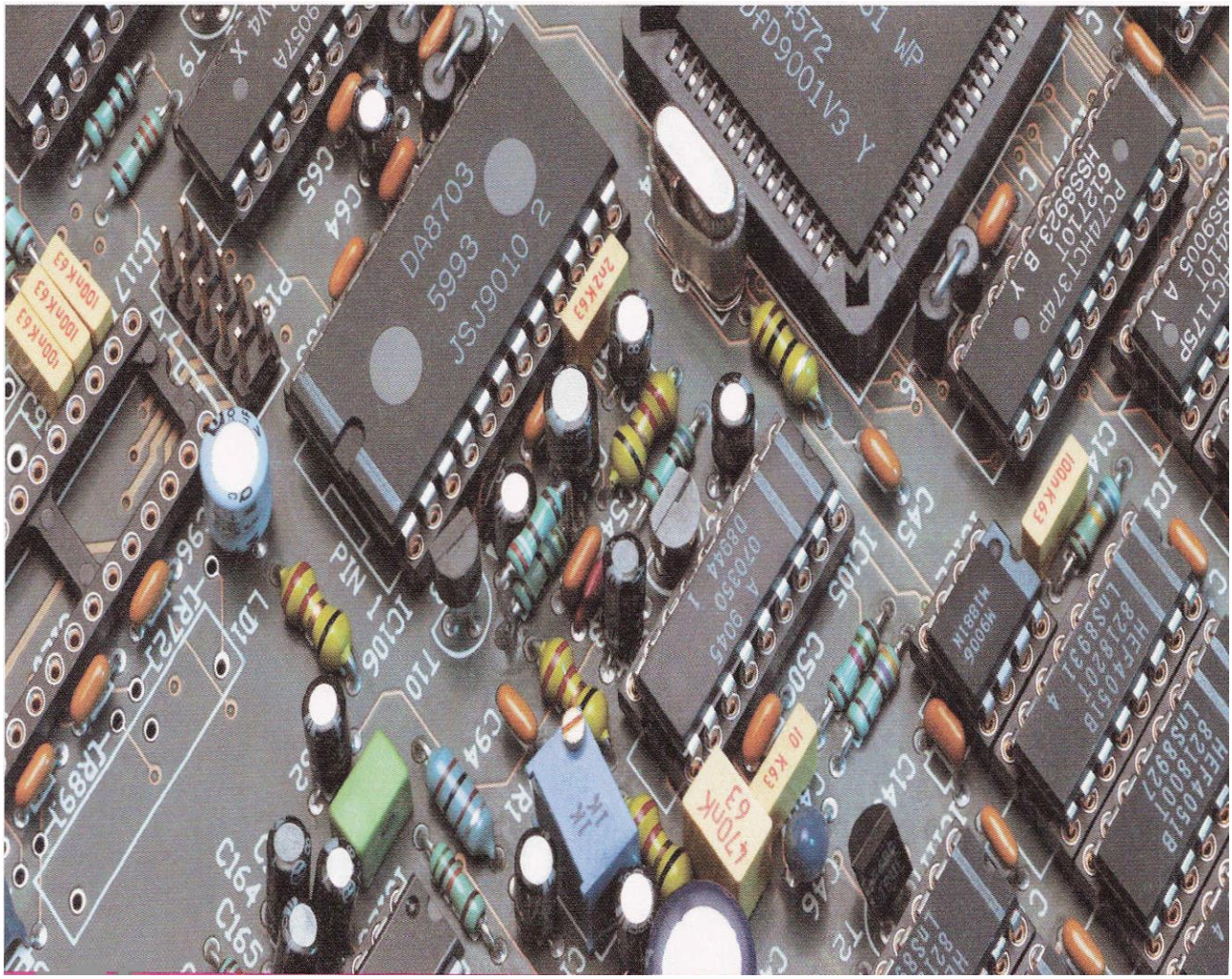
Mustansiriyah University
College of Engineering
Electrical Engineering Dept.

Fundamentals of Logic Circuits Lab.



Asst. Prof. Dr. Ammar Ghalib
Asst. Prof. Dr. Zaid Saleem

University of Mustansiriyah
Faculty of Engineering
Electrical Engineering Department

Fundamentals of Logic Circuits **Code: EE206** **Theoretical: 2 Hrs/Wk** **Tutorial: 0 Hrs/Wk** **Practical: 2 Hrs/Wk**
Introduction to Digital Techniques.
System of Numbers: General number formula: Binary, octal, decimal & hexadecimal numbers.
Numbers Base Conversions: Arithmetic operation in different numbers, complements, binary codes, BCD, Ex-3, gray codes.
Boolean Algebra: Basic definitions, basic theorem & properties, Boolean functions.
Canonical standard forms and logic gates.
Karnaugh Maps: And & OR Implementation, don't care condition.
Adders Arithmetic Operations: Half & Full Adders, Half & Full Subtractors, Binary Parallel Adders.
Code Conversion: Even and Odd party logic, decoder, encoder, comparator, multiplexers & demultiplexer.
Sequential Circuits: Introduction to sequential circuits, basic flip-flops, flip-flops excitation table, converting of flip-flops, analysis of clocked sequential circuits.
Counters: Asynchronous counter, synchronous counter operation, design of synchronous counter, properties of synchronous counter.
Registers, Shift registers, and Bidirectional universal shift register.
Shift Register Counters: Ring counter and Johnson counter.
Total hours per year (60 Theoretical + 60 Practical)

LOGIC GATES

Object: To perform the functions of gates

Theory:

A logic gates is an electronic device that perform a Boolean operations on one or more inputs to produce an output. There are 4 types of names have been used for the same types of circuits: digital circuit, switching circuit, Logic circuit. AND gates. Binary logic deals with variables that take on tow discrete values and with operations that assume logical meaning. The two values the variables take may be called by different names (e.g. true and false, yes and no, 1 or 0).

Logic Gates:

1. AND gate: the AND gate is a cct., which gives a high output (logic 1) if all inputs are high. A dot (.) is used to indicate the AND operation. In practice, however, the dot is usually omitted.

2. OR gate: the OR gate is a cct., which gives a high output if one or more of its inputs are high. A plus sign (+) is used to indicate the OR operation.

3. NOT gate: the NOT gate is cct., which produces at its output the negated (inverted) version of its input logic the cct. Is known as an inverter. If the input is A, the inverted output is written as $\bar{A}$.

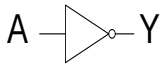
4. NAND gate: the NAND gate is a NOT- AND cct., which is equivalent to an AND cct. Followed by a NOT cct. The output of the NAND gate is high if any of its inputs is low.

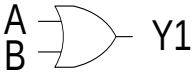
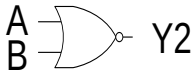
5. NOR gate: the NOR gate is a NOT- OR cct., which equivalent to AND cct. Followed by a NOT cct. The output of the NOR gate is low if any of its inputs is high.

6. EX-OR gate: the exclusive –or gate is a cct., which gives a high output if its tow inputs are different. A circuited plus sign $\oplus$ is used to indicate the (EX-OR) operation.

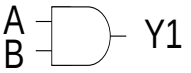
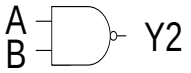
7. EX-NOR gate: the exclusive–NOR gate is a cct., which gives a high output if its tow inputs are similar.

The seven gates that are the fundamental logic elements in digital system are illustrated in Fig. (1.1)

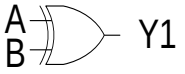
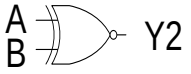
<table>
<tr><th>Logic Function</th><th>Logic Gate Symbol</th><th>Truth Table</th><th>Boolean Expression</th></tr>
<tr><td>NOT</td><td>  </td><td><table><tr><th>Input (A)</th><th>Output (Y)</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table></td><td>$Y=\overline{A}$</td></tr>
<tr><td>OR

NOR</td><td> 
  </td><td><table><tr><th colspan="2">Input</th><th colspan="2">Output</th></tr><tr><th>A</th><th>B</th><th>Y1</th><th>Y2</th></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>0</td></tr></table></td><td>Y1= A+B

$Y2=\overline{A+B}$</td></tr>
<tr><td>AND

NAND</td><td> 
  </td><td><table><tr><th colspan="2">Input</th><th colspan="2">Output</th></tr><tr><th>A</th><th>B</th><th>Y1</th><th>Y2</th></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td><td>0</td></tr></table></td><td>Y1= A.B

$Y2=\overline{A.B}$</td></tr>
<tr><td>EX-OR

EX-NOR</td><td> 
  </td><td><table><tr><th colspan="2">Input</th><th colspan="2">Output</th></tr><tr><th>A</th><th>B</th><th>Y1</th><th>Y2</th></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td><td>1</td></tr></table></td><td>$Y1= A\oplus B$

$Y1=\overline{A\oplus B}$</td></tr>
</table>

Fig.(1.1) The fundamental of basic logic gates

The function of the gates described so far can be summarized by means of the idealized waveform diagrams shown below:

