on each side of the center gap. Each row of 5 contacts is a node.
You will build your circuits on the terminal strips by inserting the leads of circuit components into the contact receptacles and making connections with 22-26 gauge wire. There are wire cutter/strippers and a spool of wire in the lab. It is a good practice to wire+5Vand 0Vpower supply connections to separate bus strips.

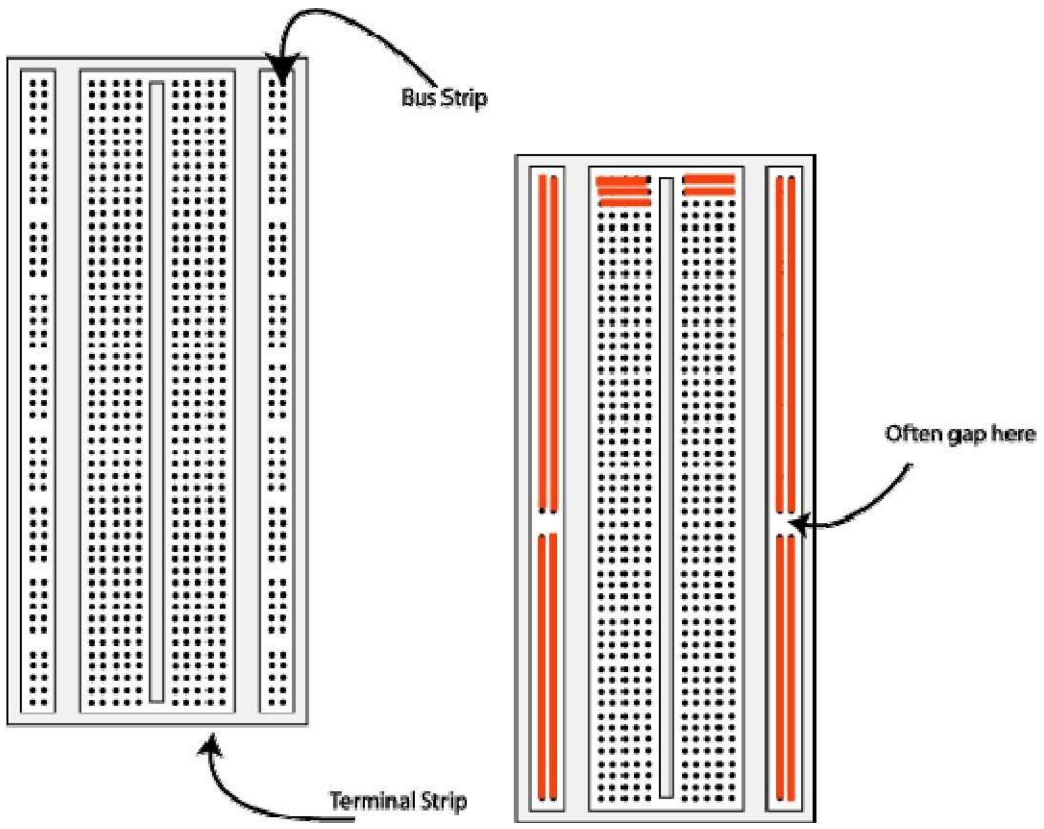


Fig1.The breadboard. The lines indicate connected holes.

The 5V supply **MUST NOT BE EXCEEDED** since this will damage the ICs(Integrated circuits) used during the experiments. Incorrect connection of power to the ICs could result in them exploding or becoming very hot-with the **possible**

serious injury occurring to the people working on the experiment! Ensure that the power supply polarity and all components and connections are correct <u>before</u> switching on power.

<u>Building the Circuit :</u>

Throughout these experiments we will use TTL chips to build circuits. The steps for wiring a circuit should be completed in the order described below:

1 Turn the power (Trainer Kit)off before you build anything!

2 Make sure the power is off before you build anything!

3 Connect the +5 V and ground (GND) leads of the power supply to the power and ground bus strips on your breadboard.

4 Plug the chips you will be using into the breadboard. Point all the chips in the same direction with pin 1 at the upper-left corner. (Pin 1 is often identified by a dot or a notch next to it on the chip package)

5 Connect+5VandGNDpinsofeachchiptothepowerandgroundbusstripsonthebreadboard.

6 Select a connection on your schematic and place a piece of hook-up wire between corresponding pins of the chips on your breadboard. It is better to make the short connections before the longer ones. Mark each connection on your schematic as you go, so as not to try to make the same connection again at a later stage.

7 Get one of your group members to check the connections, **before you turn the power on**.

8 If an error is made and is not spotted before you turn the power on. Turn the power off immediately before you begin to rewire the circuit.

9 At the end of the laboratory session, collect you hook-up wires, chips and all equipment and return them to the demonstrator.

10.Tidy the area that you were working in and leave it in the same condition as it was before you started.

<u>Common Causes of Problems:</u>

1 Not connecting the ground and/or power pins for all chips.

2 Not turning on the power supply before checking the operation of the circuit.

3 Leaving out wires.

4 Plugging wires into the wrong holes.

5 Driving a single gate input with the outputs of two or more gates

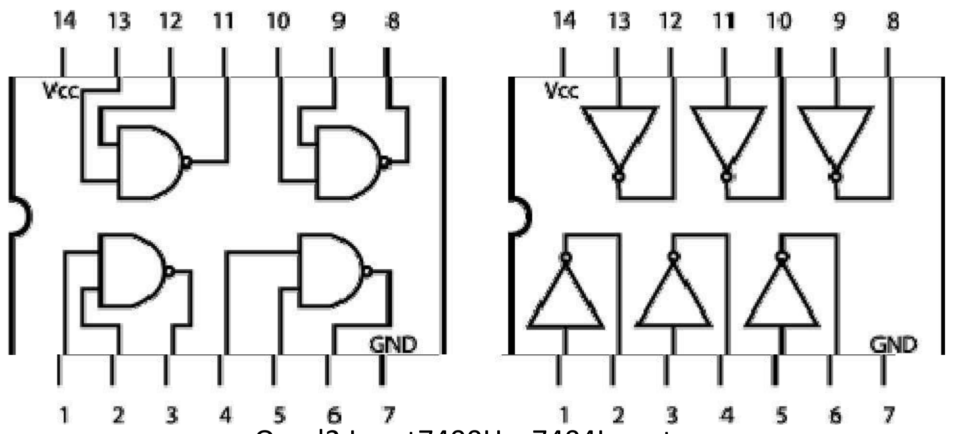
6 Modifying the circuit with the power on.

In all experiments, you will be expected to obtain all instruments , leads, components at the start of the experiment and return them to their proper place.

After you have finished the experiment. Please inform the demonstrator or technician. if you locate faulty equipment If you damage a chip, inform a demonstrator, don't put It back in the box of chips for somebody else to use.

Example Implementation of a Logic Circuit :

Build a circuit to implement the Boolean function F = /(/A./B), please note that the notation **/A** refers to $\overline{A}$. You should use that notation during the write-up of your laboratory experiments.



Quad2 Input7400Hex7404Inverter

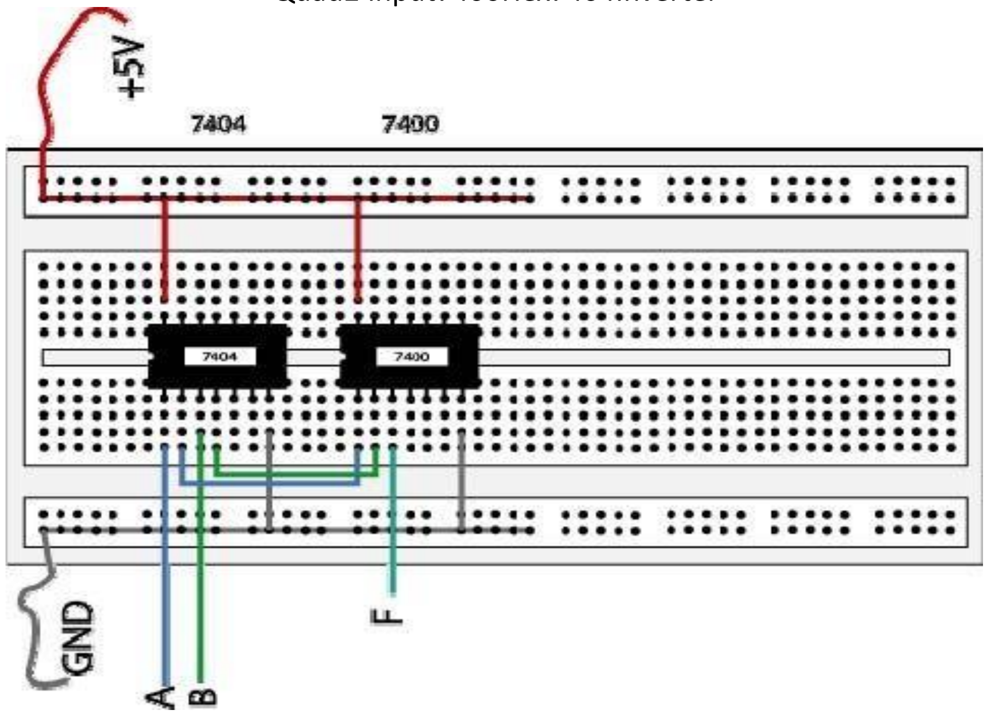
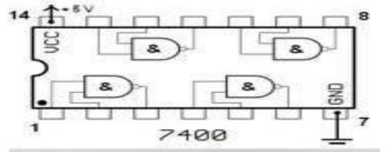


Fig2.The complete designed and connected circuit

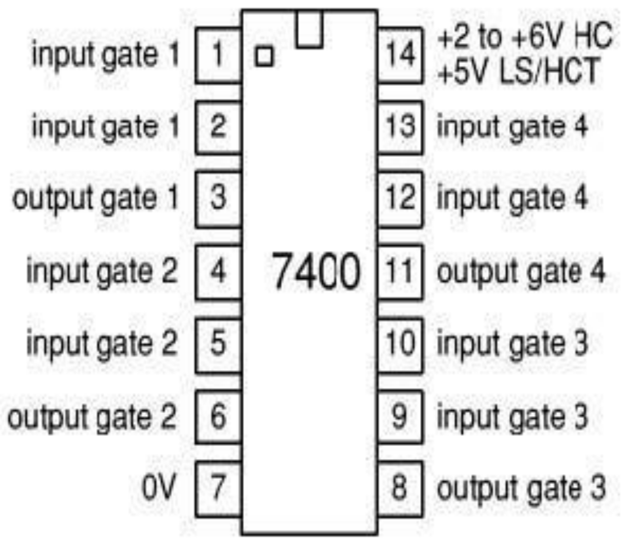
Sometimes the chip manufacturer may denote the first pin by a small indented circle above the first pin of the chip. Place your chips in the same direction, to save confusion at a later stage . Remember that you must connect power to the chips to

Get them to work.

