Salahaddin University-Erbil
College of Engineering
Electrical Engineering Dept.

Digital Electronic Circuits Lab.

Student Manual

Prepared by

Mr. Basheer A. Abdullah   Mr. Azad N. Abdulla
Mr. Goran Wnis   Mr. Niyaz O. Ali

Contents

Experiment No. 1
Basic Logic Gates Implementation and Verification

Introduction:

A gate is a logic circuit with one output and one or more inputs. An output occurs only for a certain combinations of input signals. Logic gates are digital (two states) circuits because the input and the output signals are either high or low voltages. Gates can be described with Boolean algebra.

A diode is like an electronic switch. When a diode is forward biased and a required forward voltage is applied, the diode is on (switch is on). However, when zero voltage or a revered biased voltage is applied the diode is off (switch is off). This characteristic of the diode is very useful to build logic gate such as OR and AND gates.

In a transistor, the collector-to-emitter impedance is quite low near or at saturation and large near or at cutoff. For instance, the load line defines saturation as the point where the current is quite high and the collector-to-emitter voltage quite low as shown in Figure 1.1. At cutoff, the current is relatively low and the voltage near its maximum value. The above impedance levels established by "on" and "off" transistors make it relatively easy to understand the operation of the logic gates.
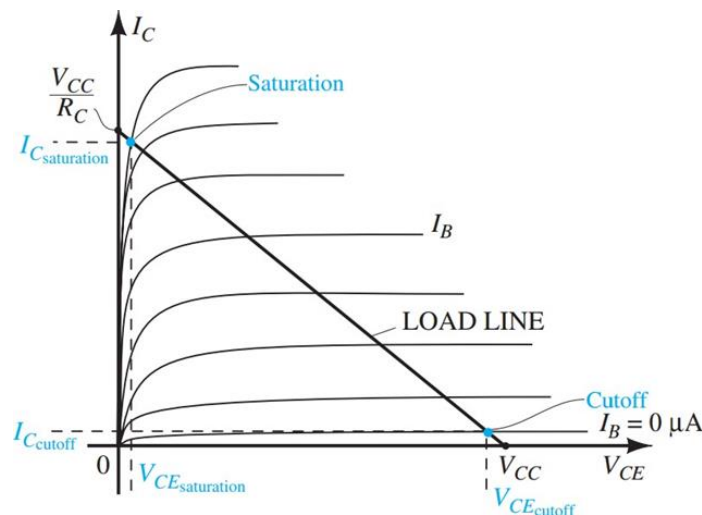
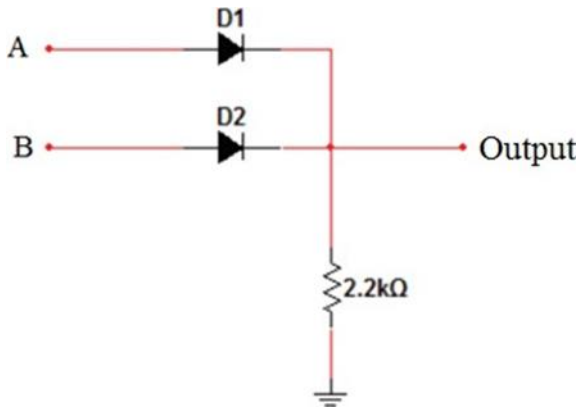Figure 1.1 Points of operation for a BJT logic gate.
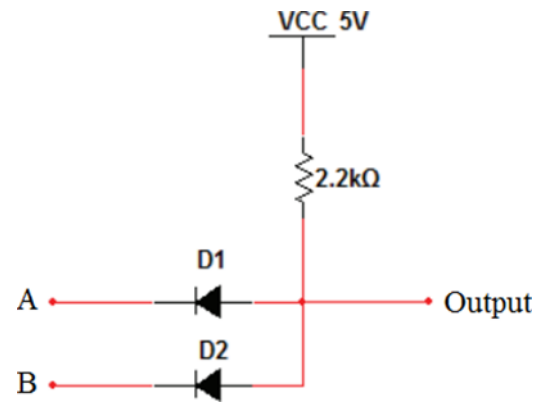
The Circuits:

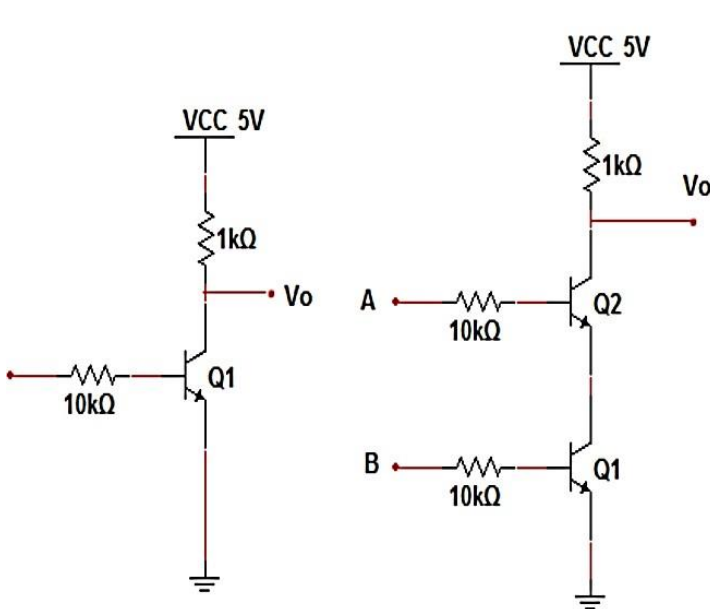Figure 1.2 OR gate

Figure 1.3 AND gate

Figure 1.4 Invertor
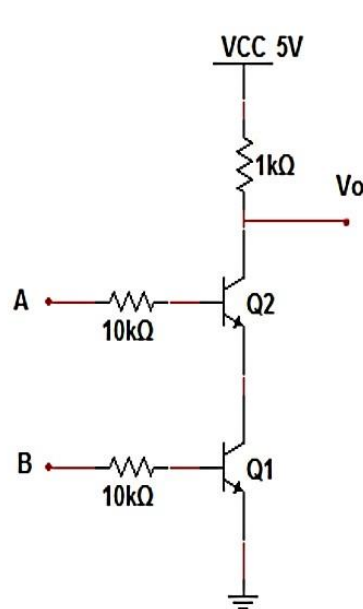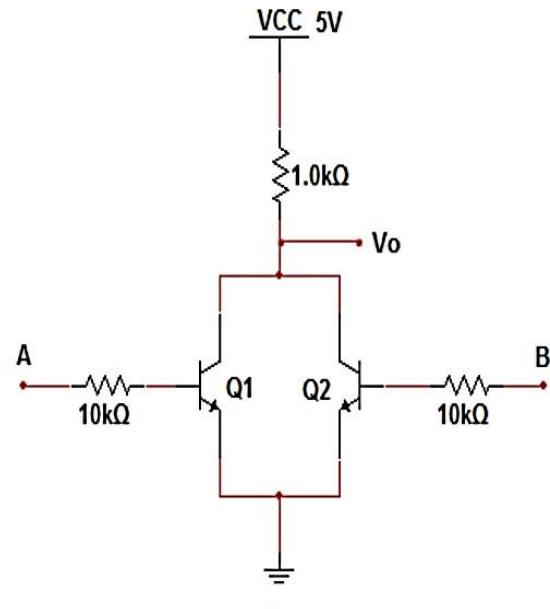

Figure 1.5 NAND gate

Figure 1.6 NOR gate

Logic Gates

1. **AND gate:** Function of AND gate is to give the output true when both inputs are true. In all the other remaining cases output becomes false.

2. **OR gate:** Function of OR gate is to give output true when at least one of the inputs is true. In the remaining case output becomes false.

3. **NOT gate:** Function of NOT gate is to reverse the nature of the input. It converts true input to false and vice versa.

4. **NAND gate:** Function of NAND gate is to give true output when at least one of the provided inputs is false.

5. **NOR gate:** NOR gate gives the output true when all provided inputs are false. In all the other cases output remains false.

6. **XOR gate:** The function of XNOR gate is to give output true when one of the inputs is true and the other is false.

7. **XNOR gate:** The function of XOR gate is to give output true only when both inputs are true or false.