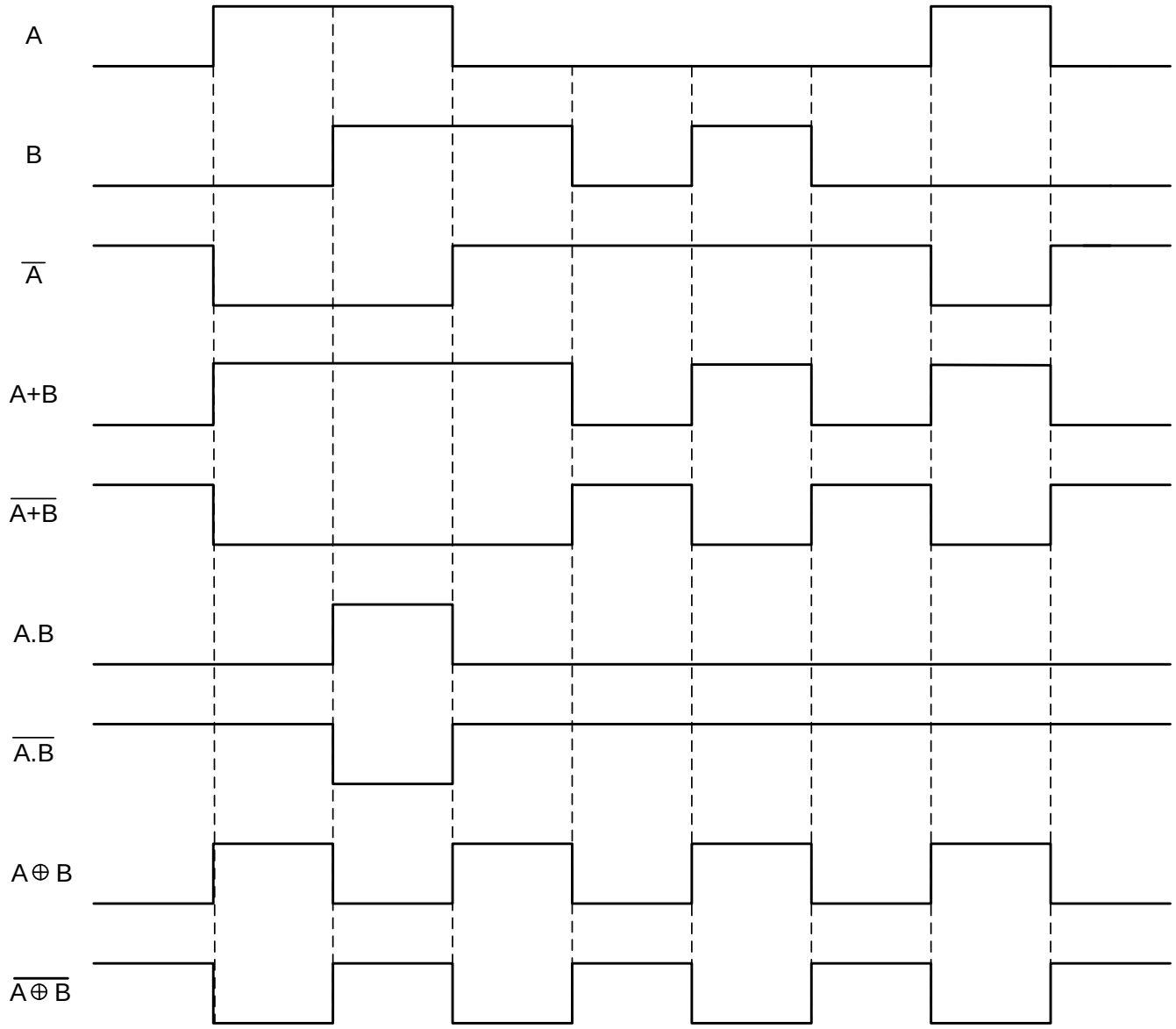
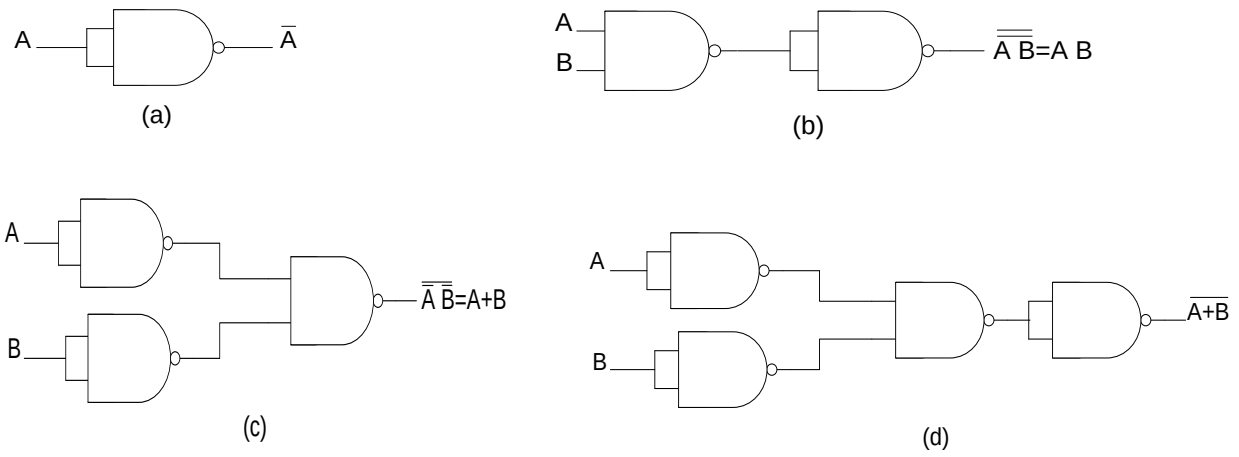


Fig.(1.2) Idealized waveforms diagram for two-inputs gates positive logic

Through there is a large variety of gates, it is often desirable to convert logic expression into a form suitable for whatever type of gates are available, its desirable to use one type of gates rather than mix different gates. So it is easy to design logic cct. using NAND or NOR gates only. NAND gates can be used to produce any logic function .For this reason, they are referred to as universal gates.

The NAND gate can be used to generate the (NOT, AND, OR, NOR) functions. An inverter can be made from NAND gate by connecting all inputs together and creating, in effect, a single common input as shown in Fig.(1.3-a) for two input gate. An AND function can be generate using NAND gates, as shown in Fig. (1.3-b) also an OR function can be produce with NAND gates as illustrated in Fig. (1.3-c).Finally, NOR function is produced, as shown in Fig. (1.3-d).



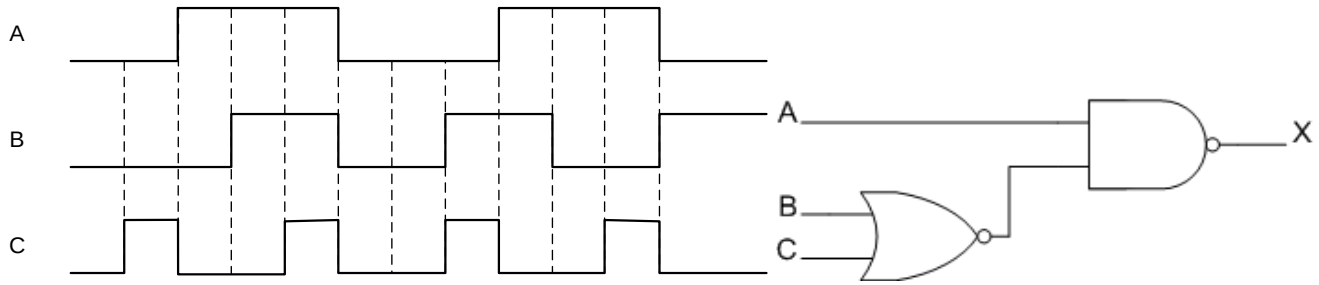
Fig(1.3) Universal application of NAND gates

Procedure:

By means of using NAND gates, find the truth table of all possible gates shown in Fig. (1.1).

Discussion:

1. Implement the following functions using:
    a. Mixing gates.
    b. NAND gates only.
        - F1=ABC+CD
        - F2= $\overline{AB}$ +BCD+EFGH
2. Implement the following functions using:
    a. Mixing gates.
    b. NOR gates only.
        - F1=A+ $\overline{B}$
        - F2=(A+B)(A+C)
3. Using only NOR gates to produce the logic functions of :
    a. NOT gate
    b. OR gate
    c. AND gate
    d. NAND gate
4. Determine the output waveform for the cct. Shown in Fig. (1.4) with the inputs as shown.



Fig(1.4) Testing a logic circuit using three-inputs

COMBINATIONAL LOGIC CIRCUITS

Object:

To study the axioms defining Boolean algebra and how to represent Boolean expressions in SOP and POS forms.

Theory:

Digital systems are composed of combinations logic gates described by a truth table and Boolean expression or a logic symbol diagram.

The fundamental Boolean operations of AND, OR and NOT can be summarized as follows:

$A + B = B + A$	$A \cdot B = B \cdot A$
$A + (B + C) = (A + B) + C$	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$
$A \cdot (B + C) = A \cdot B + A \cdot C$	$A + B \cdot C = (A + B) \cdot (A + C)$
$A + 0 = A$	$A \cdot 1 = A$
$A + 1 = 1$	$A \cdot 0 = 0$
$A + A = A$	$A \cdot A = A$
$A + \overline{A} = 1$	$A \cdot \overline{A} = 0$
$A + \overline{A} \cdot B = A + B$	$A \cdot (A + B) = A$
$A \cdot B + A \cdot C = A \cdot (B + C)$	$(A + B) \cdot (A + C) = A + B \cdot C$

In combination logic, the output of the circuit depends only on the inputs to the circuit. Combination logic problems are normally given in the form of logical statements or a truth table. To design and implement the problem, Boolean logical expressions equations are derived for the output logic function in terms of the binary variables representing the inputs. The logic expressions are given either in the form of a sum of products (SOP) of in the form of a product of sums (POS).

CANONICAL FORMS:

For the table shown below:

Input X Y Z	Output F
0 0 0	1
0 0 1	0
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	0
1 1 0	1
1 1 1	1

We can derive the logical expression for the function F:

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

This expression is called the canonical sum of product. A product term which contains each of the n-variables factors in either complemented or uncomplemented form called minterm. So F can be put in other form such as:

$$F = \sum 0,2,3,6,7$$

Since F=1 in rows 0, 2,3,6,7

A logical equation can also be expressed as a product of sums. This done by considering the combinations for which F=0. From the truth table F=0 in rows 1, 4, 5 hence:

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

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

$$F = (X + Y + \overline{Z}) \cdot (\overline{X} + Y + Z) \cdot (\overline{X} + Y + \overline{Z})$$

The product of sums can be expressed as:

$$F = \prod 1,4,5$$

A sum which contains each of n variables complement not is called a maxterm.

