

# Digital Logic Circuits

## Unit 1

## 2. Course Objectives

- ✿ To introduce various number systems and methods to simplify the logical expressions using Boolean functions
- ✿ To understand the combinational circuits
- ✿ To design various synchronous and asynchronous circuits.
- ✿ To introduce asynchronous sequential circuits and PLDs
- ✿ To introduce digital simulation for development of application oriented logic circuits

### 3. Pre Requisites (Course Names with Code)

✿ The pre requisite course is:

✿ Circuit Theory EE 8251

## 4. Syllabus (With Subject Code, Name, LTPC details)

❁ EE8351      DIGITAL LOGIC CIRCUITS      L T P C

2 2 0 3

### ❁ UNIT I NUMBER SYSTEMS AND DIGITAL LOGIC FAMILIES 6+6

Review of number systems, binary codes, error detection and correction codes (Parity and Hamming code) - Digital Logic Families -comparison of RTL, DTL, TTL, ECL and MOS families -operation, characteristics of digital logic family.

### ❁ UNIT II COMBINATIONAL CIRCUITS 6+6

Combinational logic - representation of logic functions-SOP and POS forms, K-map representations - minimization using K maps – simplification and implementation of combinational logic –multiplexers and de multiplexers – code converters, adders, subtractors, Encoders and Decoders.

### ❁ UNIT III SYNCHRONOUS SEQUENTIAL CIRCUITS 6+6

Sequential logic- SR, JK, D and T flip flops - level triggering and edge triggering – counters -asynchronous and synchronous type - Modulo counters - Shift registers – design of synchronous sequential circuits – Moore and Melay models- Counters, state diagram; state reduction; state assignment.

### ❁ UNIT IV ASYNCHRONOUS SEQUENTIAL CIRCUITS AND PROGRAMMABILITY

#### LOGIC DEVICES 6+6

Asynchronous sequential logic circuits-Transition stability, flow stability-race conditions, hazards & errors in digital circuits; analysis of asynchronous sequential logic Circuits, introduction to Programmability Logic Devices: PROM – PLA –PAL, CPLD-FPGA.

### ❁ UNIT V VHDL 6+6

RTL Design – combinational logic – Sequential circuit – Operators – Introduction to Packages – Subprograms – Test bench. (Simulation /Tutorial Examples: adders, counters, flip flops, Multiplexers & De multiplexers).

TOTAL : 60 PERIODS



#### ✿ TEXT BOOKS:

- ✿ 1. James W. Bignel, Digital Electronics, Cengage learning, 5th Edition, 2007.
- ✿ 2. M. Morris Mano, 'Digital Design with an introduction to the VHDL', Pearson Education, 2013.
- ✿ 3. Comer "Digital Logic & State Machine Design, Oxford, 2012.

#### ✿ REFERENCES

- ✿ 1. Mandal, "Digital Electronics Principles & Application, McGraw Hill Edu, 2013.
- ✿ 2. William Keitz, Digital Electronics-A Practical Approach with VHDL, Pearson, 2013.
- ✿ 3. Thomas L. Floyd, 'Digital Fundamentals', 11th edition, Pearson Education, 2015.
- ✿ 4. Charles H. Roth, Jr, Lizy Lizy Kurian John, 'Digital System Design using VHDL, Cengage, 2013.

## 8. Activity based learning

### ✿ Basic level –

Play with Binary Numbers using Binary Puzzles <http://www.binarypuzzle.com/>

Binary to create on-off pictures, Binary magic tricks

<https://www.digitaltechnologieshub.edu.au>

### ✿ Troubleshooting Logic Gates for

1. Open Input

2. Open Output conditions

### ✿ Logic Gate Testing using Multisim software

✿ Practice using open source DEEDS: <https://www.digitalelectronicsdeeds.com/>

## 9. Lecture Notes

### Table of Contents

#### ✿ UNIT I NUMBER SYSTEMS AND DIGITAL LOGIC FAMILIES

- ✿ Review of number systems
- ✿ Binary codes
- ✿ Error detection and correction codes
  - ✿ Parity and
  - ✿ Hamming code
- ✿ Digital Logic Families
  - ✿ RTL, DTL, TTL, ECL and MOS -operation, characteristics of digital logic family
  - ✿ Comparison of RTL, DTL, TTL, ECL and MOS families

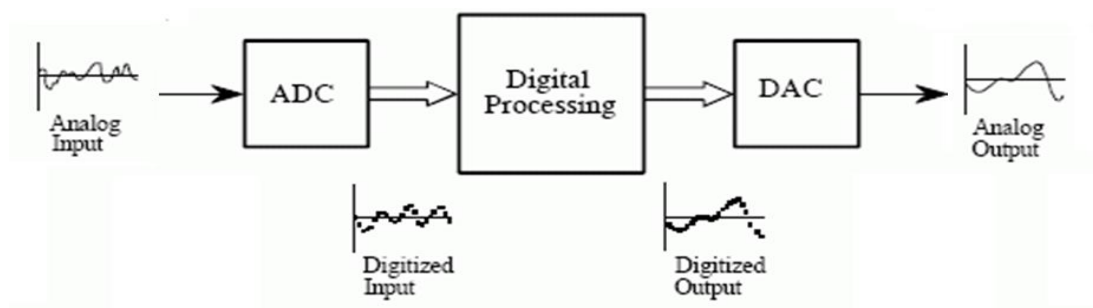
# Digital System Introduction

- ✿ We are in the digital era and digital systems are used in the industries, internet, medical treatment, space applications, communication, and all possible areas of automation.
- ✿ A few examples of digital systems start with a basic calculator to digital computer, digital telephones, digital media, digital money, digital control systems, home appliances, etc.
- ✿ A digital system is an interconnection of digital modules to give a required output or to do a specified operation.
- ✿ A basic knowledge of digital circuits and their logical function is required to understand the operation of each digital module.
- ✿ Digital System manipulates discrete elements of information represented in binary form.
- ✿ Discrete elements of information are represented with groups of bits called binary codes.
- ✿ The signals in most present-day electronic digital systems use just two discrete values 0 and 1 represented by a binary digit, called a bit.
- ✿ The natural signals generated by the nature are mostly continuous in nature like Temperature, Voltage, Current (can be converted to discrete)
- ✿ The processing in a digital system basically has the following sections:

Input: Analog-to-Digital (A/D) converter at the input end.

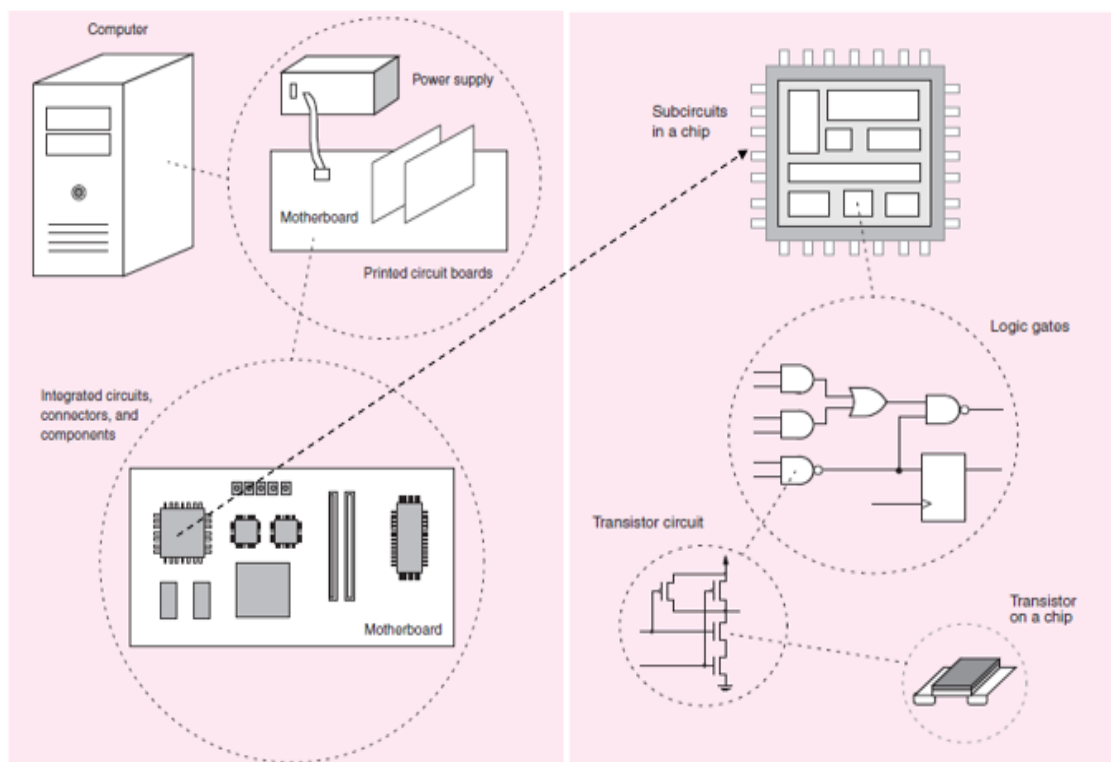
Processing: Done using a digital system.

Output: Digital-to-Analog (D/A) converter at the output end.

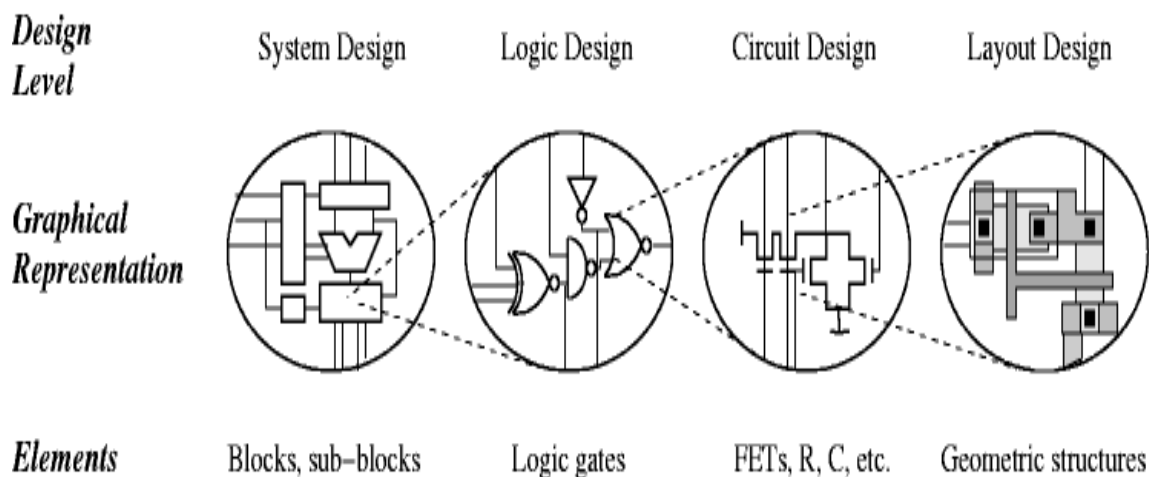


# The Hardware Levels of Digital System

- ❁ The levels of the hardware system from a CPU to a single transistor layout is shown below.
- ❁ The design level, logical representation and the elements that form the circuit at each level are also shown below.



Source: Thomas Floyd- Digital Fundamentals



# NUMBER SYSTEMS AND DIGITAL LOGIC FAMILIES

## ❁ 1. Review of number systems

### ❁ 1.1 Introduction:

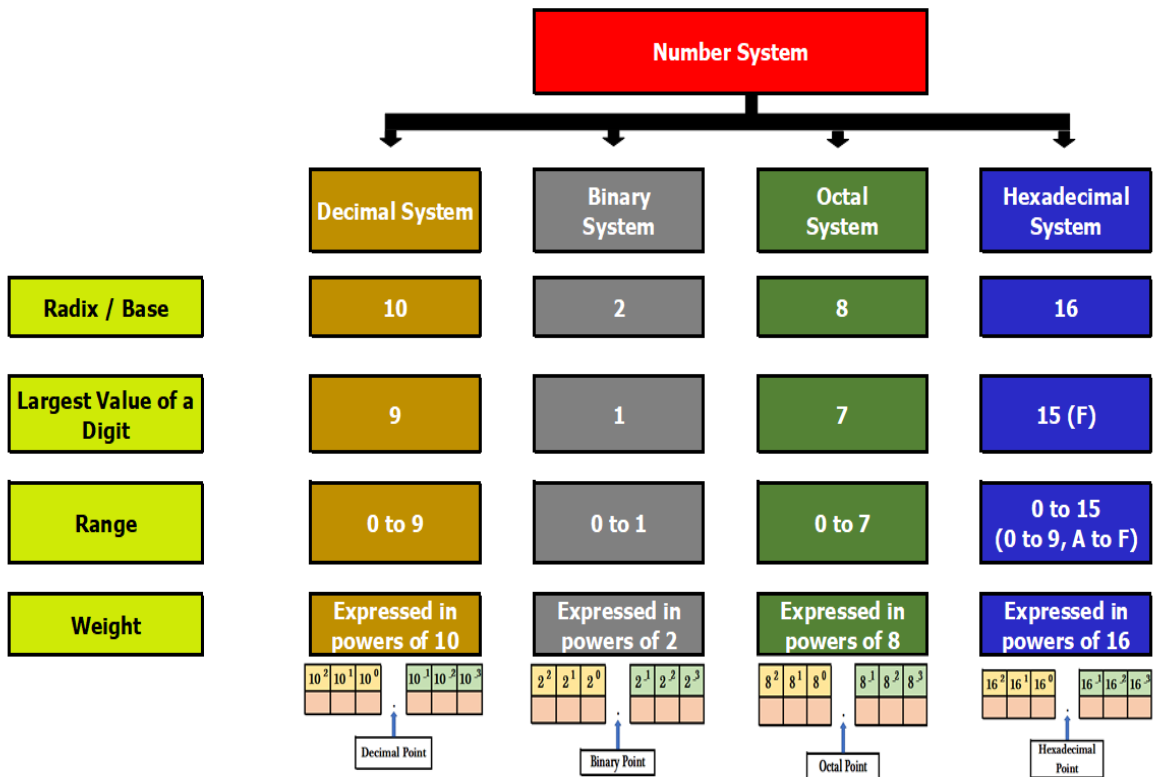
1. Number System defines a set of values used to represent a quantity.
2. Number System is a basis for counting various items.
3. Digits: A collection of various symbols in a number system is called digits.
4. Radix / Base: The number of unique digits (including Digit 0) used to represent the numbers in a positional number system is called radix or base of a number system.

(or)

The number of values that a digit (one character) can assume in a positional number system is called radix or base of a number system.

5. The largest value of a digit is always one less than radix / base of a number system.
6. Weight / Value: In a positional number system, the position of a digit with reference to the decimal point determines the weight / value.
  - a. Usually weight / value is expressed in powers of radix / base.
  - b. The sum of all digits multiplied by their weight gives the total number being represented.
7. Most Significant Digit / Bit: The leftmost digit / bit which has the greatest weight is called Most Significant Digit / Bit (MSD / MSB).
8. Least Significant Digit / Bit: The rightmost digit / bit which has the least weight is called Least Significant Digit / Bit (LSD / LSB).

## ❁ 1.2 Classification of Number System:



### ❁ 1.2.1 Decimal Number System:

1. Base: Base 10 System / Radix 10 System.
  - a. Number of values assumed by each digit: 10
  - b. Range: Uses ten digits: 0,1,2,3,4,5,6,7,8,9.
  - c. Each digit in the Decimal Number System will assume 10 different values from 0 to 9.
2. Largest Value of a digit: The largest value a digit can take is 9.
  - a. If larger values than 9 are needed, extra columns are added to the left. Each column value is now ten times the value of the column to its right. For example, the decimal value 47 is written 47 (4 tens + 7 ones).
3. Positional Weights: The positional weight of each digit is represented in the power of 10.

### ❁ 1.2.2 Binary Number System:

1. Base: Base 2 System / Radix 2 System.
  - a. Number of values assumed by each digit: 2
  - b. Range: Uses two digits: 0,1.
  - c. Each digit in the Binary Number System will assume 2 different values either 0 or 1.
2. Largest Value of a digit: The largest value a digit can take is 1.
  - a. If larger values than 1 are needed, extra columns are added to the left. Each column value is now two times the value of the column to its right. For example, the decimal value 2 is written 10 in binary (1 twos + 0 ones).
3. Positional Weights: The positional weight of each digit is represented in the power of 2.

### ❁ 1.2.3 Octal Number System:

1. Base: Base 8 System / Radix 8 System.
  - a. Number of values assumed by each digit: 8
  - b. Range: Uses ten digits: 0,1,2,3,4,5,6,7.
  - c. Each digit in the Octal Number System will assume 8 different values from 0 to 7.
2. Largest Value of a digit: The largest value a digit can take is 7.
  - a. If larger values than 7 are needed, extra columns are added to the left. Each column value is now eight times the value of the column to its right. For example, the decimal value 27 is written 33 in octal (3 eights + 3 ones).
3. Positional Weights: The positional weight of each digit is represented in the power of 8.



### ❁ 1.2.4 Hexadecimal Number System:

1. Base: Base 16 Number System / Radix 16 Number System.
  - a. Number of values assumed by each digit: 16
  - b. Range: Uses ten digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.
  - c. Each digit in the Octal Number System will assume 16 different values from 0 to 9, A to F.
2. Largest Value of a digit: The largest value a digit can take is F.
  - a. If larger values than F are needed, extra columns are added to the left. Each column value is now sixteen times the value of the column to its right. For example, the decimal value 68 is written as 44 in hexadecimal (4 sixteens + 4 ones).
3. Positional Weights: The positional weight of each digit is represented in the power of 16.

### ❁ 1.2.5 Other Number Systems:

Radix / Base	Characters / Range	Largest Value
2	0,1	1
3	0,1,2	2
4	0,1,2,3	3
5	0,1,2,3,4	4
6	0,1,2,3,4,5	5
7	0,1,2,3,4,5,6	6
8	0,1,2,3,4,5,6,7	7
9	0,1,2,3,4,5,6,7,8	8
10	0,1,2,3,4,5,6,7,8,9	9
11	0,1,2,3,4,5,6,7,8,9,A	A(10)
12	0,1,2,3,4,5,6,7,8,9,A,B	B(11)
13	0,1,2,3,4,5,6,7,8,9,A,B,C	C(12)
14	0,1,2,3,4,5,6,7,8,9,A,B,C,D	D(13)
15	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E	E(14)
16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F	F(15)

## ❁ 1.3 Conversion Between Number Systems:

### ❁ 1.3.1 Decimal to Other Number System Conversion:

#### ❁ Integer Part Conversion:

1. Successively divide by the base of the other number system.
2. Bottom to top approach.

#### ❁ Fraction Part Conversion:

1. Successively multiply by the base of the other number system.
2. Top to bottom approach.

### ❁ Decimal to Binary Conversion:

$$(450.2698)_{10} = ( )_2$$

#### Integer Part Conversion

2	450		
2	225	Remainder	0
2	112	Remainder	1
2	56	Remainder	0
2	28	Remainder	0
2	14	Remainder	0
2	7	Remainder	0
2	3	Remainder	1
2	1	Remainder	1
	0	Remainder	1

↑ LSB  
↓ MSB

#### Fraction Part Conversion

0.2698	X 2	= 0.5396	Integer Part Alone	0	MSB
Fraction Part Alone					
0.5396	X 2	= 1.0792	Integer Part Alone	1	
Fraction Part Alone					
0.0792	X 2	= 0.1584	Integer Part Alone	0	
Fraction Part Alone					
0.1584	X 2	= 0.3168	Integer Part Alone	0	LSB

**Ans:**

$$(450.2698)_{10} = (111000010.0100)_2$$

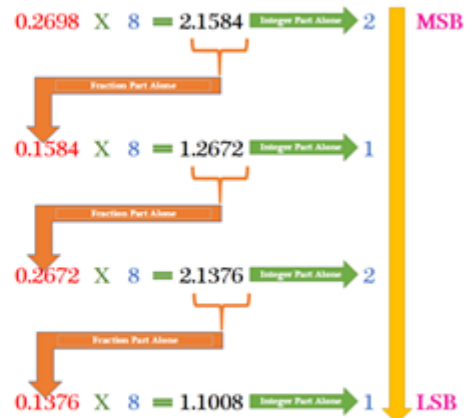
### ❁ Decimal to Octal Conversion:

$$(450.2698)_{10} = ( )_8$$

#### Integer Part Conversion



#### Fraction Part Conversion



**Ans:**

$$(450.2698)_{10} = (702.2121)_8$$

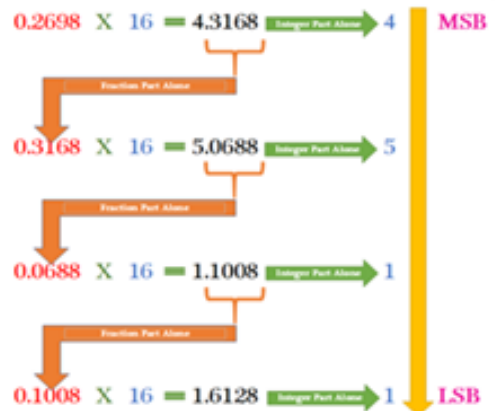
### ❁ Decimal to Hexadecimal Conversion:

$$(450.2698)_{10} = ( )_{16}$$

#### Integer Part Conversion



#### Fraction Part Conversion



**Ans:**

$$(450.2698)_{10} = (1C2.4511)_{16}$$

or 1C2.4511 H

## ❁ Decimal to Any Other Number System Conversion:

$$(450.2698)_{10} = ( )_6$$

### Integer Part Conversion



### Fraction Part Conversion



**Ans:**

$$(450.2698)_{10} = (2030.1341)_6$$

## ❁ Remember!!!!

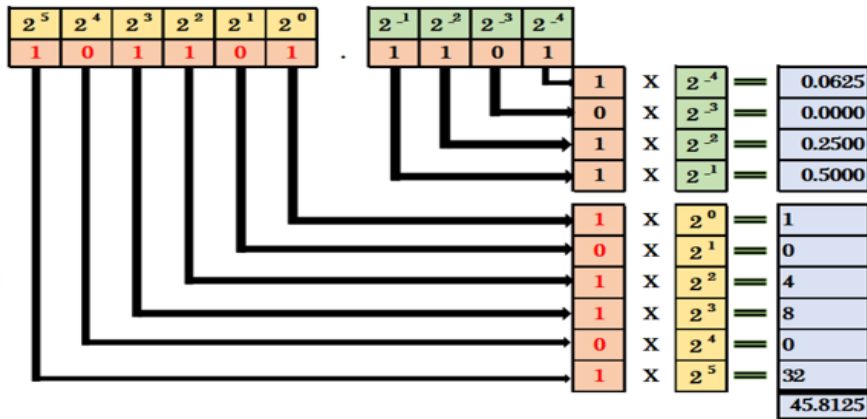
Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

### ❁ 1.3.2 Other Number System to Decimal Conversion:

❁ Express in powers of base of other number system.

#### ❁ Binary to Decimal Conversion:

$$(101101.1101)_2 = ( )_{10}$$

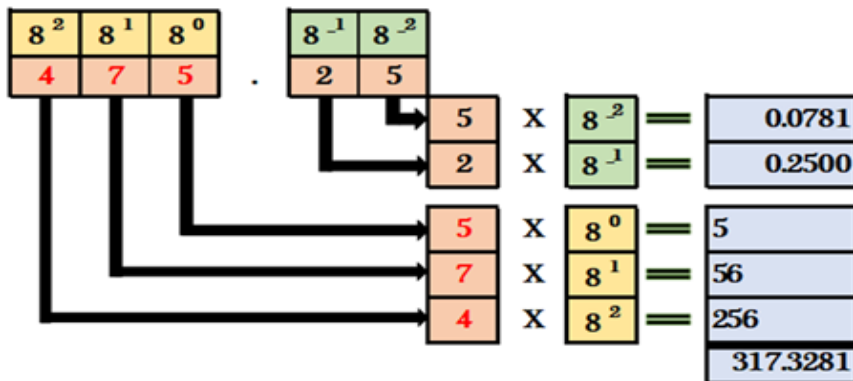


**Ans:**

$$(101101.1101)_2 = (45.8125)_{10}$$

#### ❁ Octal to Decimal Conversion:

$$(475.25)_8 = ( )_{10}$$

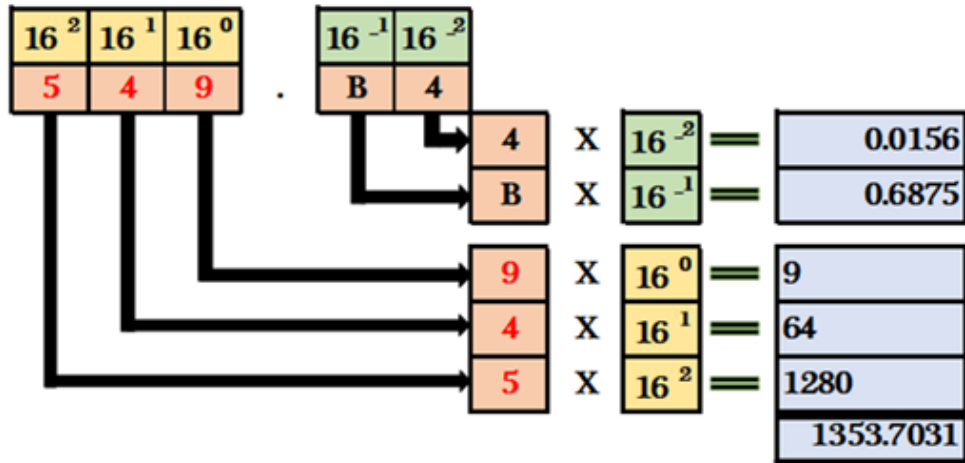


**Ans:**

$$(475.25)_8 = (317.3281)_{10}$$

✿ Hexadecimal to Decimal Conversion:

$$(549.B4)_{16} = ( )_{10}$$

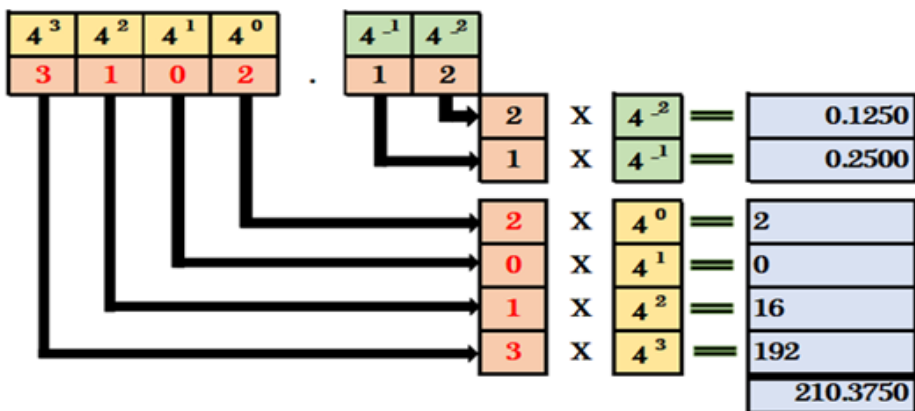


**Ans:**

$$(549.B4)_{16} = (1353.7031)_{10}$$

✿ Any Other Number System to Decimal Conversion:

$$(3102.12)_4 = ( )_{10}$$



**Ans:**

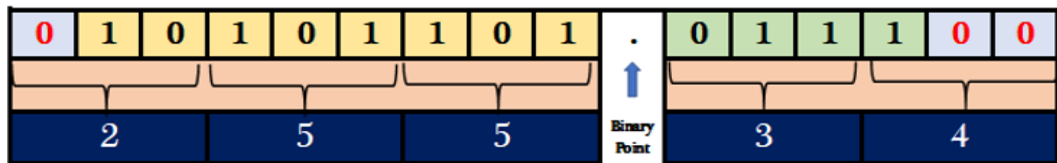
$$(3102.12)_4 = (210.3750)_{10}$$

### ❁ 1.3.3 Binary to Other Number System Conversion:

#### ❁ Binary to Octal Conversion:

❁ Group 3 binary bits and write the octal equivalent.

$$(10101101.0111)_2 = ( )_8$$



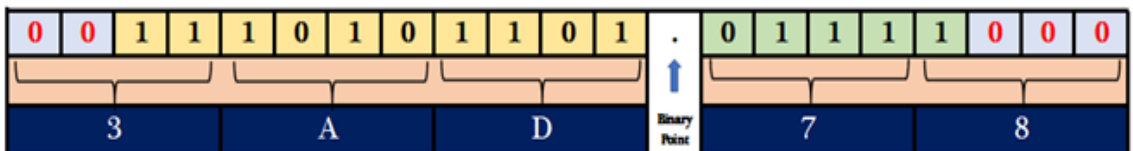
**Ans:**

$$(10101101.0111)_2 = (255.34)_8$$

#### ❁ Binary to Hexadecimal Conversion:

❁ Group 4 binary bits and write the hexadecimal equivalent.

$$(1110101101.01111)_2 = ( )_{16}$$



**Ans:**

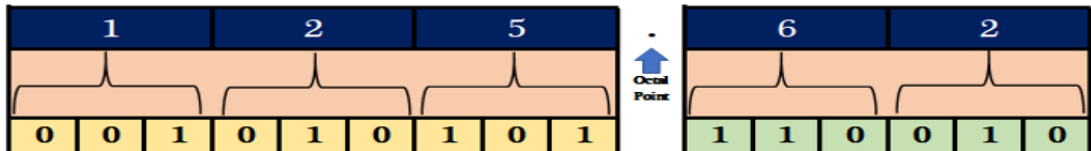
$$(1110101101.01111)_2 = (3AD.78)_{16} \text{ or } 3AD.78 \text{ H}$$

### ❁ 1.3.4 Octal to Other Number System Conversion:

#### ❁ Octal to Binary Conversion:

- ❁ Write the 3 bit binary equivalent for each of the octal digit.

$$(125.62)_8 = ( )_2$$



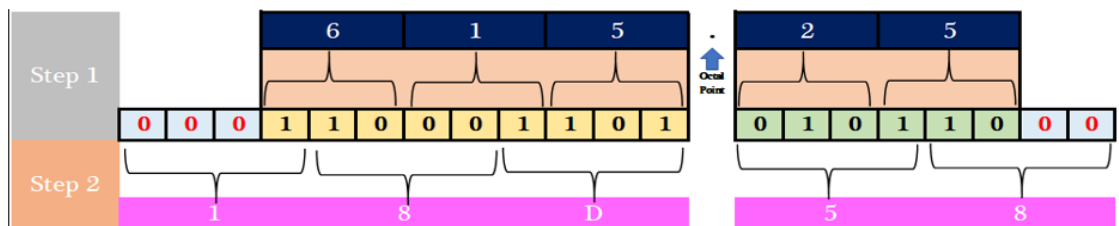
**Ans:**

$$(125.62)_8 = (001010101.110010)_2$$

#### ❁ Octal to Hexadecimal Conversion:

- ❁ Step 1: Octal to Binary Conversion.
- ❁ Step 2: Binary to Hexadecimal Conversion.

$$(615.25)_8 = ( )_{16}$$



**Ans:**

$$(615.25)_8 = (18D.58)_{16}$$

or 18D.58 H

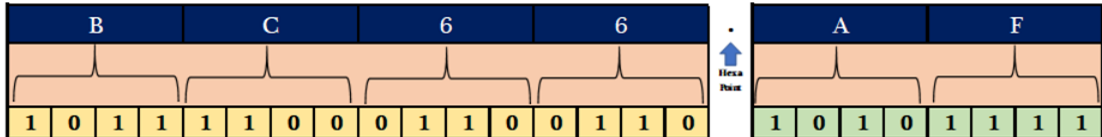


### ❁ 1.3.4 Hexadecimal to Other Number System Conversion:

#### ❁ Hexadecimal to Binary Conversion:

❁ Write the 4 bit binary equivalent for each of the hexadecimal digit.

$$(BC66.AF)_{16} = ( )_2$$



**Ans:**

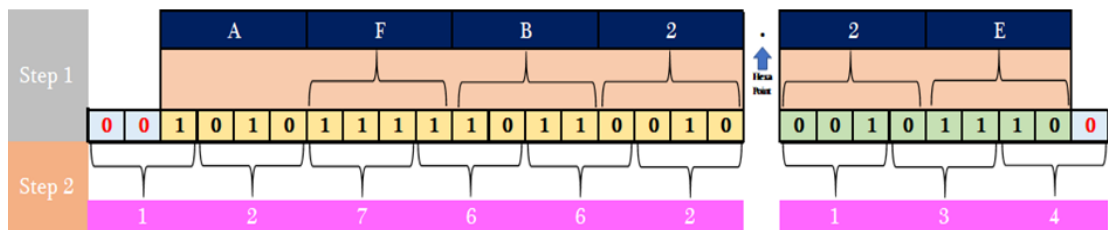
$$(BC66.AF)_{16} = (1011110001100110.10101111)_2$$

#### ❁ Hexadecimal to Octal Conversion:

❁ Step 1: Hexadecimal to Binary Conversion.

❁ Step 2: Binary to Octal Conversion.

$$(AFB2.2E)_{16} = ( )_8$$



**Ans:**

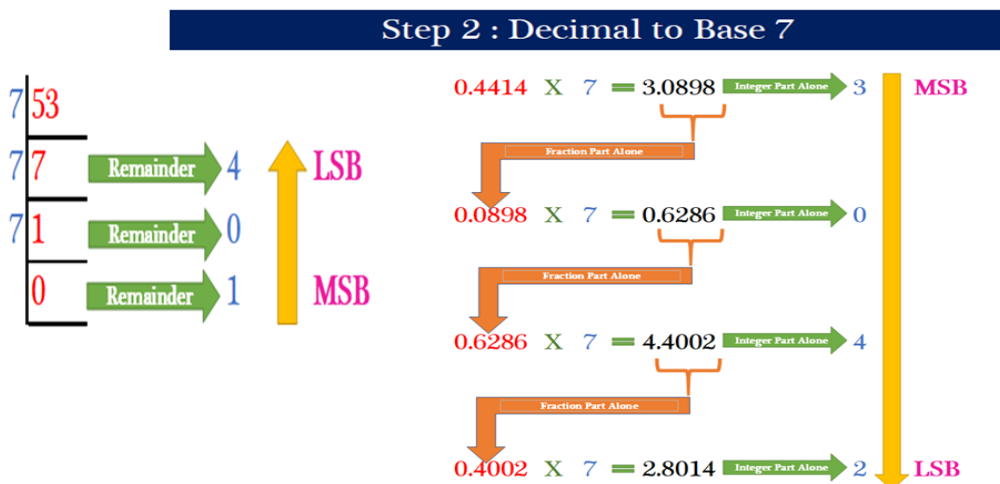
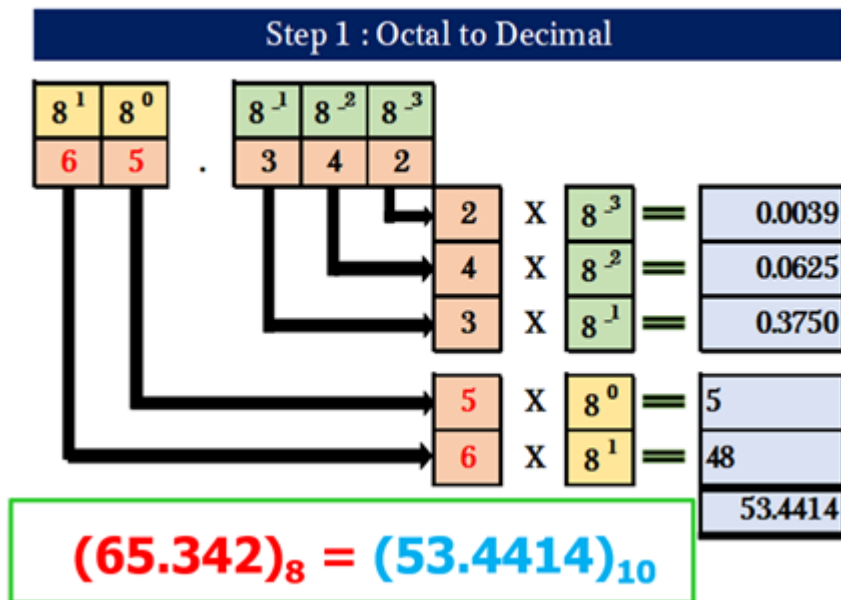
$$(AFB2.2E)_{16} = (56756.10)_8$$

### ❁ 1.3.4 Any Other Number System 1 to Any Other Number

#### System 2 Conversion:

- ❁ Step 1: Any Other Number System 1 to Decimal Conversion.
- ❁ Step 2: Decimal to Any Other Number System 2 Conversion.

$$(65.342)_8 = ( )_7$$



**Ans:**

$$(65.342)_8 = (104.3042)_7$$

## 2. Representation of Signed Number

### ✿ 2.1 Signed magnitude

- ✿ For Positive number - sign bit will be 0
- ✿ For negative number – sign bit will be 1
- ✿ The various ways of representing negative numbers are:
  - ✿ 1.sign magnitude
  - ✿ 2. 1's complement
  - ✿ 3. 2's complement
- ✿ Sign magnitude – 1 101001
- ✿ 1's complement – 1 010110
- ✿ 2's complement - 1 010111

### ✿ 2.2 1'S COMPLEMENT SUBTRACTION:

- ✿ Subtraction of smallest number from largest number

Ex 1:      43-21 = 22

43 in Binary 101011

21 in Binary 010101

1's complement of 21 = 101010

101011

101010 +

=1010101 = 010101 + 1 (End around Carry)

---

= 010110

- ✿ If carry is present. The answer is positive.
- ✿ For 1's complement you have to do End around carry.
- ✿ Ans = +22

✿ Subtraction of largest number from smallest number.