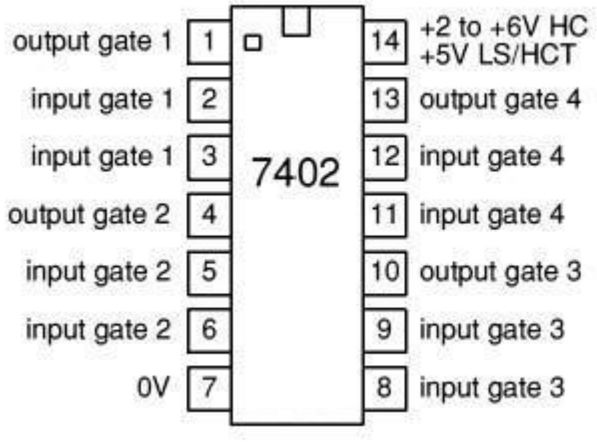
7476	Dual j-k Flip Flop



7400(NAND)



7402(NOR)



Instruction manual

Experiment- 05

Objective: To familiarise and verify the following Boolean algebra theorems and to simplify and realize the expression.

Boolean algebra: Variable, complement, and literal are terms used in Boolean algebra. A variable is a symbol used to represent a logical quantity. Any single variable can have a 1 or a 0 value. The complement is the inverse of a variable and is indicated by a bar over variable (overbar). For example, the complement of the variable A is A. If A = 1, then A = 0. If A = 0, then A = 1. The complement of the variable A is read as "not A" or "A bar." Sometimes a prime symbol rather than an overbar is used to denote the complement of a variable; for example, B' indicates the complement of B. A literal is a variable or the complement of a variable.

Boolean Addition: Recall from part 3 that Boolean addition is equivalent to the OR operation. In Boolean algebra, a sum term is a sum of literals. In logic circuits, a sum term is produced by an OR operation with no AND operations involved. Some examples of sum terms are A + B, A + B, A + B + C, and A + B + C + D. A sum term is equal to 1 when one or more of the literals in the term are 1. A sum term is equal to 0 only if each of the literals is 0. Example Determine the values of A, B, C, and D that make the sum term A + B + C + D equal to 0.

Boolean Multiplication: Also recall from part 3 that Boolean multiplication is equivalent to the AND operation. In Boolean algebra, a product term is the product of literals. In logic circuits, a product term is produced by an AND operation with no OR operations involved. Some examples of product terms are AB, $A\overline{B}$, ABC, and ABCD.

A product term is equal to 1 only if each of the literals in the term is 1. A product term is equal to 0 when one or more of the literals are 0.

Example

Determine the values of A, B, C, and D that make the product term $A\overline{B}C\overline{D}$ equal to 1.

LAWS AND RULES OF BOOLEAN ALGEBRA

■ *Laws of Boolean Algebra*

The basic laws of Boolean algebra-the commutative laws for addition and multiplication, the associative laws for addition and multiplication, and the distributive law-are the same as in ordinary algebra.

Commutative Laws

►The commutative law of addition for two variables is written as

A+B = B+A

This law states that the order in which the variables are ORed makes no difference. Remember, in Boolean algebra as applied to logic circuits, addition and the OR operation are the same. Fig.(4-1) illustrates the commutative law as applied to the OR gate and shows that it doesn't matter to which input each variable is applied. (The symbol ≡ means "equivalent to.").

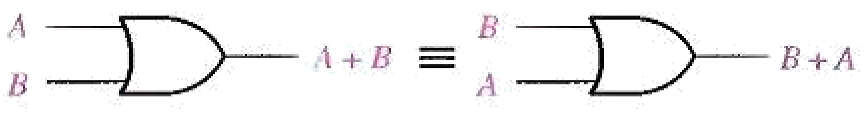


Fig.(4-1) Application of commutative law of addition.

►The commutative law of multiplication for two variables

is A.B = B.A

This law states that the order in which the variables are ANDed makes no difference. Fig.(4-2), il1ustrates this law as applied to the AND gate.

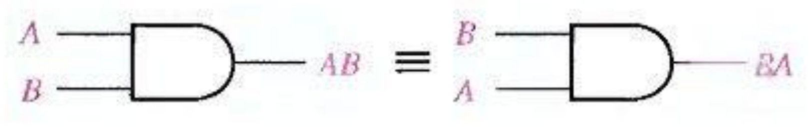


Fig.(4-2) Application of commutative law of multiplication.

Associative Laws :

►The associative law of addition is written as follows for three variables:

A + (B + C) = (A + B) + C

This law states that when ORing more than two variables, the result is the same regardless of the grouping of the variables. Fig.(4-3), illustrates this law as applied to 2-input OR gates.

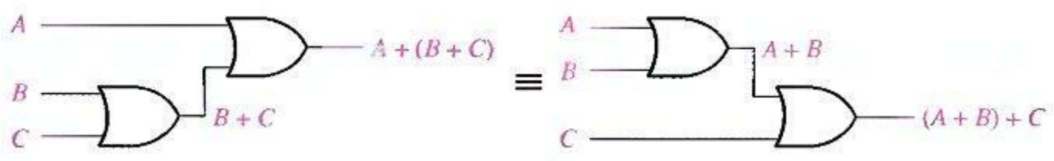


Fig.(4-3) Application of associative law of addition.

►The associative law of multiplication is written as follows for three variables:

A(BC) = (AB)C

This law states that it makes no difference in what order the variables are grouped when ANDing more than two variables. Fig.(4-4) illustrates this law as applied to 2-input AND gates.

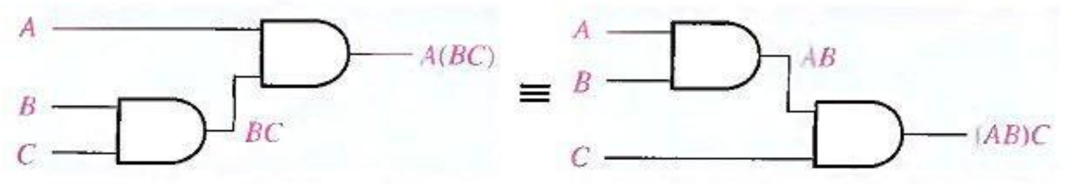


Fig.(4-4) Application of associative law of multiplication.

Distributive Law:

►The distributive law is written for three variables as follows:

$$A(B + C) = AB + AC$$

This law states that ORing two or more variables and then ANDing the result with a single variable is equivalent to ANDing the single variable with each of the two or more variables and then ORing the products. The distributive law also expresses the process of factoring in which the common variable A is factored out of the product terms, for example,

$$AB + AC = A(B + C).$$

Fig.(4-5) illustrates the distributive law in terms of gate implementation.

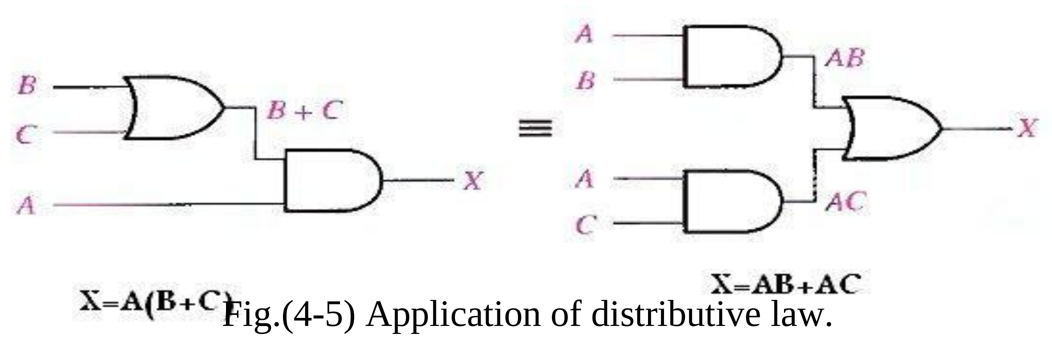
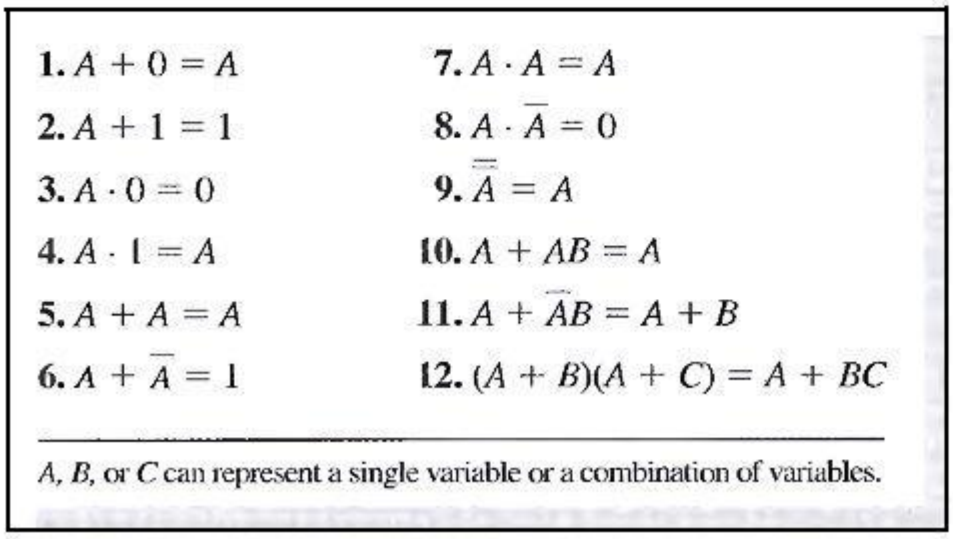


Fig.(4-5) Application of distributive law.

■ ***Rules of Boolean Algebra***

Table 4-1 lists 12 basic rules that are useful in manipulating and simplifying Boolean expressions. Rules 1 through 9 will be viewed in terms of their application to logic gates. Rules 10 through 12 will be derived in terms of the simpler rules and the laws previously discussed.

Table 4-1 Basic rules of Boolean algebra.

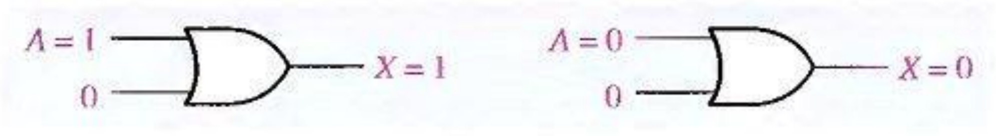


1. $A + 0 = A$	**7.** $A \cdot A = A$
2. $A + 1 = 1$	**8.** $A \cdot \overline{A} = 0$
3. $A \cdot 0 = 0$	**9.** $\overline{\overline{A}} = A$
4. $A \cdot 1 = A$	**10.** $A + AB = A$
5. $A + A = A$	**11.** $A + \overline{A}B = A + B$
6. $A + \overline{A} = 1$	**12.** $(A + B)(A + C) = A + BC$

A, B, or *C* can represent a single variable or a combination of variables.

