Al – Ma'mon University College

Computer Engineering Techniques Department

Experiments of Digital Electronic Laboratory

First Year

Experiment No. 1

Logic Gates

Object:

To verify the truth tables of the basic logic gates.

Theory:

Circuits used to process digital signals are called (Logic Gates) logic symbols are used to identify these circuits. The seven gates that are the fundamental logic elements in digital system are illustrated in (Fig 1)

The logic trainer is used to construct various logic gates and circuits the logic gates are shown clearly on this device. The lamps are used as indicators to the logic states of inputs and outputs when the lamp is ON, the logic state is HIGH or "1" while an OFF lamp indicates a Low or "0" logic state.

The switches on the logic trainer are used as inputs which provide the logic state "1" or "0".

Procedure:

1. Verify the truth table of the AND gate by connecting the circuit shown in the (fig. 2). On the logic trainer switches S1 and S2 are considered the inputs A and B while lamps L1 and L2 are used as indicators for the inputs A and B. Lamp L3 is used as indicator for the output C.
2. Repeat step 1 for all other types of the logic gates as illustrated in (fig. 1).

3. Verify the truth tables of the 3 inputs OR, NOR and NAND gates.

4. Show how the NAND, NOR and XOR gates can be used as inverters or NOT gate.

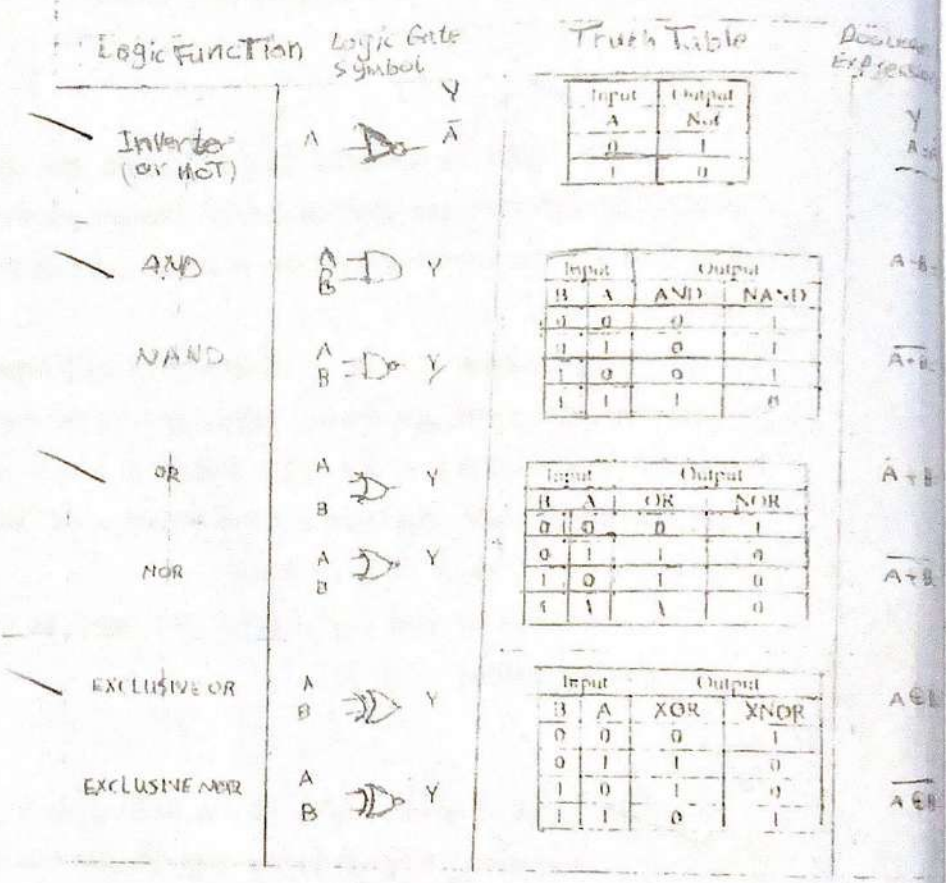


Fig. (1) Fundamental of the basic logic gates

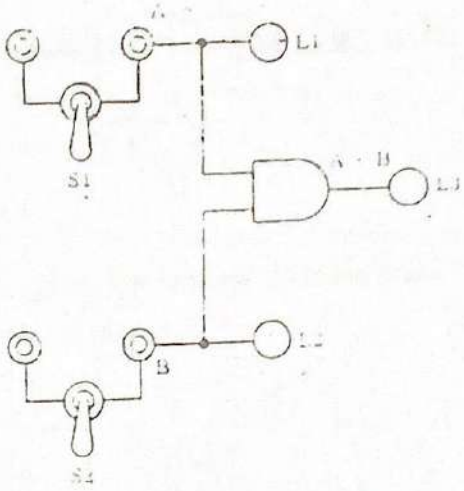


Fig . (2)Logic Circuit for "AND" Gate

Report:

1 Write the truth tables for the digital system shown in (Fig 3)

2 Describe the output form the gate shown in (fig 4)?

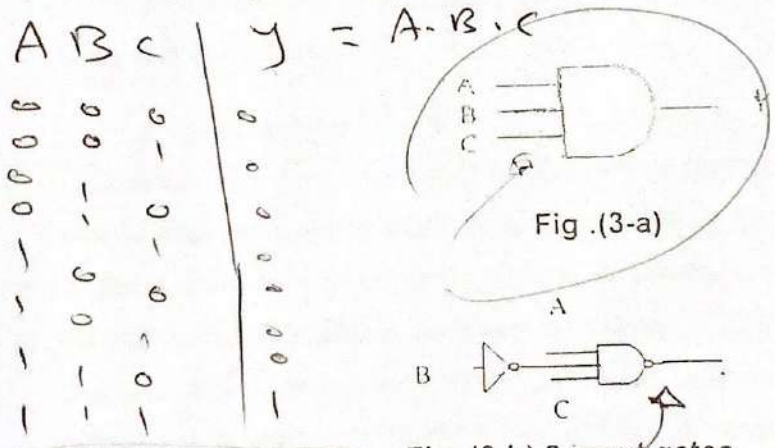
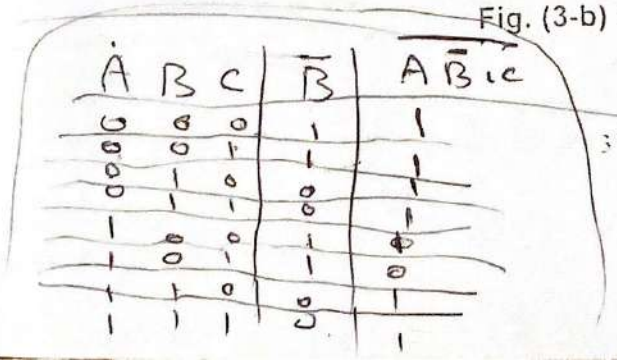


Fig .(3-a)