Ex 2:      21-43

Soln:

21 in Binary 010101

43 in Binary 101011

✿ 1's complement of 43 = 010100

010101

010100 +

---

=          101001

✿ There is no carry. Answer is negative.

✿ Ans = -(1's complement 101001)

✿          =-010110 = - 22

### ✿ 2.3 2'S COMPLEMENT SUBTRACTION:

✿ Subtraction of smallest number from largest number using 2's complement.

Ex:1)      43-21 = 22

43 in Binary 101011

21 in Binary 010101

✿ 2's complement of 21 = 1's complement of 21 + 1

= 101010 +1 =101011

101010

1 +

= 101011

(1+1+1 =10+1 =11)

101011    43 in Binary

101011+2's complement of 21

= 1 0 1 0 1 1 0    (Ignore Carry)

---

0 1 0 1 1 0

✿ If carry is present. The answer is positive.

✿ For 2's complement you have to do Ignore carry.

✿ Ans =+22

✿ Subtraction of largest number from smallest number using 2's complement.

✿ Ex: 2) 21-43

✿ Soln:

21 in Binary 010101

43 in Binary 101011

2's complement of 43 = 1's complement of 43

+1

= 010100 +1 =010101

010101 21 in Binary

010101 + 2's complement of 43

---

✿ = 101010

✿ There is no carry. Answer is negative.

✿ Ans = -(2's complement 101010) 010101 +1 = 010110

✿ =-(010110) =-22

### 3. Binary Codes

- ✿ In the coding, when numbers, letters or words are represented by a specific group of symbols, it is said that the number, letter or word is being encoded. The group of symbols is called as a code.
- ✿ The digital data is represented, stored and transmitted as group of binary bits. This group is also called as **binary code**. The binary code is represented by number as well as alphanumeric letter.

#### ✿ Advantages of Binary Code

Binary codes are suitable for computer applications.

Binary codes are suitable for the digital communications.

Binary codes make the analysis and designing of digital circuits easier since only 0 & 1 are being used.

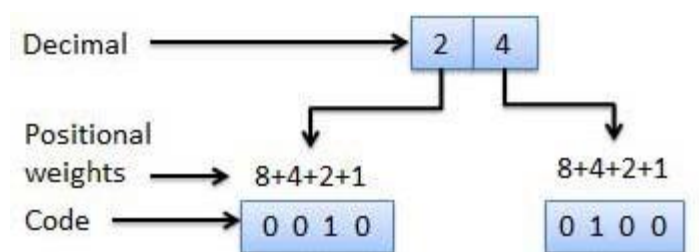
#### ✿ 3.1 Classification of binary codes

The codes are broadly categorized into following four categories.

- ✿ Weighted Codes
- ✿ Non-Weighted Codes
- ✿ Binary Coded Decimal Code
- ✿ Alphanumeric Codes
- ✿ Error Detecting Codes
- ✿ Error Correcting Codes

##### ✿ 3.1.1 Weighted Codes

Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the number represents a specific weight. Several systems of the codes are used to express the decimal digits 0 through 9. In these codes each decimal digit is represented by a group of four bits.



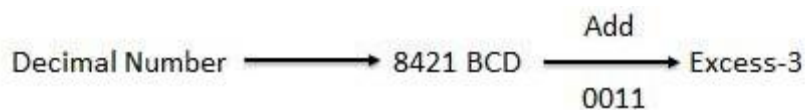
### ❁ 3.1.2 Non-Weighted Codes

In this type of binary codes, the positional weights are not assigned.

The examples of non-weighted codes are Excess-3 code and Gray code.

#### ❁ Excess-3 code

The Excess-3 code is also called as XS-3 code. It is non-weighted code used to express decimal numbers. The Excess-3 code words are derived from the 8421 BCD code words adding 00112 or 3 10 to each code word in 8421. The excess-3 codes are obtained as follows –



Decimal	BCD				Excess-3			
	8	4	2	1	BCD + 0011			
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

#### ❁ Gray Code

It is the non-weighted code and it is not arithmetic codes. That means there are no specific weights assigned to the bit position. It has a very special feature that, only one bit will change each time the decimal number is incremented as shown in fig. As only one bit changes at a time, the gray code is called as a unit distance code. The gray code is a cyclic code. Gray code cannot be used for arithmetic operation.

### ❁ Application of Gray code

Gray code is popularly used in the shaft position encoders.

A shaft position encoder produces a code word which represents the angular position of the shaft

Decimal	BCD	Gray
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1

### ❁ 3.3 Binary Coded Decimal BCD code

In this code each decimal digit is represented by a 4-bit binary number. BCD is a way to express each of the decimal digits with a binary code. In the BCD, with four bits we can represent sixteen numbers 0000 to 1111. But in BCD code only first ten of these are used 0000 to 1001. The remaining six code combinations i.e. 1010 to 1111 are invalid in BCD

Decimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001



### ❁ **Advantages of BCD Codes**

It is very similar to decimal system.

We need to remember binary equivalent of decimal numbers 0 to 9 only.

### ❁ **Disadvantages of BCD Codes**

The addition and subtraction of BCD have different rules. The BCD arithmetic is little more complicated.

BCD needs more number of bits than binary to represent the decimal number. So BCD is less efficient than binary.

### ❁ **3.4 Alphanumeric codes**

❁ A binary digit or bit can represent only two symbols as it has only two states '0' or '1'. But this is not enough for communication between two computers because there we need many more symbols for communication. These symbols are required to represent 26 alphabets with capital and small letters, numbers from 0 to 9, punctuation marks and other symbols.

❁ The alphanumeric codes are the codes that represent numbers and alphabetic characters. Mostly such codes also represent other characters such as symbol and various instructions necessary for conveying information. An alphanumeric code should at least represent 10 digits and 26 letters of alphabet i.e. total 36 items. The following three alphanumeric codes are very commonly used for the data representation.

- ❁ American Standard Code for Information Interchange ASCII.

- ❁ Extended Binary Coded Decimal Interchange Code EBCDIC.

- ❁ Five bit Baudot Code.

❁ ASCII code is a 7-bit code whereas EBCDIC is an 8-bit code. ASCII code is more commonly used worldwide while EBCDIC is used primarily in large IBM computers.

## 3.5 Error Detecting and Correcting Codes

### ✿ Error Detection

- ✿ Error detection means to decide whether the received data is correct or not without having a copy of the original message.
- ✿ Error detection uses the concept of redundancy, which means adding extra bits for detecting errors at the destination.

### ✿ Error Codes

- ✿ Parity method of error detection
- ✿ Hamming code for error detection and correction

#### ✿ 3.5.1 Parity Method for Error Detection

- ✿ Any group of bit contain either an even or an odd number of 1s.
- ✿ A parity bit is attached to a group of bits to make the total number of 1s in a group always even or always odd.
- ✿ An even parity bit makes the total number of 1s even, and an odd parity bit makes the total odd.
- ✿ The parity bit can be attached to the code at either the beginning or the end, depending on system design.
- ✿ The total number of 1s, including the parity bit, is always even for even parity and always odd for odd parity.

#### ✿ 3.5.2 Hamming Code

- ✿ In the Hamming code,  $P$  parity bits are added to an  $m$ -bit data word, forming a new word of  $m + P$  bits.
- ✿ The bit positions are numbered in sequence from 1 to  $m + P$ .
- ✿ Those positions numbered as a power of 2 are reserved for the parity bits.
- ✿ The remaining bits are the data bits.
- ✿ The code can be used with words of any length.

### ❁ 3.5.2.1 Single-bit error detection

❁ To correct an error, the receiver reverses the value of the altered bit. To do so, it must know which bit is in error.

❁ Number of redundancy (parity) bits needed

❁ Let data bits =  $m$

❁ Parity bits =  $P$

❁ Total message sent =  $m + p$

❁ The value of  $P$  must satisfy the following relation:

$$2^P \geq m + P + 1$$

❁ The parity bits,  $P_1$ ,  $P_2$ ,  $P_4$ , and  $P_8$ , are in positions 1, 2, 4, ... respectively ( $2^0$ ,  $2^1$ ,  $2^2$ , ...)

❁ The data bits are in the remaining positions.

❁ Each parity bit is calculated as follows:

❁  $P_1$  = XOR of bits (3, 5, 7, 9, 11)

❁  $P_2$  = XOR of bits (3, 6, 7, 10, 11)

❁  $P_4$  = XOR of bits (5, 6, 7, 12)

❁  $P_8$  = XOR of bits (9, 10, 11, 12)

❁ Exclusive-OR operation

❁ =1 for an odd number of 1's (Error)

❁ =0 for an even number of 1's (No Error)

❁ Each parity bit is set so that the total number of 1's in the checked positions, including the parity bit, is always even

### ✿ 3.5.2.2 Single-Bit Error Correction

#### ✿ Error Correction

✿ It can be handled in two ways:

1)Receiver can have the sender retransmit the entire data unit.

2)Receiver can use an error-correcting code, which automatically corrects certain errors.

✿ To correct an error, the receiver reverses the value of the altered bit.

✿ To do so, it must know which bit is in error.

✿ Number of redundancy (parity) bits needed

✿ Let data bits =  $m$

✿ Parity bits =  $P$

✿ Total message sent =  $m + P$

✿ The value of  $P$  must satisfy the following relation:

$$2^P \geq m + P + 1$$

✿ Each parity bit is calculated as follows:

✿  $P_1$  = XOR of bits (3, 5, 7, 9, 11)

✿  $P_2$  = XOR of bits (3, 6, 7, 10, 11)

✿  $P_4$  = XOR of bits (5, 6, 7, 12)

✿  $P_8$  = XOR of bits (9, 10, 11, 12)

✿ The check bits are evaluated as follows:

✿  $C_1$  = XOR of bits (1, 3, 5, 7, 9, 11)

✿  $C_2$  = XOR of bits (2, 3, 6, 7, 10, 11)

✿  $C_4$  = XOR of bits (4, 5, 6, 7, 12)

✿  $C_8$  = XOR of bits (8, 9, 10, 11, 12)

(XOR is done along with the Parity bit received)

# Examples

## ⌘ Hamming Error Detecting and Correcting Codes

1.Encode the data bit 1101 using Hamming Error correcting and Detecting codes.

⌘ Step 1 : To find the number of Parity bits required.

$2^p \geq m + P + 1$

$m = 4$

Let  $p = 2$      $2^2 = 4$                        $m + P + 1 = 4 + 2 + 1 = 7$

Let  $P = 3$      $2^3 = 8$                        $m + P + 1 = 4 + 3 + 1 = 8$

Now  $2^p = m + P + 1$

3 –parity bits are enough to transmit 4-bit data.

Total bits =  $m + P = 4 + 3 = 7$

⌘ Step2: Construct a Bit location table:

Parity Bit =  $2^0, 2^1, 2^2$

	P1	P2	D1	P3	D2	D3	D4
Bit Position	1	2	3	4	5	6	7
Binary Position Number	001	010	011	100	101	110	111
Data Bits			1		1	0	1
Parity Bits	1	0		0			
7-bit Hamming code	1	0	1	0	1	0	1

2. A hamming code is received as 1010111. Determine the correct code when even parity is used.

✿ Step1: Construct a Bit location table:

✿ Parity Bit =  $2^0, 2^1, 2^2$

	P1	P2	D1	P3	D2	D3	D4
Bit Position	1	2	3	4	5	6	7
Binary Position Number	001	010	011	100	101	110	111
Received code	1	0	1	0	1	1	1
Corrected code	1	0	1	0	1	0	1

✿ Step 2: Find the Error Bit

Checking for Even Parity

E1 (P1) ----- 1, 3, 5, 7----- 1 1 1 1 ----- No Error – 0 (LSB)

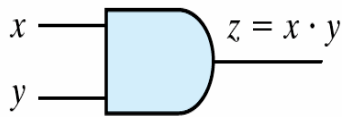
E2 (P2) ----- 2, 3, 6, 7----- 0 1 1 1 ----- Error ---1

E3 (P3) ----- 4,5,6,7----- 0 1 1 1 ----- Error ---1 (MSB)

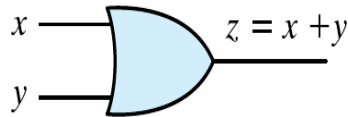
✿ Error Bit -----110

## 4.Binary Logic – The Logic Gates

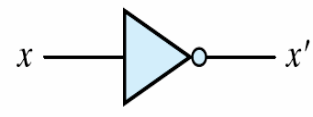
### ❁ Graphic Symbols and Input-Output Signals for Logic gates:



(a) Two-input AND gate



(b) Two-input OR gate



(c) NOT gate or inverter

❁ The OR gate IC which has 4 (Quad) OR gates is shown below.

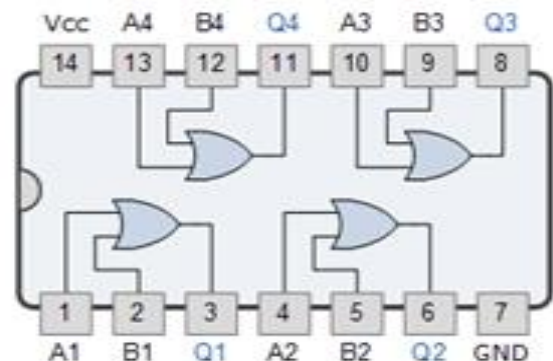
❁ The OR gate in its circuit form has 2 NPN transistors as the logic switches

❁ The NPN transistor is also shown separately

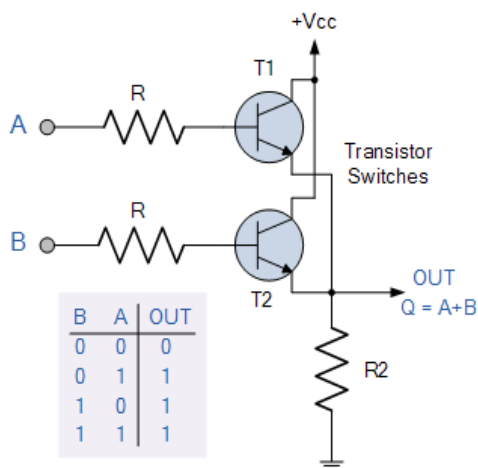
### ❁ 2-Input Logic OR Gate

Symbol	Truth Table		
 2-input OR Gate	B	A	Q
	0	0	0
	0	1	1
	1	0	1
	1	1	1
Boolean Expression $Q = A + B$		Read as A OR B gives Q	

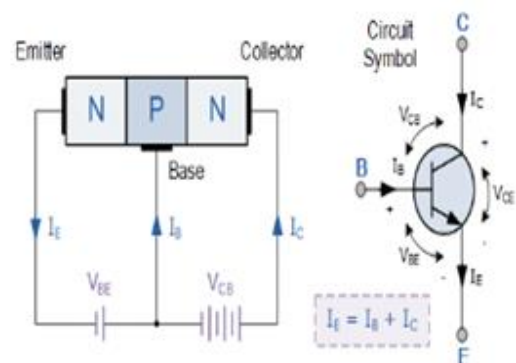
### IC 7432 Quad 2-input Logic OR Gate



### ❁ 2-input Transistor OR Gate



### NPN structure of a Transistor in OR Gate



## 5. DIGITAL LOGIC FAMILIES

### ✿ Integrated circuits

Integrated Circuits are used for producing several different circuit configurations and production technologies. The semiconductor chip consists of electronic components used for constructing circuits. Integrated circuits are classified into

- ✿ Linear Integrated Circuits

- ✿ Digital Integrated Circuits

Both operate with continuous and discrete signals respectively and are used to construct various electronic circuits.

There are different levels of Integration, based on the number of logic gates in a single IC package.

#### ✿ Small scale Integration (SSI)

These IC's contain fewer logic gates (0 to 10).The input and output pins can be directly connected to the pins in the package.

#### ✿ Medium Scale Integration (MSI)

These IC's have approximately around 10 to 1000 gates in one package, which performs some specific functions.

(Ex: adders, multiplexers)

#### ✿ Large Scale Integration (LSI)

These IC's contain thousands of gates in a single package.

(Ex: processors, memory chips and programmable logic devices)

#### ✿ Very Large Scale Integrated Devices (VLSI)

These IC's contain hundreds of thousands of gates in a single package.

(Ex: memory arrays, computer chips)



## ❁ Classification of Logic Families

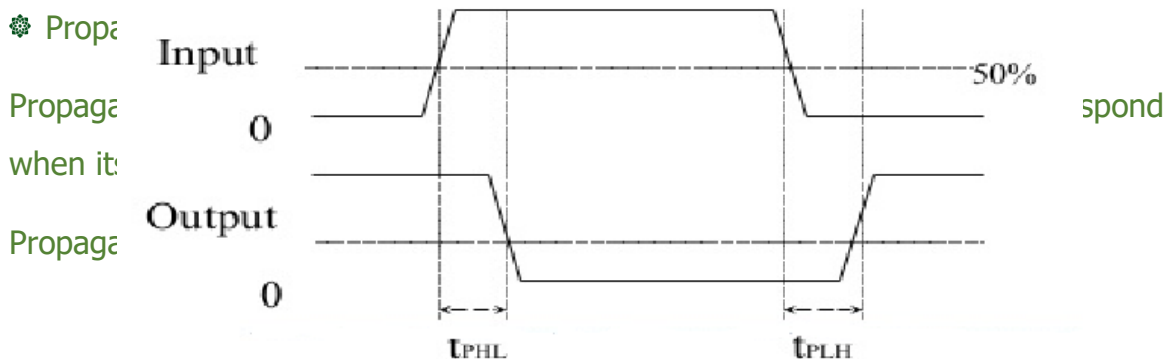
Based on the circuit technology digital IC's are classified into various types

- ❁ RTL (Resistor Transistor Logic)
- ❁ DTL (Diode Transistor Logic)
- ❁ TTL (Transistor Transistor Logic)
- ❁ ECL (Emitter Coupled Logic)
- ❁ MOS (Metal Oxide Semiconductor Logic)

## ❁ Characteristics of Logic Families

The following are the important characteristics of digital IC's

- ❁ Propagation Delay (or) Operating Speed
- ❁ Voltage and Current parameters
- ❁ Power Dissipation
- ❁ Fan-in
- ❁ Fan-out
- ❁ Noise Margin
- ❁ Operating Temperature
- ❁ Power Supply Requirements



$t_{PLH}$  : Propagation delay time from low level(0) to high level(1)

$t_{PHL}$  : Propagation delay time from high level(1) to low level(0)

The delay in output is a measure of relative speed of logic circuits.

Average propagation delay  $P_{D\text{ avg}} = (t_{PLH} + t_{PHL})/2$

#### ✿ Voltage and Current Parameters:

Digital logic gates have a certain range of voltage and current levels

corresponding to 0(low level)) and 1(high level)

#### ✿ Voltage and current levels:

High level input voltage and current: The minimum input voltage recognised as logic 1(high level) by the logic gates (2V – 3V range) and the corresponding current is high level input current.

Low level input voltage and current: The maximum input voltage recognised as logic 0(low level) by the logic gates (around 0.8V) and the corresponding current is low level input current.

High level output voltage and current: The minimum voltage available at the output corresponding to logic 1 and the corresponding current is high level output current.

Low level output voltage and current: The minimum voltage available at the output corresponding to logic 0 and the corresponding current is low level output current.

#### ✿ Power Dissipation:

Power dissipation is the measure of power consumed by the logic gate when fully driven by all its inputs. It is expressed in milliwatts or nanowatts.

Average power dissipation  $P_{dc\text{ avg}} = V_{cc} \times I_{avg}$

$V_{cc}$  - DC supply voltage

$I_{avg}$  – Average current taken from the supply

#### ✿ Fan-in:

Fan-in is the number of inputs available in a gate

#### ✿ Fan-out:

Fan-out is the number of similar logic gates that the output of a gate can drive without affecting the normal operation.

#### ✿ Noise Margin:

Noise margin is the maximum external noise voltage added to the input signal that does not cause any undesirable change in the circuit operation.

#### ✿ Operating Temperature:

All integrated circuits are semiconductor devices sensitive to temperature

- ✿ Operating range

- ✿ 0°C to +70° for consumer applications

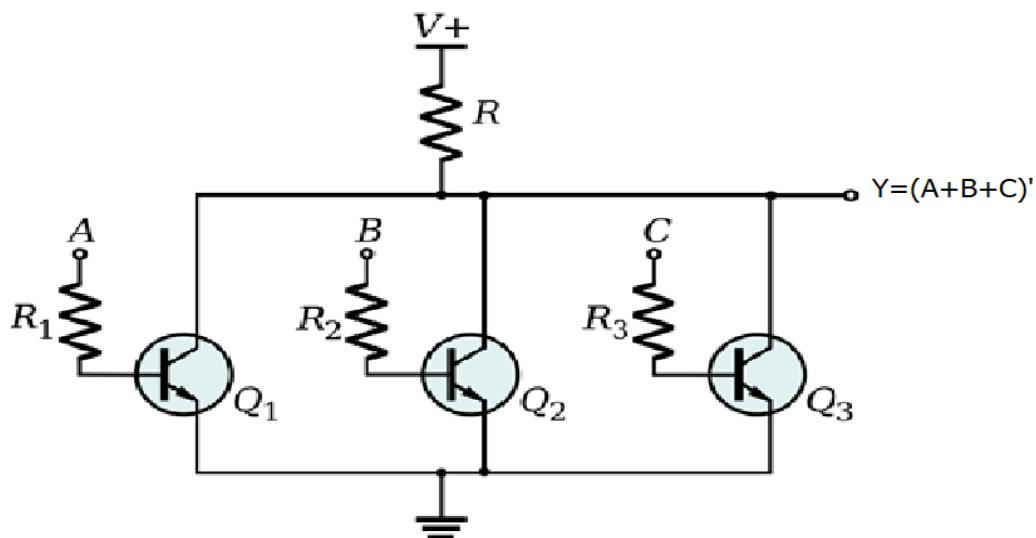
- ✿ -55° to +125°C for military applications

#### ✿ Power Supply Requirements:

The amount of power required by the IC.

## ❁ 5.1 RESISTOR TRANSISTOR LOGIC (RTL)

❁ The following diagram shows the Resistor Transistor Logic (RTL) of NOR logic function.



❁ Basic diagram of RTL NOR consists of transistor and resistors.

Here it is three input NOR gates logic diagram using RTL [i.e., A, B, C].

This follows the NOR gates truth table in its operation. i.e., whenever any one of the input is “HIGH” then it produce low output (or) all inputs are low, it produces “Low” output. This is similar to NOR logic truth table shown below.

INPUT		OUTPUT
A	B	$Y=(A+B)'$
0	0	1
0	1	0
1	0	0
1	1	0

### ❁ Operation:

When all inputs are zero (or) low, then output  $Q = 0$ .

Since, all these transistors Q1, Q2, Q3 are in OFF condition; its collector output is high.

When any one of the input is high (or) all inputs are high, then its corresponding transistor is going to ON condition. Also, it is connected with ground and collector potential which is approximately zero.

Anyhow, the base current is practically independent of the emitter junction characteristic. When the resistors increase the input resistance and reduce the switching speed of the circuit. This reduces the rise and fall times of any input pulse.

In practice, this approach to increase the speed of an RTL is to connect a capacitor called a speed-up capacitor which is parallel to resistance connected in base.

### ❁ Operation Table

Inputs			Transistor Status			Output ( $Y=A+B+C$ )'
A	B	C	Q1	Q2	Q3	
0	0	0	OFF	OFF	OFF	1(HIGH)
1	0 or 1	0 or 1	ON	ON or OFF	ON or OFF	0(LOW)

### ❁ Drawbacks:

In this logic family, some disadvantages are there, they are:

- (i) It reduces current-hogging by load transistors, which is purely because of mismatch of junction voltages. Hence it permits large fan-out.
- (ii) One more problem is that load transistor in a RTL gate are driven heavily into saturation. Hence it results in long-turn-off delays.

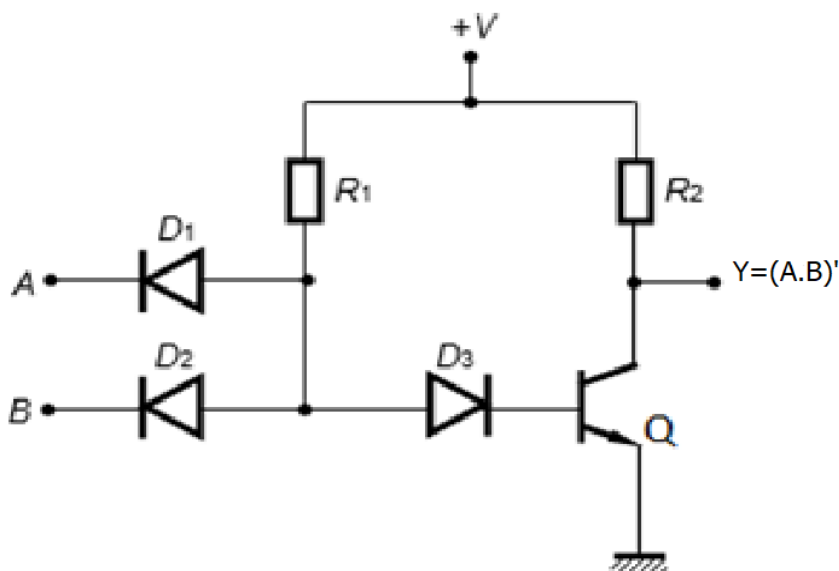
### ❁ Characteristics of RTL Family

1. Speed of operation is low, i.e., Propagation delay is of the order 500 ns. Hence it cannot operate the system speed above 4 MHz.
2. For switching delay of 50 ns, the fan-in is 4 (or) 5 and fan-out is 4.
3. Because of Base resistor in transistor, the power dissipation is more. This can be reduced by introducing DCTL (Direct coupled transistor logic).
4. It is highly sensitive to temperature.
5. Poor in noise immunity

### ❁ 5.2 DIODE TRANSISTOR LOGIC (DTL)

This DTL logic family reduces the problem of decreasing output voltage with increasing load.

The following diagram shows the DTL NAND logic circuit using diode and transistor.



✿ A and B are the inputs.

✿ D1 D2 forms AND equivalent circuit and transistor (Q) acts as a inverter. Therefore, the combinations of AND and NOT gates forms a logical NAND circuit hence it follows the following NAND truth table.

INPUT		OUTPUT
A	B	$Y=(A.B)'$
0	0	1
0	1	1
1	0	1
1	1	0

✿ Operation:

As per the above circuit diagram, the operation is as follows:

When  $A = B = 1$

Diodes D1 D2 are in reverse biased conditions [i.e., acts as open circuit].

Therefore D3 conduct. Hence the transistor base gets current flow and which is turned ON.

Q output in low cut-off

When  $A = B = 0$  (or)  $A = 0$  and  $B = *$  (0 or 1)

When all inputs are zero, the D1 and D2 is in ON condition. Hence there is no input current to base of the transistor. Q, hence it is in OFF condition. Thereby the output in collector terminal of transistor Q is high, called saturated state.

Similarly if any one of the input is low (0) that makes the above operation. So output is high (1)

Therefore, the final expression is  $Y = (A.B)'$  Then above operation is tabulated by using functional operation table

### ❁ Operation table

Inputs		Device Status				Output ( $Y=A.B$ )'
A	B	D1	D2	D3	Q	
0	0	ON	ON	OFF	OFF	1(HIGH)
0	1	ON	OFF	OFF	OFF	1(HIGH)
1	0	OFF	ON	OFF	OFF	1(HIGH)
1	1	OFF	OFF	ON	ON	0(LOW)

### ❁ Characteristics of DTL Family

#### ❁ Propagation Delay:

The turn-off delay is considerably more than the turn-on-delay. Hence propagation delay is 25 ns.

#### ❁ Fan-in and Fan-out.

Fan-in is less than 8.

Fan-out is high i.e., upto 8.

#### ❁ Noise immunity:

Noise margin is high. This is due to the additional diodes.

❁ Anyhow, whatever the drawbacks, can be reduced (or) improved in TTL family.

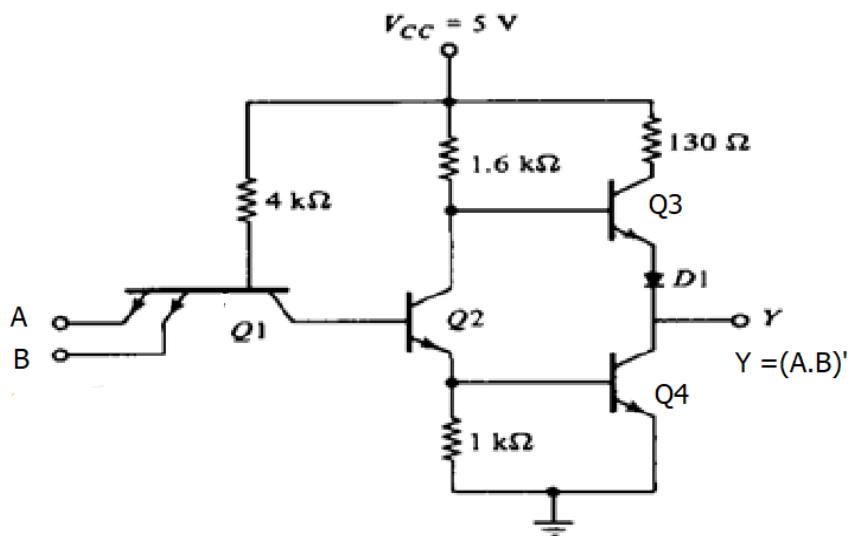
## ❁ 5.3 TRANSISTOR TRANSISTOR LOGIC (TTL)

The speed limitation of DTL is overcome by TTL family. It is the commonly used saturating family and hence operating speed is high.

Basic gate for TTL logic is NAND gate

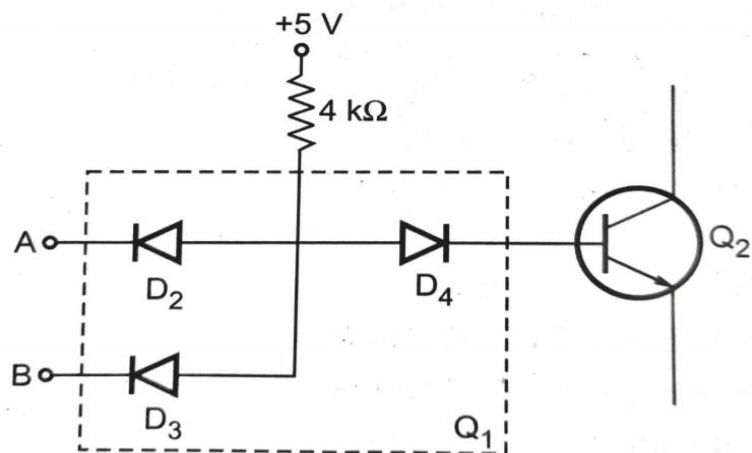


## ❁ 2-Input TTL NAND Gate

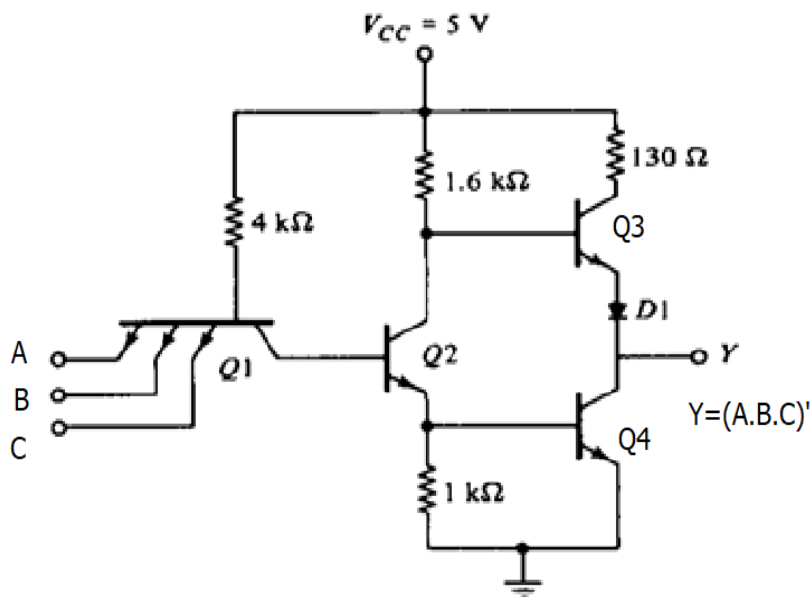


❁ The figure shows the circuit diagram of 2-input NAND gate. Its input structure consists of multiple-emitter transistor and output structure consists of totem-pole output. Here, Q1 is an NPN transistor having two emitters, one for each input to the gate. Although this circuit looks complex, we can simplify its analysis by using the diode equivalent of the multiple-emitter transistor Q1, as shown in figure. Diodes D2 and D3 represent the two E-B junctions of Q1 and D4 is the collector-base (C-B) junction.

## ❁ Diode equivalent



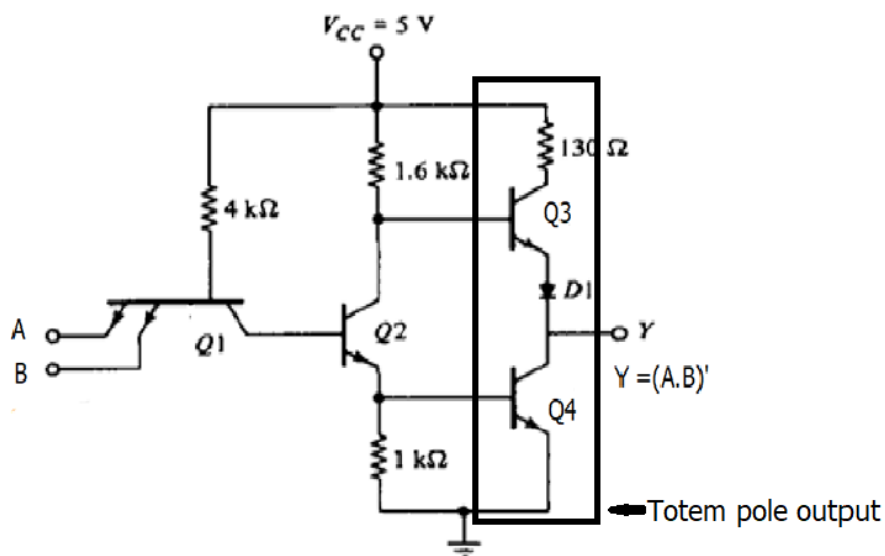
- ❁ The input voltages A and B are either LOW (ideally grounded) or HIGH (ideally + 5 volts). If either A or B or both are low, the corresponding diode conducts and the base of Q1 is pulled down to approximately 0.7 V. This reduces the base voltage of Q2 to almost zero. Therefore, Q2 cuts off. With Q2 open, Q4 goes into cut-off and the Q3 base is pulled HIGH. Since Q3 acts as an emitter follower, the Y output is pulled up to a HIGH voltage. On the other hand, when A and B both are HIGH, the emitter diode of Q1 are reversed biased making them off. This causes the collector diode D4 to go into forward conduction. This forces Q2 base to go HIGH. In turn, Q4 goes into saturation producing a low output.
- ❁ Without diode D1 in the circuit, Q3 will conduct slightly when the output is low. To prevent this diode is used; its voltage keeps the base-emitter diode of Q3 reverse biased only Q4 conducts when output is low.
- ❁ 3-input TTL NAND Gate



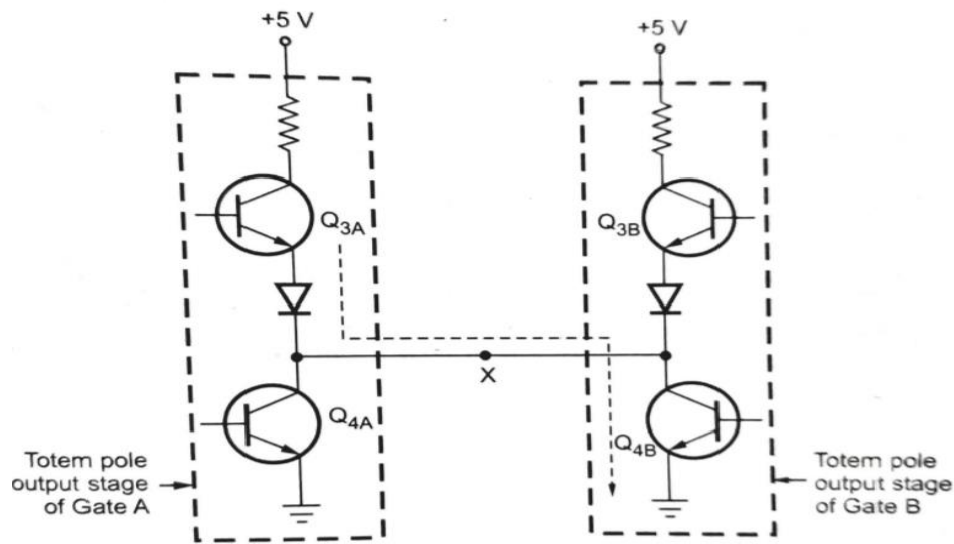
✿ The figure shows the three input TTL NAND gate. The operation of three input TTL NAND is same as that of two output TTL NAND gate except that is Q1 (NPN) transistor has three emitters instead of two. For three input NAND gate if all the inputs are logic 1 then and then only output is logic 0; otherwise output is logic 1. The operation is similar to the 2-input NAND gate. The table show the truth table for 3-input NAND gate.