Procedure:

1. Using Boolean algebra to simplify the following expressions. Then find the truth table of each after connecting the circuits.
   a. $F_1 = A \cdot B + A \cdot (B + C) + B \cdot (\bar{B} + C)$
   b. $F_2 = (A + \bar{B} + A \cdot B) \cdot (\bar{A} + \bar{B})$
   c. $F_3 = (A \oplus B) \oplus (B \oplus C)$
   d. $F_4 = (A \odot B) \oplus (B \odot C)$

2. Obtain F_1&F_2 in the form of

   a. SOP

   b. POS

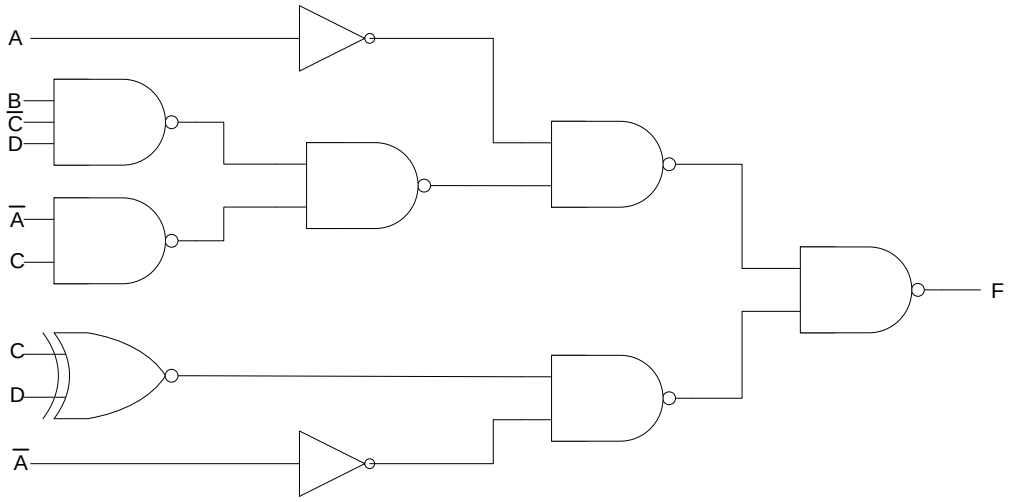
A B C	F_1	F_2
0 0 0	1	1
0 0 1	1	0
0 1 0	1	1
0 1 1	1	0
1 0 0	0	1
1 0 1	1	0
1 1 0	0	1
1 1 1	1	0

Then simplify F_1&F_2 and connect them circuits to verify the operation of both, using NAND gates only.

Discussion:

1. Simplify the following logical expression and implement them using suitable logic gates.
   a. $F_1 = \sum 2,4,6,10,14$
   b. $F_2 = \prod 2,3,6$
2. Determine whether or not the following equalities correct:
   a. $A + B \cdot C + \bar{A} \cdot C = B \cdot \mathrm{C}$
   b. $\bar{B}(A \odot \mathrm{C}) + B \cdot \bar{C} + A(B \odot \mathrm{C}) = \overline{\bar{A}C}$
3. Convert the following expressions to SOP forms:
   a. $(A + \bar{B} \cdot C) \cdot B$
   b. $(A + C)(\bar{A} \cdot B \cdot \bar{C} + A \cdot \mathrm{C} \cdot \mathrm{D})$
4. Write a Boolean expression for the following statement :
   F is a "1" if A, B&C are all 1's or if only two of the variable is a"0".

5. Find **F** for the following Figure.



CODE CONVERSION

Object: To consider various important codes and the logic for converting from one to another.

Theory:

As we know, decimal, octal, and hexadecimal numbers can be represented by binary digits. Not only numbers, but letters and other symbols, can be represented by 1's and 0's.

In fact, any entity expressible as numbers, letters, or other symbols can be represented by binary digits, and therefore can be processed by digital logic circuits.

Combination of binary digits that represent numbers, letters, or symbols are digital codes. In many applications special codes are used for such auxiliary functions as error detection.

BCD code:

To code the ten decimal digits, ten unique symbols consisting of the binary digits 0&1are needed. A code of this type, which represents the decimal digits with binary digits, is called a binary- coded decimal, or BCD. There are several such codes in use. The BCD code of a decimal number of more than one digit is obtained by replacing each digit by it's4 -bit BCD code. In general, any binary code used to represent the decimal digit is called BCD. Table below shows the most general binary codes.

Procedure:

1-Design a logic circuit to convert BCD code to EX-3code using NAND gates only.

2-Use K-map to design a logic circuit to convert from 5421 code to 8421 code using NAND gates to check the logic design.

3-Design a logic circuit to obtain a Gray code from a BCD code using proper logic gates.

Decimal	Weighted codes				Unweighted code	
	8 4 2 1	2 4 2 1	5 2 1 1	7 4 2 1	Ex-3	Gray
0	0000	0000	0000	0000	0011	0000
1	0001	0001	0001	0001	0100	0001
2	0010	0010	0100	0010	0101	0011
3	0011	0011	0110	0011	0110	0010
4	0100	0100	0111	0100	0111	0110
5	0101	1011	1000	0101	1000	0111
6	0110	1100	1001	0110	1001	0101
7	0111	1101	1011	1000	1010	0100
8	1000	1110	1110	1001	1011	1100
9	1001	1111	1111	1010	1100	1101

Table (3.1) Binary codes

Discussion:

1. Convert each Gray code to binary:

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 with an output (F) and four bit input A, B, C&D. The output (F=1) when the input is BCD number and (F=0) otherwise.

4. Design a logic circuit which converts a BCD code to 6311 code using NAND gates only.

BASIC ARITHMETIC OPERATIONS

Object: To design and implement logic circuit for basic arithmetic operations.

Theory:

An important part of the central processor of any computer is the arithmetic unit in which binary addition, subtraction, division and multiplication are carried out.

Subtraction however can be performed by adding complemented numbers. Multiplication can also be performed by repeated addition .Division can be also achieved by repeated subtraction. This means that the adder is the centre piece of the arithmetic unit. There are two types of the addition:

1. ***Half - Adder (H.A) :***

It is a device that adds two bits of binary data. In other words, the half adder performs the operation s:

$0 + 0 = 0$

$0 + 1 = 1$ ……. (4.1)

$1 + 0 = 1$

$1 + 1 = 0$, Carry = 1

The last operation is, of course, 1+1=0, which is 0 with a carry 1 to the next bit position. Equation (4.1) may be expressed n the form of a truth table as shown in table (4.1)

Input		Output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

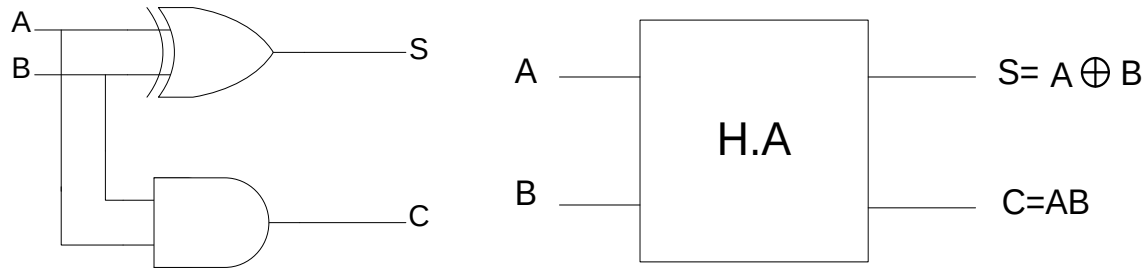
From the truth table we see that

$$S = A\bar{B} + \bar{A}B \qquad \text{…….. (4.2)}$$

$$= A \oplus B$$

and $$C = A.B \qquad \text{…….. (4.3)}$$

So the H.A adds only two bits at a time, so that it cannot be used to add two bits and a carry bit from a previous step, as is generally required in adding tow binary numbers the symbol for the H.A is given in Fig (5.1.b).



(a) (b)

Fig. (4.1) (a) Half Adder circuit diagram

(b) Half Adder block diagram

2. ***Full- Adder (F.A):***

A half adder is not very useful on its own, and a third input is often required for carries. Adding numbers that have two bits or more requires a full adder (F.A) which is capable of the previous order. The symbol of full-adder is shown in Fig. (4.2).

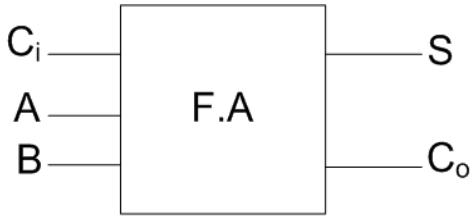


Fig. (4.2) Full –adder block diagram

Where:

C_i: carry- in from the previous addition.

C_o: carry- out to the next addition.

The truth table for a full-adder is (F.A) is determined by the 8 possible combinations of the inputs A, B and C_i, the corresponding values of S and C_o is given in table (4.2) from which we may write

$$S = \overline{A} \cdot \overline{B} \cdot C_i + \overline{A} \cdot B \cdot \overline{C_i} + A \cdot \overline{B} \cdot \overline{C_i} + A \cdot B \cdot C_i \quad \ldots\ldots\ldots \quad (4.4.a)$$

$$= \left(\overline{A} \cdot \overline{B} + A \cdot B\right) \cdot C_i + (\bar{A} \cdot B + A \cdot \bar{B}) \cdot \overline{C_i} \quad \ldots\ldots\ldots \quad (4.4.b)$$

$$C_o = \overline{A} \cdot B \cdot C_i + A \cdot \bar{B} \cdot C_i + A \cdot B \cdot \overline{C_i} + A \cdot B \cdot C_i \quad \ldots\ldots\ldots \quad (4.5.a)$$

$$= A \cdot B + C_i(A \oplus B) \quad \ldots\ldots\ldots \quad (4.5.b)$$

Input	Output	
A B C_i	S	C_o
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

Procedure:

A. Half-Adder (H.A)

1. Implement a H.A logic equation for sum and carry using NAND gates only then verify the truth table.
2. Design a Half-Subtractor (H.S) network, and verify its truth table.

B. Full-Adder (F.A):

1. Verify the truth table of F.A by means of using NAND gates only.
2. Design a Full-Subtractor (F.S) network, and verify its truth table.

Discussion:

1. By means of H.A block diagram build a F.A.
2. By means of H.S block diagram build a F.S.
3. Build a H.A using NOR gates only.
4. Use only two 2-inlput EX-OR gates and three 2-input NAND gates to build F.A.
5. Use the block diagrams of F.A to show the addition process of the binary numbers110&111.
6. By means of F.A block diagram, EX-OR gates and external switch x, design a 4-bit adder/subtractor.
7. What is meant by Parallel binary adders? For the parallel adder shown in Fig (4.3), determine the sum by analysis of the logical operation of the circuit.

