LAB MANUAL

DIGITAL ELECTRONICS LABORATORY

Subject Code: 210246

2018 - 19

INDEX Batch : -

Sr.No	Title	Page No	Date of Conduction	Date of Submission	Signature of Staff
GROUP - A					
1	Realize Full Adder and Subtractor using a) Basic Gates and b) Universal Gates				
2	Design and implement Code converters-Binary to Gray and BCD to Excess-3				
3	Design of n-bit Carry Save Adder (CSA) and Carry Propagation Adder (CPA). Design and Realization of BCD Adder using 4-bit Binary Adder (IC 7483).				
4	Realization of Boolean Expression for suitable combination logic using MUX 74151 / DMUX 74154				
5	Verify the truth table of one bit and two bit comparators using logic gates and comparator IC				
6	Design & Implement Parity Generator using EX-OR.				
GROUP - B					
7	Flip Flop Conversion: Design and Realization				
8	Design of Ripple Counter using suitable Flip Flops				
9	a. Realization of 3 bit Up/Down Counter using MS JK Flip Flop / D-FF b. Realization of Mod -N counter using (7490 and 74193)				
10	Design and Realization of Ring Counter and Johnson Ring counter				
11	Design and implement Sequence generator using JK flip-flop				

INDEX Batch : -

12	Design and implement pseudo random sequence generator.	
13	Design and implement Sequence detector using JK flip-flop Design	
14	of ASM chart using MUX controller Method.	

GROUP - C

15	Design and Implementation of Combinational Logic using PLAs.	
16	Design and simulation of - Full adder , Flip flop, MUX using VHDL (Any 2) Use different modeling styles.	
17	Design & simulate asynchronous 3- bit counter using VHDL.	
18	Design and Implementation of Combinational Logic using PALs	

GROUP - D

19	Study of Shift Registers (SISO,SIPO, PISO,PIPO)	
20	Study of TTL Logic Family: Feature, Characteristics and Comparison with CMOS Family	
21	Study of Microcontroller 8051 : Features, Architecture and Programming Model	

`

GROUP - A

Assignment: 1

- ***Title:*** Adder and Subtractor
- ***Objective:***

    1. To study combinational circuit like full adder and full substractor.
    2. To know about basic gates.
    3. To know about universal gates.

- ***Problem Statement***:

    To realize full adder and full substractor using

    a) Basic gates
    b) Universal gates

- ***Hardware & software requirements:***

Digital Trainer Kit, IC7432, IC 7408, IC 7404, (Decade Counter IC), Patch Cord, + 5V Power Supply

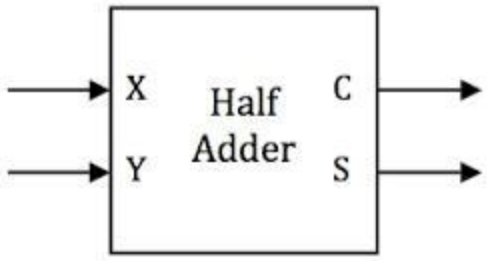
Theory:

1. Combinational circuit.
2. Half Adder.
3. Half Substractor.
4. Full Adder.
5. Full substracter.

1. **Combinational Circuit:-** Realization steps for combinational circuit

    a) Truth Table
    b) K-Map.
    c) MSI Circuits.

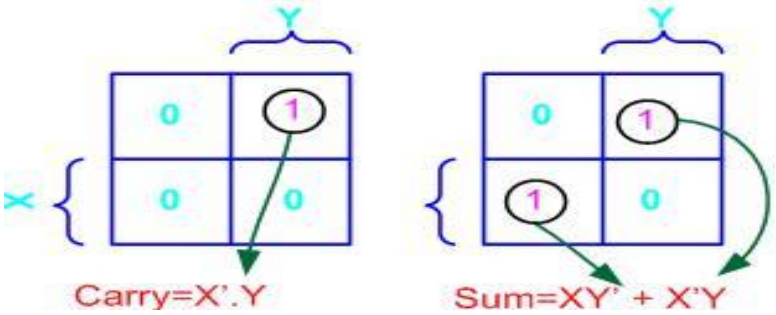
2. Half Adder:-



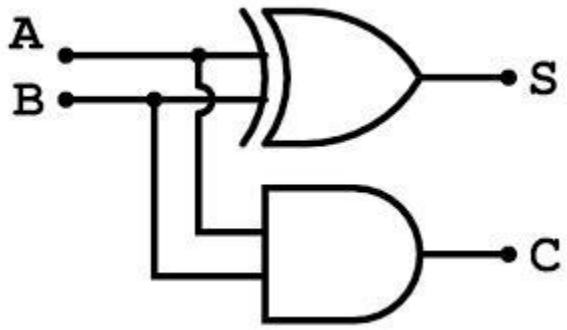
- **Truth Table:-**

Input		Output	
X	**Y**	**Sum**	**Carry**
0	**0**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**1**	**0**
1	**1**	**0**	**1**

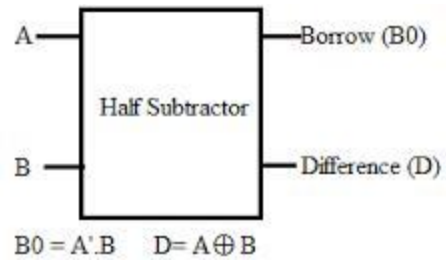
- **K-Map:-**



- **Logic Diagram:-**



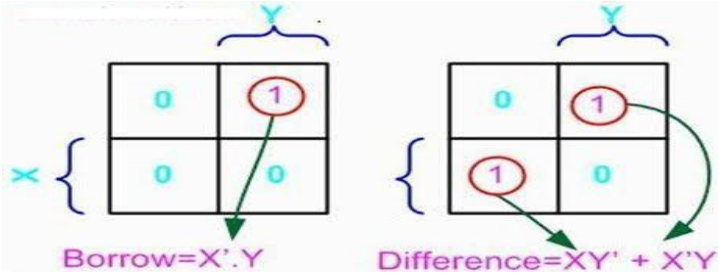
3. Half Substractor :-



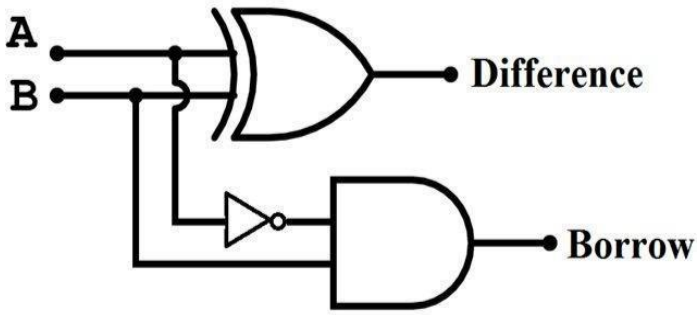
- **Truth Table:-**

Input		Output	
A	**B**	**Difference**	**Borrow**
0	**0**	**0**	**0**
0	**1**	**1**	**1**
1	**0**	**1**	**0**
1	**1**	**0**	**0**

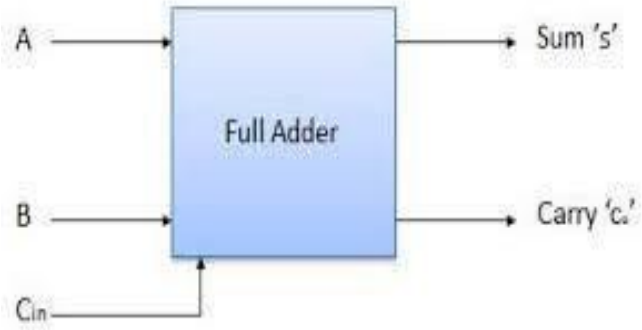
- **K-Map:-**



- **Logic Gates:-**



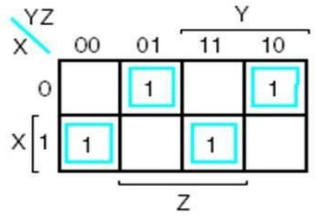
4. Full Adder:-



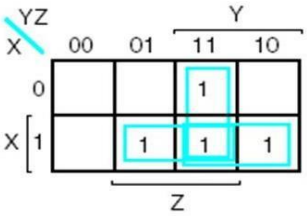
- **Truth Table:-**

X	Y	Z	Sum	Carry
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

- **K-Map:-**

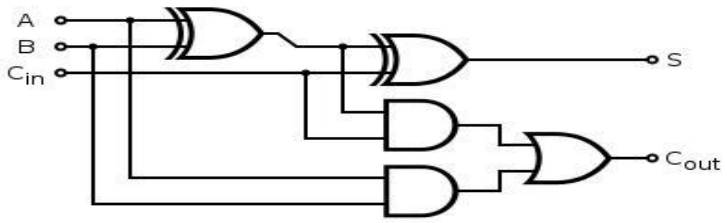


$S = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ$
$= X \oplus Y \oplus Z$



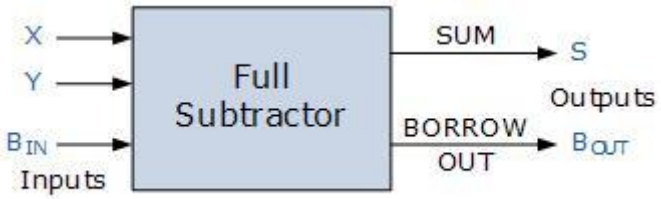
$C = XY + XZ + YZ$
$= XY + Z(X\bar{Y} + \bar{X}Y)$
$= XY + Z(X \oplus Y)$

- **Logic Gates:-**



(b) Circuit Diagram

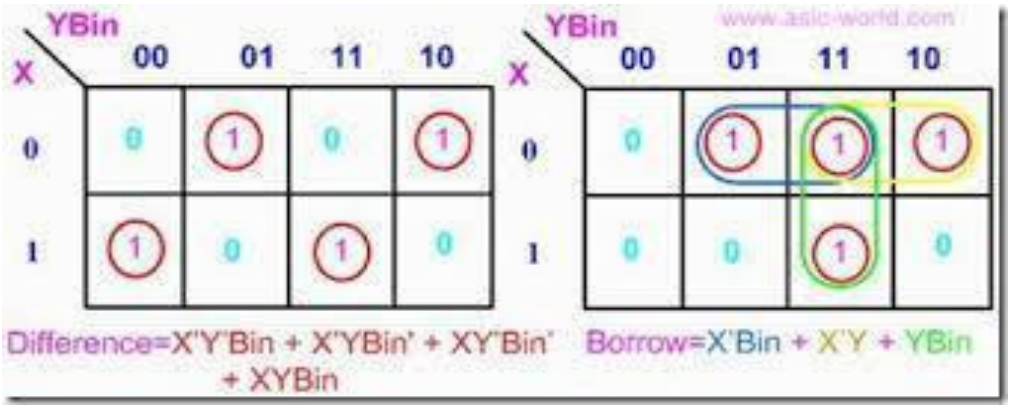
5. **Full Substractor:-**



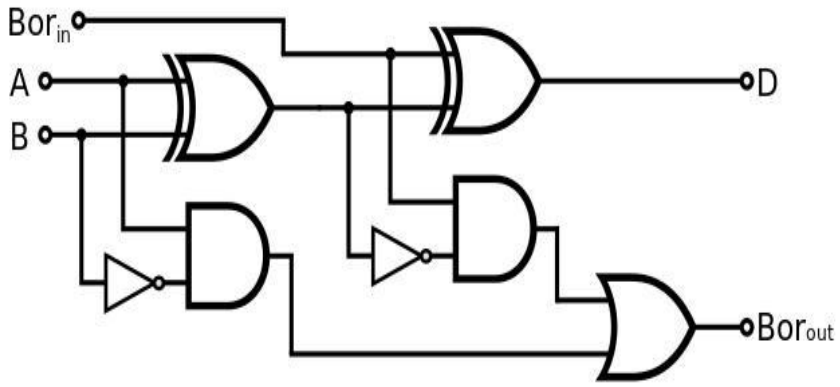
- **Truth Table:-**

X	Y	Z	Difference	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

- **K-Map:-**



- **Logic Gates:-**



Outcomes:-

Successfully designed and implemented Adder and Subtractor.

Assignments Questions:

Assignment No: 2

- **Title: Code Converter**
- **Objective:** To learn various code & its conversion
- **Problem Statement**: To Design and implement the circuit for the following 4-bit Code conversion.
  - i) Binary to Gray Code
  - ii) Gray to Binary Code
  - iii) BCD to Excess – 3 Code
  - iv) Excess-3 to BCD Code
- **Hardware & software requirements:**

  Digital Trainer Kit, IC 7404, IC 7432, IC 7408, IC 7486, Patch Cord, + 5V Power Supply

Theory:

There is a wide variety of binary codes used in digital systems. Some of these codes are binary- coded-decimal (BCD), Excess-3, Gray, octal, hexadecimal, etc. Often it is required to convert from one code to another. For example the input to a digital system may be in natural BCD and output may be 7-segment LEDs. The digital system used may be capable of processing the data in straight binary format. Therefore, the data has to be converted from one type of code to another type for different purpose. The various code converters can be designed using gates.

1) Binary Code:

It is straight binary code. The binary number system (with base 2) represents values using two symbols, typically 0 and 1.Computers call these bits as either off (0) or on (1). The binary code are made up of only zeros and ones, and used in computers to stand for letters and digits. It is used to represent numbers using natural or straight binary form.

It is a weighted code since a weight is assigned to every position. Various arithmetic operations can be performed in this form. Binary code is weighted and sequential code.

2) Gray Code:

It is a modified binary code in which a decimal number is represented in binary form in such a way that each Gray- Code number differs from the preceding and the succeeding number by a single bit. (E.g. for decimal number 5 the equivalent Gray code is 0111 and for 6 it is 0101. These two codes differ by only one bit position i. e. third from the left.) Whereas by using binary code there is a possibility of change of

all bits if we move from one number to other in sequence (e.g. binary code for 7 is 0111 and for 8 it is 1000). Therefore it is more useful to use Gray code in some applications than binary code.

The Gray code is a nonweighted code i.e. there are no specific weights assigned to the bit positions.

Like binary numbers, the Gray code can have any no. of bits. It is also known as reflected code.

Applications:

1. Important feature of Gray code is it exhibits only a single bit change from one code word to the next in sequence. This property is important in many applications such as Shaft encoders where error susceptibility increases with number of bit changes between adjacent numbers in sequence.

2. It is sometimes convenient to use the Gray code to represent the digital data converted from the analog data (Outputs of ADC).

3. Gray codes are used in angle-measuring devices in preference to straight forward binary encoding.

4. Gray codes are widely used in K-map

The disadvantage of Gray code is that it is not good for arithmetic operation

Binary to Gray Conversion

In this conversion, the input straight binary number can easily be converted to its Gray code equivalent.

1. Record the most significant bit as it is.
2. EX-OR this bit to the next position bit, record the resultant bit.
3. Record successive EX-ORed bits until completed.
4. Convert 0011 binary to Gray.

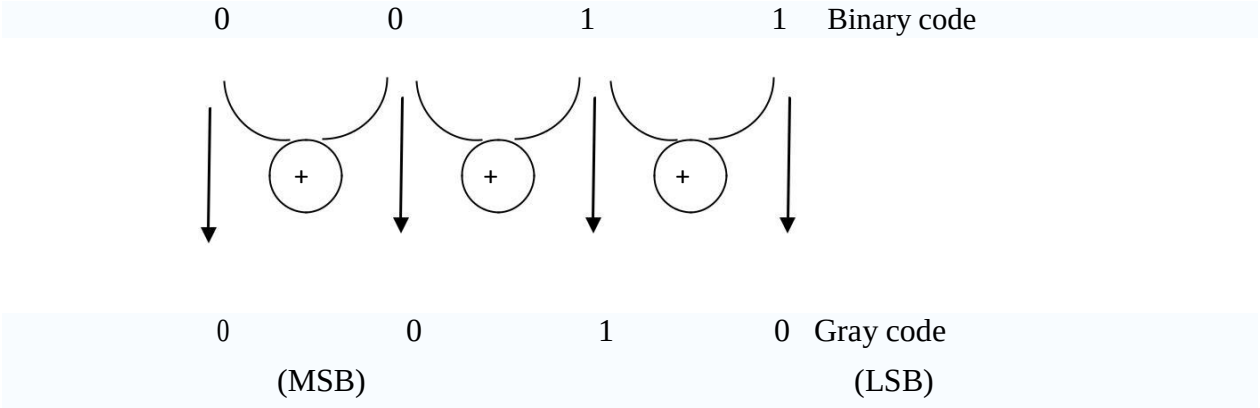


Fig. 1 Binary to Gray Conversion

Gray to Binary Conversion

1. The Gray code can be converted to binary by a reverse process.
2. Record the most significant bit as it is.
3. EX-OR binary MSB to the next bit of Gray code and record the resultant bit.
4. Continue the process until the LSB is recorded.
5. Convert 1011 Gray to Binary code.

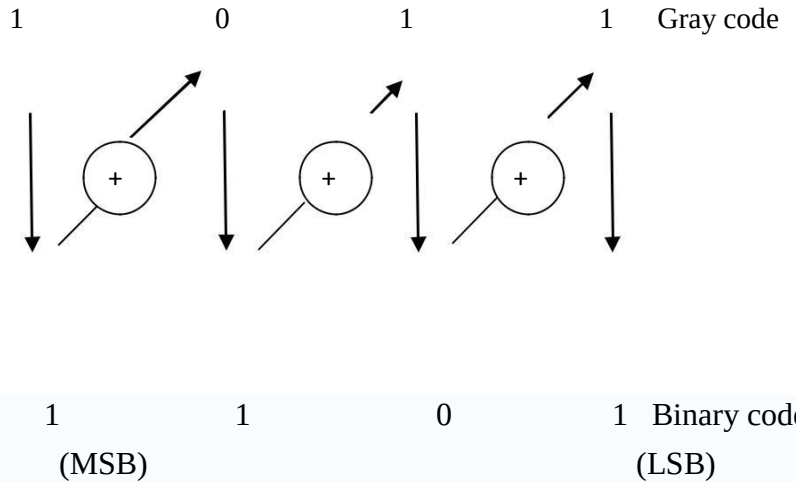


Fig. 2 Gray to Binary Conversion

3) BCD Code:

Binary Coded Decimal (BCD) is used to represent each of decimal digits (0 to 9) with a 4-bit binary code. For example $(23)_{10}$ is represented by 0010 0011 using BCD code rather than$(10111)_2$ This code is also known as 8-4-2-1 code as 8421 indicates the binary weights of four bits(2^3, 2^2, 2^1, 2^0). It is easy to convert between BCD code numbers and the familiar decimal numbers. It is the main advantage of this code. With four bits, sixteen numbers (0000 to 1111) can be represented, but in BCD code only 10 of these are used. The six code combinations (1010 to 1111) are not used and are invalid.

Applications: Some early computers processed BCD numbers. Arithmetic operations can be performed using this code. Input to a digital system may be in natural BCD and output may be 7-segment LEDs.

It is observed that more number of bits are required to code a decimal number using BCD code than using the straight binary code. However in spite of this disadvantage it is very convenient and useful code for input and output operations in digital systems.

4) EXCESS-3 Code:

Excess-3, also called XS3, is a non weighted code used to express decimal numbers. It can be used for the representation of multi-digit decimal numbers as can BCD.The code for each decimal number is obtained by adding decimal 3 and then converting it to a 4-bit binary number. For e.g. decimal 2 is coded as 0010 + 0011 = 0101 in Excess-3 code.

This is self complementing code which means 1‘s complement of the coded number yields 9‘s complement of the number itself. Self complementing property of this helps considerably in performing subtraction operation in digital systems, so this code is used for certain arithmetic operations.

BCD To Excess – 3 Code Conversions:

Convert BCD 2 i. e. 0010 to Excess – 3 codes

For converting 4 bit BCD code to Excess – 3, add 0011 i. e. decimal 3 to the respective code using rules of binary addition.

0010 + 0011 = 0101 – Excess – 3 code for BCD 2

Excess – 3 Code To BCD Conversion:

The 4 bit Excess-3 coded digit can be converted into BCD code by subtracting decimal value 3 i.e. 0011 from 4 bit Excess-3 digit.

e.g. Convert 4-bit Excess-3 value 0101 to equivalent BCD code.

0101-0011= 0010- BCD for 2

Design:

A) Binary to Gray Code Conversion:

1) Truth Table:

Table 1 Binary to Gray Code Conversion

INPUT (BINARY CODE)				**OUTPUT (GRAY CODE)**			
B_3	**B_2**	**B_1**	**B_0**	**G_3**	**G_2**	**G_1**	**G_0**
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

2) K-Map for Reduced Boolean Expressions of Each Output:

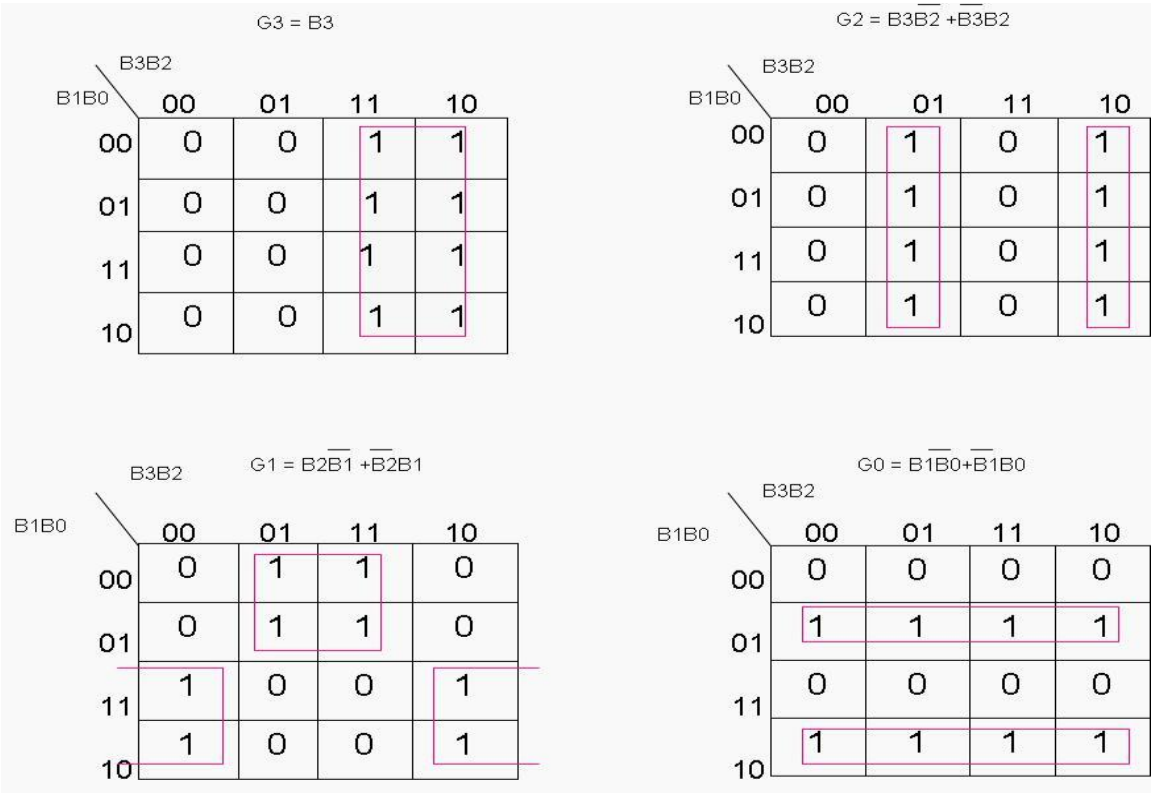


Fig. 4 K-Map for Reduced Boolean Expressions of Each Output (Gray Code)

3) Circuit Diagram:

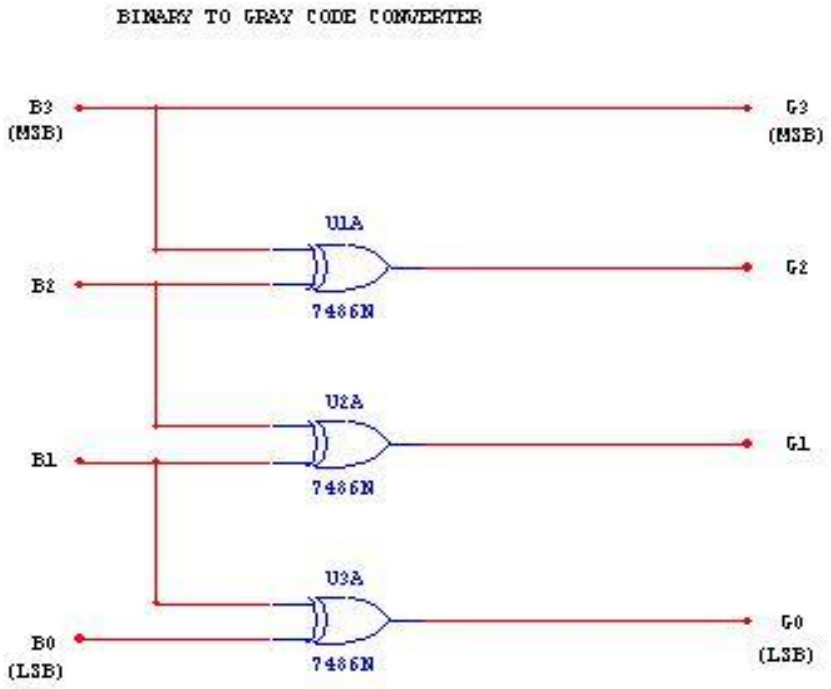


Fig. 5 Logical Circuit Diagram for Binary to Gray Code Conversion

B) Gray to Binary Code Conversion:

1) Truth Table:

Table 2 Gray to Binary Code Conversion

INPUT (GRAY CODE)				OUTPUT (BINARY CODE)			
G_3	G_2	G_1	G_0	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1

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

2) K-Map for Reduced Boolean Expressions of Each Output:

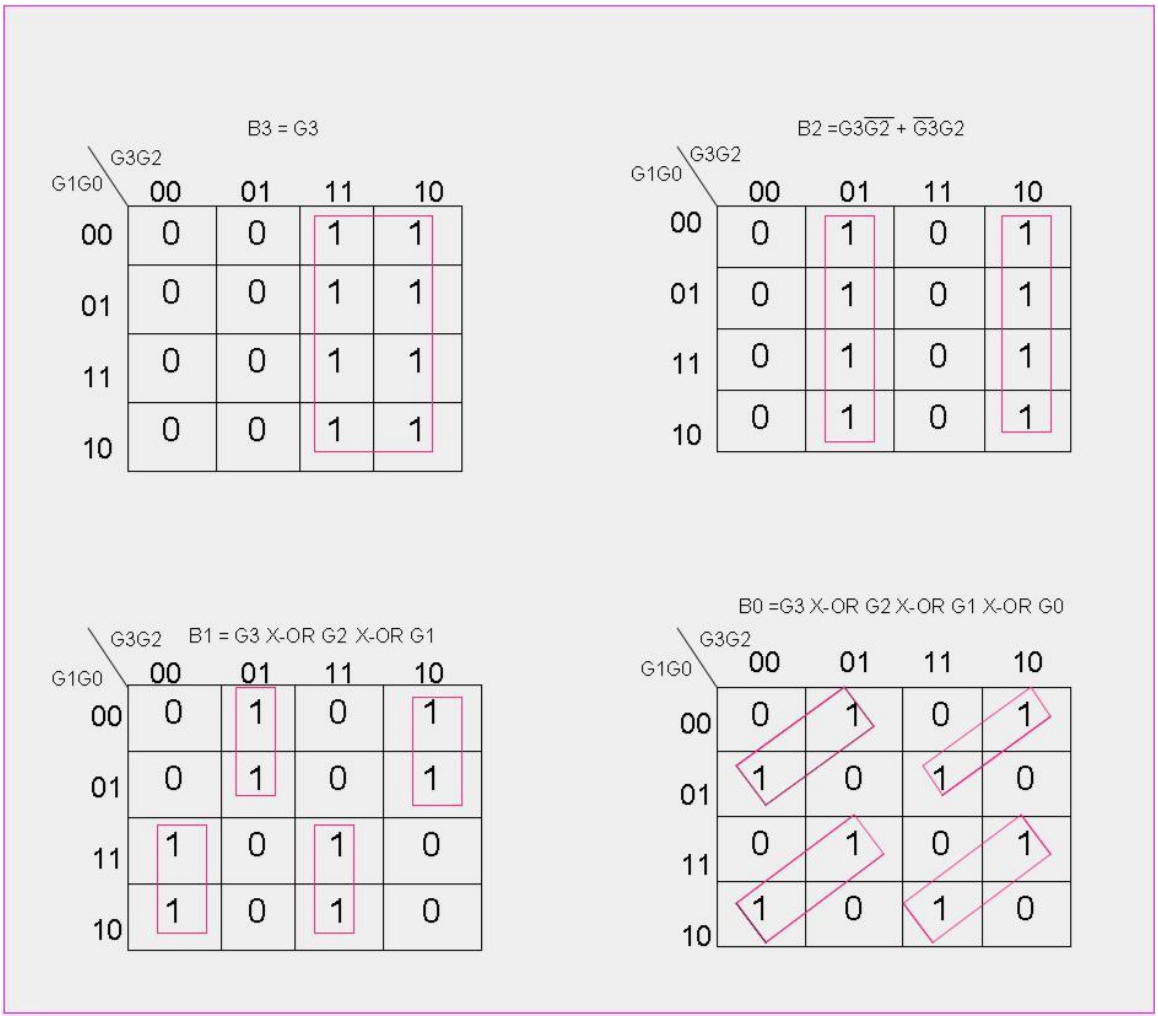
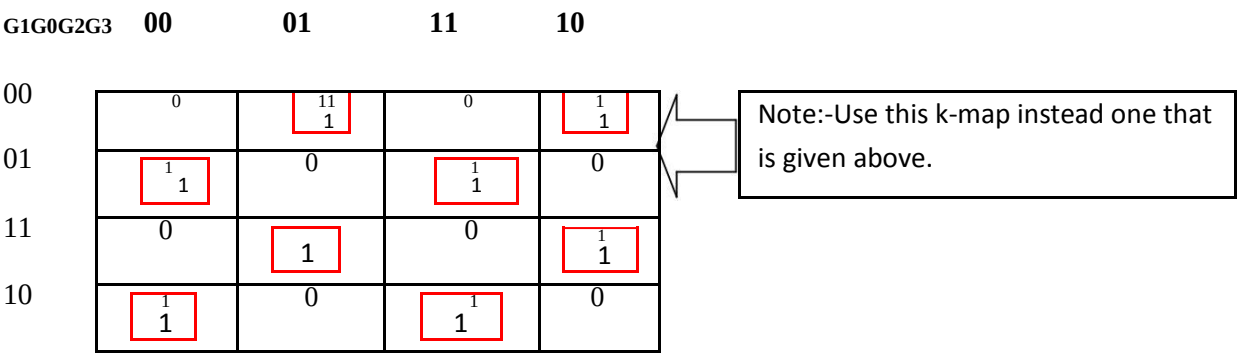


Fig. 6 K-Map for Reduced Boolean Expressions of Each Output (Binary Code)



B0 = G3 X-OR G2 X-OR G1 X-OR G0

3) Circuit Diagram:

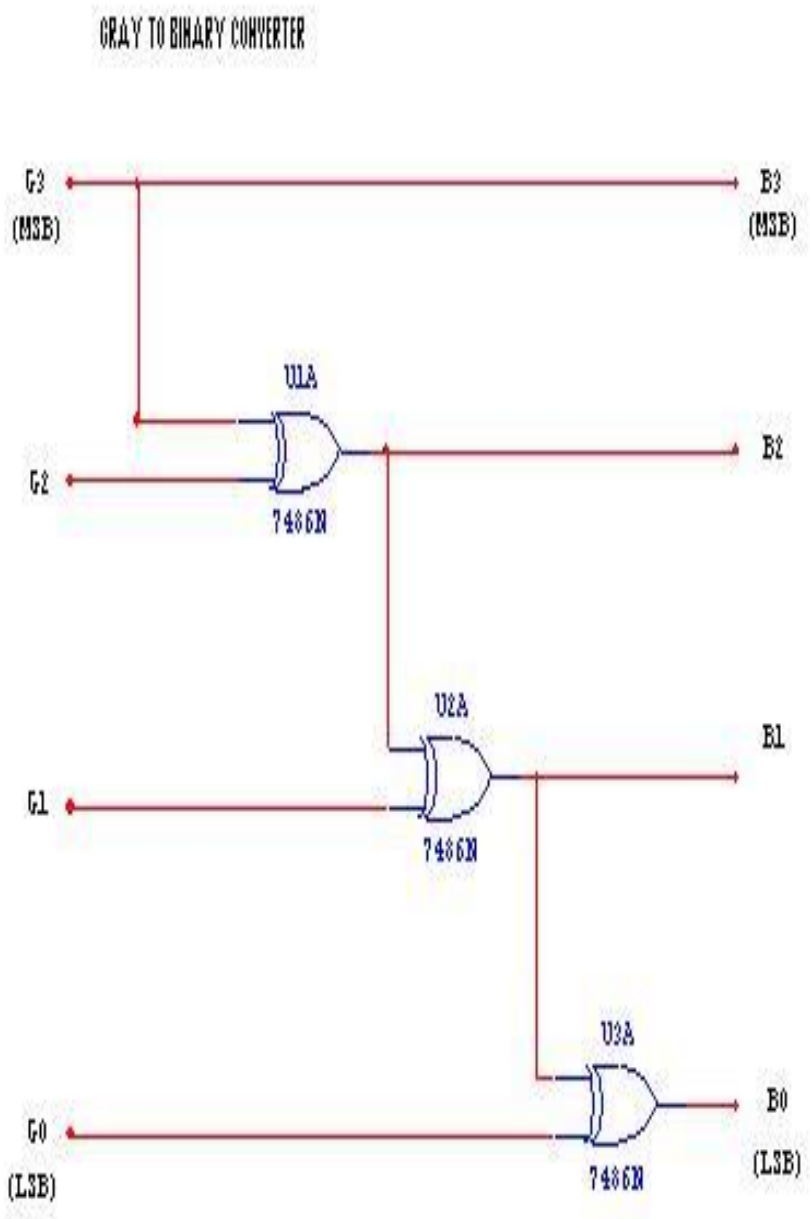


Fig. 7 Logical Circuit Diagram for Gray to Binary Code Conversion

C) BCD to Excess-3 Code Conversion:

1) Truth Table:

Table 3 BCD to Excess-3 Code Conversion

INPUT (BCD CODE)				**OUTPUT (EXCESS-3 CODE)**			
B_3	**B_2**	**B_1**	**B_0**	**E_3**	**E_2**	**E_1**	**E_0**
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

2) K-Map for Reduced Boolean Expressions of Each Output:

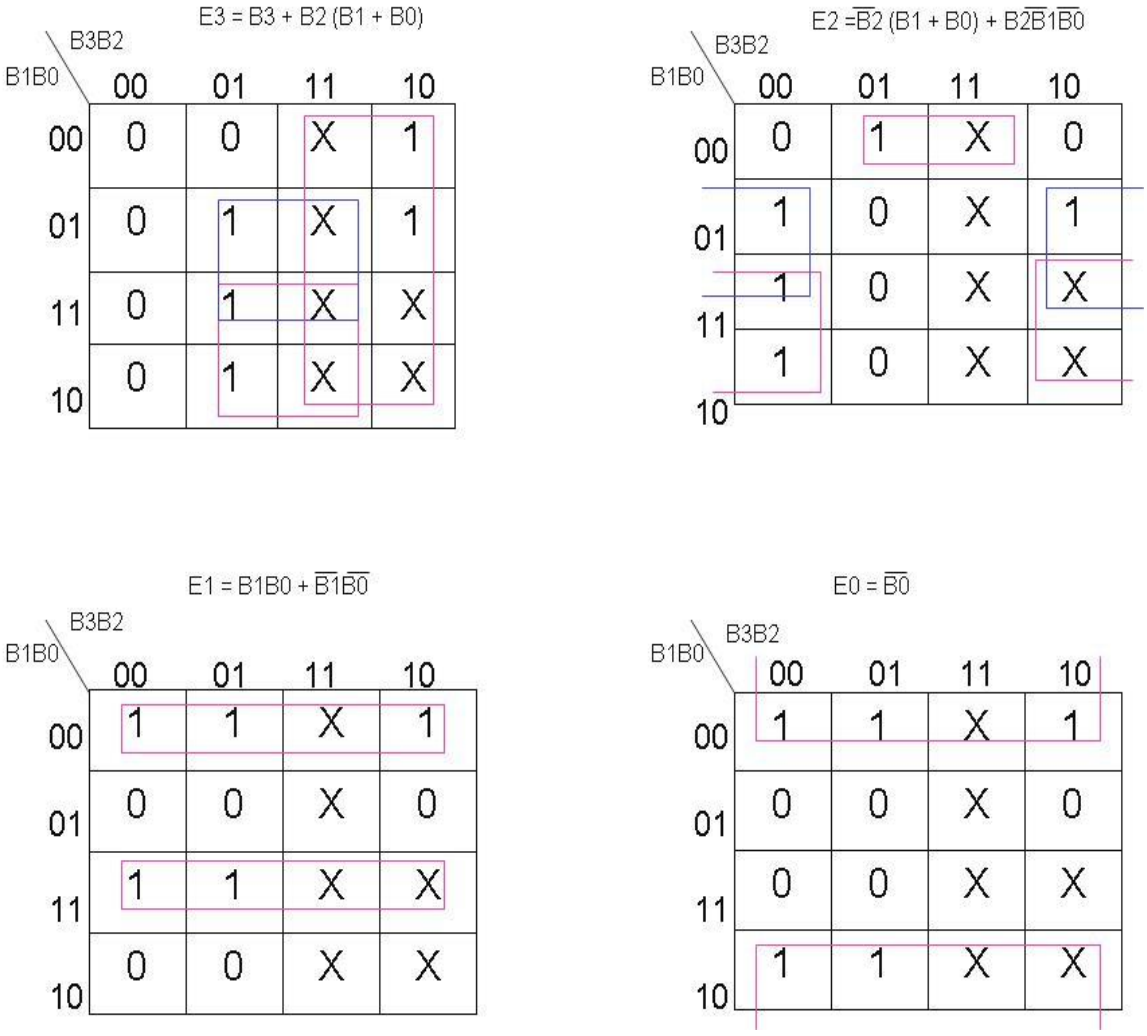


Fig. 8 K-Map for Reduced Boolean Expressions Of Each Output (Excess-3 Code)