Name	NOT	AND	NAND
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$
Symbol			
Truth Table	A X 0 1 1 0	B A X 0 0 0 0 1 0 1 0 0 1 1 1	B A X 0 0 1 0 1 1 1 0 1 1 1 0

Figure. 1.7

OR	NOR	XOR	XNOR
$A + B$	$\overline{A + B}$	$A \oplus B$	$\overline{A \oplus B}$

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

Figure 1.8

Procedure:

Verify the truth tables as shown in Figure 1.7 & 1.8 by connecting their ICs.

Discussion:

Write truth table of the logic gates for the three inputs.

Introduction:

A universal gate is a gate which can implement any Boolean function without need to use any other gate type. The NAND gate is the universal gate. In practice, this is advantageous since NAND gate is economical and easier to fabricate and is the basic gate used in all IC digital logic families. All logic gates can be implemented by NAND gate as shown in figure below.
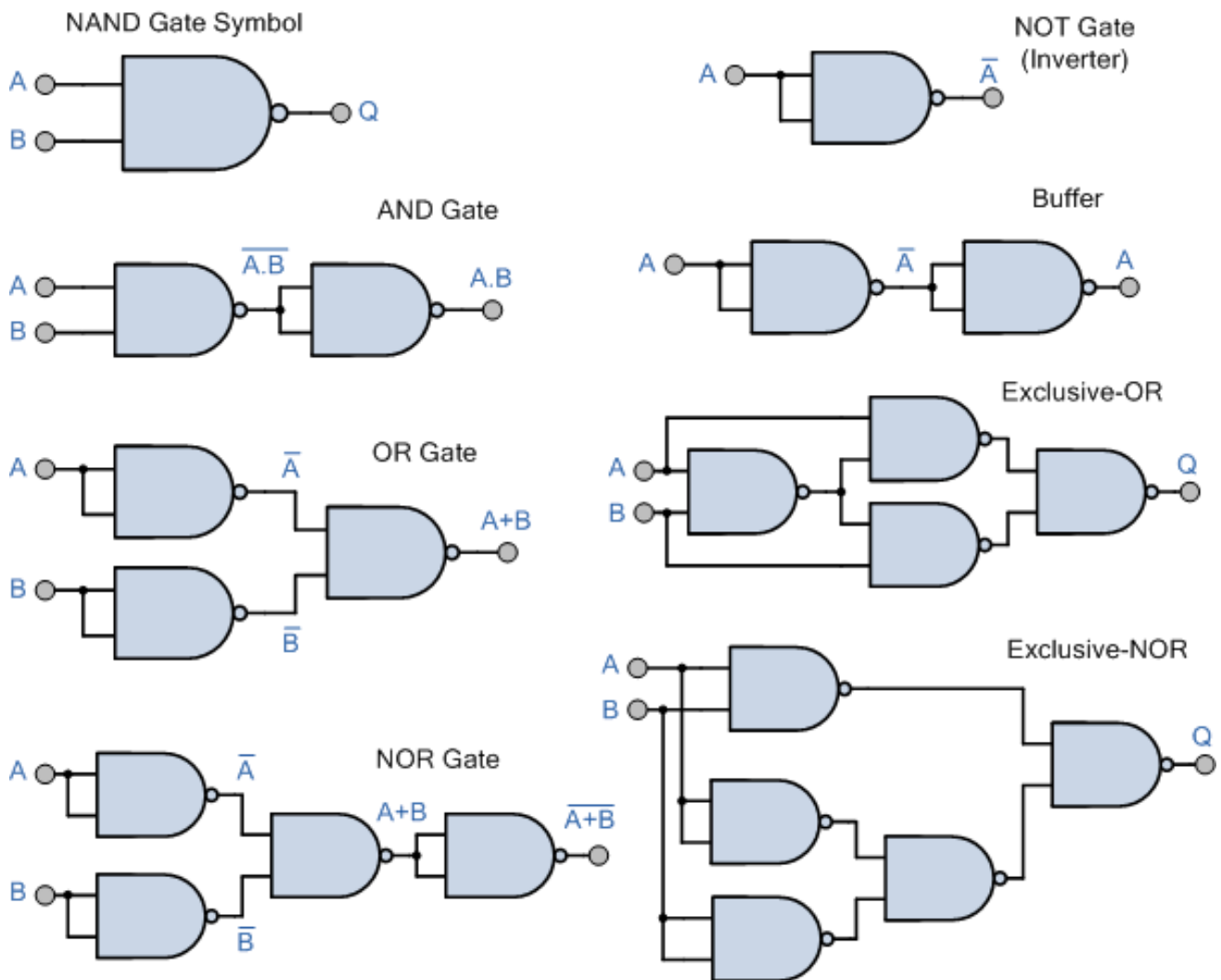
Figure 2.1

Procedure:

Connect the circuits shown in Figure 2.1. Then, verify their truth table.

Discussion:

Implementing $Y = ABC + A\bar{B}C + \bar{A}\bar{B}C + AC$ by using only NAND gates.

Implementation of Given Boolean Function
Using Logic Gates in both SOP and POS

Introduction:

a) **SOP**: It is the Sum of product form. It is denoted in the K-map expression by sigma (Σ).

   Y= A.B + A'.B'

Truth Table for this SOP expression:

A	B	A'	B'	A.B	A'.B'	Y= A.B + A'.B'
0	0	1	1	0	1	1
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	1	0	0	1	0	1

Logic Circuit of this SOP expression:
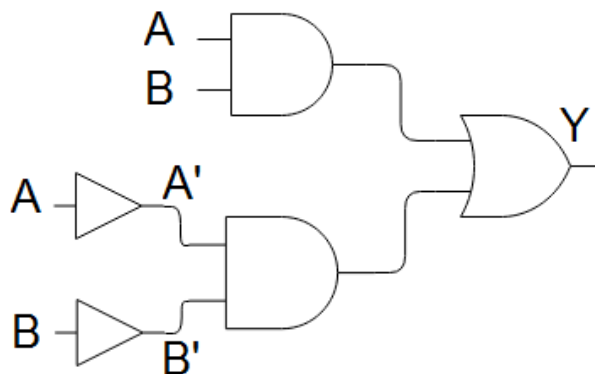
Figure.3.1

b) **POS**: It is the product of the sums form. It is denoted in the K-Map expression by the sign pie (π).

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

Truth Table for this POS expression:

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

Logic Circuit of this POS expression:
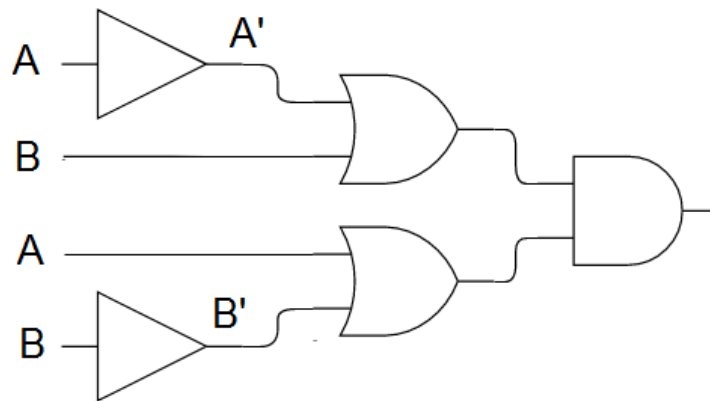
Figure 3.2

Procedure:

Connect the circuits shown in Figure 3.1 and 3.2. Then, verify their truth table.

Discussion:

Write two Boolean expressions for three inputs of SOP & POS with their truth tables and logic circuits.

<div align="center">

Experiment No. 4
Half and Full Adder

</div>

Introduction:

Digital computers perform a variety of information-processing tasks. Among the basic functions encountered are the various arithmetic operations. The most basic arithmetic operation is the addition of two binary digits. This simple addition consists of four possible elementary operations namely, 0+0=0, 0+1=1, 1+0=1, 1+1=10. The first three operations produce a sum whose length is one digit, but when both augends and addend bits are equal to 1, the binary sum consists of two digits. The higher significant bit of this result is called a carry. When the augends and addend numbers contain more significant digits, the carry obtained from the addition of two bits is added to the next Higher-order pair of significant bits. A combinational circuit that performs the addition of two bits is called a (half-adder). One that performs the addition of three bits (tow significant bits and a previous carry) is a (full-adder). The name of the former stems from the fact that tow half-adder can be employed to implement a full-adder.