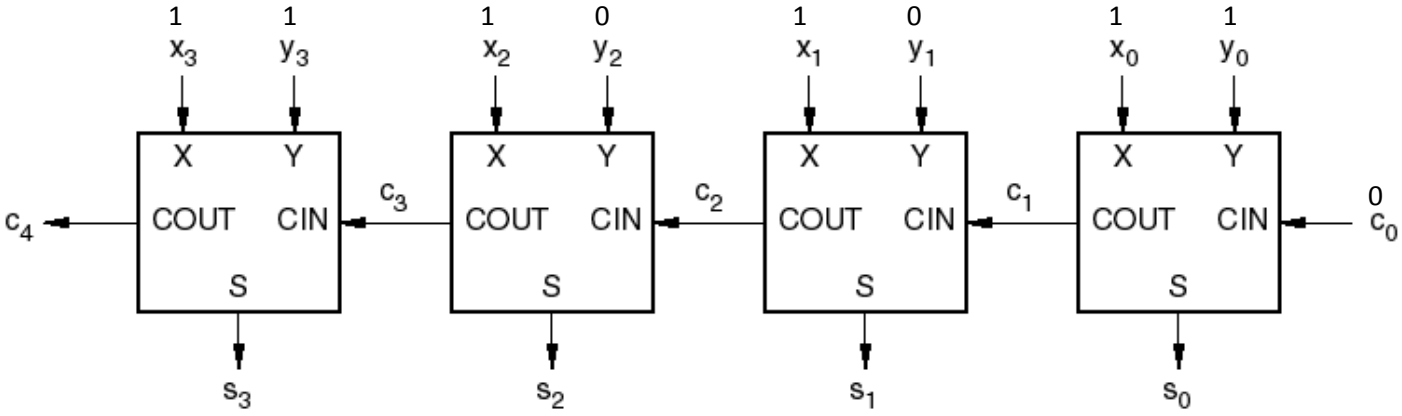


Fig. (4.3)

COMPARATORS

Object: To study the operation of magnitude comparator.

Theory:

A magnitude comparator is a combinational circuit that compares two numbers A and B and determines their relative magnitudes. The outcome of comparison is specified by three binary variables that indicate whether A>B, A=B, or A< B. the EX-OR gate is a basic comparator because it's output is 1 if it's two input bits are not equal and is 0 if the inputs are equal. Fig (5-1) shows the EX-OR as a 2-bit comparator.

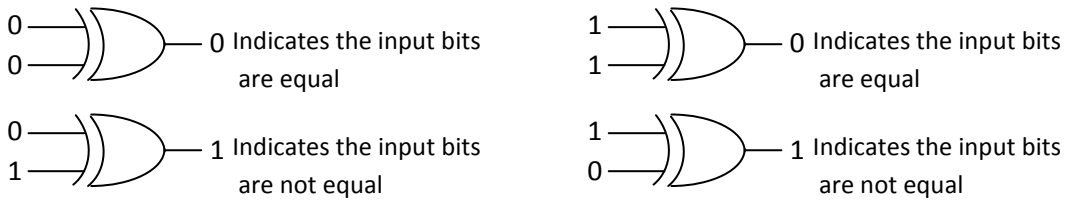


Fig.(5-1) Basic comparator operation.

The circuit for comparing two n-bit numbers has 2^{2n} entries in the truth table, and becomes too cumbersome even with n=3. Table (5-1) shows how to compare two numbers having 1-bit.

INPUTS		OUTPUTS		
A	B	Z1 A=B	Z2 A>B	Z3 A<B
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

Table (5-1)

From table (5-1), using minterms, we see that:

$$Z1 = A.B + \overline{A}.\overline{B}$$

$$Z2 = A.\overline{B} \qquad \text{……. (5-1)}$$

$$Z3 = \overline{A}.B$$

From these expressions we may obtain the digital circuit by using AND, OR, and NOT gates. The result is shown in Fig (5-2).

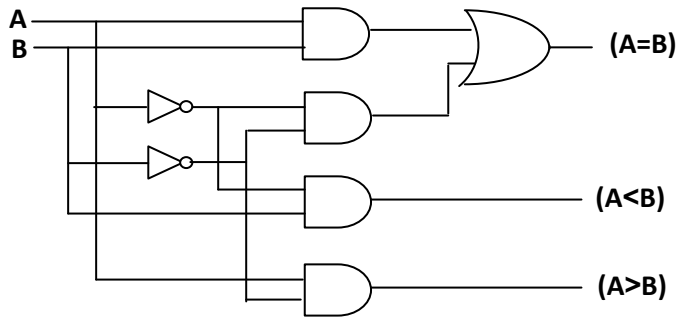


Fig.(5-2) one digit comparator.

The general algorithm for designing a n-bit comparator has the following steps (i.e. 4-bit)

a) Write the coefficients of the number as follows:

$A=A_3\ A_2\ A_1\ A_0$

$B=B_3\ B_2\ B_1\ B_0$

Where each subscribed letter represent one of the digits in the number.

b) For A=B ($A_3=B_3$, $A_2=B_2$, $A_1=B_1$, $A_0=B_0$), this can logically be expressed with an equivalence function:

$$x_i = A_i.B_i + \overline{A_i}.\overline{B_i} \qquad i=0, 1, 2, 3,\ldots., n \qquad \ldots\ldots. (5\text{-}2)$$

Where $x_i=1$ only if the pair of bits in position i are equal.

In order to determine whether A>B or A<B, compare the relative magnitudes of pair of significant digits starting from the MSB position. If the two digits are equal, we compare the next lower significant pair of digits. This comparison continues until a pair of unequal digits is reached. If the corresponding digit of A is 1 and of B is 0, we conclude that A>B. if the corresponding digit of A is 0 and that of B is 1, we have A<B.

The sequential comparison can be expressed logically by;

$$(A>B)= A_3\overline{B_3} + X_3A_2\overline{B_2} + X_3X_2A_1\overline{B_1} + X_3X_2X_1A_0\overline{B_0}$$

$$(A<B)= \overline{A_3}B_3 + X_3\overline{A_2}B_2 + X_3X_2\overline{A_1}B_1 + X_3X_2X_1\overline{A_0}B_0 \qquad \ldots\ldots. (5\text{-}3)$$

From these expressions, we may obtain the digital comparator circuit as shown in Fig. (5-3).

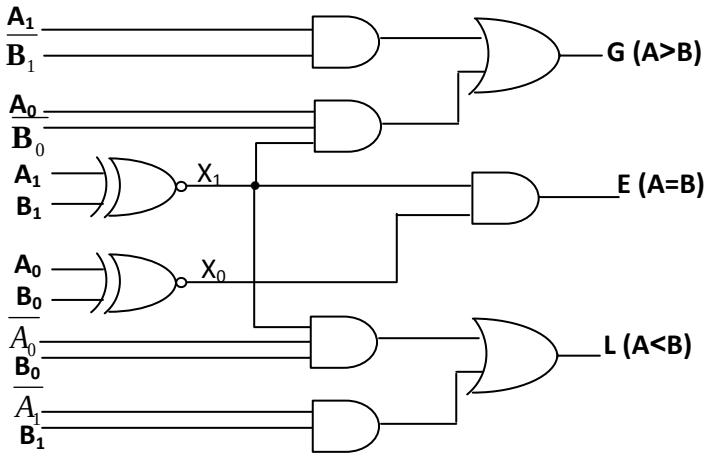


Fig.(5-3) Digital comparator circuit for 2-digit

Procedure:

1. Design a digital comparator which compares two binary numbers each with 2-bit, using truth table method.
2. The waveforms shown in Fig. (5-4), are applied to the comparator, find the output waveforms?

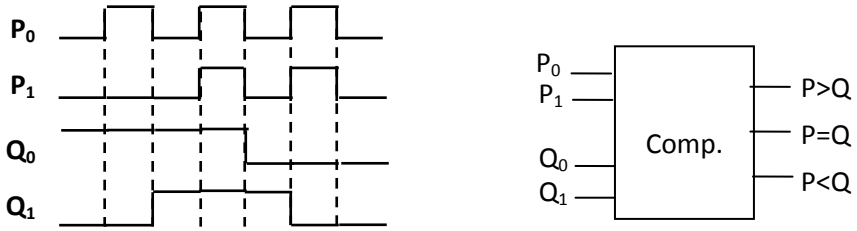


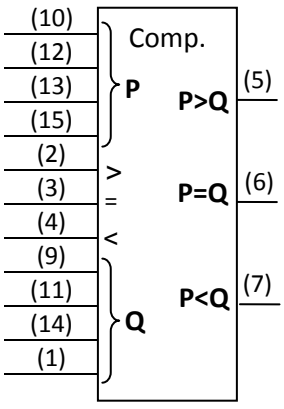
Fig. (5-4)

Discussion:

1. Given the logic symbol for the 7485 4-bit comparator. Use it to compare the magnitudes of two binary numbers of:

   a) 8-bit

   b) 12-bit



2. For each of the following set of binary numbers, determine the logic states at each point in the logic symbol of 7485 4-bit comparator.

   a) $P_3\ P_2\ P_1\ P_0$=1100

      $Q_3\ Q_2\ Q_1\ Q_0$=1010

   b) $P_3\ P_2\ P_1\ P_0$=1001

      $Q_3\ Q_2\ Q_1\ Q_0$=1101

3. Design a logic circuit to check the equality of two binary numbers of 4 bits, using NAND gates only.

PARITY GENERATORS / CHECKERS

Object: To study how to detect the error in the data.

Theory:

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 component malfunction or electrical noise. Many systems, however, employ a parity bit as a means of detecting a bit error. Binary information is normally handled by a digital system in groups of bits called words. A word always contains either an even or an odd number of 1's. An even parity bit makes the total even.

As an illustration of how parity bits are attached to a code word, table (6-1) lists the parity bits for each BCD code number for both even and odd parity. The parity bit for each BCD number is in the p column.

Even Parity		Odd Parity	
P_e	8 4 2 1	P_o	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

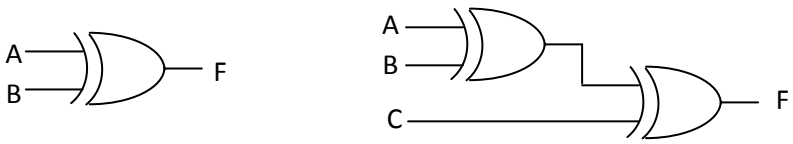
Table (6-1)

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

Notice that the total number of 1's, including the parity bit, is always even for even parity and always odd for odd parity.