Inputs			Device Status				Output ( $Y=A.B.C$ )'
A	B	C	Q1	Q2	Q3	Q4	
0	0	0	ON	OFF	ON	OFF	1(HIGH)
0	0 or 1	0 or 1	ON	OFF	ON	OFF	1(HIGH)
1	1	1	OFF	ON	OFF	ON	0(LOW)

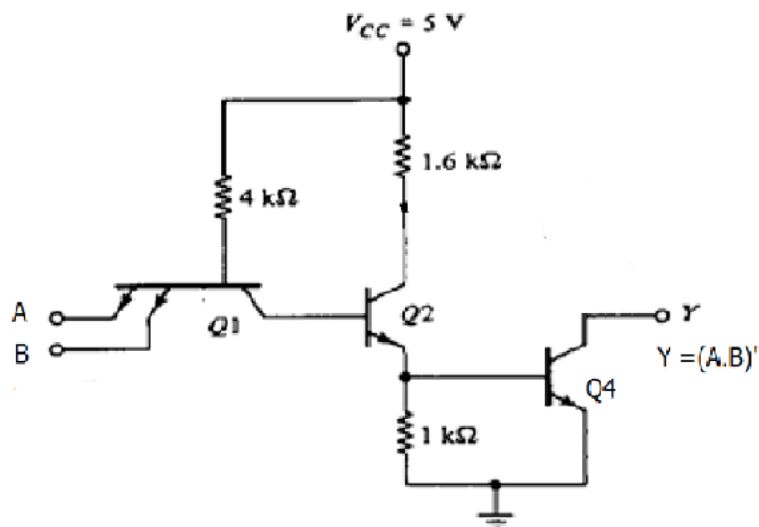
### ✿ Totem-Pole Output



- ✿ The figure shows a highlighted output configuration.
- ✿ Transistor Q3 and Q4 form a totem-pole. Such a configuration is known as active pull-up or totem pole output.
- ✿ The active pull-up formed by Q3 and Q4 has specific advantage.
- ✿ Totem-pole transistors are used because they produce LOW output impedance
- ✿ Either Q3 acts as a emitter follower (HIGH output) or Q4 is saturated (LOW output)
- ✿ When Q3 is conducting, the output impedance is approximately  $70\Omega$ ; when Q4 is saturated, the output impedance is only  $12\Omega$ .
- ✿ Either way, the output impedance is low. This means that the output voltage can change quickly from one state to the other because any stray output capacitance is rapidly charged or discharged through the low output impedance. Thus the propagation delay is low in totem-pole TTL logic.
- ✿ Open-Collector Output
- ✿ One problem with totem pole output is that two outputs cannot be tied together. See the figure, where the totem pole outputs of two separate gates are connected together at point X. Suppose that the output of gate A is high (Q3A ON and Q4A OFF) and the output of gate B is low (Q3B OFF and Q4B ON). In this situation transistor Q4B acts as a load for Q3A. Since Q4B is a low resistance load, it draws high current around 55 mA. This current might not damage Q3A or Q4B immediately, but over a period of time can cause overheating and deterioration in performance and eventual device failure.



- Some TTL devices provide another type of output called open collector output. The outputs of two different gates with open collector output can be tied together. This is known as wired logic. Figure shows a 2-input NAND gate with an open-collector output eliminates the pull-up transistor Q<sub>3</sub>, D<sub>1</sub> and R<sub>4</sub>. The output is taken from the open collector terminal of transistor Q<sub>4</sub>.



- ✿ Because the collector of Q4 is open, a gate like this will not work properly until you connect an external pull-up resistor, as shown in fig. When Q4 is ON, output is low and when Q4 is OFF output is tied to VCC through an external pull up resistor.

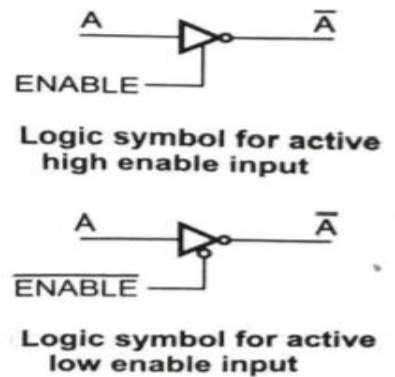
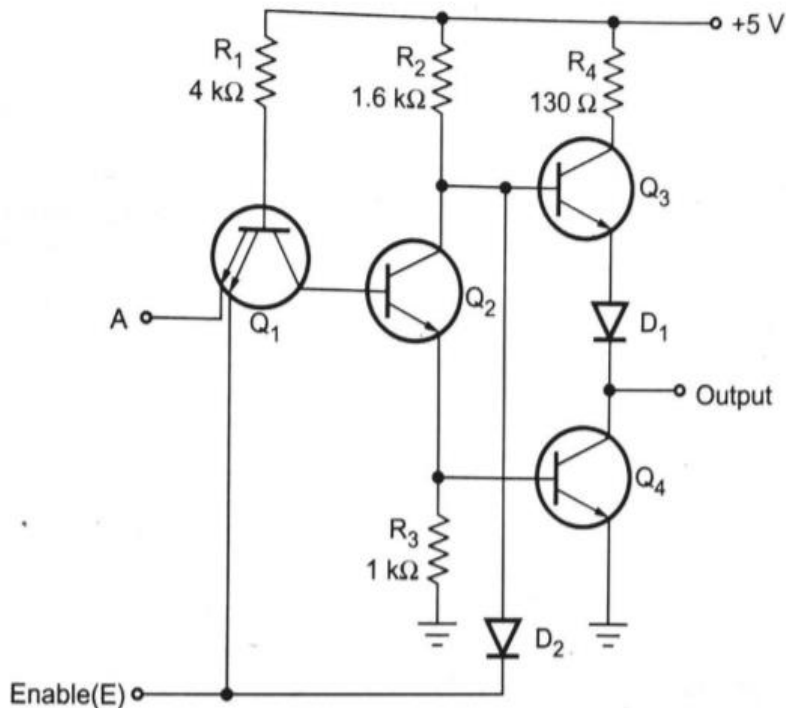
### ✿ Comparison between Totem-Pole and Open-Collector Outputs

Totem-pole	Open collector
Output stage consists of pull-up transistor (Q3), diode resistor and pull-down transistor (Q4)	Output stage consists of only pull-down transistor.
External pull-up resistor is not required.	External pull-up resistor is required for proper operation of gate.
Output of two gates cannot be tied together.	Output of two gates can be tied together using wired AND technique.
Operating speed is high.	Operating speed is low.

- ✿ Table summarizes the difference between totem-pole and open collector outputs.

### ✿ Tri-State TTL Inverter

- ✿ The tristate configuration is a third type of TTL output configuration. It utilizes the high-speed operation of the totem-pole arrangement while permitting outputs to be wired-ANDed (connected together). It is called tristate TTL because it allows three possible output stages: HIGH, LOW and high impedance. We know the transistor Q3 is ON when output is HIGH and Q4 is ON when output is LOW. In the high impedance state both transistors, transistors Q3 and Q4 in the totem-pole arrangement are turned OFF. As a result, the output is open or floating, it is neither LOW nor HIGH.



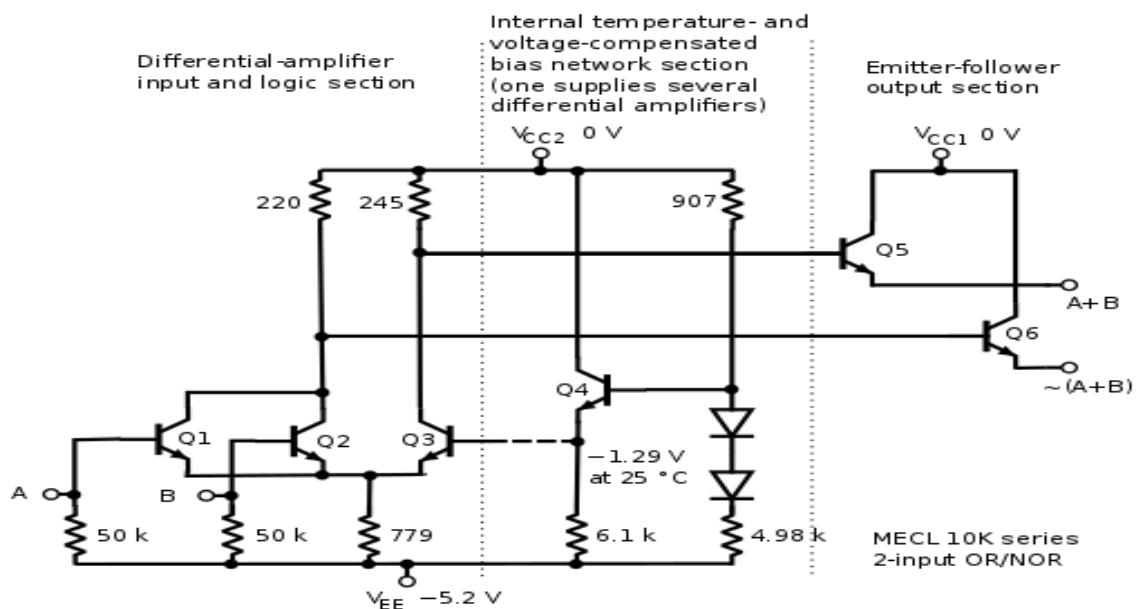
❁ The figure shows the simplified circuit for tristate inverter. It has two inputs A and E.

❁ A is the normal logic output whereas E is an ENABLE input. When ENABLE input is HIGH, the circuit works as a normal inverter. Because when E is HIGH, the state of the transistor Q1 (either ON or OFF) depends on the logic input A, and the additional component diode is open circuited as its cathode is at logic HIGH. When ENABLE input is LOW, regardless of the state of logic input A, the base-emitter junction of Q1 is forward biased and as a result it turns ON. This shunts the current through R1 away from Q2 making it OFF. As Q2 is OFF, there is no sufficient drive for Q4 to conduct and hence Q4 turns off. The LOW at ENABLE input also forward-biases diode D2 which shunt the current away from the base of Q3, making it OFF. In this way, when ENABLE input is LOW, both transistors are OFF and output is at high impedance state. Fig shows the logic symbols for tristate inverter. In above case circuit operation is enabled when ENABLE input is HIGH. Therefore, ENBLE input is active high. The logic symbol for high enable input is shown in figure. In some circuits ENABLE input can be active LOW, i.e. circuit operates when ENABLE input is LOW. The logic symbol for active low ENABLE input is shown in the figure.

- ✿ The internal temperature – and voltage -compensated bias network supplies a reference voltage (Bias voltage  $V_{BB} = -1.3 \text{ V}$ ) to the differential amplifier. The best noise immunity is obtained by connecting  $V_{CC}$  to ground and  $V_{EE}$  to  $-5.2 \text{ V}$ .

## ✿ 5.4 ECL – Emitter coupled logic:

- ✿ Emitter Coupled Logic (ECL) is a non saturated digital logic family. It achieves the propagation delay of  $2\text{ns}$ . Its required high speed system operation. The output provides both OR and NOR functions. Each input is connected to the base of transistor. The two voltage levels are about  $-0.8 \text{ V}$  for the high state and  $-1.8\text{V}$  for the low state.



- ✿ The circuit consists of

- ✿ Differential amplifier
- ✿ Temperature – and voltage -compensated bias network
- ✿ Emitter follower

- ✿ The Emitter follower output requires a pull down resistor for current to flow. This is obtained from the input resistor,  $R_p$  of other similar gates or from an external resistor connected to a negative voltage supply.



✿ Working:

- ✿ If any of the input is high the corresponding input transistor is turned ON and transistor  $Q_3$  is OFF.
- ✿ Ex: if  $V_A = -0.8V$ , the transistor  $Q_1$  starts conducting, So the  $V_{BE}(Q_1) = 0.8V$ . Now  $V_E(Q_1) = V_A - V_{BE}(Q_1) = -0.8V - 0.8V = -1.6V$
- ✿ Next to find the  $V_{BE}(Q_3)$ .  $V_{BE}(Q_3) = V_B(Q_3) - V_E(Q_3) = -1.3 - (-1.6) = 0.3V$ . Thus the transistor  $Q_3$  is OFF. So the transistor  $Q_1$  remains ON. The output voltage of the transistor  $Q_1$  is low. So the input voltage of transistor  $Q_6$  is Low. Since the transistor  $Q_6$  is a Emitter follower so the output of the transistor  $Q_6$  is also Low. This output produce the NOR output of the circuit.
- ✿ The transistor  $Q_3$  is OFF. The output voltage of the transistor  $Q_3$  is high. So the input voltage of transistor  $Q_5$  is high. Since the transistor  $Q_5$  is a Emitter follower so the output of the transistor  $Q_5$  is also high. This output produce the OR output of the circuit.
- ✿ 2. If both the inputs are low, transistors  $Q_1$  and  $Q_2$  are turned OFF and transistor  $Q_3$  is ON.
- ✿ Ex: if  $V_A = V_B = -1.8V$ , the transistor  $Q_1$  and  $Q_2 = \text{OFF}$ , the transistor  $Q_3$  is ON. So the  $V_{BE}(Q_3) = 0.8V$ . Now  $V_E(Q_3) = V_{BB} - V_{BE}(Q_1) = -1.3V - 0.8V = -2.1V$
- ✿ Next to find the  $V_{BE}(Q_1)$  or  $V_{BE}(Q_2)$  .  $V_{BE}(Q_1) = V_B(Q_1) - V_E(Q_1) = -1.8 - (-2.1) = 0.3V$ . Thus the transistor  $Q_1$  is OFF. So the transistor  $Q_3$  remains ON. The output voltage of the transistor  $Q_3$  is low. So the input voltage of transistor  $Q_3$  is Low. Since the transistor  $Q_5$  is a Emitter follower so the output of the transistor  $Q_5$  is also Low. This output produce the OR output of the circuit.
- ✿ The transistor  $Q_1$  is OFF. The output voltage of the transistor  $Q_1$  is high. So the input voltage of transistor  $Q_6$  is high. Since the transistor  $Q_6$  is a Emitter follower so the output of the transistor  $Q_6$  is also high. This output produce the NOR output of the circuit.

## ❁ Operation Table

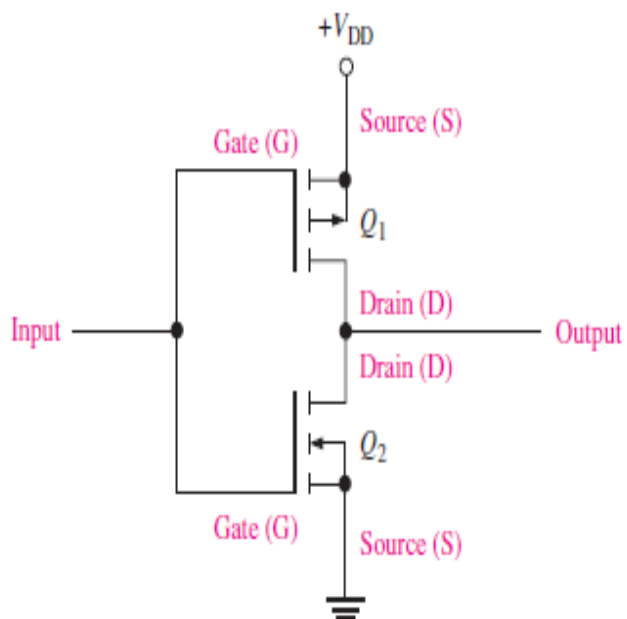
INPUT		Q1	Q2	Q3	Q5	Q6	NOR OUTPUT	OR OUTPUT
$V_A$	$V_B$							
-1.8V	-1.8V	OFF	OFF	ON	OFF	ON	HIGH	LOW
-1.8V	-0.8V	OFF	ON	OFF	ON	OFF	LOW	HIGH
-0.8V	-1.8V	ON	OFF	OFF	ON	OFF	LOW	HIGH
-0.8V	-0.8V	ON	ON	OFF	ON	OFF	LOW	HIGH

## ❁ 5.5 CMOS Families

❁ Complementary MOS (CMOS) logic uses the MOSFET in complementary pairs as its basic element.

❁ A complementary pair uses both  $p$ -channel and  $n$ -channel enhancement MOSFETs

### ❁ CMOS AS INVERTER



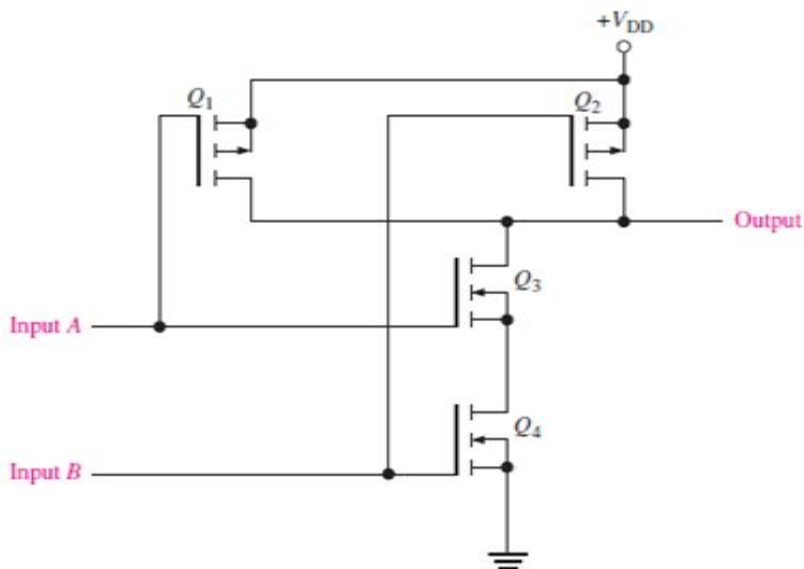
A	Q1 PMOS	Q2 NMOS	Y
0	ON	OFF	1
1	OFF	ON	0

## ❁ CMOS AS NAND LOGIC

❁ When both inputs are LOW,  $Q_1$  and  $Q_2$  are on, and  $Q_3$  and  $Q_4$  are off.

❁ The output is pulled HIGH through the *on* resistance of  $Q_1$  and  $Q_2$  in parallel.

❁ When input  $A$  is LOW and input  $B$  is HIGH,  $Q_1$  and  $Q_4$  are on, and  $Q_2$  and  $Q_3$  are off.

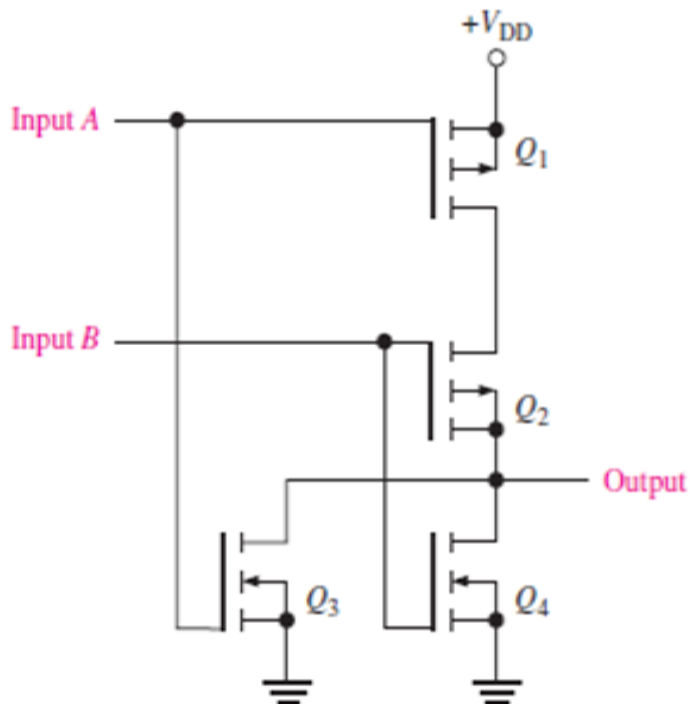


A ( $Q_1$ & $Q_3$ )	B ( $Q_2$ & $Q_4$ )	$Q_1$ PMOS	$Q_2$ PMOS	$Q_3$ NMOS	$Q_4$ NMOS	Y
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

- ✿ The output is pulled HIGH through the low *on* resistance of  $Q_1$ .
- ✿ When input  $A$  is HIGH and input  $B$  is LOW,  $Q_1$  and  $Q_4$  are off, and  $Q_2$  and  $Q_3$  are on.
- ✿ The output is pulled HIGH through the low *on* resistance of  $Q_2$ .
- ✿ When both inputs are HIGH,  $Q_1$  and  $Q_2$  are off, and  $Q_3$  and  $Q_4$  are on.
- ✿ The output is pulled LOW through the *on* resistance of  $Q_3$  and  $Q_4$  in series to ground.

### ✿ CMOS AS NOR LOGIC

- ✿ When both inputs are LOW,  $Q_1$  and  $Q_2$  are on, and  $Q_3$  and  $Q_4$  are off.
- ✿ The output is pulled HIGH through the *on* resistance of  $Q_1$  and  $Q_2$  in series.



<b>A</b> <b>(Q1 &amp; Q3)</b>	<b>B</b> <b>(Q2 &amp; Q4)</b>	<b>Q1</b> <b>PMOS</b>	<b>Q2</b> <b>PMOS</b>	<b>Q3</b> <b>NMOS</b>	<b>Q4</b> <b>NMOS</b>	<b>Y</b>
<b>0</b>	0	ON	ON	OFF	OFF	1
<b>0</b>	1	ON	OFF	OFF	ON	0
<b>1</b>	0	OFF	ON	ON	OFF	0
<b>1</b>	1	OFF	OFF	ON	ON	0

- ✿ When input *A* is LOW and input *B* is HIGH, *Q1* and *Q4* are on, and *Q2* and *Q3* are off.
- ✿ The output is pulled LOW through the low *on* resistance of *Q4* to ground.
- ✿ When input *A* is HIGH and input *B* is LOW, *Q1* and *Q4* are off, and *Q2* and *Q3* are on.
- ✿ The output is pulled LOW through the *on* resistance of *Q3* to ground.
- ✿ When both inputs are HIGH, *Q1* and *Q2* are off, and *Q3* and *Q4* are on. The output is
- ✿ pulled LOW through the *on* resistance of *Q3* and *Q4* in parallel to ground.

## Comparison between Various Digital Logic Families

Parameter	RTL	DTL	TTL	ECL	CMOS
<b>Components used</b>	Resistors and transistor	Resistor diode and transistor	Resistor, diode and transistor	Resistor and transistor	N-channel and P-channel MOSFET
<b>Circuit Complexity</b>	Simple	Moderate	Complex	Complex	Moderate
<b>Noise margin [Noise immunity]</b>	Nominal	Good	Very good	Good	Very good
<b>Fan-out</b>	Low (4)	Medium (8)	More (10)	High (25)	50
<b>Power dissipation in mW per gate</b>	12	8 – 12	10	40 – 55	0.1
<b>Basic gate</b>	NOR	NAND	NAND	OR-NOR	NAND/NOR
<b>Propagation delay in ns</b>	12	30	10	2 (ECL 10 K) 0.75 (ECL 100K)	70
<b>Speed power product (PJ)</b>	144	300	100	100 (ECL 10 K) 40 (ECL 100 K)	0.7
<b>Applications</b>	Absolute	Absolute	Laboratory instruments	High speed switching applications (low propagation delay)	Portable instrument with battery supply (low power consumption )
<b>Number of functions</b>	High	Fairly high	Very high	High	Low
<b>Clock rate MHz</b>	8	12 – 30	15 – 60	60 – 400	5

## 9. Lecture Notes

### ✿ E-Books

Digital Fundamentals\_ Global Ed - Thomas L Floyd

Digital Principles And Application - Leach & Malvino

Digital Design - M. Morris Mano and Michael D. Ciletti

Fundamentals of Digital Logic with Verilog Design-Stephen Brown and Zonko Vranesic

### ✿ ONLINE LEARNING MATERIALS:

**1:** [http://nptel.iitm.ac.in/video.php? subject Id=117106086](http://nptel.iitm.ac.in/video.php?subject%20Id=117106086)

**2:** <http://nptel.iitm.ac.in/courses/117101001>

**3:** <https://youtu.be/C-oAyXibnJU>

**4** <https://youtu.be/oYRMYSIVj1o>

**5:** <https://www.youtube.com/watch?v=XZmGGAbHqa0>

**6:** <https://www.youtube.com/watch?v=KymIDyQiXZI>

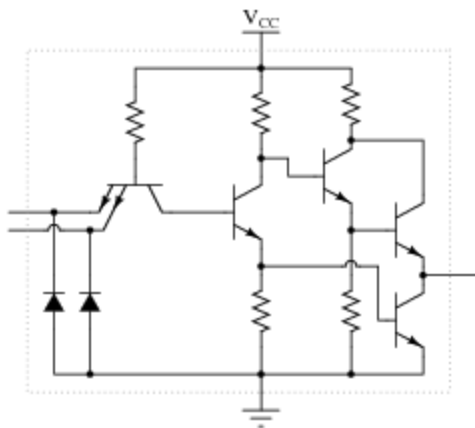
### ✿ Video Links

<https://drive.google.com/open?id=1qCP5dBvi1LZxd6S7FAUxt788iMmLLpbq>

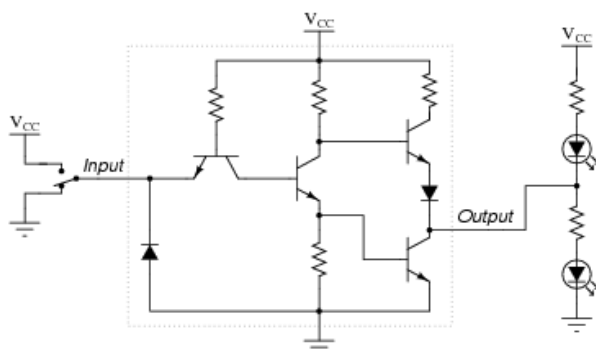
<https://drive.google.com/open?id=1hrXWfXKWnhidFbnDPDtd75hQt9OzOG6O>

## 10. Assignments

- ✿ 1. Describe the procedure to get the Hamming code for the binary word 1101100.
- ✿ 2. A 12 bit Hamming code word containing 8 bits of data and 4 parity bits is read from memory. What was the original 8 bit data word that was written into memory if the 12 bit word read out is as (i) 101110010100 and (ii) 111111110100
- ✿ 3. The following is an internal schematic of a TTL logic gate. Analyse the transistor circuit and determine what type of gate (AND, OR, NAND, NOR, XOR, etc.) it is:



- ✿ A certain gate has a propagation delay of 5 ns and  $I_{CH} = 1$  mA and  $I_{CL} = 2.5$  mA with a dc supply voltage of 5 V. Determine the speed-power product.
- ✿ Draw the paths of all currents in this circuit with the input in a "low" state and in a high state:





## 11. Part A Q & A (with K level and CO)

PART A		
Questions and Answers	Blooms Level	COs
<p>1. What is a binary number system ?</p> <p>The number system with base (or radix) two is known as the binary number system. Only two symbols are used to represent the numbers in the system and these are 0 and 1.</p>	K1	CO1
<p>2. What is an Excess3 code?</p> <p>The excess3 code is a non weighted code which is obtained from the 8-4-2-1 code by adding 3(0011) to each of the codes.</p>	K1	CO1
<p>3. What is the difference between analog and digital systems?</p> <p>In a digital system the physical quantities or signals can assume only discrete values, while in analog systems the physical quantities or signals vary continuously over a specified range.</p>	K1	CO1
<p>4. Subtract <math>X = 1010100</math> and <math>Y = 1000011</math> using 2's compliment</p> <p><math>X = 1010100</math>, 2's complement of <math>Y = + 0111101</math></p> <p>Sum = 10010001, Discard end carry, <math>X - Y = 0010001</math></p>	K1	CO1
<p>5. Subtract <math>X = 1010100</math> and <math>Y = 1000011</math>, using 1's complement.</p> <p><math>X = 1010100</math>, 1's complement of <math>Y = + 0111100</math></p> <p>Sum = 10010000, End -around carry = + 1, <math>X - Y = 0010001</math></p>	K1	CO1
<p>6. What is meant by parity bit?</p> <p>A parity bit is an extra bit included with a message to make the total number of 1's either even or odd.</p>	K1	CO1
<p>7. Represent binary number 1101.101 in power of 2 and find its decimal equivalent</p> $N = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$ <p>= 13.625</p>	K1	CO1

<p>8. What is the difference between binary code and BCD?</p> <p><u>Binary:</u> Any distinct element can be represented by a binary code. No limitation for the minimum or maximum number of elements required for coding the element.</p> <p><u>BCD:</u> Only a decimal digit can be represented. It is a four bit representation</p>	K1	CO1
<p>9. Convert 0.640625 decimal number to its octal equivalent.</p> <p><math>0.640625 \times 8 = 5.125</math>, <math>0.125 \times 8 = 1.0</math></p> <p>Ans. = 0.640 625 <math>_{10}</math> = 0.51</p>	K1	CO1
<p>10. What is a gray code?</p> <p>A gray code is a non weighted code which has the property that the codes for successive decimal digits differ in exactly one bit.</p>	K1	CO1
<p>11. Define noise margin</p> <p>It is the maximum noise voltage added to an input signal of a digital circuit that does not cause an undesirable change in the circuit output. It is expressed in volts.</p>	K1	CO1
<p>12. Define fan-out</p> <p>Number of logic gates at the next stage that can be loaded to a given logic gate output so that voltages for each of the possible logic state remain within the defined limits</p>	K1	CO1
<p>13. Mention the important characteristics of digital IC's.</p> <p>Fan out, Power dissipation, Propagation Delay, Noise Margin , Fan In, Operating temperature ,Power supply requirements.</p>	K1	CO1
<p>14. What is propagation delay?</p> <p>Propagation delay for a logic output from a logic gate means the time interval between change in a defined reference point input voltage and reflection of its effect at the output.</p>	K1	CO1
<p>15. Classify the logic family by operation?</p> <p>The RTL, DTL, TTL, I<sup>2</sup>L, HTL logic comes under the saturated logic family. The Schottky TTL, and ECL logic comes under the unsaturated logic family.</p>	K1	CO1

<p>16. Convert 0.1289062 decimal number to its hex equivalent</p> <p><math>0.1289062 \times 16 = 2.0625</math></p> <p><math>0.0625 \times 16 = 1.0</math></p> <p>Ans. = 0.21 16</p>	K1	CO1
<p>17. Convert decimal number 22.64 to hexadecimal number.</p> <p>16/ 22 R-6</p> <p>16/ 1 R-1</p> <p>R-0</p> <p><math>0.64 \times 16 = 10.24</math></p> <p><math>0.24 \times 16 = 3.84</math></p> <p><math>0.84 \times 16 = 13.44</math></p> <p><math>.44 \times 16 = 7.04</math></p> <p>Ans. = (16 : A 3 D 7)</p>	K1	CO1
<p>18. What are the two steps in Gray to binary conversion?</p> <p>The MSB of the binary number is the same as the MSB of the gray code number. To obtain the next binary digit, perform an exclusive OR operation between the bit just written down and the next gray code bit.</p>	K1	CO1
<p>19. Convert gray code 101011 into its binary equivalent.</p> <p>Gray Code : 1 0 1 0 1 1</p> <p>Binary Code 1 1 0 0 1 0</p>	K1	CO1
<p>20. Convert (9 B 2 - 1A) H to its decimal equivalent.</p> <p><math>N = 9 \times 16^2 + B \times 16^1 + 2 \times 16^0 + 1 \times 16^{-1} + A(10) \times 16^{-2}</math></p> <p><math>= 2304 + 176 + 2 + 0.0625 + 0.039</math></p> <p><math>= 2482.1 10</math></p>	K1	CO1

## 12. Part B Qs (with K level and CO)

PART B		
1. (i) Explain in detail about error detecting and error correcting codes (ii) Given the two binary numbers $X = 10101002$ and $Y = 10000112$ , perform the subtraction (a) $X - Y$ and (b) $Y - X$ using 2's complement	K1	CO1
2. Detect correct errors, if any, in the even parity Hamming code word 1010111 and write the correct code.	K1	CO1
3. (i) Perform the following using BCD and Excess-3 addition (205+569) (ii) Encode the binary word 11010 into odd parity Hamming code	K1	CO1
4. Perform the following operation and express the answer in octal form $(756)_8 - (437)_8 + (725)_8$ .	K1	CO1
5. A 12 bit hamming code word containing 8 bits of data and 4 parity bits is read from memory. What was the original 8 bit data word that was written into the memory if the 12 bit word read out as 101110010100 (2) 111111110100	K1	CO1
6. Perform each of the following computations using signed 8-bit words in 1's complement and 2's complement binary arithmetic: (i) $(+95)_{10} + (63)_{10}$ (ii) $(+42)_{10} + (-87)_{10}$ (iii) $(-13)_{10} + (-59)_{10}$ (iv) $(+38)_{10} + (-38)_{10}$ (v) $(-105)_{10} + (-120)_{10}$	K1	CO1
7. The state of a 12-cell register is 010110010111. What is its content if it represents: (a) 3 decimal digits in BCD. (b) 3 decimal digits in Excess-3 code (c) 3 decimal digits in 2421 BCD code (d) 3 decimal digits in 84-2-1 BCD code.	K1	CO1
8. Draw and explain the circuit diagram of a CMOS NAND and NOR gate	K1	CO1
9. Explain the concept and implementation of ECL NOR/ OR gate.	K1	CO1
10. Compare DTL, RTL, TTL Logic families	K1	CO1

### 13.Supportive online Certification courses (**NPTEL, Swayam, Coursera, Udemy, etc.,**)

#### ✿ Swayam

Digital Circuits: [https://swayam.gov.in/nd1\\_noc20\\_ee70/preview](https://swayam.gov.in/nd1_noc20_ee70/preview)

Duration: 12 weeks, Start Date: 14 Sep 2020, End Date: 04 Dec 2020

#### ✿ Udemy

Digital Electric Circuits & Intelligent Electrical Devices

<https://www.udemy.com/course/digital-electric-circuits-intelligent-electrical-devices/>

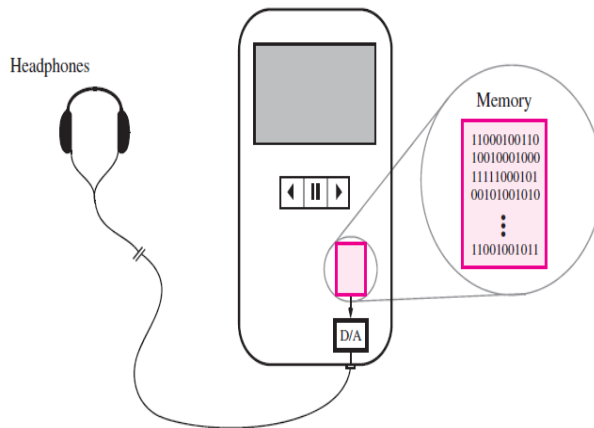
#### ✿ Coursera

Build a Modern Computer from First Principles: From Nand to Tetris (Project-Centered Course)

<https://www.coursera.org/learn/build-a-computer>

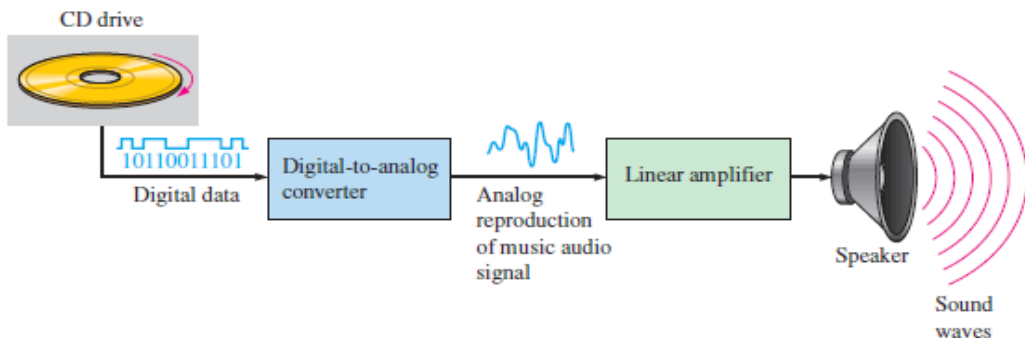
## 14. Real time Applications in day to day life and to Industry

- ✿ A few real time applications of digital systems is shown and explained.
- ✿ Basic block diagram of a iPod player
- ✿ The iPod player stores music in the digital form in its memory and the Digital to analog converter converts it into the music that we hear through the headphones.



### ✿ Basic block diagram of a CD player

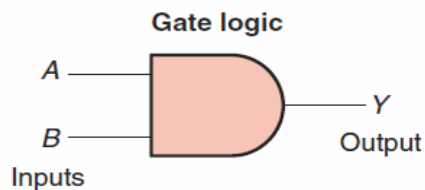
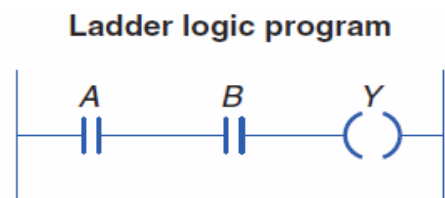
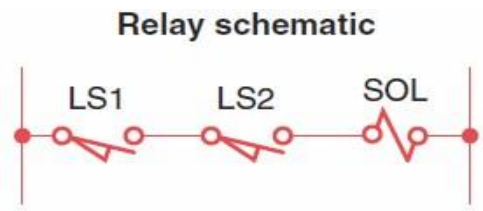
- ✿ The compact disk (CD) player has both digital and analog circuits. Music is stored on the compact disk in digital form. A laser diode optical system picks up the digital data from the rotating disk and transfers it to the digital-to-analog converter (DAC). The DAC changes the digital data into an analog signal which is an electrical reproduction of the original music. This signal is amplified and sent to the speaker. An analog-to-digital converter (ADC) is used to record the music on to the disk in digital form.