Half-Adder: From the verbal explanation of a half-adder, we find that this circuit needs tow binary inputs and tow binary outputs:

B	A	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The carry output is 0 unless both inputs are 1. The sum output represents the least significant bit of the result.

The Boolean functions for the tow outputs:

$Sum = A \oplus B$ ………… (1)

$Carry = A \cdot B$ ………… (2)

Figure 4.1 A represent the implementation of the half-adder.

Full-Adder is a combinational circuit that forms the arithmetic sum of three input bits. It consists of three inputs and two outputs. The third input, Ci, represents the carry from the previous lower significant position. The truth table of the full-adder is as follows:

B	A	C_i	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

The full-adder can be implemented with two half-adders and OR gate. The sum output from the second half-adder is the EX-OR of Ci and the output of the first half-adder, giving:

$$Sum = C_i \oplus (A \oplus B)$$

$$= A\bar{B}\bar{C_i} + \bar{A}B\bar{C_i} + ABC_i + \bar{A}\bar{B}\bar{C_i} \quad \ldots\ldots\ldots\ldots \text{(3)}$$

$$Carry = AB + AC_i + BC_i \quad \ldots\ldots\ldots\ldots \text{(4)}$$

Procedure:

1. Connect the circuit of Figure 4.1 A. Verify its truth-table.
2. Connect the circuit of Figure 4.2 B. Verify its truth-table.

Discussion:

1. Draw the full-adder circuit using NAND gates only.
2. Derive equation (4).
3. Construct a full- adder using half- adder circuit.
4. Construct a binary adder that adds two 5-bit numbers.
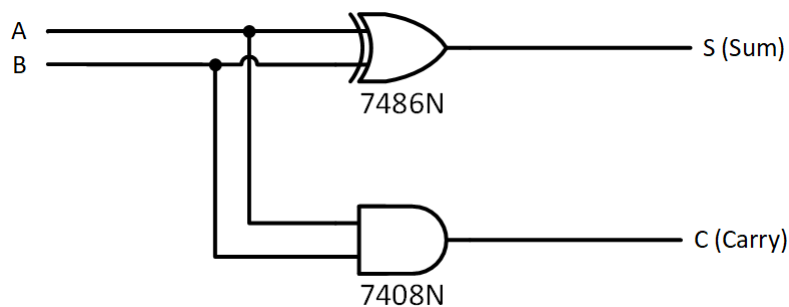
Figure 4.1 A
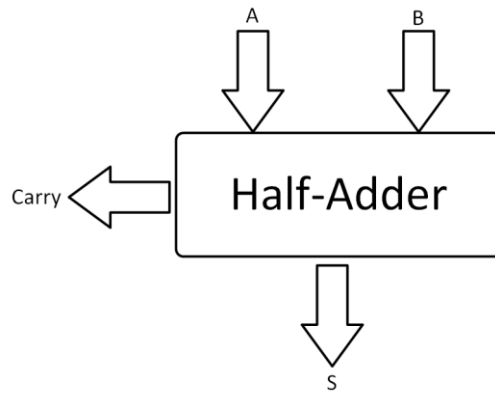
Figure 4.1 B
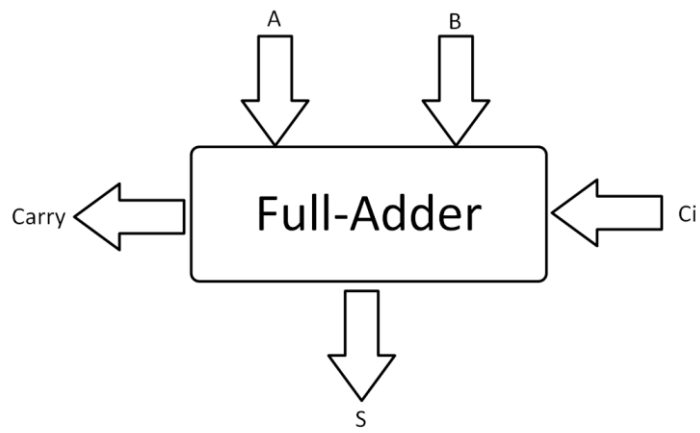
Figure 4.2 A
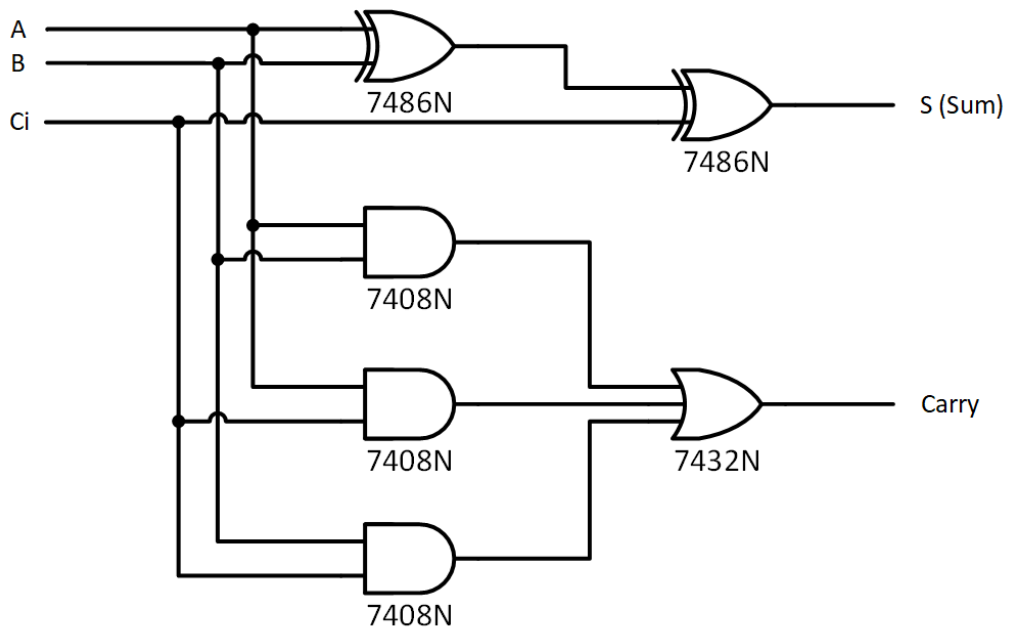
Figure 4.2 B

Experiment No. 5
Magnitude Comparator

Introduction:

The comparison of two numbers is an operation that determines if one number is greater than, less than, or equal to the other number. A magnitude comparator is a combinational circuit that compares two numbers, A and B, and determines their relative magnitudes. The outcome of the comparison is specified by three binary variables that indicates whether A > B, A = B or A < B.

The operation of a two one-bit comparator can be described by following truth-table:

A	B	E (A = B)	G (A > B)	L (A < B)
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

According to the truth table, the Boolean equation will be as follows:

$E = \overline{A}\,\overline{B} + A\,B$ .... (1)

$G = A\,\overline{B}$ .... (2)

$L = \overline{A}\,B$ .... (2)

Equation (1) represents a (2-input) EX-NOR gate, and can be rewritten as:

$E = A \odot B = \overline{A \oplus B} = \overline{A}\,\overline{B} + A\,B$ .... (4)

Therefore, from equation (4), (2), and (3), a 1-bit magnitude comparator logic circuit can be implemented as shown in Figure 5.1.

Note: MSI IC 7485 is a 4-bit magnitude comparator.

Procedure:

1. Connect the circuit shown in Figure 5.1.
2. Construct the truth-table.
3. Connect the circuit shown in Figure 5.2.
4. Construct the truth-table.