3) Circuit Diagram:

BCD TO EXCESS-3 CONVERTER

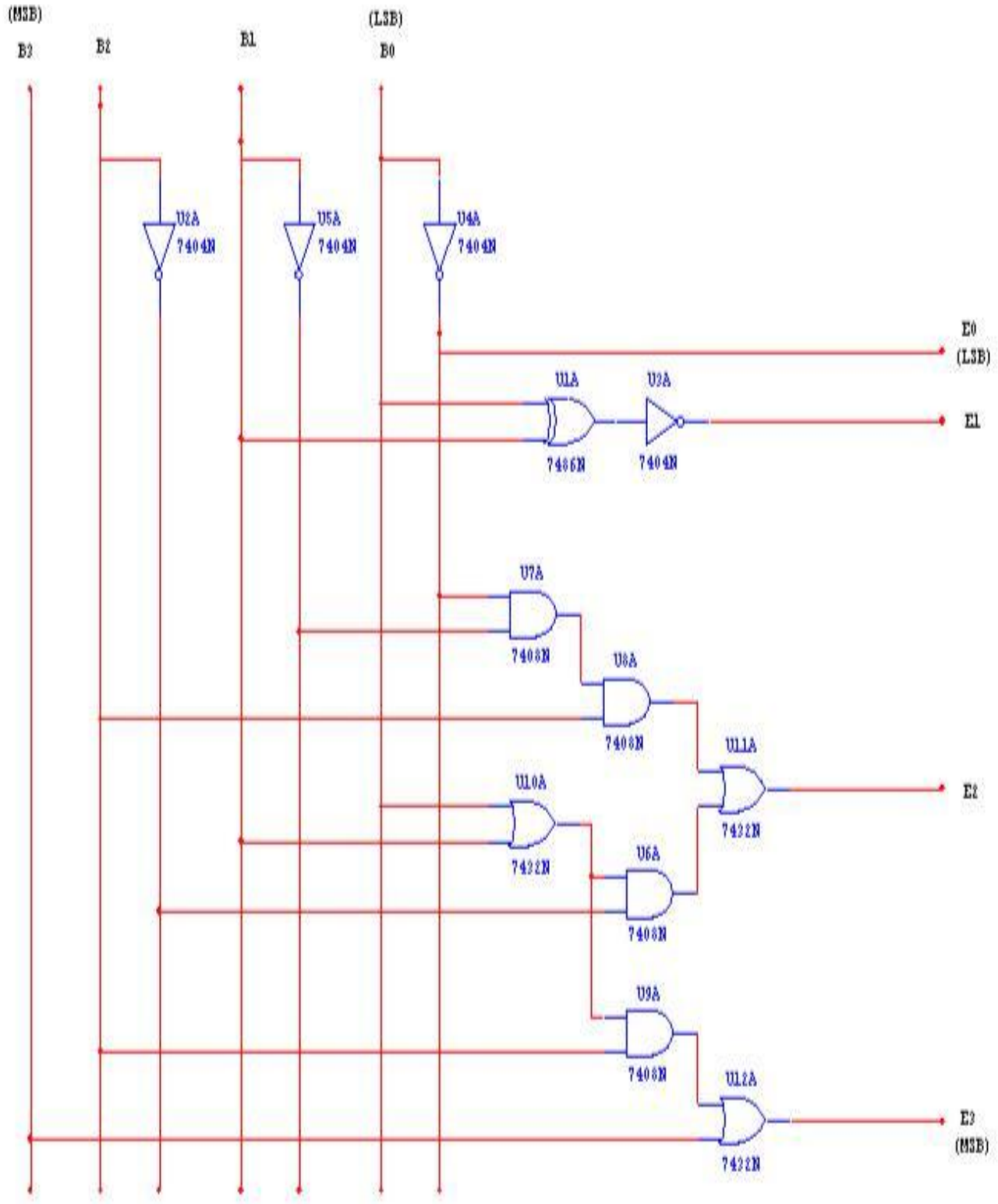


Fig.9 Logical Circuit Diagram for BCD to Excess-3 Code Conversion

D) Excess-3 to BCD Conversion:

1) Truth Table:

Table 4 Excess-3 To BCD Conversion

INPUT (EXCESS-3 CODE)				**OUTPUT (BCD CODE)**			
E_3	**E_2**	**E_1**	**E_0**	**B_3**	**B_2**	**B_1**	**B_0**
0	0	0	0	X	X	X	X
0	0	0	1	X	X	X	X
0	0	1	0	X	X	X	X
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	0
1	0	0	0	0	1	0	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	1	1
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	1
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

2) K-Map for Reduced Boolean Expressions of Each Output:

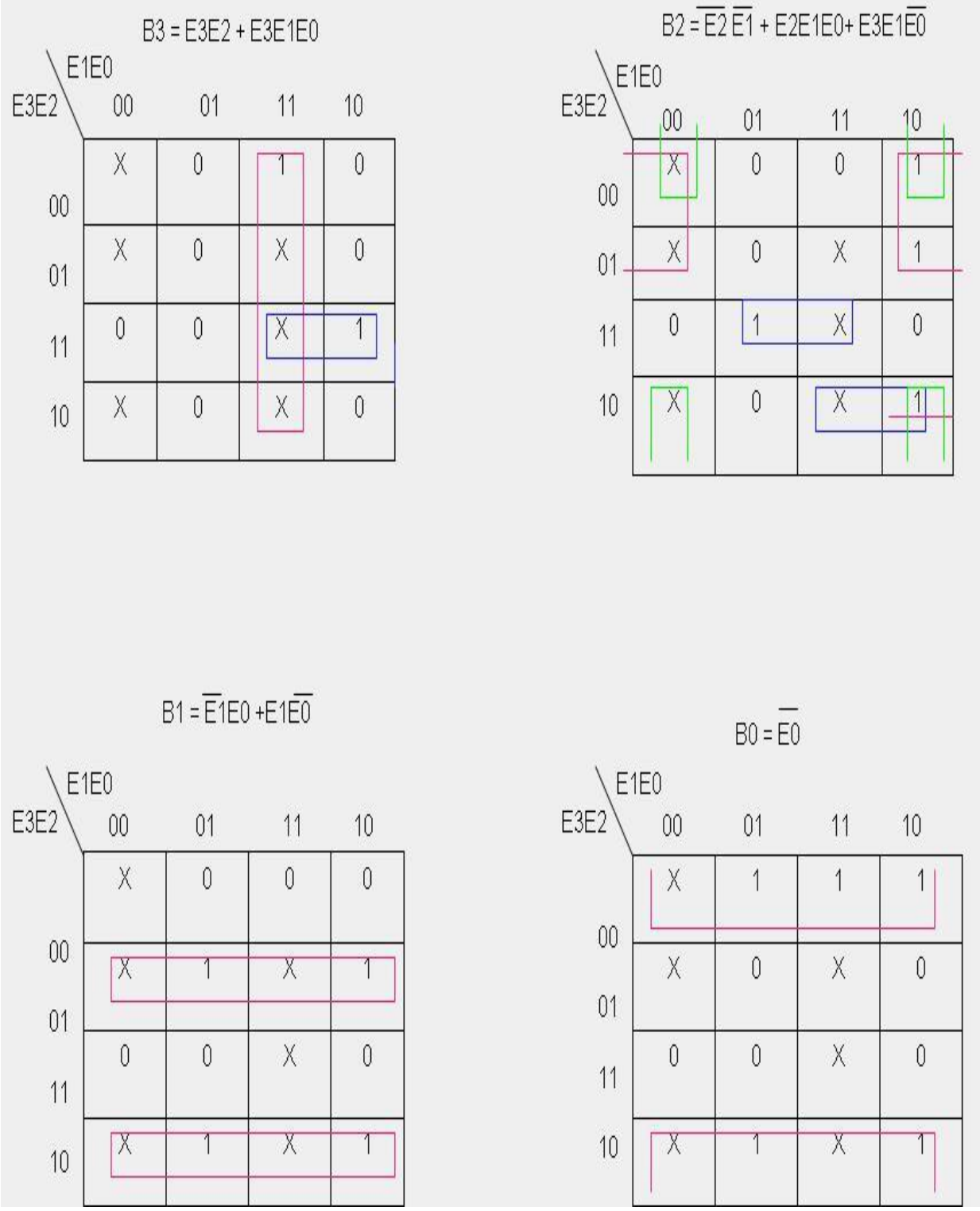


Fig 10 K-Map For Reduced Boolean Expressions of Each Output (BCD Code)

3) Circuit Diagram:

EXCESS-3 TO BCD CONVERTER

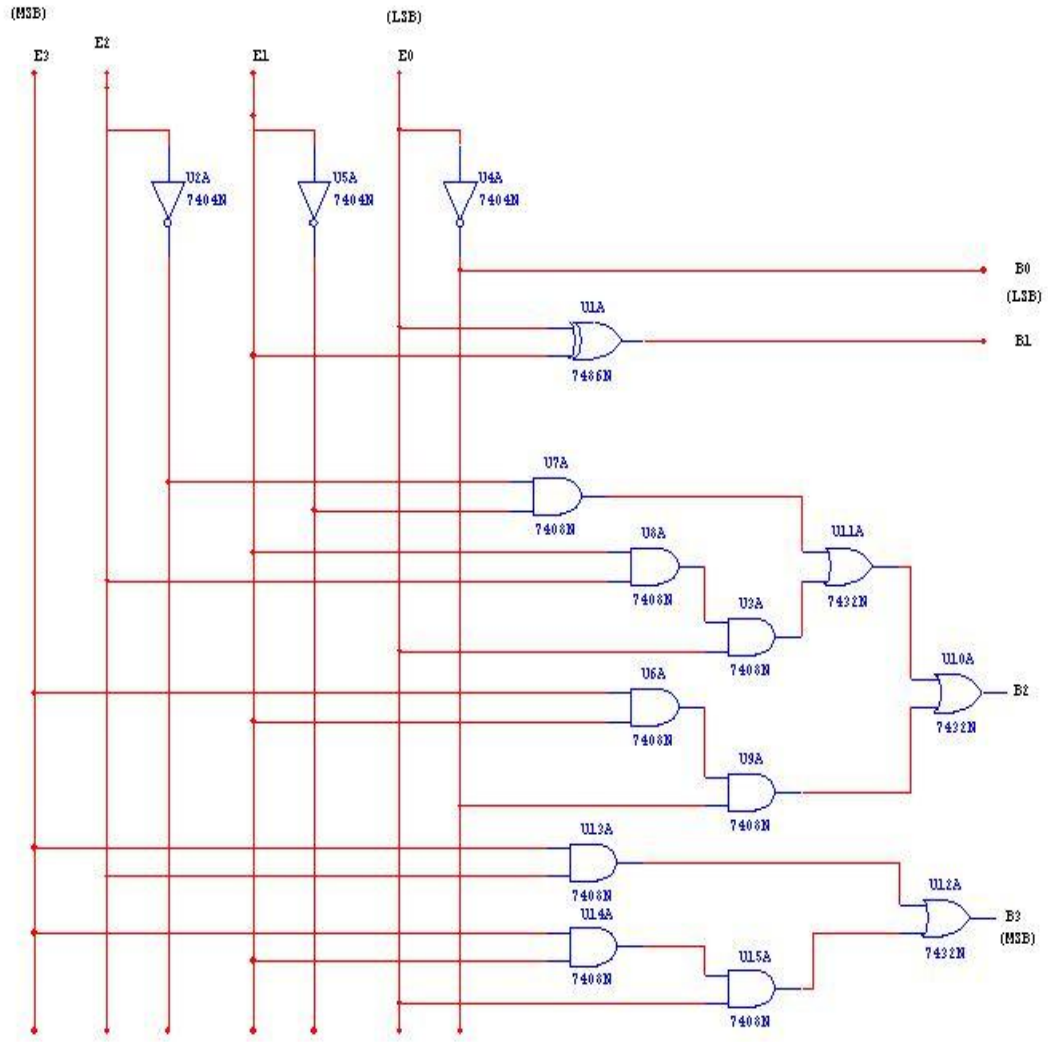


Fig.11 Logical Circuit Diagram for Excess-3 to BCD Conversion

Outcome:

Thus, we studied different codes and their conversions including applications.

The truth tables have been verified using IC 7486, 7432, 7408, and 7404.

Enhancements/modifications:

FAQ's with answers:

Q.1) What is the need of code converters?

There is a wide variety of binary codes used in digital systems. Often it is required to convert from one code to another. For example the input to a digital system may be in natural BCD and output may be 7-segment LEDs. The digital system used may be capable of processing the data in straight binary format. Therefore, the data has to be converted from one type of code to another type for different purpose.

Q.2) What is Gray code?

It is a modified binary code in which a decimal number is represented in binary form in such a way that each Gray- Code number differs from the preceding and the succeeding number by a single bit.

(e.g. for decimal number 5 the equivalent Gray code is 0111 and for 6 it is 0101. These two codes differ by only one bit position i. e. third from the left.) It is non weighted code.

Q.3) What is the significance of Gray code?

Important feature of Gray code is it exhibits only a single bit change from one code word to the next in sequence. Whereas by using binary code there is a possibility of change of all bits if we move from one number to other in sequence (e.g. binary code for 7 is 0111 and for 8 it is 1000). Therefore it is more useful to use Gray code in some applications than binary code.

Q.4) What are applications of Gray code?

1. Important feature of Gray code is it exhibits only a single bit change from one code word to the next in sequence. This property is important in many applications such as Shaft encoders where error susceptibility increases with number of bit changes between adjacent numbers in sequence.

2. It is sometimes convenient to use the Gray code to represent the digital data converted from the analog data (Outputs of ADC).

3. Gray codes are used in angle-measuring devices in preference to straight forward binary encoding.

4. Gray codes are widely used in K-map

Q.5) What are weighted codes and non-weighted codes?

In weighted codes each digit position of number represents a specific weight. The codes 8421, 2421, and 5211 are weighted codes.

Non weighted codes are not assigned with any weight to each digit position i.e. each digit position within the number is not assigned a fixed value. Gray code, Excess-3 code are non-weighted code.

Q.6) Why is Excess-3 code called as self-complementing code?

Excess-3 code is called self-complementing code because 9‘s complement of a coded number can be obtained by just complementing each bit.

Q.7) What is invalid BCD?

With four bits, sixteen numbers (0000 to 1111) can be represented, but in BCD code only 10 of these are used as decimal numbers have only 10 digits fro 0 to 9. The six code combinations (1010 to 1111) are not used and are invalid.

Assignments Questions:

Assignment No: 3

- ***Title:*** **BCD Adder**
- ***Objective:*** To learn different types of adder
- ***Problem Statement***: Design of n-bit Carry Save Adder (CSA) and Carry Propagation Adder (CPA). Design and Realization of BCD Adder using 4-bit Binary Adder (IC 7483).
- ***Hardware and software requirement:***

  Digital Trainer Kit, IC 7483,7432 7408, Patch Cord ,+ 5V Power Supply

Theory:

- **Carry Save Adder**:

  A carry save adder is just a set of one bit full adder, without any carry chaining. Therefore n-bit CSA receivers three n-bit operands,namely A(n-1),A(0) and CIN(n-1)----------CIN(0) and generate two n-bit result values, sum(n-1)----------sum(0) and count(n-1)--------count(0).

- **Carry Propagation Adder**:

  The parallel adder is ripple carry type in which the carry output of each full adder

  stage is connected to the carry input of the next highest order stage.

  Therefore, the sum and carry outputs of any stage cannot be produced until the carry occurs. This leads to a time delay in addition process.

  This is known as Carry Propagation Delay.

- **BCD Adder:** It is a circuit that adds two BCD digits & produces a sum of digits also in BCD.

✓ ***Rules for BCD addition:***

1. Add two numbers using rules of Binary addition.

2. If the 4 bit sum is greater than 9 or if carry is generated then the sum is invalid. To correct the sum add 0110 i.e. (6)10 to sum. If carry is generated from this addition add it to next higher order BCD digit.

3. If the 4 bit sum is less than 9 or equal to 9 then sum is in proper form.

✓ **The BCD addition can be explained with the help of following 3 cases -**

CASE I: Sum <= 9 & carry = 0.

Add BCD digits 3 & 4

1. 0 0 1 1
   + 0 1 0 0
   ---
   0 1 1 1

Answer is valid BCD number = **(7) BCD** & so 0110 is not added.

CASE II: Sum > 9 & carry = 0.

Add BCD digits 6 & 5

1. 0 1 1 0
   + 0 1 0 1
   ---
   1 0 1 1

Invalid BCD (since sum > 9) so 0110 is to be added

2. 1 0 1 1

+ 0 1 1 0

---

1 0 0 0 1

| |

(1 1)BCD

|

Valid BCD result = **(11) BCD**

CASE III: Sum < = 9 & carry = 1.

Add BCD digits 9 & 9

1. 1 0 0 1

+ 1 0 0 1

---

1 0 0 1 0

Invalid BCD (since Carry = 1) so 0110 is to be added

2. 1 0 0 1 0

+ 0 1 1 0

---

1 1 0 0 0

(1 8)BCD

Valid BCD result = **(18) BCD**

Design of BCD adder :

1. 4 bit binary adder is used for initial addition. i.e. binary addition of two 4 bit numbers.(with Cin = 0),
2. Logic circuit to sense if sum exceeds 9 or carry = 1, this digital circuit will produce high output otherwise its output will be zero.
3. One more 4-bit adder to add (0110)2 in the sum is greater than 9 or carry is 1.

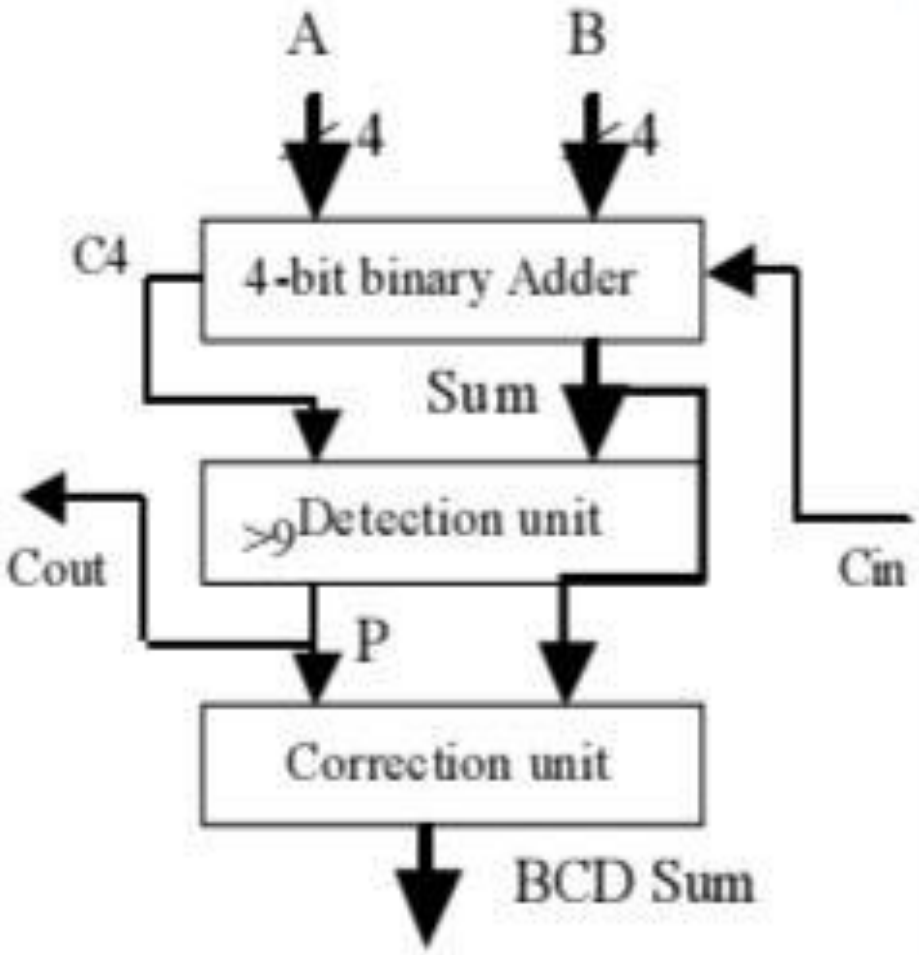


Fig. 1: Block diagram of a BCD adder

Truth Table:-

For design of combinational circuit for BCD adder to check invalid BCD

INPUT				OUTPUT
S3	**S2**	**S1**	**S0**	**Y**
0	**0**	**0**	**0**	**0**
0	**0**	**0**	**1**	**0**
0	**0**	**1**	**0**	**0**
0	**0**	**1**	**1**	**0**
0	**1**	**0**	**0**	**0**
0	**1**	**0**	**1**	**0**
0	**1**	**1**	**0**	**0**
0	**1**	**1**	**1**	**0**
1	**0**	**0**	**0**	**0**
1	**0**	**0**	**1**	**0**
1	**0**	**1**	**0**	**1**
1	**0**	**1**	**1**	**1**
1	**1**	**0**	**0**	**1**
1	**1**	**0**	**1**	**1**
1	**1**	**1**	**0**	**1**
1	**1**	**1**	**1**	**1**

K-map:-

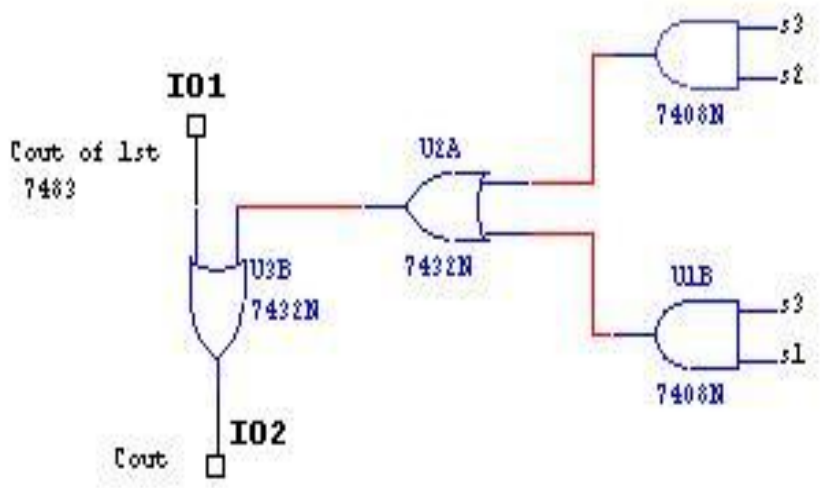
For reduced Boolean expressions of output

S3S2 \ S1S0	00	10	11	01
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

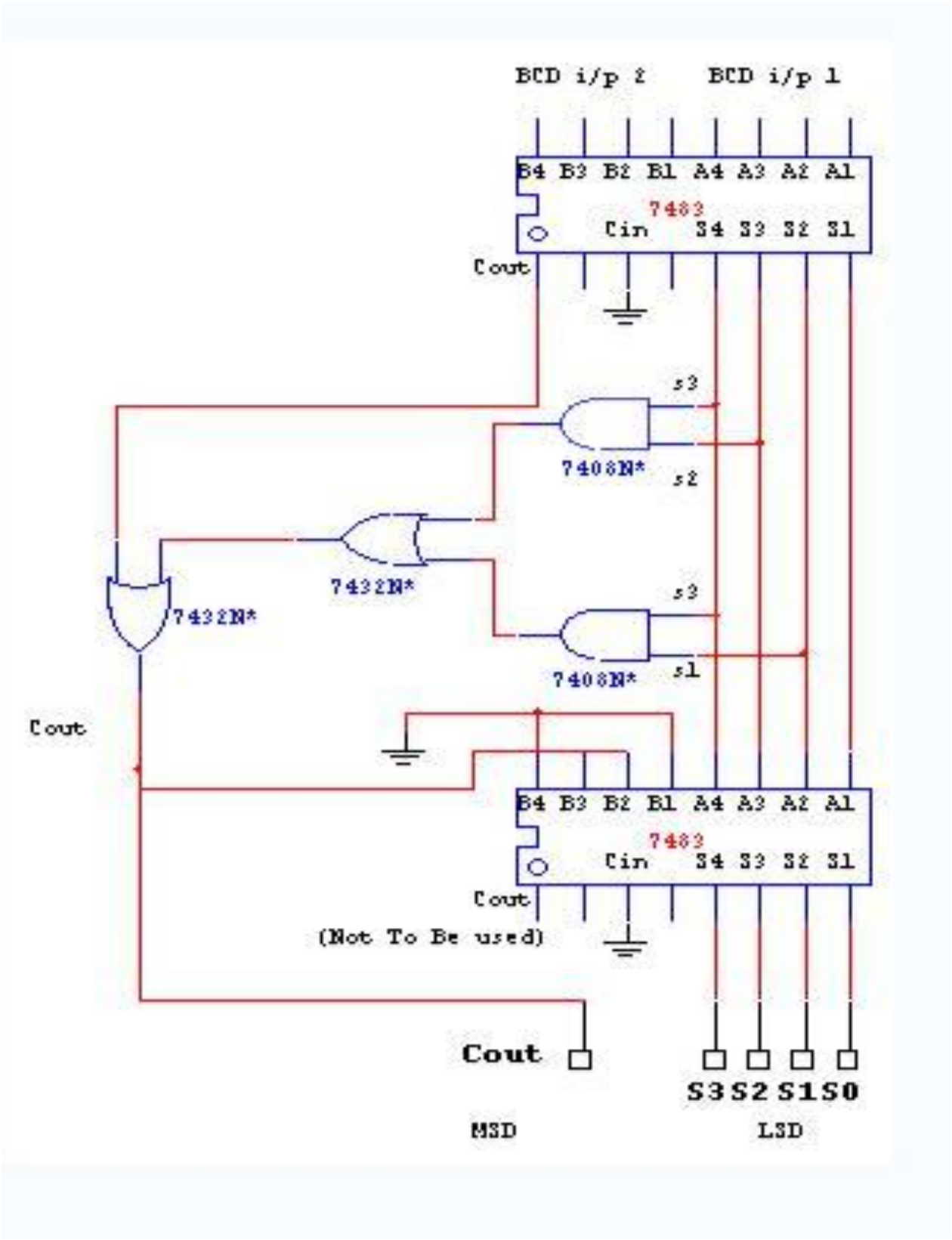
Y= $S_3S_2+S_3S_1$

Circuit diagram:

For invalid BCD detection



iv) Circuit diagram for BCD adder :



Observation Table of BCD adder:

INPUT								OUTPUT				
1st Operand				2nd Operand				MSD	LSD			
A3 (MSB)	A2	A1	A0 (LSB)	B3 (MSB)	B2	B1	B0 (LSB)	Cout	S3 (MSB)	S2	S1	S0 (LSB)

***Outcome*:**

Thus, we studied single bit BCD adder using 4 bit parallel binary adder / 4 bit full adder the observation table has been verified have been verified using IC 7483 & some logic gates.

Assignments Questions:

Assignment No.-4

- ***Title:*** **Realization of Boolean expression using 8:1 Multiplexer 74151**
- ***Objective:*** To learn different techniques of designing multiplexer
- ***Problem Statement:***
  1. Verification of Functional table.
  2. Verification of Sum of Product (SOP) and Product of Sum (POS) with the help of given Boolean expression.
  3. Verify the functional table using cascading of two multiplexers
  4. Realization of Boolean expression using hardware reduction method for the given equation.
- ***Hardware & software requirements:***

  Digital trainer board, IC 74151, IC 7404, IC 7432, patch cords, + 5V Power supply

Theory:

1 .What is multiplexer?

- Multiplexer is a digital switch which allows digital information from several sources to be routed onto a single output line. Basic multiplexer has several data inputs and a single output line.
- The selection of a particular input line is controlled by a set of selection line.
- There are 2^n input lines & n is the number of selection line whose bit combinations determines which input is selected .It is ―Many into One‖.
- *Strobe*: - It is used to enable/ disable the logic circuit ***OR*** _E' is called as enable I/P which is generally active LOW. It is used for cascading
- MUX is a *single pole* multiple way switch.

2. Necessity of multiplexer

- In most of the electronic systems, digital data is available on more than one lines. It is necessary to route this data over a single line.
- It select one of the many I/P at a time.
- Multiplexer improves the *reliability* of digital system because it reduces the number of external wire connection.

3. Enlist significance and advantages of Multiplexer

- It doesn't need K-map & logic simplification.
- The IC package count is minimized.
- It simplifies the logic design.
- In designing the combinational circuit
- It reduces the complexity & cost.
- To minimize number of connections in communication system were we need to handle thousands of connections. Ex. Telephone exchange.

4.Applications of MUX

- Data selector to select one out of many data I/P.
- In Data Acquisition system.
- In the D/A converter.

Multiplexer Tree

- It is nothing but construction of more number of line using less number of lines.
- It is possible to expand the range of inputs for multiplexers beyond the available Range in the integrated circuits. This can be accomplished by interconnecting several multiplexers.

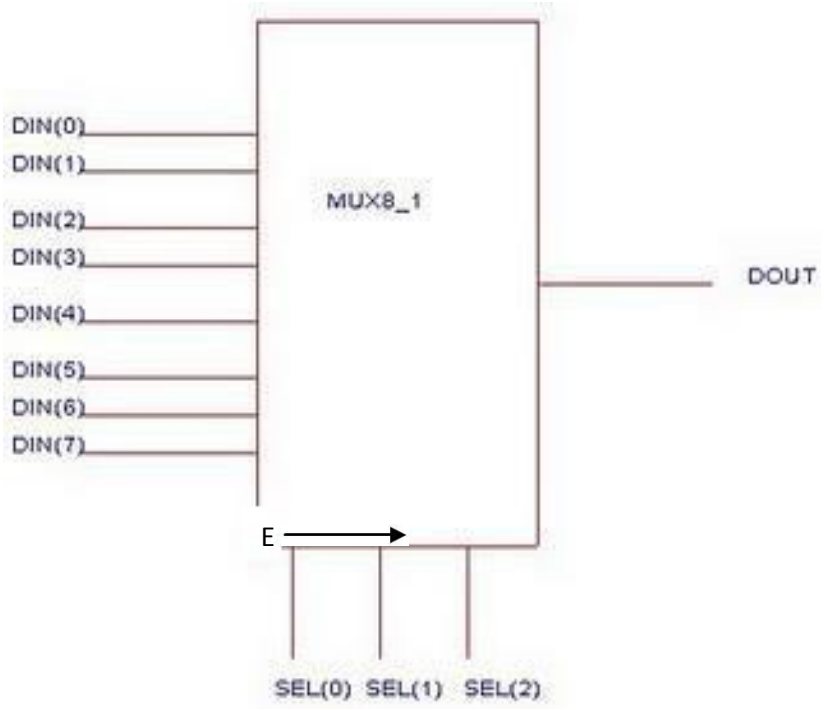
8:1 MUX:

The block diagram of 8:1 MUX & its TT is shown. It has eight data I/P & one enable input, three select lines and one O/P.

Operating principle:

When the Strobe or Enable input is active low, we can select any one of eight data I/P and connect to O/P.

Design:



Draw the connection diagram of multiplexer to verify the functional table.

SELECTION LINES			STROBE	OUTPUTS	
C	B	A	E	Y	$\overline{Y}$
X	X	X	1	0	1
0	0	0	0	D0	D0
0	0	1	0	D1	D1
0	1	0	0	D2	D2
0	1	1	0	D3	D3
1	0	0	0	D4	D4
1	0	1	0	D5	D5
1	1	0	0	D6	D6
1	1	1	0	D7	D7

X = don't care condition.

Part 1: <u>MUX as a function generator</u>.

Convert the given Boolean expression into standard SOP / POS format if required and complete the logic diagram design accordingly for realization of the same.

i) **As an example:**

Function = Sum of Product (SOP)
Y = ∑m (1, 2, 3, 4, 5, 6, 7)

SELECTION LINES			STROBE	OUTPUTS	
C	B	A		Y	$\overline{Y}$
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	0	1	0
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	0	1	0

SOP realization Diagram

SOP Y = ∑m (1, 2, 3, 4, 5, 6, 7)

Solution:-Since there are 3 variable, the multiplexer have 3 select I/P should be used.

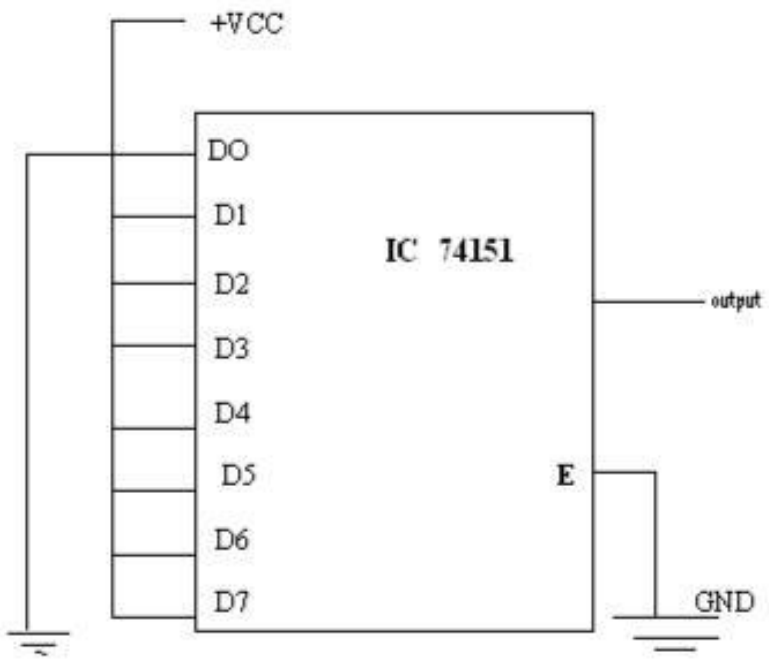
Hence one 8:1 mux should be used.

Ste p 1:-Identify the number decimal corresponding to each minterm.

Here 1,2,3,4,5,6,7

Step 2:-Connect the data input lines 1,2,3,4,5,6,7 to logic 1(+Vcc) & remaining input line 0 to logic 0(GND)

Step 3:-Connect variables A, B & C to select input.



ii) **As an example**

Function = Product of Sum (POS)

Y = ∏ M (0, 5, 6, 7)

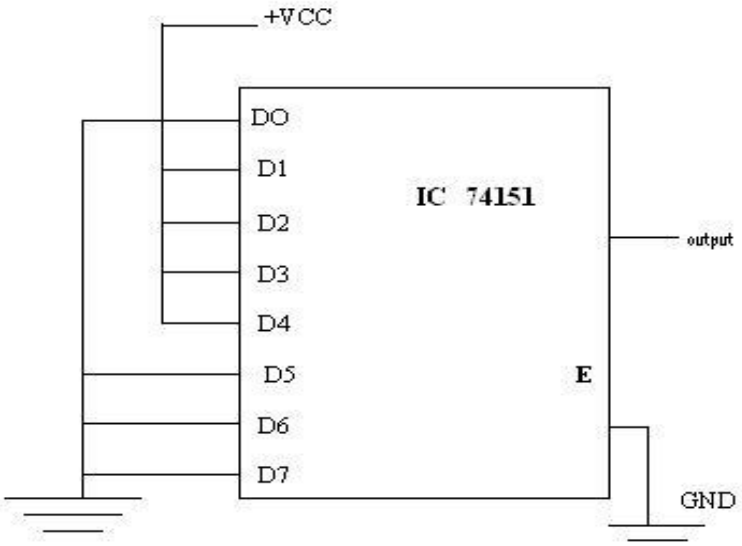
SELECTION LINES			STROBE	OUTPUTS	
C	B	A		Y	$\overline{Y}$
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	0	1	0
1	0	1	0	0	1
1	1	0	0	0	1
1	1	1	0	0	1

POS realization exp:

1. As there are 3 i/p so use 8:1 MUX
2. Connect the given min terms to GND and else decimal numbers to logic1(+VCC).

POS realization Diagram

POS Y = ∏ M (0, 5, 6, 7)



Part -2: Implementation of 16:1 MUX using 8:1 MUX

Use hardware reduction method and implement the given Boolean expression with the help of neat logic diagram. ***(N-circle Method)***

First Method: F(A,B,C,D) = ∑m (2, 4, 5, 7, 10, 14)

	D0	D1	D2	D3	D4	D5	D6	D7
$\overline{A}$	0	1	**2**	3	**4**	**5**	6	**7**
A	8	9	**10**	11	12	13	**14**	15
Input to Mux	0	0	1	0	$\overline{A}$	$\overline{A}$	A	$\overline{A}$

i. *Bold and red marks represent the minterms
ii. Consider B,C,D as a select line
iii. Use NOT gate to obtain A and complement of it.

Solution:-

Step 1:- Apply B, C, D to select I/P & design table(Implementation table).

Step 2: Encircle those min terms which are present in output.