Fig. (3-b) 3 input gates



$y = A\bar{B}C$

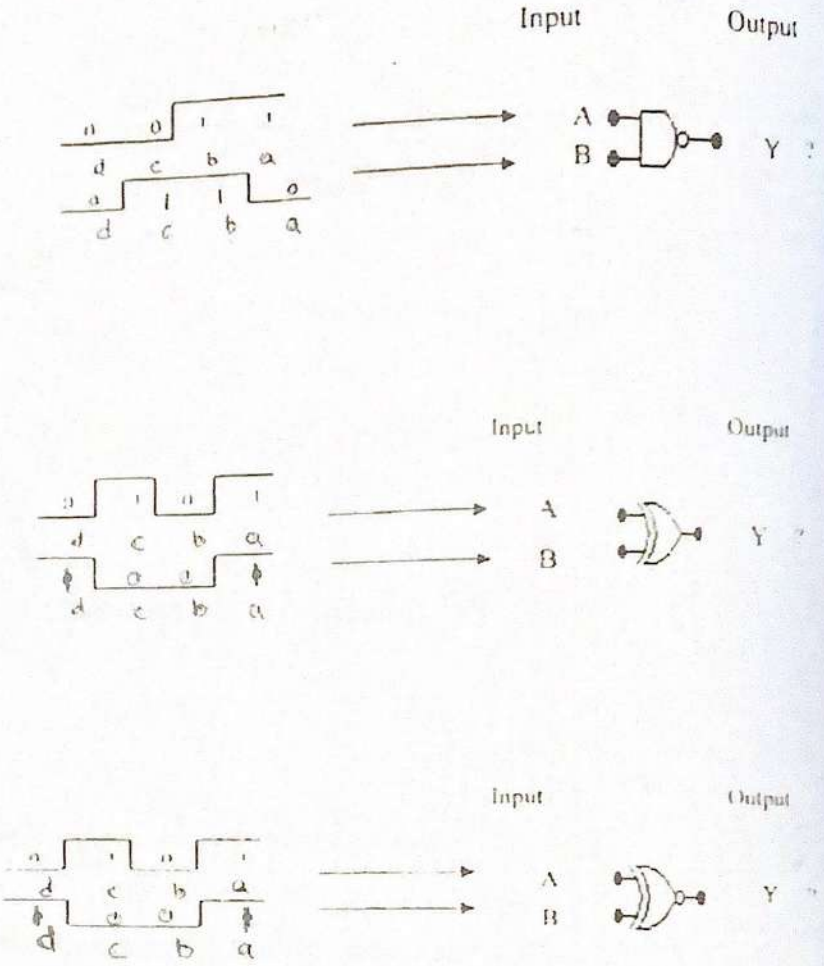


Fig. (4)

Experiment No.2

Combinational Logic Circuits

Object:

To realize logic gate combinations from truth tables and Boolean expressions and vice-versa.

Theory.

Digital systems are composed of combinations of logic gates described by a truth table, and Boolean expression, or a logic symbol diagram. Consider the truth table illustrated in fig. 1 (a). This, which shows all of the possible combination of the three inputs (C, B and A). Only the combination 010 will produce a "1" or HIGH output. An equivalent Boolean expression for this truth table is listed to the right of the table in Fig. (1) a. The inputs are ANDed, forming the Boolean expression $\overline{A}B\overline{C} = Y$ (not A AND B AND not C equals output Y).

A logic symbol diagram is developed from the Boolean expression. Such a procedure is diagramed in Fig.1 (b). Input A and C must complemented using an inverter. A 3-input AND gate is being used as the output. A commonly used simplified version of the same logic symbol diagram is shown in Fig.1 (c). In this diagram the inverters are shown as invert bubbles. These invert bubbles may also be considered as active LOW inputs.

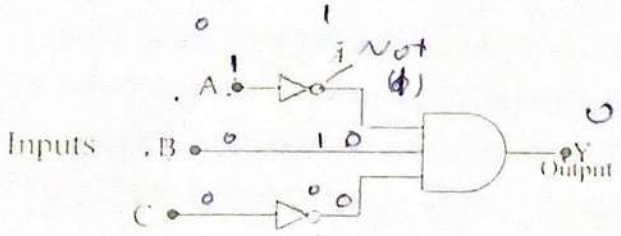
In other words, to activate the AND gate in Fig 1(c), input A and C must be LOW, while B must be HIGH. Because input B must be HIGH to activate the AND gate, they are considered active HIGH inputs

INPUT			OUTPUT
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

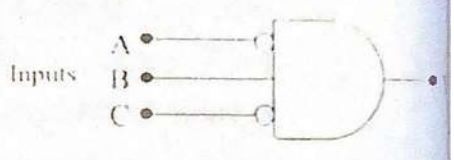
(a)

Boolean Expression

$\bar{A}.B.\bar{C} = Y$



(b)



(C)

Fig. (1) (a) converting truth table to equivalent Boolean expression

(b) Converting Boolean expression to Logic – symbol diagram.

(c) Another symbol Diagram

Consider the truth table in Fig 2 (a). Two input combination will produce "1" or HIGH output. The Boolean expression for this

truth table then becomes $A\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C} = Y$ (read as A AND B AND not C Oared with not A AND B AND not C equals the output Y)

The Boolean expression is next converted to a logic symbol diagram. This procedure is illustrated in Fig 2(b)

Note that this type of the Boolean expression creates an AND – OR pattern of logic gates with the OR gate being nearest the output. The pattern of this Boolean expression is called the sum of products (or min term form). Min term Boolean expression are generated from the 1's is the output column of the truth table is the manner shown in Fig 2(a).