Parity Logic:

In order to check for or generate the proper parity in a given code word, a very basic principle can be used. The sum of an even number of 1's is always zero, and the sum 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 EX-OR gates, as shown in Fig.(6-1-a); the sum of three bits can be formed by two EX-OR gates connected as shown in Fig.(6-1-b); and so on.



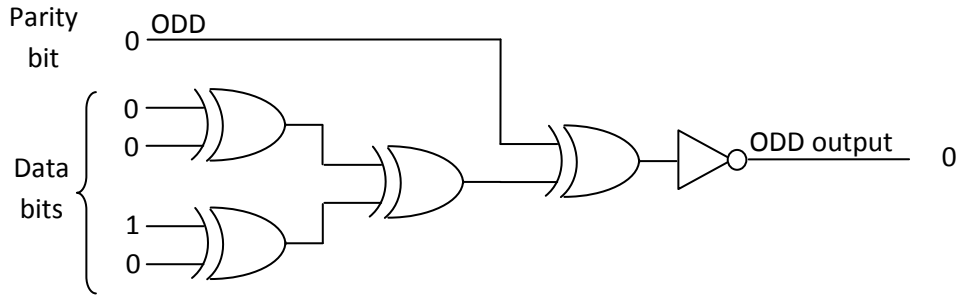
(a) Summing of two bits (b) Summing of three bits

Fig.(6-1)

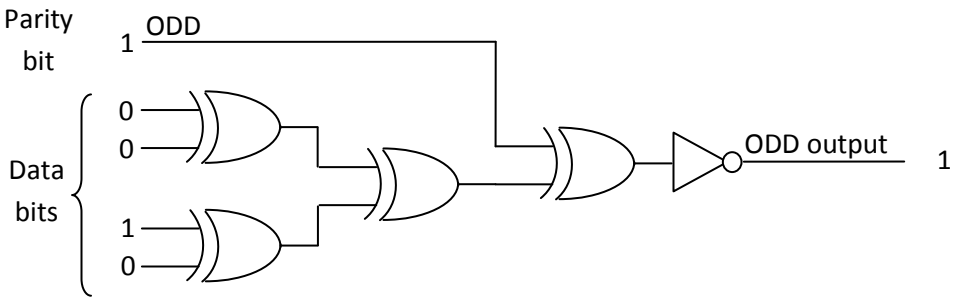
A typical 5-bit generator / checker circuit is shown in Fig. (6-2).

It can be used for either odd or even parity. When used as an odd parity checkers as shown, the operation as follows:

A 5-bit code (four data bits and one parity bit) is applied to the inputs. The four data bits are on the EX-OR inputs, and the parity bit is applied to the ODD input line. When the number of 1's in the 5-bit code is odd, the ODD output is LOW, indicating proper parity. When there is an even number of 1's, the ODD output is HIGH, indicating incorrect parity which is illustrated in Fig.(6-2).

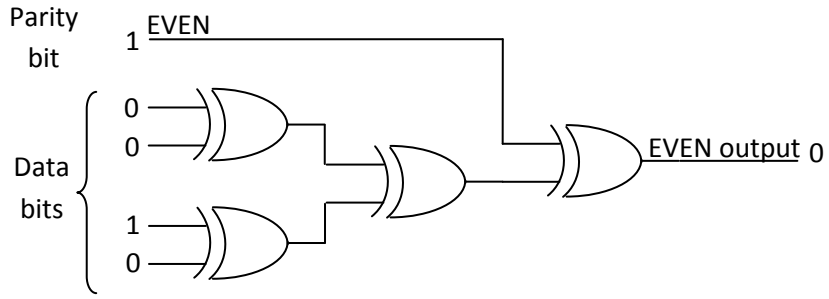


(a) Code Correct

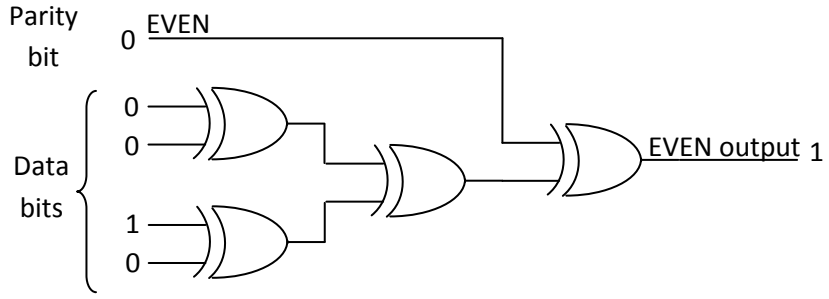


(b) Code Error

Similarly, even parity checks are illustrated for both non error and error conditions in Fig.(6-3).



(a) Code Correct



(b) Code Error

Fig.(6-3)

Procedure:

1. Design an even/odd parity generator for 4-bit data.
2. Design a parity checker circuit for a 4-bit data.
3. Design a logic circuit for a 3-bit message to be transmitted with an even parity bit.
4. Four data bits are to be transmitted. Design a parity bit generator to give an o/p of '1' if the number of logic 1's in the message is: (i) odd; (ii) even.

Discussion:

1. Attach the proper even parity bit to the following codes:

   a) 11010 b) 1001 c) 0111101

2. Repeat problem 1 for odd parity.

3. Check each of the even parity codes for an error.

4. The waveforms shown in Fig. (6-4) are applied to 4-bit parity logic. Determine the output waveform in proper relation to the inputs. How many times does even parity occur?

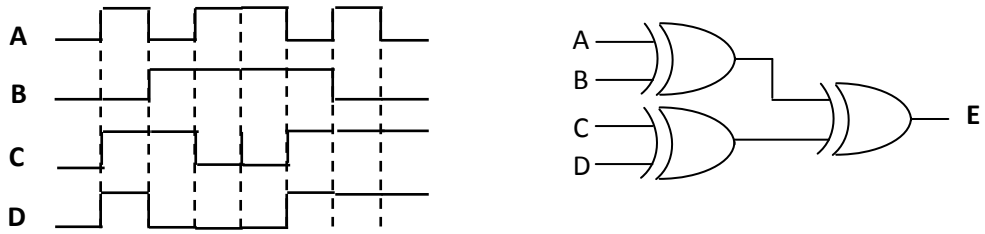


Fig.(6-4)

DECODERS & ENCODERS

Object: To study the function of decoder and encoder circuits

Theory:

(A) *Decoder:*

A decoder is a combinational circuit that converts coded information, such as binary, into a recognizable form, such as decimal. Fig. (7-1) shows a 2-to-4 line decoder circuit. The two inputs are decoded into four outputs, each output representing one of the minterms of the 2-input variables. The two inverters provide the complement of the input, each one of the minterms. However, a 2-to-4 line decoder can be used for decoding any 2-bit code to provide four outputs, one of each element at the code.

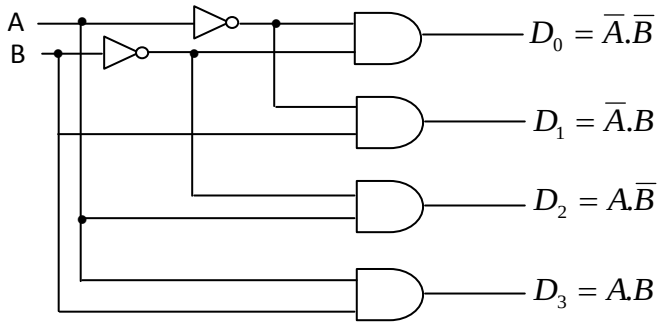


Fig.(7-1) A logic circuit of 2-to-4 line decoder

The operation of the decoder may be further classified from it's input-output relationships, listed in table(7-1) observe that one output variable are mutually exclusive because only one output can be equal to 1 at one time.

INPUTS		OUTPUTS			
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Table (7-1) Truth table of a 2-to-4 line decoder

(B) *Encoder:*

The encoder is also a combinational logic circuit; it converts information, such as a decimal number or an alphabetic character, into some coded form such as binary or BCD.

The octal-to-binary encoder consists of eight inputs, one for each of the eight digits, and three outputs that generate the corresponding binary number. It is constructed with OR gates whose inputs can be determined from the truth table given in table(7-2). The lower-order output bit Z is 1 if the input octal digit is odd. Output X is 1 for octal digits 4, 5, 6 or 7. Note that D_0 is not connected to any OR gate, the binary inputs are all 0's.

The encoder in Fig.(7-2) assumes that only one input line can be equal to 1 at any time; otherwise the circuit has no meaning.

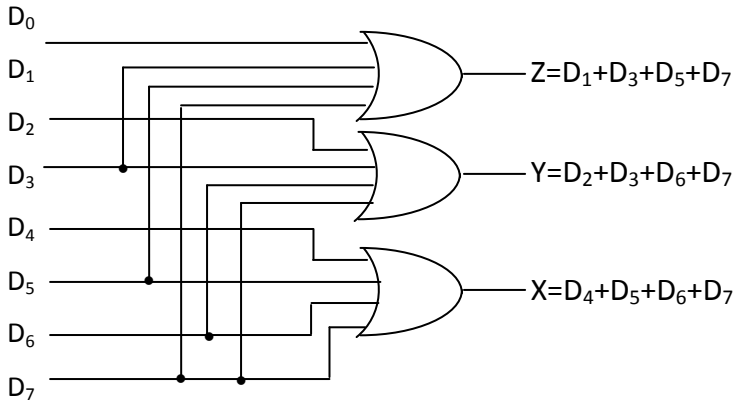


Fig. (7-2) Logic diagram of Octal-to-binary encoder

Note that the circuit has eight inputs and could have 2^8=256 possible input combinations. Only eight of these combinations have any meaning. The other inputs combinations are don't care conditions. The operation of the encoder listed in table (7-2).

INPUTS								OUTPUTS		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	X	Y	Z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Table (7-2) Truth table of an Octal-to-binary encoder

Procedure:

(A) *Decoder:*

1. Connect the circuit as shown in Fig.(7-1) using NAND gates only. Check it's truth table.
2. Design a BCD-to-Decimal decoder using NAND gates only.