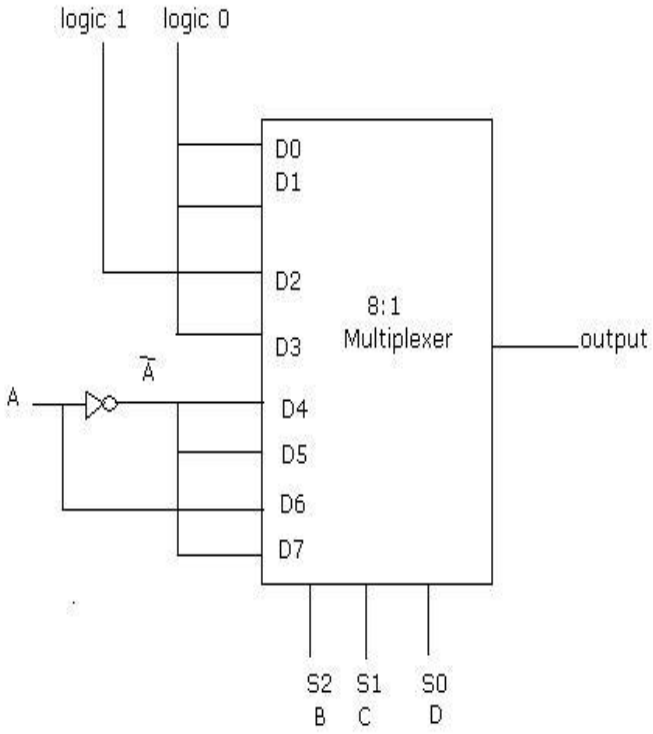
Step 3: If the min terms in a column are not circled then apply logic 0.

Step 4: If the min terms in a column are circled then apply logic 1.

Step 5: If only min term in 2nd row is encircled then _A' should be applied to that data input. Hence apply _A' to D6.

Step 6: If only min term in 1st row is encircled then _ $\overline{A}$ ' should be applied to that data input. Hence apply _ $\overline{A}$ ' to D4, D5, D7.

16:1 MUX using 8:1 MUX Diagram

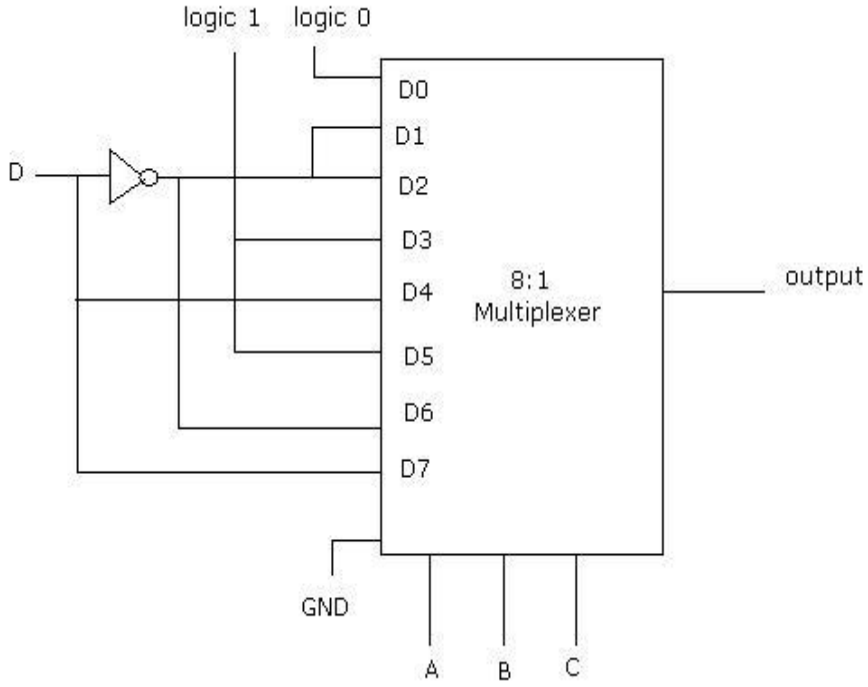


<u>Second method:</u> F (A, B, C, D) = ∑m (2, 4, 6, 7, 9, 10, 11, 12, 15)

1. Make a combination of pair according to same Values of A, B and C
2. Check the output values with respect to value of D.

A	B	C	D	output	output
0	0	0	0	0	0
0	0	0	1	0	
0	0	1	0	1	$\bar{D}$
0	0	1	1	0	
0	1	0	0	1	$\bar{D}$
0	1	0	1	0	
0	1	1	0	1	1
0	1	1	1	1	
1	0	0	0	0	D
1	0	0	1	1	
1	0	1	0	1	1
1	0	1	1	1	
1	1	0	0	1	$\bar{D}$
1	1	0	1	0	
1	1	1	0	0	D
1	1	1	1	1	

16:1 MUX using 8:1 MUX Diagram: Second Method



Part-3 **Implementation of 16:1 MUX using two 8:1 MUX (Cascading Method)**

$$F(A,B,C,D)= \sum m(2, 4, 5, 7, 10, 14)$$

Solution:-

Step 1: Connect S2, S1, S0 select lines of two 8:1 MUX parallel where as MSB select input is used for enabling MUX.

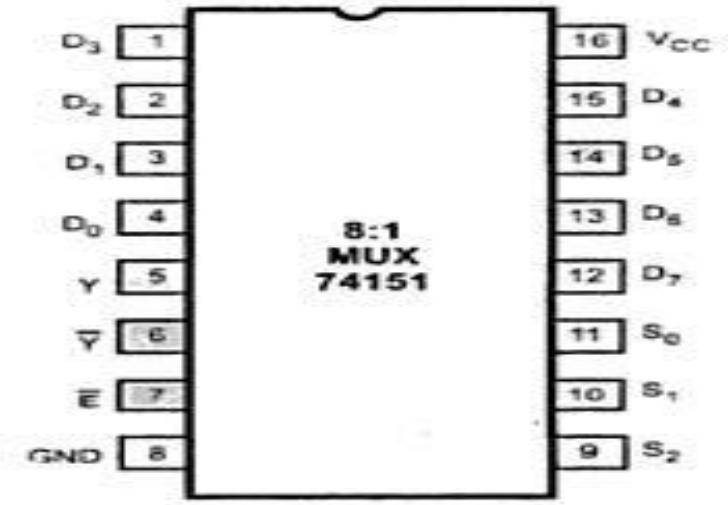
Step 2: S3 is connected directly to the enable (E) to mux-2 where as $\overline{S3}$ is connect to enable input of mux-1

Step 3: The output of two MUX are OR to get final output.

Truth table: - PART_3

Select line				Output		Final Output
S3	S2	S1	S0	Y1	Y2	Y
0	0	0	0	D0	--	D0
0	0	0	1	D1	--	D1
0	0	1	0	D2	--	D2
0	0	1	1	D3	--	D3
0	1	0	0	D4	--	D4
0	1	0	1	D5	--	D5
0	1	1	0	D6	--	D6
1	1	1	1	D7	--	D7
1	0	0	0	--	D8	D8
1	0	0	1	--	D9	D9
1	0	1	0	--	D10	D10
1	0	1	1	--	D11	D11
1	1	0	0	--	D12	D12
1	1	0	1	--	D13	D13
1	1	1	0	--	D14	D14
1	1	1	1	--	D15	D15

Truth Table for 16:1 MUX using two 8:1 MUX



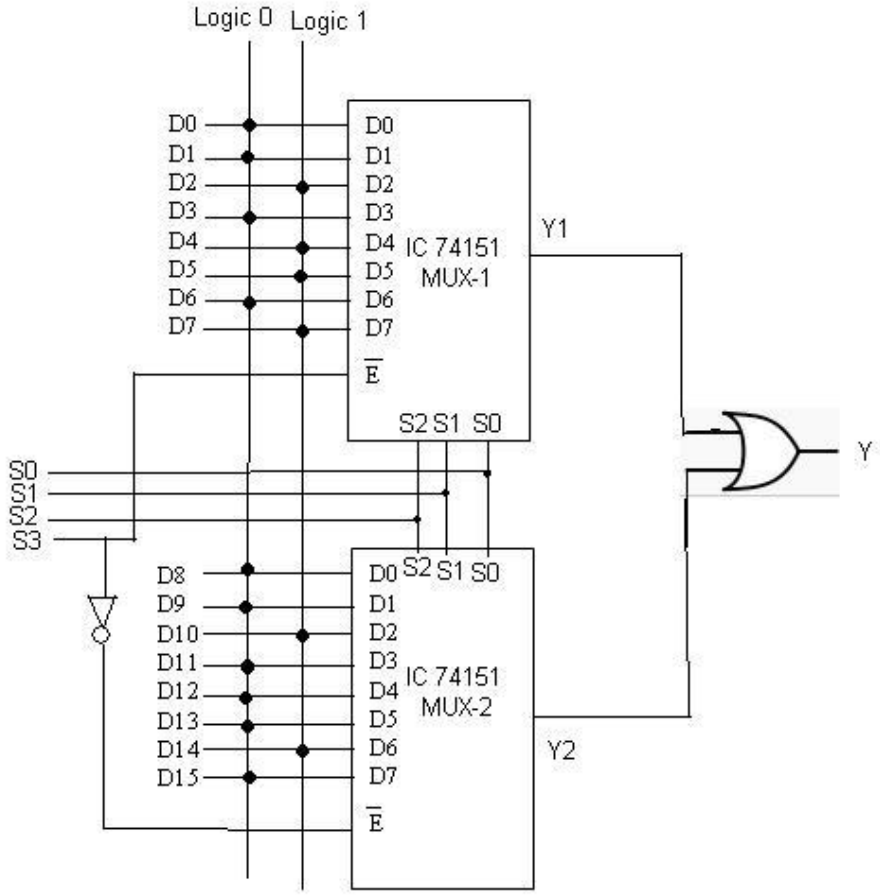
Pin DiagramIC 74151

8:1 mux

$\overline{Y}$: Complemented output

$\overline{E}$: Active low enable input

Multiplexer Tree according to given equation:-



Outcome:

Multiplexer is used as a data selector to select one out of many data inputs.

It is used for simplification of logic design.

It is used to design combinational circuit.

Use of multiplexer minimizes no. of connections.

FAQ:

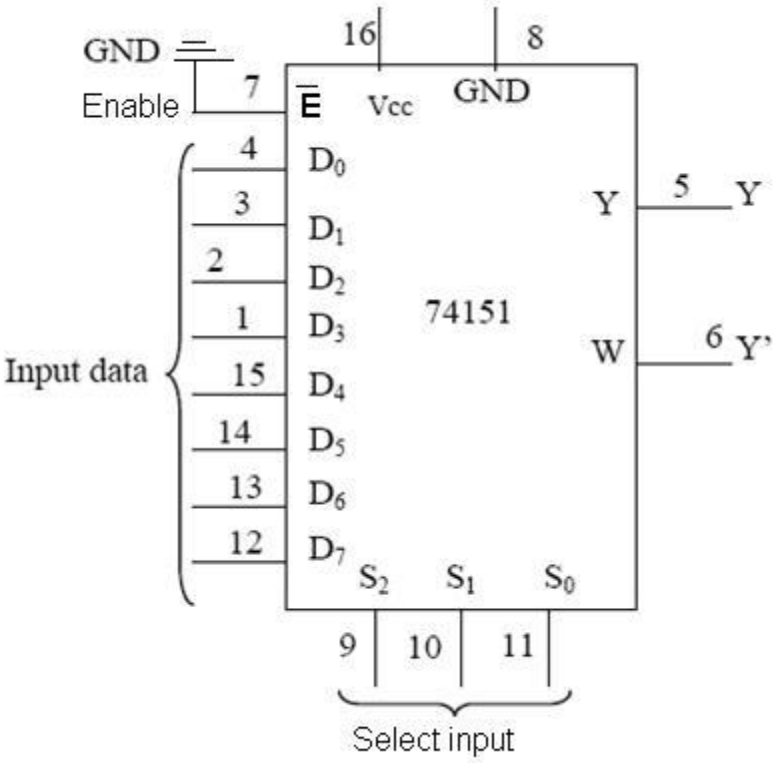
.

1. ***Enlist applications of MUX***

➔

1. MUX is used as data selector.
2. It is used to design combinational circuit.

3. Less number of wires required which reduces complexity
4. There is no need to design k-map
5. We design equation using truth table.



2. Define the terms Encoder and Decoder

➔ Encoders are used to encode given digital number into different numbering format .like decimal to BCD Encoder, Octal to Binary.

➔ Decoders are used to decode a coded binary word like BCD to seven segment decoder. Thus encoder and decoder are application specific logic develop, we cannot use any type of input for any encoder and decoder.

Assignments Questions:

Assignment No: 5

- **Title:** - Comparators
- **Objective: -** 1 bit, 2 bit Comparator.
- ***Problem Statement:*** To verify truth table of 1 bit and 2 bit comparator using logic gate and comparator IC.
- **Hardware & Software Requirement's :**

  Digital Trainer Kit, Comparator IC-7485, patch cords, +5V power supply.

Theory:

Another common and very useful combinational logic circuit is that of the **Digital Comparator** circuit. Digital or Binary Comparators are made up from standard AND, NOR and NOT gates that compare the digital signals present at their input terminals and produce an output depending upon the condition of those inputs.

For example, along with being able to add and subtract binary numbers we need to be able to compare them and determine whether the value of input A is greater than, smaller than or equal to the value at input B etc. The digital comparator accomplishes this using several logic gates that operate on the principles of *Boolean algebra*. There are two main types of **Digital Comparator** available and these are

1. Identity Comparator – an *Identity Comparator* is a digital comparator that has only one output terminal for when A = B either ―HIGH‖ A = B = 1 or ―LOW‖ A = B = 0

- 2. Magnitude Comparator – a *Magnitude Comparator* is a digital comparator which has three output terminals, one each for equality, A = B greater than, A > B and less than A < B.

The purpose of a **Digital Comparator** is to compare a set of variables or unknown numbers, for example A (A1, A2, A3… An, etc) against that of a constant or unknown value such as B (B1, B2, B3… Bn, etc) and produce an output condition or flag depending upon the result of the comparison. For example, a magnitude comparator of two 1-bits, (A and B) inputs would produce the following three output conditions when compared to each other. Which means: A is greater than B, A is equal to B, and A is less than B

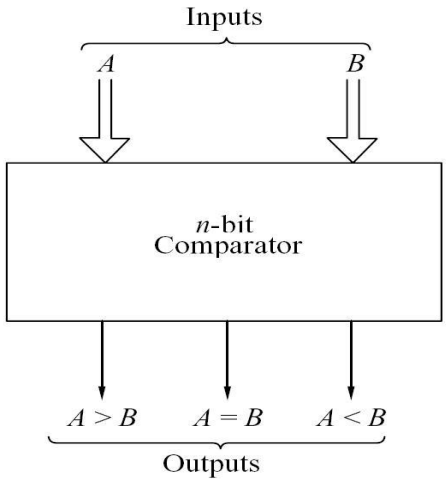


Fig. 6.22 ***Block Diagram of***

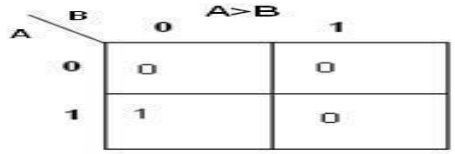
This is useful if we want to compare two variables and want to produce an output when any of the above three conditions are achieved. For example, produce an output from a counter when a certain count number is reached. Consider the simple 1-bit comparator below

1. **1-bit comparator**

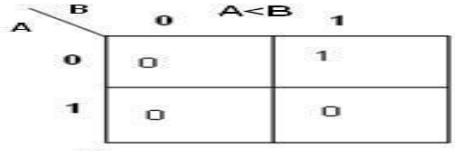
- **Truth Table:-**

Inputs		Outputs		
B	A	A > B	A = B	A < B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

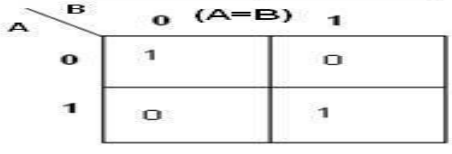
- **K-Map**



Equation is $A>B = A.\overline{B}$



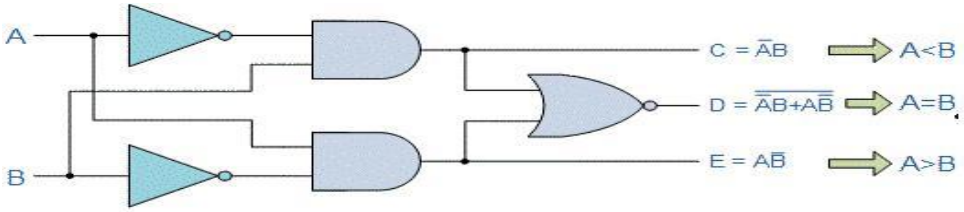
Equation is $A<B = \overline{A}.B$



The equation is $f(A=B) = \overline{A}.\overline{B} + A.B$
$= A \text{ XNOR } B$

or we can write the equation for $f(A=B)$ as $\overline{\overline{A}.\overline{B} + \overline{A}.B} = \overline{f(A>B)+f(A<B)}$

- **Logic Diagram of 1 bit Comparator**



---

2 Bit Comparator:-

- **Truth Table:-**

Inputs				Outputs		
A_1	A_0	B_1	B_0	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

- **K-map :-**

1. For A>B:

f(A>B)

A_1A_0 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

We get the equation as f(A>B)

$= A_1\overline{B_1} + A_0\overline{B_1}\,\overline{B_0} + A_1A_0\overline{B_0}$

2. For A=B

f(A=B)

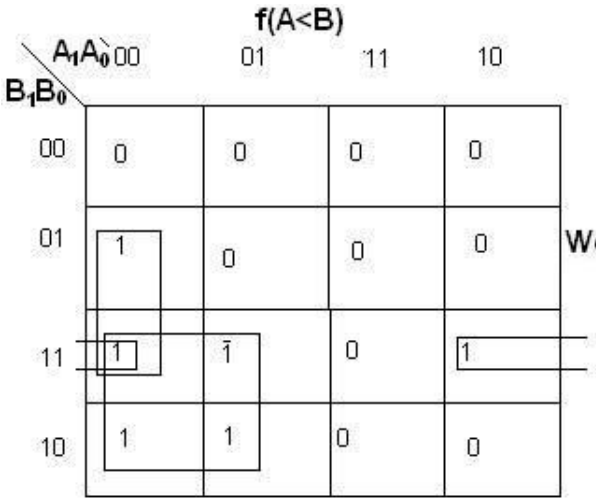
A_1A_0 \ B_1B_0	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

We get the equation as f(A=B)

$=(A_1 \text{XOR } B_1).(A_0 \text{XOR } B_0)$

or we can write the equation for f(A=B) as $= \overline{f(A>B)+f(A<B)}$

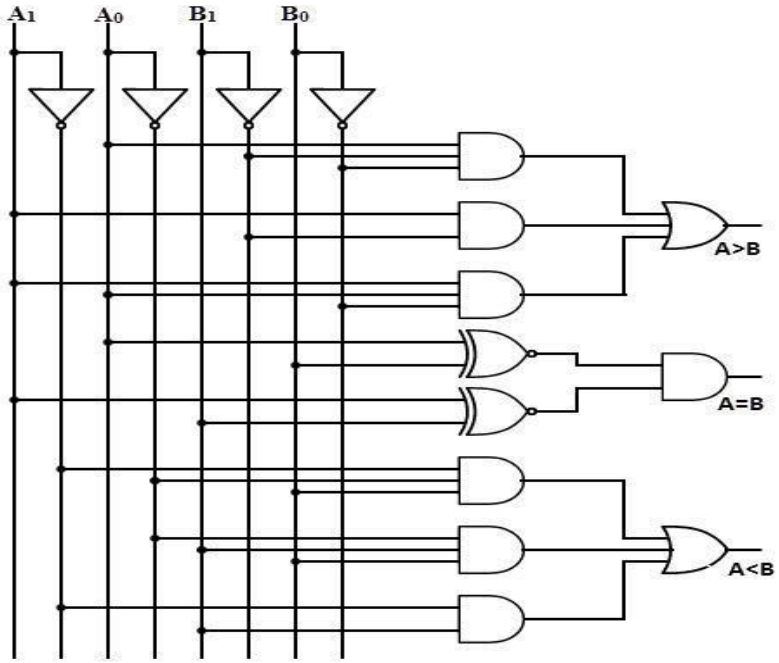
3. For A<B :-

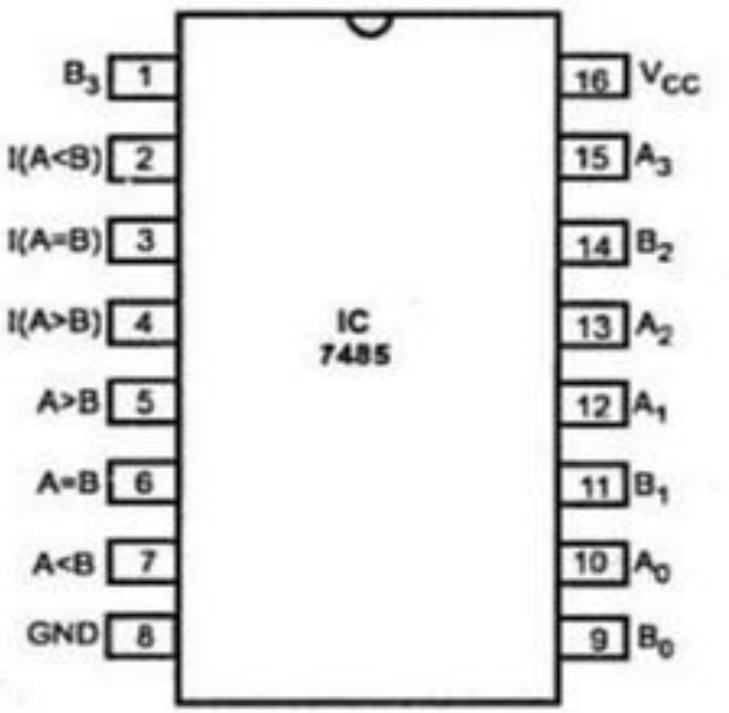


We get the equation as f(A<B)

$$= \overline{A_1}B_1 + \overline{A_0}B_1B_0 + \overline{A_1}\overline{A_0}B_0$$

- **Circuit Diagram:-**





(a) Pin diagram (IC 7485)

- **For n bit Comparator :-**

Digital comparators actually use Exclusive-NOR gates within their design for comparing their respective pairs of bits. When we are comparing two binary or BCD values or variables against each other, we are comparing the ―magnitude‖ of these values, a logic ―0‖ against a logic ―1‖ which is where the term **Magnitude Comparator** comes from.

As well as comparing individual bits, we can design larger bit comparators by cascading together n of these and produce a n-bit comparator just as we did for the n-bit adder in the previous tutorial. Multi-bit comparators can be constructed to compare whole binary or BCD words to produce an output if one word is larger, equal to or less than the other.

Outcome:

Up and down counters are successfully implemented, the comparators are studied & o/p are checked. The truth table is verified.

Assignments Questions:

Assignment: 6

- ***Title:*** **Parity Generator and Parity Checker**.
- ***Objective:*** Learn Even/Odd parity Generator/Checker using logic gates
- ***Problem Statement***: Design & Implement Parity Generator using EX-OR
- ***Hardware & software requirement:*** Digital Trainer Kit, IC 7486 (Ex-OR), IC 7404 (NOT), IC 74180, Patch Cord ,+ 5V Power Supply

Theory:

In digital communication, the digital data is sent over the telephone lines using different binary codes.

During the transmission, because of ―*noise*‖(i.e: Unwanted voltage fluctuation) , signal 0 may become 1 or 1 may become 0 and wrong information (i.e: corrupted data) may be received at the destination and must be resent.

This problem of communication is overcome by using Error-detecting code. To detect these errors, *Parity Bit* is usually transmitted along with the data bits.

At the receiving end, parity will be checked.

Parity: A term used to specify the number of one‘s in a digital word as odd or even.

There are two types of Parity - even and odd.

Even Parity Generator will produce a logic 1 at its output if the data word contains an odd number of ones. If the data word contains an even number of ones then the output of the parity generator will be low. By concatenating the Parity bit to the data word, a word will be formed which always has an even number of ones i.e. has even parity.

Parity bit: An extra bit attached to a binary word to make the parity of resultant word even or odd. Parity bits are extra signals which are added to a data word to enable error checking.

Definition:- 2 A check bit appended to an array of binary digit to make the sum of all binary digits.

Parity generator: A logic circuit that generates an additional bit which when appended to a digital word makes its parity as desired (odd or even).

- Parity generators calculate the parity of data packets and add a parity amount to them.

- Parity is used on communication links (e.g. Modem lines) and is often included in memory systems.

Parity checker: At the receiving end a logic circuit is used to check the parity of received information, and determines whether the error is included in the message or not.

Even bit Parity *Code*: The total number of ones in parity code word is even.

Odd bit Parity Code: The total number of ones in parity code word is odd.

The single parity bit code can detect the single bit error. If error is more than *1* bit, it is not possible to detect the error.

Eg:- 1) Assume the even parity code word is sent by the transmitter is *10111,* the code word received by the receiver is *10011.*

The parity of received code word is odd, it shows that one bit error is introduced over the channel.

Eg:- 2) But if the received code word is *10001,* the parity of received code is even and shows that there is no error introduced over the channel.

Actually two bits are changed over the path.

Limitations: -

1) The one bit parity code word can detect *one bit* error.

2) It cannot detect the *location* of error and hence error cannot be corrected

A) Even Parity Generator:

1) *Truth Table:*

INPUT			OUTPUT
B_2	B_1	B_0	P
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

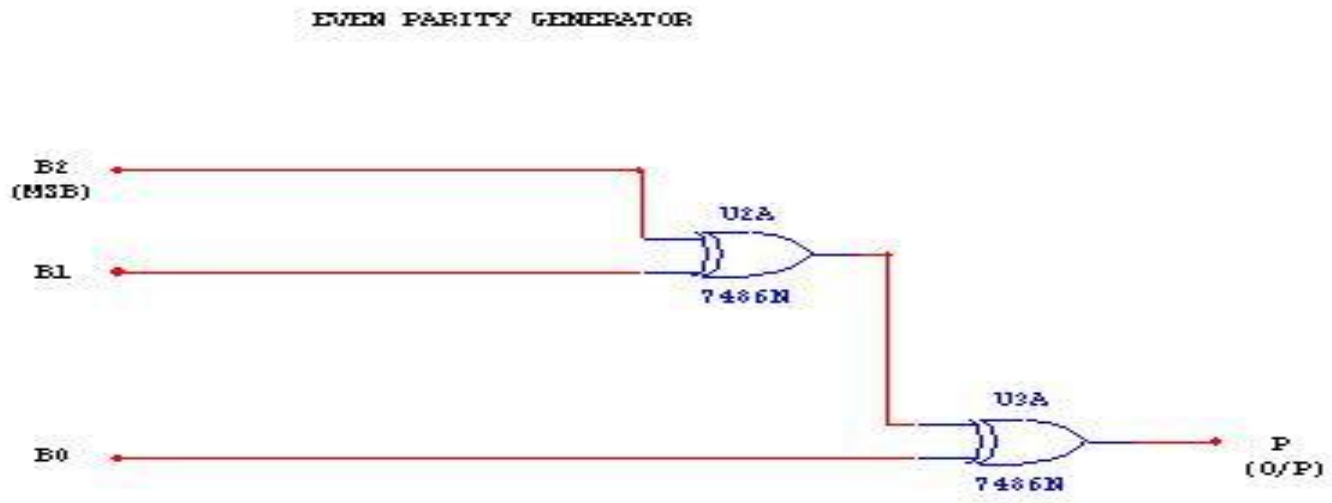
2) K-Map For Reduced Boolean Expressions Of Output:

B_2 \ B_1B_0	00	01	11	10
0	0	1	0	1
1	1	0	1	0

P = B2 (EX-OR) **B1** (EX-OR) **B0**

3) Circuit Diagram:

Even parity generator:



Even parity generator O/p P = B2 (EX-OR) **B1** (EX-OR) **B0**

B) Odd Parity Generator:

1) *Truth Table:*

INPUT			**OUTPUT**
B_2	**B_1**	**B_0**	**P**
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

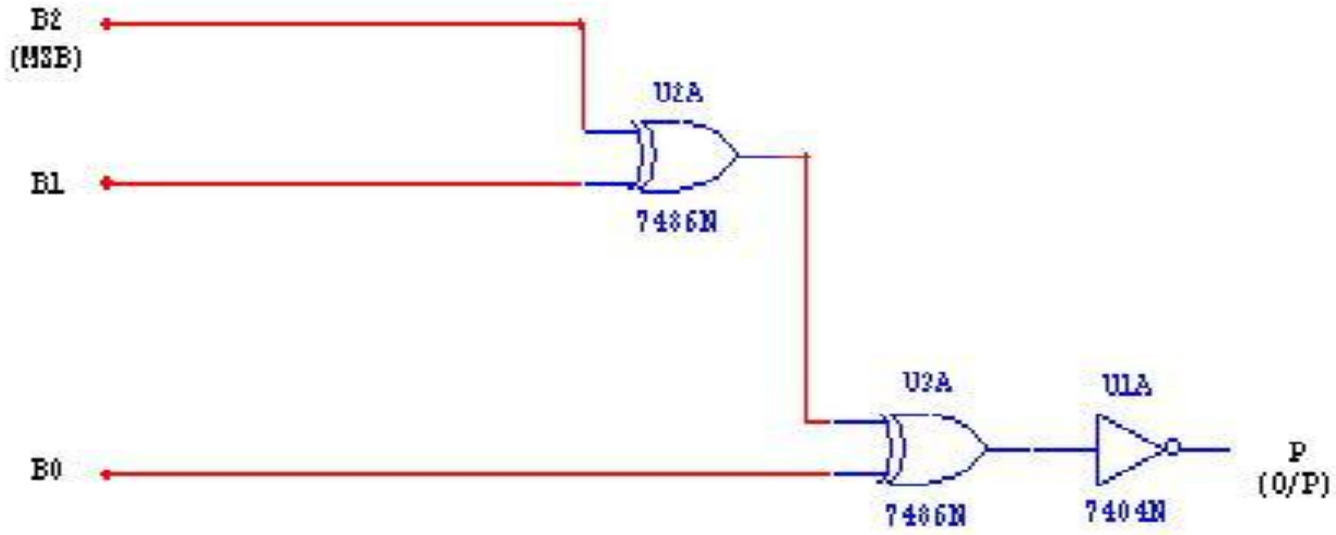
2) K-Map For Reduced Boolean Expressions Of Output:

B_2 \ B_1B_0	00	01	11	10
0	1	0	1	0
1	0	1	0	1

P = B2 (EX-NOR) **B1** (EX-NOR) **B0**

3) Circuit Diagram:

Odd parity generator:



Odd parity generator O/p P = B2 (EX-NOR) **B1** (EX-NOR**) B0**

C) Even Parity Detector:

1) ***Truth Table:***

0– Error

1– No Error

INPUT				OUTPUT
P Parity Bit	**B_2**	**B_1**	**B_0**	**PEC**
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

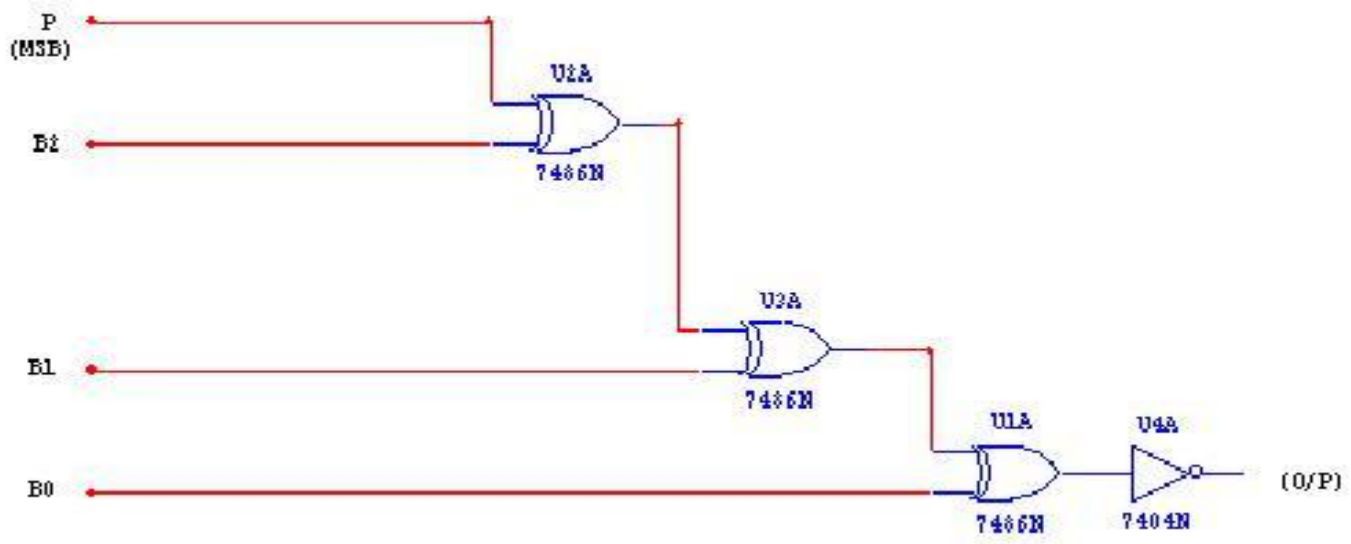
2) K-Map For Reduced Boolean Expressions Of Output:

PB_2 \ B_1B_0	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

PEC = P (EX- NOR) **B2** (EX- NOR) **B1** (EX- NOR) **B0**

3) Circuit Diagram:

Even parity detector:



Even parity detector O/p: PEC **= P** (EX- NOR) **B2** (EX- NOR) **B1** (EX- NOR) **B0**

d) Odd Parity Detector:

1) ***Truth Table:***

0– Error

1– No Error

INPUT				OUTPUT
P Parity Bit	**B_2**	**B_1**	**B_0**	**PEC**
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

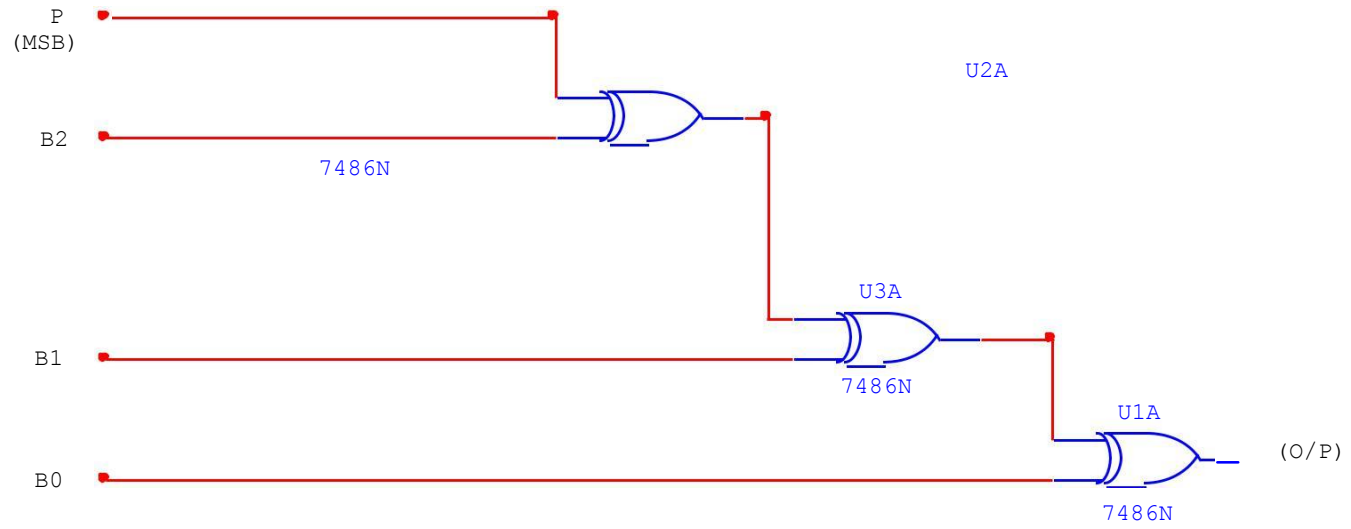
2) K-Map For Reduced Boolean Expressions Of Output:

PB 2 B1B0	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

PEC = P (EX- OR) **B2** (EX- OR) **B1** (EX- OR) **B0**

3) Circuit Diagram:

Odd parity detector:



Odd parity detector O/p: PEC = P (EX- OR) **B2** (EX- OR) **B1** (EX- OR) **B0**

Outcome: Thus, we studied parity generator / checker and their working & limitation

GROUP – B

Assignment No: 7

- ***Title:*** Flip-flop.
- ***Objective:*** Conversion of Flip-flop.
- ***Problem Statement:*** Conversion from one type of flip-flop to another type of flip-flop.

.

- ***Hardware & Software Requirement's :***

Digital Trainer Kit, IC 7476, IC 7474, IC 7408, IC 7432 & IC 7404.patch cords, +5V power supply.

Theory:

A Flip – flop is an electronic device which is having two stable states and a feedback path which is used to store 1 – bit of information by using the clock signal as input. Latches are also used to do the same task except that they do not use a clock signal. Hence to say it simply, ―Flip – flops are clocked latches‖. They are used to store only 1 – bit of information and it can remain in the same state until the clock signal affects the state of the input.

There are four types of flip – flops

- SR flip – flop
- D flip – flop
- JK flip – flop
- T flip-flop

Generally, JK flip – flops and D flip – flops are the most widely used flip – flops. And so their availability in the form of integrated circuits (IC's) is abundant. Numerous varieties of JK flip – flop and D flip – flop are available in the semiconductor market. The less popular SR flip – flop and T flip – flop are not available in the market as integrated circuits (IC's) (even though a very few number of SR flip – flops are available as IC's, they are not frequently used).