INPUT			OUTPUT
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

$\bar{A}.B.\bar{C}$

$A\bar{B}\bar{C}$

(a)

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

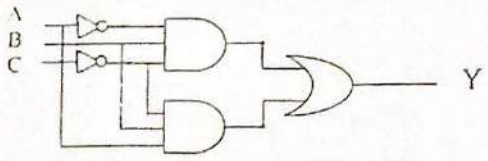


Fig (2) (a) Converting truth table to equivalent Boolean Expression

(b) Converting minterm Boolean expression to logic-symbol diagram

In addition to the sum of product method, another method can be used to obtain the Boolean expression from the truth table and it is known as the product of sums method (or Max Term method).

In order for realized logic circuit to be with minimum number of gates the Boolean expression is simplified using Boolean algebra for example:

F(W,X,Y,Z)= $X + YZ\,X + YZ\,\overline{X} + W\,X + \overline{W}\,X + \overline{X}\,Y$

= $X + YZ(X + \overline{X}) + \overline{X}\,Y + X(W + \overline{W})$

= $(X + \overline{X}\,Y) + YZ + X$

= $X + Y + YZ$

= $X + (Y + YZ)$

= $X + Y$

Hence the required logic circuit is a symbol OR gate alternatively the simplified Boolean expression can be obtained directly from the truth table using the Karnaugh – Map (K-map) technique.

Procedure:

1. Find the Boolean expression for the circuits shown in Fig. (3) then connect them and find their truth Table.

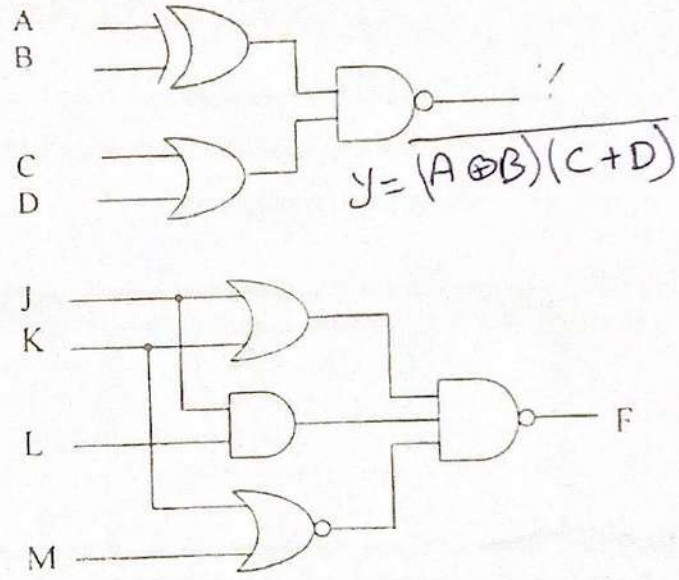


Fig (3)

2. Realize the following Boolean expression using AND, OR and NOT gates

   (i) $F = \bar{A}B\bar{C} + B\bar{C}$

   (ii) $F = W(X + \bar{Y})$

3. Repeat step 2 using:

   (i) NAND gates only.

   (ii) NOR gates only.

Report

1. Realize the logic circuits, which implement the following Boolean expression, simplify the expression whenever it is possible before realization.
   (a) F= $ab + a\bar{c} + bc$
   (b) F= $(x + \bar{y} + xy)(x + \bar{y})$
2. Realize the logic circuit that its truth table is shown below by using sum of products.

INPUTS			OUTPUT	
A	B	C	F	Q
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

3. Realize the logic circuits which implement the following Boolean expression, using K-map for simplification:

   (i) F(A,B,C,D) = $\sum_m (0,2,5,7,8,10,13,15)$

   (ii) F(A,B,C,D) = $\sum_m (1,2,14) + d(0,3,5,10)$

Experiment No.3

Codes Conversion

Object:

To consider various important Binary Coded Decimal (BCD) codes and the logic circuit used for converting from one code to another.

Theory:

The BCD codes combine features of decimal and binary number. In general, a BCD code is one in which the digits of a decimal number are encoded-one at a time-into a group of binary digits. For this encoding we can use 4-bit groups, 6-bit ... etc groups.

The 8421 code is a mixed – base code; it is binary within each group of 4-bits, but it is decimal from group to group, because the 8421 BCD code is the most natural type of BCD code, it is often referred to as BCD without qualifying it. In another words, if we refer to the BCD code, we mean the 8421 code.