Rule 1. $A + 0 = A$

A variable ORed with 0 is always equal to the variable. If the input variable A is 1, the output variable X is 1, which is equal to A. If A is 0, the output is 0, which is also equal to A. This rule is illustrated in Fig.(4-6), where the lower input is fixed at 0.



A+0=A

Fig.(4-6)

Rule 2. A + 1 = 1

A variable ORed with 1 is always equal to 1. A 1 on an input to an OR gate produces a 1 on the output, regardless of the value of the variable on the other input. This rule is illustrated in Fig.(4-7), where the lower input is fixed at 1.

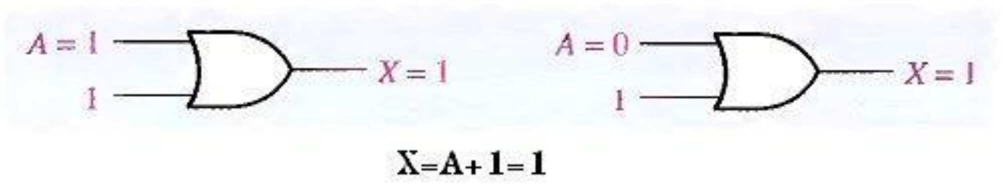


Fig.(4-7)

Rule 3. A . 0 = 0

A variable ANDed with 0 is always equal to 0. Any time one input to an AND gate is 0, the output is 0, regardless of the value of the variable on the other input. This rule is illustrated in Fig.(4-8), where the lower input is fixed at 0.

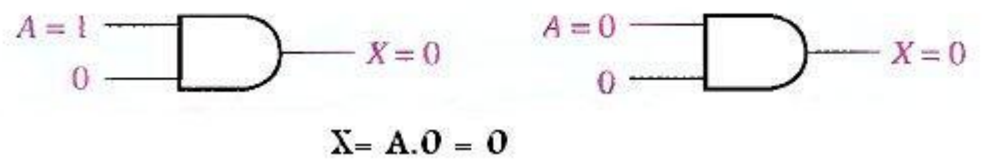


Fig.(4-8)

Rule 4. A . 1 = A

A variable ANDed with 1 is always equal to the variable. If A is 0 the output of the AND gate is 0. If A is 1, the output of the AND gate is 1 because both inputs are now 1s. This rule is shown in Fig.(4-9), where the lower input is fixed at 1.

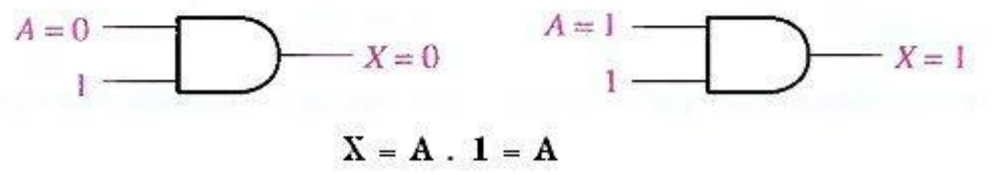


Fig.(4-9)

Rule 5. $A + A = A$

A variable ORed with itself is always equal to the variable. If A is 0, then 0 + 0 = 0; and if A is 1, then 1 + 1 = 1. This is shown in Fig.(4-10), where both inputs are the same variable.

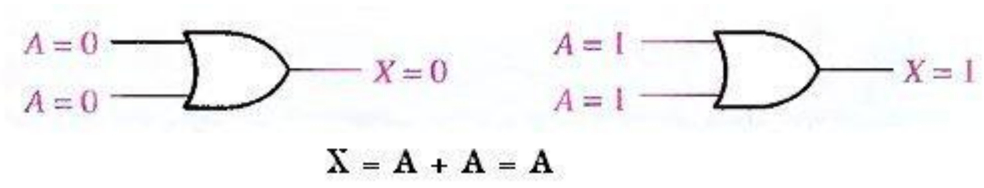


Fig.(4-10)

Rule 6. $A + \overline{A} = 1$

A variable ORed with its complement is always equal to 1. If A is 0, then 0 + $\overline{0}$ = 0 + 1 = 1. If A is l, then 1 + 1 = 1+ 0 = 1. See Fig.(4-11), where one input is the complement of the other.

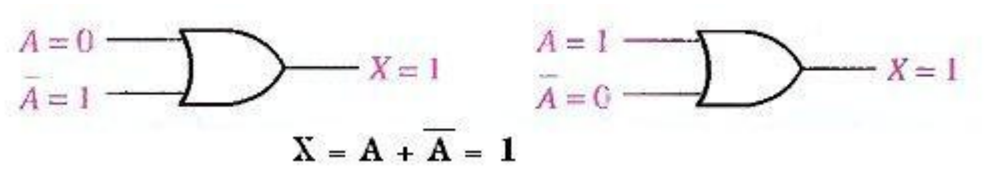


Fig.(4-11)

Rule 7. $A . A = A$

A variable ANDed with itself is always equal to the variable. If A = 0, then 0.0 = 0; and if A = 1. then 1.1 = 1. Fig.(4-12) illustrates this rule.

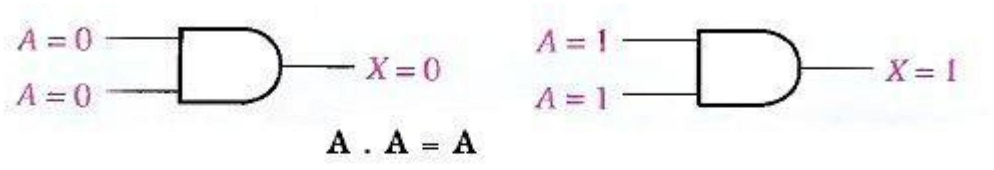


Fig.(4-12)

Rule 8. $A \cdot \overline{A} = 0$

A variable ANDed with its complement is always equal to 0. Either A or $\overline{A}$ will always be 0: and when a 0 is applied to the input of an AND gate. the output will be 0 also. Fig.(4-13) illustrates this rule.

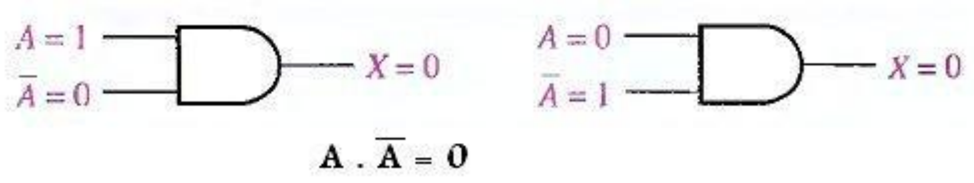
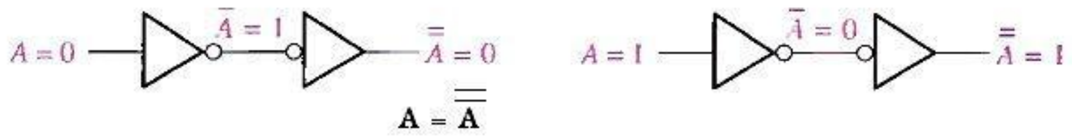


Fig.(4-13)

Rule 9 $A = \overline{\overline{A}}$

The double complement of a variable is always equal to the variable. If you start with the variable A and complement (invert) it once, you get $\overline{A}$. If you then take $\overline{A}$ and complement (invert) it, you get A, which is the original variable. This rule is shown in Fig.(4-14) using inverters.



$A = 0$
$\overline{A} = 1$
$\overline{\overline{A}} = 0$
$A = 1$
$\overline{A} = 0$
$\overline{\overline{A}} = 1$
$A = \overline{\overline{A}}$

Fig.(4-14)

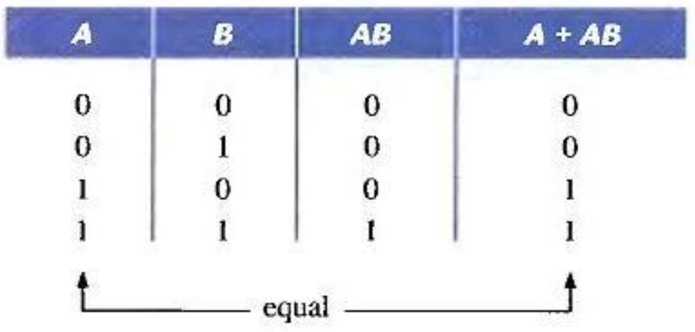
Rule 10. $A + AB = A$

This rule can be proved by applying the distributive law, rule 2, and rule 4 as follows:

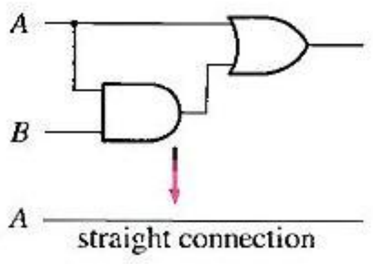
$A + AB = A(1 + B)$	Factoring (distributive law)
$= A \cdot 1$	Rule 2: $(1 + B) = 1$
$= A$	Rule 4: $A \cdot 1 = A$

The proof is shown in Table 4-2, which shows the truth table and the resulting logic circuit simplification.

Table 4-2



A	B	AB	$A + AB$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1



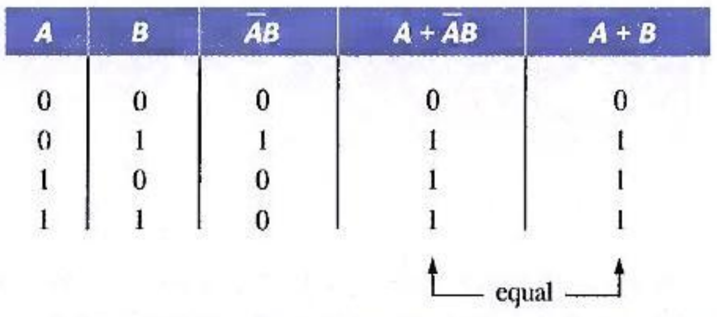
Rule 11. $A + A\overline{B} = A + B$

This rule can be proved as follows:

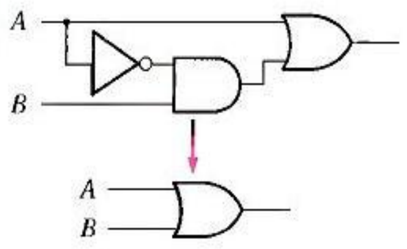
$A + \overline{A}B = (A + AB) + \overline{A}B$	Rule 10: $A = A + AB$
$= (AA + AB) + \overline{A}B$	Rule 7: $A = AA$
$=AA + AB + A\overline{A} + \overline{A}B$	Rule 8: adding $A\overline{A} = 0$
$= (A + \overline{A})(A + B)$	Factoring
$= 1 . (A + B)$	Rule 6: $A + \overline{A} = 1$
$=A + B$	Rule 4: drop the 1

The proof is shown in Table 4-3, which shows the truth table and the resulting logic circuit simplification.