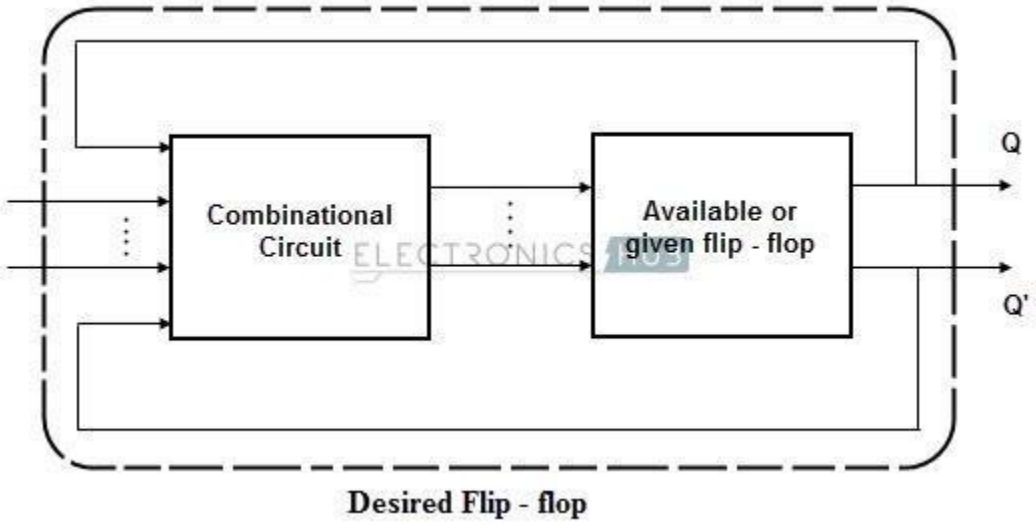
There might be a situation where the less popular flip – flops are required in order to implement a logic circuit. In order use the less popular flip – flops, we will convert one type of flip – flop into another. Some of the most common flip – flop conversions are:-

1. SR Flip – flop to JK Flip – flop
2. SR Flip – flop to D Flip – flop
3. SR Flip – flop to T Flip – flop
4. JK Flip – flop to SR Flip – flop
5. JK Flip – flop to D Flip – flop

6. JK Flip – flop to T Flip – flop
7. D Flip – flop to SR Flip – flop
8. D Flip – flop to JK Flip – flop
9. D Flip-flop to T Flip-flop

- **General model used to convert one type of FF to other**

In order to convert one flip – flop to other type of flip – flop, we should design a combinational circuit that is connected to the actual flip – flop. Inputs to combinational circuit are same as the inputs of the desired flip – flop. Outputs of combinational circuit are same as the inputs of the available flip – flop. So the output of combinational circuit is connected to the input of our available flip – flop. The pictorial representation of the same is shown below.



1. SR Flip – flop to JK Flip – flop

Here we are required to convert the SR flip – flop to JK flip – flop. So first we design a combinational circuit with J and K as its inputs and we connect its output to the input of our available flip flop i.e. an SR flip – flop. So its outputs are same as that of JK flip – flop.

Let‘s write a truth table for the two inputs, J and K. For two inputs along with the Q_P, we get 8 possible combinations in truth table. Consider that when the two inputs are applied, Q_P is the *present state* and Q_N is the *next state*. For every combination of J, K , Q_P , we find the corresponding Q_N state. Here Q_N will give the state values that to which the output of the JK flip – flop will jump after the present state, on applying the inputs. Now we write all the combinations of S and R in the truth table to get each Q_N value from corresponding Q_P. Hence these are the values of S and R that are used to change the state of flip flop from Q_P to Q_N.

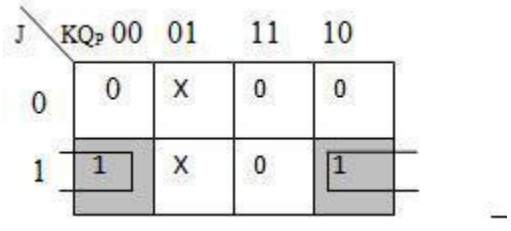
The conversion table:-

From SR flip – flop to JK flip – flop is shown below.

F/F INPUTS		PRESENT STATE	NEXT STATE	OUTPUTS	
J	K	Q_n	Q_{n+1}	S	R
0	0	0	0	0	X
0	1	0	0	0	X
1	0	0	1	1	0
1	1	0	1	1	0
0	1	1	0	0	1
1	1	1	0	0	1
0	0	1	1	X	0
1	0	1	1	X	0

In order to deduce the Boolean equations of S and R in terms of J and K, we use Karnaugh maps from the above table.

The K – map:-



The Boolean equation for S is $S = JQ_P$

.***K map :-***

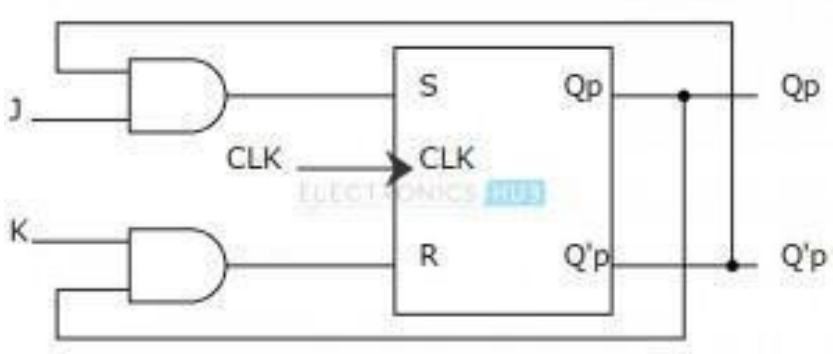
J \ KQ_P	00	01	11	10
0	X	0	1	X
1	0	0	1	0

The Boolean equation for R is $R = KQ_P$.

The Boolean equations of S and R in terms of J, K and Q_P are: $S = JQ'_P$ and $R = KQ_P$

***The logic diagram*:-**

JK flip – flop implemented from SR flip – flop is shown below. Here J and K are external inputs to the circuit. S and R are the outputs of the designed combinational outputs.



2. SR Flip – flop to D Flip – flop

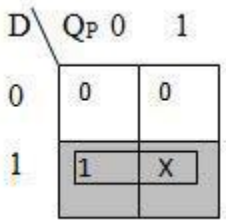
Converting the SR flip – flop to D flip – flop involves connecting the Data input (D) to the SR flip – flop. Here the Data input is connected directly to the S input and the inverted D input (using a NOT gate) is connected to R input. The same can be derived from truth table and corresponding K – maps. S and R are the inputs of the flip – flop while Q_P and Q_P' are the present state and its complementary outputs of the flip – flop. We should design a combinational circuit such that its input is D and outputs are S and R. Outputs from the combinational circuit S and R are connected as inputs to the SR flip – flop.

The truth table for conversion of SR flip – flop to D flip – flop is shown below. The truth table is drawn for the D input and Q_P output to find the corresponding Q_N output.

The conversion table:-

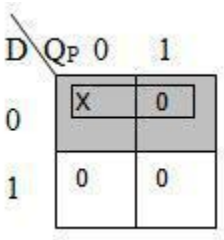
F/F INPUTS	PRESENT STATE	NEXT STATE	OUTPUTS	
D	Q_n	Q_{n+1}	S	R
0	0	0	0	X
1	0	1	1	0
0	1	0	0	1
1	1	1	X	0

The K – map:-



The Boolean equation of S is S = D.

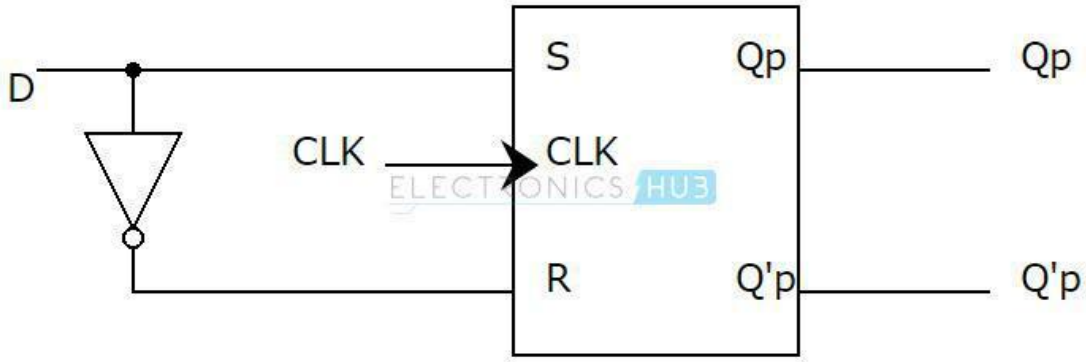
The K – map:-



The Boolean equation of R is $R = \overline{D}$.

The Boolean equation for S and R in terms of D are: S = D and R = D. The logic diagram of implementation of D flip – flop from SR flip – flop is shown below.

***The logic diagram*:-**



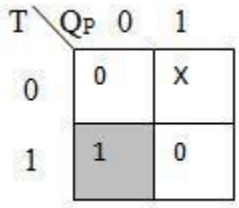
3. SR Flip – flop to T Flip – flop

The combinational circuit required in order to convert an SR flip – flop to T flip – flop can be constructed from the truth table. The input to the combinational circuit is T (Toggle input) and the outputs of the combinational circuit are S and R. Here S and R are the inputs of the actual flip – flop. The output and the complement output of the flip – flop are Q_P and Q'_P. The truth table consists of combinations of T and Q_P in order to get Q_N where Q_N is the next state output of the flip – flop. The combinations of S and R which results in Q_N are also tabulated in the same table.

The conversion table:-

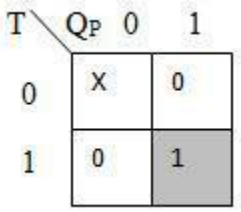
F/F INPUT	PRESENT STATE	NEXT STATE	OUTPUT	
T	Q_n	Q_{n+1}	S	R
0	0	0	0	X
1	0	1	1	0
1	1	0	1	0
0	1	1	X	0

The K – map:-



The Boolean equation of S is $S = T\overline{Q}_P$.

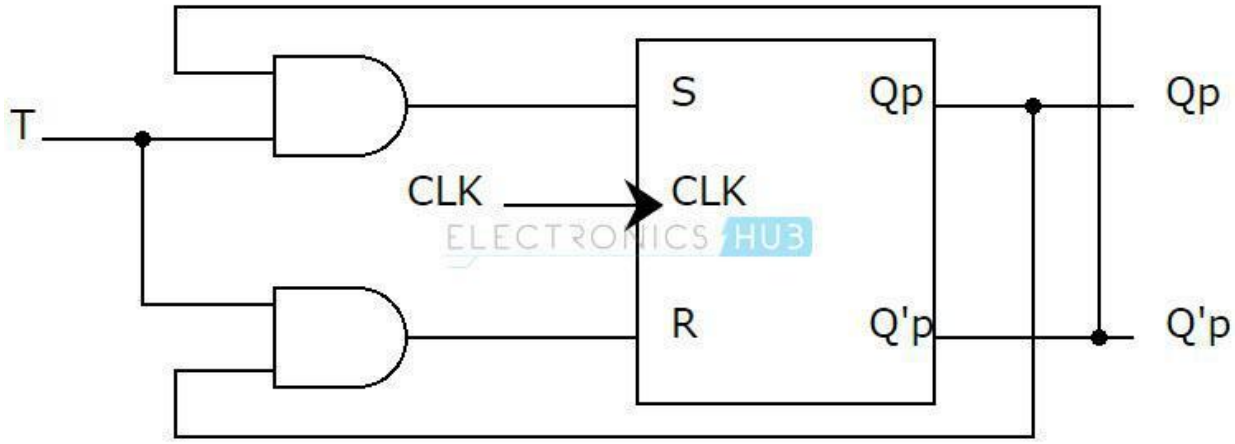
The K – map:-



The Boolean equation for R is $R = TQ_P$.

The Boolean equations of S and R are: $S = T\overline{Q}_P$ and $R = TQ_P$. The logic circuit for the implementation of T flip – flop from SR flip – flop is shown below.

***The logic diagram*:-**



JK Flip – flop to other Flip – flops

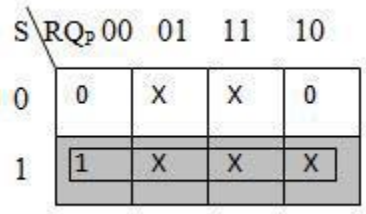
4. JK Flip – flop to SR Flip – flop

To convert the JK flip – flop into SR flip – flop, we design a combinational circuit with S and R as its inputs and J and K as its outputs. Here J and K are the inputs of actual flip – flop. So for making this conversion, we should obtain the J &amp, K values in terms of S, R and Q_P. Consider that when the two inputs S and R are applied, Q_P is the present state output and Q_N is the next state output. For each combination of S, R and Q_P, we find the corresponding Q_N state. Now, we prepare a truth table for the possible combination of the inputs S, R and Q_P. We can make 8 possible combinations for the two S and R inputs along with Q_P. For each combination of S and R inputs and Q_P we find the corresponding value of Q_N. Now we write all the values of J and K in the truth table to get each Q_N value from corresponding Q_P. In SR flip – flop, when the 2 inputs are high i.e. S = 1 & R = 1,

The conversion table:-

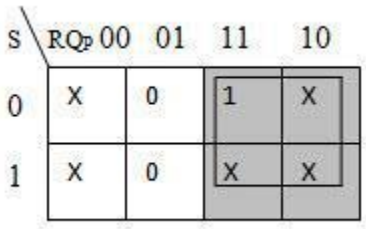
F/F INPUTS		PRESENT STATE	NEXT STATE	OUT PUT	
S	R	Q_n	Q_{n+1}	J	K
0	0	0	0	0	X
0	1	0	0	0	X
1	0	0	1	1	X
1	0	0	1	1	X
0	1	1	0	X	1
0	1	1	0	X	1
0	0	1	1	X	0
1	0	1	1	X	0

The K – map:-



The Boolean equation for J is J = S.

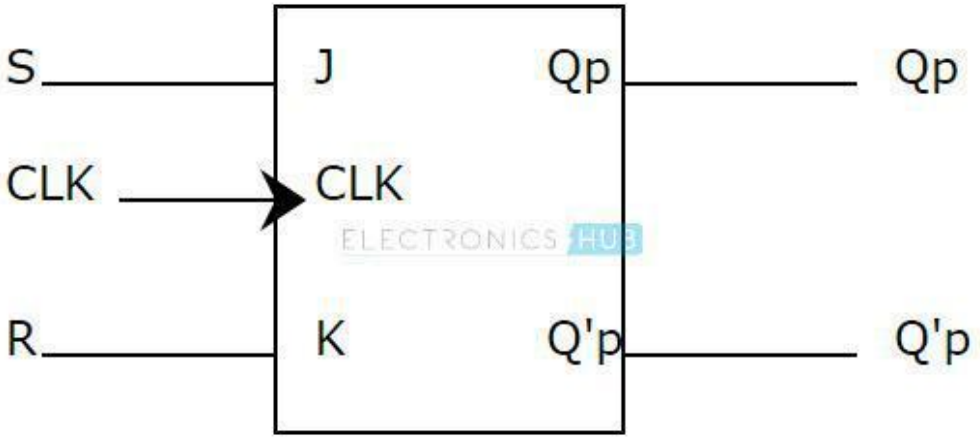
The K – map:-



The Boolean equation for K is K = R.

The Boolean equations for J and K in terms of S and R are: J = S and K = R. Hence, there is no requirement of any additional combinational circuit as S and R inputs are same as J and K inputs. The logic circuit of implementing SR flip – flop from JK flip – flop is shown below.

***The logic diagram*:-**



5. JK Flip – flop to D Flip – flop

Converting the JK flip – flop to D flip – flop, involves in connecting the Data input (D) to the JK flip – flop through a combinational circuit. Here the Data input is connected directly to the J input and the inverted D input (using a NOT gate) is connected to K input.

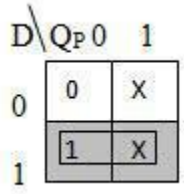
The design of the combinational circuit should be in such a way that D is its input and J & K are its outputs. The outputs of the combinational circuit J & K are connected as inputs to the flip – flop. QP is the present state output of the flip – flop. Q'P is its complementary and QN is the next state output. The truth table for converting JK flip – flop to D flip – flop is shown below.

The conversion table:-

The K – maps in order to solve for J and K in terms of D and QP are shown below.

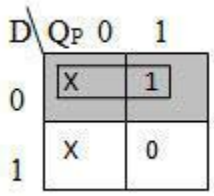
F/F INPUTS	PRESENT STATE	NEXT STATE	OUTPUTS	
D	Q_n	Q_{n+1}	J	K
0	0	0	X	0
1	0	1	X	1
0	1	0	0	X
1	1	1	1	X

K – Map:-



The Boolean equation for J is J = D.

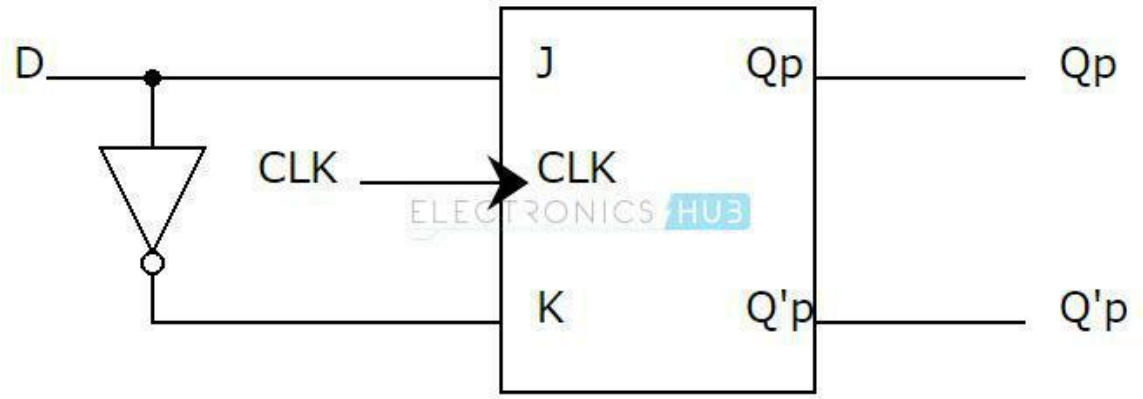
K – Map:-



The Boolean equation for K is $K = \overline{D}$.

The Boolean equations for J and K are J = D and K = D‘. The logic diagram that represents the implementation of D flip – flop from JK flip – flop is shown below.

***The logic diagram*:-**



6. JK Flip – flop to T Flip – flop

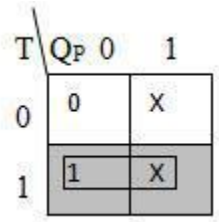
Converting the JK flip – flop to T flip flop, involves in connecting the Toggle input (T) directly to the J and K inputs. So toggle (T) will be the external input to the combinational circuit. Its output is connected to the Input of actual flip – flop (JK flip – flop).

We prepare a truth table by considering 4 possible combinations of the Toggle input (T) along with Q_P. Q_P and Q_P‘ are the present state output and its complement output of the flip – flop. Q_N is the next state output. The truth table is drawn for the T input and Q_P output to find the corresponding Q_N output. The truth table is shown below.

The conversion table:-

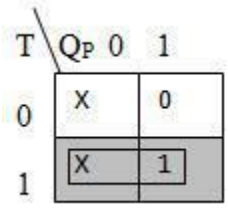
F/F INPUT	PRESENT STATE	NEXT STATE	OUTPUT	
T	**Q_n**	**Q_{n+1}**	**J**	**K**
0	0	0	0	X
1	0	1	1	X
1	1	0	X	1
0	1	1	X	0

The K – map:-



The Boolean equation for J is J = T.

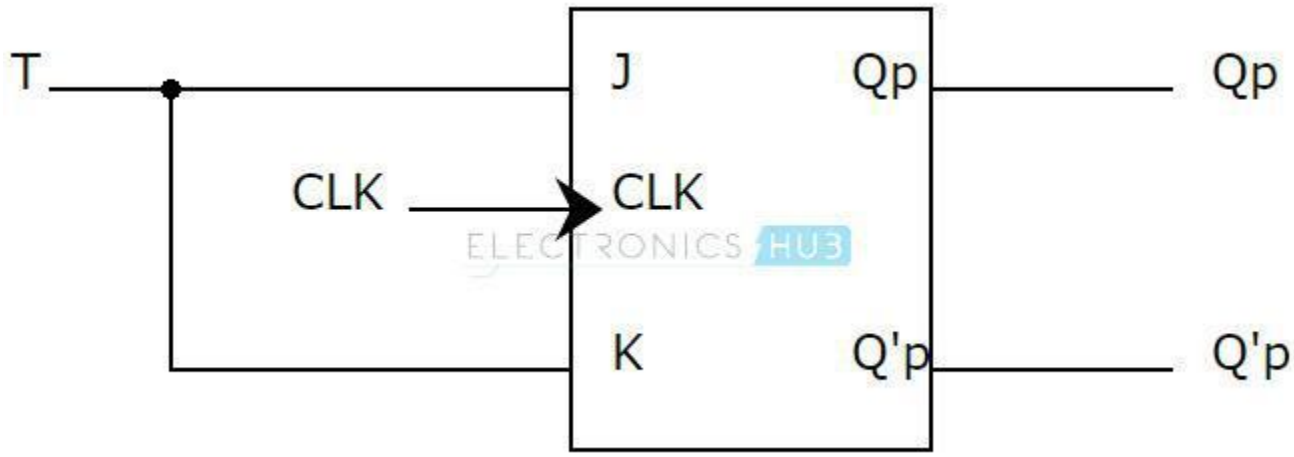
The K – map:-



The Boolean equation for K is K = T.

The logic circuit for converting JK flip – flop to T flip – flop is shown below.

***The logic diagram*:-**



D Flip – flop to other Flip – flops

I). D Flip – flop to SR Flip – flop

To convert the D flip – flop into SR flip – flop, a combinational circuit should be constructed where its inputs are S and R and its output is D. Here Data (D) is the input of actual flip – flop. The truth table is drawn with the 8 possible combinations of the two inputs S & R and Q_P. Q_P and Q_P‘ are the present state and its complement outputs of the flip – flop.

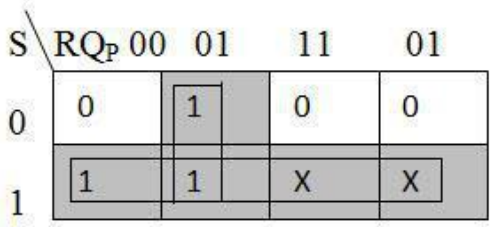
When the two inputs of SR flip – flop are high i.e. S = 1 and R = 1, then the Q_P value is invalid and hence the Data (D) inputs for the corresponding Q_P‘s are considered as _Don‘t cares‘.
The truth table for S, R and Q_P in order to get Q_N is shown below. It also consists of D inputs in order to get the same Q_N.

The Conversion Tabel:-

F/F INPUT		PRESENT STATE	NEXT STATE	OUT PUT
S	R	Q_n	Q_{n+1}	D
0	0	0	0	0
0	1	0	0	0
1	0	0	1	1
0	1	1	0	0
0	0	1	1	1
1	0	1	1	1

The K – map for solving the equation of D in terms of S, R and Q_P.

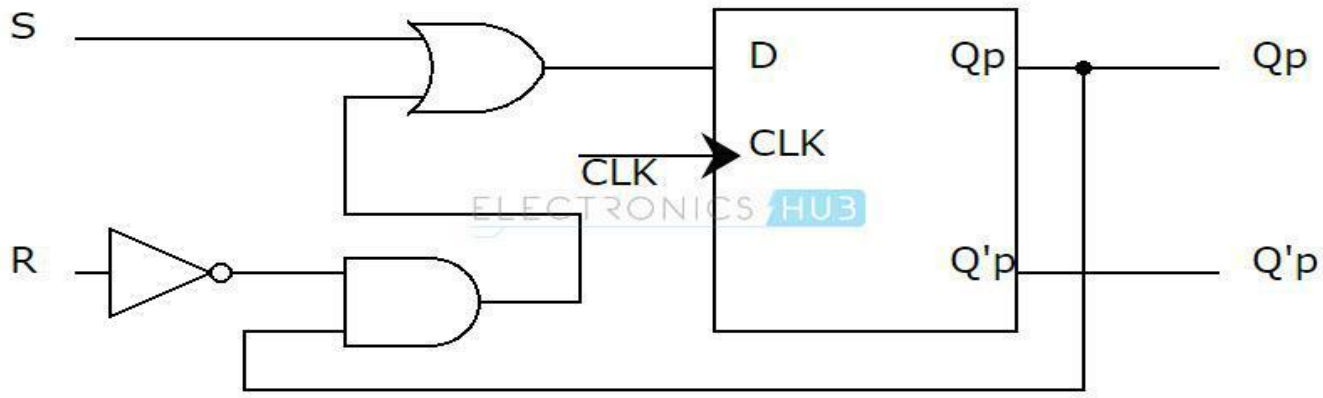
K MAP:-



The Boolean equation of D is $D = S + \overline{R}Q_P$.

The logic diagram using this equation to implement an SR flip – flop from D flip – flop is shown below.

***The logic diagram*:-**



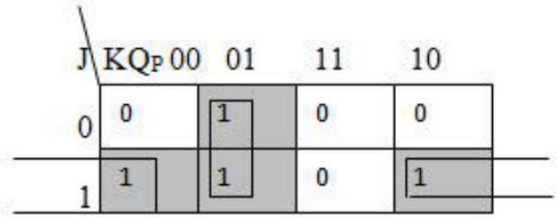
II). D Flip – flop to JK Flip – flop

When we need to convert the D flip – flop into JK flip – flop, J and K are the inputs of the combinational circuit with D as its output. Here Data (D) is the input of actual flip – flop. The truth table is drawn with the 8 possible combinations of the two inputs J & K along with Q_P. Q_P and Q_P‘ are the present state and its complement outputs of the flip – flop.The truth table consists of combinations of J, K and Q_P in order to get Q_N. Here Q_N is the next state output of the flip – flop. The truth table also consists of D inputs that lead to Q_N output.

The conversion table:-

F/F INPUT		PRESENT STATE	NEXT STATE	OUT PUT
J	K	Q_n	Q_{n+1}	D
0	0	0	0	0
0	1	0	0	0
1	0	0	1	1
1	1	0	1	1
0	1	1	0	0
1	1	1	0	0
0	0	1	1	1
1	0	1	1	1

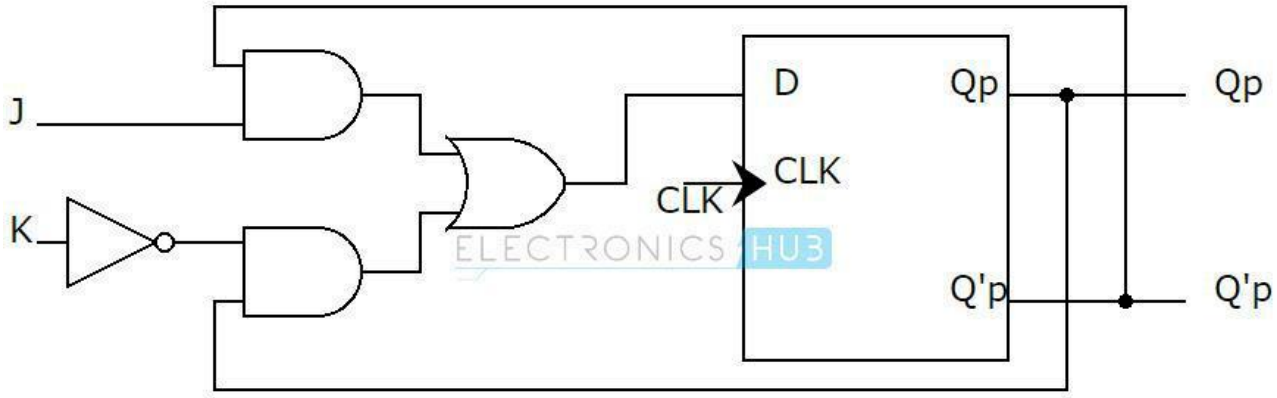
The K – map:-



$D = JQ'_P + K'Q_P$.

The Boolean equation of D deduced from the above K – map is the logical representation of implementing JK flip – flop from D flip – flop is shown below.

***The logic diagram*:-**



III). D Flip – flop to T Flip – flop

When we need to convert the D flip – flop into T flip – flop, T (Toggle input) is the input of the combinational circuit with D as its output. Here Data (D) is the input of actual flip – flop. The truth table is drawn with the 4 possible combinations of the input T along with Q_P. Q_P and Q_P' are the present state and its complement outputs of the flip – flop.

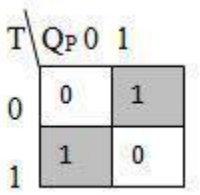
The truth table consists of combinations of T and Q_P in order to get Q_N. Here Q_N is the next state output of the flip – flop. The truth table also consists of D inputs that lead to Q_N output.

The conversion table is shown below.

The Conversion Table:-

F/F INPUT	PRESENT STATE	NEXT STATE	OUTPUT
T	Q_n	Q_{n+1}	D
0	0	0	0
1	0	1	1
1	1	0	0
0	1	1	1

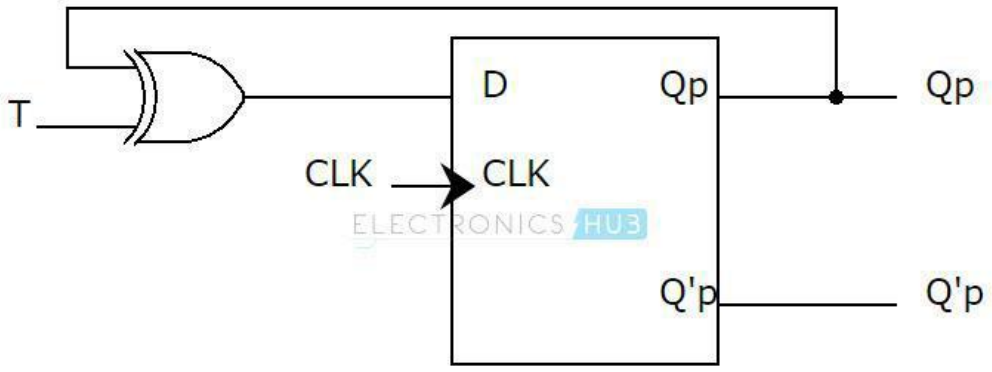
The K – map:-



$D = \overline{T}Q_P + T\overline{Q}_P$.

The Boolean equation of D in terms of T and Q_P is The logic circuit for implementing T flip – flop with D flip – flop is shown below.

The logic diagram:-



Application of Flip-flop:

1. Elimination of keyboard de-bounce.
2. As a memory element.
3. In a various types of Registers.
4. In counters/timers.
5. As a delay element.

Outcomes: Successfully implement the conversion of flip-flop.

University Asked Conversions:

1. SR Flip – flop to JK Flip – flop
2. SR Flip – flop to D Flip – flop
3. SR Flip – flop to T Flip – flop
4. JK Flip – flop to SR Flip – flop
5. JK Flip – flop to D Flip – flop
6. JK Flip – flop to T Flip – flop

Assignment No: 8

- ***Title:*** **Ripple Counter**
- **Objective:** Ripple up and down counter using IC 7476
- ***Problem statement:*** To design and implement 3 bit UP, Down, Ripple Counter using JK Flip-flop.
- ***Hardware & software requirements:***

  IC 7476 (MS-JK Flip-flop), Digital Trainer Kit, patch cords, +5V power supply.

Theory:

1) Asynchronous counter:

A digital counter is a set of flip flop. An Asynchronous counter uses T flip flop to perform a counting function. The actual hardware used is usually J-K flip-flop connected to logic 1.

In ripple counter, the first flip-flop is clocked by the external clock pulse & then each successive flip-flop is clocked by the Q or /Q‘ output the previous flip-flop. Therefore in an asynchronous counter the flip-flop are not clocked simultaneously. The input of MS-JK is connected to V_{CC} because when both inputs are one output is toggled. As MS-JK is negative edge triggered at each high to low transition the next flip-flop is triggered. On this basis the design is done for MOD-8 counter.

2) Up Counter:

Fig 1 shows 3 bit Asynchronous Up Counter. Here Flip-flop A act as a MSB Flip-flop and Flip-flop C can act as a LSB Flip-flop. Clock pulse is connected to the Clock of flip-flop C. Output of Flip-flop C (Qc) is connected to clock of next flip-flop(i.e. Flip-flop B) and so on. As soon as clock pulse changes out put is going to -change(at the negative edge of clock pulse) as a Up count sequence. For 3 bit Up counter Truth table is as shown below.

3) Down Counter:

Fig 2 shows 3 bit Asynchronous Down Counter. Here Flip-flop a act as a MSB Flip-flop and Flip-flop C can act as a LSB Flip-flop. Clock pulse is connected to the Clock of flip-flop C. Output of Flip-flop C (Qc‘) is connected to clock of next flip- flop (i.e.
Flip-flop B) and so on. As soon as clock pulse changes output is going to change (at the negative edge of clock pulse) as a down count sequence. For 3 bit down counter Truth table is as shown below.

In both the counters Inputs J and K are connected to Vcc, hence J-K Flip flop can work in toggle mode. Preset and Clear both are connected to logic 1.

- ***Truth Table:***

Up Counter

Counter States	F/F Output		
	Q_A	Q_B	Q_C
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Down Counter

Counter States	F/F Output		
	Q_A	Q_B	Q_C
7	1	1	1
6	1	1	0
5	1	0	1
4	1	0	1
3	0	1	1
2	0	1	0
1	0	0	1
0	0	0	0

- ***Logic diagram:***

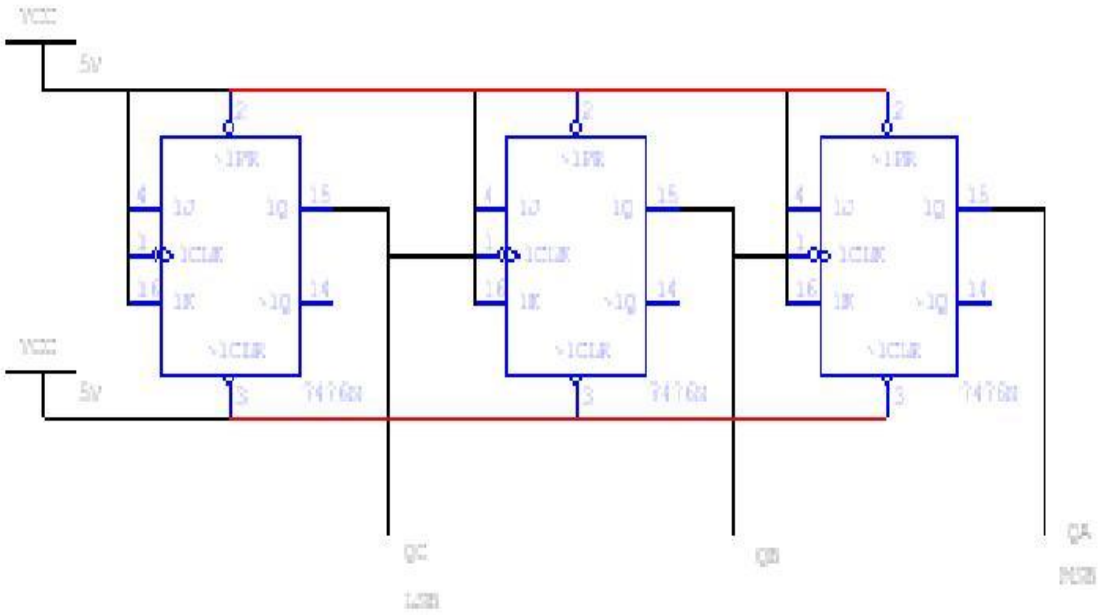


Fig 1:

3 Bit Asynchronous Up Counter

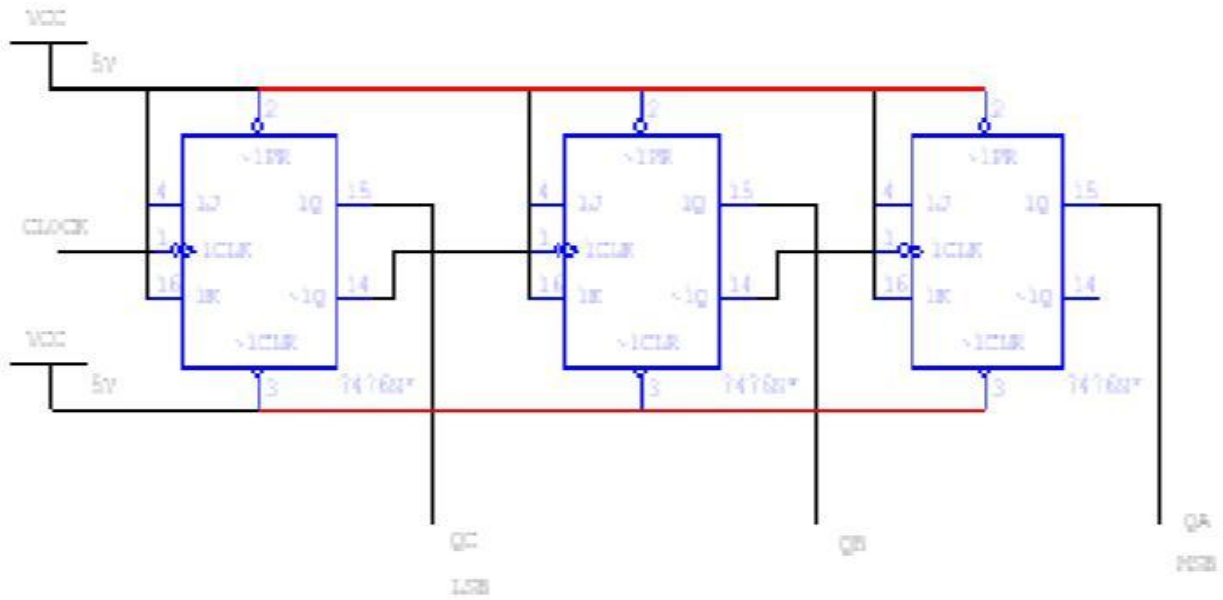
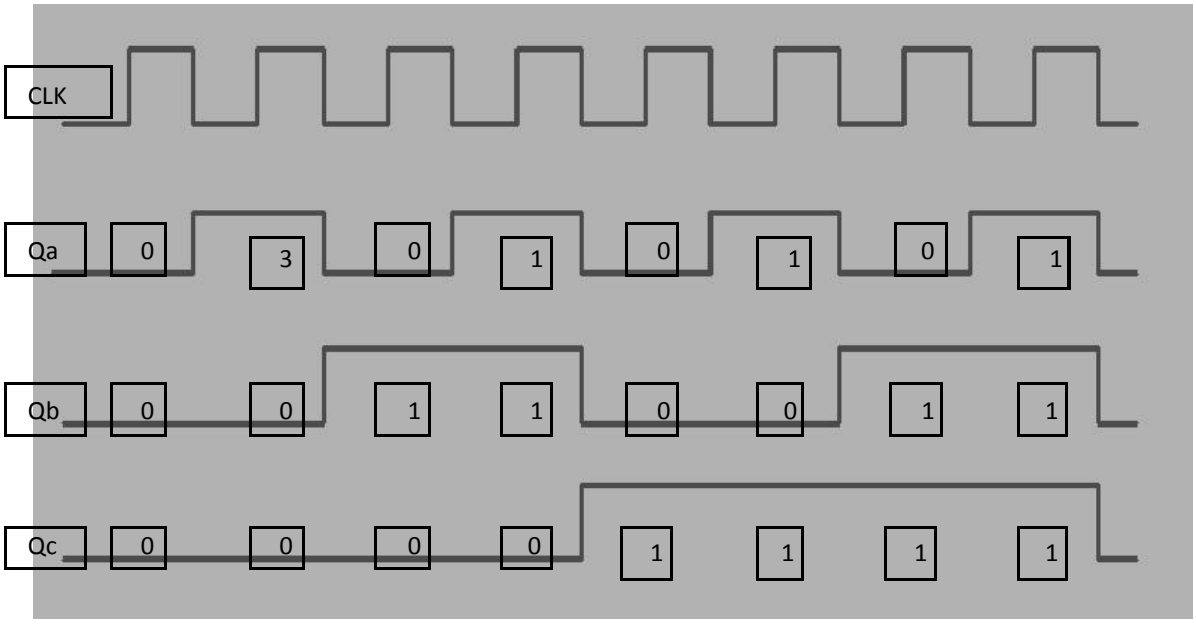


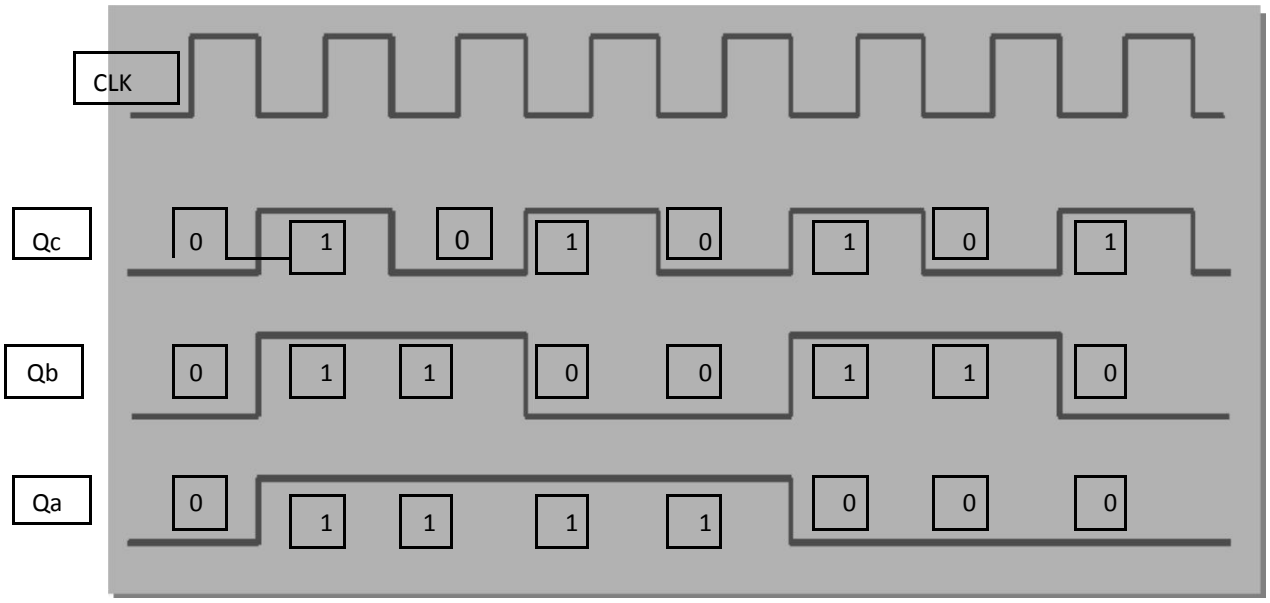
Fig 2: 3 Bit Asynchronous Down Counter

Timing Diagram:

1. 3 Bit Asynchronous Up Counter



2. 3 Bit Asynchronous Down Counter:



➢

Uses:

1) The counters are specially used as the counting devices.
2) They are also used to count number of pulses applied.
3) It also works for dividing frequency.
4) It helps in counting the number of product coming out of the machinery where product is coming out at equal interval of time.