## 15. Contents beyond the Syllabus ( COE related Value added courses)

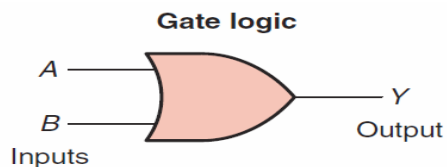
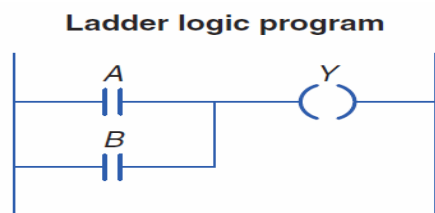
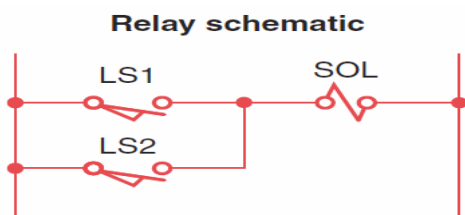
✿ **Simulation of logic gates using PLC simulation software available at Factory Automation CoE.**

✿ Example: Two limit switches connected in **series** and used to control a solenoid valve.



Boolean equation:  $AB = Y$

✿ Example: Two limit switches connected in **parallel** and used to control a solenoid valve.



Boolean equation:  $A + B = Y$

## 17. Prescribed Text Books & Reference Books

### ✿ TEXT BOOKS:

1. James W. Bignel, Digital Electronics, Cengage learning, 5th Edition, 2007.
2. M. Morris Mano, 'Digital Design with an introduction to the VHDL', Pearson Education, 2013.
3. Comer "Digital Logic & State Machine Design, Oxford, 2012.

### ✿ REFERENCES

1. Mandal, "Digital Electronics Principles & Application, McGraw Hill Edu, 2013.
2. William Keitz, Digital Electronics-A Practical Approach with VHDL, Pearson, 2013.
3. Thomas L. Floyd, 'Digital Fundamentals', 11th edition, Pearson Education, 2015.
4. Charles H. Roth, Jr, Lizy Lizy Kurian John, 'Digital System Design using VHDL, Cengage, 2013.



# Digital Logic Circuits

## Unit 2

## 8. Activity based learning

- ✿ Understanding the working of the combinational logic circuits through simulation using Logisim
- ✿ Simulation of
  - ✿ Half adder
  - ✿ Full adder
  - ✿ Multiplexer
  - ✿ De-multiplexer
- ✿ Practice using open source DEEDS: <https://www.digitalelectronicsdeeds.com/>

## 9. Lecture Notes

### Table of Contents

#### ✿ UNIT II COMBINATIONAL CIRCUITS

- ✿ 1. Combinational logic – Representation of logic functions-SOP and POS forms,
- ✿ 2. K-map representations - minimization using K maps
- ✿ 3. Simplification and implementation of combinational logic
- ✿ Multiplexers and De multiplexers
- ✿ Code converters,
- ✿ Adders, Subtractors,
- ✿ Encoders and Decoders.

# COMBINATIONAL CIRCUITS

## ❁ Introduction:

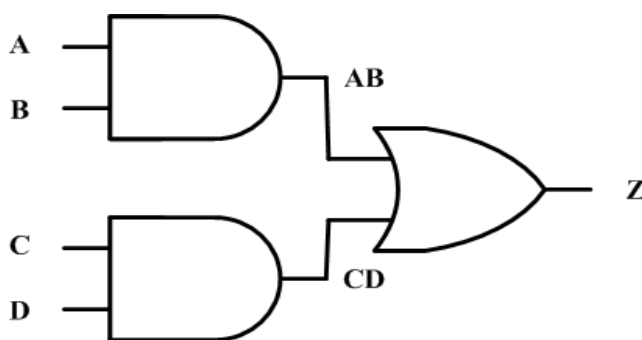
1. In digital logic, the inputs and output of a function are in the form of binary numbers (boolean values) i.e., the values are either zero (0) or one (1). Therefore, digital logic is also known as 'Boolean logic'.
2. These inputs and output can be termed as 'Boolean Variables'.
3. The output boolean variable of a digital signal can be expressed in terms of input boolean variables which forms the 'Boolean Expression'.
4. A boolean expression is an expression which consists of variables, constants (0-false and 1-true) and logical operators which results in true or false.
5. They are used to describe Switching Function / Boolean Function.
6. In Boolean function, the variables are appeared either in complemented or in uncomplemented form. Each occurrence of a variable in either a complemented or a uncomplemented form is called a literal.
7. A Boolean function can be represented as:

- Algebraic expression Eg:  $Z(A,B,C,D) = AB + CD$

Truth table

- Truth table

- Logic diagram using gates



Logic diagram for  $Z(A,B,C,D) = AB + CD$

A	B	C	D	Z	AB	CD
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	1	0	1
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	1	0	1
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	1	0	1
1	1	0	0	1	1	0
1	1	0	1	1	1	0
1	1	1	0	1	1	0
1	1	1	1	1	1	1

# Boolean Algebra

- ✿ Boolean algebra is a convenient and systematic way of expressing and analysing the operation of logic circuits.

It was developed by George Boole in 1847 and was proposed for design by Claude E. Shannon in 1938.

Boolean algebra provides economical and straightforward approach and is used extensively in designing electronic circuits used in computers

## ✿ Laws of Boolean Algebra

### ✿ Basic Identities

1.  $X + 0 = X$
2.  $X \cdot 1 = X$
3.  $X + 1 = 1$
4.  $X \cdot 0 = 0$
5.  $X + X = X$
6.  $X \cdot X = X$
7.  $X + X' = 1$
8.  $X \cdot X' = 0$
9.  $(X')' = X$

### ✿ Basic Laws

#### ✿ Commutative

10.  $X + Y = Y + X$
11.  $X \cdot Y = Y \cdot X$

#### ✿ Associative

12.  $X + (Y + Z) = (X + Y) + Z$
13.  $X(YZ) = (XY)Z$

#### ✿ Distributive

14.  $X(Y + Z) = XY + XZ$
15.  $X + YZ = (X + Y)(X + Z)$

✿ De Morgan's Theorem

$$16. (X + Y)' = X' \cdot Y'$$

$$17. (XY)' = X' + Y'$$

✿ Consensus Theorem

$$18. XY + X'Z + YZ = XY + X'Z$$

✿ The *dual* of a function is obtained by interchanging OR and AND operations and replacing 1s and 0s with 0s and 1s.

Theorem	Dual Theorem	
$X + X = X$	$X \cdot X = X$	Idempotent Law
$X + 1 = 1$	$X \cdot 0 = 0$	Annulment Law
$X + X \cdot Y = X$	$X \cdot (X + Y) = X$	Absorption Law
$(X')' = X$		Involution Law
$X + (Y + Z) = (X + Y) + Z$	$X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$	Associative Law
$(X + Y)' = X' \cdot Y'$	$(XY)' = X' + Y'$	De Morgan's Theorem

## ❁ Simplification using Boolean rules

❁ Reduce  $AB + (AC)' + AB'C(AB + C)$

$$AB + (AC)' + AB'C(AB + C) = AB + (AC)' + AAB'BC + AB'CC$$

$$= AB + (AC)' + AB'CC \quad [A.A' = 0]$$

$$= AB + (AC)' + AB'C \quad [A.A = 1]$$

$$= AB + A' + C' = AB'C \quad [(AB)' = A' + B']$$

$$= A' + B + C' + AB'C \quad [A + AB' = A + B]$$

$$= A' + B'C + B + C' \quad [A + A'B = A + B]$$

$$= A' + B + C' + B'C$$

$$= A' + B + C' + B'$$

$$= A' + C' + 1$$

$$= 1 \quad [A + 1 = 1]$$

❁ Simplify the following using De Morgan's theorem  $[((AB)'C)'' D]'$

$$[((AB)'C)'' D]' = ((AB)'C)'' + D' \quad [(AB)' = A' + B']$$

$$= (AB)' C + D'$$

$$= (A' + B')C + D'$$

❁ Show that  $(X + Y' + XY)(X + Y')(X'Y) = 0$

$$(X + Y' + XY)(X + Y')(X'Y) = (X + Y' + X)(X + Y')(X' + Y) \quad [A + A'B = A + B]$$

$$= (X + Y')(X + Y')(X'Y) \quad [A + A = 1]$$

$$= (X + Y')(X'Y) \quad [A.A = 1]$$

$$= X.X' + Y'.X'.Y$$

$$= 0 \quad [A.A' = 0]$$

- ✿ A logic circuit can be designed and realized or implemented with minimal gates by simplifying the expression.

✿ Simplify  $F = X'YZ + X'YZ' + XZ$

$$X'YZ + X'YZ' + XZ = X'Y(Z + Z') + XZ$$

$$= X'Y \cdot 1 + XZ \quad [A + A' = 1]$$

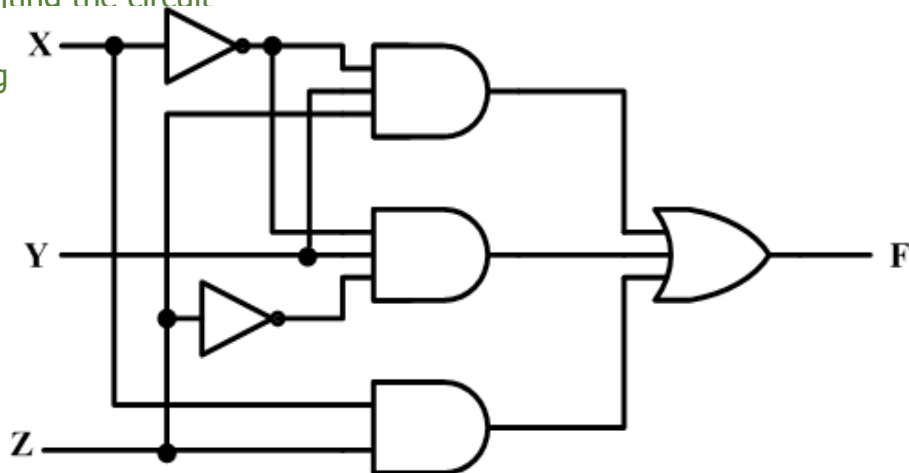
$$= X'Y + XZ$$

$$\mathbf{F = X'Y + XZ}$$

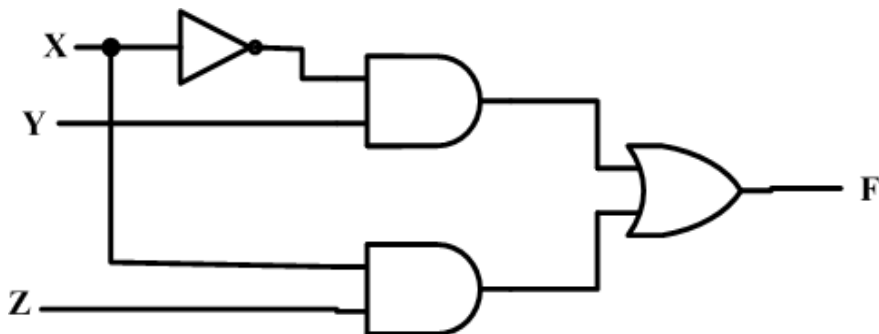
- ✿ The simplification of the equation reduces the number of gates used for implementing the circuit

- ✿ For the g

WS:



- ✿ For the reduced expression  $F = X'Y + XZ$  the logic diagram is minimized as follows:





# 1. Representation of logic functions

## - SOP and POS forms

1. Representation of Boolean expression can be primarily done in two ways. They are as follows:
  - a. Sum of Products (SOP) form
  - b. Product of Sums (POS) form
2. The terms “product” and “sum” have been borrowed from mathematics to describe AND and OR logic operations.
3. Note: If the number of input variables are  $n$ , then the total number of combinations in Boolean algebra is  $2^n$ .

### ✿ 1.1 Sum of Products (SOP) Form :

1. It is one of the ways of writing a boolean expression, formed by adding (OR operation) the product terms.
2. Product Term: Is defined as either a literal or a product of literals (also called as Eg.:  $F(A,B,C) = \bar{A} B \bar{C} + A \bar{B} C + \bar{A} B C + A B \bar{C} + A B C$  that are ANDed together.  
Eg: ABC , XYZ
3. SOP: Is defined as group of product terms that are ORed together.  
Eg:  $F(A,B,C) = AB + BC + AC$
4. Other Names: Disjunctive Normal Form, Disjunctive Normal Formula.
5. Standard SOP / Canonical SOP: SOP is said to be a Standard SOP / Canonical SOP, if each product term consists of all literals in either complemented form or uncomplemented form.  
Eg.:  $F(A,B,C) = A B C + A B \bar{C} + A \bar{B} C + A \bar{B} \bar{C} + \bar{A} B C + \bar{A} B \bar{C} + \bar{A} \bar{B} C + \bar{A} \bar{B} \bar{C}$
6. Minterms (  $\Sigma m$  ): Is defined as each individual product term in Standard SOP / Canonical SOP.

0	Complement of a Variable
1	Variable

7. SOP Form is obtained by writing one product term for each input combination

## ❁ 1.2 Product of Sums (POS) Form :

1. It is one of the ways of writing a boolean expression, formed by multiplying (AND operation) the sum terms.
2. Sum Term: Is defined as either a literal or a sum of literals (also called as disjunction). It is also defined as a group of literals that are ORed together.

Eg:  $A+B+C$  ,  $X+Y+Z$

3. POS: Is defined as group of sum terms that are ANDed together.

Eg:  $F(A,B,C) = (A+B) (B+C) (A+C)$

4. Other Names: Conjunctive Normal Form, Conjunctive Normal Formula.

5. Standard POS / Canonical POS: POS is said to be a Standard POS / Canonical POS, if each sum term consists of all literals in either complemented form or uncomplemented form.

Eg.:  $F(A,B,C) = (\bar{A}+B+\bar{C}) (\bar{A}+B+C) (A+\bar{B}+C) (A+B+\bar{C}) (A+B+C)$

6. Maxterms (IIM ): Is defined as each individual sum term in Standard POS / Canonical POS.

0	Variable
1	Complement of a Variable

7. POS Form is obtained by writing one sum term for each input combination that produces an output 0.

## ❁ 1.3 Conversion between SOP & POS forms:

### ❁ 1.3.1 SOP to Standard SOP (Canonical SOP) Conversion:

**Step 1:** Find the missing literals in each product term.

**Step 2:** AND each product term with (Missing Literal + Missing Literal)

**Step 3:** Expand and reorder.

**Step 4:** Omit more than once repeated terms.

### ❁ 1.3.2 POS to Standard POS (Canonical POS) Conversion:

**Step 1:** Find the missing literals in each sum term.

**Step 2:** OR each sum term with (Missing Literal . Missing Literal)

**Step 3:** Expand and reorder.

**Step 4:** Omit more than once repeated terms.

### ❁ 1.3.3 SOP to POS Conversion:

**Step 1:** SOP to Standard SOP (Canonical SOP)

**Step 2:** Standard SOP (Canonical SOP) to Minterms ( $\Sigma m$ )

**Step 3:** Minterms ( $\Sigma m$ ) to Maxterms ( $\Pi M$ )

**Step 4:** Maxterms ( $\Pi M$ ) to Standard POS (Canonical POS)

### ❁ 1.3.4 POS to SOP Conversion:

**Step 1:** POS to Standard POS (Canonical POS)

**Step 2:** Standard POS (Canonical POS) to Maxterms ( $\Pi M$ )

**Step 3:** Maxterms ( $\Pi M$ ) to Minterms ( $\Sigma m$ )

**Step 4:** Minterms ( $\Sigma m$ ) to Standard SOP (Canonical SOP)

### Example Problems:

Convert the expression  $Y = AB + ACD$  into the canonical SOP form.

**Step 1:** Find the missing literals in each product term.

$$\begin{array}{ccc} Y = & AB & + ACD \\ & \downarrow & \downarrow \\ & C, D & B \end{array}$$

**Step 2:** AND each product term with (Missing Literal + Missing Literal)

$$Y = AB(C + \bar{C})(D + \bar{D}) + ACD(B + \bar{B})$$

**Step 3:** Expand and reorder.

$$Y = (ABC + AB\bar{C})(D + \bar{D}) + ABCD + A\bar{B}CD$$

$$Y = ABCD + AB\bar{C}D + ABC\bar{D} + AB\bar{C}\bar{D} + ABCD + A\bar{B}CD$$

**Step 4:** Omit more than once repeated terms.

$$Y = ABCD + AB\bar{C}D + ABC\bar{D} + AB\bar{C}\bar{D} + ~~ABCD~~ + A\bar{B}CD$$

<b>Ans.:</b> $Y = ABCD + AB\bar{C}D + ABC\bar{D} + AB\bar{C}\bar{D} + A\bar{B}CD$
---

**Convert the expression  $Y = (A + \bar{B})(B + C)(A + \bar{C})$  into the canonical POS form.**

**Step 1:** Find the missing literals in each sum term.

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

$\downarrow \quad \quad \downarrow \quad \quad \downarrow$   
 $C \quad \quad A \quad \quad B$

**Step 2:** OR each sum term with (Missing Literal . Missing Literal)

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

**Step 3:** Expand and reorder.

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

**Step 4:** Omit more than once repeated terms.

$$Y = (A + \bar{B} + C)(A + \bar{B} + \bar{C})(A + B + C)(\bar{A} + B + C) \\ (A + B + \bar{C})(A + \cancel{B} + \bar{C})$$

**Ans.:**  $Y = (A + \bar{B} + C)(A + \bar{B} + \bar{C})(A + B + C) \\ (\bar{A} + B + C)(A + B + \bar{C})$

**Convert the expression  $Y = \bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + ABC$  into the canonical POS form.**

**Step 1:** SOP to Standard SOP (Canonical SOP)

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

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

**Step 2:** Standard SOP (Canonical SOP) to Minterms ( $\Sigma m$ )

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

$$Y = 001 + 011 + 100 + 111$$

$$Y = m1 + m3 + m4 + m7$$

$$Y = \Sigma m(1,3,4,7)$$

**Step 3:** Minterms ( $\Sigma m$ ) to Maxterms ( $\Pi M$ )

$$Y = \Pi M(0,2,5,6)$$

**Step 4:** Maxterms ( $\Pi M$ ) to Standard POS (Canonical POS)

$$Y = M0 \cdot M2 \cdot M5 \cdot M6$$

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

<b>Ans.:</b> $Y = (A + B + C)(A + \bar{B} + C)(\bar{A} + B + \bar{C})(\bar{A} + \bar{B} + C)$
---

**Convert the expression  $Y = (A + \bar{B})(B + C)(A + \bar{C})$  into the canonical SOP form.**

**Step 1:** POS to Standard POS (Canonical POS)

**(Refer Problem 2)**

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

**Step 2:** Standard POS (Canonical POS) to Maxterms ( $\Pi M$ )

$$Y = (0+1+0) (0+1+1) (0+0+0) (1+0+0) (0+0+1)$$

$$Y = M_2 \cdot M_3 \cdot M_0 \cdot M_4 \cdot M_1$$

$$Y = \Pi M(2,3,0,4,1)$$

**Step 3:** Maxterms ( $\Pi M$ ) to Minterms ( $\Sigma m$ )

$$Y = \Sigma m(5,6,7)$$

**Step 4:** Minterms ( $\Sigma m$ ) to Standard SOP (Canonical SOP)

$$Y = m_5 + m_6 + m_7$$

$$Y = A\bar{B}C + AB\bar{C} + ABC$$

<b>Ans.: <math>Y = A\bar{B}C + AB\bar{C} + ABC</math></b>
---



**Write the logical expression in Canonical SOP & Canonical POS form for the truth table given below.**

Input			Output
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

**Ans.:**

**SOP Form:**

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

**POS Form:**

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

**Write the logical expression in Canonical SOP & Canonical POS form for the truth table given below.**

Input			Output
A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

**Ans.:**

**SOP Form:**

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

**POS Form:**

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

## ❁ 2. K-map Representations - Minimization using K-maps

### ❁ 2.1 Introduction:

1. Maurice Karnaugh, a Telecommunications Engineer, developed the Karnaugh Map at Bell Labs in 1953 while designing digital logic based telephone switching circuits.
2. Karnaugh Map was developed with the aid of Venn Diagrams.
3. Karnaugh Maps reduce logic functions more quickly and easily compared to Boolean Algebra, which in turn reduces the number of gates and inputs.
4. We like to simplify logic to a lowest cost form, to save costs by elimination of components. We define lowest cost as being the lowest number of gates with the lowest number of inputs per gate.

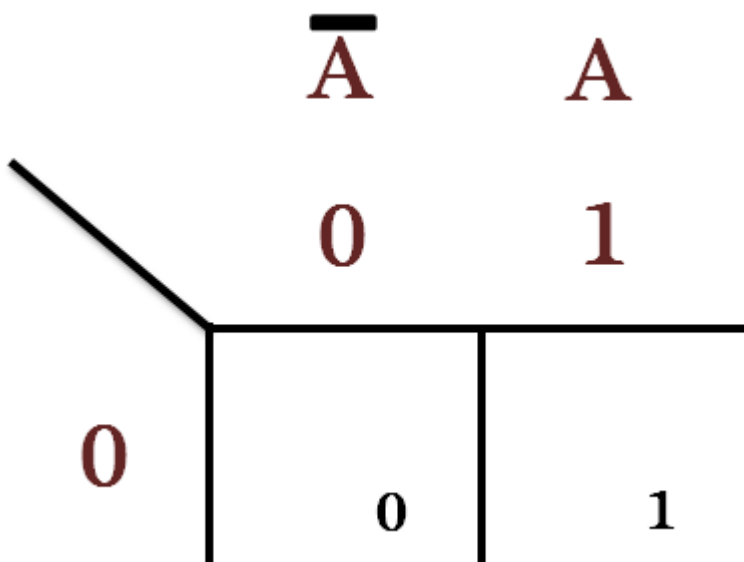
### ❁ 2.2 Karnaugh Map:

1. Karnaugh Maps is a graphical method of reducing a digital circuit to its minimum number of gates.
2. The map is a simple table containing 1s and 0s that can express a truth table or complex boolean expression describing the operation of a digital circuit.
3. The map is then used to work out the minimum number of gates needed, by graphical means rather than by algebra.
4. Karnaugh Maps can be used on small circuits having two or three inputs as an alternative to Boolean Algebra, and on more complex circuits having up to 6 inputs, it can provide quicker and simpler minimization than Boolean Algebra.

## ❁ 2.3 Structure of Karnaugh Map :

1. The shape and size of the map is dependent on the number of binary inputs in the circuit to be analyzed.
2. The map needs one cell for each possible binary word applied to the inputs.
3. In general, n input circuit, requires  $2^n$  cells:
  - a. 2 input circuits with inputs A and B, require maps with  $2^2 = 4$  cells
  - b. 3 input circuits with inputs A B and C, require maps with  $2^3 = 8$  cells
  - c. 4 input circuits with inputs A B C and D, require maps with  $2^4 = 16$  cells
4. The input labels are written at the top left hand corner, divided by a diagonal line. The top and left edges of the map then represent all the possible input combinations for the inputs allocated to that edge.
5. Notice that this edge numbering does not follow the normal binary counting sequence, but uses a Gray Code sequence where only one bit changes from one cell to the next. This is an important feature of Karnaugh Map.

## One Variable K-Map (SOP)



Two Variable K-Map (SOP)

		$\overline{B}$	B
		0	1
$\overline{A}$	0	0	1
A	1	2	3

Three Variable K-Map (SOP)

		$\overline{B} \ \overline{C}$	$\overline{B} \ C$	B C	B $\overline{C}$
		0 0	0 1	1 1	1 0
$\overline{A}$	0	0	1	3	2
A	1	4	5	7	6

# Four Variable K-Map (SOP)

			$\bar{C} \bar{D}$	$\bar{C} D$	$C D$	$C \bar{D}$
			0 0	0 1	1 1	1 0
$\bar{A} \bar{B}$	0 0		0	1	3	2
$\bar{A} B$	0 1		4	5	7	6
$A B$	1 1		12	13	15	14
$A \bar{B}$	1 0		8	9	11	10

# Five Variable K-Map (SOP)

		$\bar{A}$			
		$\bar{D}\bar{E}$	$\bar{D}E$	$DE$	$D\bar{E}$
		0 0	0 1	1 1	1 0
$\bar{B}\bar{C}$	0 0	0	1	3	2
$\bar{B}C$	0 1	4	5	7	6
$BC$	1 1	12	13	15	14
$B\bar{C}$	1 0	8	9	11	10

		$A$			
		$\bar{D}\bar{E}$	$\bar{D}E$	$DE$	$D\bar{E}$
		0 0	0 1	1 1	1 0
$\bar{B}\bar{C}$	0 0	16	17	19	18
$\bar{B}C$	0 1	20	21	23	22
$BC$	1 1	28	29	31	30
$B\bar{C}$	1 0	24	25	27	26

## ❁ 2.4 Karnaugh Map Rules:

### SOP

1. Select K-Map according to the number of variables.
2. Identify minterms or maxterms as given in problem.
3. For SOP put 1's in blocks of K-map respective to the minterms (0's elsewhere).
4. Make rectangular groups containing total terms in power of two like 2,4,8 ..... and try to cover as many elements as you can in one group (Group 1s).
5. From the groups made in step 5, find the product terms and sum them up for SOP form.

### POS

1. Select K-Map according to the number of variables.
2. Identify minterms or maxterms as given in problem.
3. For POS put 0's in blocks of K-map respective to the maxterms (1's elsewhere).
4. Make rectangular groups containing total terms in power of two like 2,4,8 ..... and try to cover as many elements as you can in one group (Group 0s).
5. From the groups made in step 5, find the sum terms and multiply them up for POS form.



- **Note 1:**

- a. If outputs are not defined for some combination of inputs, then those output values will be represented with don't care symbol 'x'.
- b. That means, we can consider them as either '0' or '1'.

- **Note 2:**

- a. If don't care terms also present, then place don't cares 'x' in the respective cells of K-map.
- b. Consider only the don't cares 'x' that are helpful for grouping maximum number of adjacent ones. In those cases, treat the don't care value as '1' for SOP & as '0' for POS.

- **Note 3:**

- a. Groups must contain 1, 2, 4, 8, 16 ( $2^n$ ) cells.
- b. ★ For SOP : Groups must contain only 1 (and X if don't care is allowed).  
★ For POS : Groups must contain only 0 (and X if don't care is allowed).
- c. Groups may be horizontal or vertical, but not diagonal.
- d. Groups should be as large as possible.
- e. ★ For SOP : Each cell containing a 1 must be in at least one group.  
★ For POS : Each cell containing a 0 must be in at least one group.
- f. Groups may overlap.
- g. Groups may wrap around the table. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.
- h. There should be as few groups as possible.

✿ Obtain minimum SOP form for the following function:

$$F = \Sigma m(0,1,2)$$

		$\bar{B}$	B
		0	1
$\bar{A}$	0	1 <sub>0</sub>	1 <sub>1</sub>
A	1	1 <sub>2</sub>	0 <sub>3</sub>

$$\text{Ans.: } F = \bar{A} + \bar{B}$$

✿ Obtain minimum SOP form for the following function:

$$F = \Sigma m(0,1,2,3)$$

		$\bar{B}$	B
		0	1
$\bar{A}$	0	1 <sub>0</sub>	1 <sub>1</sub>
A	1	1 <sub>2</sub>	1 <sub>3</sub>

$$\text{Ans.: } F = 1$$

✿ Obtain minimum SOP form for the following function:

$$F = \Sigma m(0,3)$$

		$\bar{B}$	B
		0	1
$\bar{A}$	0	1 <sub>0</sub>	0 <sub>1</sub>
A	1	0 <sub>2</sub>	1 <sub>3</sub>

$$\text{Ans.: } F = \bar{A}\bar{B} + AB = A \oplus B$$

✿ Obtain minimum SOP form for the following function:

$$F = \Sigma m(1,2) + \Sigma d(3)$$

		$\bar{B}$	B
		0	1
$\bar{A}$	0	0 <sub>0</sub>	1 <sub>1</sub>
A	1	1 <sub>2</sub>	X <sub>3</sub>

$$\text{Ans.: } F = A + B$$

✿ Obtain minimum SOP form for the following function:

$$F = \sum m(0,1,2,3,4)$$

		$\bar{B} \bar{C}$	$\bar{B} C$	$B C$	$B \bar{C}$
		0 0	0 1	1 1	1 0
$\bar{A}$	0	1 <sub>0</sub>	1 <sub>1</sub>	1 <sub>3</sub>	1 <sub>2</sub>
A	1	1 <sub>4</sub>	0 <sub>5</sub>	0 <sub>7</sub>	0 <sub>6</sub>

$$\text{Ans.: } F = \bar{A} + \bar{B}\bar{C}$$

✿ Obtain minimum SOP form for the following function:

$$F = \sum m(1,2,4,6,7)$$

		$\bar{B} \bar{C}$	$\bar{B} C$	$B C$	$B \bar{C}$
		0 0	0 1	1 1	1 0
$\bar{A}$	0	0 <sub>0</sub>	1 <sub>1</sub>	0 <sub>3</sub>	1 <sub>2</sub>
A	1	1 <sub>4</sub>	0 <sub>5</sub>	1 <sub>7</sub>	1 <sub>6</sub>

$$\text{Ans.: } F = A\bar{C} + B\bar{C} + AB + \bar{A}\bar{B}C$$

✿ Obtain minimum SOP form for the following function:

$$F = \sum m(0,2,3,4,5,6)$$

		$\bar{B} \bar{C}$	$\bar{B} C$	$B C$	$B \bar{C}$
		0 0	0 1	1 1	1 0
$\bar{A}$	0	1 <sub>0</sub>	0 <sub>1</sub>	1 <sub>3</sub>	1 <sub>2</sub>
A	1	1 <sub>4</sub>	1 <sub>5</sub>	0 <sub>7</sub>	1 <sub>6</sub>

$$\text{Ans.: } F = \bar{C} + A\bar{B} + \bar{A}B$$

✿ Obtain minimum SOP form for the following function:

$$F = \sum m(0,1,3,5,6) + \sum d(2,4)$$

		$\bar{B} \bar{C}$	$\bar{B} C$	$B C$	$B \bar{C}$
		0 0	0 1	1 1	1 0
$\bar{A}$	0	1 <sub>0</sub>	1 <sub>1</sub>	1 <sub>3</sub>	X <sub>2</sub>
A	1	X <sub>4</sub>	1 <sub>5</sub>	0 <sub>7</sub>	1 <sub>6</sub>

$$\text{Ans.: } F = \bar{A} + \bar{C} + \bar{B}$$

✿ Obtain minimum SOP form for the following function:

$$F = \sum m(0,2,4,5,6,7,8,10,11,12,14,15)$$

	$\bar{C} \bar{D}$ 0 0	$\bar{C} D$ 0 1	$C D$ 1 1	$C \bar{D}$ 1 0
$\bar{A} \bar{B}$ 0 0	1 <sub>0</sub>	0 <sub>1</sub>	0 <sub>3</sub>	1 <sub>2</sub>
$\bar{A} B$ 0 1	1 <sub>4</sub>	1 <sub>5</sub>	1 <sub>7</sub>	1 <sub>6</sub>
$A B$ 1 1	1 <sub>12</sub>	0 <sub>13</sub>	1 <sub>15</sub>	1 <sub>14</sub>
$A \bar{B}$ 1 0	1 <sub>8</sub>	0 <sub>9</sub>	1 <sub>11</sub>	1 <sub>10</sub>

$$\text{Ans.: } F = \bar{D} + \bar{A}B + AC$$

✿ Obtain minimum SOP form for the following function:

$$F = \sum m(0,1,2,3,11,12,14,15)$$

	$\bar{C} \bar{D}$ 0 0	$\bar{C} D$ 0 1	$C D$ 1 1	$C \bar{D}$ 1 0
$\bar{A} \bar{B}$ 0 0	1 <sub>0</sub>	1 <sub>1</sub>	1 <sub>3</sub>	1 <sub>2</sub>
$\bar{A} B$ 0 1	0 <sub>4</sub>	0 <sub>5</sub>	0 <sub>7</sub>	0 <sub>6</sub>
$A B$ 1 1	1 <sub>12</sub>	0 <sub>13</sub>	1 <sub>15</sub>	1 <sub>14</sub>
$A \bar{B}$ 1 0	0 <sub>8</sub>	0 <sub>9</sub>	1 <sub>11</sub>	0 <sub>10</sub>

$$\text{Ans.: } F = \bar{A}\bar{B} + ACD + AB\bar{D}$$

✿ Obtain minimum SOP form for the following function:

$$F = \sum m(1,5,6,7,11,12,13,15)$$

	$\bar{C} \bar{D}$ 0 0	$\bar{C} D$ 0 1	$C D$ 1 1	$C \bar{D}$ 1 0
$\bar{A} \bar{B}$ 0 0	0 <sub>0</sub>	1 <sub>1</sub>	0 <sub>3</sub>	0 <sub>2</sub>
$\bar{A} B$ 0 1	0 <sub>4</sub>	1 <sub>5</sub>	1 <sub>7</sub>	1 <sub>6</sub>
$A B$ 1 1	1 <sub>12</sub>	1 <sub>13</sub>	1 <sub>15</sub>	0 <sub>14</sub>
$A \bar{B}$ 1 0	0 <sub>8</sub>	0 <sub>9</sub>	1 <sub>11</sub>	0 <sub>10</sub>

**Ans. :**  
 $F = \bar{A}\bar{C}D + \bar{A}BC + ACD + AB\bar{C}$

✿ Obtain minimum SOP form for the following function:

$$F = \sum m(0,2,4,5,6,7,8,10,13,15)$$

	$\bar{C} \bar{D}$ 0 0	$\bar{C} D$ 0 1	$C D$ 1 1	$C \bar{D}$ 1 0
$\bar{A} \bar{B}$ 0 0	1 <sub>0</sub>	0 <sub>1</sub>	0 <sub>3</sub>	1 <sub>2</sub>
$\bar{A} B$ 0 1	1 <sub>4</sub>	1 <sub>5</sub>	1 <sub>7</sub>	1 <sub>6</sub>
$A B$ 1 1	0 <sub>12</sub>	1 <sub>13</sub>	1 <sub>15</sub>	0 <sub>14</sub>
$A \bar{B}$ 1 0	1 <sub>8</sub>	0 <sub>9</sub>	0 <sub>11</sub>	1 <sub>10</sub>

**Ans. :  $F = \bar{B}\bar{D} + \bar{A}B + BD$**

✿ Obtain minimum SOP form for the following function:

$$F = \Sigma m(0,7,8,9,10,12) + \Sigma d(2,5,13)$$

	$\bar{C} \bar{D}$ 0 0	$\bar{C} D$ 0 1	$C D$ 1 1	$C \bar{D}$ 1 0
$\bar{A} \bar{B}$ 0 0	1 <sub>0</sub>	0 <sub>1</sub>	0 <sub>3</sub>	X <sub>2</sub>
$\bar{A} B$ 0 1	0 <sub>4</sub>	X <sub>5</sub>	1 <sub>7</sub>	0 <sub>6</sub>
$A B$ 1 1	1 <sub>12</sub>	X <sub>13</sub>	0 <sub>15</sub>	0 <sub>14</sub>
$A \bar{B}$ 1 0	1 <sub>8</sub>	1 <sub>9</sub>	0 <sub>11</sub>	1 <sub>10</sub>

Ans.:  $F = \bar{B}\bar{D} + A\bar{C} + \bar{A}BD$

✿ Obtain minimum SOP form for the following function:

$$F = \Sigma m(1,3,4,6,11) + \Sigma d(0,8,10,12,13)$$

	$\bar{C} \bar{D}$ 0 0	$\bar{C} D$ 0 1	$C D$ 1 1	$C \bar{D}$ 1 0
$\bar{A} \bar{B}$ 0 0	X <sub>0</sub>	1 <sub>1</sub>	1 <sub>3</sub>	0 <sub>2</sub>
$\bar{A} B$ 0 1	1 <sub>4</sub>	0 <sub>5</sub>	0 <sub>7</sub>	1 <sub>6</sub>
$A B$ 1 1	X <sub>12</sub>	X <sub>13</sub>	0 <sub>15</sub>	0 <sub>14</sub>
$A \bar{B}$ 1 0	X <sub>8</sub>	0 <sub>9</sub>	1 <sub>11</sub>	X <sub>10</sub>

Ans.:  
 $F = \bar{C}\bar{D} + \bar{A}\bar{B}D + \bar{B}CD + \bar{A}B\bar{D}$



❁ Obtain minimum SOP form for the following function:

$$F = \Sigma m(0,2,4,5,6,8,10,15) + \Sigma d(7,13,14)$$

	$\bar{C} \bar{D}$ 0 0	$\bar{C} D$ 0 1	$C D$ 1 1	$C \bar{D}$ 1 0
$\bar{A} \bar{B}$ 0 0	1 <sub>0</sub>	0 <sub>1</sub>	0 <sub>3</sub>	1 <sub>2</sub>
$\bar{A} B$ 0 1	1 <sub>4</sub>	1 <sub>5</sub>	X <sub>7</sub>	1 <sub>6</sub>
$A B$ 1 1	0 <sub>12</sub>	X <sub>13</sub>	1 <sub>15</sub>	X <sub>14</sub>
$A \bar{B}$ 1 0	1 <sub>8</sub>	0 <sub>9</sub>	0 <sub>11</sub>	1 <sub>10</sub>

$$\text{Ans.: } F = \bar{B}\bar{D} + \bar{A}B + BC$$

❁ Obtain minimum SOP form for the following function:

$$F = \Sigma m(0,5,7,8,9,10,11,14,15) + \Sigma d(1,4,13)$$

	$\bar{C} \bar{D}$ 0 0	$\bar{C} D$ 0 1	$C D$ 1 1	$C \bar{D}$ 1 0
$\bar{A} \bar{B}$ 0 0	1 <sub>0</sub>	X <sub>1</sub>	0 <sub>3</sub>	0 <sub>2</sub>
$\bar{A} B$ 0 1	X <sub>4</sub>	1 <sub>5</sub>	1 <sub>7</sub>	0 <sub>6</sub>
$A B$ 1 1	0 <sub>12</sub>	X <sub>13</sub>	1 <sub>15</sub>	1 <sub>14</sub>
$A \bar{B}$ 1 0	1 <sub>8</sub>	1 <sub>9</sub>	1 <sub>11</sub>	1 <sub>10</sub>

$$\text{Ans.: } F = A\bar{B} + AC + \bar{A}\bar{C} + BD$$

✿ Obtain minimum SOP form for the following function:

$$F = \sum m(0, 2, 4, 6, 9, 11, 13, 15, 17, 21, 25, 27, 29, 31)$$

		$\bar{A}$						A			
		$\bar{D}\bar{E}$	$\bar{D}E$	$DE$	$D\bar{E}$			$\bar{D}\bar{E}$	$\bar{D}E$	$DE$	$D\bar{E}$
		0 0	0 1	1 1	1 0			0 0	0 1	1 1	1 0
$\bar{B}\bar{C}$	0 0	1 <sub>0</sub>	0 <sub>1</sub>	0 <sub>3</sub>	1 <sub>2</sub>	$\bar{B}\bar{C}$	0 0	0 <sub>16</sub>	1 <sub>17</sub>	0 <sub>19</sub>	0 <sub>18</sub>
$\bar{B}C$	0 1	1 <sub>4</sub>	0 <sub>5</sub>	0 <sub>7</sub>	1 <sub>6</sub>	$\bar{B}C$	0 1	0 <sub>20</sub>	1 <sub>21</sub>	0 <sub>23</sub>	0 <sub>22</sub>
$BC$	1 1	0 <sub>12</sub>	1 <sub>13</sub>	1 <sub>15</sub>	0 <sub>14</sub>	$BC$	1 1	0 <sub>28</sub>	1 <sub>29</sub>	1 <sub>31</sub>	0 <sub>30</sub>
$B\bar{C}$	1 0	0 <sub>8</sub>	1 <sub>9</sub>	1 <sub>11</sub>	0 <sub>10</sub>	$B\bar{C}$	1 0	0 <sub>24</sub>	1 <sub>25</sub>	1 <sub>27</sub>	0 <sub>26</sub>