Discussion:

1. Design a 1-bit magnitude comparator using (NAND) gate only.
2. Design a 3-bit magnitude comparator using (NAND) gates only.
3. Design a 3-bit magnitude comparator using the 7485 IC. Write down its truth-table.
4. Design an 8-bit magnitude comparator, using the 7485 ICS.
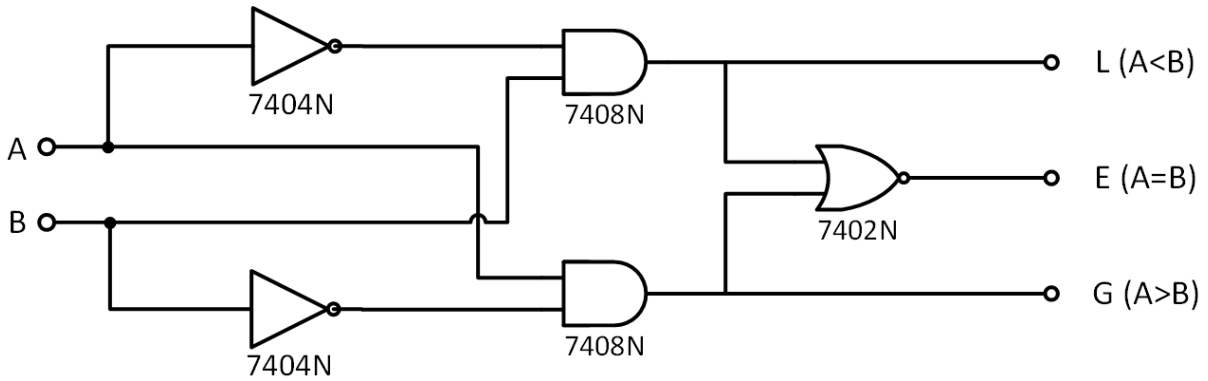
Figure 5.1
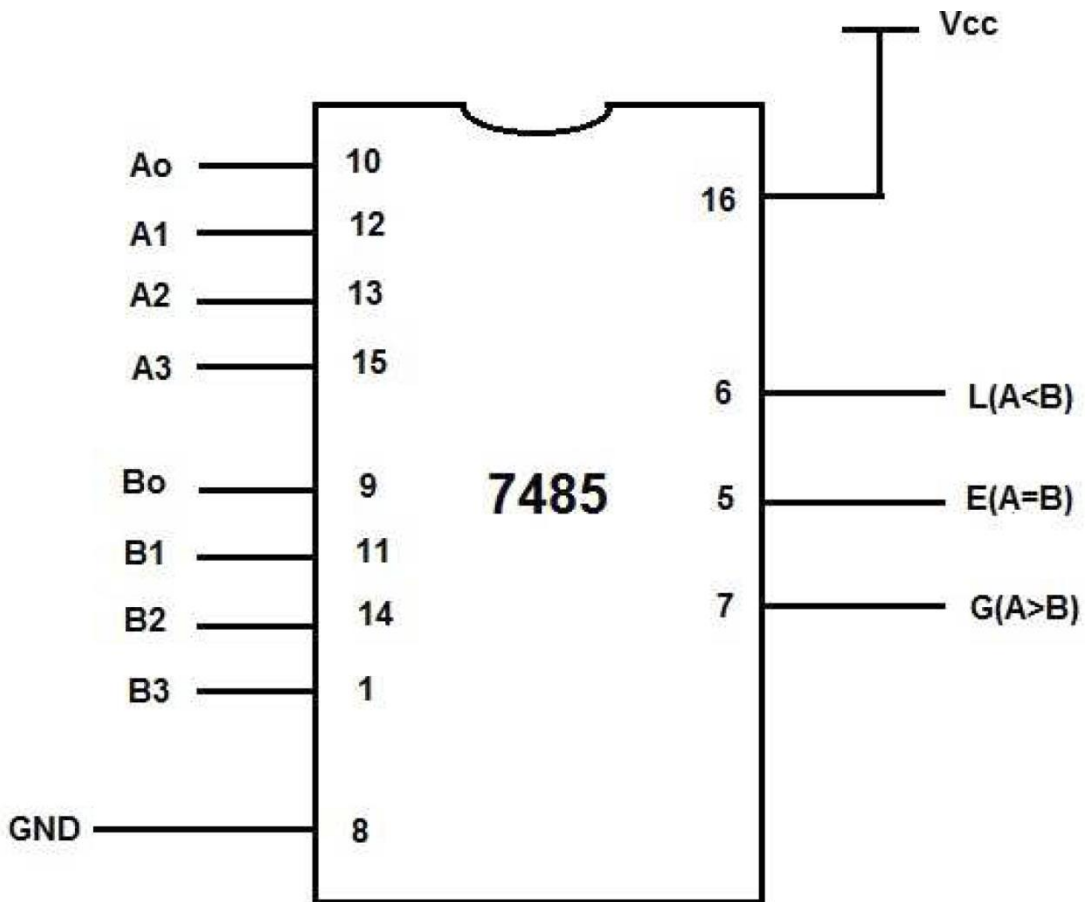
Figure 5.2

Introduction:

A. Multiplexer (Data Selector):

A multiplexer (MUX) is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to common destination. The basic multiplexer, then has several data input lines and a single output line. It also has data selector inputs that permit digital data on any one of the inputs to be switched to the output line.

A simple multiplexer can be represented by switch operation that sequentially connects each of the input lines with the output, as illustrated in Figure 6.1.
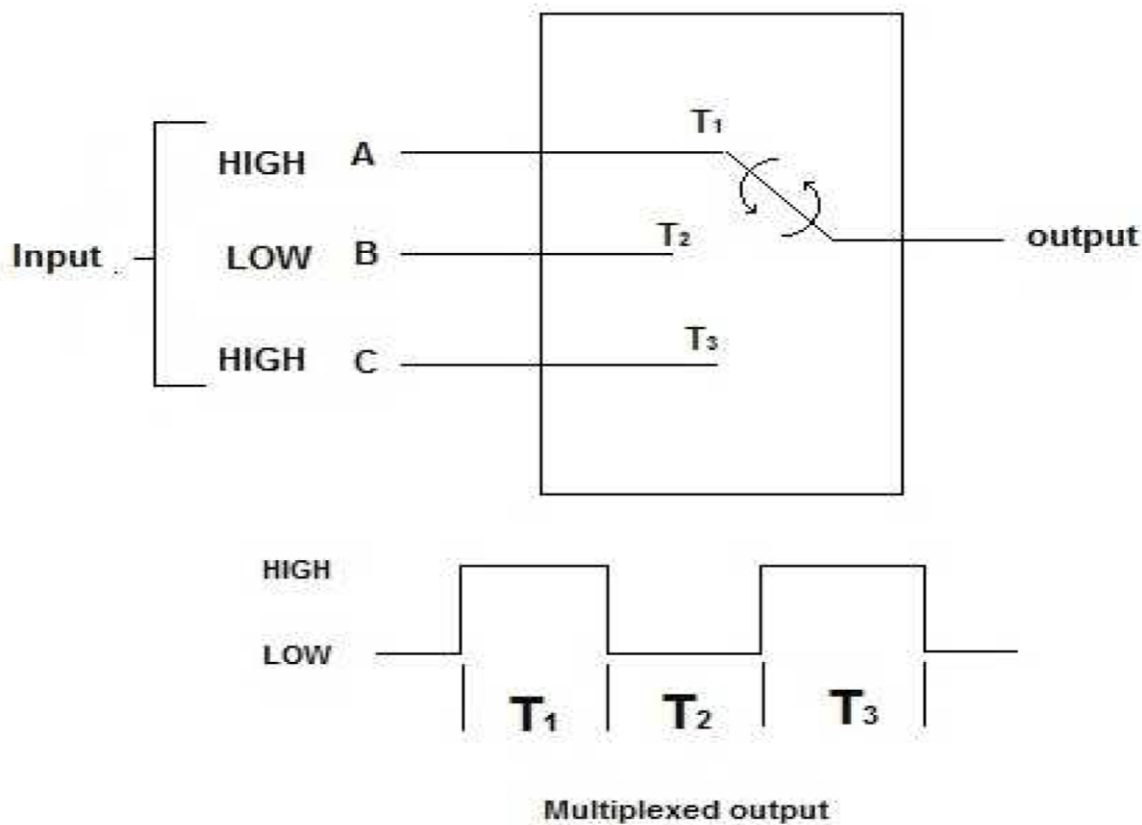
Figure 6.1 Simple Multiplexer operation