Outcomes: Thus, we implemented up and down ripple counter. Using IC 7476

Enhancements/modifications:

As the design part is done for the 3 bit Counter, we can implement the same for 4 bit counter.

FAQ's with answers:

- **What do you mean by Counter?**

A Counter is a register capable of counting the no. of clock pulses arriving at Its clock-inputs. Count represents the no. of clock pulses arrived. A specified sequence of states appears as the counter output.

- **What are the types of Counters? Explain each.**

There are two types of counters as Asynchronous Counter and Synchronous Counter.

Asynchronous Counter: In this counter, the first flip-flop is clocked by the external clock pulse and then each successive flip-flop is clocked by the Q or Q‘ o/p of the previous flip-flop. Hence in Asynchronous Counter flip-flops are not clocked simultaneously and hence called as Ripple Counter.

Synchronous Counter: In this counter, the common clock input is connected to all the flip-flops simultaneously.

- **What are the problems involved in Ripple Counter?**

  There are two problems in Ripple Counter as

  i. Glitch

  ii. Propagation delay of flip-flop.

- **Why asynchronous counters are called as ripple counters?**

  In asynchronous counter the first flip-flop is clocked by the external clock pulse & then each successive flip-flop is clocked by the Q or /Q‘ output of the previous flip-flop i.e. clock (pulses) applied ripple from stage to stage to stage (LSB to MSB) hence asynchronous counters are called as ripple counters.

- **What do you mean by pre-settable counters?**

  A counter in which starting state is not zero can be designed by making use of the Preset inputs of the flip flops. This is referred to as loading the counter asynchronously. This is referred to as pre-settable counter.

- **What are the applications of asynchronous counters?** Digital clock

  Frequency divider circuits

- **Whether frequency division takes place in asynchronous counters?**

  Yes. In counter, the signal at the output of last flip flop (i.e. MSB) will have a frequency equal to the input clock frequency divided by the MOD number of the counter.

- **Can n- bit up asynchronous counter will act as n- bit down asynchronous counter without changing the position of the clock?**

  Yes. Instead of taking output of a counter from uncomplimentary output (Q), if we take it from complimentary output (Q bar), the same counter circuit will work as down counter.

Assignments Questions:

Assignment No: 09(a)

- ***Title:*** **Synchronous counter**
- ***Objective:*** 3 Bit up/down synchronous Counter
- ***Problem Statement:*** To design and implement 3 bit UP and Down, Controlled UP/Down Synchronous Counter using MS-JK Flip-flop.
- **Hardware & Software Requirement's :**

  Digital Trainer Kit, IC 7476, IC 7408, IC 7432 & IC 7404.patch cords, +5V power supply.

Theory:

Counters: counters are logical device or registers capable of counting the no of states or no of clock pulse arriving at its clock input where clock is a timing parameter arriving at regular intervals of time, so counters can be also used to measure time & frequencies. They are made up of flip flops. Where the pulse are counted to be made of it goes up step by step & the o/p of counter in the flip flop is decoded to read the count to its starting step after counting n pulse incase of module & counters.

➢ **Synchronous Counter:**

In this counter, all the flip flops receive the external clock pulse simultaneously.

Ex:- Ring counter & Johnson counter

The gates propagation delay at reset time will not be present or we may say will not occur.

➢ **Classification of synchronous counter:**

Depending on the way in which counting processes, the synchronous counter is classified is :-

1) Up counter.
2) Down counter.
3) Up down counter.

- **Up Counter:**

The up counter counts binary form 0 to7 i.e.(000 to 111).It counts from small to large number. It‘s O/P goes on increasing as they receive clock pulse

- **Down Counter:**

This down counter counts binary from 7-0 i.e.(111-000).It counts from large to small number. It‘s O/P goes on increasing as they receive clock pulse

- **Excitation Table**:- The tabular representation of the operation of flip flop (i.e: Operational Characteristic)

Present State	Next State	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

For M = 0, it acts as an Up counter and for M =1 as an Down counter.

State Table for 3 bit Up-Down Synchronous Counter:

Control input M	Present State			Next State			Input for Flip-flop					
	Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	0	1	0	X	0	X	1	X
0	0	0	1	0	1	0	0	X	1	X	X	1
0	0	1	0	0	1	1	0	X	X	0	1	X
0	0	1	1	1	0	0	1	X	X	1	X	1
0	1	0	0	1	0	1	X	0	0	X	1	X
0	1	0	1	1	1	0	X	0	1	X	X	1
0	1	1	0	1	1	1	X	0	X	0	1	X
0	1	1	1	0	0	0	X	1	X	1	X	1
1	0	0	0	1	1	1	1	X	1	X	X	1
1	0	0	1	0	0	0	0	X	0	X	1	X
1	0	1	0	0	0	1	0	X	X	1	X	1
1	0	1	1	0	1	0	0	X	X	0	1	X
1	1	0	0	0	1	1	X	1	1	X	X	1
1	1	0	1	1	0	0	X	0	0	X	1	X
1	1	1	0	1	0	1	X	0	X	1	X	1
1	1	1	1	1	1	0	X	0	X	0	1	X

K- map Simplification:

MQC \ QBQA	00	01	11	10
00	1	X	X	1
01	1	X	X	1
11	1	X	X	1
10	1	X	X	1

JA = 1

MQC \ QBQA	00	01	11	10
00	X	1	1	X
01	X	1	1	X
11	X	1	1	X
10	X	1	1	X

KA = 1

MQC \ QBQA	00	01	11	10
00	0	1	X	X
01	0	1	X	X
11	1	0	X	X
10	1	0	X	X

JB = M $\overline{QA}$ + $\overline{M}$ QA

MQC \ QBQA	00	01	11	10
00	X	X	1	0
01	X	X	1	0
11	X	X	0	1
10	X	X	0	1

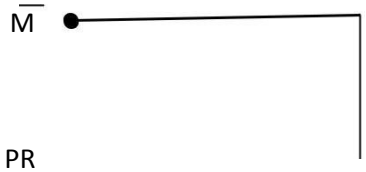
KB = M $\overline{QA}$ + $\overline{M}$ QA

MQC \ QBQA	00	01	11	10
00	0	0	1	0
01	X	X	X	X
11	X	X	X	X
10	1	X	0	0

JC = M $\overline{QA\ QB}$ + $\overline{M}$ QA QB

MQC \ QBQA	00	01	11	10
00	X	X	X	X
01	0	0	1	0
11	1	0	0	0
10	X	X	X	X

KC = $\overline{M}$ QA QB + M $\overline{QA\ QB}$



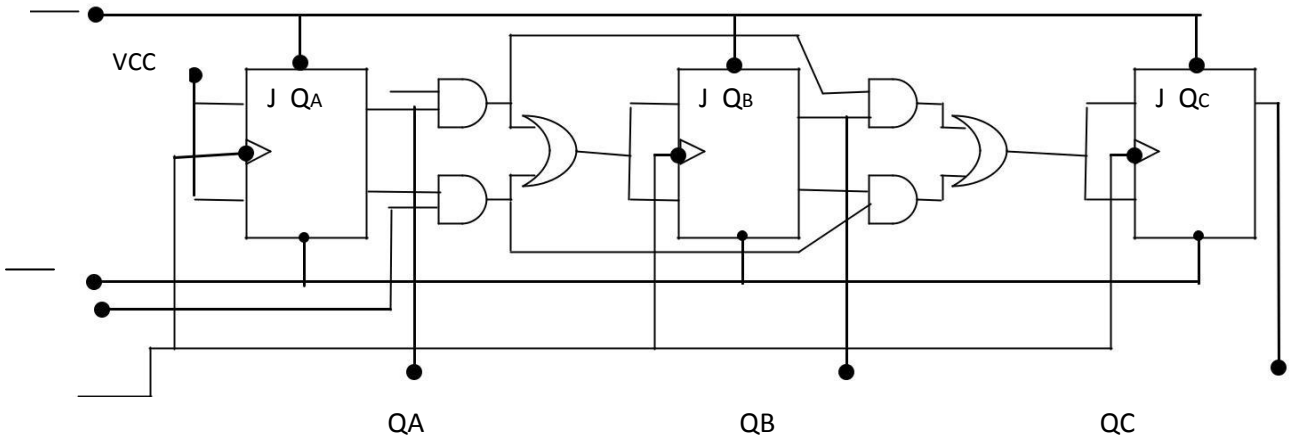


Fig : Logical Diagram of Up down counter using JK Flip Flop

Uses:

2) The synchronous counter is specially used as the counting devices.
3) They are also used as counter to count the no of clock pulses applied.
4) It also works for counting frequency & is used in frequency divider circuit.
5) It is used in digital voltmeter.
6) It is also used in counter type A to D converter.
7) It is also used for time measurement.
8) It is also used in digital triangular wave generator.
9) It helps in counting the no of product coming out from machinery where product is coming out at equal interval of time.

Outcome:

Up and down counters are successfully implemented, the counters are studied & o/p are checked. The truth table is verified.

Enhancements/modifications:

As the design part is done for the 3 bit Counter, we can implement the same for 4 bit counter.

Assignments Questions:

FAQ's with answers:

- **What do you mean by Counter?**

  A Counter is a register capable of counting the no. of clock pulses arriving at its

clock inputs. Count represents the no. of clock pulses arrived. A specified sequence of states appears as the counter output.

- **What are the types of Counters? Explain each.**

There are two types of counters as Asynchronous Counter and Synchronous Counter. *Asynchronous Counter*: In this counter, the first flip-flop is clocked by the external clock pulse and then each successive flip-flop is clocked by the Q or Q‘ o/p of the previous flip-flop. Hence in Asynchronous Counter flip-flops are not clocked simultaneously and hence called as Ripple Counter.

Synchronous Counter: In this counter, the common clock input is connected to all the flip-flops simultaneously.

- **What do you mean by pre-settable counters?**

A counter in which starting state is not zero can be designed by making use of the preset inputs of the flip flops. This is referred to as loading the counter asynchronously. This is referred to as pre-settable counter.

- **What are the applications of synchronous counters?** Digital clock

  Frequency divider circuits

  Frequency counters

  Used in analog to digital converters

- **What are the advantages of synchronous counters over asynchronous counters?**

  Propagation delay time is reduced

  Can operate at a much higher frequency than the asynchronous counters.

- **Ring counter is an example of synchronous counters or asynchronous counter?** Synchronous counter. Since all the flip flops are clocked simultaneously.

- **Twisted Ring (Johnson’s) counter is an example of synchronous counters or asynchronous counter?**

  Synchronous counter. Since all the flip flops are clocked simultaneously.

- **What is the difference between ring counter and twisted ring counter?**

  In ring counter pulses to be counted are applied to a counter , it goes from state to state and the output of the flip flop s in the counter is decoded to read the count. Here the

uncomplimentary output (Q) of last flip flop is fed back as an input to first flip flop. Ring counters are referred as MOD _N‘ counters.

But in Twisted ring counter the complimentary output (Q bar) of last flip flop is fed back as an input to first flip flop. Twisted Ring counters are referred as MOD _2N‘ counters.

- **What are the applications of ring counters?**

  Ring counter outputs are sequential non-overlapping pulses which are useful for control state counters,

  Used in stepper motor, which requires pulses to rotate it from one position to the next.

  Used as divide by _N‘ ((MOD _N‘) counters.

- **What are the applications of ring counter twisted ring counters?**

  Used as divide by _2N‘ ((MOD _2N‘) counters.

  Used for control state counters.

  Used for generation of multiphase clock.

- **List the Synchronous Counter ICs.**

  IC 74160 : Decade Up Counter

  IC 74161 : 4 bit binary Up Counter

  IC 74162 : Decade Up Counter

  IC 74163 : 4 bit binary Up Counter

  IC 74168 : Decade Up/Down Counter

  IC 74169 : 4 bit Binary Up/Down Counter

  IC 74190 : Decade Up/Down Counter

  IC 74191 : 4 bit Binary Up/Down Counter

  IC 74192 : Decade Up/Down Counter

  IC 74193 : 4 bit Binary Up/Down Counter

Assignment: 09(b)

- ***Title:*** **Ripple Counter**
- ***Objective:*** Modulo N counter using 7490(N>10).
- ***Problem Statement***: Realization of mod N counter using IC 7490.
- ***Hardware & software requirements:***

Digital Trainer Kit, 7490 (Decade Counter IC), Patch Cord, + 5V Power Supply

Theory:

Ripple counter IC-7490 (decade counter): i.e .having various name like mod-n counter, decade counter ,BCD counter.

IC-7490 is a TTL MSI decade counter. It contains four master slave flip flops and additional gating to provide a divide-by-two counter and a three stage binary counter which provides a divide by 5 counter.

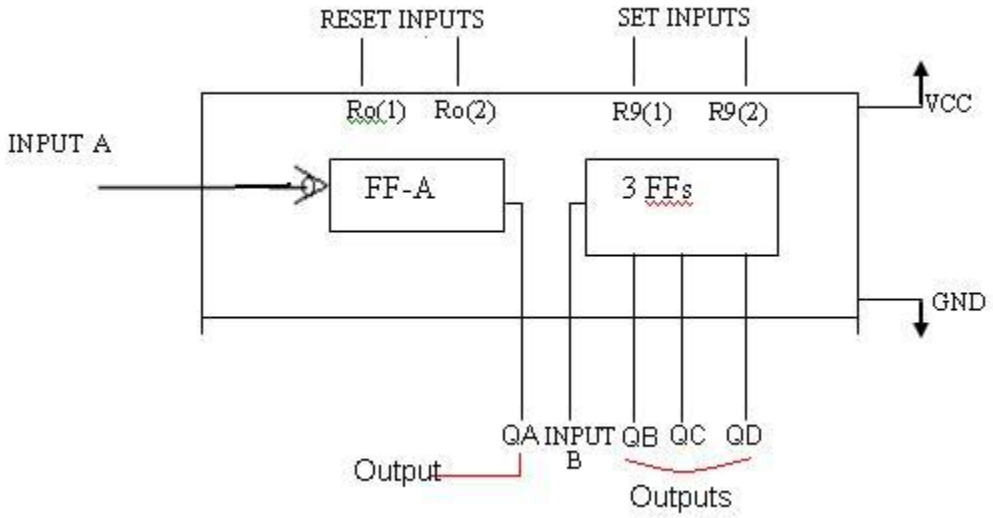


Fig.1 The Basic internal structure of IC 7490

Conclusion:

1. If both the reset input $R_{O(1)}$ & $R_{O(2)}$ are at logic 1 then all the flip-flop will be reset and the output is given by

$$Q_D\ Q_C\ Q_B\ Q_A = 0000$$

2. If both the reset input $R_{9(1)}$ & $R_{9(2)}$ are at logic 1 then the counter output is set to decimal 9.

$$Q_D\ Q_C\ Q_B\ Q_A = 1001$$

3. If any one pin of $Ro_{(1)}$ & $Ro_{(2)}$ and one of $R_{9(1)}$ & $R_{9(2)}$ are at low, then the counter will be in counting mode.

The reset/count function table of IC7490 is shown in table 1.

Table 1:-Reset/count truth table

Reset inputs				**Output**			
R0(1)	**R0(2)**	**R9(0)**	**R9(1)**	**QD**	**QC**	**QB**	**QA**
1	1	0	X	0	0	0	0
1	1	X	0	0	0	0	0
X	X	1	1	1	0	0	1
X	0	X	0	COUNTER			
0	X	0	X	COUNTER			
0	X	X	0	COUNTER			
X	0	0	x	COUNTER			

IC 7490 is MOD-10 or decade counter. It is a 14 pin IC with the pin configuration as shown in fig:

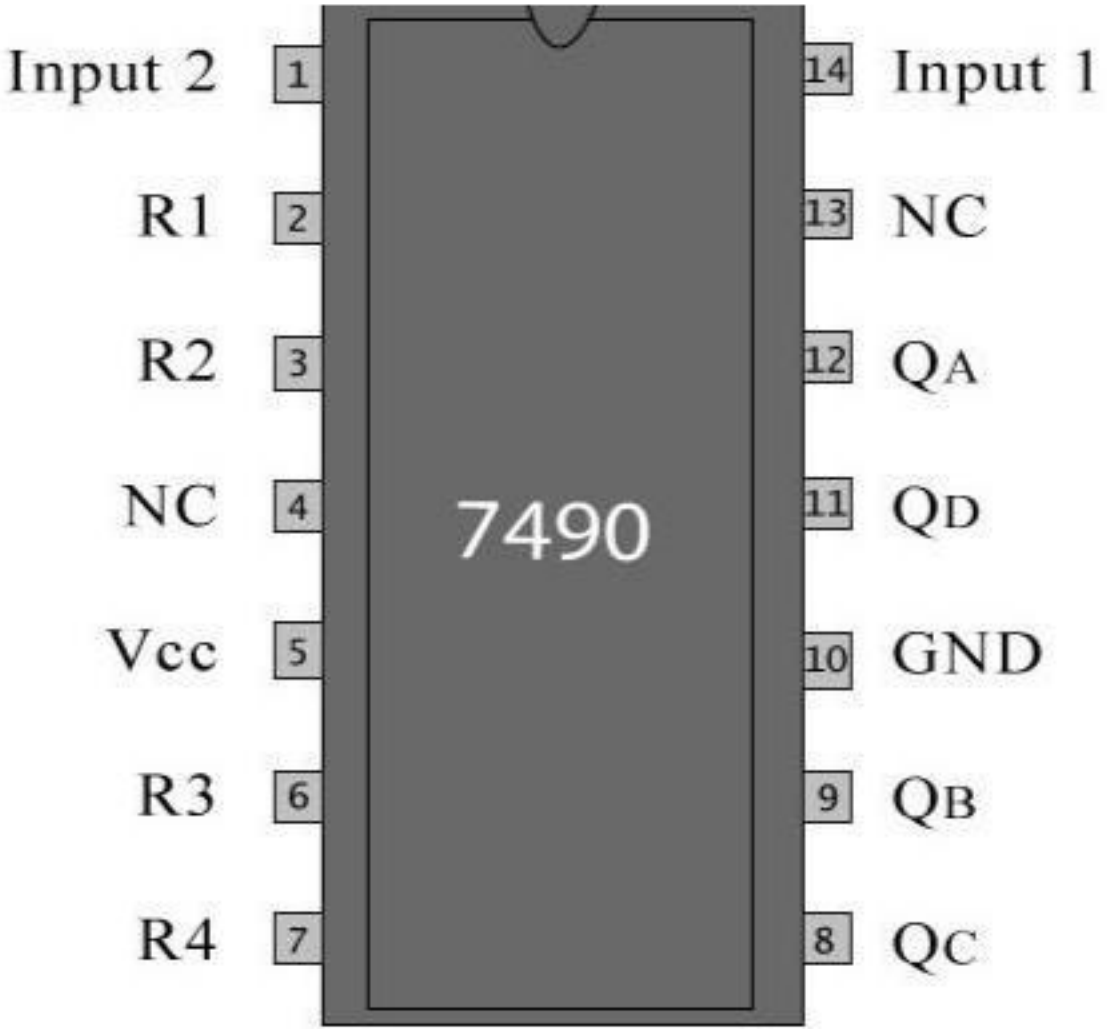


Fig. 2 Pin configuration of IC7490.

Table 2:-Pin name and description of IC 7490

Pin name	Description
Input B	This is clock input to the internal MOD-5 ripple counter, which is negative edge triggered.
R0(1),R0(2)	Gated zero reset inputs
R9(1),R9(2)	These are gated set to nine inputs
QD,QC,QB	Output of internal MOD-5 counter with QD as MSB.
QA	Output of internal MOD-2 counter with QA as LSB.
Input A	Clock input to FF-A which is negative edge triggered.

Decade Counter Operation :

1. The output of MOD-2 is externally connected to the input B which is the clock input of the internal MOD-5 counter.

2. Hence Q_A toggles on every falling edge of clock input whereas the output Q_D,Q_C,Q_B of the MOD-5 counter will increment from 000 to 100 on low going change of Q_A output.

3. Due to cascading of MOD-2 and MOD-5 counter, the overall configuration becomes a MOD-10 i.e. decade counter.

4. The reset inputs Ro(1), Ro(2) and preset inputs R9(1), R9(2) are connected to ground so as to make them inactive.

Table: Summarizes the opération of the 7490 as décade counter

O/p of MOD-5			O/p of MOD-2	CLK	Count
QD	QC	QB	QA		
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	2	2
0	0	1	1	3	3
0	1	0	0	4	4
0	1	0	1	5	5
0	1	1	0	6	6
0	1	1	1	7	7
1	0	0	0	8	8
1	0	0	1	9	9

Implémentations:

Realization of MOD 10 counter using IC 7490

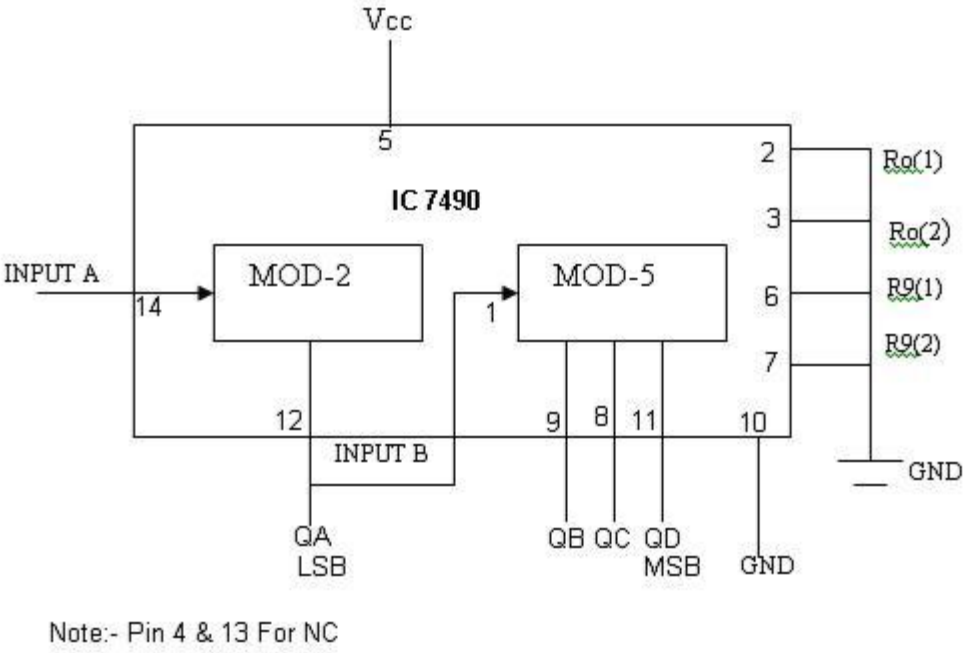
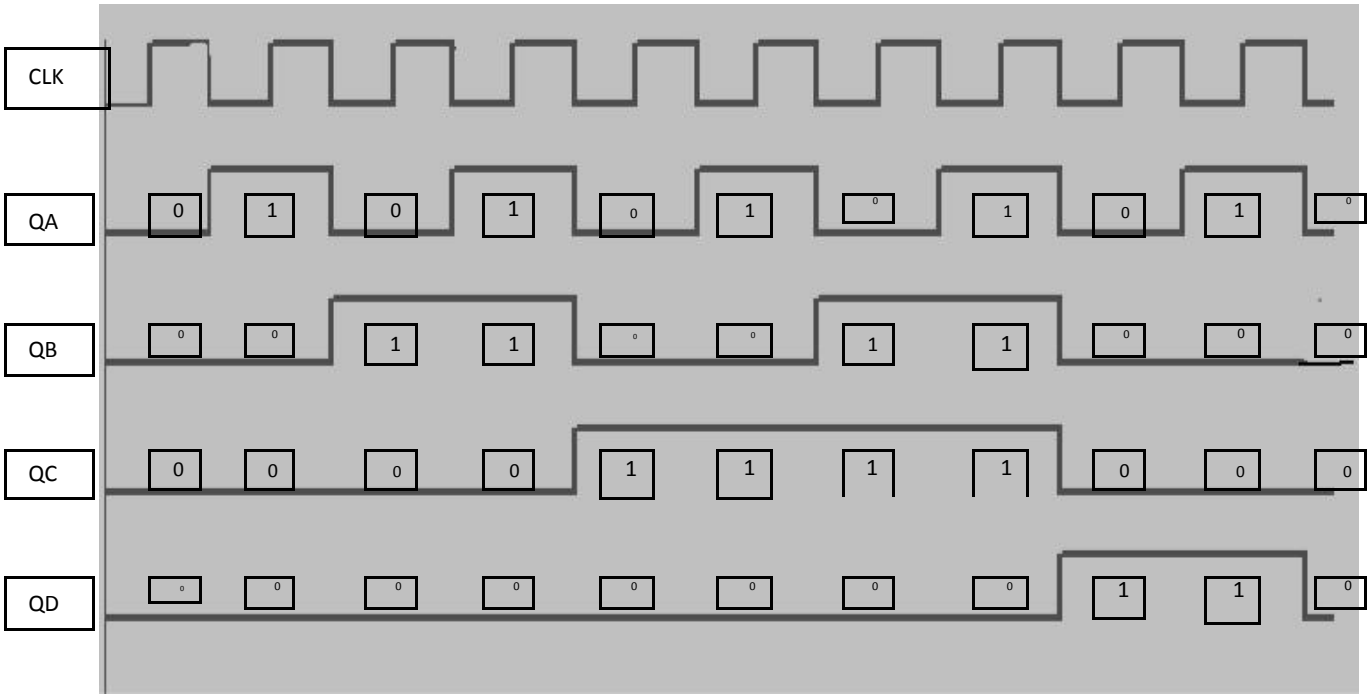


Fig 3.MOD-10 counter using IC 7490

***Timing diagram of mod10*:**



Application Of IC 7490:

1. Symmetrical Bi- quinary divide by ten counter.

Outcomes:

Thus, we implemented divide by two (MOD-2) and divide by 5 (MOD-5) counter. Using IC7490.

Assignments Questions:

1. Design using counter using 7490.
   1) MOD-27
   2) MOD-17
   3) MOD-24
   4) MOD-20
   5) MOD-7
   6) MOD- 98
   7) MOD -97
   8) MOD -45
   9) MOD-56

Assignment: 10

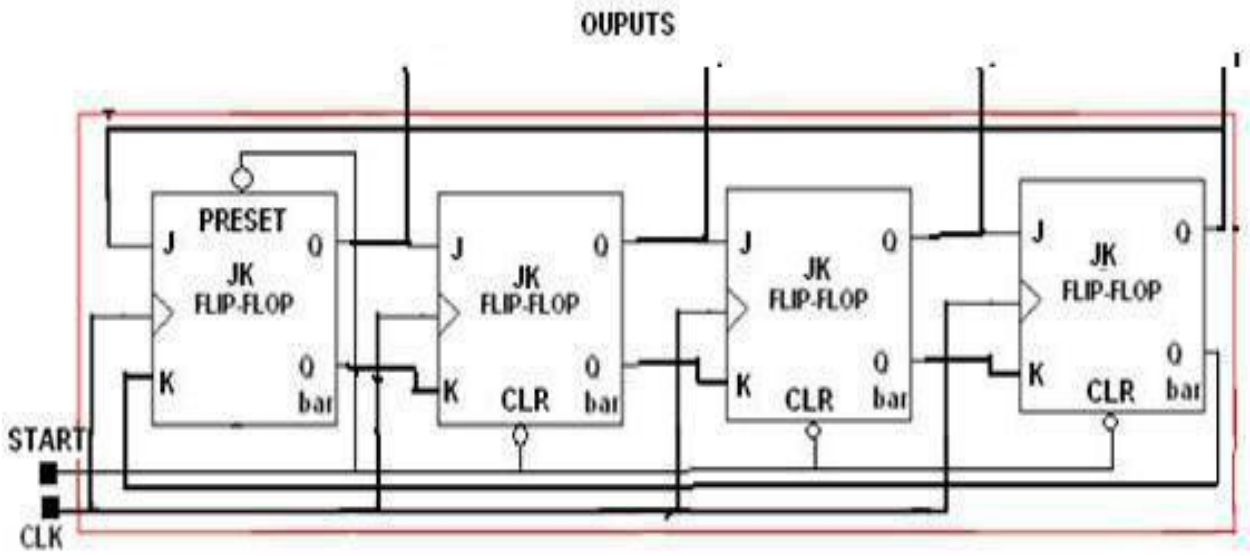
- ***Title:*** **Ring Counter**
- ***Objective:*** To learn register work as a counter
- ***Problem Statement***: Design and realization of ring counter & Twisted pair ring counter (Johnson counter)
- ***Hardware & software requirements:*** Digital trainer kit, patch chords,+5v power supply, IC 7474, IC 7476

Theory:

1) Ring Counter:

Ring counter is a typical application of shift register. The connections reveal that they are similar to the connections for shift right operation, except for one change. Ring counter is a special type of shift register.

Operation:



Initially a low clear (CLR) pulse is applied to all flip-flops. Hence FF-3, FF-2, FF-1 will be reset but FF-0 will be set. So outputs are:

Q3 Q2 Q1 Q0 = 0001

The clear terminal is made inactive by applying a high level to it. The clock signal is then applied to all the flip-flops simultaneously. Note that all the flip-flops are negative edge triggered.

On first negative going CLK edge:

As soon as first falling edge of clock hits, only FF-1 will be set Q0 = J1 = 1. The FF-0 will reset because J0=Q3=0 and there is no change in the status of FF-2 and FF-3. Hence after the first clock pulse the outputs are:

Q3 Q2 Q1 Q0 = 0010

On the second falling edge of clock:

At the second falling edge of clock, only FF-2 will be set as J2 = Q1 = 1.The FF-1 will reset since J1 = Q0 = 0.There is no change in status of FF-3 and FF-0.So after second clock pulse the outputs are,

Q3 Q2 Q1 Q0 = 0100.

Similarly after third clock pulse outputs are:

Q3 Q2 Q1 Q0 = 1000

After fourth pulse outputs are:

Q3 Q2 Q1 Q0 = 0001

These are the outputs from where we started.

Number Of Output States: The number of output states will be equal to number of flip-flops. So for a 4 bit counter the number of states is equal to 4.

A_3	A_2	A_1	A_0
0	0	0	1
0	0	1	0
0	1	0	0

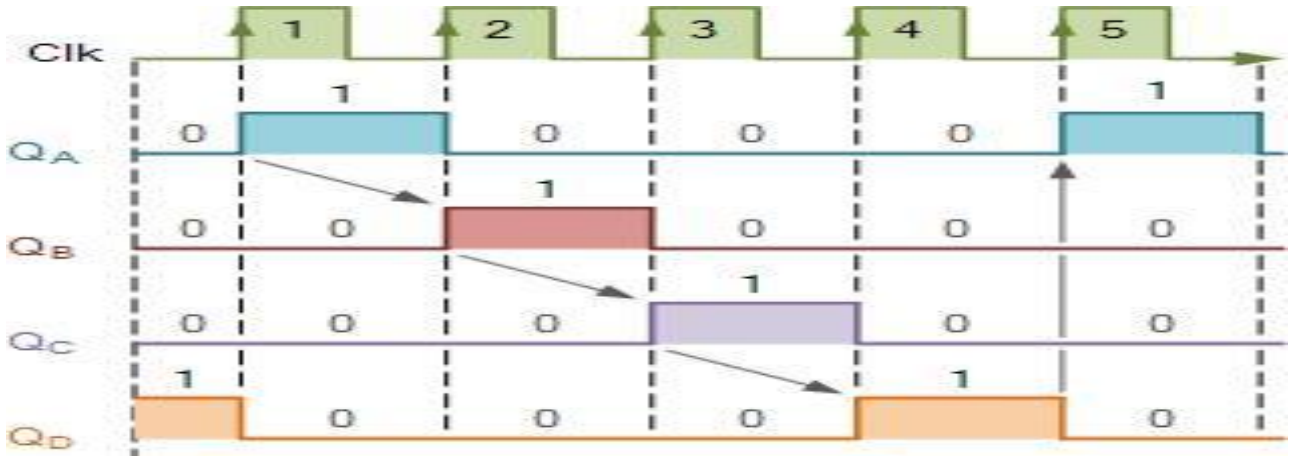
1	0	0	0

Applications:

Ring counter are used in those applications in which several operations are to be controlled in sequential manner. In welding operations such as squeeze, hold, weld, etc.