Many other 4 – bit code exist. Table (1) shows some common ones. As usual, a decimal numbers greater than 9 are encoded a digit at a time.

Decimal	Weighted Codes															
	8	4	2	1	2	4	2	1	5	2	1	1	8	4	2	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	1
2	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	1	1	0	1	1	1	0	0	0	1
4	0	1	0	0	0	1	0	0	0	1	1	0	0	1	0	0
5	0	1	0	1	1	0	1	1	1	0	0	1	1	0	1	1
6	0	1	1	0	1	1	0	0	1	0	0	0	1	1	1	0
7	0	1	1	1	1	1	0	1	1	0	1	1	1	1	0	1
8	1	0	0	0	1	1	1	0	1	1	1	0	1	1	0	0
9	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

Table (1)
Binary Codes (weighted codes)

All the codes in the table are weighted codes. All use positive weights except the last code of table (1). This code uses negative weights, as well as positive.

Table (2) shows other types of un weighted BCD codes, Ex-3 and Gray code.

To encode a decimal number into its excess-3 form, add 3 to each decimal digit before converting to binary. The Gray code is also an unweighted code. Each Gray number differs from the preceding number by a signal bit. To convert binary number to Gray number the first digit is the same as the first binary digit. Then add each pair of adjacent bits to get the next Gray digit.

Decimal	Unweighted							
	Ex – 3				Gray			
0	0	0	1	1	0	0	0	0
1	0	1	0	0	0	0	0	1
2	0	1	0	1	0	0	1	1
3	0	1	1	0	0	0	1	0
4	0	1	1	1	0	1	1	0
5	0	0	0	0	0	1	1	1
6	1	0	0	1	0	1	0	1
7	1	0	1	0	0	1	0	0
8	1	0	1	1	1	1	0	0
9	1	1	0	0	1	1	0	1

Table (2)
Binary Codes (Unweighted Codes)

Procedure:

1. Design a logic circuit to convert a BCD code to Gray code.
(a)From the truth table shown below, since it is dealing with BCD code, therefore states (10 – 15) are don't care.

Decimal Code	Inputs (BCD) 8421				Outputs (Gray)			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1

(b) Then simplify each of the output function using K-map, getting:

W= A ; X= A + B

Y= B ⊕ C ; Z= C ⊕ D

(c) Therefore the logic circuit diagram is shown in fig (1):

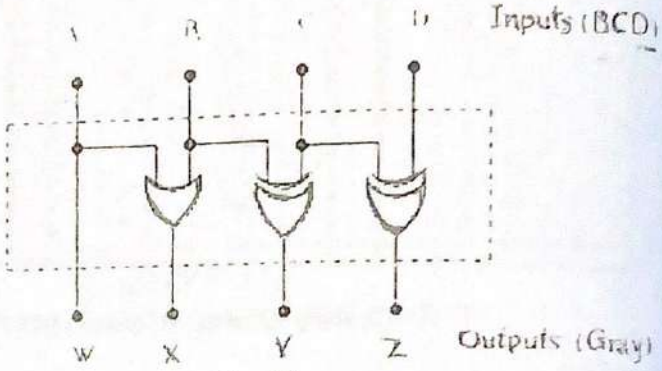


Fig.(1)

(d) Connect the circuit shown in fig (1), and then verify its truth table

2. Design a logic circuit to convert a BCD code to Excess – 3 codes.

(a) Fill the following table:

Decimal Code	Inputs (BCD) 8421				Outputs (Excess – 3)			
	A	B	C	D	W	X	Y	Z
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				

(b)Use K-map to obtain the code conversion of expressions of W, X, Y, Z.

(c) Implement the circuit using available gates on the Logic trainer and then check the result.

Report:

1. Convert each Gray code to binary code.
   A. 1010 B. 00010 C. 11000010001
2. Convert Each Ex-3 code number to decimal.
   A. 0011 B. 1001 C. 10000101
3. Design a logic circuit to convert a 4 – bit Gray code to a binary as in the table shown below.