**Ans. :  $F = BE + \bar{A}\bar{B}\bar{E} + A\bar{D}E$**

✿ Obtain minimum SOP form for the following function:

$$F = \sum m(0, 1, 2, 3, 10, 12, 13, 14, 20, 21, 22, 23, 26, 27, 28, 29, 30, 31)$$

		$\bar{A}$						A			
		$\bar{D}\bar{E}$	$\bar{D}E$	$DE$	$D\bar{E}$			$\bar{D}\bar{E}$	$\bar{D}E$	$DE$	$D\bar{E}$
		0 0	0 1	1 1	1 0			0 0	0 1	1 1	1 0
$\bar{B}\bar{C}$	0 0	1 <sub>0</sub>	1 <sub>1</sub>	1 <sub>3</sub>	1 <sub>2</sub>	$\bar{B}\bar{C}$	0 0	0 <sub>16</sub>	0 <sub>17</sub>	0 <sub>19</sub>	0 <sub>18</sub>
$\bar{B}C$	0 1	0 <sub>4</sub>	0 <sub>5</sub>	0 <sub>7</sub>	0 <sub>6</sub>	$\bar{B}C$	0 1	1 <sub>20</sub>	1 <sub>21</sub>	1 <sub>23</sub>	1 <sub>22</sub>
$BC$	1 1	1 <sub>12</sub>	1 <sub>13</sub>	0 <sub>15</sub>	1 <sub>14</sub>	$BC$	1 1	1 <sub>28</sub>	1 <sub>29</sub>	1 <sub>31</sub>	1 <sub>30</sub>
$B\bar{C}$	1 0	0 <sub>8</sub>	0 <sub>9</sub>	0 <sub>11</sub>	1 <sub>10</sub>	$B\bar{C}$	1 0	0 <sub>24</sub>	0 <sub>25</sub>	1 <sub>27</sub>	1 <sub>26</sub>

**Ans. :  $F = AC + \bar{A}\bar{B}\bar{C} + ABD + BD\bar{E} + BCD$**

✿ Obtain minimum POS form for the following function:

$$F = \Pi M(0,4,5,7,10,11,14,15)$$

	$C + D$ $0 + 0$	$C + \bar{D}$ $0 + 1$	$\bar{C} + \bar{D}$ $1 + 1$	$\bar{C} + D$ $1 + 0$
$A + B$ $0 + 0$	0 <sub>0</sub>	1 <sub>1</sub>	1 <sub>3</sub>	1 <sub>2</sub>
$A + \bar{B}$ $0 + 1$	0 <sub>4</sub>	0 <sub>5</sub>	0 <sub>7</sub>	1 <sub>6</sub>
$\bar{A} + \bar{B}$ $1 + 1$	1 <sub>12</sub>	1 <sub>13</sub>	0 <sub>15</sub>	0 <sub>14</sub>
$\bar{A} + B$ $1 + 0$	1 <sub>8</sub>	1 <sub>9</sub>	0 <sub>11</sub>	0 <sub>10</sub>

$$\text{Ans.: } F = (\bar{A} + \bar{C})(A + C + D)(A + \bar{B} + \bar{D})$$

✿ Obtain minimum POS form for the following function:

$$F = \Pi M(0,3,4,7,8,10,12,14) + d(2,6)$$

	$C + D$ $0 + 0$	$C + \bar{D}$ $0 + 1$	$\bar{C} + \bar{D}$ $1 + 1$	$\bar{C} + D$ $1 + 0$
$A + B$ $0 + 0$	0 <sub>0</sub>	1 <sub>1</sub>	0 <sub>3</sub>	X <sub>2</sub>
$A + \bar{B}$ $0 + 1$	0 <sub>4</sub>	1 <sub>5</sub>	0 <sub>7</sub>	X <sub>6</sub>
$\bar{A} + \bar{B}$ $1 + 1$	0 <sub>12</sub>	1 <sub>13</sub>	1 <sub>15</sub>	0 <sub>14</sub>
$\bar{A} + B$ $1 + 0$	0 <sub>8</sub>	1 <sub>9</sub>	1 <sub>11</sub>	0 <sub>10</sub>

$$\text{Ans.: } F = D(A + \bar{C})$$

## 4. Multiplexer

### Multiplexer (Data selector):

- ❁ **Multiplexer** is a combinational circuit that has maximum of  $2^n$  data inputs, 'n' selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines.
- ❁ Since there are 'n' selection lines, there will be  $2^n$  possible combinations of zeros and ones. So, each combination will select only one data input. Multiplexer is also called as **MUX**.
- ❁ Digital multiplexer (MUX) selects binary information from one of many input lines and directs it to a single output line.

2:1 MUX:

- ❁ 2x1 Multiplexer has two data inputs  $W_0$ ,  $W_1$  one selection line  $S$  and one output  $f$ . The **block diagram** of 2x1 Multiplexer is shown in the following figure.

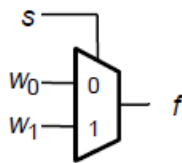


Fig Block diagram of 2:1 MUX

- ❁ One of these 2 inputs will be connected to the output based on the combination of inputs present at the selection line. Truth table of 4x1 Multiplexer is shown below.

s	f
0	$W_0$
1	$W_1$

Fig Truth table of 2:1 MUX

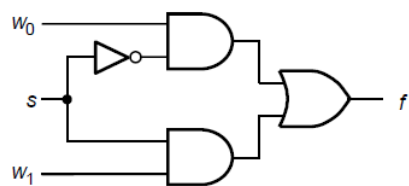


Fig Logic diagram of 2:1 MUX

- ❁ From Truth table, we can directly write the Boolean function for output,  $f$  as

$$f = W_0 \bar{S} + W_1 S$$

- ❁ We can implement this Boolean function using Inverters, AND gates & OR gate. The Logic diagram of 2x1 multiplexer is shown in the figure.

4:1 MUX:

- ❁ 4x1 Multiplexer has four data inputs  $I_3, I_2, I_1$  &  $I_0$ , two selection lines  $s_1$  &  $s_0$  and one output Y. The **block diagram** of 4x1 Multiplexer is shown in the following figure.

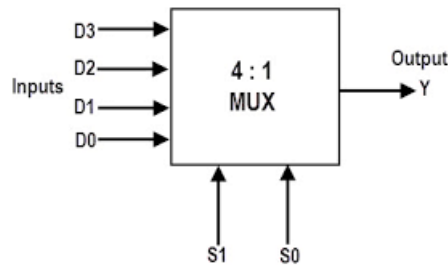


Fig Block diagram of 4:1 MUX

- ❁ One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. **Truth table** of 4x1 Multiplexer is shown below.

$s_1$	$s_0$	Out
0	0	$D_0$
0	1	$D_1$
1	0	$D_2$
1	1	$D_3$

Fig Truth table of 4x1 MUX

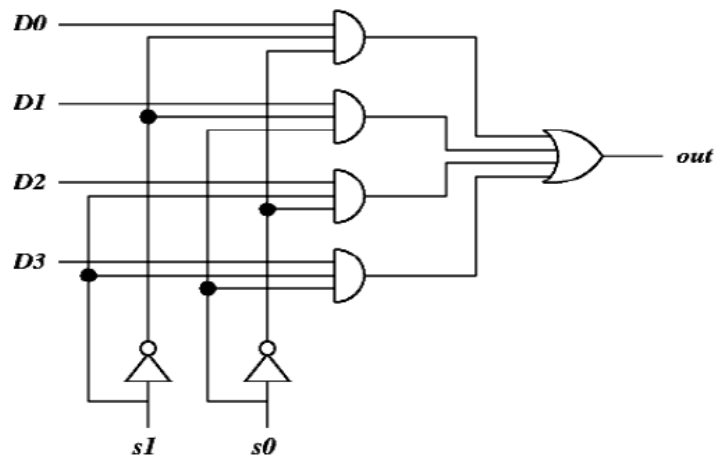


Fig Logic diagram of 4:1 MUX

- ❁ From Truth table, we can directly write the **Boolean function** for output, Y as

$$y = \overline{s_1} \overline{s_0} D_0 + \overline{s_1} s_0 D_1 + s_1 \overline{s_0} D_2 + s_1 s_0 D_3$$

- ❁ We can implement this Boolean function using Inverters, AND gates & OR gate. The Logic diagram of 4x1 multiplexer is shown in the figure.

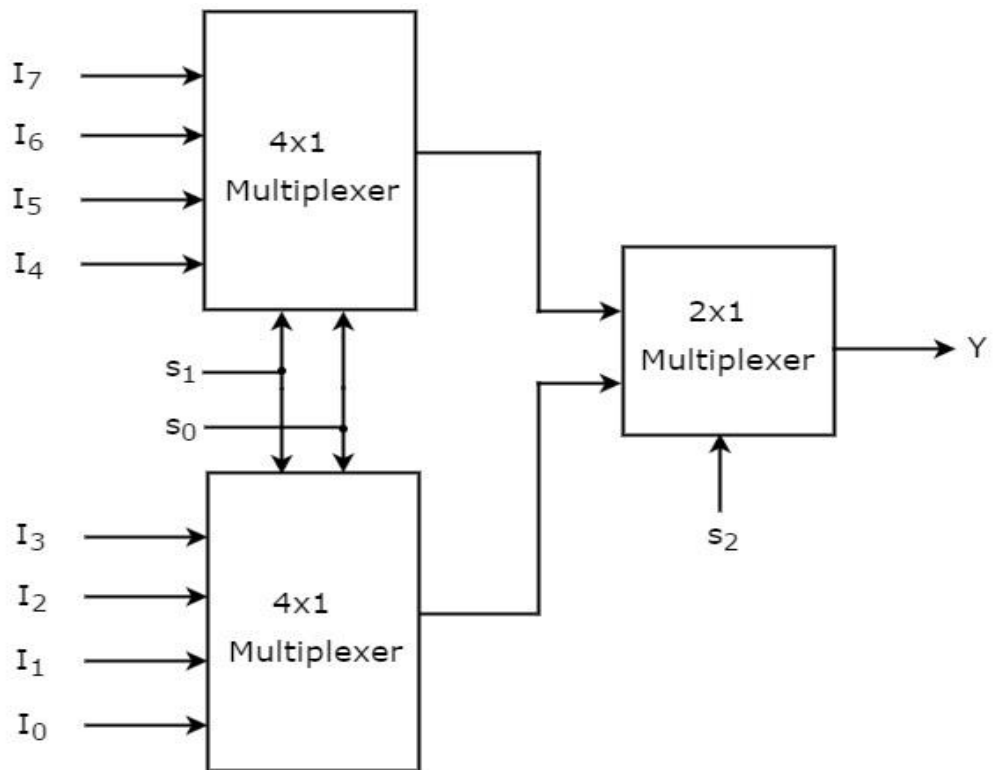
## Implementation of Higher-order Multiplexer:

### Implementation of 8:1 MUX using 4:1 MUX:

- ✿ We know that 4x1 Multiplexer has 4 data inputs, 2 selection lines and one output. Whereas, 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output.
- ✿ So, we require two **4x1 Multiplexers** in first stage in order to get the 8 data inputs. Since, each 4x1 Multiplexer produces one output, we require a **2x1 Multiplexer** in second stage by considering the outputs of first stage as inputs and to produce the final output.
- ✿ Let the 8x1 Multiplexer has eight data inputs  $I_7$  to  $I_0$ , three selection lines  $s_2$ ,  $s_1$  &  $s_0$  and one output Y. The **Truth table** of 8x1 Multiplexer is shown below.

Select Data Inputs			Output
$S_2$	$S_1$	$S_0$	Y
0	0	0	$D_0$
0	0	1	$D_1$
0	1	0	$D_2$
0	1	1	$D_3$
1	0	0	$D_4$
1	0	1	$D_5$
1	1	0	$D_6$
1	1	1	$D_7$

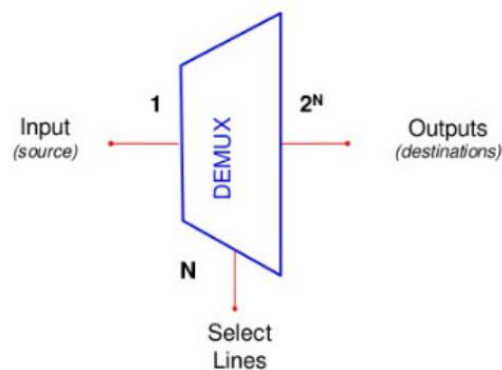
- ✿ We can implement 8x1 Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 8x1 Multiplexer is shown in the following figure.



- ✿ The same **selection lines,  $s_1$  &  $s_0$**  are applied to both 4x1 Multiplexers. The data inputs of upper 4x1 Multiplexer are  $I_7$  to  $I_4$  and the data inputs of lower 4x1 Multiplexer are  $I_3$  to  $I_0$ . Therefore, each 4x1 Multiplexer produces an output based on the values of selection lines,  $s_1$  &  $s_0$ .
- ✿ The outputs of first stage 4x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line,  $s_2$**  is applied to 2x1 Multiplexer.
- ✿ If  $s_2$  is zero, then the output of 2x1 Multiplexer will be one of the 4 inputs  $I_3$  to  $I_0$  based on the values of selection lines  $s_1$  &  $s_0$ .
- ✿ If  $s_2$  is one, then the output of 2x1 Multiplexer will be one of the 4 inputs  $I_7$  to  $I_4$  based on the values of selection lines  $s_1$  &  $s_0$ .
- ✿ Therefore, the overall combination of two 4x1 Multiplexers and one 2x1 Multiplexer performs as one 8x1 Multiplexer.

## 5. Demultiplexer

- ✿ It is also known as a **demux** or **data distributor**.
- ✿ Demultiplexer is a combinational circuit that accepts multiplexed data and distributes over multiple output lines
- ✿ In other words, the function of Demultiplexer is the inverse of the multiplexing operation. Similar to Multiplexer the output depends on the control input.
- ✿ It is a circuit which can distribute or deliver multiple outputs from a single input.
- ✿ It can perform as single input many output switch.
- ✿ The output lines of demultiplexer are  $2^N$  and one Data input=1
- ✿ where N- no.of Selection lines.



### ✿ Types of Demultiplexer:

1:2 Demultiplexer

1:4 Demultiplexer

1:8 Demultiplexer

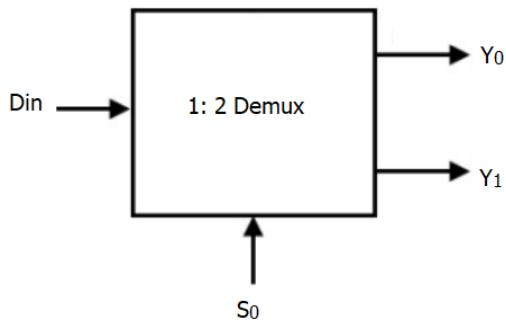
1:16 Demultiplexer



## 1 to 2 Demultiplexer:

✿ It consists of 1 data input, 1 Selection line and 2 output bits.  $Y_0$  and  $Y_1$  are the two output bits.

✿ Block Diagram:

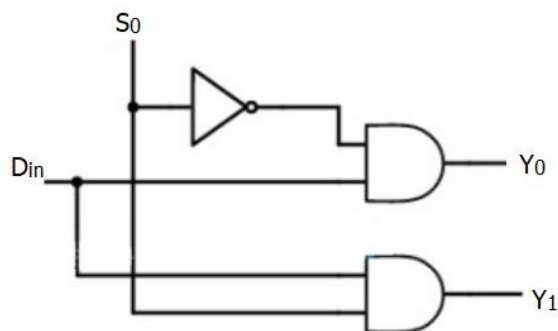


Output:

$$Y_0 = D_{in} S_0'$$

$$Y_1 = D_{in} S_0$$

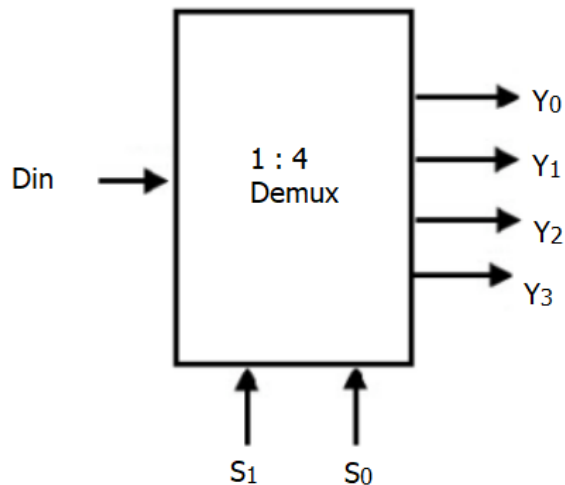
✿ Logic Diagram:



## 1 to 4 Demultiplexer:

✿ It consists of 1 data input, 2 Selection lines and 4 output bits.  $Y_0, Y_1, Y_2$  and  $Y_3$  are the four output bits.

✿ Block Diagram:



Truth Table:

INPUT			OUTPUT			
Data	$S_1$	$S_0$	$Y_0$	$Y_1$	$Y_2$	$Y_3$
$D_{in}$	0	0	$D_{in}$	0	0	0
$D_{in}$	0	1	0	$D_{in}$	0	0
$D_{in}$	1	0	0	0	$D_{in}$	0
$D_{in}$	1	1	0	0	0	$D_{in}$

Output:

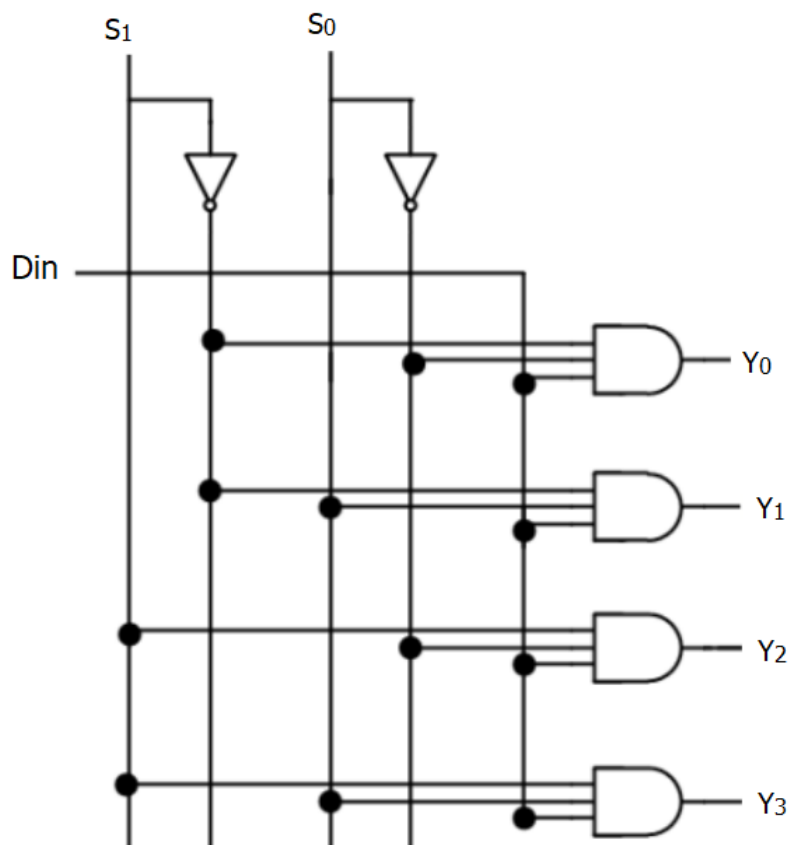
$$Y_0 = D_{in} S_1' S_0'$$

$$Y_1 = D_{in} S_1' S_0$$

$$Y_2 = D_{in} S_1 S_0'$$

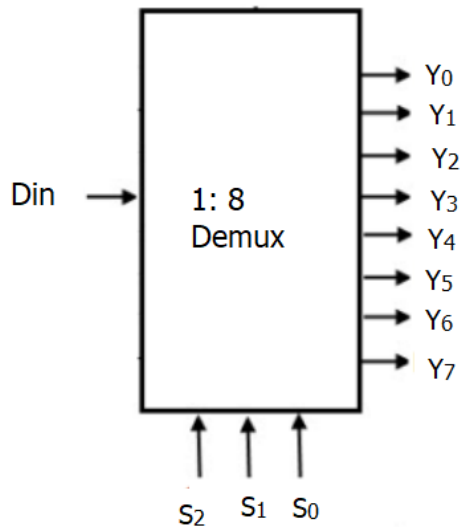
$$Y_3 = D_{in} S_1 S_0$$

## ❁ Logic Diagram:



## 1 to 8 Demultiplexer:

- It consists of 1 data input, 3 Selection lines and 8 output bits.  $Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, Y_7$  are the eight output bits.
- Block Diagram:



### Truth Table:

[illegible]

## ❁ Output

$$❁ Y_0 = D_{in} S_2' S_1' S_0'$$

$$❁ Y_1 = D_{in} S_2' S_1' S_0$$

$$❁ Y_2 = D_{in} S_2' S_1 S_0'$$

$$❁ Y_3 = D_{in} S_2' S_1 S_0$$

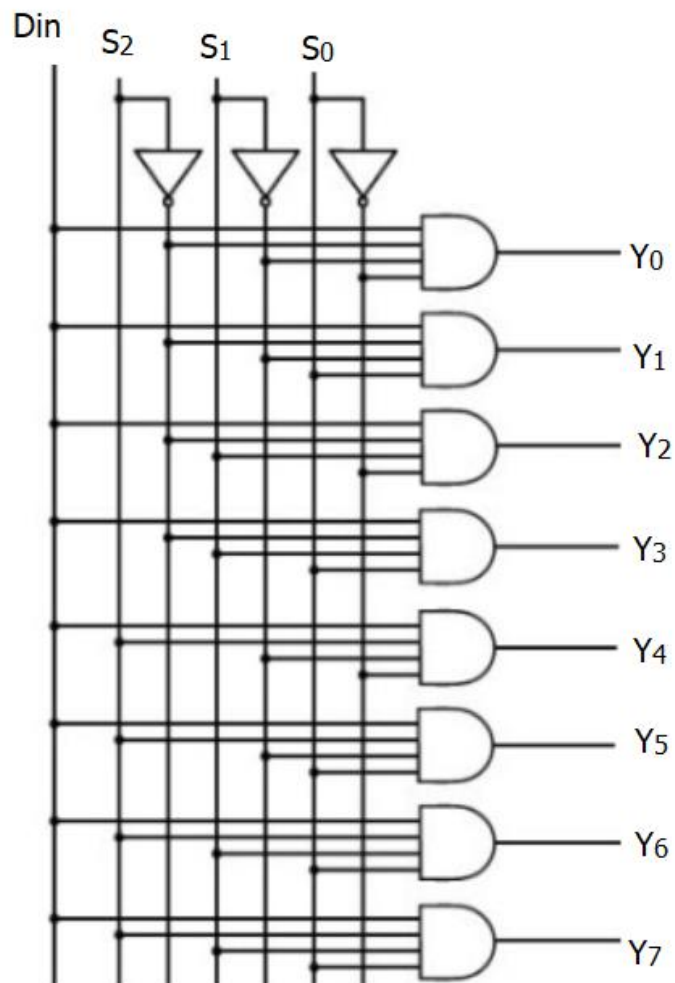
$$❁ Y_4 = D_{in} S_2 S_1' S_0'$$

$$❁ Y_5 = D_{in} S_2 S_1' S_0$$

$$❁ Y_6 = D_{in} S_2 S_1 S_0'$$

$$❁ Y_7 = D_{in} S_2 S_1 S_0$$

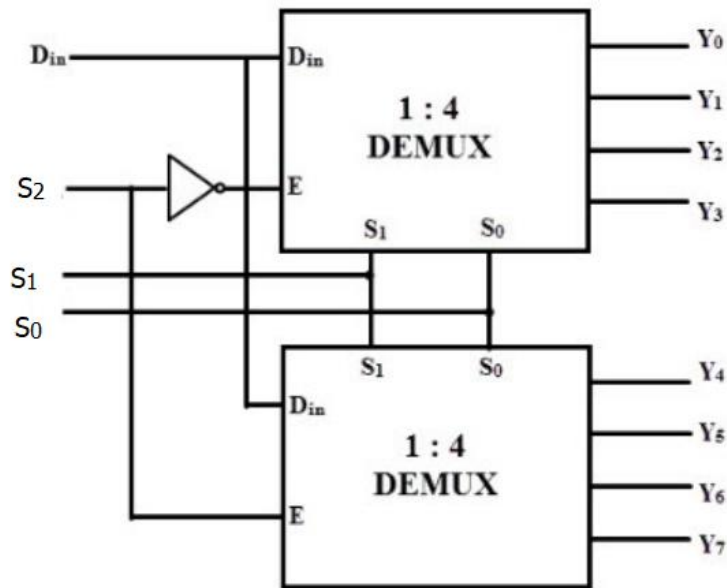
## ❁ Logic Diagram:



## Cascading of Demultiplexers

❁ Cascading refers to a process where large Demuxes can be designed and implemented using smaller Demuxes.

❁ 1:8 Demux Designed Using Two 1:4 Demuxes:

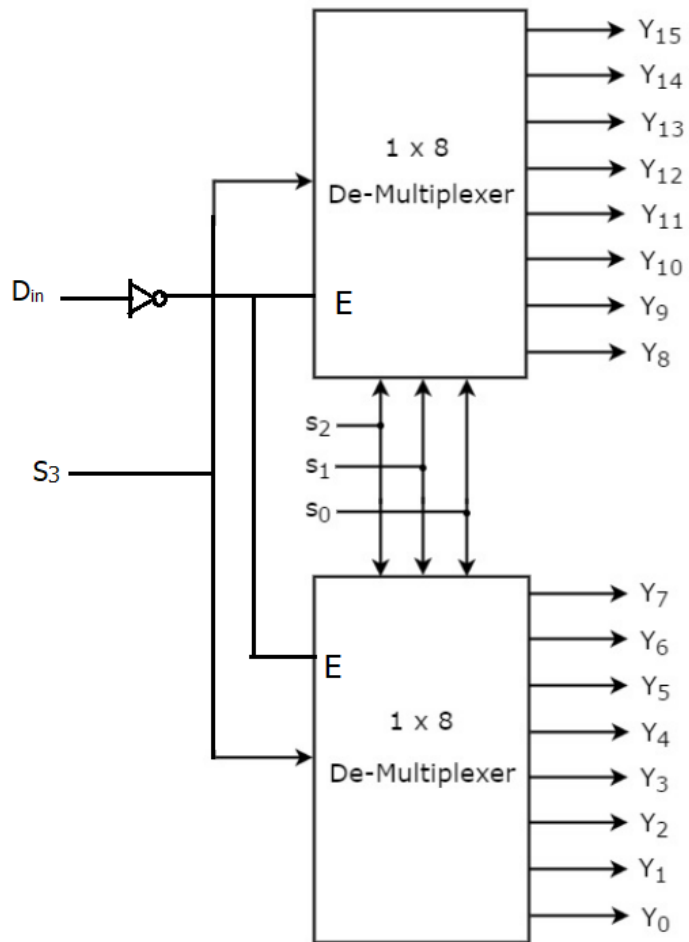


❁ From the truth table of the 1 : 8 Demux, value of first half of  $S_2$  is 0 which gives  $E=1$  so First 1 :4 Demux is enabled where the second Demux is disabled . So we get first four output.

❁ value of Second half of  $S_2$  is 1 which gives  $E=1$  so Second 1 :4 Demux is enabled where the First Demux is disabled . So we get Second four output.

❁ 1:16 Demux Designed Using Two 1:8 Demuxes:

❁ Similar to 1:8 Demux 1:16 Demux can be implemented using two 1:8 Demux.



## 6.Code Converters

- ✿ Code converters are logic circuits that implement the number conversion from one binary code to another.

- ✿ Types of Code converters:

  - ✿ Binary-to-Gray code

  - ✿ Gray-to-Binary code

  - ✿ BCD-to-Excess-3

  - ✿ Excess-3-to-BCD

  - ✿ Binary-to-BCD

  - ✿ BCD-to-binary

- ✿ Steps to design code converters:

- ✿ Construction of a truth table to meet input -output requirements.

- ✿ Determine Boolean expressions for various output variables in terms of input variables.

- ✿ Simplify Boolean expression by minimization using Karnaugh map method.

- ✿ Implement the simplified Boolean expression with minimum number of logic gates.

- ✿ Don't care conditions are considered for minimization using K-Map

  - \* Don't care conditions for BCD are ( $X'=10,11,12,13,14,15$ )

  - \* For Excess 3 code ( $X'=0,1,2,13,14,15$ )

- ✿ **6.1 Binary to Gray Code converter**

- ✿ Gray code is a reflected code with a value of a single bit between successive numbers

- ✿ Truth table for the Binary-to-Gray code converter with Binary code as input and Gray Code as output



## ✿ Truth Table

Decimal	Binary code				Gray code			
	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	G <sub>3</sub>	G <sub>2</sub>	G <sub>1</sub>	G <sub>0</sub>
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

## ✿ K-Map Minimization

✿ From the truth table, the logic expression for the binary code outputs obtained are:

✿  $G_3 = \sum m(8, 9, 10, 11, 12, 13, 14, 15)$

✿  $G_2 = \sum m(4, 5, 6, 7, 8, 9, 10, 11)$

✿  $G_1 = \sum m(2, 3, 4, 5, 10, 11, 12, 13)$

✿  $G_0 = \sum m(1, 2, 5, 6, 9, 10, 13, 14)$

**K- Map for  $G_3$**

B <sub>3</sub> B <sub>2</sub>		B <sub>1</sub> B <sub>0</sub>			
		00	01	11	10
00		0	0	0	0
01		0	0	0	0
11		1	1	1	1
10		1	1	1	1

$$G_3 = B_3$$

**K- Map for  $G_2$**

B <sub>3</sub> B <sub>2</sub>		B <sub>1</sub> B <sub>0</sub>			
		00	01	11	10
00		0	0	0	0
01		1	1	1	1
11		0	0	0	0
10		1	1	1	1

$$G_2 = B_3'B_2 + B_3B_2'$$

$$= B_2 \oplus B_3$$

**K-Map for  $G_1$**

$B_1 B_0$		00	01	11	10
$B_3 B_2$	00	0	0	1	1
	01	1	1	0	0
	11	1	1	0	0
	10	0	0	1	1

$$G_1 = B_2'B_1 + B_2B_1'$$

$$\mathbf{G_1 = B_2 \oplus B_1}$$

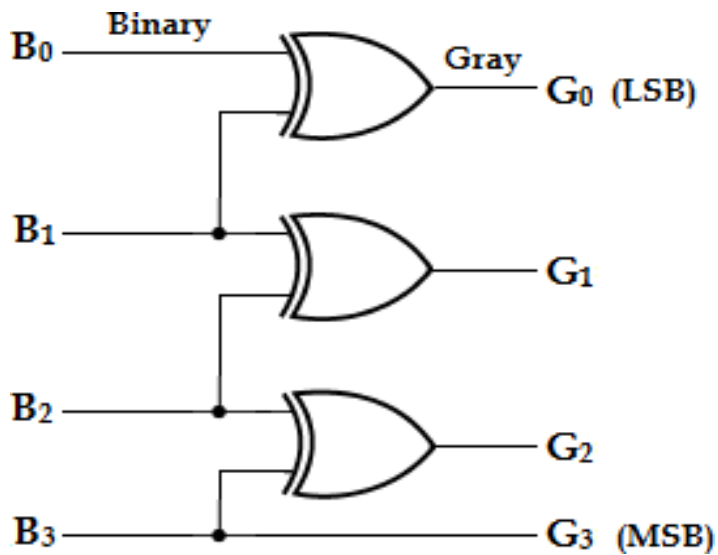
**K-Map for  $G_0$**

$B_1 B_0$		00	01	11	10
$B_3 B_2$	00	0	1	0	1
	01	0	1	0	1
	11	0	1	0	1
	10	0	1	0	1

$$G_0 = B_1'B_0 + B_1B_0'$$

$$\mathbf{G_0 = B_1 \oplus B_0}$$

### ✿ Logic Diagram



## ❁ 6.2 Gray to Binary Code converter

- ❁ The same procedure is followed to obtain the circuit for Gray to Binary code conversion.

### ❁ Truth Table

Decimal	Gray code				Binary code			
	G <sub>3</sub>	G <sub>2</sub>	G <sub>1</sub>	G <sub>0</sub>	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	0	1	1	0
6	0	1	1	0	0	1	0	0
7	0	1	1	1	0	1	0	1
8	1	0	0	0	1	1	1	1
9	1	0	0	1	1	1	1	0
10	1	0	1	0	1	1	0	0
11	1	0	1	1	1	1	0	1
12	1	1	0	0	1	0	0	0
13	1	1	0	1	1	0	0	1
14	1	1	1	0	1	0	1	1
15	1	1	1	1	1	0	1	0

- ❁ From the truth table, the logic expression for the binary code outputs obtained are:

- ❁  $B_3 = \sum m(8, 9, 10, 11, 12, 13, 14, 15)$

- ❁  $B_2 = \sum m(4, 5, 6, 7, 8, 9, 10, 11)$

- ❁  $B_1 = \sum m(2, 3, 4, 5, 8, 9, 14, 15)$

- ❁  $B_0 = \sum m(1, 2, 4, 7, 8, 11, 13, 14)$

✿ K-Map Minimization

**K-Map for  $B_3$**

$G_3 \ G_2$		$G_1 \ G_0$			
		0 0	0 1	1 1	1 0
0 0		0	0	0	0
0 1		0	0	0	0
1 1		1	1	1	1
1 0		1	1	1	1

$$B_3 = G_3$$

**K-Map for  $B_2$**

$G_3 \ G_2$		$G_1 \ G_0$			
		0 0	0 1	1 1	1 0
0 0		0	0	0	0
0 1		1	1	1	1
1 1		0	0	0	0
1 0		1	1	1	1

$$B_2 = G_3'G_2 + G_3G_2'$$

$$B_2 = G_2 \oplus G_3 = G_2 \oplus B_3$$

**K-Map for  $B_1$**

$G_3 \ G_2$		$G_1 \ G_0$			
		0 0	0 1	1 1	1 0
0 0		0	0	1	1
0 1		1	1	0	0
1 1		0	0	1	1
1 0		1	1	0	0

**K-Map for  $B_0$**

$G_3 \ G_2$		$G_1 \ G_0$			
		0 0	0 1	1 1	1 0
0 0		0	1	0	1
0 1		1	0	1	0
1 1		0	1	0	1
1 0		1	0	1	0

$$\text{✿ } B_1 = G_3'G_2'G_1 + G_3'G_2G_1' + G_3G_2G_1 + G_3G_2'G_1'$$

$$= G_3' (G_2'G_1 + G_2G_1') + G_3 (G_2G_1 + G_2'G_1')$$

$$= G_3' (G_2 \oplus G_1) + G_3 (G_2 \oplus G_1)'$$

$$\text{Since } [A \oplus B = A'B + AB'], [(A \oplus B)' = AB + A'B']$$

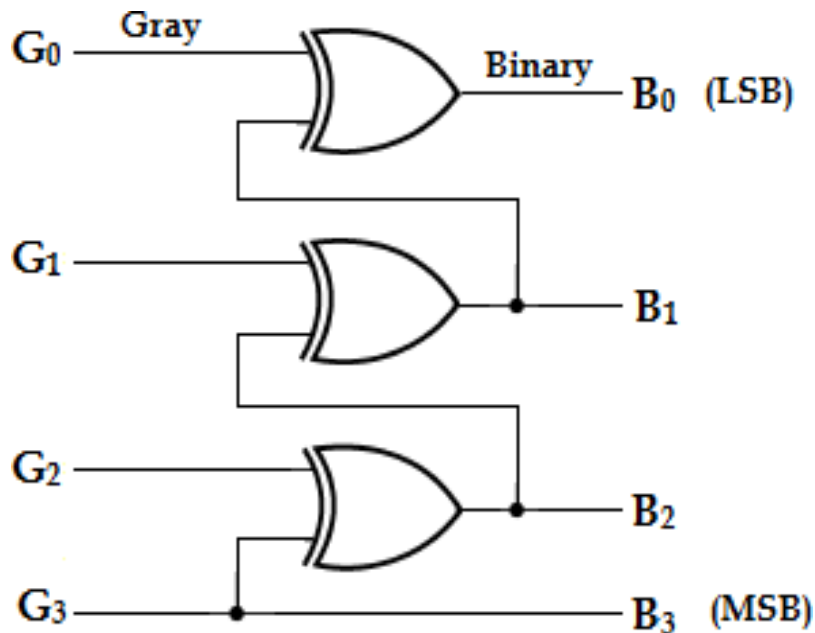
$$\text{✿ } B_1 = G_3 \oplus G_2 \oplus G_1$$

$$\text{✿ } B_1 = B_2 \oplus G_1$$

$$\begin{aligned}
 * B_0 &= G_3'G_2'G_1'G_0 + G_3'G_2'G_1G_0' + G_3G_2G_1'G_0 + G_3G_2G_1G_0' + G_3'G_2G_1'G_0' + \\
 &\quad G_3G_2'G_1'G_0' + G_3'G_2G_1G_0 + G_3G_2'G_1G_0 \\
 * &= G_3'G_2' (G_1'G_0 + G_1G_0') + G_3G_2 (G_1'G_0 + G_1G_0') + G_1'G_0' (G_3'G_2 + G_3G_2') + G_1G_0 (G_3'G_2 + G_3G_2') \\
 * &= G_3'G_2' (G_0 \oplus G_1) + G_3G_2 (G_0 \oplus G_1) + G_1'G_0' (G_2 \oplus G_3) + G_1G_0 (G_2 \oplus G_3) \\
 * &= G_0 \oplus G_1 (G_3'G_2' + G_3G_2) + G_2 \oplus G_3 (G_1'G_0' + G_1G_0) \\
 * &= (G_0 \oplus G_1) (G_2 \oplus G_3)' + (G_2 \oplus G_3) (G_0 \oplus G_1) \\
 * \mathbf{B_0} &= \mathbf{(G_0 \oplus G_1) \oplus (G_2 \oplus G_3)} \\
 * \mathbf{B_0} &= \mathbf{G_0 \oplus B_1}
 \end{aligned}$$

### \* Logic Diagram:

- \* The logic diagram for the output binary code expressions can be implemented using XOR gates



### ❁ 6.3 BCD to Excess-3 Converter

❁ Excess-3 code is a modified binary code derived from the BCD code by adding 3 to each coded number.