Waveforms For The Ring Counter:

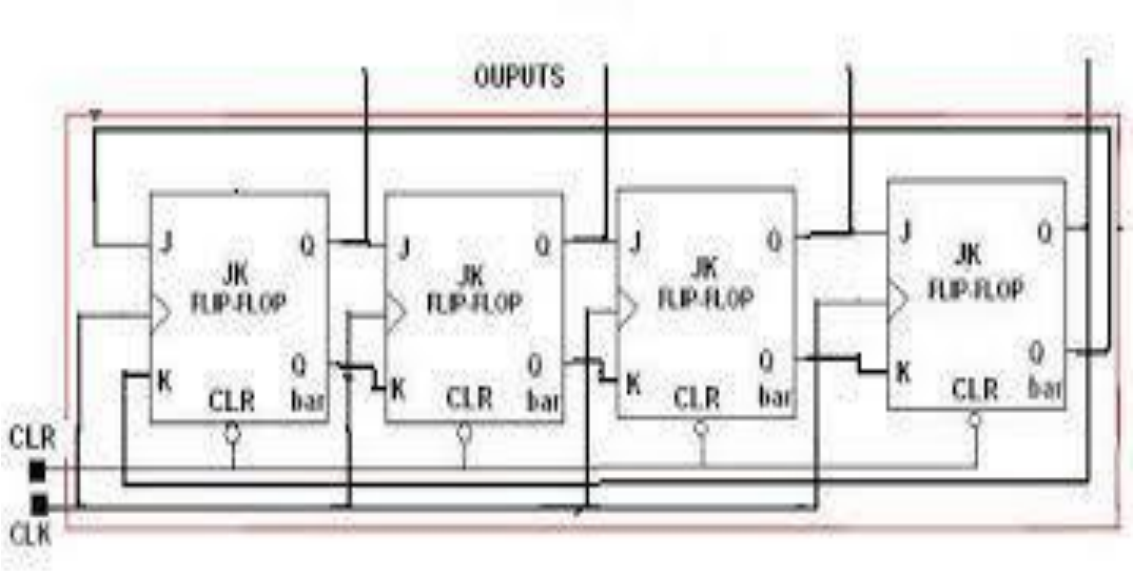
The waveforms for the ring counter as shown in fig.



2) Johnson Counter(Twisted Pair Ring Counter):-

- In the ring counter the outputs of FF-3 were connected directly to inputs of FF-0 i.e. Q3 to J0 Q3 $\overline{to}$ k0.

- Instead if the outputs are cross coupled to the inputs i.e. if Q3 is connected to k0 & Q3 is $\overline{con}$nected to J0 then the circuit is called as twisted ring counter or Johnson's counter.



- All the Flip-flops are negative edge, triggered, and clock pulses are applied to all of them and simultaneously.
- The clear inputs of all the flip-flops are connected together and connected to an external clear signal. Note that all there clear inputs are active low inputs.

Operations:

- Initially a short negative going pulses is applied to the clear input of all the flip-flop . This will reset all the flip-flops. Hence initially the outputs are

Q3, Q2, Q1, Q0=0000

- But $\overline{Q3}$ =1 and since it is coupled to J0 it is also equal to 1.Hence, J0=1 and k0 =0 …… initially.

- **On the first falling edge of clock pulse :**

1) As soon as the first negative edge of clock arrives, FF-0 will be set Hence Q0 will become 1.
2) But there is no change in the status of any other flip-flop.
3) Hence after the first negative going edge of clock the flip –flop outputs are

Q3 Q2 Q1 Q0 = 0001

- **On the second negative going clock edge:**

  1) Before the second negative going clock edge, Q3=0 & $\overline{Q3}$ =1, Hence J0=1and K0 =0. Also Q0=1.Hence J1=1.
  2) Hence the second falling clock edge arrives, FF-1 continues to be in the set mode and FF-1 will now set. Hence Q1 will become 1 & Q1=0.
  3) There is no change in the status of any other FF.
  4) Hence the second clock edge the outputs are

Q3 Q2 Q1 Q0=0011

- Similarly after the third clock pulses , the outputs are

Q3 Q2 Q1 Q0=0111

- And after the fourth clock pulses ,the outputs are

Q3 Q2 Q1 Q0=1111

- Hence as soon as the fifth negative going clock pulses strikes FF-0 will reset. But the outputs of the other flip-flops will remain unchanged ,So after the fifth clock ,the outputs are,

Q3 Q2 Q1 Q0=1110

- This operation will continue till we reach to all zero output state

Q3 Q2 Q1 Q0=0000

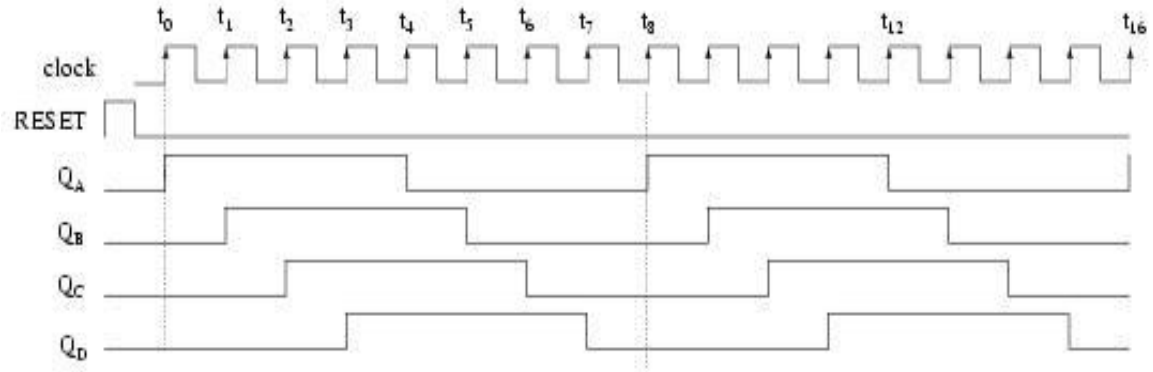
Truth Table:- There are 8 distinct states of outputs

In general we can say that the number of states of a Johnson‘s counter is twice the number of FF used therefore for a 4-FF Johnson‘s counter; there are 8-distinct output states.

CLEAR	CLK	Q3	Q2	Q1	Q0	State no.	Decimal
	Initially	0	0	0	0	1	0
1	↓	0	0	0	1	2	1
1	↓	0	0	1	1	3	3
1	↓	0	1	1	1	4	7

1	↓	1	1	1	1	5	15
1	↓	1	1	1	0	6	14
1	↓	1	1	0	0	7	12
1	↓	1	0	0	0	8	8
1	↓	0	0	0	0	1	0

Wave forms for Johnsons counter:



Four stage Johnson counter waveforms

Outcomes:

Successfully implemented register as a counter

Assignments Questions:

Assignment N0. – 11

AIM : To design and implement sequence generator with and without bushing using

IC 7476.

OBJECTIVE : To understand sequence generator, one of the sequential circuit.

IC's USED : IC 7476(Dual JK), 7408 (AND-gate), 7432 (OR-gate).

THEORY :

A sequential circuit which generates a prescribed sequence of bits in synchronism with a clock is referred to as a sequence generator. These pulse trains or sequence of bits can be used to open valves, close gates, turn on lights, and turn off machines and other variety of jobs.

For the design of sequence generator, we first determine the required no. of flip flops and the logic circuit for the next state decoder.

No. of flip flops required to generate particular sequence can be determined as follows.

1) Find the no. of 1's in the sequence.
2) Find the no. of 0's in the sequence.
3) Take the maximum out of two.
4) If N is the required no. of flip flops, choose minimum value of _n' to satisfy equation given below.

$$\text{Max } (0\text{'s} , 1\text{'s}) \leq 2^{n-1}$$

The sequence generator can be classified as

1) sequence generator without bushing
2) sequence generator with bushing

The aim in this experiment is to design a sequence generator to generate a sequence of bit i.e. 10101.

For finding the no. of flip flops we use the formula

$$m \leq 2^{n-1}$$

where m= maximum count and

n= no. of flip flops required.

For the given sequence no. of 1‘s=3 no. of 0‘s=2, so minimum value of n which satisfies above relation is 3. Once the no. of flip flops are decided we have to assign unique states corresponding to each bit in the given sequence such that flip flop representing least significant bit generates the given sequence (usually the o/p of flip flop representing the least significant bit is used to generate the given sequence)

Design -

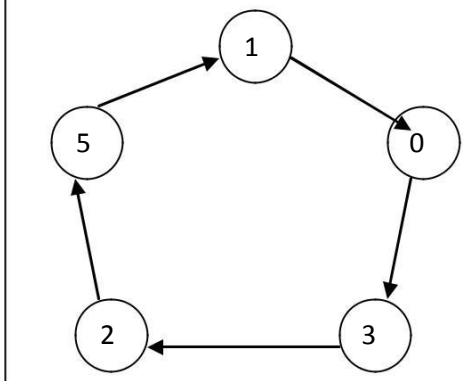
For the sequence of bits 10101 we require three flip flops as calculated above. The State Diagram, state assignment for this problem is shown below. Where we will use the o/p FF 0 i.e. o/p of first flip flop Q0 as a sequence of bits & assign unique states corresponding to each bit in the sequence as shown in state assignment table.

1. **Sequence Generator Without**

Bushing: State assignment table :

Q2	Q1	Q0	STATES
0	0	1	1
0	0	0	0
0	1	1	3
0	1	0	2
1	0	1	5

State Diagram :-



State table :

Present states			**Next states**			**Flip flop input**					
Q2	Q1	Q0	Q2	Q1	Q0	J2	K2	J1	K1	J0	K0
0	0	0	0	1	1	0	X	1	X	1	X
0	0	1	0	0	0	0	X	0	X	X	1
0	1	0	1	0	1	1	X	X	1	1	X
0	1	1	0	1	0	0	X	X	0	X	1
1	0	0	X	X	X	X	X	X	X	X	X
1	0	1	0	0	1	X	1	0	X	X	0
1	1	0	X	X	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X	X	X

K-Map simplification:

1) For J0 –

0 0 0 1 1 1 1 0

1	X	X	1
X	X	X	X

J0 = 1

2) For K0 –

0 0	0 1	1 1	1 0
X	1	1	X
X	0	X	X

K0 = $\underline{Q2}$

3) For J1 –

0 0	0 1	1 1	1 0
1	0	X	X
X	0	X	X

J1 = $\underline{Q0}$

3) For K1 –

0 0	0 1	1 1	1 0
X	X	0	1
X	X	X	X

K1 = $\underline{Q0}$

3) For J2–

0 0	0 1	1 1	1 0
0	0	0	1
X	X	X	X

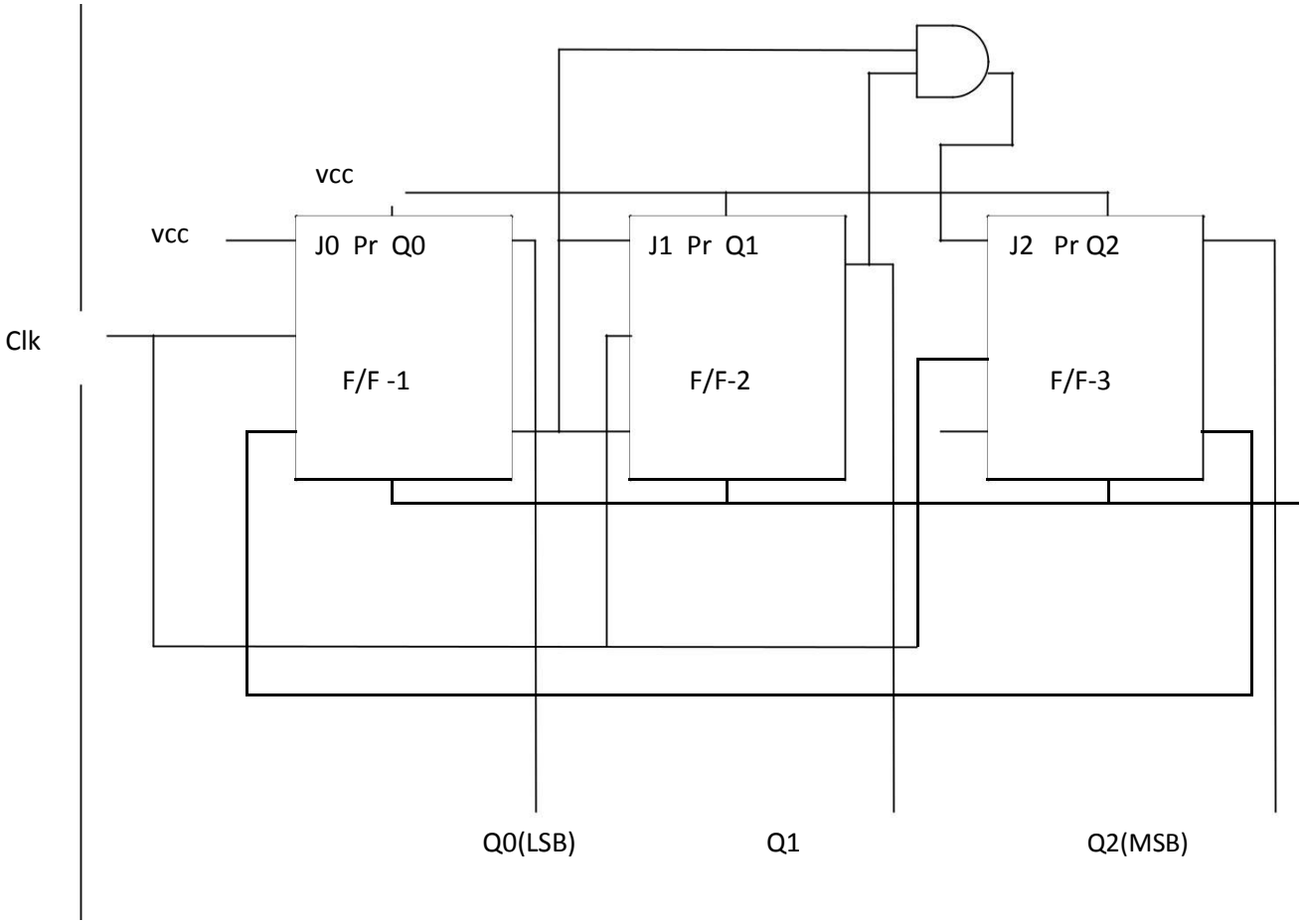
$J2 = Q1\overline{Q0}$

3) For J1 –

0 0	0 1	1 1	1 0
X	X	X	X
X	1	X	X

K2 = 1

Logic Diagram-



Hardware requirements:

GATE	ICs	Quantity
JK F/F	7476	2
AND	7408	1

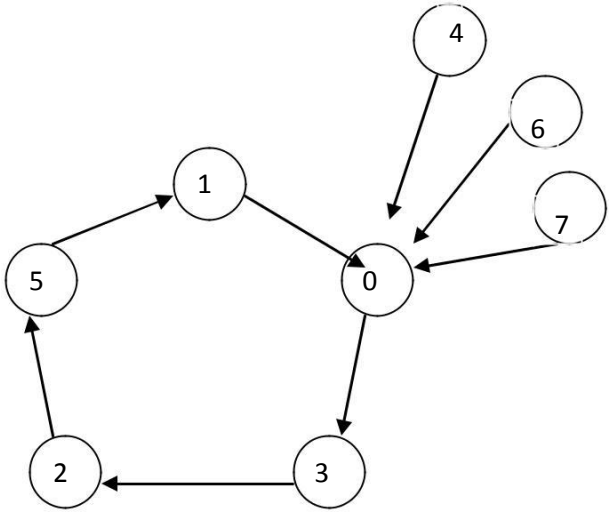
1. **Sequence Generator with Bushing :-**

The state assignment for this is shown below as we have seen during designing of sequence generator without using bushing we will have the output _C' i.e. the output of first Flip-Flop is a sequence generator of bits and assign unique states corresponding to each bit in the sequence out with bushing means we have to use unassigned states also that is 4,6,7.

State – Assignment Table :-

Q2	Q1	Q0	STATES
0	0	1	1
0	0	0	0
0	1	1	3
0	1	0	2
1	0	1	5

State Diagram-



State Table :-

Present state			**Next state**			**Flip flop inputs**					
Q2	Q1	Q0	Q2	Q1	Q0	J0	K0	J1	K1	J2	K2
0	0	0	0	1	1	1	X	1	X	0	X
0	0	1	0	0	0	X	1	0	X	0	X
0	1	0	1	0	1	1	X	X	1	1	X
0	1	1	0	1	0	X	1	X	0	0	X
1	0	0	0	0	0	0	X	0	X	X	1
1	0	1	0	0	1	X	0	0	X	X	1
1	1	0	0	0	0	0	X	X	1	X	1
1	1	1	0	0	0	X	1	X	1	X	1

K Map Simplification :

For J0 –

	0 0	0 1	1 1	1 0
1	X	X	1	
0	X	X	0	

J0 = Q2

3) For K0 –

0	0 0	0 1	1 1	1 0
	X	1	1	X
	X	0	1	X

K0 = Q2 + Q1

3) For J1 –

0 0	0 1	1 1	1 0
1	0	X	X
0	0	X	X

J1 = Q2Q0

3) For K1 –

0 0	0 1	1 1	1 0
X	X	0	1
X	X	1	1

K1 = Q2 + Q0

3) For J2–

0 0	0 1	1 1	1 0
0	0	0	1
X	X	X	X

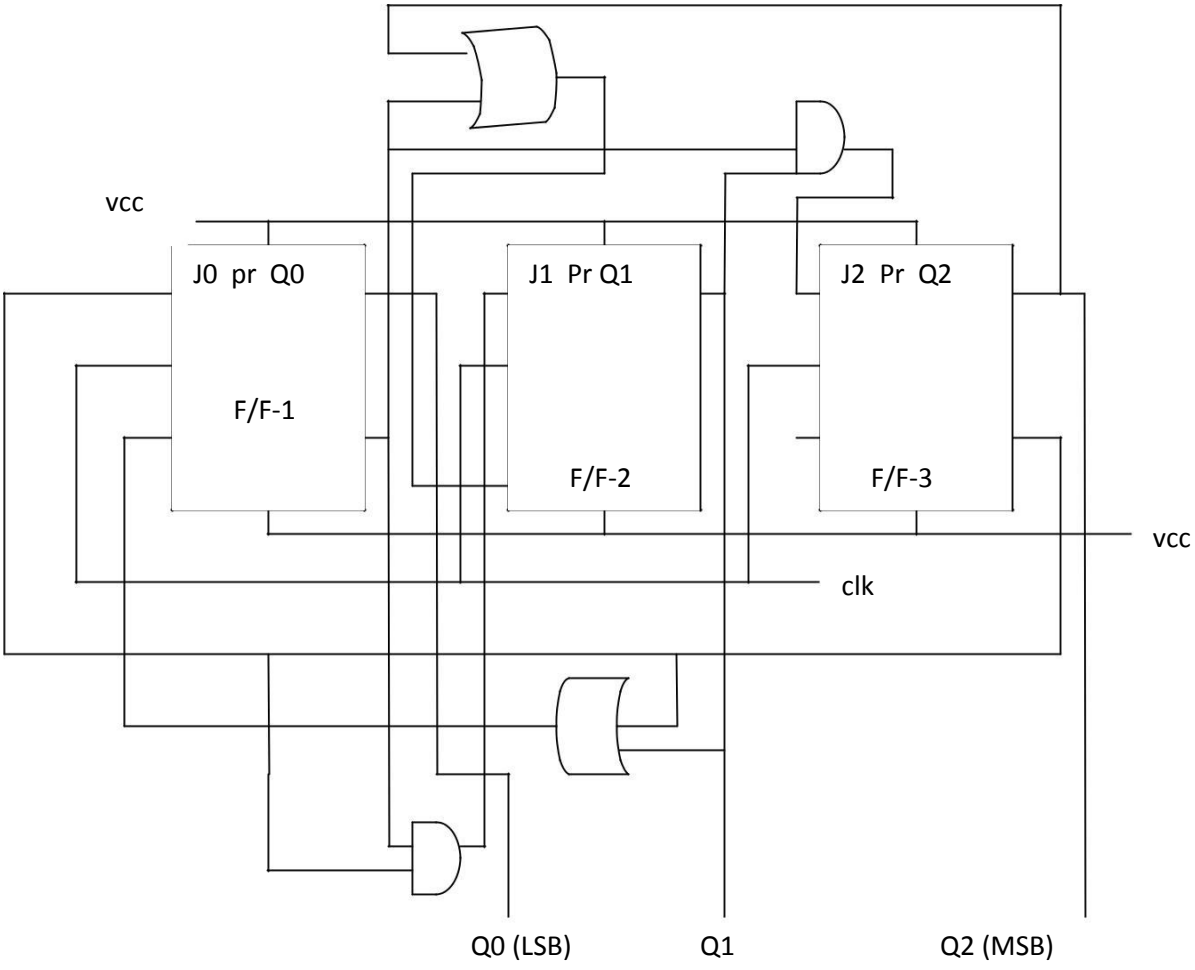
J2 = Q1Q0

3) For J1 –

0 0	0 1	1 1	1 0
X	X	X	X
1	1	1	1

K2 = 1

Logic Diagram



4) Hardware requirements:

GATE	ICs	Quantity
JK F/F	7476	2
AND	7408	1
OR	7432	1

Conclusion: In this way sequence generator with & without bushing is studied and implemented.

Enhancements / Modifications – Sequence generator can also be implemented with shift register instead of flip flops. Use IC 7495 universal shift register IC and try to implement sequence generator.

FAQs :

1. What is sequential logic circuit?

   A sequential logic circuit consists of a memory elements in addition to

   combinational circuit. Its output at any instant of time depends upon

   the present input as well as present state of memory element.

2. What is meant by delay line?

   It is used to introduce time delays in digital signals.

3. What is meant by following terms

   a) Synchronous preset b) Asynchronous preset

   c) Synchronous clear d) Asynchronous clear

- a)Preset operation is synchronised with the clock

- b)Preset operation is independent of the clock
- c) clear is performed in synchronous with clock
- d) clear is performed independent with clock

4. Is asynchronous counter faster than synchronous counter ?

   In a synchronous counter the time required for change of any state is same and is equal to delay time of one flip flop where as in asynchronous counter all flip flops are not clocked simultaneously, hence time required is not same.

5. What is mean by lockout in counter?

   In a counter design for a fewer state than the maximum possible state some time it may so happen that counter enters in unused state and goes from one unused state to another unused state and never comes to used state.

6. What is mean by state table?

   It consists of complete information about present state and next state and outputs of a sequential system.

7. What is mean by state diagram?

   The information available in a state table can be represented as graphically. the graphical representation is known as state diagram.

8. What is the advantage of state reduction in the design of sequential circuit? It reduces the number of flip flops

9. What is meant by excitation table?

   This gives information about what should be the flip flop inputs if outputs are specified before and after the clock pulse.

10. How many flip flops are required to design sequence generator using Counters:
    max $(0^{`S}, 1^{`S})$ in a given sequence $<= 2^{(N - 1)}$

    Where, N = Number of flip flops

11. How many flip flops are required to design sequence generator using shift registers:
    $S <= 2^N - 1$

Where, N=Number of flip flops

S= Length of sequence

12. What is Lock out condition? How it is avoided?

When counter enters into one of the invalid state and after application of pulses remains in invalid states only i. e. counter gets locked into invalid state & this is called as lock out. Lock out can be avoided by providing bushing to all the invalid states in such a way that after application of one or more clock pulses counter will fall into one of the valid state.

Assignment No. – 12

Title: Pseudo Random Number Generator Using IC 74194.

Aim: To study pseudo random number generator using IC 74194.

Equipments: Logic board with IC sockets & LEDs, DC power supply, IC 74194,7404,7486.

Theory: IC 74194 : Universal Shift Register

We know that a register may operate in any of the modes , like SISO, SIPO, PISO, PIPO or bi-directional. 74194 has 4 parallel data i/p (D_0-D_3) & S_0 & S_1 are the control i/ps.When S_0 & S_1 are high , data appearing on D_0-D_3 i/ps is transffered to the Q_0-Q_3 o/ps respectively, following the next Low to High transition of the clock shift right is accomplished by setting S_1 S_0 = 0 1, & serial data is entered at the shift right serial i/p, DSR. Shift left is accomplished by setting S_1 S_0 = 1 0 , & serial data is entered at the shift left serial i/p, DSL.CP is clock pulse (positive edge triggered).

Operation Mode	I/Ps							O/Ps			
	CP	MR	S_1	S_0	DSR	DSL	Dn	Q_0	Q_1	Q_2	Q_3
Reset (Clear)	×	0	×	×	×	×	×	0	0	0	0
Shift Left	↑	1	1	0	×	0	×	Q_1	Q_2	Q_3	0
	↑	1	1	0	×	1	×	Q_1	Q_2	Q_3	1
Shift Right	↑	1	0	1	0	×	×	0	Q_0	Q_1	Q_2
	↑	1	0	1	1	×	×	1	Q_0	Q_1	Q_2
Parallel Load	↑	1	1	1	×	×	Dn	D_0	D_1	D_2	D_3
Hold	×	1	0	0	×	×	×	Q_0	Q_1	Q_2	Q_3

Pseudo Random Number Generator Using 74194:

Another important application of a shift register is the pseudo random generator. It is used for generating the random sequences. The PRBS generator consists of a number of flip-flops & a combinational circuit for providing a suitable feedback.

Q_3 Q_2 Q_1 Q_0 = 0 0 1 1

Clock Pulse Number	Shift Register				Ex-OR gate	PRBS Sequence
	Q_3	Q_2	Q_1	Q_0	$Q_3 \oplus Q_2$	Q_3
0	0	0	1	1	0 ⊕ 0=0	0
1	0	1	1	0	0 ⊕ 1=1	0
2	1	1	0	1	1 ⊕ 1=0	1
3	1	0	1	0	1 ⊕ 0=1	1
4	0	1	0	1	0 ⊕ 1=1	0
5	1	0	1	1	1 ⊕ 0=1	1
6	0	1	1	1	0 ⊕ 1=1	0
7	1	1	1	1	1 ⊕ 1=0	1
8	1	1	1	0	1 ⊕ 1=0	1
9	1	1	0	0	1 ⊕ 1=0	1
10	1	0	0	0	1 ⊕ 0=1	1
11	0	0	0	1	0 ⊕ 0=0	0
12	0	0	1	0	0 ⊕ 0=0	0
13	0	1	0	0	0 ⊕ 1=1	0
14	1	0	0	1	1 ⊕ 0=1	1
15	0	0	1	1	0 ⊕ 0=0	0
16	0	1	1	0	0 ⊕ 1=1	0
17	1	1	0	1	1 ⊕ 1=0	1

The PRBS generator cannot generate a truly random sequence because this stucture is a deterministic stucture. This is the reason why the sequence repeats itself. The maximum length of the sequence will be 2^m-1.This is because the state 0 0 0…….0 must be excluded.

0 0 1 1 0 1 0 1 1 1 1 0 0 0 1 ← Binary sequence of Q_3

Length of PRBS : 2^m-1

For m=4 : 2^4-1 =15

PRBS sequence repeats itself after every 15 clock cycles.

Application of PRBS:

- Since the sequence produced is random, PRBS generator is also called as a Pseudo Noise Generator. This noise can be used to test the noise immunity of the system under test.
- PRBS generator is an important part of data encryption system. Such a system is required to protect the data from data hackers.

Procedure:

1) Adjust data o/p of $Q_3\ Q_2\ Q_1\ Q_0$ = 0 0 1 1 using parallel load operation mode.
2) Connect EX-OR gate o/p to DSR pin of IC 74194 & i/p for EX-OR Q_2 & Q_3.
3) Apply nagative clock pulse to pin-11 of IC 74194 & press trigger button to get PRB sequence at Q_3 o/p pi-12 of IC 74194.

Method-II

- Shift right operation mode of IC 74194.
- Connect MR pin to ground . All o/p of $Q_0\ Q_1\ Q_2\ Q_3$ = 0 0 0 0.
- Then shift right operation mode.

Clock Pulse	Shift Register				EX-NOR Gate	PRB Sequence
	Q_3	Q_2	Q_1	Q_0		Q_3
0	0	0	0	0	$\overline{0 \oplus 0} = 1$	0
1	0	0	0	1	$\overline{0 \oplus 0} = 1$	0
2	0	0	1	1	$\overline{0 \oplus 0} = 1$	0
3	0	1	1	1	$0 \oplus 1 = 0$	0
4	1	1	1	0	$\overline{1 \oplus 1} = 1$	1
5	1	1	0	1	$\overline{1 \oplus 1} = 1$	1
6	1	0	1	1	$\overline{1 \oplus 0} = 0$	1
7	0	1	1	0	$\overline{0 \oplus 1} = 0$	0
8	1	1	0	0	$\overline{1 \oplus 1} = 1$	1
9	1	0	0	1	$\overline{1 \oplus 0} = 0$	1

10	0	0	1	0	$\overline{0 \oplus 0} = 1$	0
11	0	1	0	1	$\overline{0 \oplus 1} = 0$	0
12	1	0	1	1	$\overline{1 \oplus 0} = 0$	1
13	0	1	0	0	$\overline{0 \oplus 1} = 0$	0
14	1	0	0	0	$\overline{1 \oplus 0} = 0$	1
15	0	0	0	0	$\overline{0 \oplus 0} = 1$	0
16	0	0	0	1	$\overline{0 \oplus 0} = 1$	0
17	0	0	1	1	$\overline{0 \oplus 0} = 1$	0
18	0	1	1	1	$\overline{0 \oplus 1} = 0$	0
19	1	1	1	0	$\overline{1 \oplus 1} = 1$	1

Procedure:

1. First reset all flip-flops of IC 74194 i.e. $Q_3\ Q_2\ Q_1\ Q_0 = 0\ 0\ 0\ 0$.
2. Shift right operation mode .
3. Connect EX-NOR means o/p of EX-OR is connected to OR gate & OR gate o/p is connected to DSR.

Conclusion:……
……
……
……
……..

Assignment No. – 13

AIM : To design & implement sequence detector to detect the given binary sequence.

OBJECTIVE : To understand sequence detector. One of the sequential circuit.

IC's USED : IC 7476, IC 7404, IC 7408, IC 7432.

THEORY : A. Clocked sequential circuit design

For designing a clocked sequential circuit, 2 models are most commonly used.

1. Mealy Model 2. Moore Mode

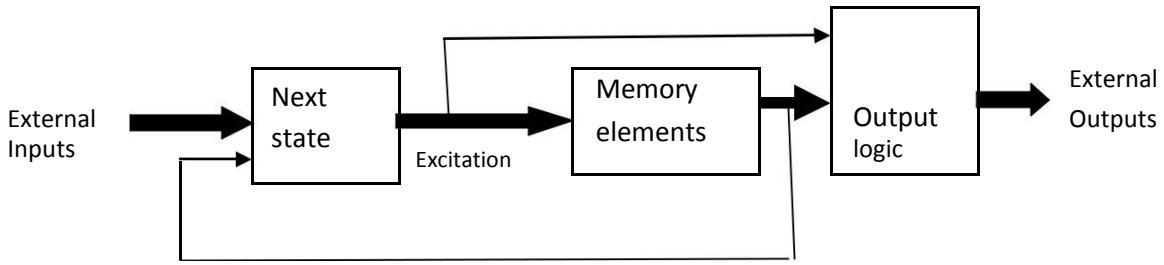


Fig. 1 Block diagram of clocked sequential circuit – Mealy Model

So for Mealy Model

Next state = F1 (present state, External inputs)

Outputs = F2 (present state, External inputs)

For Moore Model block diagram, signal path from the external inputs to the o/p. logic is not present.

so for Moore Model,

Next state = F1 (present state, External inputs)

Outputs = F2 (present state)

For clocked sequential circuits, i.e. state Machines, sequence of inputs, present state, next State, outputs can be represented by a state table or a state diagram.

State Table – General format

Sr. No.	Present State				External Inputs			Next State				External Outputs	

State Diagram

Represents information in a state table in graphical form. In this state is represented by a circle with the state indicated inside the circle. Directed lines connecting the states indicate the transition between the states when the input is applied and the circuit is clocked.

Input and output conditions for a particular transition to take place are labeled with the directed lines, 1^{st} binary number indicates input and 2^{nd} binary number indicates output.
In case, if input condition doesn‘t cause change of state, the fact is indicated by the directed line terminating on the same circle from which it is originated.

B. Sequence detector design

Sequence detector is an example of a clocked sequential circuit which is used to detect desired binary sequence.

Sequence detector is a state Machine with total no. of States = No. of bits in the binary sequence which is to be detected, one external input & one external output.

General steps to design sequence Detector

Find no. of states, in turn number of flip flops required. To do this general rule is Total no. of states (n) = Total no. of bits in the binary sequence to be detected. Minimum number of flp flops required (m) is given by relation $2^m <= n$.

1. State Assignment -

   Assign states preferably in binary.

2. State diagram –

   To draw the state diagram, start from the first state, if the bit applied on the external input is the desired bit on the sequence to be detected, we have to go to the next state otherwise we have to go to the previous state from where we can continue the desired sequence. When complete sequence is detected, make external output high or otherwise low. Once complete sequence is detected, go to the initial state.

3. Draw state table from state diagram.
4. Get reduced Boolean expression for every flip flop input and external output in terms of external input and present state.
5. Implement Boolean expression using logic gates and draw complete circuit diagram with flip flops and logic gates.

PROBLEM STATEMENT -

Design sequence detector to detect sequence **1010** using JK flip flop IC 7476.

(Use Mealy Model)

Design -

1. No. of states = 4 (no. of bits in the sequence to be detected)
   Min. no. of flip flops required (m) is given by relation $2^m < 4$.
   Min. value of m to satisfy this relation is **2**.
2. State Assignment - Binary assignment and states will be assigned as M0 (00), M1 (01), M2 (10) & M3 (11).
3. State diagram.
   a. Start with initial state. If the bit applied on the external input is the desired bit in the length to be detected, we have to go the next state when complete sequence is detected output must go

high and detector must go to the initial state otherwise the external output is low.

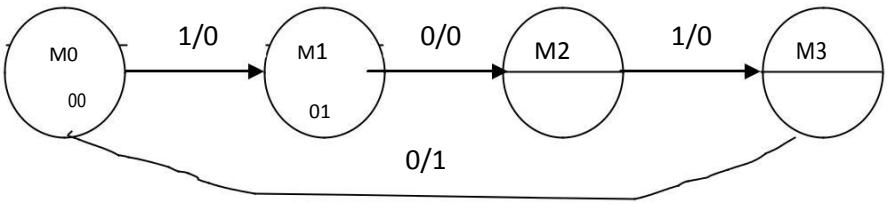


Fig. 2 State Diagram

   b. If the bit applied on the external input is not the desired bit in the sequence to be detected, we have to go to the previous state from where we can continue the desired sequence and will arrive at the minimized state diagram. To do this we have to find out the largest subsequence that is L–prefix of the given sequence to be detected i.e.

D_k (where k = length of sequence) and is also L- suffix of the received sequence on the external input, i.e. R_n (n = length of the received sequence) L= length of subsequence. Detection happens when L = k. L – Subsequence is referred as the greatest common prefix – Suffix subsequence (**GPSS**)

Finding GPSS

1. Identify the length L of the shorter sequence and produce L – prefix of D_k and L – suffix of R_n. Compare prefix and suffix. If they are same you have GPSS. Note length of GPSS as ‗m'.
2. If prefix and suffix are different reduce prefix and suffix lengths by 1 & go back to step 1.
3. If GPSS doesn't exist GPSS becomes null and length is 0 (m = 0).

In general state name will be indicated as M_m which indicates that following reset to arrive at the given state, GPSS must have a length of m bits. m is also the representation of the current number of potential matches between the received sequence and the given sequence to be detected.

Half of the state diagram is already drawn, to fill in the rest of the state diagram remember that a one bit input causes each state to lead to two possible states.

i. Consider M_o(oo) & suppose that a following a reset, a ‗0' is received instead of 1, what will be the next state.

For our problem, sequence to be detected is 1010, having length of 4, so D_4 = 1010

if R_1 = 0. To find Gpss, L = 1 (length of shorter sequence)

1 – prefix of D_4 = 1,

1 – suffix of R_1 = 0

Prefix, suffix not same , So m = 0 GPSS = Null

So if present state is M_0, received bit is 0, next state is **M_0**.

ii. Suppose following a reset 11 is received what will be the final state? So, $R_2 = 11$, $D_4 = 1010$. To find GPSS, L = 2

2 - prefix of $D_4 = 10$

2 - suffix of $R_2 = 11$

Prefix suffix not same. So reduce L by 1. L = 1

1 - prefix of $D_4 = 1$

1 - suffix of $R_2 = 1$

Prefix, suffix are same, so GPSS = 1 and its length m=1. So final state is M_1.

iii. Suppose following a reset 100 is received what will be the final state?

So, $R_3 = 100$, $D_4 = 1010$. To find GPSS, L = 3

3 – Prefix of $D_4 = 101$

3 – Suffix of $R_3 = 100$

Not same so reduce L by 1, L = 2

2 – prefix of $D_4 = 10$

2 – suffix of $R_3 = 00$.

Not same so reduce L by 1, . . L = 1

1 – Prefix of $D_4 = 1$

1 – Suffix of $R_3 = 0$

Not same, so GPSS doesn‘t exist. GPSS = Null & m = 0

So final state is **M_0**.

iv. Suppose following a reset 1011 is received what will the final state ?

So, $R_4 = 1011$, $D_4 = 1010$. To find GPSS, L = 4

4 - Prefix of $D_4 = 1010$

4 - Suffix of $R_4 = 1011$.

Not same so reduce L by 1 . . L = 3

3– Prefix of D_4 = 101

3– Prefix of D_4 = 011.

Not same reduce L by 1, . . L = 2

2 – Prefix of D_4 = 10

2 – Suffix of R_4 = 11

Not same reduce L by 1, . . L = 1

2 – Prefix of D_4 = 1

2 – Suffix of R_4 = 1

Prefix & suffix are Same, so GPSS = 1, length m = 1

So final state is $\mathbf{M_1}$.

11. Sequence Detector

Above discussion leads to following state diagram.