(B) *Encoder:*

1. Connect the circuit as shown in Fig.(7-2) using NAND gates only. Check its truth table.

Discussion:

1. Design a 3-bit binary decoder (3-to-8 decoder), then construct this circuit using NOR gates only.
2. Design a BCD-to-seven segment decoder (7447 IC).

MULTIPLEXERS & DEMULTIPLEXERS

Object: To study the function of multiplexer and demultiplexer circuits.

Theory:

(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 a 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 input to be switched to the output line.

A simple multiplexer can be represented by a switch operation that sequentially connects each of the input lines with the output, as illustrated in Fig.(8-1).

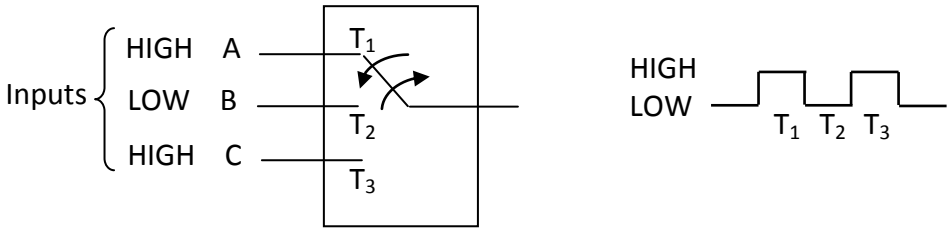


Fig. (8-1) Simple Multiplexer operation

Assume that we have logic levels an indicated on the three inputs.
During time interval T_1, input A is connected to the output; during interval T_2, input B is connected to the output; and during interval T_3, input C is connected to the output.

The logic symbol for a 4-input multiplexer is shown in Fig.(8-2). Notice that there are two selection lines because with two selection bits, each of the four data-input lines can be selected.

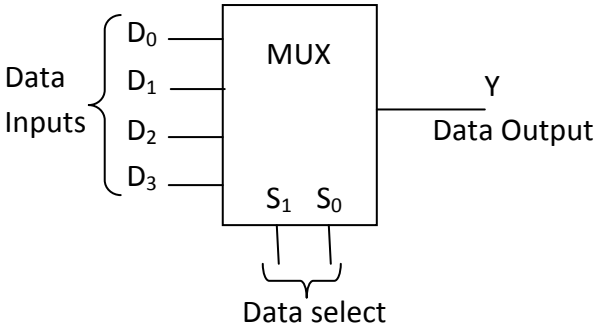


Fig. (8-2) Logic symbol for 4-to-1 data selector

If a binary 0 (S_1=0 and S_0=0) is applied to the data-select lines, the data on input D_0 appear on the data-output line. If a binary 1 (S_1=0 and S_0=1) is applied to the data-select lines, the data on the input D_1 is appear on the data output. If a binary 2 (S_1=1 and S_0=0) is applied, the data on D_2 appear on the output. If a binary 3 (S_1=1 and S_0=1) is applied, the data on D_3 are switched to the output line. A summary of this operation is given in table (8-1).

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 (8-1) 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 and S_0=0;

$$Y = D_0 \overline{S_1}.\overline{S_0}$$

The data output Y is equal to D_1 if and only if S_1=0 and S_0=1;

$$Y = D_1 \overline{S_1}.S_0$$

The data output is equal to D_2 if and only if S_1=1 and S_0=0;

$$Y = D_2 S_1.\overline{S_0}$$

The data output is equal to D_3 if and only if S_1=1 and S_0=1;

$$Y = D_3 S_1.S_0$$

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

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

The implementation of this equation is shown in Fig. (8-2).

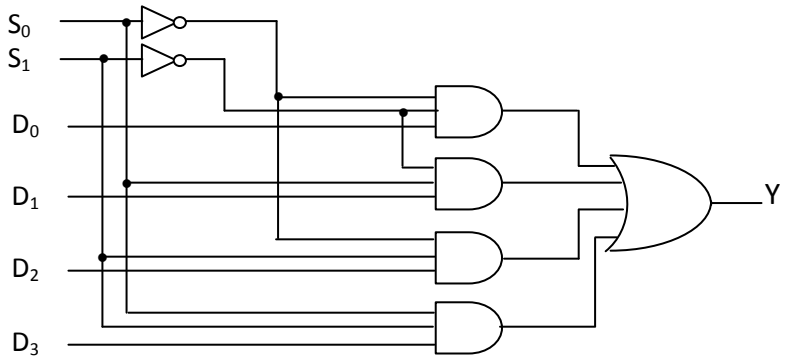


Fig. (8-2) Logic diagram for a 4-input Multiplexer

(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. Fig. (8-3) shows a one-line to four-line demultiplexer block diagram.

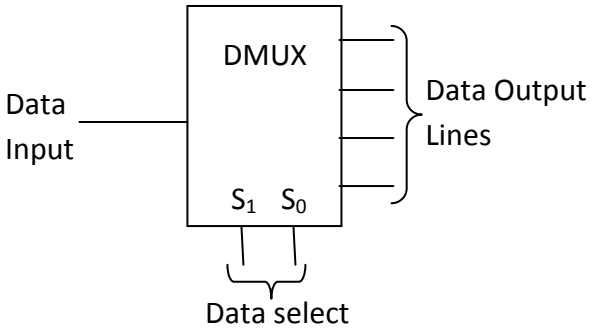


Fig. (8-3) Logic symbol 1-line to 4-line demultiplexer

Fig. (8-4) shows a 1-line-to-4-line demultiplexer circuit. The two select lines enable only one gate at a time, and the data appearing on the input line will pass through the selected gate to the associated output line.

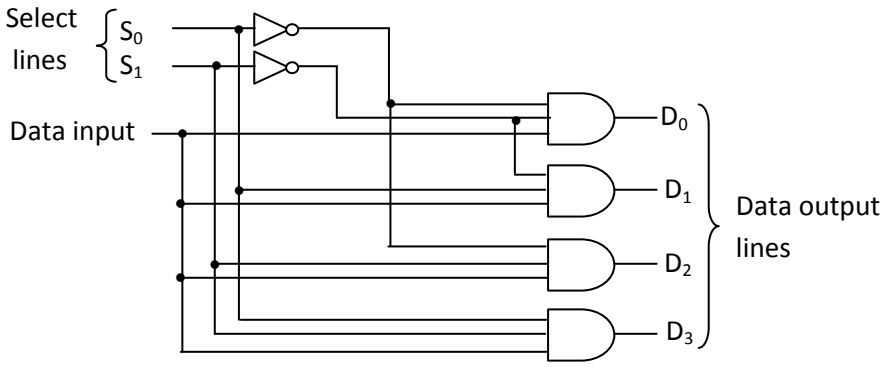


Fig. (8-4) A 1-to-4-line demultiplexer

Procedure:

(A) *Multiplexer:*

1. Connect a circuit of 2-to-1 multiplexer and observe its table.
2. Connect the circuit of Fig. (8-2) and observe its table.

(B) *Demultiplexer:*

1. Connect the circuit of Fig. (8-4) and observe its table.

Discussion:

1. The data input and data select waveforms in Fig. (8-5) are applied to the multiplexer in Fig. (8-2). Determine the output waveform in relation to the input.

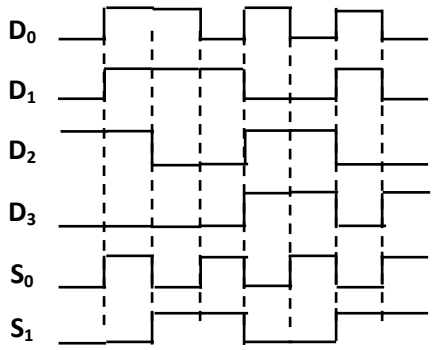


Fig. (8-5)

2. The serial data input waveform and data selectors are shown in Fig. (8-6). Determine the data-output waveform for the demultiplexer shown in Fig. (8-4).

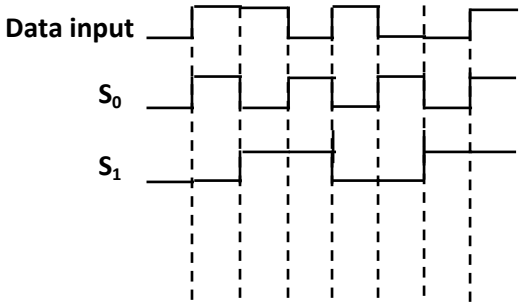


Fig. (8-6)

3. Design 8-to-1 MUX and verify its truth table.

4. Design 1-to-8 DEMUX and verify its truth table.

SEQUENTIOL LOGIC CIRCUITS

Object: To investigate the properties of logic circuit configurations possess memory.

Theory:

The logic elements studied in last experiments all required a continuous input level to operate. Once the input level was removed, the gate or network does not retain its output condition. Logic gates in other works do not have the property of memory. A machine often needs additional devices, which have a memory to retain their output states after the inputs are removed. These devices called, flip-flops, may be combined with non memory logic gates to form networks capable of controlling industrial machinery, solving mathematical problems, and storing information. The combination of many such memory devices with logic gates forms what is known as a sequential logic circuit.

Fig.(9-1a) shows the basic construction of as S-R (set-reset) Flip-flop. $\mathbf{Q_n}$ indicates the **Q** output before the clock pulse is applied and $\mathbf{Q_{n+1}}$ indicate the **Q** output after the clock pulse is applied.

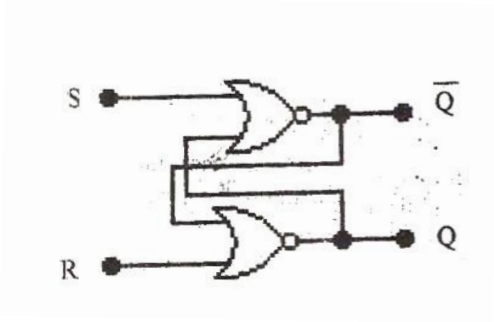


Fig.(9-1a) S-R Flip-flop construction

S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

Fig.(9-1b) S-R Flip-flop from two NOR gates truth table

In S-R Flip-flop truth table Fig.(9-1b), $\mathbf{Q_n}$ refers to the state of **Q** before the application of the inputs, while $\mathbf{Q_{n+1}}$ refer to the state of **Q** after the application of the S-R inputs (state of **Q`** us the complement of that **Q**)

Hence, the output depends not only on the input (S and R) but also on the previous state of output ($\mathbf{Q_n}$), the don't care entries in the last two rows reflect the fact that in normal operation both inputs should not be permitted to be "1" at the same time. These are two reasons for this restriction, first if both inputs are "1". Both outputs will be driven to "0" which violates the basic definition of flip-flop operation which requires that the output should always be the complement of each other.