Table 4-3



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



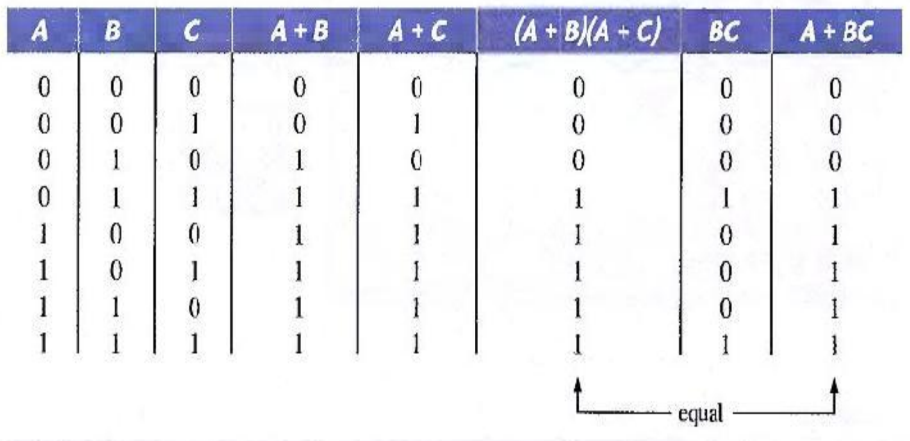
Rule 12. (A + B)(A + C) = A + BC

This rule can be proved as follows:

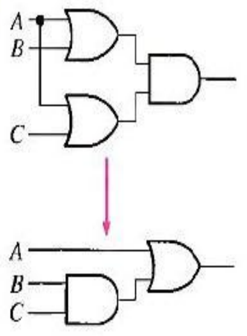
(A + B)(A + C)	= AA + AC + AB + BC	Distributive law
	= A + AC + AB + BC	Rule 7: AA = A
	= A(1 + C) + AB + BC	Rule 2: 1 + C = 1
	= A. 1 + AB + BC	Factoring (distributive law)
	= A(1 + B) + BC	Rule 2: 1 + B = 1
	= A. 1 + BC	Rule 4: A . 1 = A
	= A + BC	

The proof is shown in Table 4-4, which shows the truth table and the resulting logic circuit
simplification.

Table 4-4



A	B	C	A + B	A + C	(A + B)(A + C)	BC	A + BC
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1



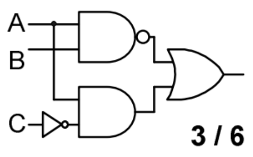
Instruction Manual

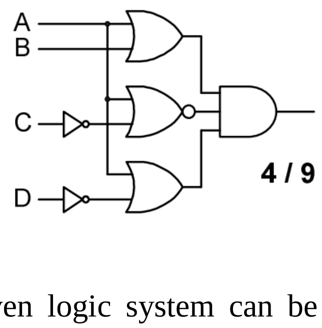
Experiment – 06

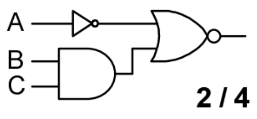
Objective: To minimize a given logic circuit.

Theory:

Logic gates in the standard circuits with transistor-minimum gate equivalents (by taking advantage of NAND/NOR logic) results in the minimized POS and SOP circuits shown in the green boxes. As engineers, one of our primary goals is to implement circuits efficiently. The most efficient circuit can use the fewest number of transistors, or it can operate at the highest speeds, or it can use the least amount of power. Often, these three measures of efficiency cannot all be optimized at the same time, and designers must trade-off circuit size for speed of operation, or speed for power, or power for size, etc. Here, we will define the most efficient circuit as the one that uses the minimum number of transistors, and leave speed and power considerations for later consideration. Because we have chosen the minimum-transistor measure of efficiency, we will look for "minimum" circuits. The best method of determining which of several circuits is the minimum is to count the needed transistors. For now, we will use a simpler method – the minimal circuit will be defined as the one that uses the fewest number of logic gates (or, if two forms use the same number of gates, then the one that uses the fewest number of total inputs to all gates will be considered the simplest). The following examples show circuits with the gate/input number shown below. Inverters are not included in the gate or input count, because often, they are absorbed into the logic gates themselves.







A minimal logic equation for a given logic system can be obtained by eliminating all non-essential or redundant inputs. Any input that can be removed from the equation without changing the input/output relationship is redundant. To find minimal equations, all redundant inputs must be identified and removed. In the truth table above, note the SOP terms generated by rows 1 and 3. The A input is ‘0’ in both rows, and the C input is ‘1’ in both rows, but the B input is ‘0’ in one row and ‘1’ in the other. Thus, for these two rows, the output is a ‘1’ whether B is a ‘0’ or ‘1’ and B is therefore redundant.

The goal in “minimizing” logic systems is to find the simplest form by identifying and removing all redundant inputs. For a logic function of N inputs, there are 22 N logic functions, and for each of these functions, there exists a minimum SOP form and a minimum POS form. The SOP form may be more minimal than the POS form, or the POS form may be more minimal, or they may be equivalent (i.e., they may both require the same number of logic gates and inputs). In general, it is difficult to identify the minimum form by simply staring at a truth table. Several methods have evolved to assist with the minimization process, including the application of Boolean algebra, the use of logic graphs, and the use of searching algorithms. Although any of these methods can be employed using pen and paper, it is far easier (and more productive) to implement searching algorithms on a computer.

Boolean algebra:

Boolean algebra is perhaps the oldest method used to minimize logic equations. It provides a formal algebraic system that can be used to manipulate logic equations in an attempt to find more minimal equations. It is a proper algebraic system, with three set elements {‘0’, ‘1’, and ‘A’} (where ‘A’ is any variable that can assume the values ‘0’ or ‘1’), two binary operations (and or intersection, or or union), and one unary operation (inversion or complementation). Operations between sets are closed under the three operations. The basic laws governing and, or, and inversion operations are easily derived from the logic truth tables for those operations. The associative, commutative, and distributive laws can be directly demonstrated using truth tables. Only the distributive law truth table is shown in the truth table below, with colors used to highlight the columns that show the equivalency of both sides of the distributive law equations. Truth tables to demonstrate the simpler associative and commutative laws are not shown, but they can be easily derived.

AND Operation		**OR Operation**		**INV Operation**	
Truth table	**Laws**	**Truth table**	**Laws**	**Truth table**	**Laws**
0 · 0 = 0	A · 0 = 0	0 + 0 = 0	A + 0 = A	0' = 1	A'' = A
1 · 0 = 0	A · 1 = A	1 + 0 = 1	A + 1 = 1	1' = 0	
0 · 1 = 0	A · A = A	0 + 1 = 1	A + A = A		
1 · 1 = 1	A · A' = 0	1 + 1 = 1	A + A' = 1		

Associative Laws	**Commutative Laws**	**Distributive Laws**
(A·B)·C = A·(B·C) = A·B·C (A+B)+C = A+(B+C) = A+B+C	A·B·C = B·A·C = ... A+B+C = B+C+A = ...	A·(B+C) = (A·B) + (A·C) A+(B·C) = (A+B) · (A+C)

Truth table to verify distributive laws												
A	**B**	**C**	**A+B**	**B+C**	**A+C**	**A.B**	**B.C**	**A.C**	**A.(B+C)**	**(A.B)+(A.C)**	**A+(B.C)**	**(A+B).(A+C)**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**1**
1	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**
1	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**
1	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**1**
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

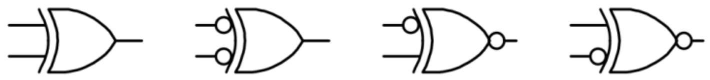
AND’ing operations take precedence over OR’ing operations. Parenthesis can be used to eliminate any possible confusion. Thus, the following two sets of equations show equivalent logic equations.

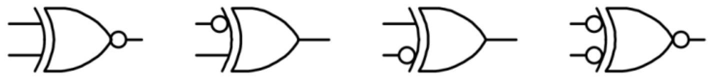
A·B+C = (A·B) + C   A+B·C = A + (B·C)

DeMorgan’s Law provides a formal algebraic statement for the property observed in defining the conjugate gate symbols: the same logic circuit can be interpreted as implementing either an AND or an OR functions, depending how the input and output voltage levels are interpreted. DeMorgan’s law, which is applicable to logic systems with any number of inputs, states

(A·B)’ = A’ + B’ (nand form) and

(A+B)’ = A’·B’ (nor form).