Decimal Code	Gray Code				Binary			
	G_4	G_3	G_2	G_1	B_4	B_3	B_2	B_1
0	0	0	0	0				
1	0	0	0	1				
2	0	0	1	1				
3	0	0	1	0				
4	0	1	1	0				
5	0	1	1	1				
6	0	1	0	1				
7	0	1	0	0				
8	1	1	0	0				
9	1	1	0	1				
10	1	1	1	1				
11	1	1	1	0				
12	1	0	1	0				
13	1	0	1	1				
14	1	0	0	1				
15	1	0	0	0				

Experiment No. 4

Parity Generators / Checkers

Object:

To study how to detect an error in the data.

Theory:

Errors can occur as digital codes being transferred from one point to another within a digital system or while codes are being transmitted from one system to another. The errors take the form of undesired changes in the bits that make up the coded information; that is, "1" can change to "0", or a "0" to "1", due to component malfunction or electrical noise. Many systems, however, employ parity bit as a means of detecting error. A word always contains either an even or an odd number of 1's.

For this reason digital system employ some method for detection (some times correction) of errors. One of the simplest and most widely used schemes for error detection is the parity bit method. The two different methods used, are:

1. Even Parity Method:

Even parity means attaching an extra bit to a group of bits to produce an even number of 1's. For instance, suppose we have a word like 0111 there are three ones in this word, an odd number of 1's. We attach an extra 1 to the word to get 0111 1. This new word can be checked for even parity at different points to assume that no errors have occurred into the word.

2. Odd Parity method:

It is used in exactly the same way of the even parity method except that the parity bit has to be chosen so that the total number of 1's (including the parity bit) is an odd number.

As an illustration of how parity bits are attached to a code word, Table (1) lists the parity bits for each BCD code number for both even and odd parity. The parity bit for each BCD number is given in the P column.

Even Parity					Odd Parity				
P	8	4	2	1	P	8	4	2	1
0	0	0	0	0	1	0	0	0	0
1	0	0	0	1	0	0	0	0	1
1	0	0	1	0	0	0	0	1	0
0	0	0	1	1	1	0	0	1	1
1	0	1	0	0	0	0	1	0	0
0	0	1	0	1	1	0	1	0	1
0	0	1	1	0	1	0	1	1	0
1	0	1	1	1	0	0	1	1	1
1	1	0	0	0	0	1	0	0	0
0	1	0	0	1	1	1	0	0	1

Table (1)

Even and Odd Parity

The parity bit can be attached to the code group at either the beginning or the ends depending on system design.

Note that the odd parity bit is always the complement of the even parity bit.

Parity Logic

In order to check for all generate the proper parity in a given code word, a very basic principle can be used. The sum (disregarding carriers) of an even number of 1's is always zero, and the some of an odd number of 1's is always one. Therefore, in order to determine if a given code word is even or odd parity, all of the bits in that code word are summed. The sum of two bits can be generated by an XOR gate, as shown in Fig.1 (a), the sum of the three bits can be formed by two XOR gates connected as shown in fig.1 (b); and so on.

The parity generation/ detection logic for a four-bit code (including parity) is shown in Fig (1) c.

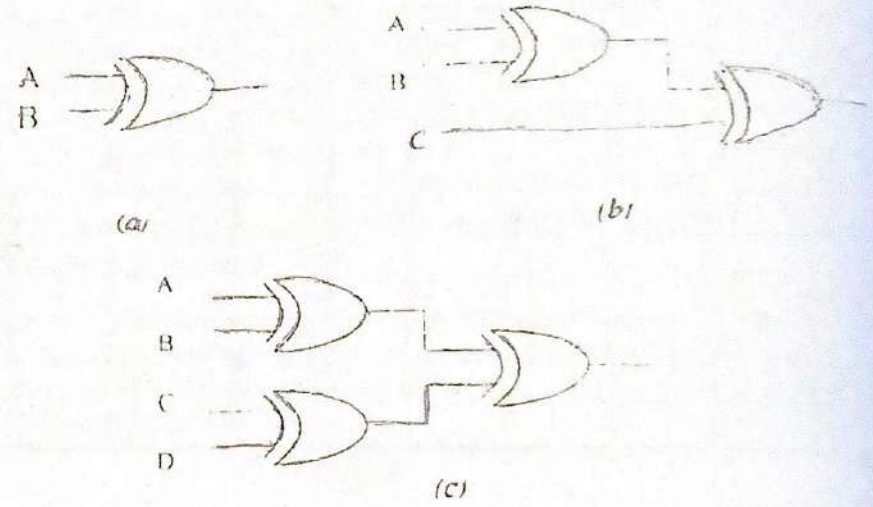


Fig.(1) (a) Summing of two bits.
(b) Summing of three bits.
(c) Four-bit parity checker.