Second S and R are "1" at the same time in the input both NOR gate and both outputs will try, to go to "0", because of the feedback it is impossible to be "1" at the same time with the result that the flip-flop will switch unpredictably and may even go into oscillation. The truth table can be written as follows:

S	R	Q_{n+1}
0	0	No change from previous state
0	1	The Flip-flop is in the reset state
1	0	The Flip-flop is in the set state
1	1	Illegal input condition

The symbol shown in fig.(9-2) is frequently used to represent the S-R Flip-flop.

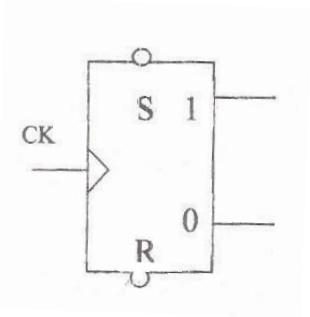


Fig.(9-2) S-R Flip-flop Symbol

Another type of flip-flop which combines both, static and dynamic data transfer is the D-type flip-flop which may be realized from the S-R as shown in Fig.(8-3)

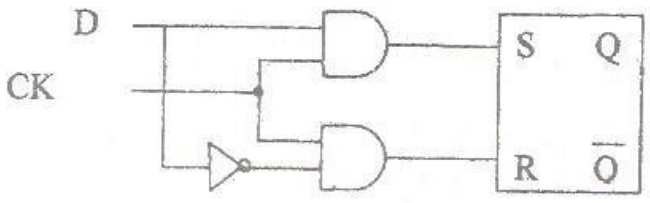


Fig.(9-3) D-Type Flip-flop

In the D-type flip-flop if clock is "0" both AND gates will be disabled and the flip-flop outputs remain unchanged whatever the state of **D** is. However if clock is "1", the **Q** output will take the value of **D**. If **D** is "0" **S** will be "0" and **R** will be 1, hence **Q** will be "0" and **Q`** "1". If **D** is "1" **S** will be "1" and **R** will be "0", hence **Q** will be "1" and **Q`** "0".

Normally the clock input is connected to a clock signal which is in the "0" state but can be changed to the "1" for short period of time when required, such short period is sufficient to cause the change. The state of **D** input could be considered as a data, which can be stored at the **Q** output after the application of a clock pulse. After the clock signal reset to "0" the state of the **D** line can be changed without affecting the previously stored state at **Q.** The truth table of the D-flip flop can be written as follows:

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

Where $\mathbf{Q_n}$ is the value of **Q** before the application of a clock pulse, and $\mathbf{Q_{n+1}}$ is the value of $\mathbf{Q_n}$ afrer the application of a clock pulse.

Another type of Flip-flop is the T-type Flip-flop shown in Fig.(8-4). Application of a clock pulse on this type of flip-flop will make it toggle (if the flip-flop is reset, it will be set after application of clock pulse and vice versa) this type of flip-flop is used in counter circuits.

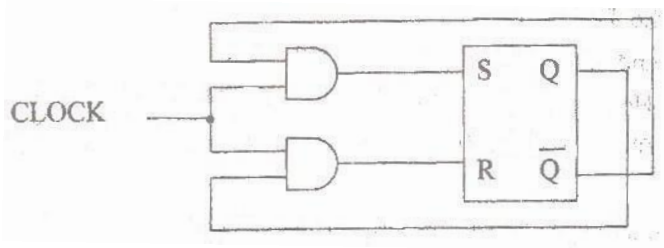


Fig.(9-4) T-Type Flip-flop

Another type of flip-flop is the J-K Flip-flop shown in Fig.(9-5).

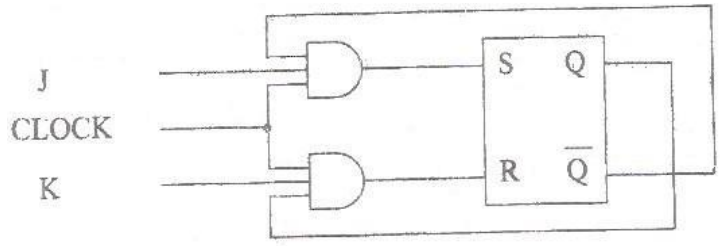


Fig.(9-5) J-K Flip-flop

In this type of flip-flop, the effect of clock pulse depends on logic states of J-K inputs (also known as steering inputs) as follows:

The advantage of the J-K flip-flop is that it avoids the undetermined output condition which occurs with the S-R Flip-flop, when both of its inputs are "1" at the same time.

J	K	Q_{n+1}
0	0	No change
0	1	The Flip-flop is in the Reset
1	0	The Flip-flop is in the Set
1	1	The Flip-flop Toggles

We can write the truth table for the type of J-K Flip-flop:

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

J-K Flip-flop can be used in register circuits to store binary numbers. The symbol of this type is shown in Fig.(9-6).

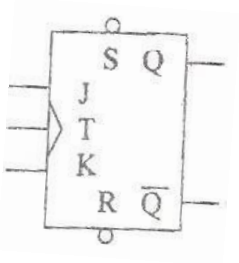


Fig.(9-6)

Procedure:

1. Using the circuit shown Fig.(9-1a), verify the truth table of the S-R flip-flop.
2. Using the unit shown in Fig.(9-2), then compare S-R Flip-flop using NOR gates only with S-R Flip-flop in Fig(9-1a).
3. Connect D-type from S-R Flip-flop.
4. Connect T-type from S-R Flip-flop.
5. Connect J-K Flip-flop from S-R Flip-flop.

Discussion:

1. Realize the S-R Flip-flop shown in Fig.(9-6) using NAND gates only, and verify its operation.
2. Draw timing waveforms for all types of Flip-flops.
3. Realize all D-type, T-type using J-K Flip-flops.

COUNTERS

Object: To study the operation and design of counters.

Theory:

A counter is a sequential circuit that counts the number of input pulses it receives. Basically, a counter is a memory device that stores the number of input pulses. Counters are used in timing circuits, signal generators, and many other digital circuits. Binary counters can be placed into two categories relating to the method by which they ate clocked. Counting circuits may be clocked ***synchronously*** or ***asynchronously***. An asynchronous counter is a one in which the Flip-flops are not simultaneously triggered. Each Flip-flop, after the least significant stage, is clocked by the output of the preceding one; they also called "ripple counters". Synchronous counters are those which all Flip-flops are simultaneously triggered from the same clock input. Asynchronous counters "up" and "down" and synchronous counters usually recycle on a number of clock pulses equal to some power of "2"; e.g. with three stages reset occurs on 8 counts, and with four stages reset occur on 16, etc.That means

$$2^N = \text{Number of pulses (counts);}$$

Where **N= Number of stages (Flip-flops).**

(I) Asynchronous Counters:

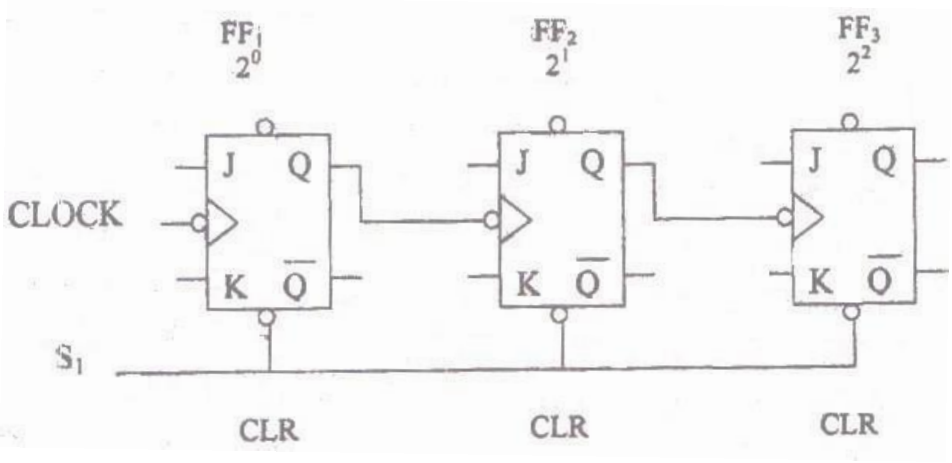


Fig.(10-1) Asynchronous 3-bit "UP" counter

Procedure:

1- Connect the circuit shown in Fig.(11-1) to observe counter to count from (0-7). *Hint: Reset the counter by flipping S1 line momentarily up, then down.*

2- With clock selector in "manual presses the manual clock button once. *Observe that the number 001 appears as "100" since FF1 contains the least significant bit.*

3- To continue press the manual clock button and observe that the count progress until "111" and resets on the 8^{th} pulse, then connect the clock pulse to continuous slow speed and notice the output.

4- Verify the truth table for this counter and draw the counter waveforms.

5- Change the input of FF2 from Q to Q` and of FF3 from Q to Q`. Then verify the truth table for this type of counter.

Discussion:

1- How many Flip-flops are needed in an up-asynchronous counter, which can count up to 64?

2- If two binary up and down counter start counting at the same time and from the same initial numbers, what is the relation between the two numbers in each counter?

(II) Asynchronous Decade Counters:

The decade counter is designed so that it will count from "0" (0000) to "9" (1001) and then reset on the next count. One type of such a counter, called a BCD counter, is shown in Fig.(10-2).