Assume that we have logic levels, an indicated on three inputs. During time interval T1, input A is connected to the output; during time interval T2, input B is connected to the output; during time interval T3, input C is connected to the output. The logic symbol for 4-input multiplexer is shown in Figure 6.2. Notice that there are

two selection lines because with two selection bits, each of the four data-input lines can be selected.
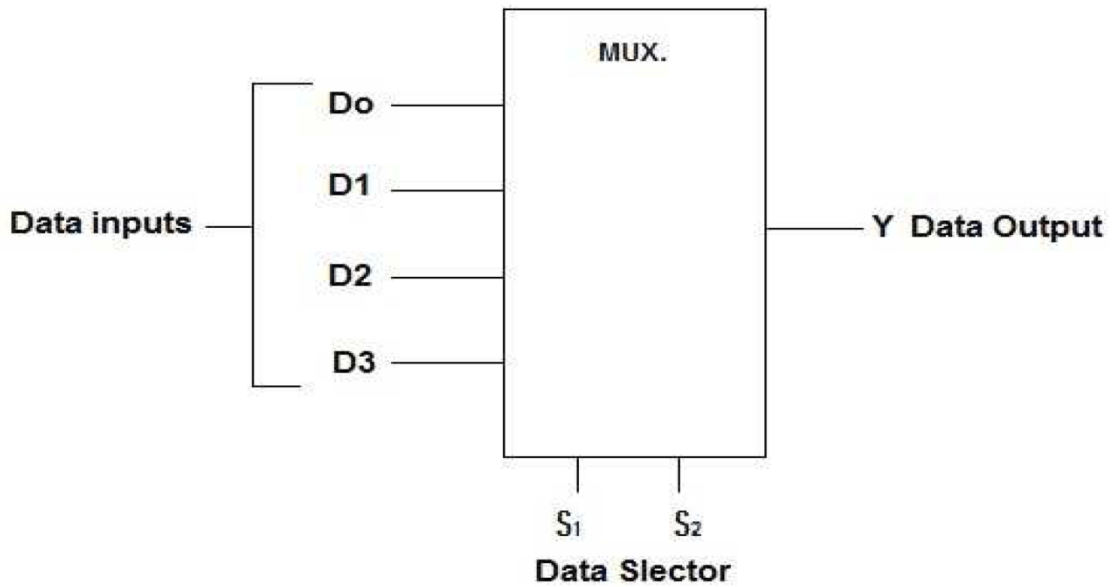
Figure 6.2 Logic symbol for 4-to-1 Data Selector

If a binary 0 ($S_1 = 0$ & $S_0 = 0$) is applied to the data-select lines, the data on input D_0 appears on the output. If a binary 1 ($S_1 = 0$ & $S_0 = 1$) is applied to the data-select lines, the data on input D1 appears on the output. If a binary 2 ($S_1 = 1$ & $S_0 = 0$) is applied to the data-select lines, the data on input D_2 appears on the output. If a binary 3 ($S_1 = 1$ & $S_0 = 1$) is applied to the data-select lines, the data on input D_3 are switched to the output. A summary of this operation is given in Table 6.1.

Data-Select Inputs		Input Selected
S_1	S_0	
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Table 6.1 Data Selection for a 4- input Multiplexer

The data output Y is equal to the data input D_0 if and only if $S_1 = 0$ & $S_0 = 0$: $Y = D_0 \cdot \overline{S_1} \cdot \overline{S_0}$

The data output Y is equal to the data input D_1 if and only if $S_1 = 0$ & $S_0 = 1$: $Y = D_1 \cdot \overline{S_1} \cdot S_0$

The data output Y is equal to the data input D_2 if and only if $S_1 = 1$ & $S_0 = 0$: $Y = D_2 \cdot S_1 \cdot \overline{S_0}$

The data output Y is equal to the data input D_3 if and only if $S_1 = 1$ & $S_0 = 1$: $Y = D_3 \cdot S_1 \cdot S_0$

There terms are (OR), the total expression for the data output is:

$$Y = D_0 \cdot \overline{S_1} \cdot \overline{S_0} + D_1 \cdot \overline{S_1} \cdot S_0 + D_2 \cdot S_1 \cdot \overline{S_0} + D_3 \cdot S_1 \cdot S_0$$

The implementation of the above equation is shown in Figure 6.3.
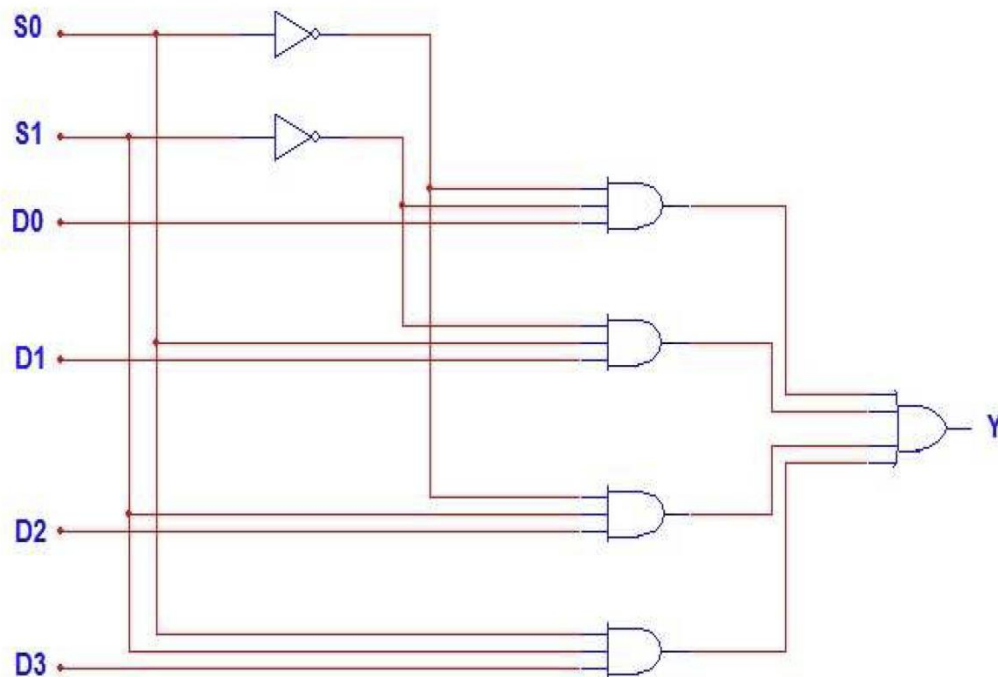
Figure 15.3 Logic Diagram for a 4-input multiplexer

B. Demultiplexer:

A demultiplexer (DMUX) basically reverses the multiplexing function. It takes data from one line and distributes them to a given number of output lines. Figure 6.4 shows a one-line to four-line demultiplexer circuit.
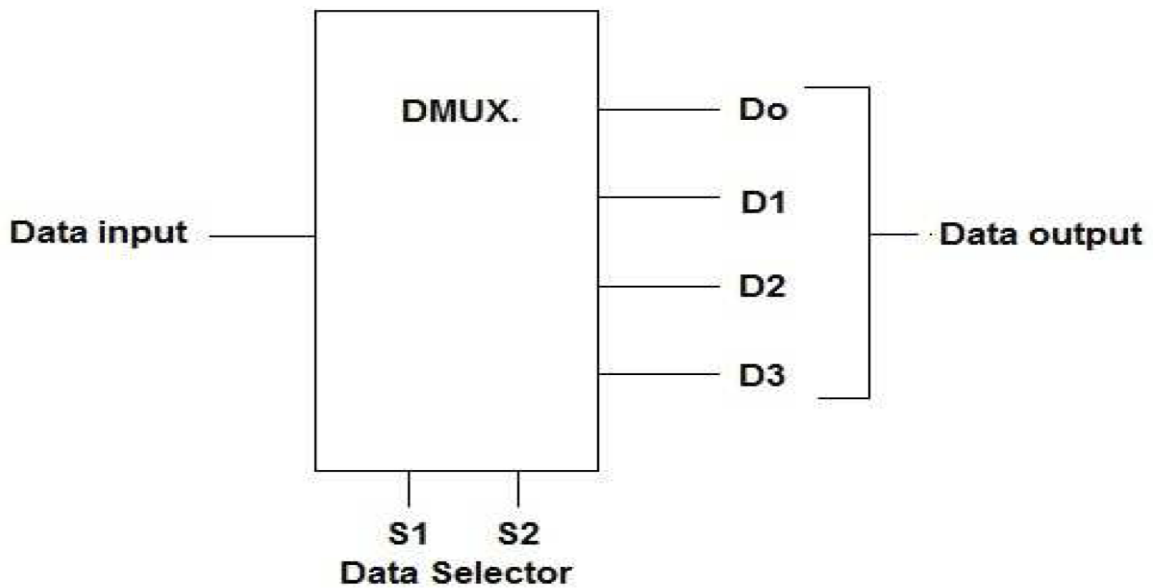
Figure 6.4 Logic Symbol 1-to-4 line Demultiplexer

The input data line goes to all AND gates. The two select lines will pass through the selected gate to the associated output line.