❁ Truth Table:

Decimal	BCD code				Excess-3 code			
	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	E <sub>3</sub>	E <sub>2</sub>	E <sub>1</sub>	E <sub>0</sub>
<b>0</b>	0	0	0	0	0	0	1	1
<b>1</b>	0	0	0	1	0	1	0	0
<b>2</b>	0	0	1	0	0	1	0	1
<b>3</b>	0	0	1	1	0	1	1	0
<b>4</b>	0	1	0	0	0	1	1	1
<b>5</b>	0	1	0	1	1	0	0	0
<b>6</b>	0	1	1	0	1	0	0	1
<b>7</b>	0	1	1	1	1	0	1	0
<b>8</b>	1	0	0	0	1	0	1	1
<b>9</b>	1	0	0	1	1	1	0	0

❁ The logic expression for the Excess-3 code outputs from the truth table are:

$$❁ E_3 = \sum m (5, 6, 7, 8, 9) + \sum d (10, 11, 12, 13, 14, 15)$$

$$❁ E_2 = \sum m (1, 2, 3, 4, 9) + \sum d (10, 11, 12, 13, 14, 15)$$

$$❁ E_1 = \sum m (0, 3, 4, 7, 8) + \sum d (10, 11, 12, 13, 14, 15)$$

$$❁ E_0 = \sum m (0, 2, 4, 6, 8) + \sum d (10, 11, 12, 13, 14, 15)$$

🌀 Minimization using K-Map

**K- Map for  $E_3$**

$B_3 B_2$		$B_1 B_0$			
		00	01	11	10
00		0	0	0	0
01		0	1	1	1
11		x	x	x	x
10		1	1	x	x

$$E_3 = B_3 + B_2(B_0 + B_1)$$

**K- Map for  $E_2$**

$B_3 B_2$		$B_1 B_0$			
		00	01	11	10
00		0	1	1	1
01		1	0	0	0
11		x	x	x	x
10		0	1	x	x

$$E_2 = B_2 B_1' B_0' + B_2' (B_0 + B_1)$$

**K- Map for  $E_1$**

$B_3 B_2$		$B_1 B_0$			
		00	01	11	10
00		1	0	1	0
01		1	0	1	0
11		x	x	x	x
10		1	0	x	x

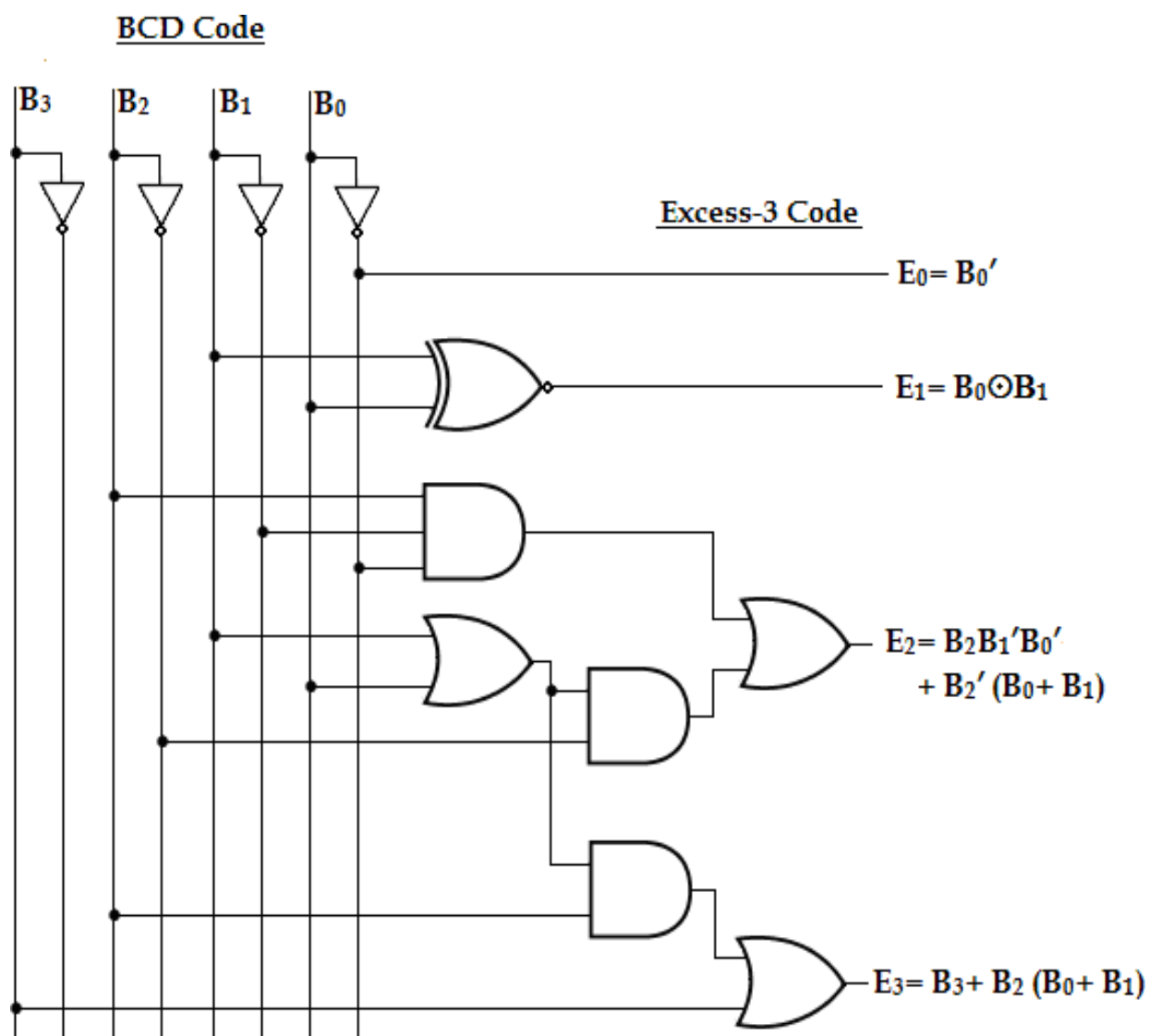
$$E_1 = B_1' B_0' + B_1 B_0$$

**K- Map for  $E_0$**

$B_3 B_2$		$B_1 B_0$			
		00	01	11	10
00		1	0	0	1
01		1	0	0	1
11		x	x	x	x
10		1	0	x	x

$$E_0 = B_0'$$

## ❁ Logic Diagram





#### ❁ 6.4 Excess-3 to BCD Converter

❁ The same procedure is followed for the implementation of the Excess 3 to BCD converter

❁ Truth Table

Decimal	Excess-3 code				BCD code			
	E3	E2	E1	E0	B3	B2	B1	B0
<b>3</b>	0	0	1	1	0	0	0	0
<b>4</b>	0	1	0	0	0	0	0	1
<b>5</b>	0	1	0	1	0	0	1	0
<b>6</b>	0	1	1	0	0	0	1	1
<b>7</b>	0	1	1	1	0	1	0	0
<b>8</b>	1	0	0	0	0	1	0	1
<b>9</b>	1	0	0	1	0	1	1	0
<b>10</b>	1	0	1	0	0	1	1	1
<b>11</b>	1	0	1	1	1	0	0	0
<b>12</b>	1	1	0	0	1	0	0	1

❁ The logic expression for the Excess-3 code outputs are:

❁  $B3 = \sum m(11, 12) + \sum d(0, 1, 2, 13, 14, 15)$

❁  $B2 = \sum m(7, 8, 9, 10) + \sum d(0, 1, 2, 13, 14, 15)$

❁  $B1 = \sum m(5, 6, 9, 10) + \sum d(0, 1, 2, 13, 14, 15)$

❁  $B0 = \sum m(4, 6, 8, 10, 12) + \sum d(0, 1, 2, 13, 14, 15)$

🌀 Minimization using K-Map

**K- Map for  $B_3$**

$E_3 E_2$		$E_1 E_0$			
		00	01	11	10
00		x	x	0	x
01		0	0	0	0
11		1	x	x	x
10		0	0	1	0

$$B_3 = E_3 E_2 + E_3 E_1 E_0$$

**K- Map for  $B_2$**

$E_3 E_2$		$E_1 E_0$			
		00	01	11	10
00		x	x	0	x
01		0	0	1	0
11		0	x	x	x
10		1	1	0	1

$$B_2 = E_2' E_1' + E_2 E_1 E_0 + E_3 E_1 E_0'$$

**K- Map for  $B_1$**

$E_3 E_2$		$E_1 E_0$			
		00	01	11	10
00		x	x	0	x
01		0	1	0	1
11		0	x	x	x
10		0	1	0	1

$$B_1 = E_1' E_0' + E_1 E_0'$$

**K- Map for  $B_0$**

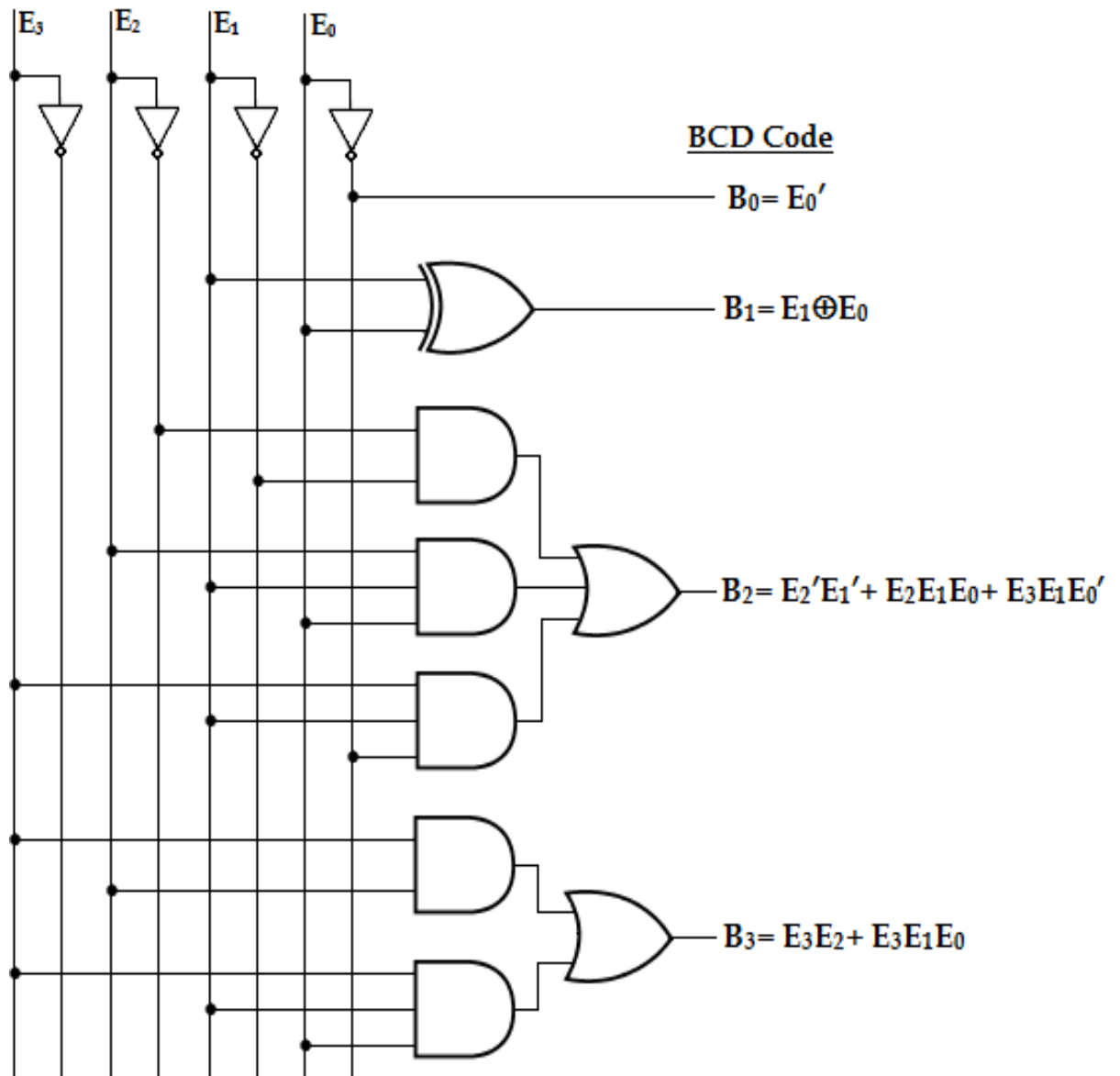
$E_3 E_2$		$E_1 E_0$			
		00	01	11	10
00		x	x	0	x
01		1	0	0	1
11		1	x	x	x
10		1	0	0	1

$$B_0 = E_0'$$

## ❁ Logic Diagram

- ❁ The logic diagram for the output expressions obtained can be implemented as follows:

### Excess-3 Code

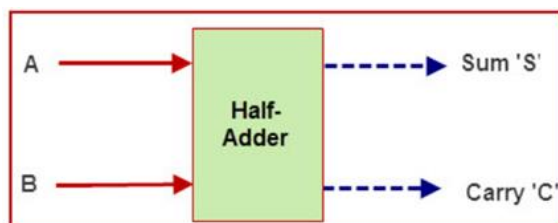


## 7. ADDERS

### 7.1 HALF ADDER

- A combinational circuit that performs the addition of two bits is called a half adder
- It has two input A and B, two output Sum, S and Carry, C.

Block Diagram:



Truth Table:

INPUT		OUTPUT	
A	B	Sum ,S	Carry , C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$\text{Sum, } S = \sum m(1,2)$$

$$\text{Carry, } C = \sum m(2)$$

K-Map for Sum

A \ B	0	1
0	0	1
1	1	0

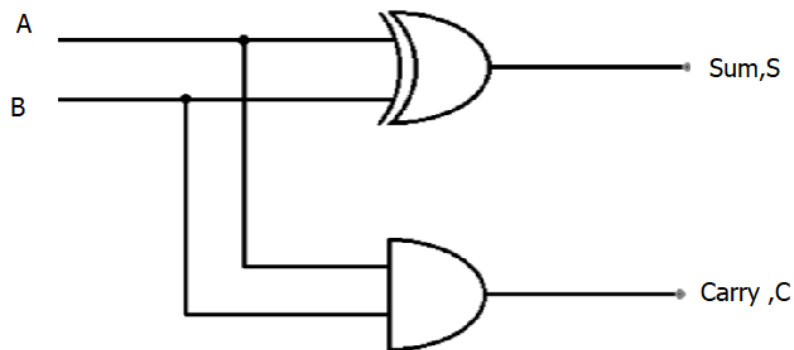
$$S = \overline{A}B + A\overline{B} = A \oplus B$$

K-Map for Carry

A \ B	0	1
0	0	0
1	0	1

$$C = AB$$

### ❁ Logic Diagram:



## 7.2 FULL ADDER

- ❁ A full adder is a combinational circuit that forms the arithmetic sum of three bits.
- ❁ It consists of three inputs and two outputs.

### ❁ Truth Table:

INPUT			OUTPUT	
A	B	Carry In ,C <sub>in</sub>	Sum	Carry Out , Cout
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

❁  $\text{Sum, } S = \sum m(1, 2, 4, 7)$

❁  $\text{Carry Out, } C_{out} = \sum m(3, 5, 6, 7)$

### ❁ K-Map for Sum:

		B Cin			
		00	01	11	10
A	0		1		1
		0	1	3	2
1	1	1		1	
		4	5	7	6

$$S = \bar{A} \bar{B} C_{in} + \bar{A} B \bar{C}_{in} + A \bar{B} C_{in} + A B C_{in}$$

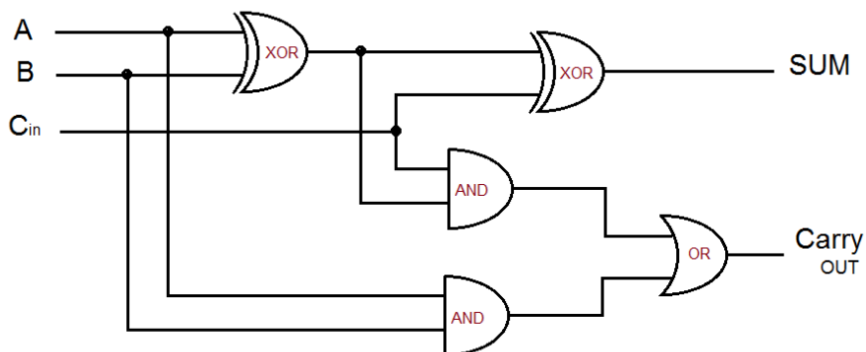
$$S = A \oplus B \oplus C_{in}$$

### ❁ K-Map for Carry Out, Cout:

		B Cin			
		00	01	11	10
A	0			1	
		0	1	3	2
1	1		1	1	1
		4	5	7	6

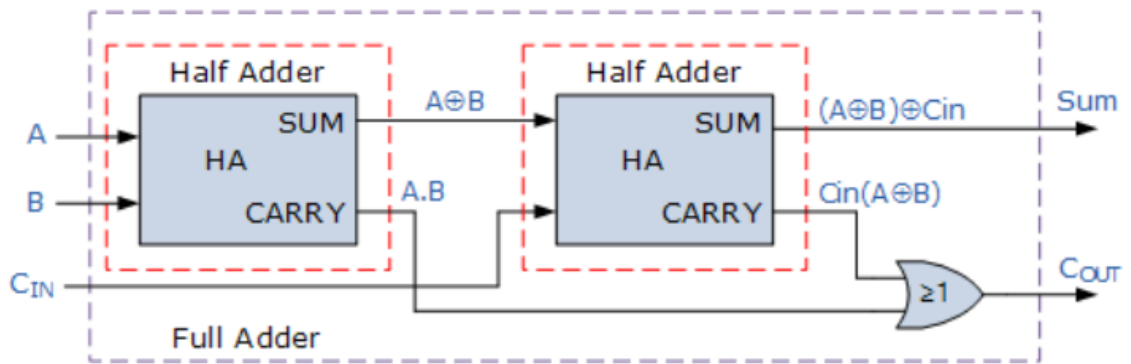
$$C_{out} = AB + BC_{in} + AC_{in}$$

### ❁ Logic Diagram:



## Full Adder Using Half Adder :

❁ Two Half Adders can be connected as shown below to produce full adder output.



From the Truth Table:

$$S = \bar{A} \bar{B} C_{in} + \bar{A} B \bar{C}_{in} + A \bar{B} C_{in} + A B C_{in}$$

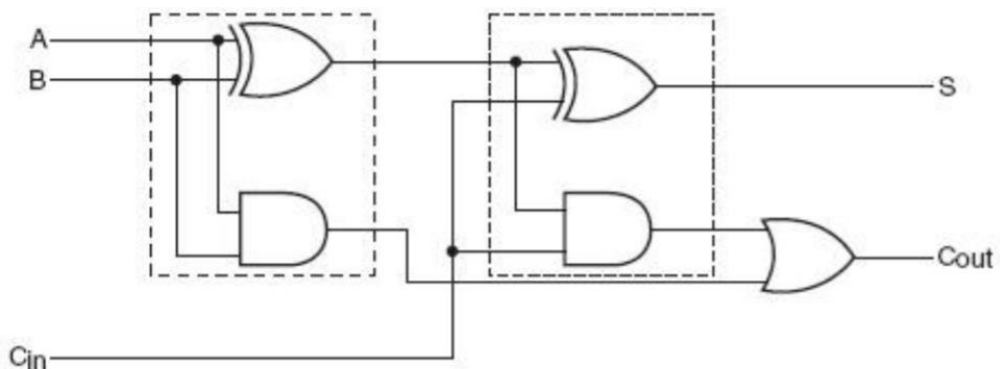
$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = A' B C_{in} + A B' C_{in} + A B C_{in}' + A B C_{in}$$

$$C_{out} = C_{in} (A' B + A B') + A B (C_{in}' + C_{in})$$

$$C_{out} = C_{in} (A \oplus B) + A B$$

❁ Logic Diagram:



## 7.3 Half Subtractor

✿ A combinational circuit that performs the subtraction of two bits is called a half subtractor.

✿ It has two input A and B, two output Difference, D and Borrow, B.

✿ Truth Table:

INPUT		OUTPUT	
A	B	Difference , D	Borrow ,B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Difference,  $D = \sum m(1, 2)$

Borrow , $B = \sum m(1)$

✿ K-Map for Difference:

A \ B	0	1
0	0	1
1	1	0

$$D = A' B + A B'$$

$$D = A \oplus B$$

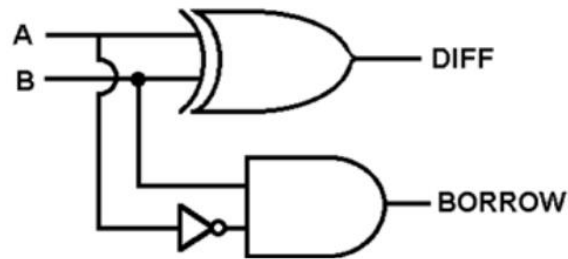
✿ K-Map for Borrow:

A \ B	0	1
0	0	1
1	0	0

$$\text{Borrow, } B = A' B$$

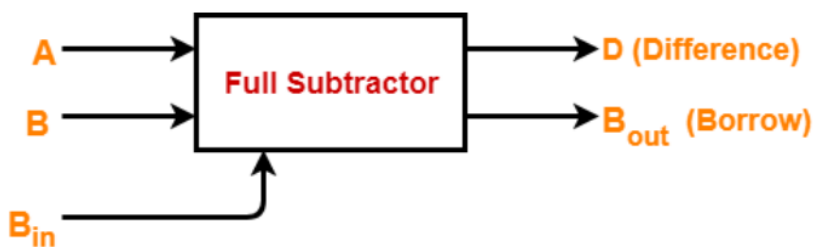


### ❁ Logic Diagram:



### ❁ 7.4 Full Subtractor:

- ❁ Full Subtractor is a combinational logic circuit used for the purpose of subtracting two single bit numbers.
- ❁ It also takes into consideration borrow of the lower significant stage.
- ❁ Thus, full subtractor has the ability to perform the subtraction of three bits.
- ❁ Full subtractor contains 3 inputs and 2 outputs (Difference and Borrow) as shown-



❁ Truth Table:

INPUT			OUTPUT	
A	B	Borrow In, Bin	Difference, D	Borrow Out, Bout
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

Difference,  $D = \sum m(1, 2, 4, 7)$

Borrow Out,  $Bout = \sum m(1, 2, 3, 7)$

❁ K-Map For Difference, D :

A \ B Bin					
		00	01	11	10
0			1		1
1		1		1	

$$\begin{aligned}
 \text{Difference, } D &= A' B' \text{Bin} + A' B \text{Bin}' + A B' \text{Bin}' + A B \text{Bin} \\
 &= (A' B' + A B) \text{Bin} + (A' B + A B') \text{Bin}' \\
 &= A \oplus B \oplus \text{Bin}
 \end{aligned}$$

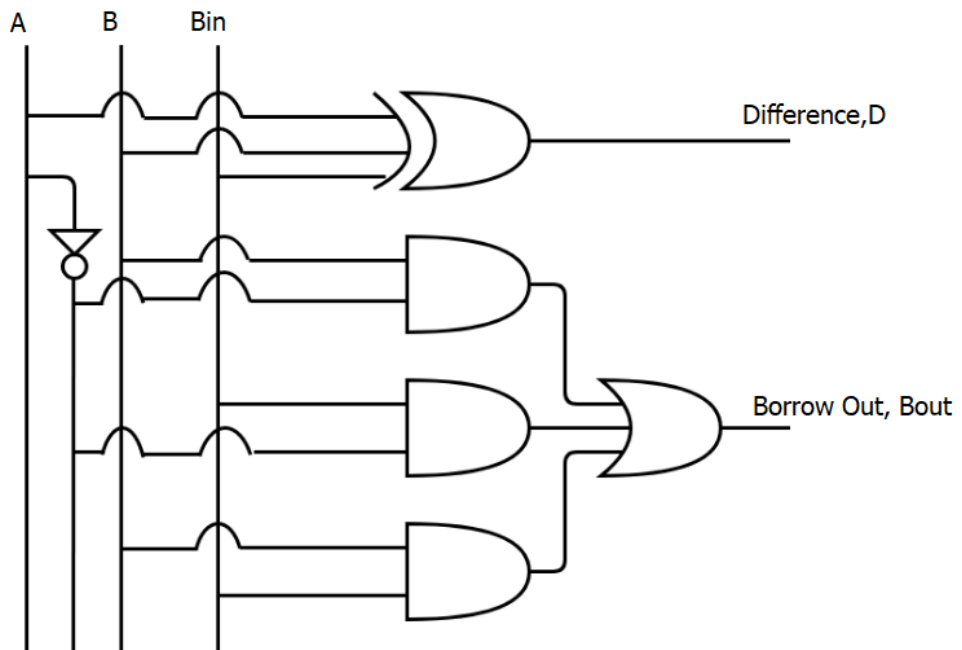
### ❁ K-Map For Borrow Out, Bout :

A \ B Bin	00	01	11	10
0		1	1	1
1			1	

Red numbers in the original image: 0, 1, 2, 3, 4, 5, 6, 7

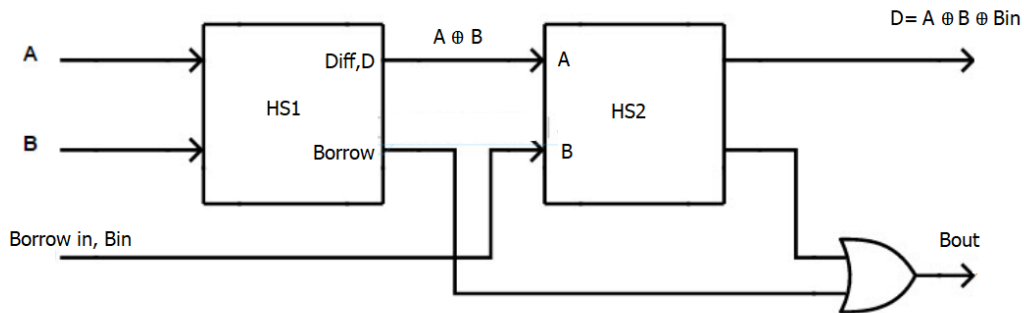
$$\text{Bout} = A' \text{Bin} + A' B + B \text{Bin}$$

### ❁ Logic Diagram For Full Subtractor:



# Full Subtractor using Half Subtractor:

## ❁ Block Diagram:

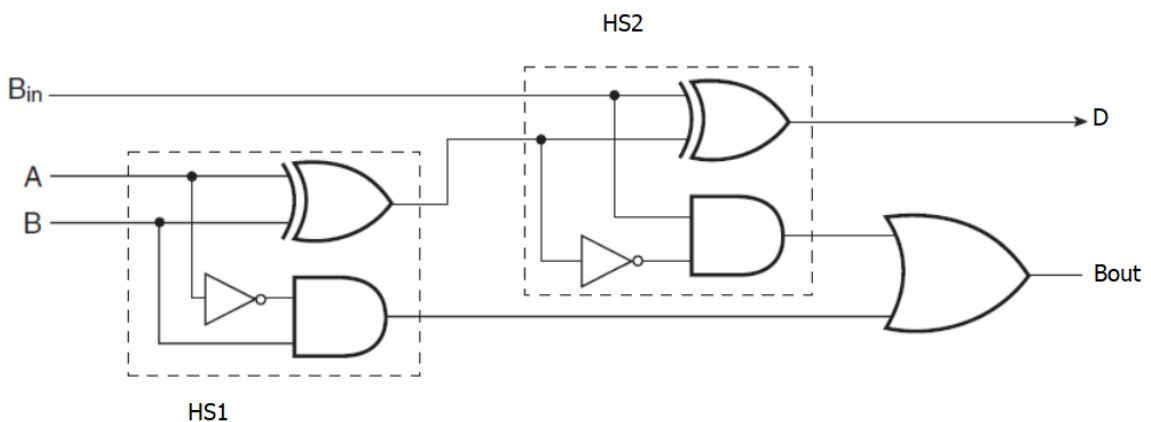


## ❁ From The Truth Table:

$$\begin{aligned} \text{Difference, } D &= A' B' \text{Bin} + A' B \text{Bin}' + A B' \text{Bin}' + A B \text{Bin} \\ &= (A' B' + A B) \text{Bin} + (A' B + A B') \text{Bin}' \\ &= A \oplus B \oplus \text{Bin} \end{aligned}$$

$$\begin{aligned} \text{Borrow Out, } \text{Bout} &= A' B' \text{Bin} + A' B \text{Bin} + A' B \text{Bin}' + A B \text{Bin} \\ &= (A' B' + A B) \text{Bin} + A' B (\text{Bin} + \text{Bin}') \\ &= (A \oplus B) \text{Bin} + A' B \end{aligned}$$

## ❁ Logic Diagram:



## 7.5 Four Bit Parallel Binary Adder/Subtractor: (Ripple Carry Adder)

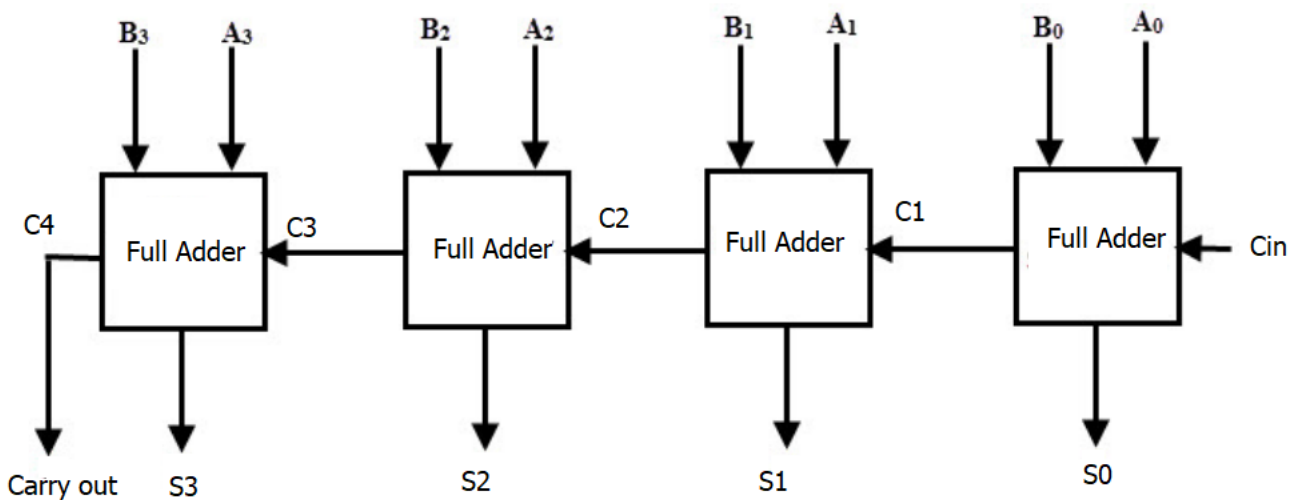
### ❁ 4 bit parallel Binary Adder:

- ❁ A 4 bit adder using Full Adder is capable of adding 4 bit numbers resulting in 4 bit sum and a carry output.
- ❁ Here all bits of Augend and Addend are fed simultaneously, hence named as parallel adder.

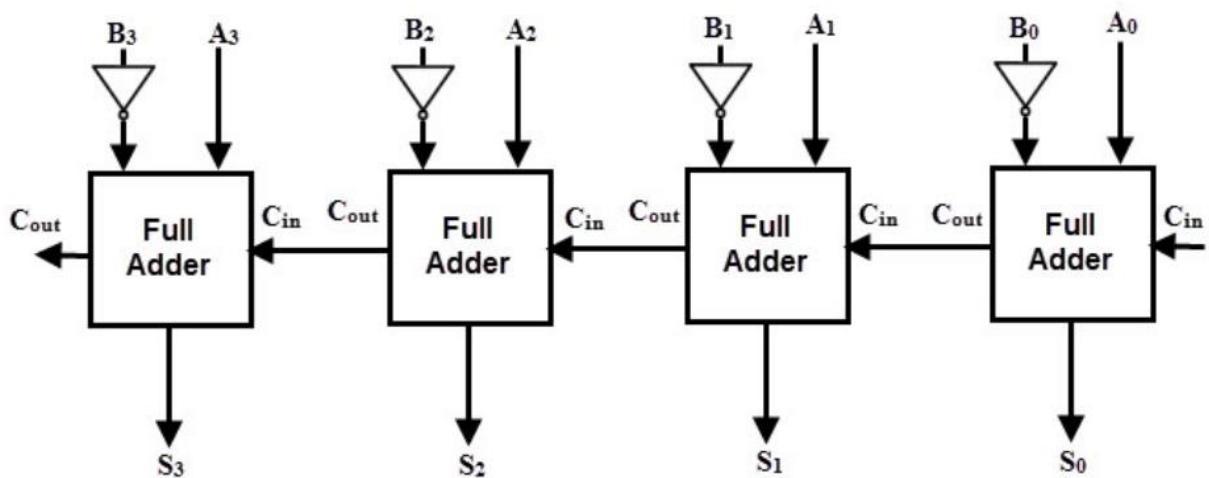
$$A = A_3 A_2 A_1 A_0$$

$$B = B_3 B_2 B_1 B_0$$

$$S = S_3 S_2 S_1 S_0$$



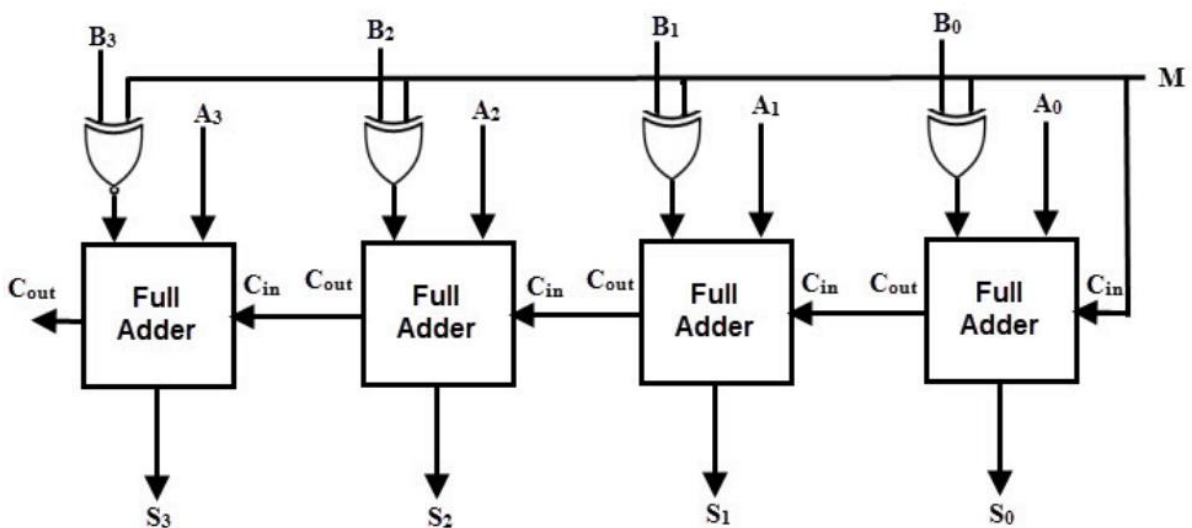
### 4 bit parallel Binary Subtractor:



- ✿ If  $C_{in} = 0$ , it performs 1's Complement Subtraction
- ✿ If  $C_{in} = 1$ , it performs 2's Complement Subtraction

#### ✿ 4 bit parallel Binary Adder/Subtractor:

- ✿ This performs both addition and subtraction of 4 bit numbers with single hardware.