XOR Conjugates

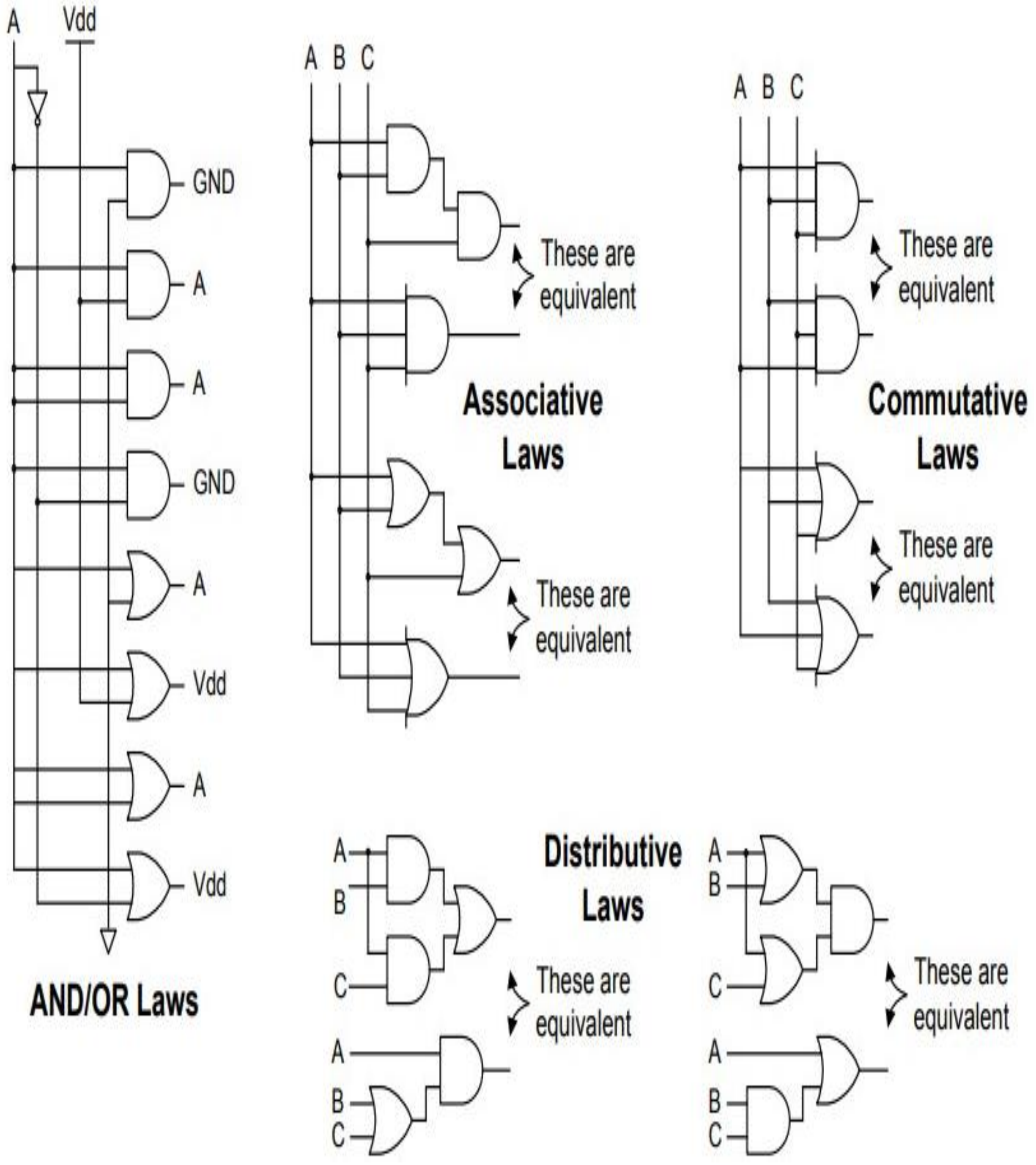

XNOR Conjugates

The laws of Boolean algebra generally hold for XOR functions as well, except that DeMorgan’s law takes a different form. Recall from the previous module that the XOR function output is asserted whenever an odd number of inputs are asserted, and that the XNOR function output is asserted whenever an even number of inputs are asserted. Thus, inverting a single input to an XOR function, or inverting its output, yields the XNOR function. Likewise, inverting a single input to an XNOR function, or inverting its output, yields the XOR function. Inverting an input together with the output, or inverting two inputs, changes an XOR function to XNOR, and vice-versa. These observations lead to a version of DeMorgan’s Laws that hold for XOR functions of any number of inputs:

F = A xnor B xnor C ⇔ F <= (A ⊕ B ⊕ C)’ ⇔ F <= A’ ⊕ B ⊕ C ⇔ F <= (A’ ⊕ B’ ⊕ C)’ etc.
F = A xor B xor C ⇔ F <= A ⊕ B ⊕ C ⇔ F <= A’ ⊕ B’ ⊕ C ⇔ F <= (A ⊕ B’ ⊕ C)’ etc.

Note that a single input inversion can be moved to any other signal in a multi-input XOR circuit without changing the logical result. Note also that any signal inversion can be replaced with a non-inverted signal and an XNOR function. These properties will be useful in later work.

The circuits below also serve illustrate the laws of Boolean Algebra.



The following examples illustrate the use of Boolean Algebra to find simpler logic equations.

$F = A\cdot B\cdot C + A\cdot B\cdot\overline{C} + \overline{A}\cdot B\cdot C + \overline{A}\cdot B$	
$F = A\cdot B\cdot(C+\overline{C}) + \overline{A}\cdot B\cdot(C+1)$	Factoring
$F = A\cdot B\cdot(1) + \overline{A}\cdot B\cdot(1)$	OR law
$F = A\cdot B + \overline{A}\cdot B$	AND law
$F = B\cdot(A+\overline{A})$	Factoring
$F = B\cdot(1)$	OR law
$F = B$	AND law

$F = (A+B+C)\cdot(A+B+\overline{C})\cdot(A+\overline{C})$	
$F = (A+B+C)\cdot(A+\overline{C})\cdot(B+1)$	Factoring
$F = (A+B+C)\cdot(A+\overline{C})\cdot(1)$	OR law
$F = (A+B+C)\cdot(A+\overline{C})$	AND law
$F = A+((B+C)\cdot(\overline{C}))$	Factoring
$F = A+(B\cdot\overline{C}+C\cdot\overline{C})$	Distributive
$F = A+(B\cdot\overline{C}+0)$	AND law
$F = A+(B\cdot\overline{C})$	OR law

$F = \overline{A\cdot B\cdot C} + \overline{A}\cdot B\cdot C + \overline{A\cdot C}$	
$F = (\overline{A}+\overline{B}+\overline{C}) + \overline{A}\cdot B\cdot C + (\overline{A}+\overline{C})$	DeMorgan's
$F = \overline{A} + \overline{A} + (\overline{A}\cdot B\cdot C) + \overline{B} + \overline{C} + \overline{C}$	Commutative
$F = \overline{A}\cdot(1+1+B\cdot C) + \overline{B} + \overline{C}$	Factoring
$F = \overline{A}\cdot(1) + \overline{B} + \overline{C}$	OR law
$F = \overline{A} + \overline{B} + \overline{C}$	AND law

$F = (A \oplus B) + (A \oplus \overline{B})$	
$F = \overline{A}\cdot B + A\cdot\overline{B} + \overline{A}\cdot\overline{B} + A\cdot B$	XOR expansion
$F = \overline{A}\cdot B + \overline{A}\cdot\overline{B} + A\cdot B + A\cdot\overline{B}$	Commutative
$F = \overline{A}\cdot(B+\overline{B}) + A\cdot(B+\overline{B})$	Factoring
$F = \overline{A}\cdot(1) + A\cdot(1)$	OR law
$F = \overline{A} + A$	AND law
$F = 1$	

$F = (\overline{A \oplus B}) + A\cdot B\cdot C + \overline{A\cdot B}$	
$F = \overline{A}\cdot\overline{B} + A\cdot B + A\cdot B\cdot C + (\overline{A}+\overline{B})$	DeMorgan's
$F = \overline{A}\cdot\overline{B} + \overline{A} + \overline{B} + A\cdot B + A\cdot B\cdot C$	Commutative
$F = \overline{A}\cdot(B+1) + \overline{B} + A\cdot B\cdot(1+C)$	Factoring
$F = \overline{A} + \overline{B} + A\cdot B$	OR law
$F = \overline{A} + (\overline{B}+A)\cdot(\overline{B}+B)$	Factoring
$F = \overline{A} + (\overline{B}+A)\cdot(1)$	OR law
$F = \overline{A} + \overline{B} + A$	AND law
$F = 1$	OR law

$F = (\overline{\overline{A}+\overline{B}}) + (\overline{\overline{A}+B}) + (\overline{A+\overline{B}})$	
$F = (\overline{\overline{A}}\cdot\overline{\overline{B}}) + (\overline{A}\cdot\overline{B}) + (\overline{A}\cdot B)$	DeMorgan's
$F = A\cdot B + \overline{A}\cdot\overline{B} + \overline{A}\cdot B$	NOT law
$F = A\cdot B + \overline{A}\cdot(\overline{B}+B)$	Factoring
$F = A\cdot B + \overline{A}\cdot(1)$	OR law
$F = A\cdot B + \overline{A}$	AND law
$F = (A+\overline{A})\cdot(B+\overline{A})$	Factoring
$F = (1)\cdot(B+\overline{A})$	OR law
$F = \overline{A}+B$	AND/Commutative

$F = A + \overline{A}\cdot B$	$= A+B$
$F = (A+\overline{A})\cdot(A+B)$	Factoring
$F = (1)\cdot(A+B)$	OR law
$F = A+B$	AND law

$F = A \cdot (\overline{A}+B)$	$= A\cdot B$
$F = (A\cdot\overline{A})+(A\cdot B)$	Distributive
$F = (0)+(A\cdot B)$	AND law
$F = A\cdot B$	OR law

$F = A\cdot\overline{B} + \overline{B}\cdot C + \overline{A}\cdot C$	$= A\cdot\overline{B} + A\cdot\overline{C}$
$F = A\cdot\overline{B} + \overline{B}\cdot C\cdot 1 + \overline{A}\cdot C$	AND law
$F = A\cdot\overline{B} + \overline{B}\cdot C\cdot(A+\overline{A}) + \overline{A}\cdot C$	OR law
$F = A\cdot\overline{B} + A\cdot\overline{B}\cdot C + \overline{A}\cdot\overline{B}\cdot C + \overline{A}\cdot C$	Distributive
$F = A\cdot\overline{B}\cdot(1+C) + \overline{A}\cdot C\cdot(\overline{B}+1)$	Factoring
$F = A\cdot\overline{B}\cdot(1) + A\cdot\overline{C}\cdot(1)$	OR law
$F = A\cdot\overline{B} + A\cdot\overline{C}$	AND law

The last two examples on the left (with the blue boxes) shows relationships that are sometimes called the “absorptive” laws, and the example on the right (with the green box) is often called the “consensus” law. The so-called absorptive laws are easily demonstrated with other laws, so it is not necessary or even convenient to use these relationships as laws – particularly because different forms of equations can make it difficult to identify when the law might apply. The consensus law is also easily derived, if the “trick” of AND’ing a ‘1’ into the equation, and then expanding that AND into an OR relationship is used (this trick is perfectly acceptable, if not entirely obvious).

Instruction Manual

Experiment – 07

Objective: To study the half adder, full adder, half Subtractor and full Subtractor.

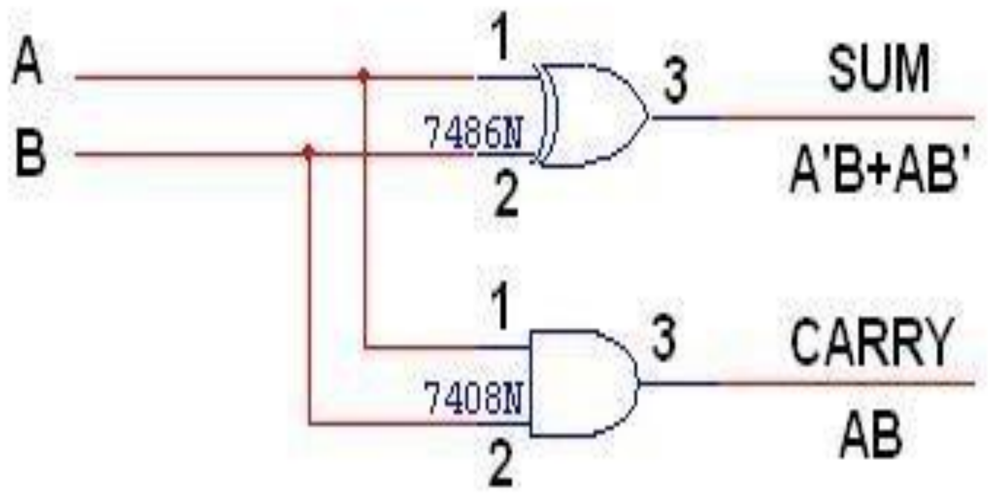
Apparatus Required – DIT board and its power supply.