Figure 6.5 shows a 1-line to 4-line demultiplexer circuit. The two select lines enable only one gate at time, and the data appearing on the input line will pass through the selected gate to the selected gate to the associated output line.
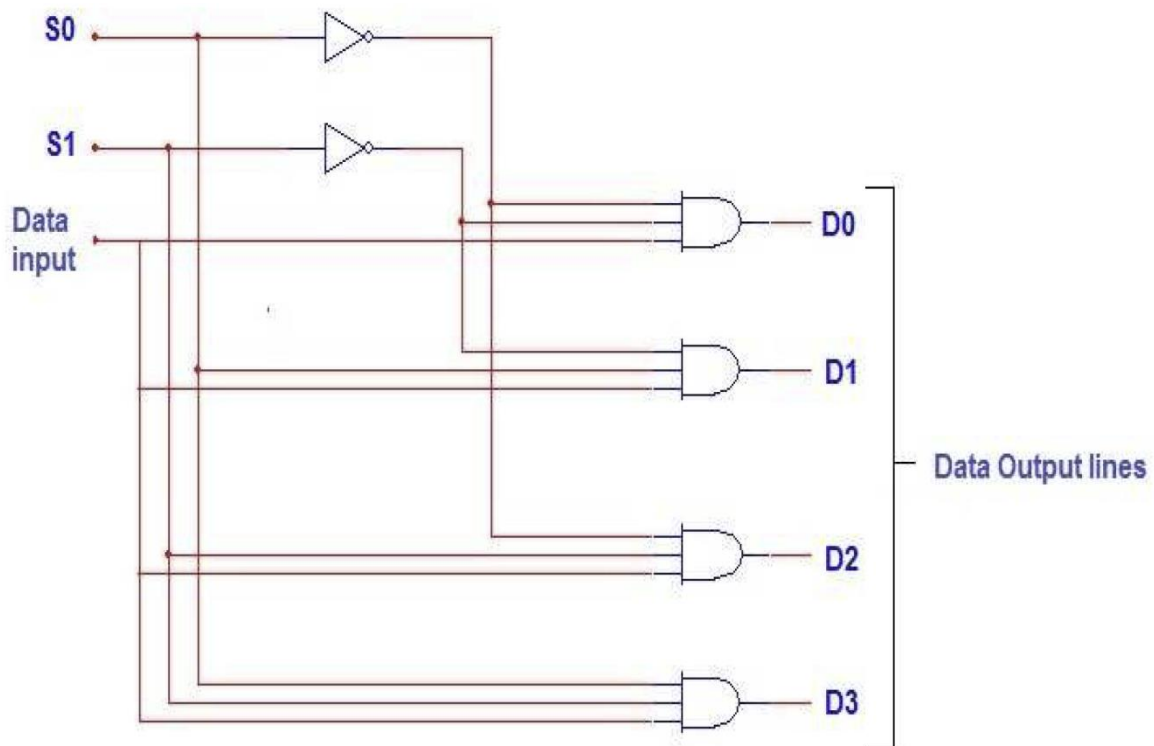
Figure 6.5 1-to-4 line Demultiplexer

Procedure:

A. Multiplexer:

    1. Connect a circuit of 2-to-1 multiplexer and observe its truth table.

    2. Connect a circuit of Figure 15.3 and observe its truth table.

B. Demultiplexer:

    Connect a circuit of Figure 15.5 and observe its truth table.

Discussion:

1. The data input and select waveforms in Figure 6.6 are applied to the multiplexer in Figure 6.3. Determine the output waveform in relation the input.
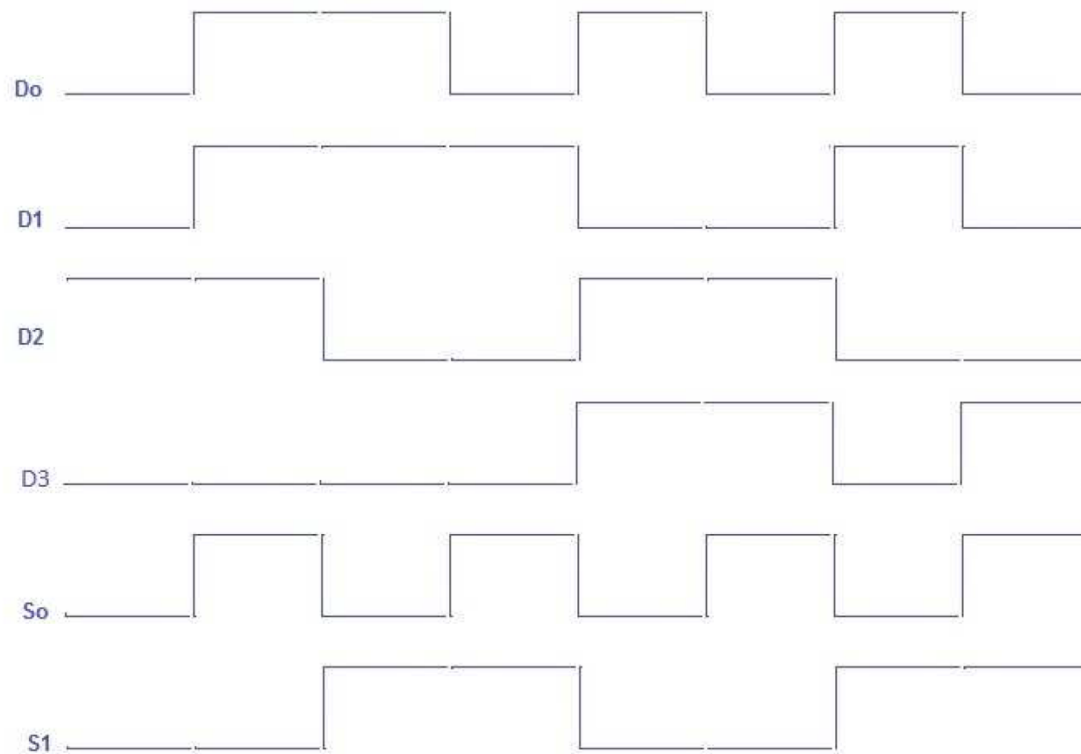
Figure 6.6

2. The serial data input waveform and data selectors are shown in Figure 6.7. Determine the data-output waveform for the demultiplexer shown in Figure 6.5.

Figure 6.7

3. Design 6-to-1 MUX. and verify its truth table.
4. Design 1-to-5 DMUX. and verify its truth table.

Experiment No. 7

Parity Checker and Generator

Introduction:

Errors can occur as digital codes are 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, a "1" can change to a "0", or a "0" to "1", due to co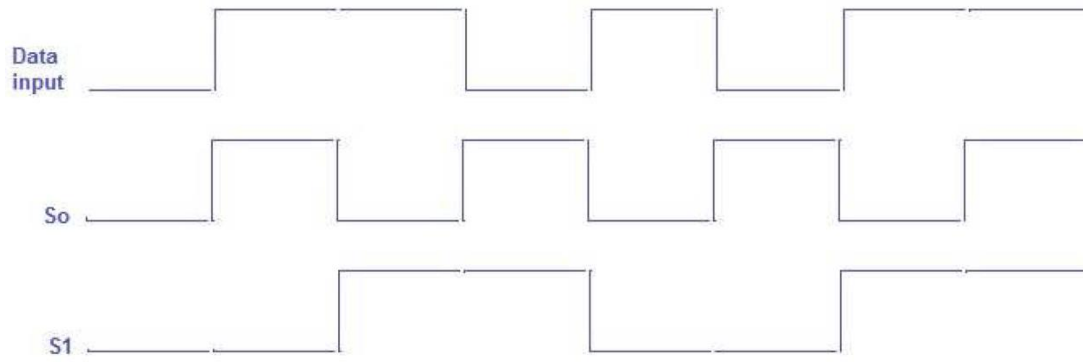mponent malfunction or electrical noise. Many systems, however, employ a parity bit as a means of detecting a bit error. A parity bit is scheme for detecting errors during transmission of binary information. An extra bit included with the binary massage to make the number of 1s either odd or even. The massage including, the parity bit, is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond to the one transmitted. The circuit that generates the parity bit in the transmitter is called a parity generator, the circuit that checks the parity in the receiver is called a parity checker.