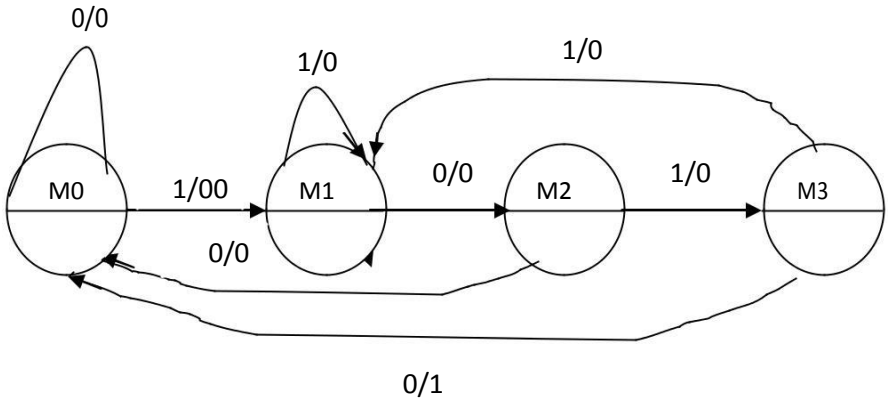


Fig. 3 Complete State Diagram

4. State Table

Sr. No.	Present State		Ext. Input	Next State		Ext. Output	Flip Flop inputs			
	Q_B	Q_A	X	Q_B	Q_A	Y	J_B	J_B	J_A	J_A
1	0	0	0	0	0	0	0	X	0	X

2	0	0	1	0	1	0	1	X	0	X
5.3 **G**	1	0	0	1	0	0	X	1	1	X
4 **e**	1	0	1	0	1	0	X	0	0	X
5 **t**	0	1	0	0	0	0	0	X	X	1
6 **t**	0	1	1	1	1	0	1	X	X	0
7 **I**	1	1	0	0	0	1	X	1	X	1
8 **II**	1	1	1	0	1	0	X	0	X	1

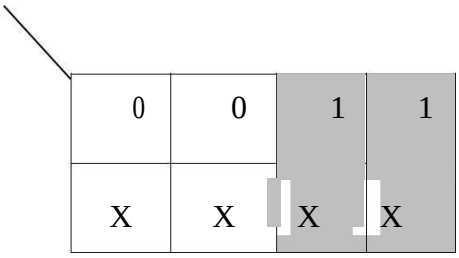
g

M

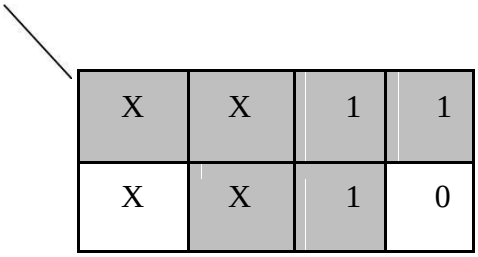
6. M

11. Sequence Detector

5. Minimized Boolean expression for each flip flop input & external output with K-Map



$$J_A = X$$



$$K_A = A + X$$

0	X	X	0
1	X	X	0

$$J_B = Q_A X$$

X	1	0	X
X	1	1	1

$$K_B = X + Q_A$$

11. Sequence Detector

0	0	0	0
0	0	1	0

$$Y = XQ_BQ_A$$

7. Logic Diagram –

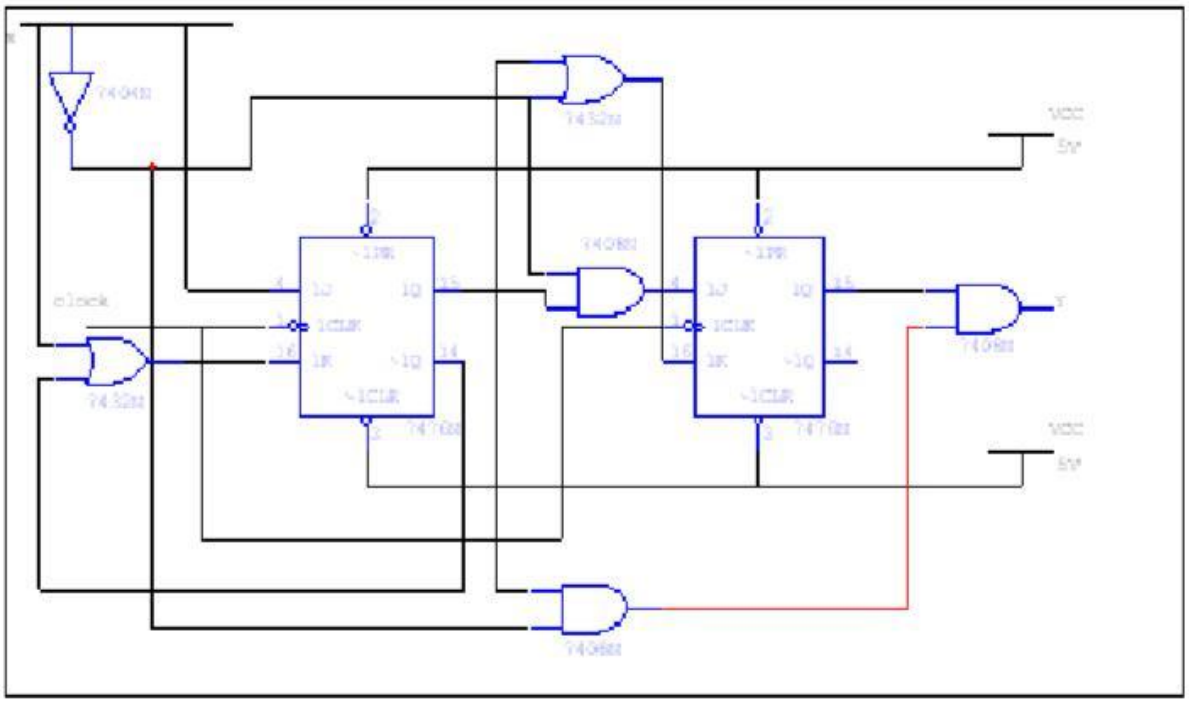


Fig. 4 Logic Diagram of Sequence Detector

8. Hardware Requirements Table:

GATE / Flip Flop	IC	Quantity
MS JK FF	7476	1
NOT	7404	1
AND	7408	1
OR	7432	1

Conclusion: Thus GPSS (Greatest common Prefix Suffix Subsequence) can be used to design sequence detector to get optimized state diagram with minimum

hardware. Designed sequence detector is successfully implemented and tested for different input binary sequences.

Enhancements / Modifications – Design (upto state diagram) the sequence detector to detect different binary sequences**.**

FAQ's :

1. What is the state diagram?

   In this state is represented by a circle with the state indicated inside the circle. Directed lines connecting the states indicate the transition between the states when the input is applied and the circuit is clocked.

   Input and output conditions for a particular transition to take place are labeled with the directed lines, 1^{st} binary number indicates input and 2^{nd} binary number indicates output. In case, if input condition doesn't cause change of state, the fact is indicated by the directed line terminating on the same circle from which it is originated.

2. What do you mean by sequence detector?

   Sequence detector is an example of a clocked sequential circuit which is used to detect desired binary sequence.

11. Sequence Detector

Sequence detector is a state Machine with total no. of States = No. of bits in the binary sequence which is to be detected, one external input & one external output.

3. Sequence detector is a combinational or sequential logic circuit?

   Sequence detector is an example of a clocked sequential circuit

4. How many flip-flops we need to design _n' bit sequence detector?

Find no. of states, in turn number of flip flops required. To do this general rule is :

Total no. of states (n) = Total no. of bits in the binary sequence to be detected.

Minimum number of flp flops required (m) is given by relation $2^m <= n$.

5. What is state table?

State table is a table which provides information about sequence of external inputs, present state, next State, external outputs of a state machine and its general format is

Sr. No.	**Present State**				**External Inputs**			**Next State**				**External Outputs**	

6. What do you mean by Moore model and Mealy model? for Mealy Model

Next state = F1 (present state, External inputs)

Outputs = F2 (present state, External inputs)

For Moore Model block diagram, signal path from the external inputs to the o/p. logic is not present.

so for Moore Model,

11. Sequence Detector

Next state = F1 (present state, External inputs)

Outputs = F2 (present state)

7. What is the advantage of state reduction in the design of sequential circuits?

State reduction gives optimized state diagram & minimum hardware.

8. Which one is preferred in FSM design? Mealy or Moore model? Why?

The option to include input in output generation logic gives certain advantage to Mealy model.

1. Usually it requires less number of states and thereby less hardware to solve any problem.
2. Also the output is generated one clock cycle earlier.

9. What are the disadvantages of Mealy model ?

The input transients, glitches are directly conveyed to the output. Also if we want Output transitions to be synchronized while input can change any time . Hence Mealy model is not preferred.

10. What are the advantages of Moore model over Mealy model?

In this model the output remains stable over entire clock period and changes only when there occurs a state change at clock trigger based on input available at that time.

11. Is there any difference in hardware requirement between Moore and Mealy machine? Yes. Comparatively less hardware is requirement in Mealy machine than Moore machines.

12. What are the disadvantages of state transition diagrams?

Though state transition diagrams are more compact in representation, for relatively more complex problem where number of inputs and states are higher the state diagram space becomes so crowded that it is difficult to read. In such situations ASM charts are preferred

Assignment No - 14

- ***Title:*** **Simple ASM using**
- ***Objective:*** Learn multiplexer controller method
- ***Problem Statement :*** Design of 2 bit Up counter using multiplexer controller method

1. 2 bit counter has 4 states i.e. 00,01,10,11.
2. In the state diagram if mode control M =0, counter will be latched in the same state and will start incrementing to the next state if M=1.
3. By referring the state diagram, ASM chart is drawn.

- ***Hardware & software requirements:*** Digital Trainer Kit, 74151(8:1 MUX), 7474 (D Flip-flop), Power supply, Patch Cord.

Theory:

1. ASM chart means algorithmic state machine chart.
2. It is a type of flowchart that can be used to represent the state transitions and generated outputs for finite state machine(FSM)
3. ASM charts are similar to traditional flowcharts.
4. Unlike a traditional flowchart, this includes timing information. This chart specifies that the FSM flows from one state to another only after each active clock edge.

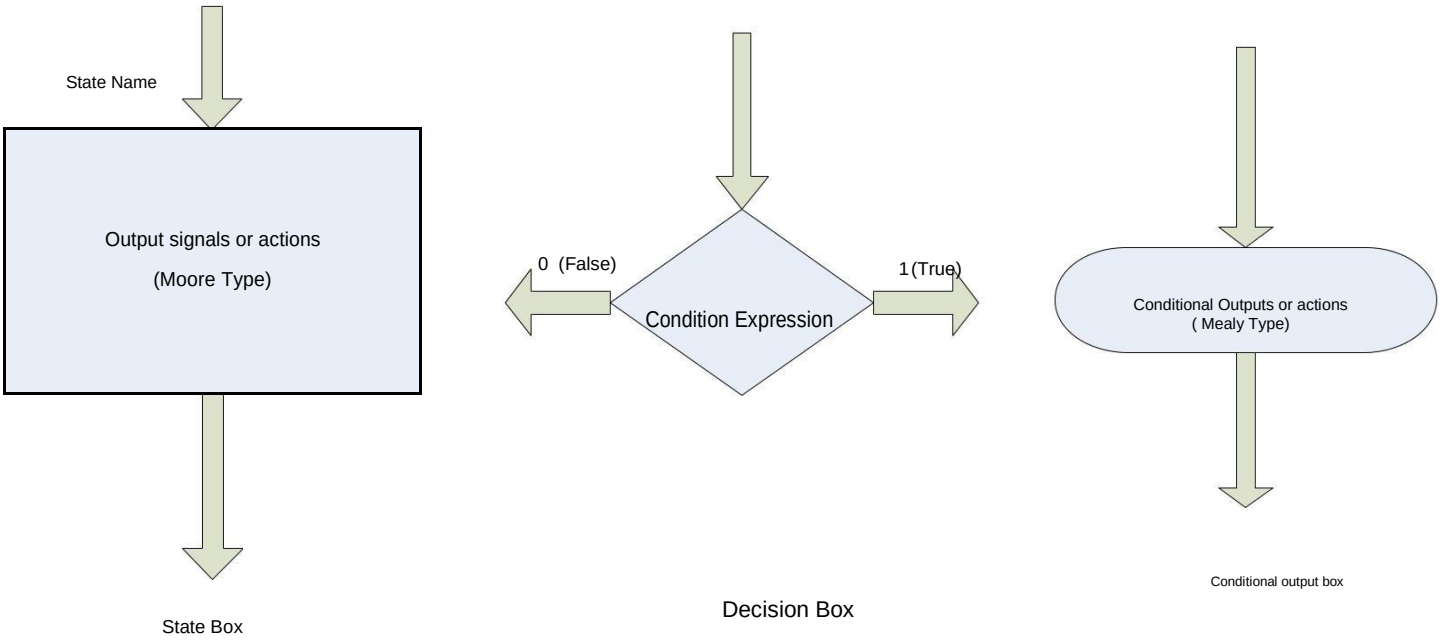
- *Elements used in ASM Chart*

1. **State Box-** A rectangle represents a state of the FSM. It is equivalent to node in the state diagram or row in the state table. The name of the state should be indicated outside the box in left top corner. Moore type of outputs is listed inside the box.

2. **Decision Box-** A diamond indicates that the stated condition expression has to be tested and an exit path has to be chosen accordingly. The condition expression consists of one or many inputs.

3. **Conditional output Box-** The oval denotes the output signals that are of Mealy type. These output depend on the values of state variables and the Inputs of FSM .the condition that determines whether such Outputs are generated is specified in a decision box.

These are shown in the figure given below.



Significance : It is an aid to design the complex circuits.ASM charts are used to describe complex circuits that include one or more FSM's and another circuitry such as registers,counters,adders,multipliers,etc.

ASM Block

1. It is a structure which consists of single state box and any decision and conditional output boxes that the state box may be connected to.
2. It has one entrance and any number of exit paths.

Each block describes the state of the system during the interval of one clock pulse. Multiplexer controller method of design has three levels of components as shown in the figure below:

ASM using Multiplexer controller Method:-

1. The multiplexer outputs are applied to the input of the flip-flop forming the register at the second level to hold the present state inputs.
2. The multiplexers decide the next state of the register as outputs of MUX has been connected to flip-flop inputs.
3. Third level is the decoder which provides separate output for each control state. The decoder can be replaced by the combinational circuit.

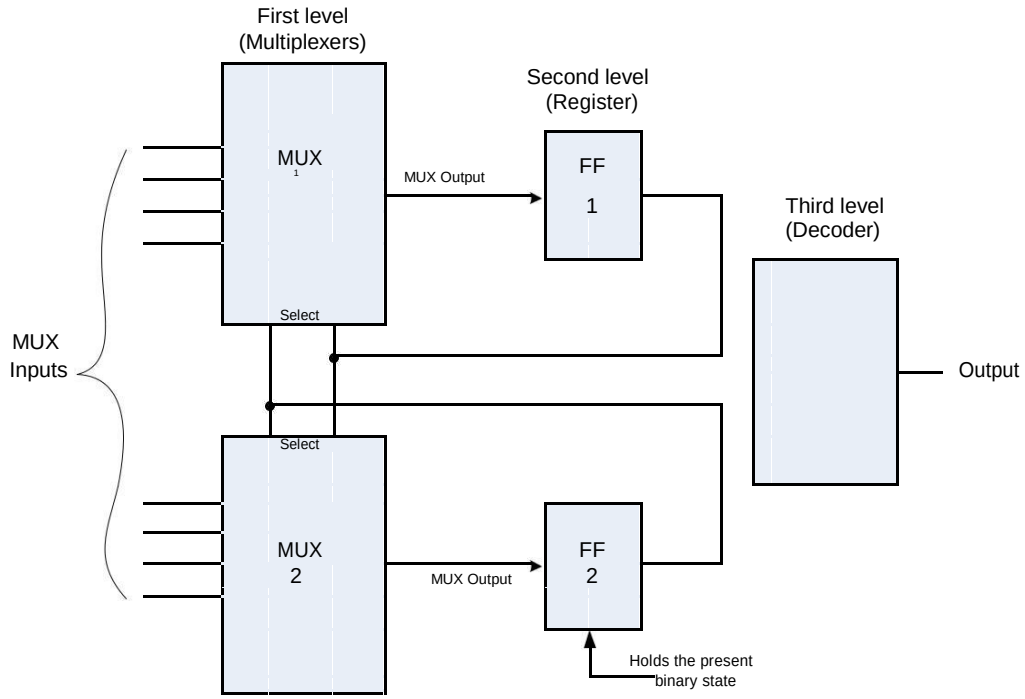
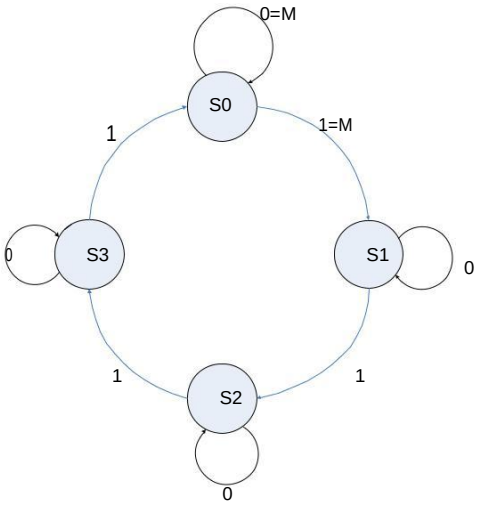


Fig. Block schematic for a 3-level scheme for multiplexer design

State diagram:- for 2 bit Up counter:



Up counting takes place for M=1and the counter progresses from states S0, S1, S2, S3.

Fig. 1 State Diagram for 2 bit Up Counter

ASM chart of the above state diagram is as shown below in Fig.2

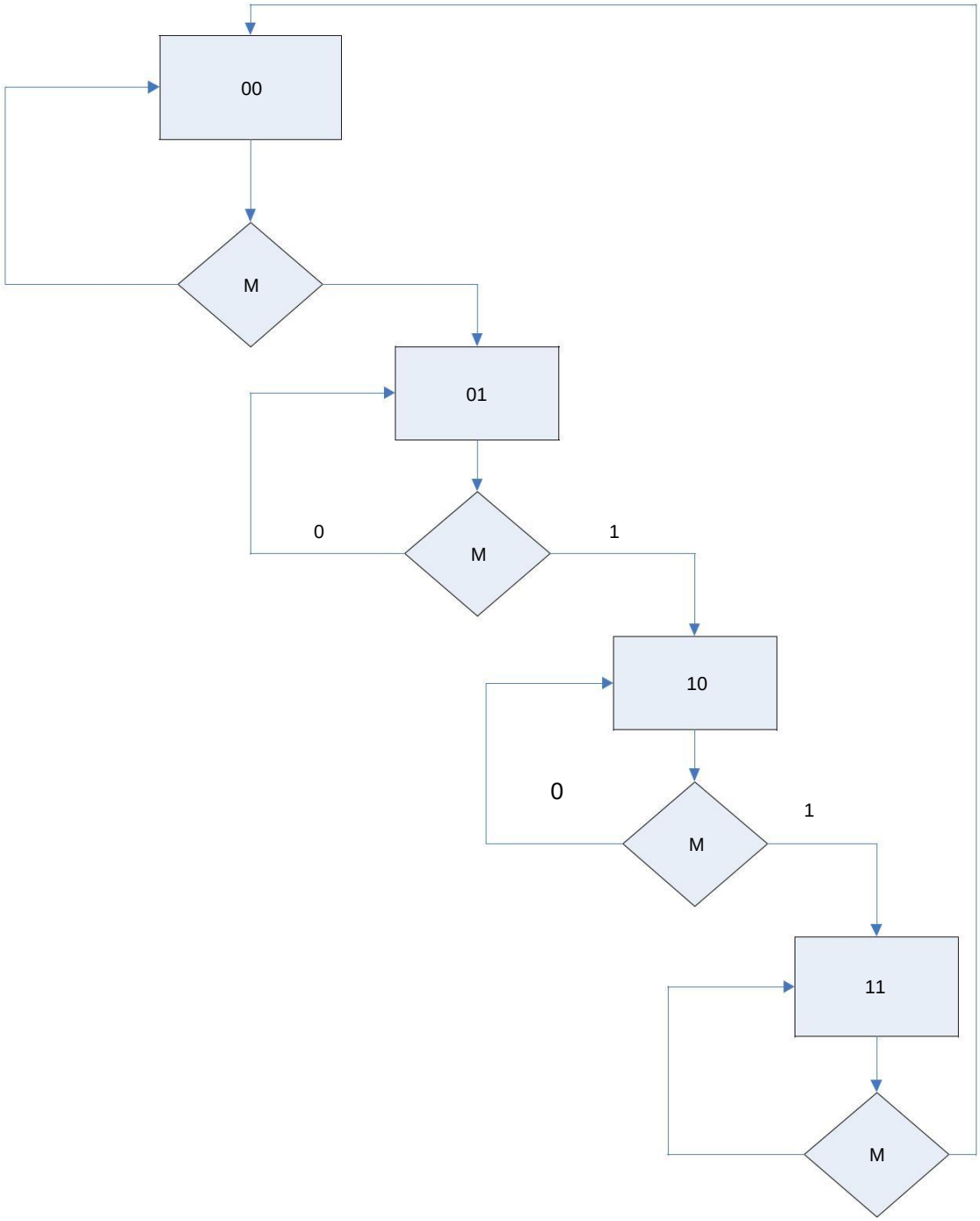


Fig.2 ASM Chart for 2-bit Up Counter

State transition table:

Mode control i/p	Present state (Qn)		Next state (Q n+1)	
M	Qb	Qa	Qb+1	Q a+1
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

Excitation table for D Flip-flop:

Present State	Next state	Input
Qn	Qn+1	Dn
0	0	0
0	1	1
1	0	0
1	1	1

State Table :

Mode control i/p	Present state (Qn)		Next state (Qn+1)		Input	
M	Qb	Qa	Qb+1	Q a+1	Db	Da
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	1	0	1	0
0	1	1	1	1	1	1
1	0	0	0	1	0	1
1	0	1	1	0	1	0
1	1	0	1	1	1	1
1	1	1	0	0	0	0

Logic Diagram:

ASM Using MUX

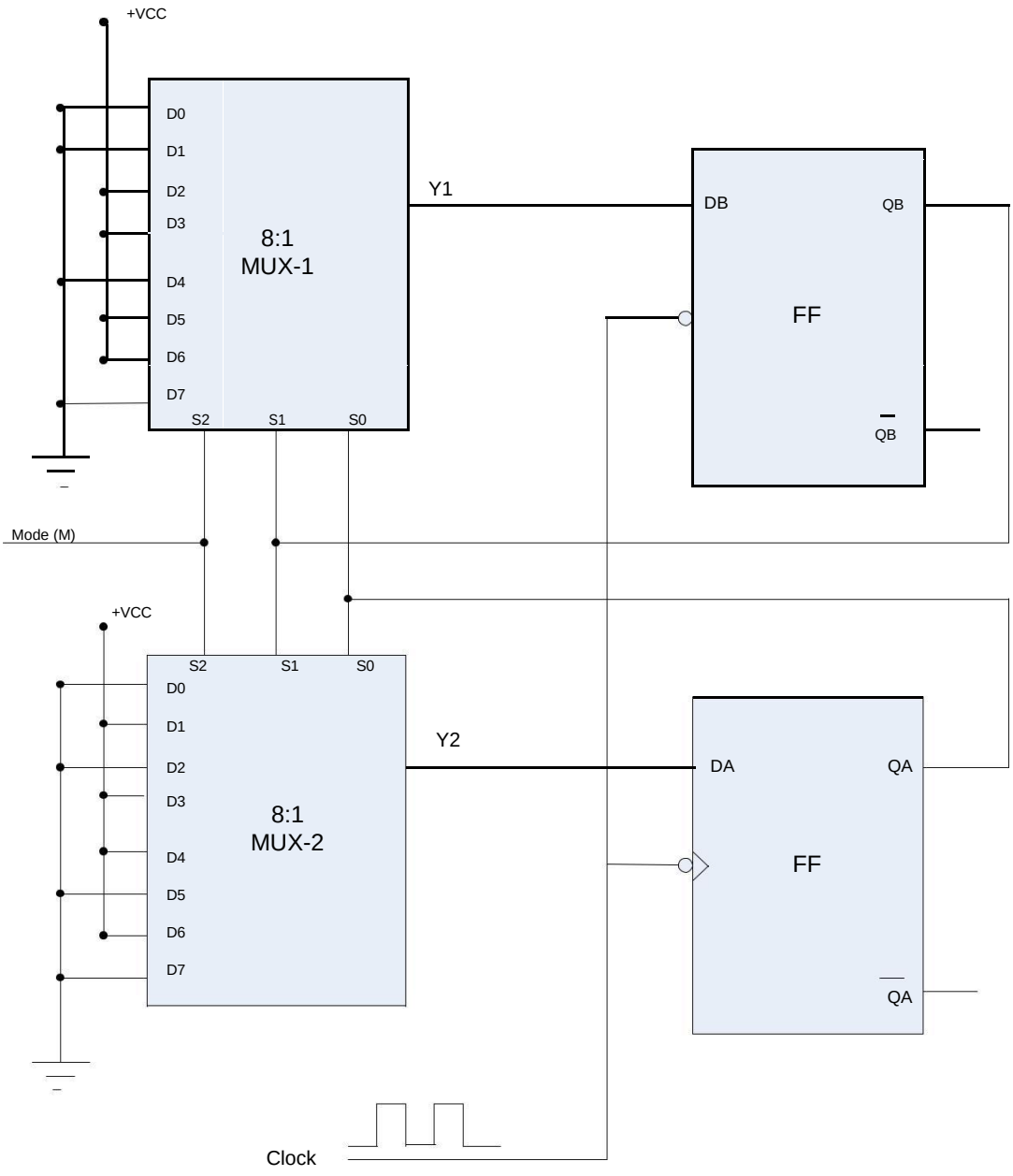


Fig. Logic Diagram

Outcome: ASM chart is drawn as per the state diagram and verified the functionality of given FSM using multiplexer controller method.

FAQ's:

1. **What is meaning of ASM and FSM**

ASM is algorithmic state machine chart.It is a method to implement FSM. It is a type of flowchart that can be used to represent the state transitions and generated outputs for finite state machine (FSM).

A finite state machine (FSM) or finite state automaton (plural: *automata*) or simply a state machine, is a model of behavior composed of a finite number of states, transitions between those states, and actions. A finite state machine is an abstract model of a machine with a primitive internal memory.

2. **What is the major difference between ASM chart and traditional flowchart?**

ASM charts are similar to traditional flowcharts. Unlike a traditional flowchart, this includes timing information. This chart specifies that the FSM flows from one state to another only **after each active clock edge**.

3. **Write the significance of ASM chart in the design of FSM.**

It is an aid to design the complex circuits. ASM charts are used to describe complex circuits that include one or more FSM‘s and another circuitary such as registers, counters, adders, multipliers etc.

Assignments Questions:

GROUP - C

Assignment No: 15

- ***Title:*** **Programmable Logic Array**
- ***Objective*:** To learn Programmable logic device how to design it
- ***Problem Statement***: To Design and implement the combinational logic using PLA
- ***Hardware & software requirements:*** Digital Trainer Kit, IC 7432**,** IC 7408, IC 7404, Patch Cords, +5V Power Supply

Theory:

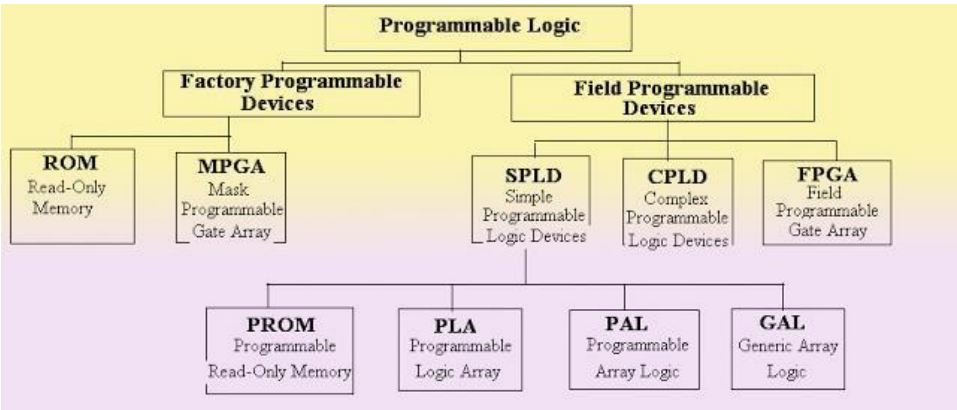


Fig 1: Type of PLD

PLA is a programmable logic device. PLA consist of two level AND-OR configuration. The input are applied to the AND matrix through input buffer and output buffer of AND matrix is applied to the OR matrix. The input of two or more
PLA devices should be connected individually in parallel, then this connection
the Number of input and the number of product terms will remain unchanged but
the Number of output lines is increased.

✓ The size of PLA specified as I*P*O. I= denotes the number of input.
P = corresponds to the no of product terms.
O= denotes the number of output.

Applications:

1. We can implement the combinational circuit using PLA.

2. We can also implement the sequential circuit using PLA.

3. For implementation the combinational circuit, the PLA device with only output buffer are used whereas to implement the sequential circuit we use the PLA device with flip flop and buffers included in output stage.

Block Diagram:

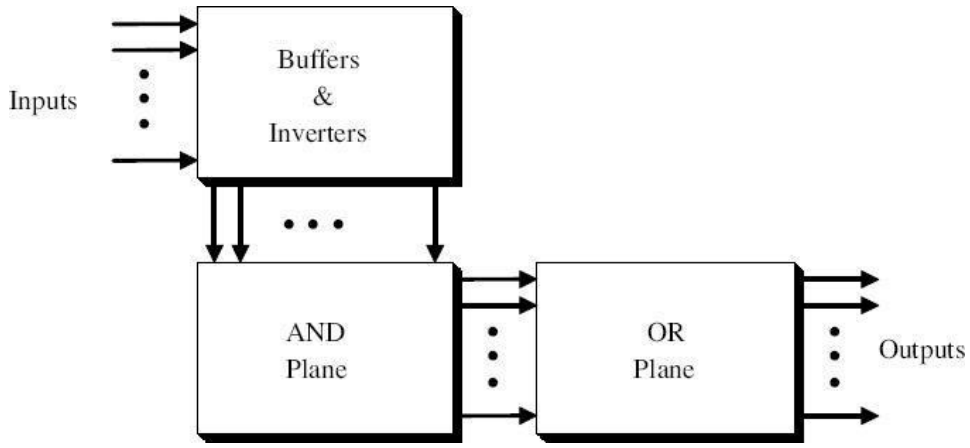


Fig 2: Building Blocks of PLA

✓ **BCD To Excess – 3 Code Conversion:**

Convert BCD 2 i. e. 0010 to Excess – 3 code
For converting 4 bit BCD code to Excess – 3, add 0011 i. e. decimal 3 to the respective code using rules of binary addition.

0010 + 0011 = 0101 – Excess – 3 code for BCD 2

✓ ***Design:***

Step 1: Truth Table

Table 1: BCD to Excess-3 Code Conversion

INPUT (BCD CODE)				OUTPUT (EXCESS-3 CODE)			
B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Step 2: K-Map

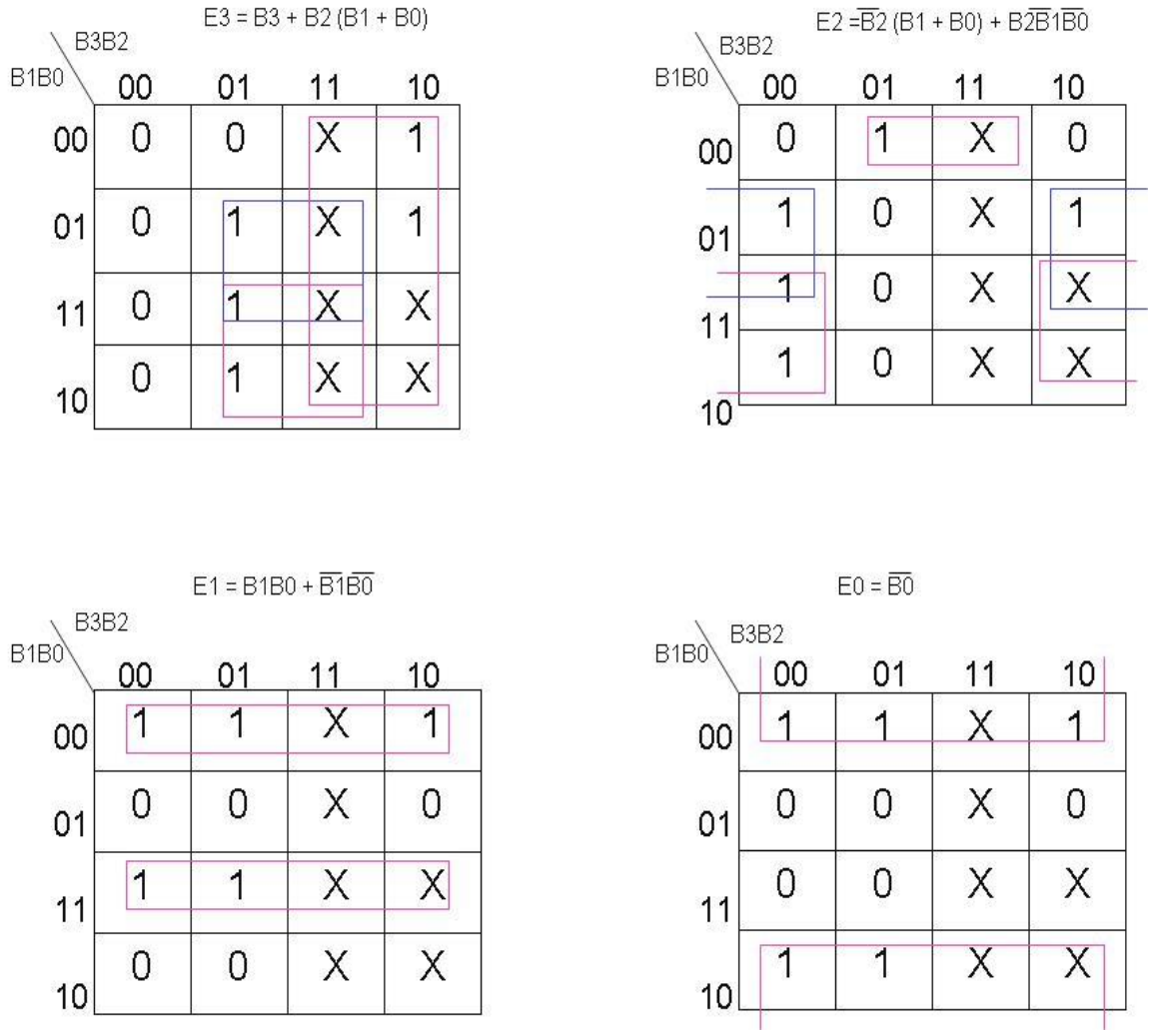


Fig. 8 K-Map For Reduced Boolean Expressions Of Each Output (Excess-3 Code)

The four simplified expression and associated product terms are as follow:-**E3**=

B3 + B2B0 + B2B1

| | |

(1) (2) (3)

E2= $\overline{B2}$B1 + $\overline{B2}$B0 + B2$\overline{B1}\overline{B0}$

| | |

(4) (5) (6)

E1= B1B0 + B1B0

| |

(7) (8)

E0 =$\overline{B0}$

|

(**9**)

Step 3: Prepare the PLA program table

Table 2: PLA programming table

Product term No.	**Product term**	**I/P**				**O/P**			
		B3	**B2**	**B1**	**B0**	**E3**	**E2**	**E1**	**E0**
1	B3	1	_	_	_	1	_	_	_
2	B2B0	_	1	_	1	1	_	_	_
3	B2B1	_	1	1	_	1	_	_	_
4	$\overline{B}$2B1	_	0	1	_	_	1	_	_
5	$\overline{B2B0}$	_	0	_	1	_	1	_	_
6	$\overline{B2}\,\overline{B1B0}$	_	1	0	0	_	1	_	_
7	$\overline{B1B0}$	_	_	0	0	_	_	1	_
8	B1B0	_	_	1	1	_	_	1	_
9	$\overline{B0}$	_	_	_	0	_	_	_	1

Step 4: Implementation of the PLA circuit

The logic circuit for BCD to Excess-3 converter using a PLA device Is shown below

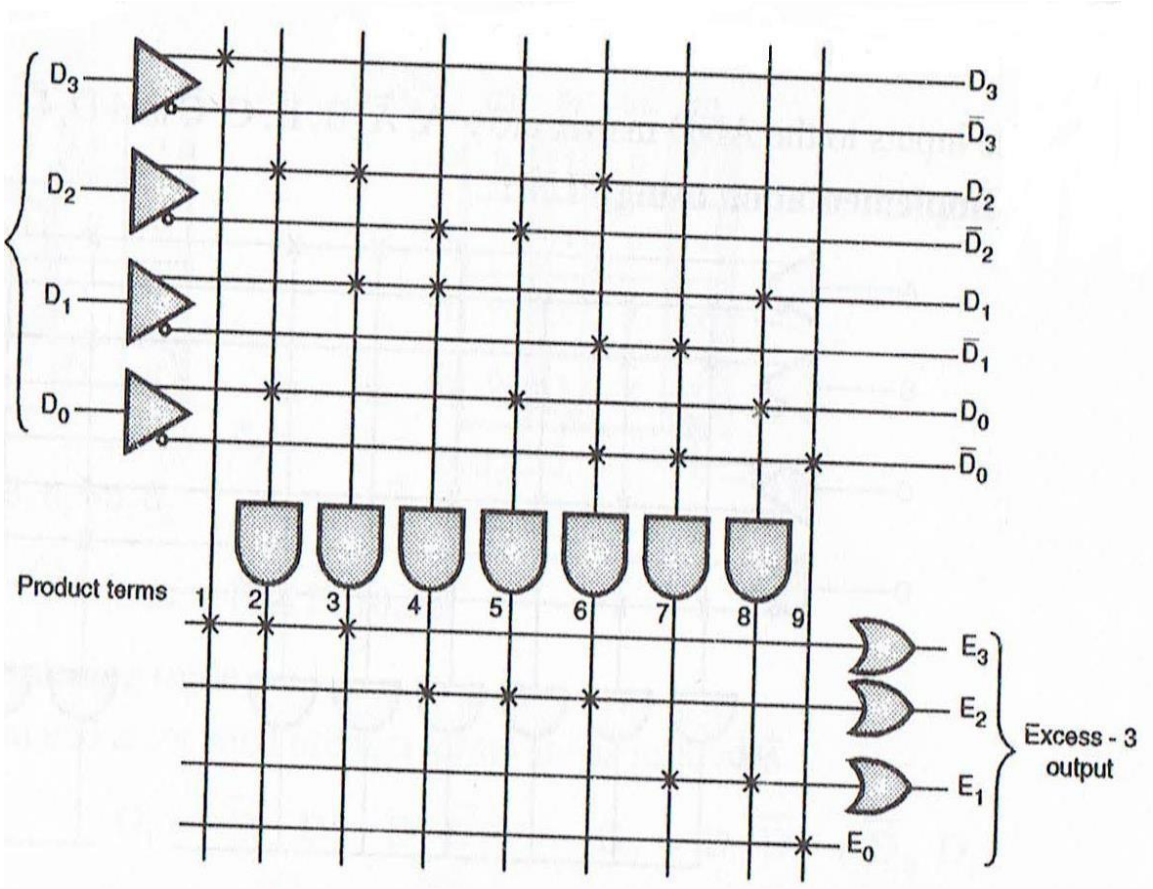


Fig 3: PLA logic circuit for BCD to Excess-3 Convertor

Outcome:

Test the circuit for all possible combinations of input and output codes.