Theory:

Half Adder

A half adder has two inputs for the two bits to be added and two Outputs one from the sum 'S' and other from the carry 'C' into the higher adder positive. Above circuit is called as a carry signal from the addition of the less significant bits sum from the X-OR Gate the carry out from the AND gate.

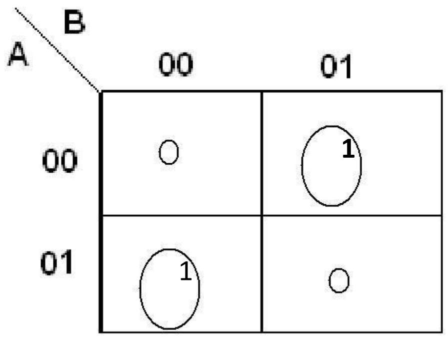
Logic Diagram of half adder:



Truth Table:

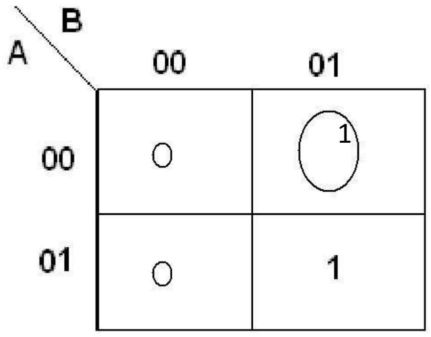
A	B	CARRY	SUM
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

K-Map for SUM:



SUM = A'B + AB'

K-Map for CARRY:

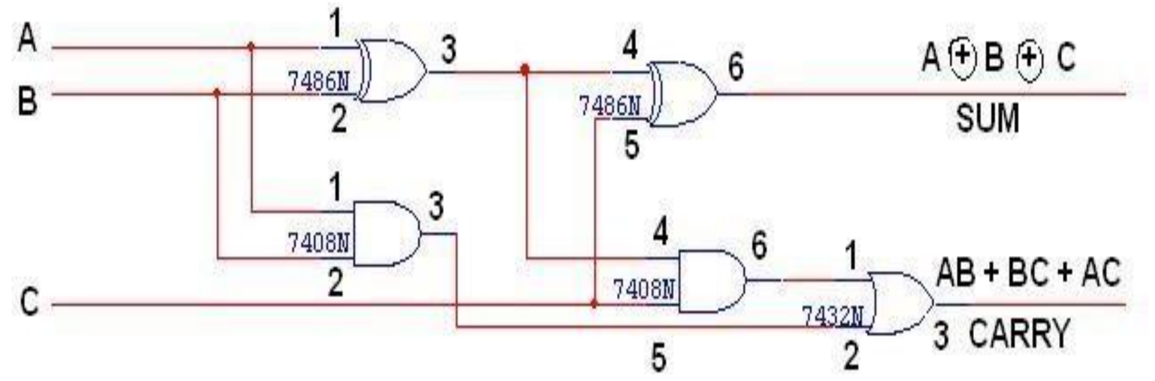


CARRY = AB

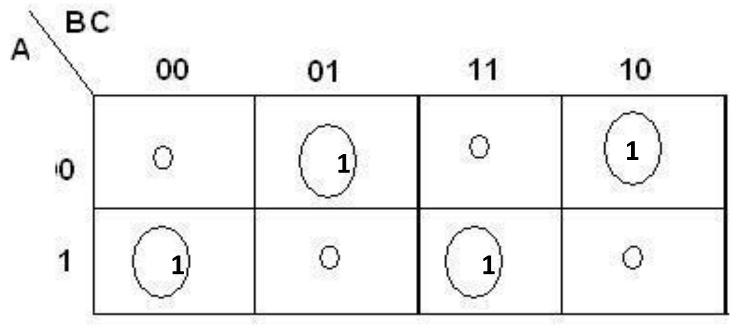
Full Adder:

A full adder is a combinational circuit that forms the arithmetic sum of input; it consists of three inputs and two outputs. A full adder is useful to add three bits at a time but a half adder can't do so. In full adder sum output will be taken from X-OR Gate, carry output will be taken from OR Gate.

Logic Diagram of half adder:



K-Map for SUM

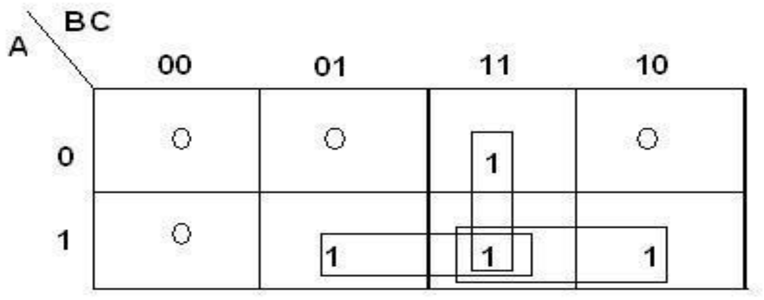


SUM = A’B’C + A’BC’ + ABC’ + ABC

Truth Table:

A	B	C	CARRY	SUM
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

K-Map for CARRY

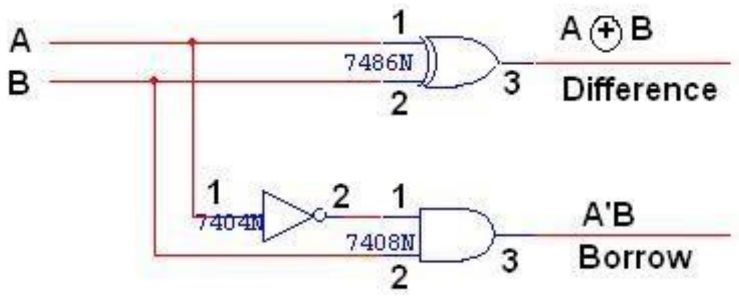


CARRY = AB + BC + AC

Half Subtractor:

The half subtractor is constructed using X-OR and AND Gate. The half substractor has two input and two outputs. The outputs are difference and borrow. The difference can be applied using X-OR Gate, borrow output can be implemented using an AND Gate and an inverter.

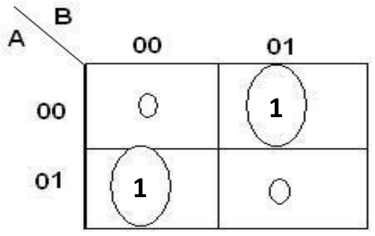
Logic Diagram of half subtractor:

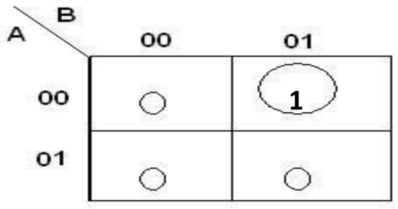


Truth Table:

A	B	BORROW	DIFFERENCE
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

K-Map for DIFFERENCE **K-Map for BORROW**



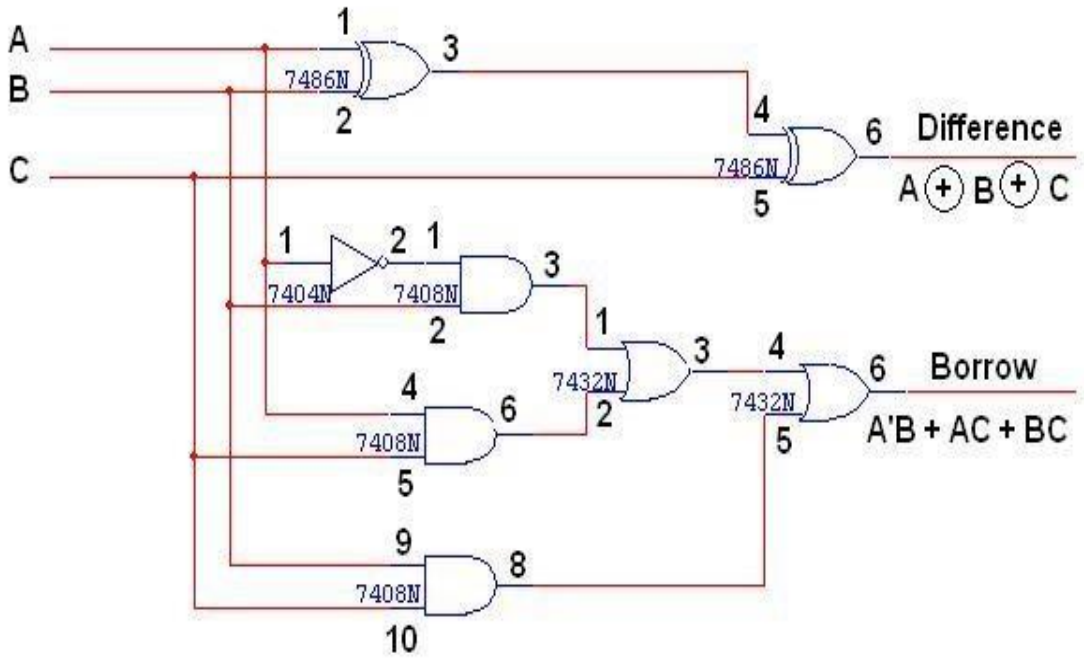


DIFFERENCE = A’B + AB’ **BORROW = A’B**

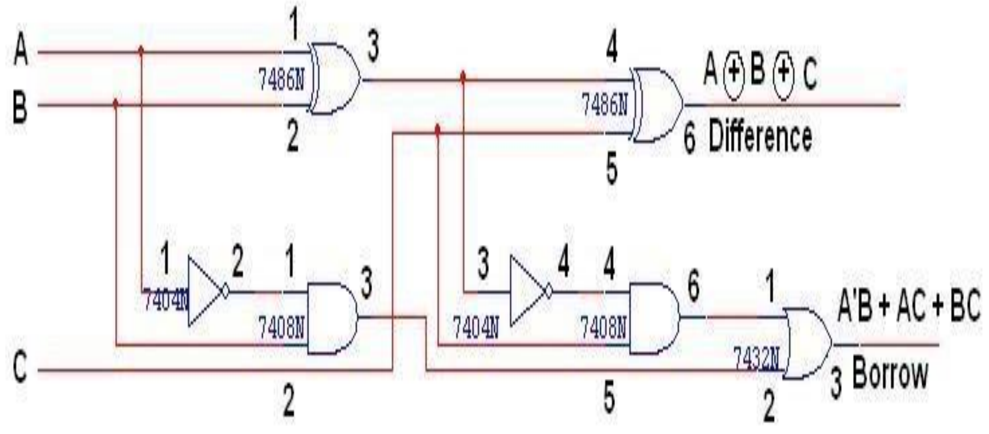
Full Subtractor:

The full subtractor is a combination of X-OR, AND, OR, NOT Gates. In a full subtractor the logic circuit should have three inputs and two outputs. The two half subtractor put together gives a full subtractor. The expression AB assembles the borrow output of the half subtractor and the second term is the inverted difference output of first XOR.