The parity of a binary word can be determined with an XOR gates circuit. The Figure 7.1 shows the parity generator circuit at the transmitter end, and figure 7.2 shows the parity checker circuit at the receiver end.
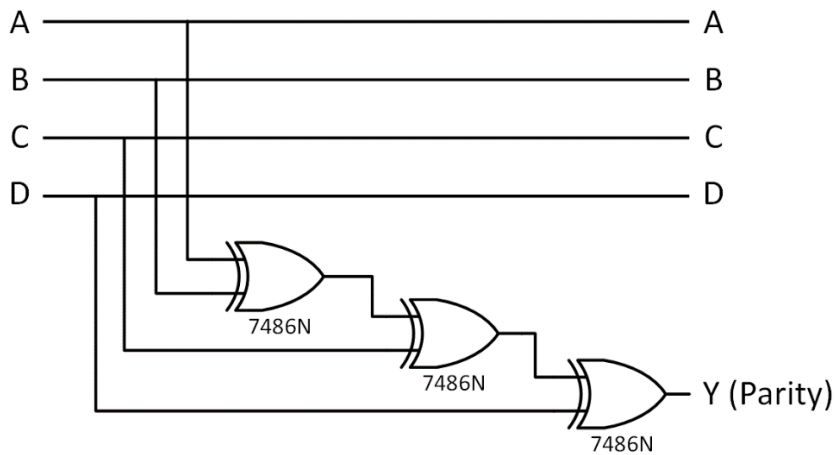
Figure 7.1 Parity Generator Circuit
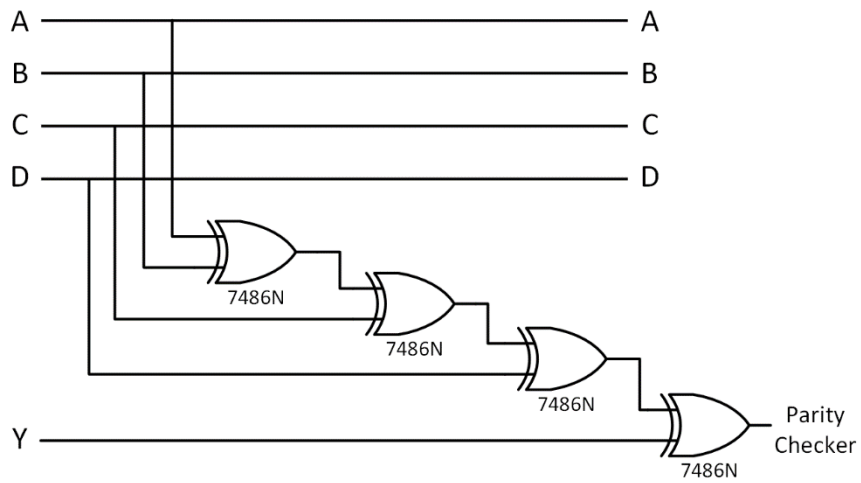
Figure 7.2 Parity Checker Circuit
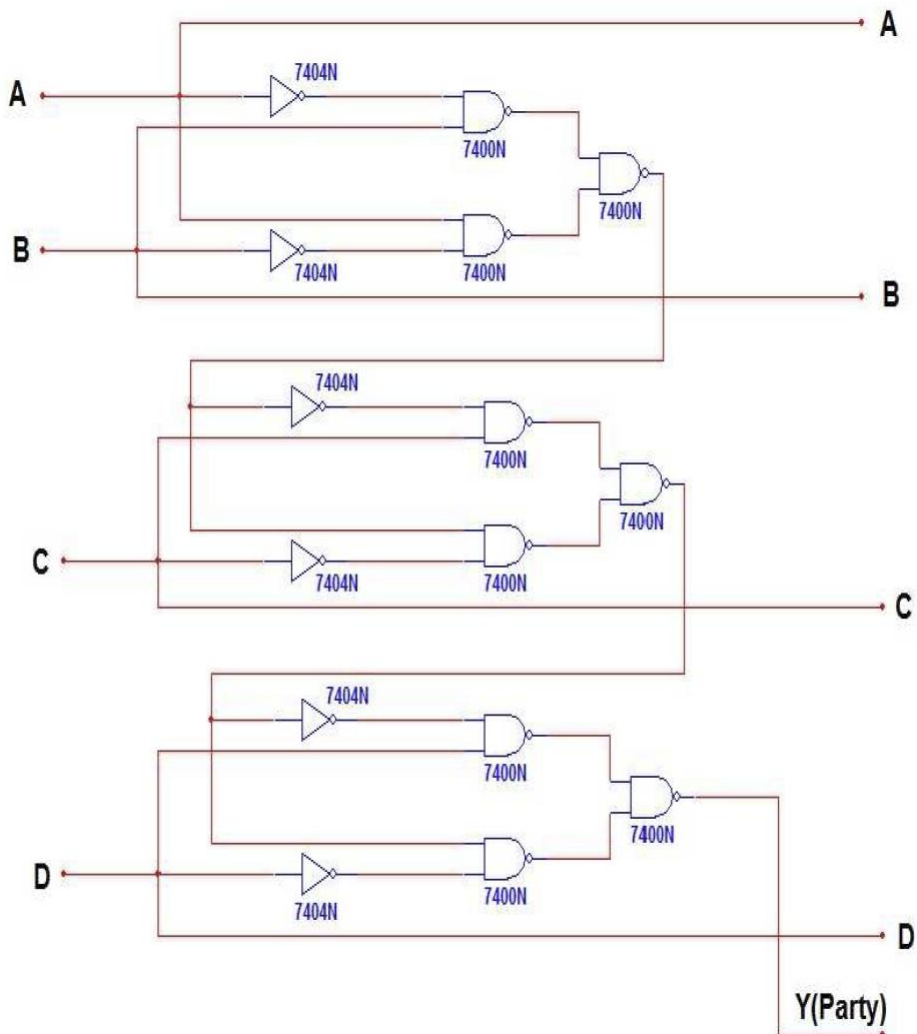
Figure 7.3 Parity Generator Circuit using NAND Gates

Procedure:

1. Connect the circuit shown in Figure 7.1.
2. Derive the truth-table.
3. Connect the circuit shown in Figure 7-2. Repeat step (2).
4. Connect the circuit shown in Fig 7.3. Repeat step (2).

Discussion:

1. Write down the Boolean equation for Figure 7.3.
2. Construct a truth-table and add to the binary code, the odd-parity bit. Draw the logic circuit using NOR gates.
3. Repeat step (2) for NAND gate.

Experiment No. 8
Binary-to-Gray and Gray-to-Binary Code Convertor

Introduction:

The gray code is an unweighted code not suited to arithmetic operation, but useful for input – output device, analog-to-digital converters, shaft encoder, and other peripheral equipment.

Table 8.1 shows the gray code, along with the corresponding binary numbers. The feature of gray code is that in progressing from one number to the next in sequence only one of the binary digits (bits) suffers a change.

Decimal	Binary				Gray			
	A	B	C	D	X_1	X_2	X_3	X_4
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
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

Table 8.1

Sometimes, we have to convert binary numbers into gray code numbers and vice versa. A Binary- Gray code converter is a device for converting a binary number applied at its input terminals into the equivalent gray code number at its output terminals as shown in Figure 8.1.

For instance, in converting from Binary digit (A). Add each pair of adjacent bits to get next gray digit as follows:

$$X_1 = A, \qquad X_2 = A + B, \qquad X_3 = B + C, \qquad X_4 = C + D$$

And to convert from gray code to binary we use the following formula:

$$A = X_1, \qquad B = A + X_2, \qquad C = B + X_3, \qquad D = C + X_4$$

Figure 8.2 and 8.3 shows a way to build a 4-bits binary-to-Gray and gray-binary converter by using exclusive-OR circuits.

Procedure:

1. Connect the circuit shown in figure 8.2.

2. Use all combinations of states of switches (A, B, C, and D) to get the data of table 8.1. Note that the lamps (X_1, X_2, X_3, and X_4) display the gray code number.

3. Re-arrange the circuit connected in step (1) to be as shown in figure 8.3. The circuit in this case will be a Gray-to-binary converter instead of binary-to-Gray converter.

4. Use all combinations of states of switches (X_1, X_2, X_3, and X_4) to get the data of table 8.1, and note that lamps (A, B, C and D) display the binary number.