Assignment Question:

Assignment No. – 16

- ***Title:*** Design and simulation of - Full adder, Flip flop, MUX using VHDL (Any 2) Use different modeling styles.
- ***Objective:*** Learn H/W programming
- ***Problem Statement :*** Implementation of Full Adder and VHDL for 4:1 MUX
- ***Hardware & software requirements:*** Verilog Simulator

Introduction to VHDL:

Very High-Speed Integrated Circuit (VHSIC) HDL (VHDL) was developed by US army in 1982.VHDL is a programming language for describing the behavior of digital systems. This language has constructs that enable to express the concurrent or sequential behavior of digital system with or without timing, it also allows interconnecting component.

VHDL is one of a few HDLs in widespread use today. VHDL is recognized as a standard HDI by the Institute of Electrical and Electronics Engineers (IEEE Standard 1076, ratified in 1987 and by the United States Department of Defense (MIL-STD-454L).

VHDL has many features appropriate for describing the behavior of electronic component ranging from simple logic gates to complete microprocessors and custom chips.VHDL allows the behavior of complex electronic circuits to be captured into a design system for automatic circuit synthesis or for system simulation.

One of the most important applications of VHDL is to capture the performance specification for circuit, in the form of what is commonly referred to as a test bench.

Features Of VHDL:

- World wide Popularity.
- VHDL supports different types of modeling.
  - Structural
  - Dataflow
  - Behavioral
  - Mixed
- Also VHDL can be used at different complexity levels-from single transistor up to Complete systems and everything remains in the same simulation environment.
- The language supports flexible design methodologies top down, bottom up or mixed. The language can be used as communication medium between different CAD and CAE tools.
- The language supports hierarchy that is digital system can be modeled as a set of interconnected components or subcomponents.

- It supports both synchronous and asynchronous timing models.
- Test benches can be written using the same language to test other VHDL models.

VHDL Program Format:

As shown in Fig. a standalone piece of VHDL code is composed of at least three fundamental sections:

- **LIBRARY declarations :** Contains a list of all libraries to be used in the design. For example: ieee, std, work, etc.
- **ENTITY:** Specifies the 110 pins of the circuit.
- **ARCHITECTURE :** Contains the VHDL code properties, which describes how the circuit should behave (function).

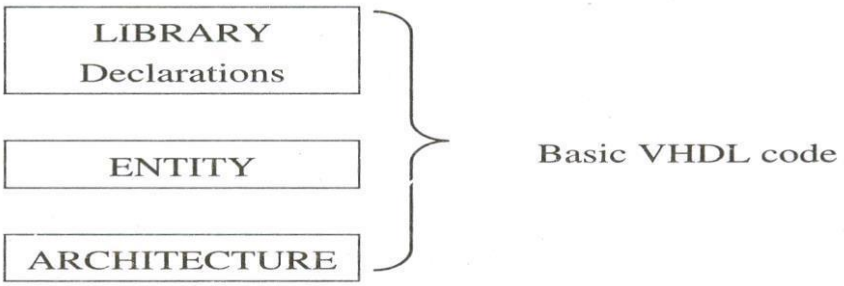


Fig.1: Fundamental units of VHDL code

1.LIBRARY Declarations :

A LIBRARY is a collection of commonly used pieces of code. Placing such pieces inside a library allows them to be reused or shared by other designs.

The typical structure of a library is illustrated in fig. 2 . The code is usually written in the form of FUNCTIONS, PROCEDURES, or COMPONENTS, which are placed inside PACKAGES, and then compiled into the destination library. To declare a LIBRARY two lines of code are needed one containing the name of the library and the other a use clause, as shown in the syntax below.

```
LIBRARY     library_name;

USE         Library_name_package_name.package_parts;
```

At least three packages, from three different libraries, are usually needed in a design that is:

- ieee.std_logic_1 164.all (from the ieee library)
- standard (from the std library)
- work (work library).

Their declarations are as follows:

```
LIBRARY ieee;                -- A semi-colon (;) indicates
```

```
USE ieee.std_logic_1164.all;        -- the end of a statement or
LIBRARY std;                        -- declaration, while a double
USE std.standard.all;               -- dash (--) indicates a comment.
LIBRARY work;
USE work.all;
```

2.Entity:

A VHDL entity specifies the name of the entity, the ports of the entity, and entity-related information. All designs are created using one or more entities. Entity is the description of the interface between a design and external environment. An entity defines the input and output ports of a design.A design can contain more than one entity.Each entity has its own architecture statement.

The syntax of entity is as shown in following:

entity entity_name is

port (port_name : **mode** port_type;

port_name: **mode** port_type);

end [entity_name];

The following sections explains the different elements. of entity syntax.

Entity_name:

The name of the entity.

Port_name:

The name of the port.

Port_type:

A previously defined data type.

MODE:

There are five modes available in VHDL for ports:

- In: input port. A variable or a signal can read a value from a port of mode in, but is not allowed to assign a value to it.
- Out : output port. It is allowed to make signal assignments to a port of the mode out, but it is not legal to read from it.

- Inout : bi-directional port. Bothassignments to such a port and reading from it are allowed.
- Buffer : output port with read capability. It differs from inout in that it can be updated by at most one source, whereas inout can be updated by zero or more sources.

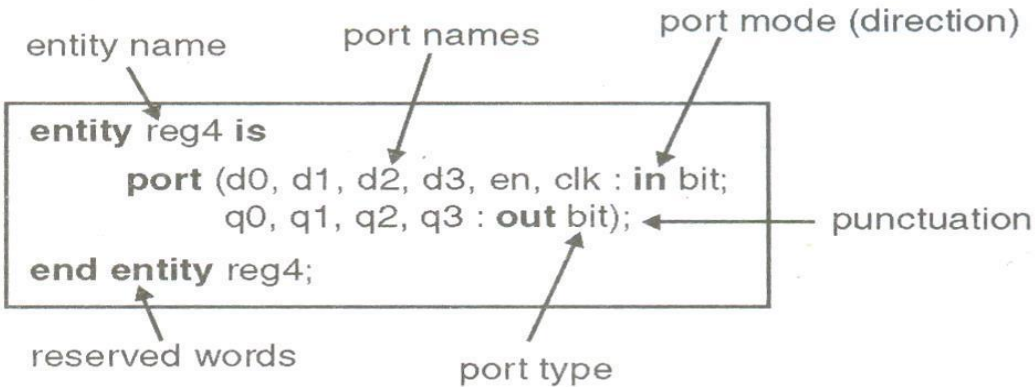


Fig. Legend of Entity declaration

3.Architecture:

The architecture describes the underlying functionality or internal organization or operation of the entity and contains the statements that model the behavior of the entity. Architecture body is used to describe the behavior, data flow or structure of a design entity. Architecture body is used to describe internal details of a design entity using following modeling styles.

1. Behavior- As a set of sequential assignment statements.

2. Dataflow- As a set of concurrent assignment statements.

3. Structure- As a set of interconnected components.

4. Mixed- As a set of combination of above three.

- Architecture is always related to an entity and describes the behavior of that entity.

The syntax is:
architecture architecture_name **of** entity_name **is**

{ block_declarative_item }

begin

{ concurrent_statement }

end [architecture_name];

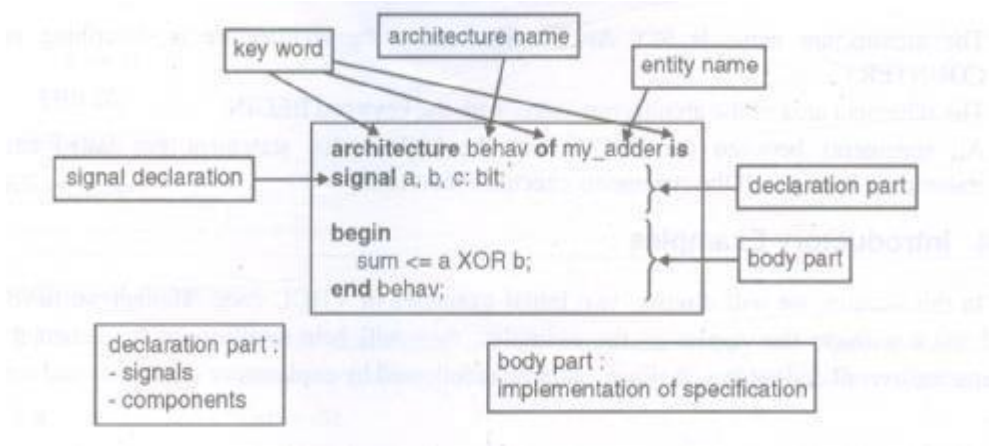


Fig. Architecure Declaration

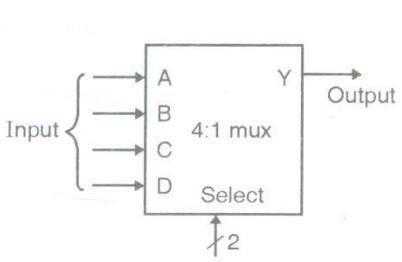
4:1 mux using if statement:

• An If statement generally produces priority-encoded logic, therefore it results in slower circuit overall.

• An If statement can contain a set of different expressions.

• Most current synthesis tools can determine if the if-elsif conditions are mutually exclusive, and will not create extra logic to build the priority tree.

Example: Implement 4:1 Multiplexer using VHDL.



(a) Symbol

Select		Output Y
S_1	S_0	
0	0	A
0	1	B
1	0	C
1	1	D

(b) Truth table

• The following examples use an If construct in a 4:1 multiplexer design.

```
library ieee;
use ieee.std_logic_1164.all;

entity MUX4_1 is
        port   (Sel : in std_logic_vector(1 downto 0);
                A, B, C, D : in std_logic;
                Y : out std_logic );
end MUX4_1;

architecture behavior1 of MUX4_1 is
begin
    process (Sel, A, B, C, D) -- Using an if statement is not as clear as using case
        begin
            if      (Sel = "00") then  Y<= A;
            elsif   (Sel = "01") then  Y<= B;
            elsif   (Sel = "10") then  Y<= C;
            else    Y<= D;
            end if;
        end process;
end behavior1;
```

----------Vhdl code for 4:1 Mux ----------

-- Company:

-- Engineer:

--

-- Create Date: 14:24:21 10/09/2012

-- Design Name:

-- Module Name: mux - Behavioral

-- Project Name:

-- Target Devices:

-- Tool versions:

-- Description:

--

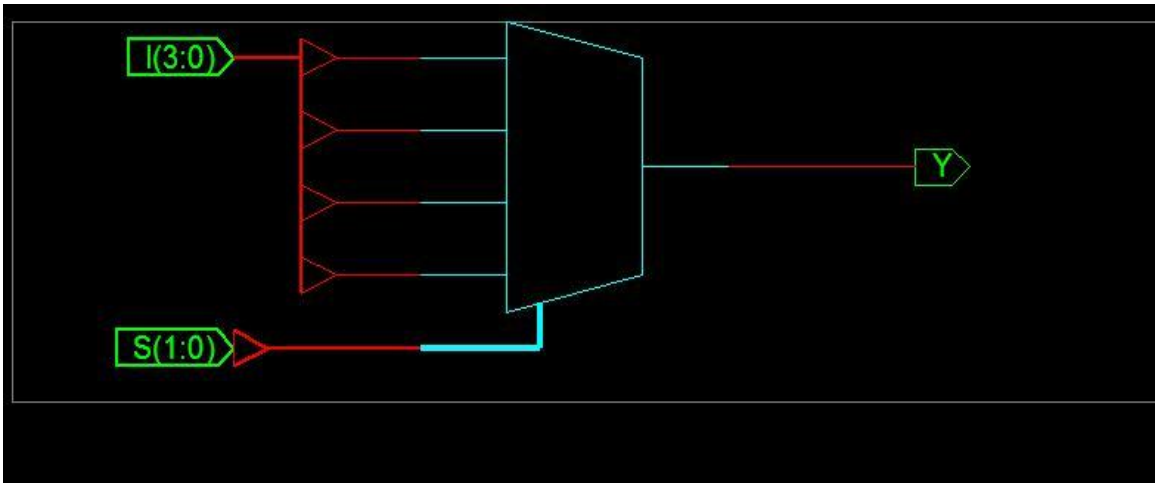
-- Dependencies:

```
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity mux is
    Port ( I : in STD_LOGIC_VECTOR (3 downto 0);
           S : in STD_LOGIC_VECTOR (1 downto 0);
           Y : out STD_LOGIC);
end mux;

architecture Behavioral of mux is

begin
with S select
```

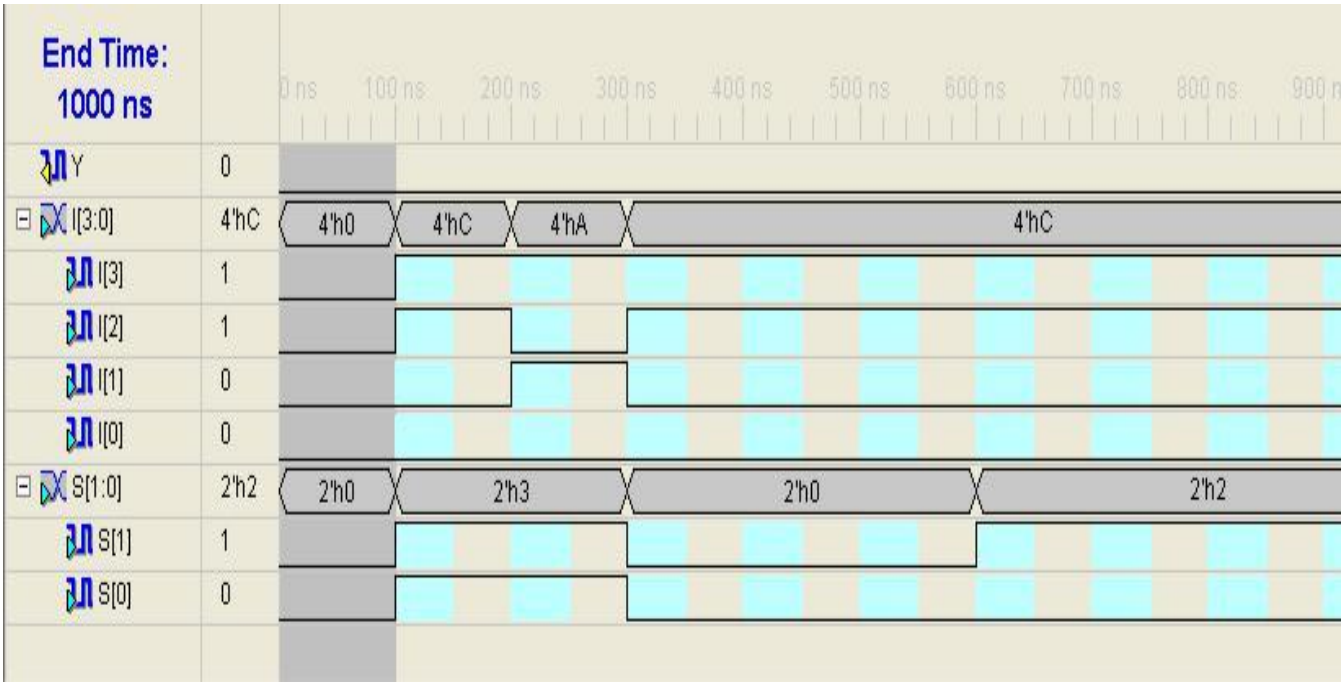
```
Y<=I(0) when "00",

I(1) when "01",

I(2) when "10" ,

I(3) when others;



end Behavioral;
```

----------end code----------

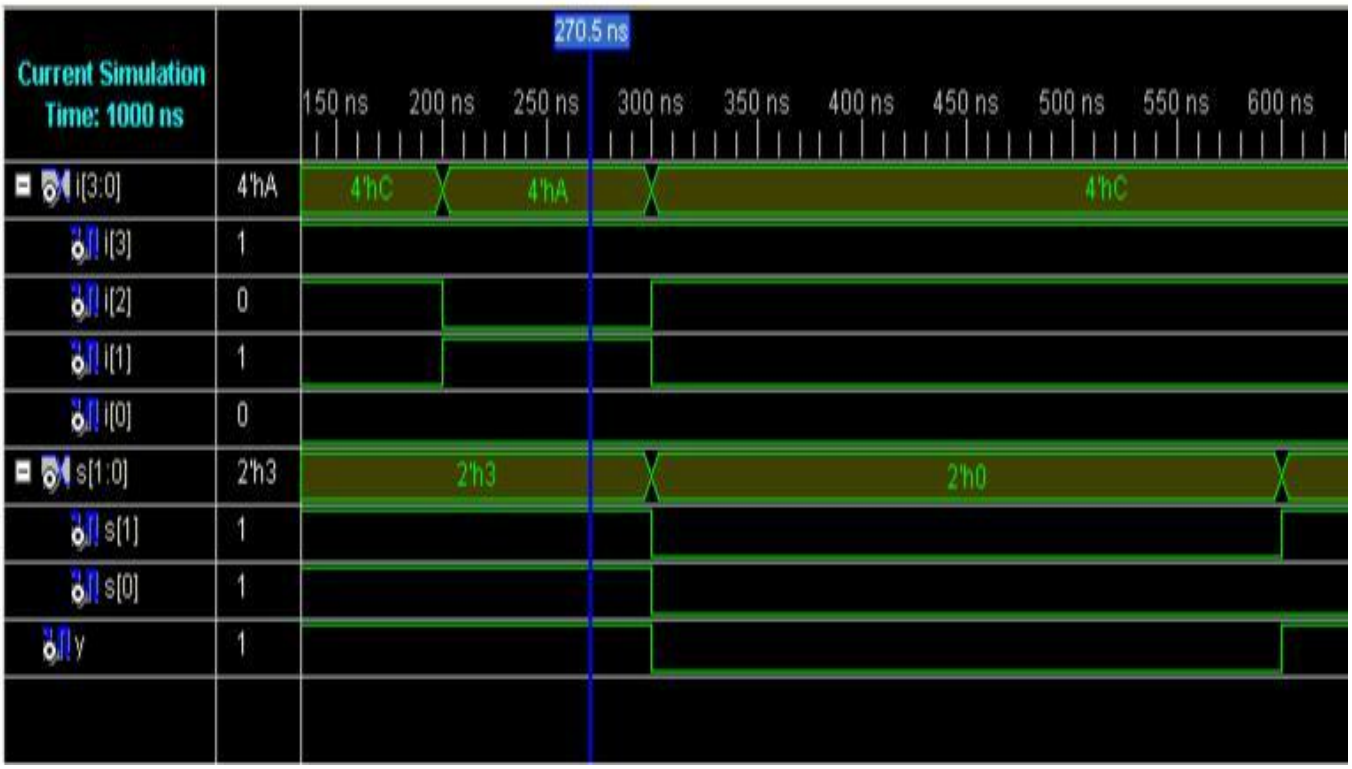
View RTL Schematics:-



Test Beanch Wave Form:-



Simulation:-



Outcome: Successfully implemented 4:1 MUX using behavioral modeling

Assignment No. – 17

AIM: - Design & simulate asynchronous 4- bit counter using VHDL. .

Theory: VHDL Statements:

There are two classes of statements you can use in your VHDL descriptions:

- Sequential statements
- Concurrent statements

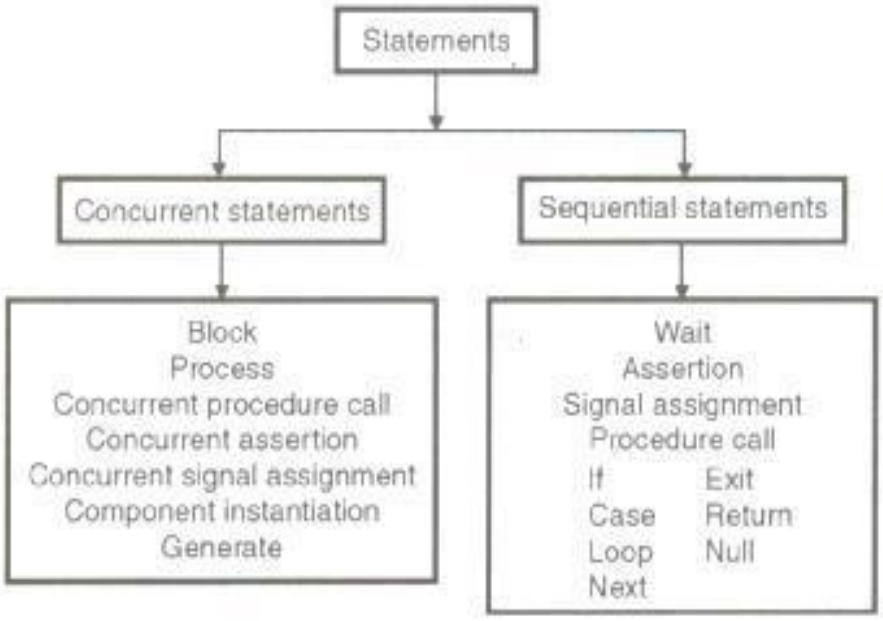


Fig. A VHDL Statements

A VHDL architecture construct comprises a set of interconnected concurrent statements, such as processes and blocks, which describe an overall design in terms of behavior or structure. Concurrent statements in a design execute simultaneously, unlike sequential statements, which execute one after another.

Sequential Statements:

A set of VHDL statement that executes. in sequence is called sequential statements. Sequential statements can appear only in processes and subprograms. You can use sequential statements only inside a process statement or within a subprogram (procedure or function). Each statement executes in the order in which it is encountered. The preceding BNF description listed the sequential statements available in VHDL.

Sequential statements are divided into categories, based on their operation. The following list shows the categories for the sequential statements: The types of sequential statements are:

- Assignment Statements and Targets

- Variable Assignment Statements
- Signal Assignment Statements
- If Statements
- Case Statements
- Loop Statements
- Next Statements
- Exit Statements
- Subprograms
- Return Statement
- Wait Statements
- Null Statements

Concurrent Statements:

The functionality of a design is defined in VHDL by a set of concurrent statements. These statements mimic hardware in that many of these statements can be active at the same time. All concurrent statements describe the functionality of multiplexer structures. It is impossible to model storage elements, like Flip Flops with concurrent statements, only. The main concurrent statements:

1. Process Statements
2. Block Statements

Concurrent Versions of Sequential Statements:

1. Concurrent Procedure Calls
2. Concurrent Signal Assignments
3. Component Instantiation Statements
4. Direct Instantiation
5. Generate Statements

Process Statements:

A process statement (which is concurrent) contains a set of sequential statements. Although all processes in a design execute concurrently, the sequential statements within each process one at a time.A process communicates with the rest of the design by reading values from or writing them to signals or ports outside the process.The process statement in VHDL is the primary means by which sequential operations (such as registered circuits) can be described.

The process statement represents the behavior of some portion of the design. It consists of the sequential statements whose execution is made in order defined by the user. Each process can be assigned an optional label. The process declarative part defines local items for the process and may contain declarations of: subprograms, types, subtypes, constants, variables, files, aliases, attributes, use clauses and group declarations. It is not allowed to declare signals or shared variables inside processes.

The statements, which describe the behavior in a process, are executed sequentially, in the order in which the designer specifies them. The execution of statements however does not terminate with the last statement in the process, but is repeated in an infinite loop.

1.Assignment Statements and Targets:

Use an assignment statement to assign a value to a variable or signal.

The syntax is:

target := expression; -- Variable assignment

target <= expression; --Signal assignment

Target:

The target can be a variable or a signal (or part of a variable or a signal, such as a sub array) that receives the value of the expression. The expression must evaluate to the same type as the target.

The difference in syntax between variable assignments and signal assignments is that

— Variables use the: = operator

Variables are local to a process or subprogram, and their assignments take effect immediately.

— Signals use the <= operator

Signals need to be global in a process or subprogram, and their assignments take effect at the end of a process. Signals are the only means of communication between processes.

1.1 Variable Assignment Statements:

A variable assignment changes the value of a variable.

The syntax is:

target: — expression;

Target:

Names the variables that receive the value of expression.

Expression:

Determines the assigned value; its type must be compatible with the target.

When a variable is assigned a value, the assignment takes place immediately. A variable keeps its assigned value until another assignment takes place.

1.2 Signal Assignment Statements:

A signal assignment changes the value being driven on a signal by the current process.

The syntax is:

target <= expression;

Target:

Names the signals that receive the value of expression.

Expression

Determines the assigned value; its type must be compatible with target.

Signals and variables act in different ways when they receive values. The differences lie in the way the two kinds of assignments take effect and how that influence the value reads from either variables or signals.

2. IF Statements:

The if statement is a statement that depending on the value of one or more corresponding conditions, selects for execution one or none of the enclosed sequences of statements.

The syntax is:

1) **if-then statement**

if condition **then**

sequential_statements

end if;

2) **if-then-else statement**

if condition **then**

sequentiaL_statements

else

sequential_statements

end if;

3) **if-then-elsif-else statement**

if condition **then**

sequential_statements

eLsif condition **then**

sequential_statements

else

sequential_statements

end if;

3.Case Statements:

Definition:

The case statement selects for execution one of several alternative sequences of statements; the alternative is chosen based on the value of the associated expression.

The syntax is:

case expression **is**

when choices =>

{ sequentiaL_statement

} **{ when** choices =>

{ sequentiaL_statement } }

end case;

4.Loop Statements:

Definition:

A loop statement repeatedly executes a sequence of statements.

The syntax is:

[Label:] [iteration_scheme]**loop**

{ sequential_statement }

{ next [tabel] [**when** condition] ; }

{ exit [label] [condition] ; }

end Loop [label];

5.For---Loop Statements

The for loop is a sequential statement that allows you to specify a fixed number of iterations in a behavioral design description. The for loop includes an automatic declaration for the index (i in following example). You do not need to separately declare the index variable.

The syntax is:

[label:] for identifier **in range Loop**

{ sequential_ statement }

end loop [label]

6. Next Statements:

The next statement allows skipping a part of an iteration loop. If the condition specified after the when reserved word is TRUE, or if there is no condition at all, then the statement is executed.

The syntax is:

next [label] [**when** condition];

7. Exit Statements:

The exit statement completes execution of an enclosing iôop statement, called label in the syntax. The completion is conditional if the statement includes a condition, such as the when condition.

The execution of the exit statement depends on a condition placed at the end of the statement, right after the when reserved word. When the condition is TRUE (or if there is no condition at all) the exit statement is executed and the control is passed to the first statement after the end loop.

The syntax is:

exit [**label**] [**when** condition];

8. Return Statements:

The return statement ends the execution of a subprogram (procedure or function) in which it appears. It causes an unconditional jump to the end of a subprogram.

The syntax is:

return expression ;. -- Functions

return; -- Procedures

9. Null Statements:

The null statement explicitly states that no action is required. It is often used in case statements because all choices must be covered, even if some of the choices are ignored.

The Syntax is:

null;

Example: Implement 4-bit binary asynchronous down counter.

The behavioral VHDL code for the 4-bit binary down counter is Shown in Fig.B

The statement USE IEEE.STD_LOGIC_UNSIGNED. ALL is needed in order to perform subtractions on STD_LOGIC_VECTORs.

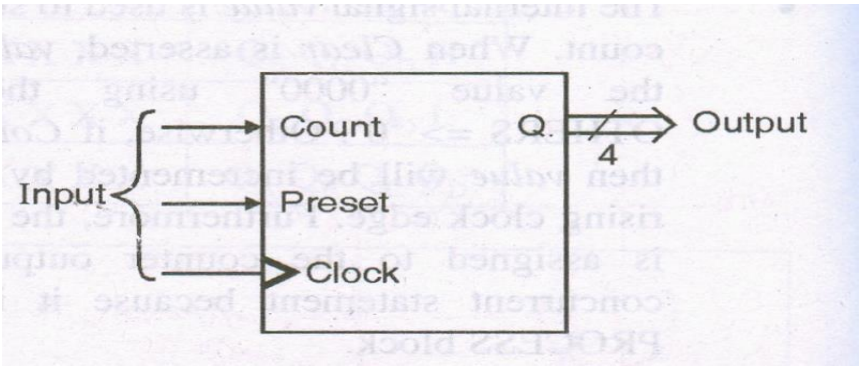


Fig.B 4-bit binary down counter

Assignment No. – 18

Title: Implementation of combinational logic using PALs.

Aim: To study implementation of combinational logic using PALs.

Equipments: Logic board with IC sockets & LEDs, DC power supply, IC 7432,7404,7408, 7411.

Theory:

While designing with PAL, the function in its sum of products form need to be simplified to fit into each section. Since OR array is fixed, number of product terms per OR gate cannot be changed. Hence if number of product terms in any function are more, then it may be necessary to section to implement the Boolean function. In PALs , unlike the PALs, a product term cannot be shared among two or more OR gates.

Example:

3 i/p, 4 o/p combinational circuit has following o/p functions.

Implement the circuit using suitable PAL.

$A(x,y,z) = \sum m (1,2,4,6)$

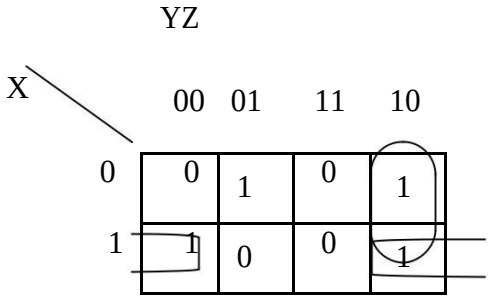
$B(x,y,z) = \sum m (0,1,3,6,7)$

$C(x,y,z) = \sum m(2,6)$

$D(x,y,z) = \sum m(1,2,3,5,7)$

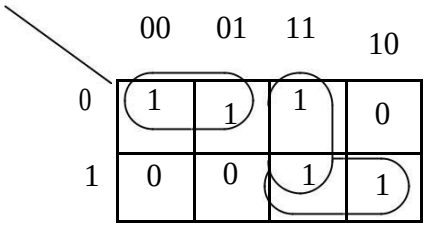
Solution:

K-Maps for o/p A,B,C & D are as follows. **For A**



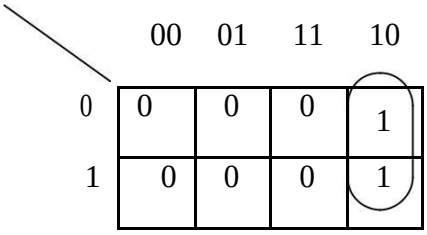
$A = \overline{X}\,\overline{Y}\,Z + X\,\overline{Z} + Y\,\overline{Z}$

For B



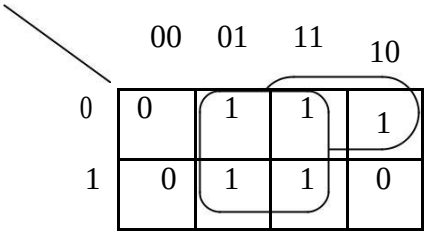
$$B = \overline{X}\,\overline{Y} + Y\,Z + X\,Y$$

For C



$$C = \overline{Y}\,Z$$

For D



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

The PAL program table shown in below table contains three columns specifying product terms , inputs & outputs.

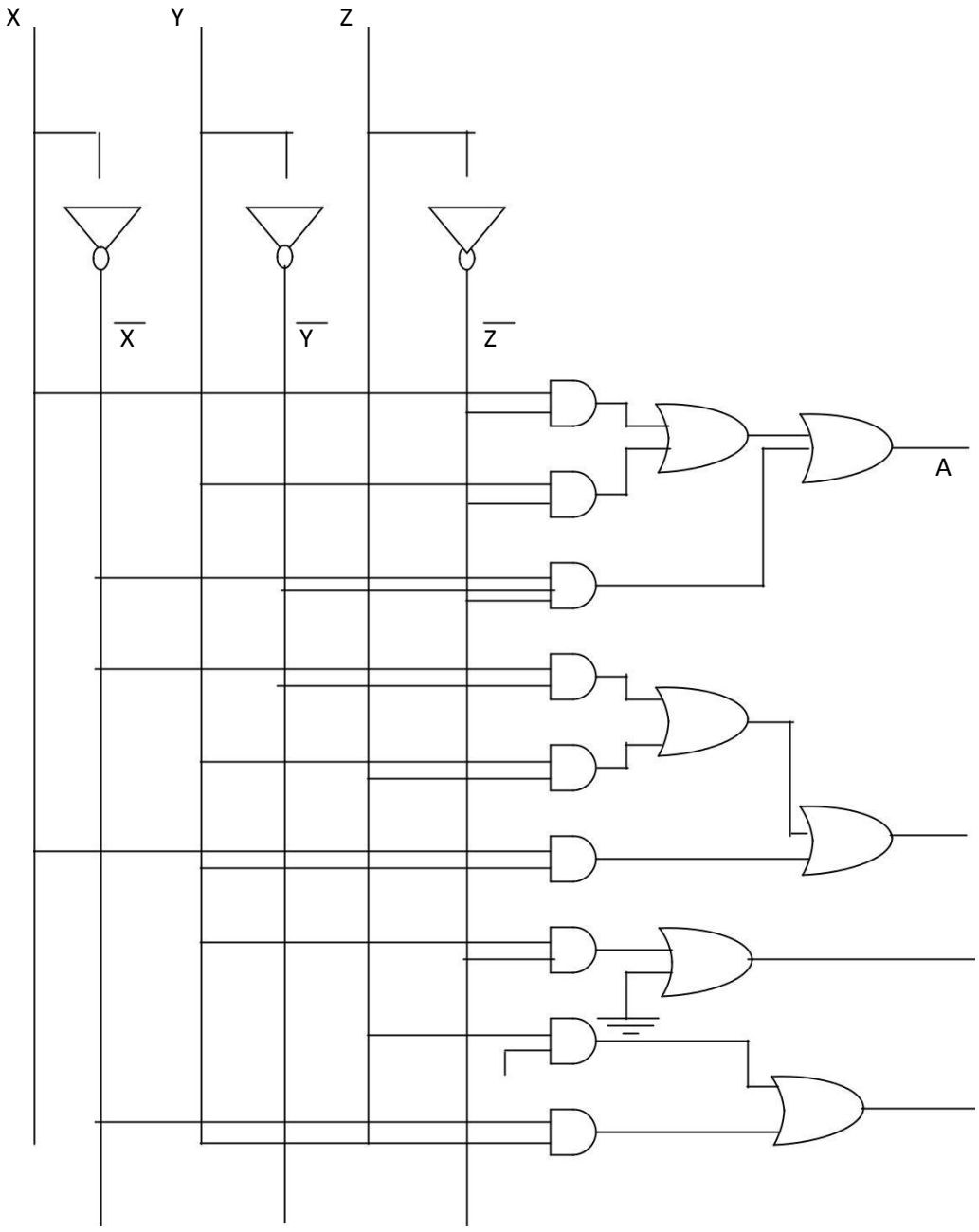
	Product Terms	AND Input			Output
		X	Y	Z	
$X\overline{Z}$	1	-	-	0	$A = X\overline{Z} + Y\overline{Z} + \overline{X}\,\overline{Y}\,Z$
$Y\overline{Z}$	2	-	1	0	
$\overline{X}\,\overline{Y}\,Z$	3	0	0	1	
$\overline{X}\,\overline{Y}$	4	0	0	-	$B = \overline{X}\,\overline{Y} + X\,Y + Y\,Z$
YZ	5	-	1	1	
XY	6	1	1	-	
$Y\overline{Z}$	7	-	1	0	$C = Y\overline{Z}$
-	8	-	-	-	
-	9	-	-	-	
Z	10	-	-	1	$D = Z + \overline{X}\,Y$
$\overline{X}\,Y$	11	0	1	-	
-	12	-	-	-	

Procedure:

1. Connect the circuit as shown in diagram.

2. Connect DC power supply and check all the conditions given in the PAL program table.

Practically Implemented PAL:



Design Expt.

1) Implement the circuit using suitable

PAL. A(x,y,) = ∑m (0,1)

B(x,y,) = ∑m (2,3)

2) Implement the circuit using suitable

PAL. A(x,y,z) = ∑m (0,7,6,4)

B(x,y,z) = ∑m (1,2,3,4)

C(x,y,z) = ∑m(2,6)

D(x,y,z) = ∑m(1,2,3,5,7)

3) Implement the circuit using suitable

PAL. A(x,y,z,w) = ∑m (4,6,8,10)

B(x,y,z,w) = ∑m (1,3,5,7)

C(x,y,z,w) = ∑m(12,14)

D(x,y,z,w) = ∑m(10,11,12,15)

Conclusion:

GROUP - D