#### ✿ Binary Addition:

$$1 \oplus B = 1.\bar{B} + 0.B = \bar{B}$$

$$0 \oplus B = 0.\bar{B} + 1.B = B$$

- ✿ when  $M=0$ , the circuit becomes adder.
- ✿  $B \oplus 0 = B$ .
- ✿ full adders add the B with A with carry input zero and hence an addition operation is performed.

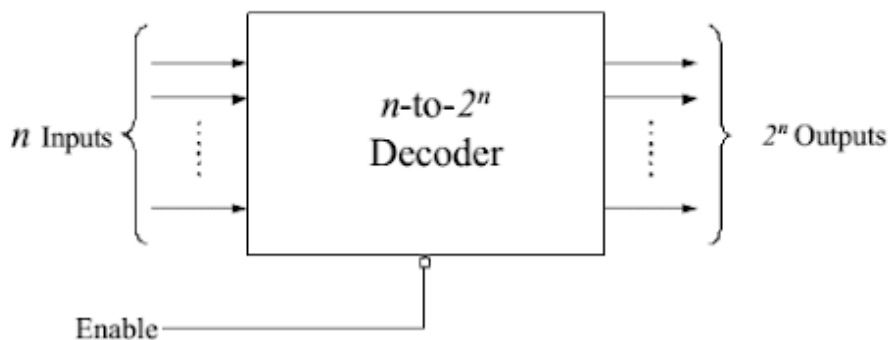
#### ✿ Binary Subtraction:

- ✿ When  $M= 1$ , the circuit is a subtractor.
- ✿  $B \oplus 1 = B'$
- ✿ Hence the complemented B inputs are added to A .
- ✿ If  $C_{in} = 0$ , it performs 1's Complement Subtraction
- ✿ If  $C_{in} = 1$ , it performs 2's Complement Subtraction

## 8. DECODER

### DECODER:

- A Decoder is a combinational circuit converts binary information from  $n$  input lines to  $2^n$  unique output lines. If  $n$  bit coded information has some unused combinations, the decoder may have fewer than  $2^n$  outputs.



Block diagram of decoder

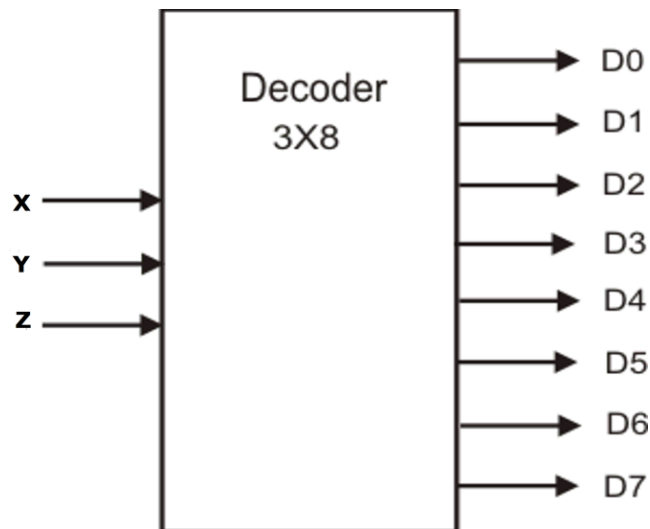
### Decoder configurations

- 2 to 4 decoder
- 3 to 8 decoder
- 4 to 16 decoder etc.,

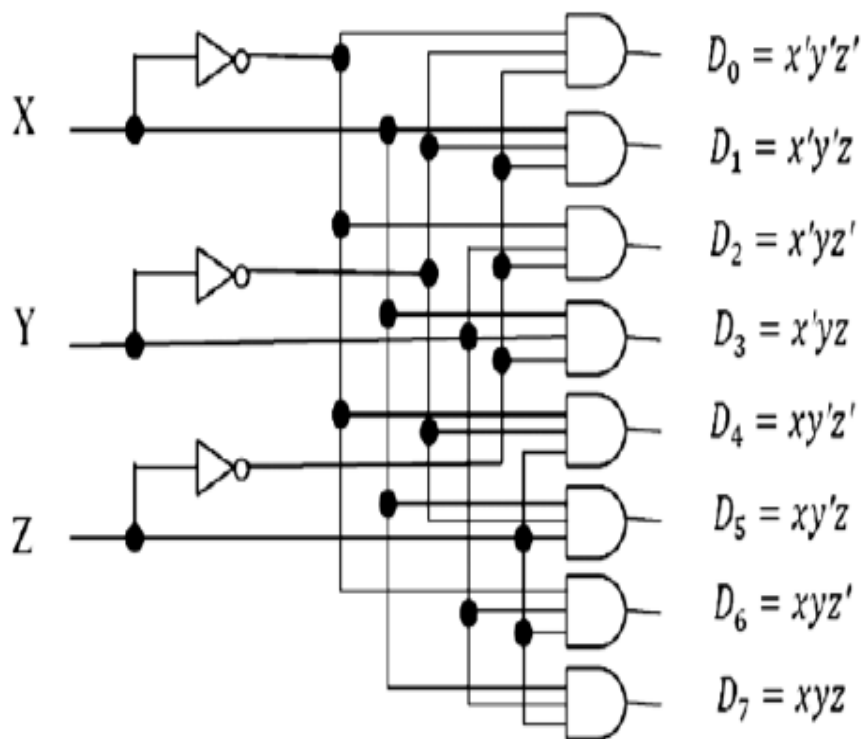
#### ❖ 3 to 8 line decoder

Three to eight line decoder is also called as binary to octal converter. The input variables represents a binary number and the output represents the eight digits of a number in octal in the octal system. However, a three-to-eight-line decoder can be used for decoding any three-bit code to provide eight outputs, one for each element of the code.

### Three to Eight line Decoder



Block diagram of 3x8 Decoder



Circuit Diagram of 3X8 Decoder

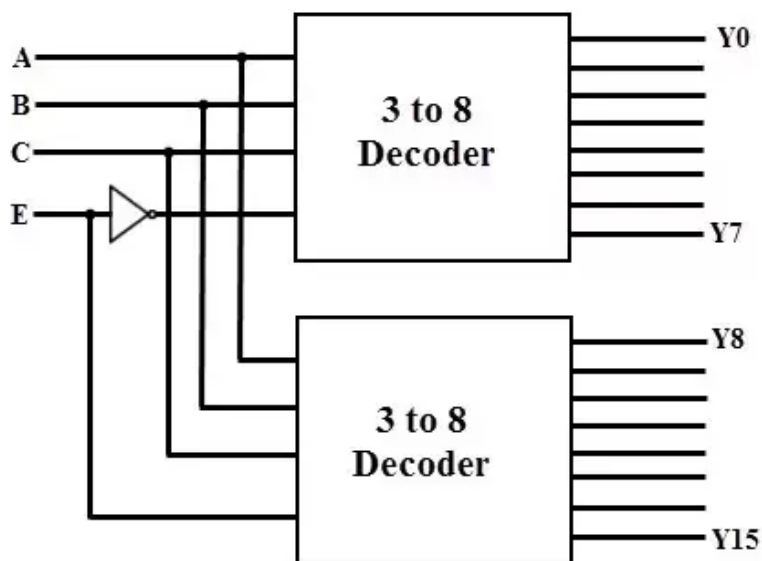


INPUTS			OUTPUTS							
X	Y	Z	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Truth Table of 3X8 decoder

The other configurations can be obtained in a similar method. A decoder with enable input can function as a demultiplexer circuit that receives information from a single line and directs it to one of  $2^n$  possible output lines. The selection of a specific output is controlled by the bit combination of  $n$  selection lines. Decoder and demultiplexer operations are obtained from the same circuit. a decoder with an enable input is referred as a demultiplexer.

Decoders with enable inputs can be connected together to form a larger decoder circuit. the figure shows two 3 to 8 line decoders with enable inputs connected to form a 4 to 16 line decoder. When  $E = 0$ , the top decoder is enabled and the other is disabled. The bottom decoder outputs are all 0's. and the top eight outputs generate minterms 0000 to 0111. When  $E = 1$ , the enable conditions are reversed. The bottom decoder outputs generate minterms from 1000 to 1111, while the outputs of the top decoder are all 0's. In general enable inputs are a convenient feature for interconnecting two or more standard components for the purpose of combining them into a similar function with more inputs and outputs.



4 to 16 decoder using 3 to 8 decoder

## Combinational Logic Implementation with Decoder:

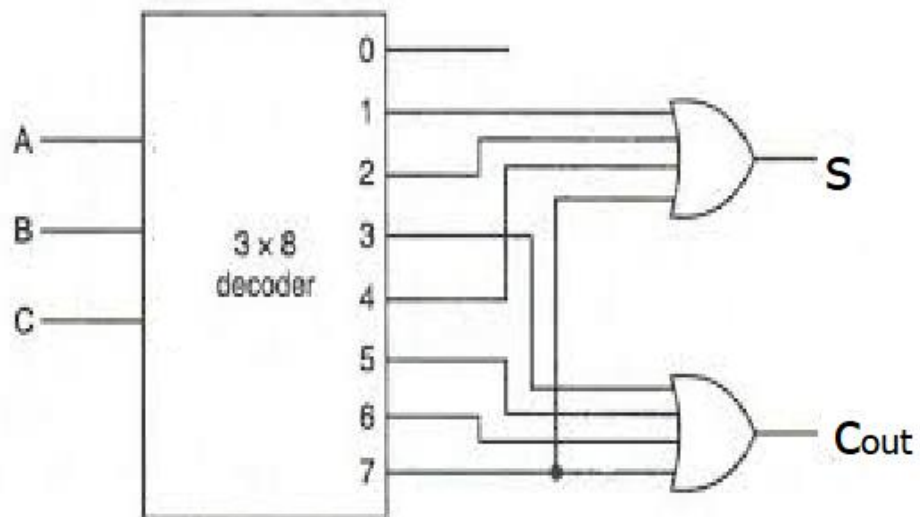
A decoder provides the  $2^n$  minterms of  $n$  input variables. Since any Boolean function can be expressed in sum-of-minterms form a decoder that generates the minterms of the function, together with an external OR gate that forms their logical sum, provides a hardware implementation of the function. In this way, any combinational circuit with  $n$  inputs and  $m$  outputs can be implemented with an  $n$  to  $2^n$  line decoder and  $m$  number of OR gates.

### Implementation of Full adder using Decoder

From the truth table of full adder minterms of sum and output carry are as follows

$$S(A,B,C) = \sum m(1,2,4,7)$$

$$C_{out}(A,B,C) = \sum m(3,5,6,7)$$



Full Adder Implementation with decoder

# 9. ENCODER

## Octal to binary encoder (or) 8 to 3 encoder

An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has  $2^n$ (or fewer) input lines and n output lines. The output lines generate the binary code corresponding to the input value. An example of an encoder is the octal-to-binary encoder whose truth table is given in table. It has eight inputs (one for each of the octal digits) and three outputs that generate the corresponding binary number. It is assumed that only one input has a value of 1 at any given time. The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output z is equal to 1 when the input octal digit is 1, 3, 5 or 7. Output y is 1 for octal digits 2, 3, 6 or 7. and output x is 1 for digits 4, 5, 6 or 7.

INPUTS								OUTPUTS		
D0	D1	D2	D3	D4	D5	D6	D7	X	Y	Z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Output functions of encoder are

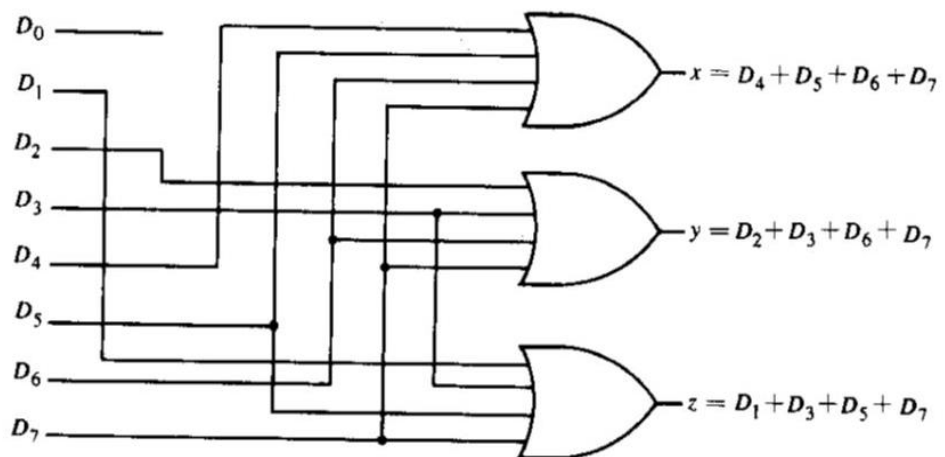
$$X = D_4 + D_5 + D_6 + D_7$$

$$Y = D_2 + D_3 + D_6 + D_7$$

$$Z = D_1 + D_3 + D_5 + D_7$$

Limitation of encoder

The encoder has the limitation that only one input can be active at any given time. If two inputs are active simultaneously, the output produces an undefined combination. Another ambiguity in the octal-to-binary encoder is that an output with all 0's is generated when all the inputs are 0; but this output is the same as when  $D_0$  is equal to 1. The discrepancy can be resolved by providing one more output to indicate whether at least one input is equal to 1.



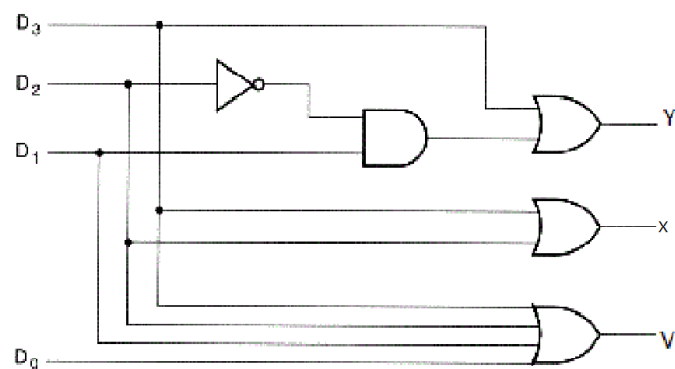
Octal to binary encoder

# Priority Encoder

A priority encoder is an encoder circuit that includes the priority function. The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence. The truth table of a four-input priority encoder is given in table , In addition to the two outputs x and y, the circuit has a third output designated by V, this is a valid bit indicator that is set to 1 when one or more inputs are equal to 1.If all input s are 0, there is no valid input and V is equal to 0. The other two outputs are not inspected when v equals 0 and are specified as don't -care conditions.

INPUTS				OUTPUTS		
D0	D1	D2	D3	X	Y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Truth table of priority encoder



Priority Encoder

## 9. Lecture Notes

### ✿ E-Books

Digital Fundamentals\_ Global Ed - Thomas L Floyd

Digital Principles And Application - Leach & Malvino

Digital Design - M. Morris Mano and Michael D. Ciletti

Fundamentals of Digital Logic with Verilog Design-Stephen Brown and Zonko Vranesic

### ✿ ONLINE LEARNING MATERIALS:

**1:** [http://nptel.iitm.ac.in/video.php? subject Id=117106086](http://nptel.iitm.ac.in/video.php?subject%20Id=117106086)

**2:** <http://nptel.iitm.ac.in/courses/117101001>

**3:** <https://youtu.be/C-oAyXibnJU>

**4** <https://youtu.be/oYRMYSIVj1o>

**5:** <https://www.youtube.com/watch?v=XZmGGAbHqa0>

**6:** <https://www.youtube.com/watch?v=KymIDyQiXZI>

### ✿ Video Links

<https://drive.google.com/open?id=1qCP5dBvi1LZxd6S7FAUxt788iMmLLpbq>

<https://drive.google.com/open?id=1hrXWfXKWnhidFbnDPDtd75hQt9OzOG6O>

## 10. Assignments

- ✿ Design a combinational circuit with 4 inputs and 2 outputs. The output is 01 whenever the input binary value is greater than 10, 11 whenever the input binary value is between 6 to 9, 10 whenever the input binary value is less than 4 and 00 for remaining conditions.
  - ✿ Design a combinational circuit with two inputs and four outputs. The output binary number should be the square of the input binary number
  - ✿ Design a combinational circuit with 4 inputs and 2 outputs. The output is 01 whenever the input binary value is greater than 10, 11 whenever the input binary value is between 6 to 9, 10 whenever the input binary value is less than 4 and 00 for remaining conditions
  - ✿ Design a 3 input combinational circuit whose output is equal to 1 if the input variables have more number of 1's than 0's. The output is '0' otherwise
  - ✿ The input to a combinational logic circuit is a 4 bit binary number .Design the logic circuit with minimum hardware for the following (i)Output  $Y_1=1$ , if input binary number is 5 or less than 5 (ii)Output  $Y_2=1$ ,if input binary number is 9 or more than 9
  - ✿ Reduce the given function using K-Map and draw the logic circuit using NAND logic.
- $F(A, B, C, D, E) = \sum m(6, 9, 13, 18, 19, 25, 27, 29, 31) + d(2, 3, 11, 15, 17, 24, 28)$
- ✿ Reduce the following function using K-map and draw the logic circuit using only NOR gates  $F(A, B, C, D) = \prod M(0, 2, 3, 8, 9, 12, 13, 15) \cdot \prod d(1, 4, 5, 10, 14)$
  - ✿ Write the maxterms corresponding to the logical expression  $Y = (A + B + C')(A + B' + C')(A' + B' + C)$  and simplify using K-map and draw the logic circuit using NOR logic



## 11. Part A Q & A (with K level and CO)

PART A		
Questions and Answers	Blooms Level	COs
<p>1. What are combinational circuits?</p> <p>A combinational circuit consists of logic gates whose outputs at any time are determined from the present combination of inputs. A combinational circuit performs an operation that can be specified logically by a set of Boolean functions. It consists of input variables, logic gates, and output variables.</p>	K1	CO2
<p>2. Define half adder.</p> <p>A combinational circuit that performs the addition of two bits is called a half adder. A half adder needs two binary inputs and two binary outputs. The input variables designate the augend and addend bits; the output variables produce the sum and carry</p>	K1	CO2
<p>3. Define full adder?</p> <p>A combinational circuit that performs the addition of three bits is a full adder. It consists of three inputs and two outputs.</p>	K1	CO2
<p>4. Define binary adder.</p> <p>A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders constructed in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.</p>	K1	CO2
<p>5. What are decoders?</p> <p>A decoder is a combinational circuit that converts binary information from <math>n</math> input lines to a maximum of <math>2^n</math> unique output lines. If the <math>n</math> bit coded information has unused combinations, the decoder may have fewer than <math>2^n</math> outputs.</p>	K1	CO2
<p>6. What are encoders?</p> <p>An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has <math>2^n</math> and <math>n</math> output lines. The output lines generate the binary code corresponding to the input value.</p>	K1	CO2
<p>7. Define priority encoder?</p> <p>A priority encoder is an encoder circuit that includes the priority function. The operation of priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.</p>	K1	CO2

<p>8. Define multiplexer?</p> <p>A multiplexer is combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally there are <math>2^n</math> input lines and <math>n</math> selection lines whose bit combinations determine which input is selected.</p>	K1	CO2
<p>9. Define binary decoder?</p> <p>A decoder which has an <math>n</math>-bit binary input code and a one activated output out-of-<math>2^n</math> output code is called binary decoder. A binary decoder is used when it is necessary to activate exactly one of <math>2^n</math> outputs based on an <math>n</math>-bit input value.</p>	K1	CO2
<p>10. Simplify the following expression <math>Y = (A + B)(A + C')(B' + C')</math></p> $Y = (A + B)(A + C')(B' + C')$ $= (AA' + AC + A'B + BC)(B' + C') [A.A' = 0]$ $= (AC + A'B + BC)(B' + C')$ $= AB'C + ACC' + A'BB' + A'BC' + BB'C + BCC'$ $= AB'C + A'BC'$	K2	CO2
<p>11. State De Morgan's theorem.</p> <p>De Morgan suggested two theorems that form important part of Boolean algebra. They are,</p> <p>1) The complement of a product is equal to the sum of the complements.  <math>(AB)' = A' + B'</math></p> <p>2) The complement of a sum term is equal to the product of the complements.  <math>(A + B)' = A'B'</math></p>	K1	CO2
<p>12. Reduce <math>A.A'C</math></p> $A.A'C = 0.c [A.A' = 1] = 0$	K2	CO2
<p>13. Reduce <math>A(A + B)</math></p> $A(A + B) = AA + AB$ $= A(1 + B) [1 + B = 1] = A.$	K2	CO2
<p>14. Reduce <math>A'B'C' + A'BC' + A'BC</math></p> $A'B'C' + A'BC' + A'BC = A'C'(B' + B) + A'BC$ $= A'C' + A'BC [A + A' = 1]$ $= A'(C' + BC) = A'(C' + B) [A + A'B = A + B]$	K2	CO2

<p>15. Simplify the following expression <math>Y = (A + B)(A + C')(B' + C')</math></p> $  \begin{aligned}  Y &= (A + B)(A + C')(B' + C') \\  &= (AA' + AC + A'B + BC)(B' + C') [A.A' = 0] \\  &= (AC + A'B + BC)(B' + C') \\  &= AB'C + ACC' + A'BB' + A'BC' + BB'C + BCC' \\  &= AB'C + A'BC'  \end{aligned}  $	K2	CO2
<p>16. Simplify the following using De Morgan's theorem <math>[((AB)'C)'' D']'</math></p> $  \begin{aligned}  [((AB)'C)'' D']' &= ((AB)'C)'' + D' [(AB)' = A' + B'] \\  &= (AB)' C + D' \\  &= (A' + B')C + D'  \end{aligned}  $	K2	CO2
<p>17. Show that <math>(X + Y' + XY)(X + Y')(X'Y) = 0</math></p> $  \begin{aligned}  (X + Y' + XY)(X + Y')(X'Y) &= (X + Y' + X)(X + Y')(X' + Y) [A + A'B = A + B] \\  &= (X + Y')(X + Y')(X'Y) [A + A = 1] \\  &= (X + Y')(X'Y) [A.A = 1] \\  &= X.X' + Y'.X'.Y \\  &= 0 [A.A' = 0]  \end{aligned}  $	K2	CO2
<p>18. Convert the given expression in canonical SOP form</p> $  \begin{aligned}  Y &= AC + AB + BC \\  Y &= AC + AB + BC \\  &= AC(B + B') + AB(C + C') + (A + A')BC \\  &= ABC + ABC' + AB'C + AB'C' + ABC + ABC' + ABC \\  &= ABC + ABC' + AB'C + AB'C' [A + A = 1]  \end{aligned}  $	K2	CO2
<p>19. Convert the given expression in canonical POS form</p> $  \begin{aligned}  Y &= (A + B)(B + C)(A + C) \\  Y &= (A + B)(B + C)(A + C) \\  &= (A + B + C.C')(B + C + A.A')(A + B.B' + C) \\  &= (A + B + C)(A + B + C')(A + B + C)(A' + B + C)(A + B + C)(A + B' + C) [A + BC] \\  &= (A + B)(A + C) \text{ Distributive law} \\  &= (A + B + C)(A + B + C')(A' + B + C)(A' + B + C)(A + B' + C)  \end{aligned}  $	K2	CO2
<p>20. Find the minterms of the logical expression</p> $  \begin{aligned}  Y &= A'B'C' + A'B'C + A'BC + ABC' \\  Y &= A'B'C' + A'B'C + A'BC + ABC' \\  &= m_0 + m_1 + m_3 + m_6  \end{aligned}  $	K1	CO2

## 12. Part B Qs (with K level and CO)

PART B		
1. Expand $A+BC'+ABD'+ABCD$ to canonical SOP  Show that $AB'C+B+BD'+ABD'+A'C = B+C$	K2	CO2
2. Simplify the following functions using K-Map and draw the logic circuit. (i) $F(A,B,C,D) = \sum m(0,1,2,4,5,6,8,10,11,14,15)$  (ii) $F(A,B,C) = \sum m(0,3,5,8,9,13) + \sum d(1,4,7,12)$	K2	CO2
3. Implement the following boolean function using Multiplexer $Y(A,B,C) = \sum(0,1,3,5,6)$  $Y(A,B,C,D) = \sum(0,1,6,12,14,15)$	K2	CO2
4. Design full adder and full subtractor.	K2	CO2
5. Implement full adder and subtractor using demultiplexer.	K2	CO2
6. Using the K-map simplify the function $F = \sum m(4,5,6,7,9,11,13,15,16,18,27,28,31)$	K2	CO2
7. Design BCD to Excess 3 code converter	K2	CO2
8. Design Binary to gray code converter	K2	CO2
9. Construct a 4 bit binary Adder/Subtractor and Design a BCD adder	K2	CO2
10. Design Gray to binary code converter	K2	CO2

### 13.Supportive online Certification courses (**NPTEL, Swayam, Coursera, Udemy, etc.,**)

#### ✿ Swayam

Digital Circuits: [https://swayam.gov.in/nd1\\_noc20\\_ee70/preview](https://swayam.gov.in/nd1_noc20_ee70/preview)

Duration: 12 weeks, Start Date: 14 Sep 2020, End Date: 04 Dec 2020

#### ✿ Udemy

Digital Electric Circuits & Intelligent Electrical Devices

<https://www.udemy.com/course/digital-electric-circuits-intelligent-electrical-devices/>

#### ✿ Coursera

Build a Modern Computer from First Principles: From Nand to Tetris (Project-Centered Course)

<https://www.coursera.org/learn/build-a-computer>

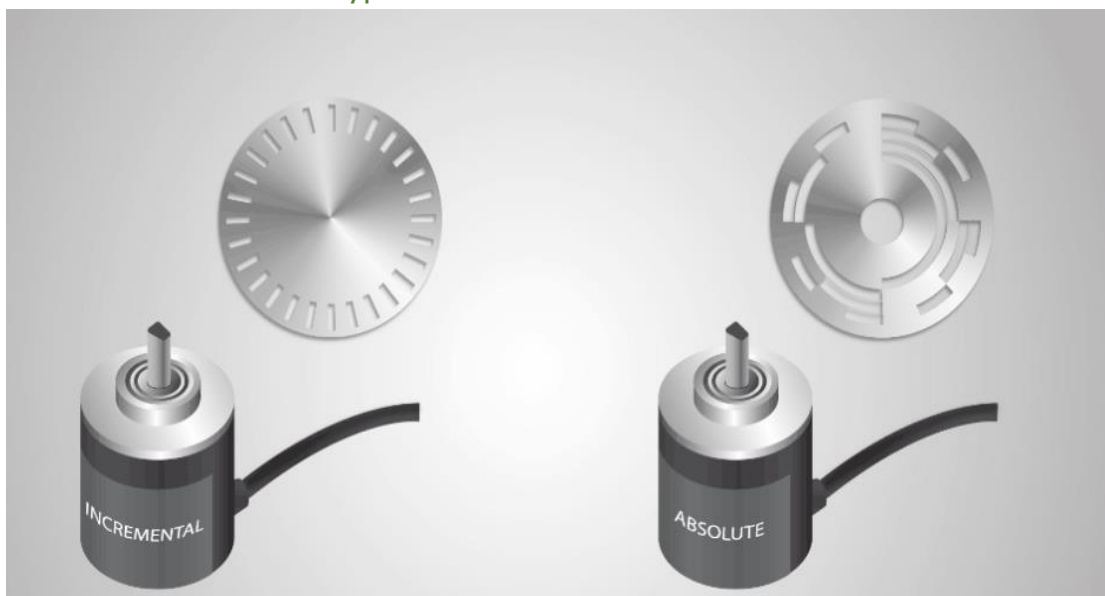
# Real time Applications in day to day life and to Industry

## ❁ Encoder applications in industries

- ❁ An encoder is a device that is used in many industries to provide feedback.
- ❁ In the most basic terms, an encoder, regardless of the type, senses “position”, “direction”, “speed”, or “counts”.
- ❁ Encoders will use motion, under a variety of technologies, and translate it into an electrical signal.
- ❁ That signal is then sent back to a controlling device, and is interpreted, meaning scaled, to represent a value that will then be used within the program.

### ❁ **Encoder Types and Technologies:**

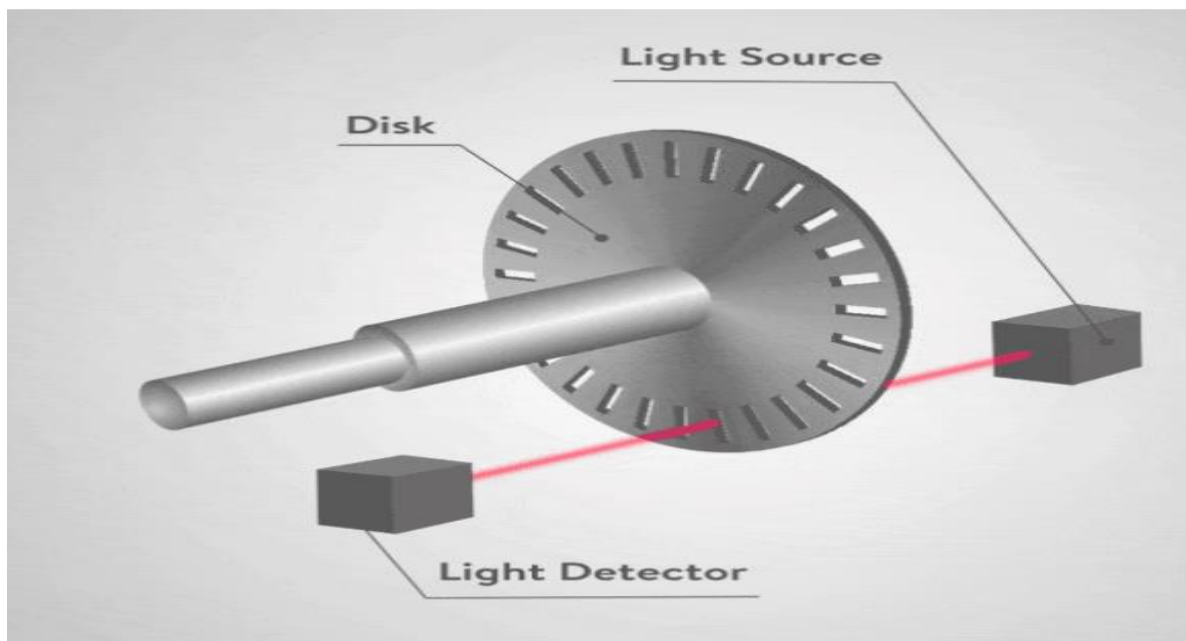
- ❁ Some of the technologies involved in encoders are:
  - ❁ Magnetic
  - ❁ Mechanical
  - ❁ Resistive
  - ❁ Optical
- ❁ “Optical” is the most widely used encoder motion translating technology.
- ❁ There are different types of encoders such as “Absolute” and “Incremental”.



***Fig. Source: realpars.com***

## ❁ Encoder Working Principle:

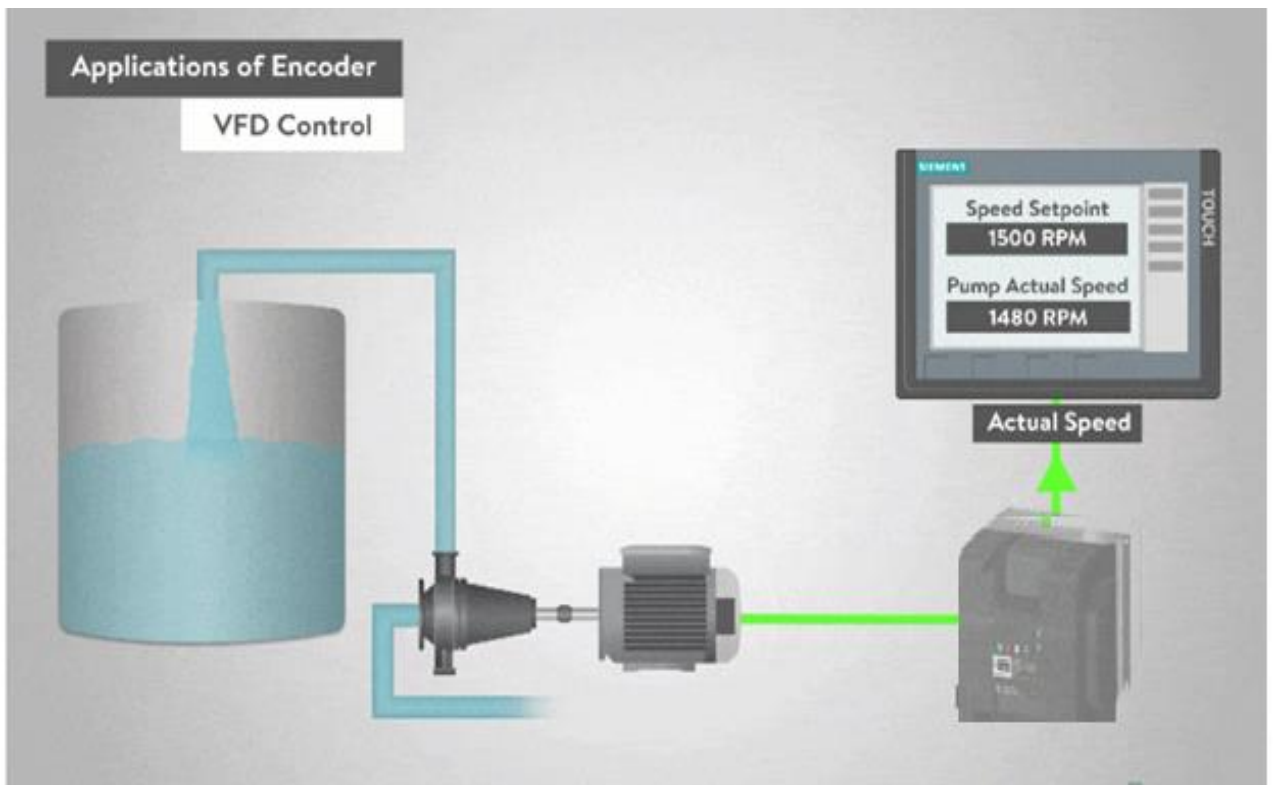
- ❁ An incremental, optical type encoder uses a beam of light that passes through a disk that has opaque lines in a specific pattern, somewhat like the spokes of a wheel.
- ❁ On the other side of the disk is a photo sensing device that will interpret the light, based on the pattern on the disk, picture a shutter, blocking and unblocking the light.
- ❁ The pulses of light are then converted to an electrical signal to be sent back to the processor, through the encoder's output.



*Fig. Source: realpars.com*

## ❁ Encoders for Controlling the Speed of a VFD:

- ❁ For VFD control, you may be running a pump, on a VFD, to fill a tank full of a liquid. You are requesting a certain speed and want to verify that the pumps VFD is at the requested speed.
- ❁ An “encoder” on the VFD may be used for feedback of speed.

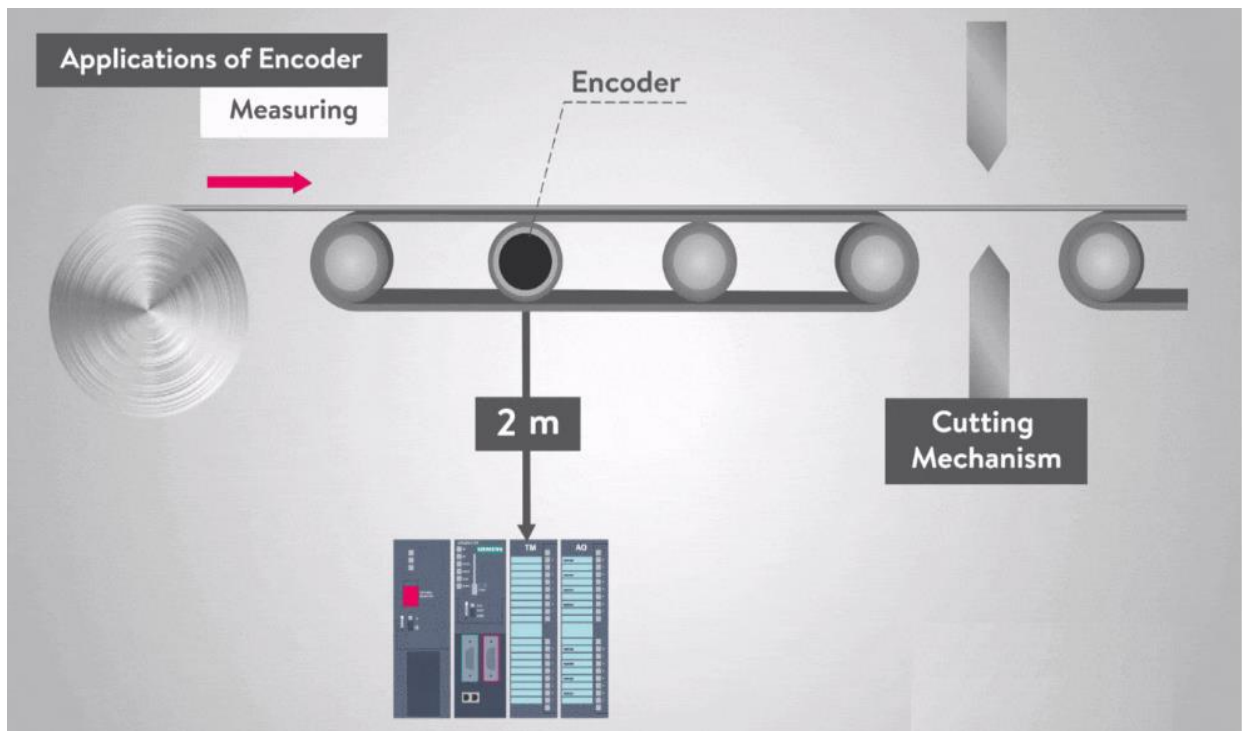


**Fig. Source: [realpars.com](http://realpars.com)**



## ❁ Encoder in Measuring Processes:

- ❁ In this application, you will need to cut some aluminum product to a particular size. You are passing a long roll, meaning hundreds of feet, of the aluminum sheet through a cutting mechanism.
- ❁ You need to determine the amount of aluminum fed, so that you can cut the sheets to the proper size that will be used in a separate manufacturing process.
- ❁ An encoder, attached to the conveyor and reading the material that is feeding through your cutting assembly, will indicate the length of material that has been fed since the last cut. That feedback can then be used to adjust the cutting blade to sever the length required.