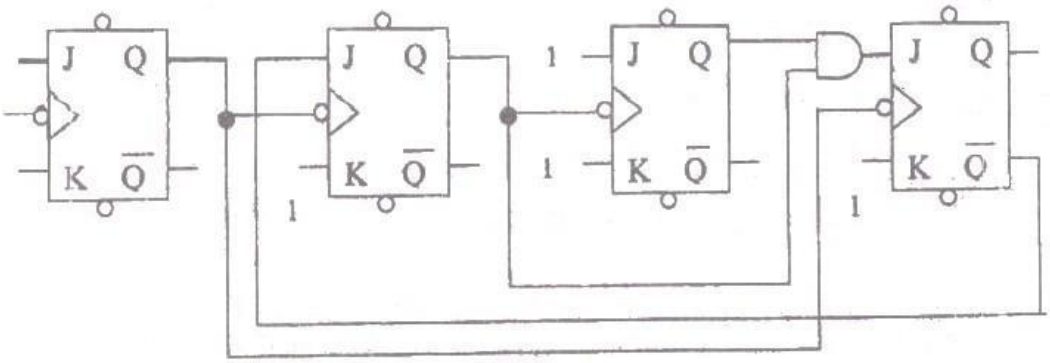


Fig.(10-2) Logic diagram of a BCD asynchronous counter

Procedure:

1- Connect the circuit shown in Fig.(10-2).
2- Connect the clock pulse to continuous slow speed and observe the O/P.

Discussion:

1- Design an asynchronous counter to count from 0 to 11 (MOD 12).
2- Design a MOD-6 asynchronous counter.

(III) Synchronous And Modulus Counters:

Procedure:

1- Reset the counter shown in Fig. (10-3).
2- Set the clock selector to slow and verify that the counter operates correctly.

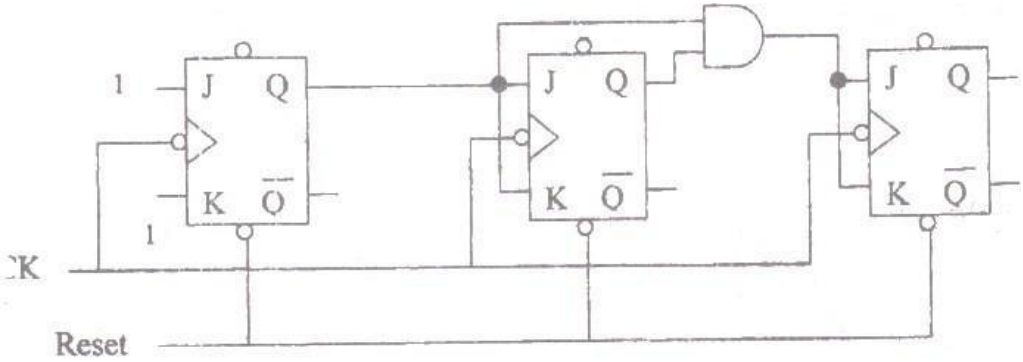


Fig.(10-3) Three Stage Synchronous "UP" counter

3- Now, you notice that the circuit you just verify Fig. (10-3) is a 3-bit up counter. To modify this circuit to be a 3-bit down counter, try to modify the connection of the circuit shown in Fig. (10-4):

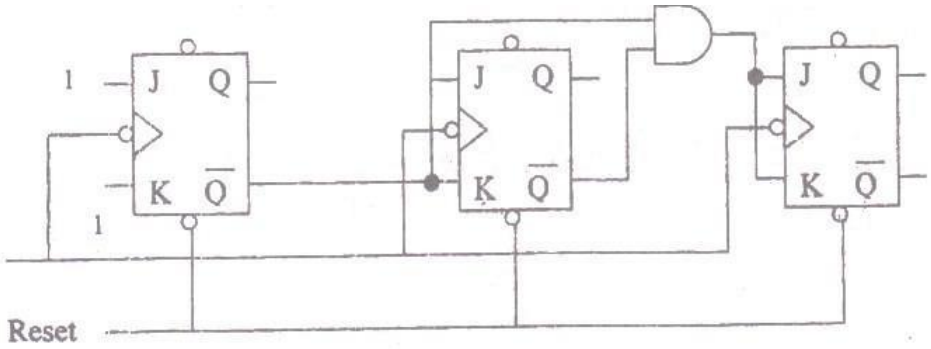


Fig.(10-4) Three Stage Synchronous "DOWN" counter

4- Connect the circuit in Fig. (10-5) and verify its operation as a synchronous counter.

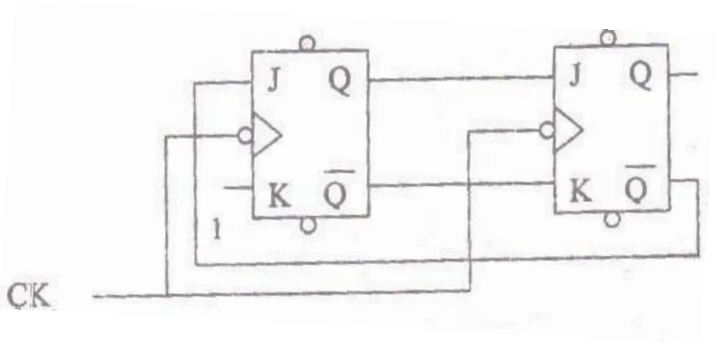


Fig.(10-5) "MOD 3" Synchronous counter

Discussion:

1- Design a 3-bit UP-DOWN synchronous counter such that the UP or DOWN counter is selected by a switch. Connect and verify operation.
2- Design a divide-by-6 counter and illustrate its operation.
3- Design a synchronous counter to count this following sequence:

{ 2 – 3 – 5 – 7 }

SHIFT REGISTERS

Object: To verify the types and functions of shift registers.

Theory:

Registers are memory devices used for storing and manipulating data, and they are essential components of most digital systems. Registers may be classified according to how their stored information is entered or removed one bit at a time, and a parallel register accepts or transfers all bits of data simultaneously. We may also have serial-parallel or parallel-serial registers, in which the data is entered one way and removed the other.

(I) Memory Registers:

A memory register, or storage register, is a device capable of accepting information in the form of a binary number, holding that information after the input that provided it has been removed, and making the information available as an output. Memory registers may be constructed with Flip-flops. Fig.(11-1) shows how to represent a memory register for 8-bit storing the number 10010100.

1	0	0	1	0	1	0	0

Fig.(11-1) Symbol for an 8-bit memory register

(II) Shift Registers:

Another important type of register is the shift registers, which can store data as a memory register does but is more often used to process, or move data. Usually the data movement is made by shifting data serially from one stage of the register to adjacent stage. The shift may be from left to right (a right-shift register), from right to left (a left-shift register), or in both directions (a bi-directional shift register).

Also, the data can be rotate left or right. Finally the data can be shifted in the serially in and parallel out, or a parallel in and serial out, or parallel in and parallel out. This is shown in Fig.(11-2).

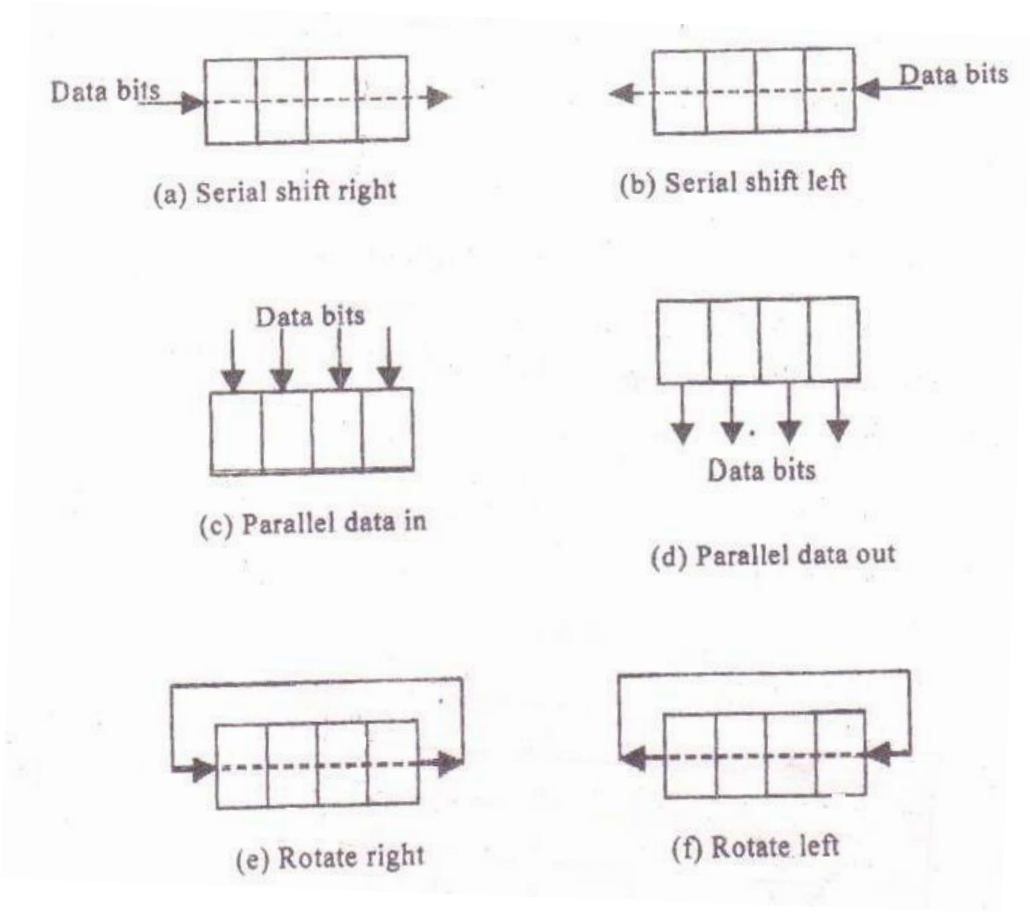


Fig.(11-2) Types of shift registers

Procedure:

1- Connect the circuit shown in Fig.(11-1) to observe counter to count from (0-7). *Hint: Reset the counter by flipping S1 line momentarily up, then down.*

2- With clock selector in "manual presses the manual clock button once. *Observe that the number 001 appears as "100" since FF1 contains the least significant bit.*

3- To continue press the manual clock button and observe that the count progress until "111" and resets on the 8^{th} pulse, then connect the clock pulse to continuous slow speed and notice the output.

4- Verify the truth table for this counter and draw the counter waveforms.

5- Change the input of FF2 from Q to Q` and of FF3 from Q to Q`. Then verify the truth table for this type of counter.

Discussion:

1. Generally, what is the difference between a counter and a shift register?
2. What two principle functions are performed by a shift register?
3. How many clock pulses are required to enter a byte of data serially into an 8-bit shift register?
4. What are the differences between SISO and PIPO shift registers.