Discussion:

1. Convert the following gray numbers into binary number:

   a) 10101   b) 100110011   c) 00111000100110

2. Convert the following binary numbers into gray number:

   a) 1010110   b) 1011001001   c) 10101101110011

3. Draw a logic circuit that converts 10-bits binary numbers into gray code numbers, use exclusive-OR gates.

4. Draw a logic circuit that converts 10-bits gray numbers into binary numbers, use exclusive-OR-gates.
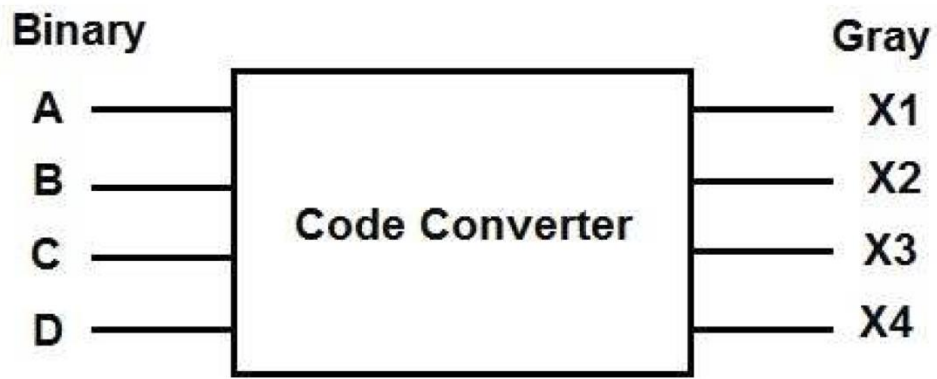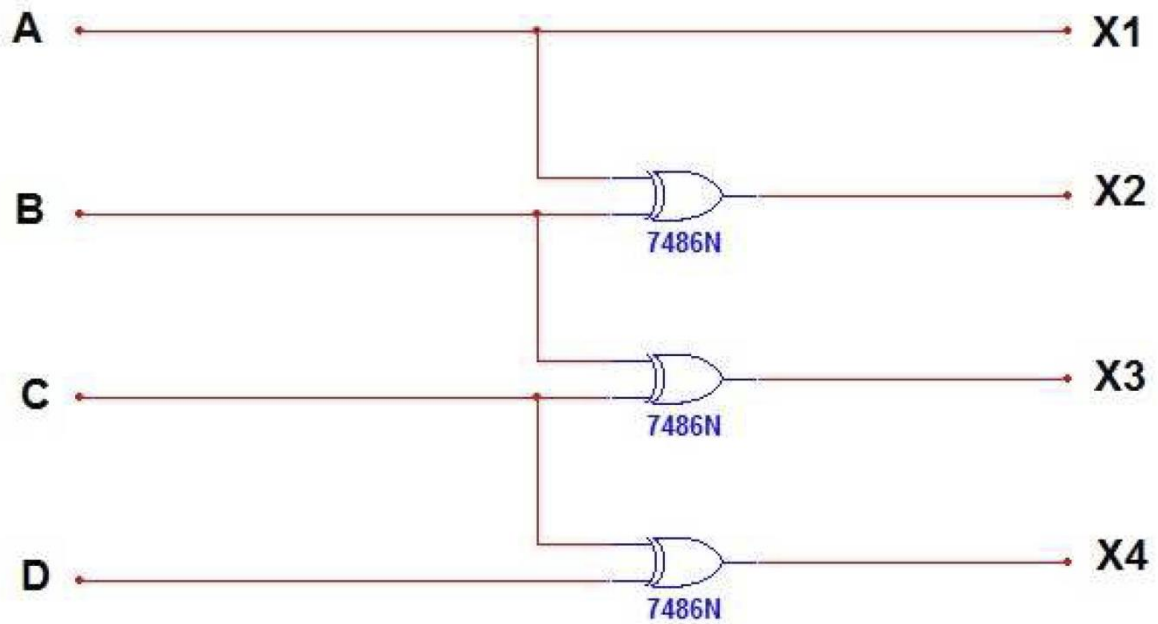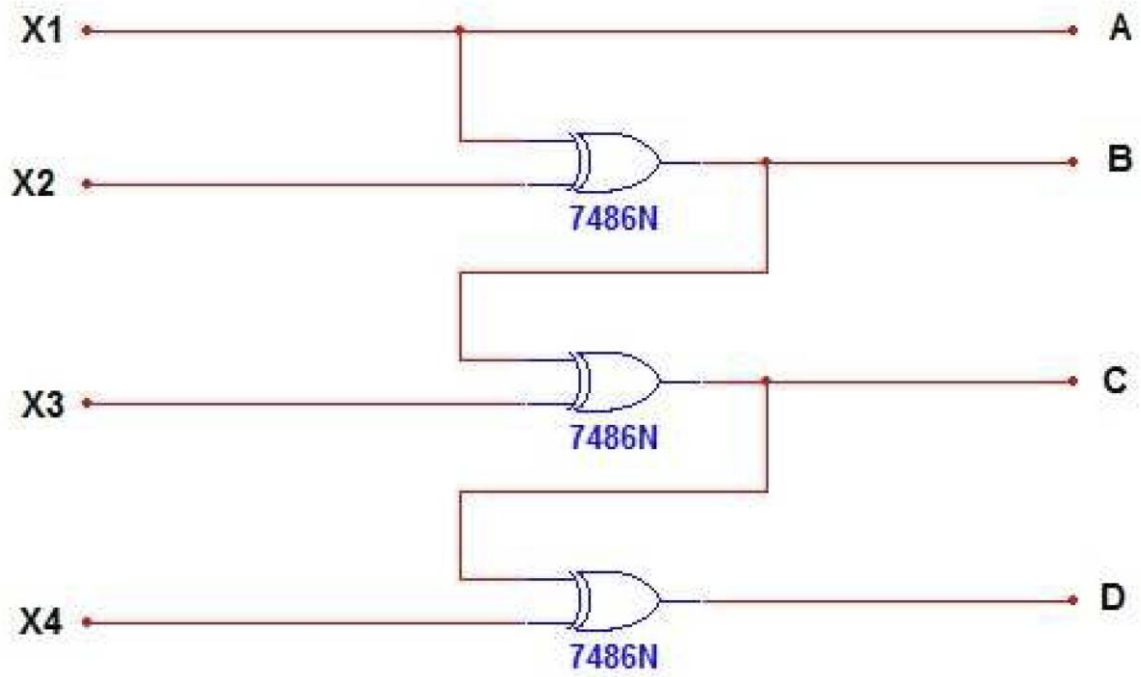
Figure 8.1

Figure 8.2 Binary-to-Gray Convertor

Figure 8.3 Gray-to-Binary Converter

Experiment No. 9
Controlled Invertors

Introduction:

   Another application of XOR gate is the controlled or programmed inverter. See Figure 9.1. When the input, control, is low, it transmits the binary word to the output (unchanged). But when, control, is high it transmits the 1s complement.
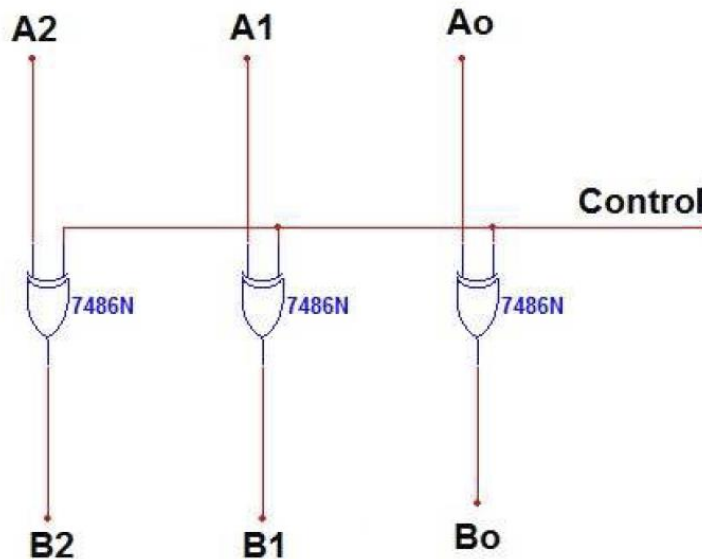

Figure 9.1 Controlled Inverter Circuit

Procedure:

   1. Connect the circuit shown in Figure 9.1.
   2. Derive the truth-table.

Discussion:

   1. Construct a controlled inverter circuit using NAND gates only.
   2. How many NOR gates do we need to construct a controlled inverter for 48-bit word.