Logic Diagram of Full Subtractor:



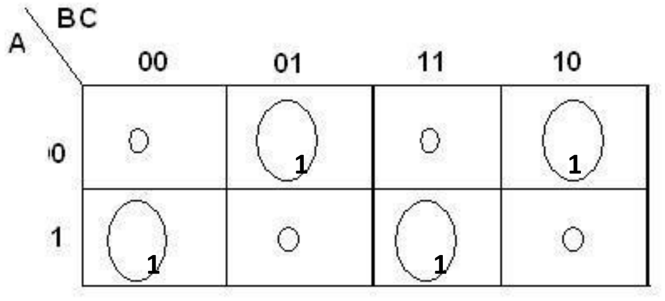
FULL SUBTRACTOR USING TWO HALF SUBTRACTOR

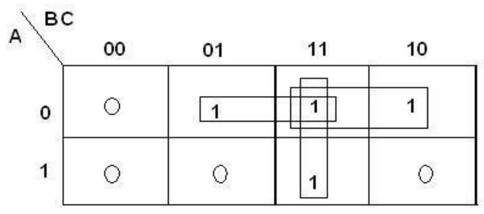


Truth Table:

A	B	C	BORROW	DIFFERENCE
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-Map for Difference **K-Map for Borrow**





Difference = A'B'C + A'BC' + AB'C' + ABC **Borrow = A'B + BC + A'C**

PROCEDURE:

B) HALF ADDER:

1. Connect A & B I/P of Half Adder to switches from input switches section.
2. Connect SUM O/P of Half Adder to L1 & CARRY O/P of Half Adder to L2 in O/p SECTION.
3. Switch on the power supply of the Kit.
4. Provide proper inputs to half adder using switches as per truth table of half adder shown above.
5. Observe the O/P of Half Adder on LEDs.
6. Verify the functionality of half Adder as per truth table & Note it down.

C) FULL ADDER:

1. Connect A, B and C I/P of Full adder to switches from input switches section. Connect SUM & CARRY O/P of Full Adder to leds from O/P LED section.
2. Switch ON the power supply of the Kit.
3. Provide proper inputs to Full adder using switches as per truth table of Full adder shown above.
4. Observe the O/P of Full Adder on LEDs.
5. Verify the functionality of Full Adder as per truth table & Note it down**.**

D) HALF SUBTRACTOR:

1. Connect A & B I/P of Half Subtractor to switches from input switches section. Connect DIF & BOR O/P of Half subtractor to leds in O/P LED section.
2. Switch on the power supply of the Kit.
3. Provide proper inputs to half subtractor using switches as per truth table of half adder shown above.
4. Observe the O/P of Half subtractor on LEDs.
5. Verify the functionality of half subtractor as per truth table & Note it down.

E) FULL SUBTRACTOR:

1) Connect A, B & C I/P of Full Subtractor to switches from input switches section.

2) Connect DIF & BOR O/P of Full Subtractor to leds from O/P SECTION.

3) Switch ON the power supply of the Kit.

4) Provide proper inputs to Full Subtractor using switches as per truth table of Full Subtractor shown above.

5) Observe the O/P of Full Subtractor on LEDs.

6) Verify the functionality of Full Subtractor as per truth table & Note it down.

Instruction Manual

Experiment -8

Objective: To build Flip-Flop (RS, Clocked RS, D-Type and JK) circuits using NAND gates.

Flip Flops

A digital computer needs devices which can store information. A flip flop is a binary storage device. It can store binary bit either 0 or 1. It has two stable states HIGH and LOW i.e. 1 and 0. It has the property to remain in one state indefinitely until it is directed by an input signal to switch over to the other state. It is also called bistable multivibrator.

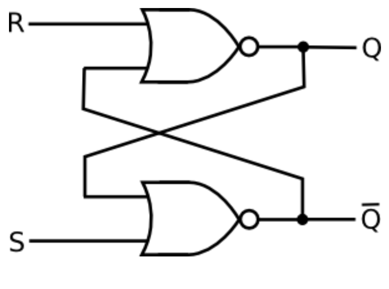
The basic formation of flip flop is to store data. They can be used to keep a record or what value of variable (input, output or intermediate). Flip flop are also used to exercise control over the functionality of a digital circuit i.e. change the operation of a circuit depending on the state of one or more flip flops. These devices are mainly used in situations which require one or more of these three.

Operations, storage and sequencing.

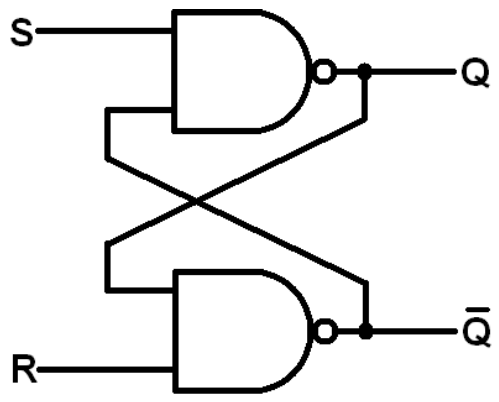
Latch Flip Flop

The R-S (Reset Set) flip flop is the simplest flip flop of all and easiest to understand. It is basically a device which has two outputs one output being the inverse or complement of the other and two inputs. A pulse on one of the inputs to take on a particular logical state. The outputs will then remain in this state until a similar pulse is applied to the other input. The two inputs are called the Set and Reset input (sometimes called the preset and clear inputs).

Such flip flop can be made simply by cross coupling two inverting gates either NAND or NOR gate could be used Figure 1(a) shows on RS flip flop using NAND gate and Figure 1(b) shows the same circuit using NOR gate.



(a) Latch Flip Flops NOR Gate.



(b) RS Latch Flip Flop NAND Gate.

To describe the circuit of Figure 1(a), assume that initially both R and S are at the logic 1 state and that output is at the logic 0 state.

Now, if Q=0 and R=1 then these are the states of inputs of gate B, therefore the outputs of gate B is at 1 ((making it the inverse of Q i.e. 0). The output of gate B is connected to an input of gate A so if S=1, both inputs of gate A are at the logic 1 state. This means that the output of gate A must be 0 (as was originally specified). In other words, the 0 state at Q is continuously disabling gate B so that any change in R has no effect. Also the 1 state at Q' is continuously enabling gate A so that any change S will be transmitted through to Q. The above conditions constitute one of the stable states of the device referred to as the Reset state since Q = 0.

Now suppose that the R-S flip flop in the Reset state, the S input goes to 0. The output of gate A i.e. Q will go to 1 and with Q=1 and R=1, the output of gates B (Q') will go to 0 with Q' now 0 gate A is disabled keeping Q at 1. Consequently, when S returns to the 1 state it has no effect on the flip flop whereas a change in R will cause a change in the output of gate B. The above conditions constitute the other stable state of the device, called the Set state since Q=1 Note that the change of the state of S from 1 to 0 has caused the flip flop to change from the Reset state to the Set state.

There is another input condition which has not yet been considered. That is when both the R and S inputs are taken to the logic state 0. When this happens both Q and Q' will be forced to 1 and will remain so far as long as R and S are kept at 0. However when both inputs return to 1 there is no way of knowing whether the flip flop will latch in the Reset state or the Set state. The condition is said to be indeterminate because of this indeterminate state great care must be taken when using R-S flip flop to ensure that both inputs are not instructed simultaneously.

Table 1: The truth table for the NAND R-S flip-flop.

Initial Conditions	Inputs (Pulsed)		Final Output
Q	S	R	Q
1	0	0	Indeterminate
1	0	1	1
1	1	0	0
1	1	1	1
0	0	0	Indeterminate
0	0	1	1
0	1	0	0
0	1	1	0

Table 2: Simplified NAND R-S Flip-Flop Truth Table.

S	R	Q
0	0	Indeterminate
0	1	1 (Set)
1	0	0 (Reset)
1	1	No Change

When NOR gate are used the R and S inputs are transposed compared with the NAND version. Also the stable state when R and S are both 0. A change of state is effected by pulsing the appropriate input to the 1 state. The indeterminate state is now when both R and S are simultaneously at logic 1. Table 3 shows this operation.

Table 3: NOR Gate R-S Flip-Flop Truth Table.

S	R	Q
0	0	No Change
0	1	0 (Reset)
1	0	1 (Set)
1	1	Indeterminate

Clocked R-S Flip Flop

The RS latch flip flop required the direct input but no clock. It is very use full to add clock to control precisely the time at which the flip flop changes the state of its output.

In the clocked R-S flip flop the appropriate levels applied to their inputs are blocked till the receipt of a pulse from another source called clock. The flip flop changes state only when clock pulse is applied depending upon the inputs. The basic circuit is shown in Figure 2. This circuit is formed by adding two AND gates at inputs to the R-S flip flop. In addition to control inputs Set (S) and Reset (R), there is a clock input (C) also.

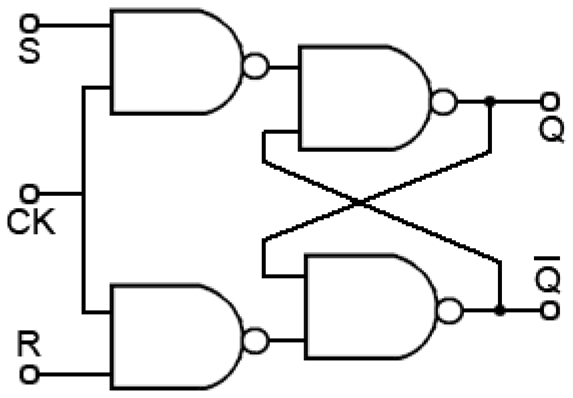


Figure 2: Clocked R-S flip flop circuit diagram.

Table 4: The truth table for the Clocked R-S flip-flop.

Initial Conditions	Inputs (Pulsed)	Final Output	Comment

Q	S	R	Q (t+1)	No Change
1	0	0	0	No Change
1	0	1	0	Clear Q
1	1	0	1	Set Q
1	1	1	???	Indeterminate
0	0	0	1	No Change
0	0	1	0	Clear Q
0	1	0	1	Set Q
0	1	1	???	Indeterminate

The excitation table for R-S flip flop is very simply derived as given below.

Table 5: Excitation table for R-S Flip Flop.

S	R	Q
0	0	No Change
0	1	0 (Reset)
1	0	1 (Set)
1	1	Indeterminate

D Flip Flop

A D type (Data or delay flip flop) has a single data input in addition to the clock input as shown in Figure 3.