To explain these properties, the following examples are given:

Example 1:

Design an even/odd parity generator for 3-bit data if:

(a) The truth table for the even parity generator is shown in table two.

A	B	C	P/EVEN
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

Table (2)

Form the truth table the output function using K-map:

$P = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$

Hence, the above expression can be rewritten as :

$P = A \oplus B \oplus C$

And this expression can be implemented by the logic circuit shown in Fig(2).

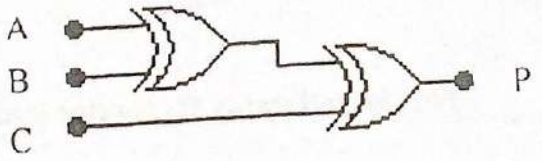


Fig (2) Even Parity Generator (3-bit data)

OR

(b) The truth table for the odd parity generator is shown in table three.

A	B	C	P/ODD
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Table (3)

Form the truth table the output function using K-map:

$P = \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}C + ABC$

Hence, the above expression can be rewritten as:

P= $\overline{A \oplus B \oplus C}$ or $\bar{P} = A \odot B \odot C$

And this expression can be implemented by the logic circuit shown in Fig (3).

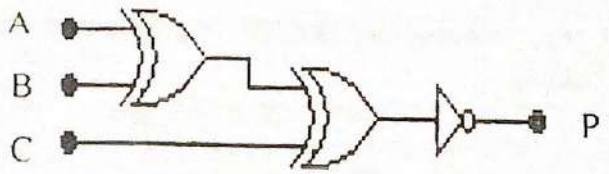


Fig (3) Odd Parity Generator (3-bit data)

Example 2:

Show how to design a parity checker circuit for a three bit data.

A	B	C	O/P (Y)
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

Y=0 For even parity

Y=1 For odd parity

Table (4)

After simplification of the truth table for using K-map we get:

$Y = A \oplus B \oplus C$

And this expression can be implemented by the logic circuit shown in Fig (4).

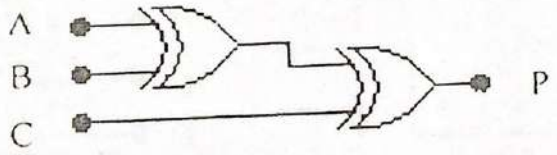


Fig. (4) Parity Checker For a 3-bit data

Procedure:

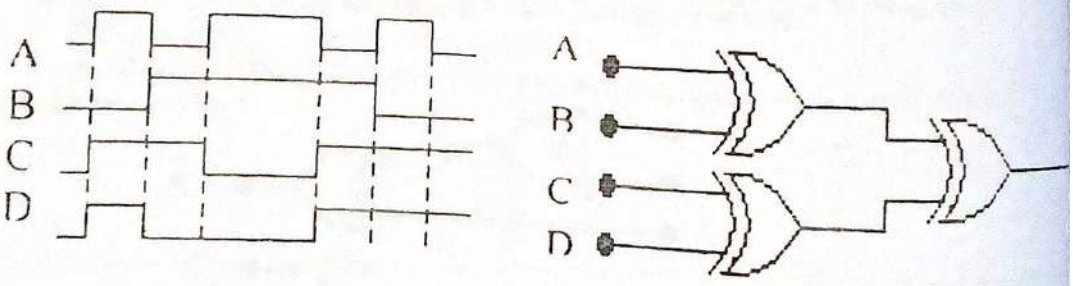
1. Design an even/odd parity generator for 4-bit data then implement it on the Logic trainer.
2. Design a parity checker circuit for a 4-bit data then implements it on the Logic trainer.

Report:

1. Attach the proper parity bit to the following code to get an even parity:

A. 11010 B. 1001 c. 0111101

2. Repeat problem 1 for odd parity.
3. The waveforms shown in Fig (5) are applied to 4-bit parity logic. Determine the output wave form in proper relation to the input. How many times does even parity occur?



(a) Waveforms (b) Circuit diagram

Fig.(5)Parity Checker For a 4-bit data