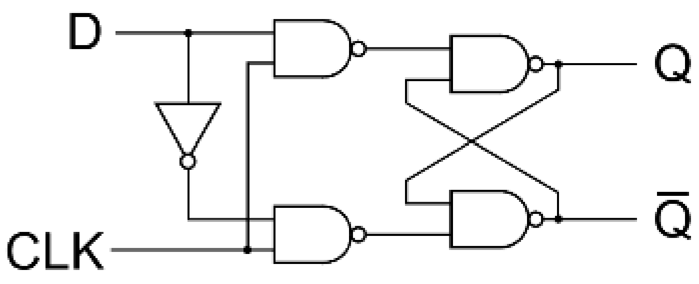


Figure 3: D Flip Flop

Basically, such type of flip flop is a modification of clocked RS flip flop gates from a basic Latch flip flop and NOR gates modify it in to a clock RS flip flop. The D input goes directly to S input and its complement through NOT gate is applied to the R input.

This kind of flip flop prevents the value of D from reaching the output until a clock pulse occurs. The action of circuit is s forward as follows.

When the clock is low, both AND gates are disabled, therefore D can change values without affecting the value of Q. On the other hand, when the clock is high, both AND gates are enabled. In this case, Q is forced equal to D when the clock again goes low, Q retains or stores the last value of D. The truth table for such a flip flop is as given below in table 6.

Table 6: Truth table for D Flip-Flop.

S	R	Q (t+1)
0	0	0
0	1	1
1	0	0
1	1	1

The excitation table for D flip flop is very simply derived given as under.

Table 7: Excitation table for D Flip-Flop.

S	R
0	0
1	1

JK Flip Flop

One of the most useful and versatile flip flop is the JK flip flop the unique features of a JK flip flop are:

1. If the J and K input are both at 1 and the clock pulse is applied, then the output will change state, regardless of its previous condition.

2. If both J and K inputs are at 0 and the clock pulse is applied there will be no change in the output. There is no indeterminate condition; in the operation of JK flip flop i.e. it has no ambiguous state. The circuit diagram for a JK flip flop is shown in figure 4.

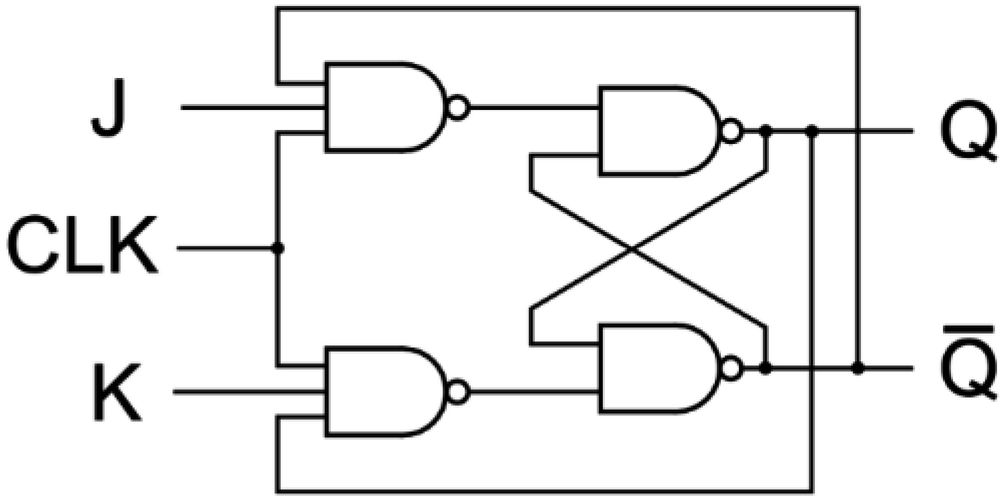


Figure 4: J-K Flip Flop circuit diagram.

When J = 0 and K = 0

These J and K inputs disable the NAND gates, therefore clock pulse have no effect on the flip flop. In other words, Q returns it last value.

When J = 0 and K = 1

The upper NAND gate is disabled the lower NAND gate is enabled if Q is 1 therefore, flip flop will be reset (Q = 0, =1) if not already in that state.

When J=1 and K=0

The lower NAND gate is disabled and the upper NAND gate is enabled if Q is at 1, As a result we will be able to set the flip flop (Q =1,= 0) if not already set

When J=1 and K=1

If Q = 0 the lower NAND gate is disabled the upper NAND gate is enabled. This will set the flip flop and hence Q will be 1. On the other hand if Q =1 the lower NAND gate is enabled and flip flop will be reset and hence Q will be 0. In other words, when J and K are both high, the clock pulses cause the JK flip flop to toggle. Truth table for JK flip flop is shown in table 8.

Table 8: The truth table for the J-K flip flop

Initial Conditions	Inputs (Pulsed)		Final Output
Q	S	R	Q (t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

The excitation table for D flip flop is very simply derived given as under.

Table 9: Excitation table for J-K Flip-Flop.

S	R	Q
0	0	No Change
0	1	0
1	0	1
1	1	Toggle

T Flip - Flop

A method of avoiding the indeterminate state found in the working of RS flip flop is to provide only one input (the T input) such, flip flop acts as a toggle switch. Toggle means to change in the previous stage i.e. switch to opposite state. It can be constructed from clocked RS flip flop be incorporating feedback from output to input as shown in Figure 5.

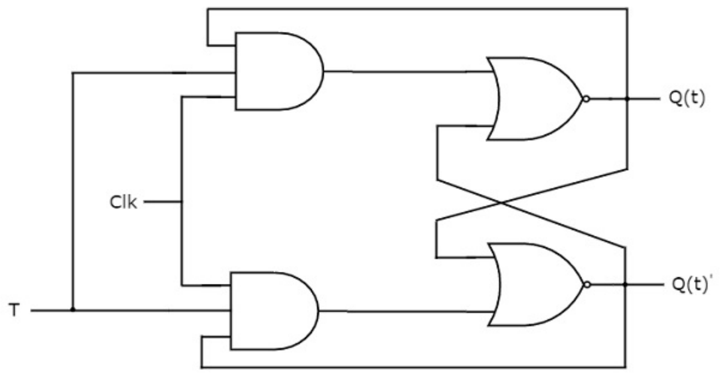


Figure: 5 T flip- flop circuit diagram.

Such a flip flop is also called toggle flip flop. In such a flip flop a train of extremely narrow triggers drives the T input each time one of these triggers, the output of the flip flop changes stage. For instance Q equals 0 just before the trigger. Then the upper AND gate is enable and the lower AND gate is disabled. When the trigger arrives, it results in a high S input.

This sets the Q output to 1. When the next trigger appears at the point T, the lower AND gate is enabled and the trigger passes through to the R input this forces the flip flop to reset.

Since each incoming trigger is alternately changed into the set and reset inputs the flip flop toggles. It takes two triggers to produce one cycle of the output waveform. This means the output has half the frequency of the input stated another way, a T flip flop divides the input frequency by two. Thus such a circuit is also called a divide by two circuits.

A disadvantage of the toggle flip flop is that the state of the flip flop after a trigger pulse has been applied is only known if the previous state is known. The truth table for a T flip flop is as given table 7.

Table 7: The truth table for a T flip flop

Q_n	T	Q_n+1
0	0	0
0	1	1
1	0	1
1	1	0

Instruction Manual

Experiment- 09

Objective: To set up and test a 7-segment static display system to display numbers 0 to 9.

Learning Objective:

To learn about various applications of decoder

To learn and understand the working of IC 7447

To learn about types of seven-segment display

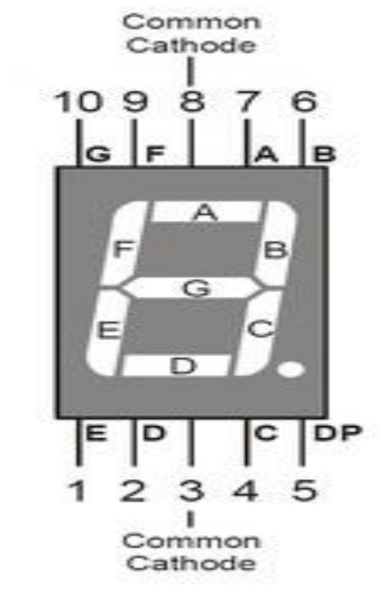
Apparatus Required:

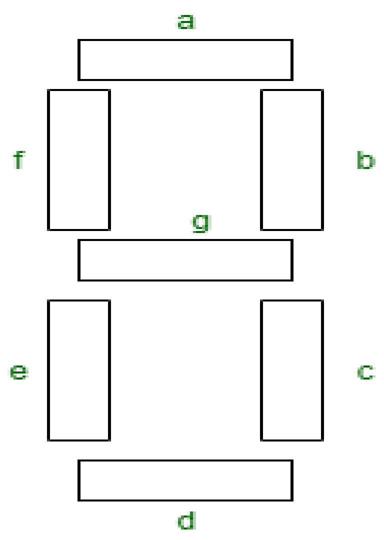
IC7447, 7-Segment display (commor anode), Patch chords, Breadboard.

Theory:

The Light Emitting Diode (LED) finds its place in many applications in these modern electronic fields. One of them is the Seven Segment Display. Seven-segment displays contains the arrangement of the LEDs in "Eight" (8) passion, and a Dot (.) with a common electrode, lead (Anode or Cathode). The purpose of arranging it in that passion is that we can make any number out of that by switching ON and OFF the particular LED's. Here is the block diagram of the Seven Segment LED arrangement.

The Light Emitting Diode (LED) finds its place in many applications in these modern electronic fields. One of them is the Seven Segment Display. Seven segment displays contains the arrangement of the LEDs in "Eight" (8) passion, and a Dot (.) with a common electrode, lead (Anode or Cathode). The purpose of arranging it in that passion is that we can make any number out of that by switching ON and OFF the particular LED's. Here is the block diagram of the Seven Segment LED arrangement.





Seven Segment Display

LED's are basically of two types-

Common Cathode (CC) - All the 8 anode legs uses only one cathode, which is common. Common Anode (CA)-The common leg for the entire cathode is of Anode type.

A decoder is a combinational circuit that connects the binary information from 'n' input lines to a maximum of 2" unique output lines. The IC7447 is a BCD to 7-segment pattern converter. The IC7447 takes the Binary Coded Decimal (BCD) as the input and outputs the relevant 7 segment code.

Circuit Diagram:

BCD Inputs				**Output Logic Levels from IC 7447 to 7-segments**							**Decimal number display**
D	C	B	A	a	b	c	d	e	f	g	
0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	1	0	0	1	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0	2
0	0	1	1	0	0	0	0	1	1	0	3
0	1	0	0	1	0	0	1	1	0	0	4
0	1	0	1	0	1	0	0	1	0	0	5
0	1	1	0	1	1	0	0	0	0	0	6

0	1	1	1	0	0	0	1	1	1	1	7
1	0	0	0	0	0	0	0	0	0	0	8
1	0	0	1	0	0	0	1	1	0	0	9

Procedure:

1. Check all the components for their working.
2. Insert the appropriate IC into the Breadboard.
3. Make connections as shown in the circuit diagram.
4. Verify the Truth Table and observe the outputs.