*Fig. Source: realpars.com*

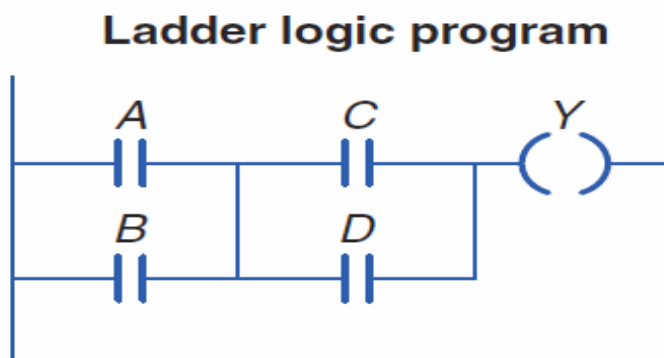
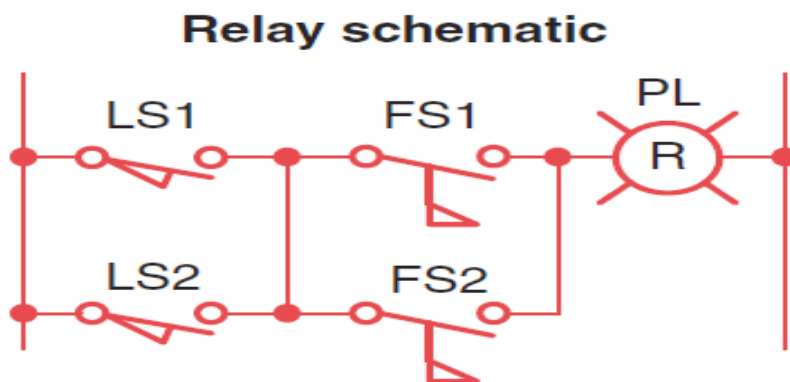
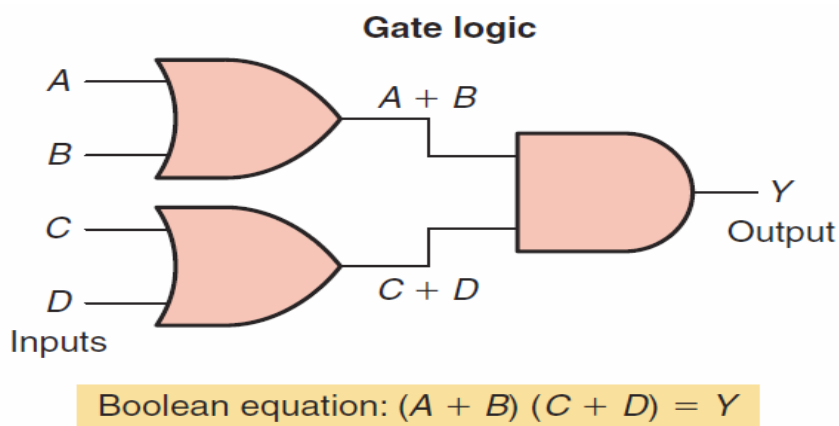
## ❁ Encoder Application in industries: Video Link.:

<https://realpars.com/encoder/>

## 15. Contents beyond the Syllabus ( COE related Value added courses)

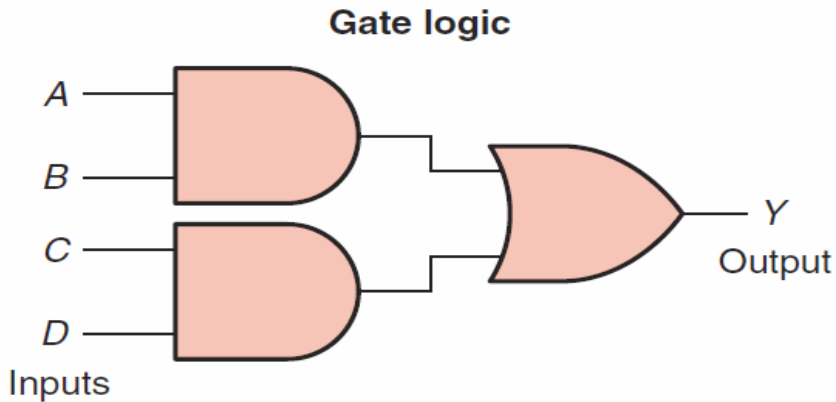
❁ Simulation of SOP & POS functions using PLC simulation software available at Factory Automation CoE.

❁ Example: Two limit switches connected in parallel with each other and in series with two sets of flow switches (in parallel), and used to control a pilot light.

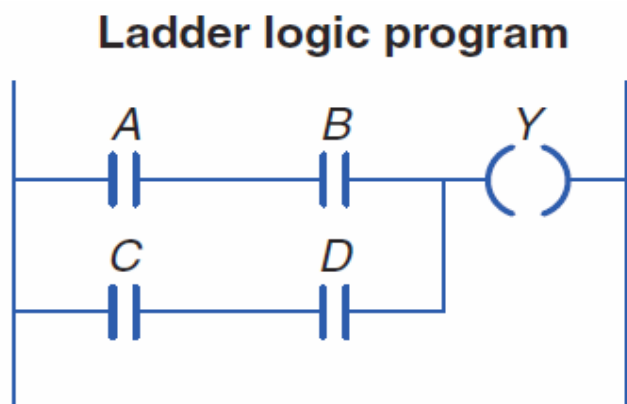
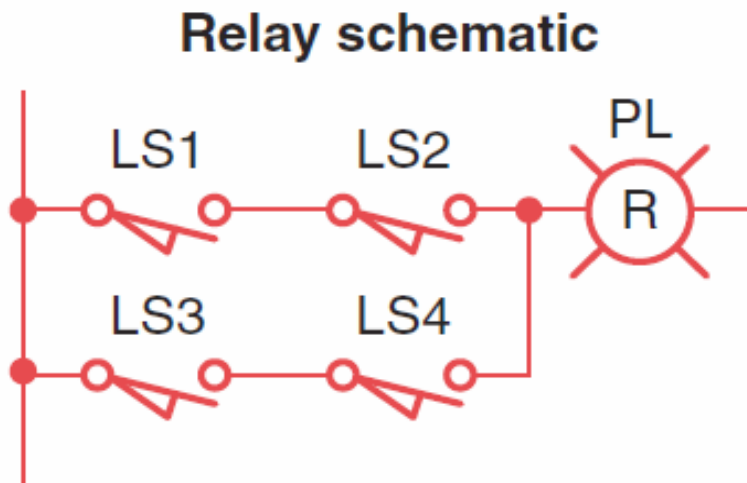


## Contents beyond the Syllabus ( COE related Value added courses)

- ❁ Example: Two limit switches connected in series with each other and in parallel with two other limit switches (in series), and used to control a pilot light.



Boolean equation:  $(AB) + (CD) = Y$



## 17. Prescribed Text Books & Reference Books

### ✿ TEXT BOOKS:

1. James W. Bignel, Digital Electronics, Cengage learning, 5th Edition, 2007.
2. M. Morris Mano, 'Digital Design with an introduction to the VHDL', Pearson Education, 2013.
3. Comer "Digital Logic & State Machine Design, Oxford, 2012.

### ✿ REFERENCES

1. Mandal, "Digital Electronics Principles & Application, McGraw Hill Edu, 2013.
2. William Keitz, Digital Electronics-A Practical Approach with VHDL, Pearson, 2013.
3. Thomas L. Floyd, 'Digital Fundamentals', 11th edition, Pearson Education, 2015.
4. Charles H. Roth, Jr, Lizy Lizy Kurian John, 'Digital System Design using VHDL, Cengage, 2013.

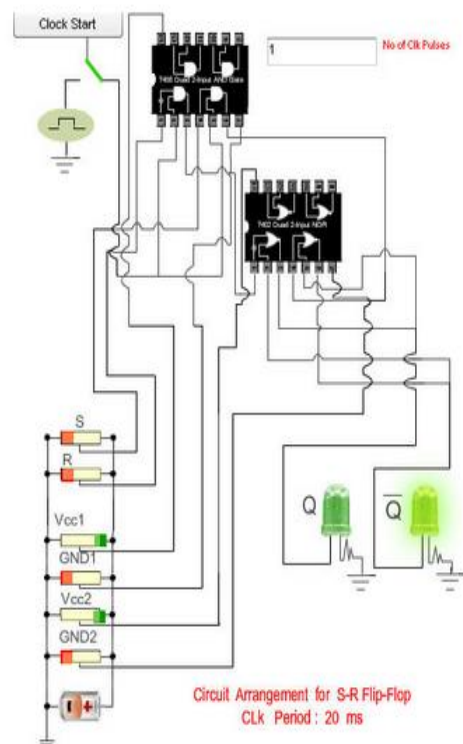
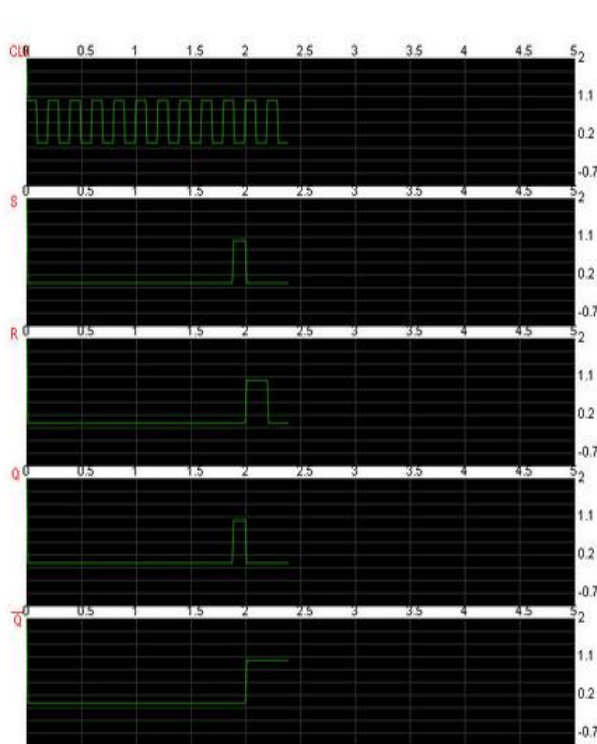
# Digital Logic Circuits

## Unit 3

## 8. Activity based learning

- ✿ Understanding the working of the synchronous sequential circuits through simulation using virtual labs [http://vlabs.iitb.ac.in/vlabs-dev/vlab\\_bootcamp/bootcamp/cool\\_developers/labs/exp6/index.html](http://vlabs.iitb.ac.in/vlabs-dev/vlab_bootcamp/bootcamp/cool_developers/labs/exp6/index.html)
- ✿ Simulation of Flip Flops, Registers, Counters
- ✿ Practice using open source DEEDS: <https://www.digitalelectronicsdeeds.com/>
- ✿ Experiment to understand the working of SR Flip Flop

<http://vlabs.iitkgp.ernet.in/dec/exp8/index.html#>



## Table of Contents

### ❁ UNIT III SYNCHRONOUS SEQUENTIAL CIRCUITS

Page No:

1. Sequential logic-----15

❁ SR, JK, D and T flip flops-----19

❁ level triggering and edge triggering

2. Counters -----31

❁ Asynchronous and

❁ Synchronous type

❁ Modulo counters

3. Shift registers -----62

4. Design of synchronous sequential circuits-----71

❁ Moore and Melay models

Design

❁ State diagram

❁ State reduction

❁ State assignment

# 1. Introduction to Sequential Logic

- ✿ Output depends not only on current input but also on past input values, e.g., design a counter
- ✿ Need some type of memory to remember the past input values

S.No	Combinational Circuit	Sequential Circuit
1	Output depends on only the present states.	Output depends not only on current input but also on past input values.
2	No need of memory elements.	It requires memory elements
3	There is no clock pulse.	It requires clock pulse.

## Types of Sequential Logic Circuits:

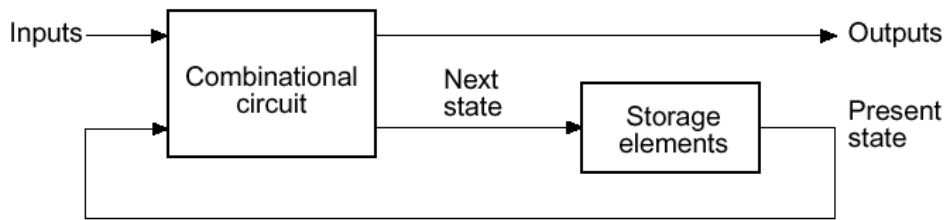
- ✿ Synchronous Sequential Logic - **Unit-3**
- ✿ Asynchronous Sequential Logic - **Unit-4**

## Difference between Synchronous and Asynchronous Sequential Circuit:

S.No	Synchronous Sequential Circuit	Asynchronous Sequential Circuit
1	circuit output changes only at some discrete instants of time	circuit output can change at any time.
2	This type of circuits achieves synchronization by using a timing signal called the <i>clock</i> .	It performs clockless operation.



## ❁ Block Diagram of Sequential circuit



- ❁ Sequential Logic circuits remember past inputs and past circuit state.
- ❁ Outputs from the system are “fed back” as new inputs With gate delay and wire delay
- ❁ The storage elements are circuits that are capable of storing binary information: memory
- ❁ The binary information stored in these elements at any given time defines the **state** of the sequential circuit at that time.

## Latches and Flip-Flops

- ❁ Latches are “transparent” (= any change on the inputs is seen at the outputs immediately when  $C=1$ ).
- ❁ This causes synchronization problems.
- ❁ Solution: use latches to create flip-flops that can respond (update) only on specific times (instead of any time).
- ❁ A flip-flop circuit maintains a binary state indefinitely till power is on or a change in the input signal occurs.
- ❁ The state of a latch or flip-flop is switched by a change in the control input

- ❁ This momentary change is called a **trigger**



- ❁ Latch: level-sensitive



- ❁ Flip-Flop: edge-triggered



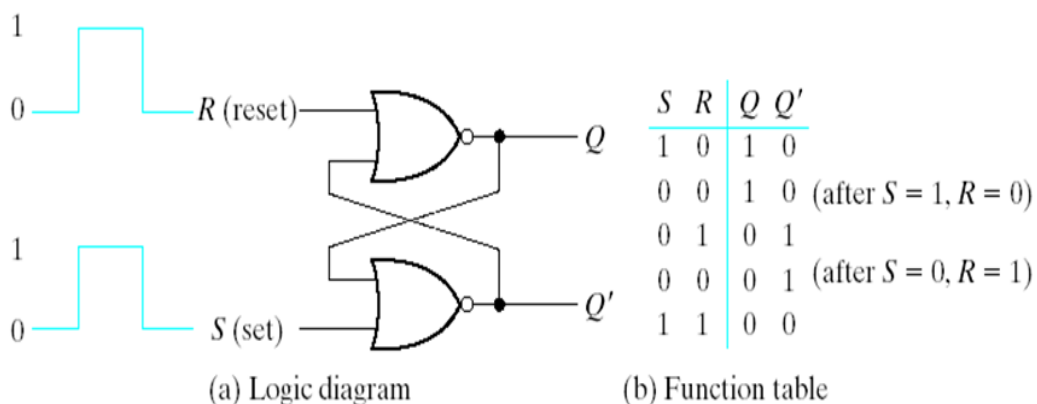
## Difference between Latches and Flip Flops :

Latches	Flip Flops
Latches are building blocks of sequential circuits and these can be built from logic gates	Flip flops are also building blocks of sequential circuits. But, these can be built from the latches.
Latch continuously checks its inputs and changes its output correspondingly.	Flip flop continuously checks its inputs and changes its output correspondingly only at times determined by clocking signal
The latch is sensitive to the duration of the pulse and can send or receive the data when the switch is on	Flipflop is sensitive to a signal change. They can transfer data only at the single instant and data cannot be changed until next signal change. Flip flops are used as a register.
It is based on the enable function input	It works on the basis of clock pulses
It is a level triggered, it means that the output of the present state and input of the next state depends on the level that is binary input 1 or 0.	It is an edge triggered, it means that the output and the next state input changes when there is a change in clock pulse whether it may a +ve or -ve clock pulse.

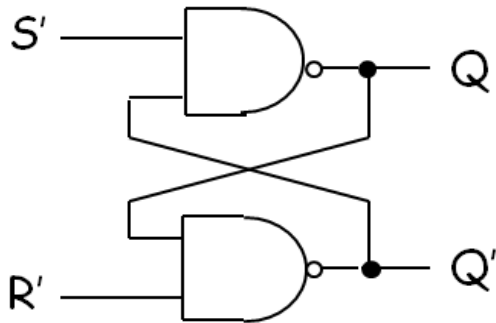
### SR Latch:

The SR latch is a circuit with two cross-coupled NOR gates or two cross-coupled NAND gates. It has two inputs labeled S for set and R for reset.

### SR Latch ( using NOR Gates) :



SR Latch ( using NAND Gates ) :



Functional Table:

S	R	Q	Q'	Remarks
0	1	0	1	Resets
1	1	0	1	Holds the previous value
1	0	1	0	Sets
1	1	1	0	Holds the previous value

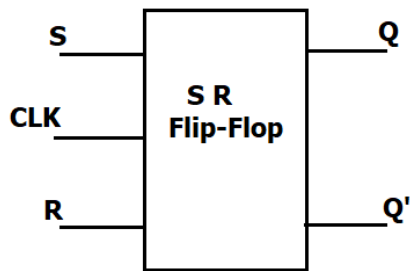
## 2. FLIP –FLOPS

### Types of Flip Flops:

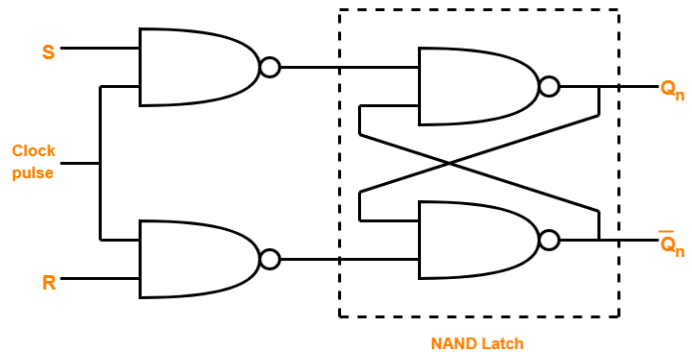
1. SR (Set-Reset) FF
2. JK FF
3. Delay/ Data (D) FF
4. Toggle (T) FF

### SR Flip –Flop:

#### 1. Block Diagram:



#### 2. Logic Diagram:



SR Flip Flop Using NAND Latch

#### 3.Characteristic Table:

- ✿ Characteristic table specifies the next state  $Q(t+1)$  with known inputs and present state  $Q(t)$ .

$$Q(t+1) = \sum m(1, 4, 5) + \sum d(6, 7)$$

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

#### 4.Simplified Characteristic Table with inputs and next state

- ✿ Two useful states:

$S=1, R=0$  set state (Q will become to 1)

$S=0, R=1$  reset state (Q will become to 0)

- ✿ When  $S=0$  and  $R=0$  No change in current value

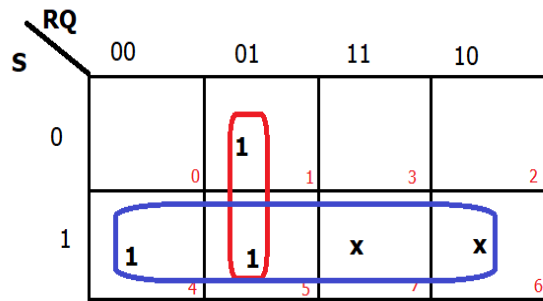
- ✿ When  $S=1$  and  $R=1$  Indeterminate state

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

## 5. Characteristic Equation:

From the Characteristic table  $Q(t+1) = \sum m(1,4,5) + \sum d(6,7)$

Minimization using K-Map



$$Q(t+1) = S + R'Q$$

## 6. Excitation Table:

Excitation table gives the required inputs for a given change of state.

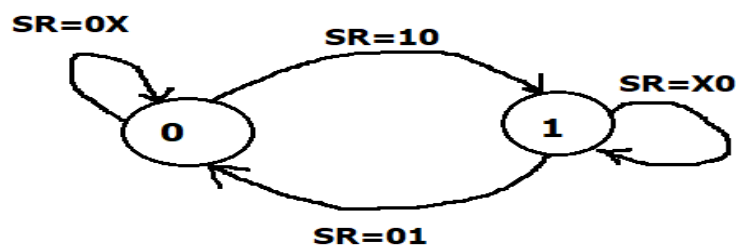
Flip-flop input conditions that will cause the required transition from present state to next state are required during design of a sequential circuit

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

## 7. State Diagram:

The information in characteristic table is represented graphically in a state diagram.

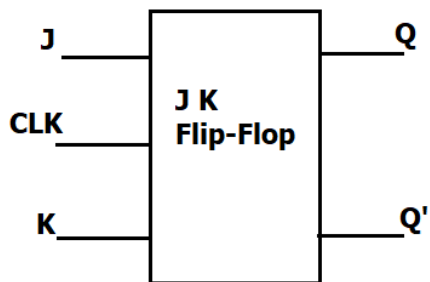
A state is represented by a circle, and transition from one state to another is given by directed lines.



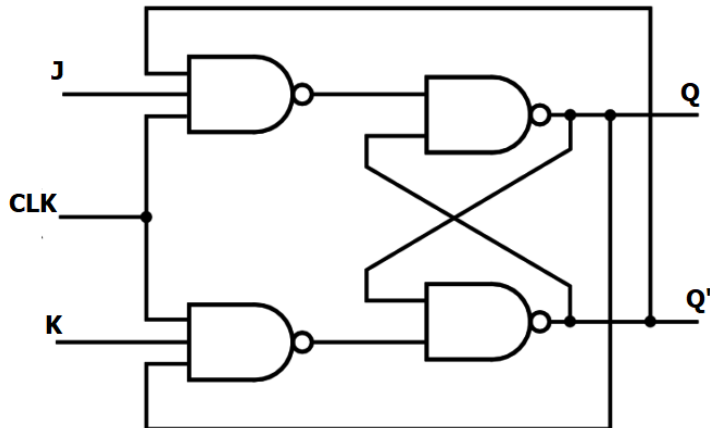
## JK Flip –Flop:

- ✿ The indeterminate state of the SR FF is defined in the JK FF
- ✿ When J and K are equal to 1, the flip-flop switches to its complement state, if  $Q = 1$ , it switches to  $Q = 0$

### ✿ 1.Block Diagram:



### 2. Logic Diagram:

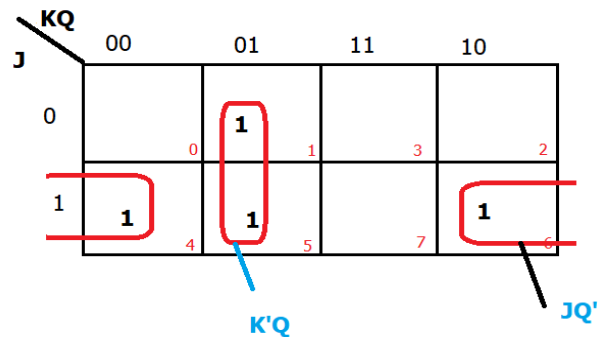


### ✿ 3.Characteristic Table:

J	K	Q(t)	Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

✿  $Q(t+1) = \sum m(1, 4, 5, 6)$

#### 4. Characteristic Equation , $Q(t+1)$

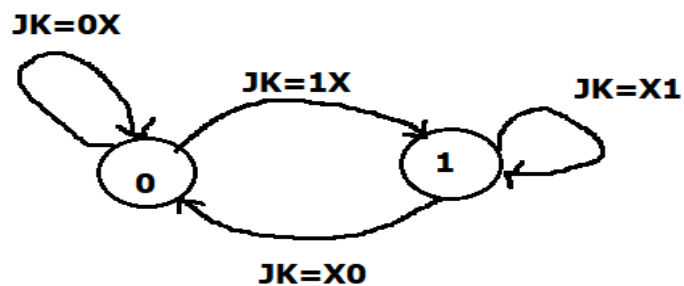


✿  $Q(t+1) = K'Q + JQ'$

#### ✿ 5. Excitation Table:

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

#### ✿ 6. State Diagram:

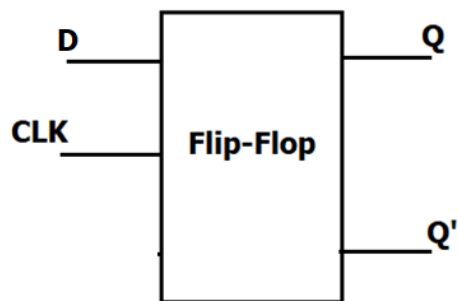


#### ✿ 7. Simplified Characteristic Table:

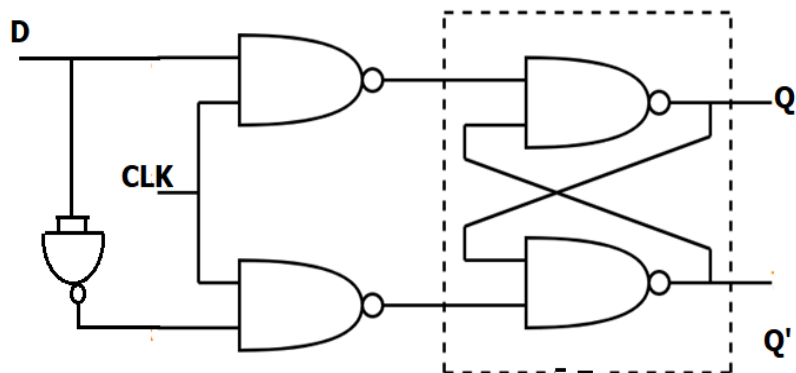
J	K	$Q(t+1)$
0	0	NC
0	1	0
1	0	1
1	1	$Q'$

# D Flip –Flop:

## ❁ 1.Block Diagram:



## ❁ 2. Logic Diagram:



## ❁ 3.Characteristic Table:

D	Q(t)	Q(t+1)
0	0	0
0	1	0
1	0	1
1	1	1

❁  $Q(t+1)=\Sigma m (2,3)$



#### 4. Characteristic Equation , $Q(t+1)$

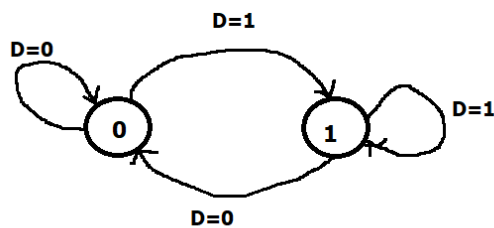
		Q	0	1
D	0			
			0	1
1		1	2	3

❁  $Q(t+1)=D$

#### ❁ 5. Excitation Table:

Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

#### ❁ 6. State Diagram:

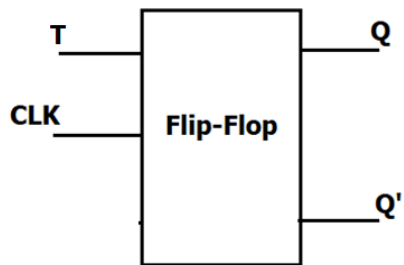


#### ❁ 7. Simplified Characteristic Table:

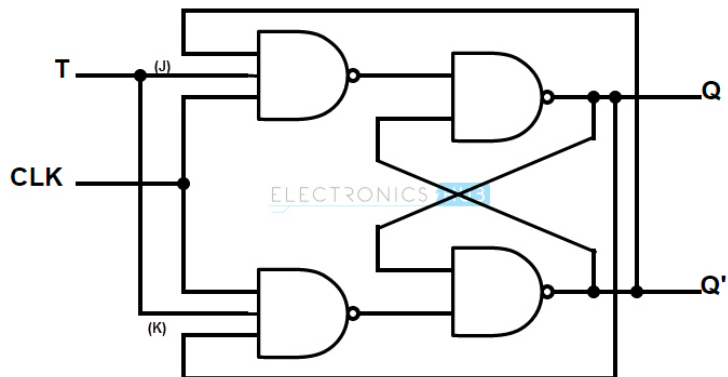
D	Q(t+1)
0	0
1	1

# T Flip –Flop:

## ❁ 1.Block Diagram:



## ❁ 2. Logic Diagram:

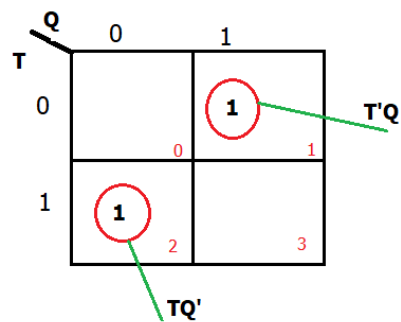


## ❁ 3.Characteristic Table:

T	Q(t)	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

❁  $Q(t+1) = \Sigma m(1,2)$

#### 4. Characteristic Equation , $Q(t+1)$

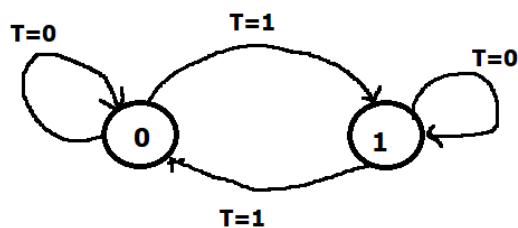


$$Q(t+1) = T'Q + TQ' = T \oplus Q$$

#### ❁ 5. Excitation Table:

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

#### ❁ 6. State Diagram:



#### ❁ 7. Simplified Characteristic Table:

T	Q(t+1)
0	NC
1	Q'

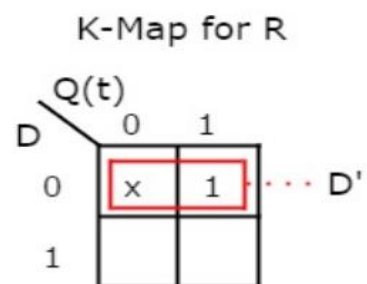
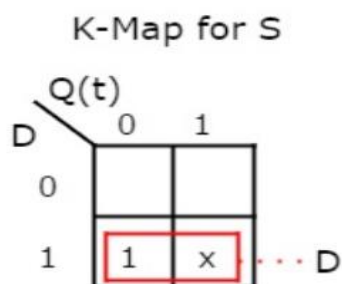
## Conversion of Flip Flops

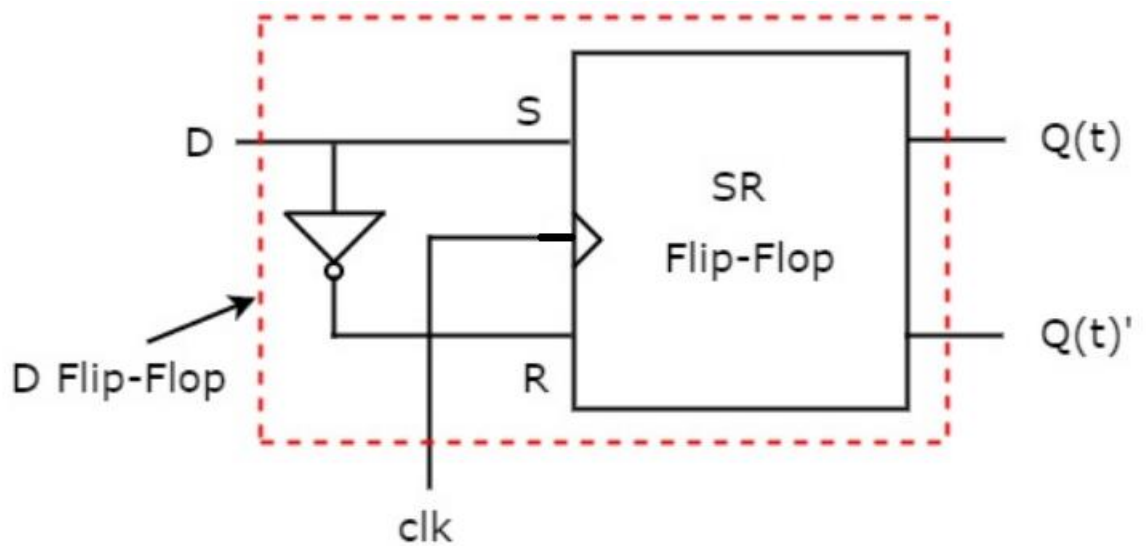
steps for converting one flip-flop to the other:

- ✿ Consider the characteristic table of desired flip-flop.
- ✿ Fill the excitation values inputs of given flip-flop for each combination of present state and next state. The excitation table for all flip-flops is shown below.
- ✿ Get the simplified expressions for each excitation input using K-Maps for simplifying.
- ✿ Draw the circuit diagram of desired flip-flop according to the simplified expressions using given flip-flop and necessary logic gates.

### Convert SR Flip-Flop to D Flip-Flop

D flip-flop input (Desired)	Present State	Next State	SR flip-flop inputs (Given)	
D	Q(t)	Q(t+1)	S	R
0	0	0	0	x
0	1	0	0	1
1	0	1	1	0
1	1	1	x	0





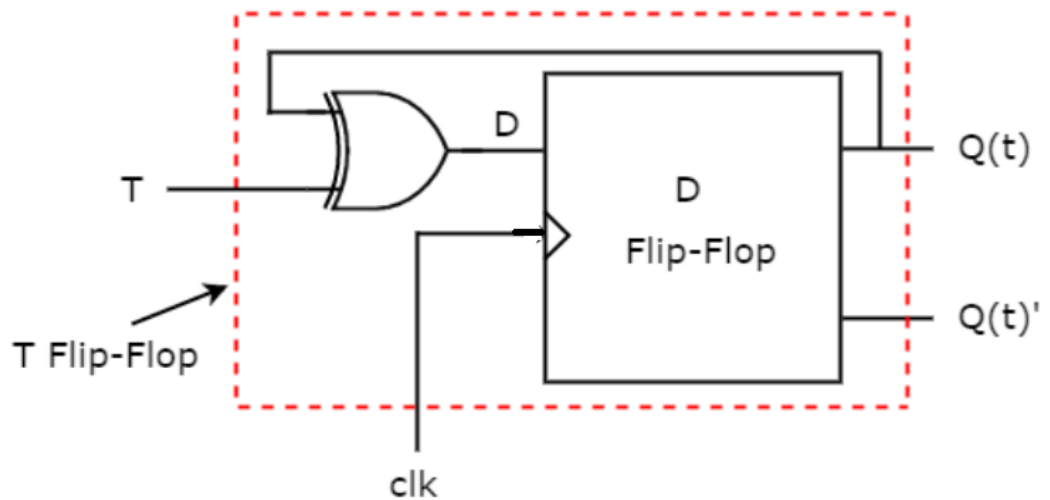
❁ Convert D flip-flop to T flip-flop :

T flip-flop input (Desired)	Present State	Next State	D flip-flop inputs (Given)
T	Q(t)	Q(t+1)	D
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

	Q(t) 0	1
T 0		1
1	1	

0 1 2 3

✿  $D = TQ' + T'Q = T \oplus Q$



✿ JK flip-flop to T flip-flop

T flip-flop input (Desired)	Present State	Next State	JK flip-flop inputs (Given)	
D	Q(t)	Q(t+1)	S	R
0	0	0	0	x
0	1	1	x	0
1	0	1	1	x
1	1	0	x	1

K-Map for J

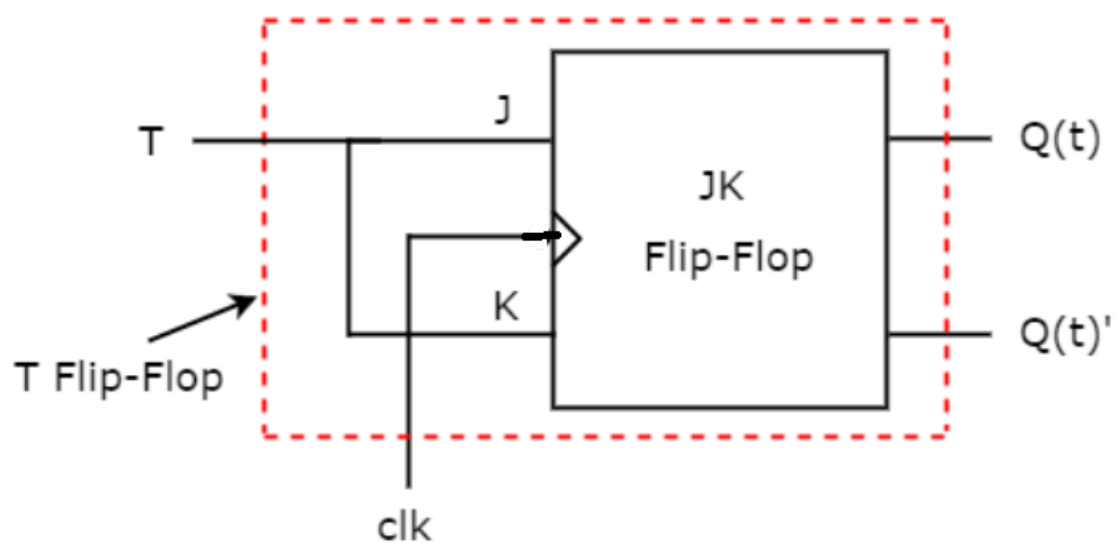
		Q(t)	
		0	1
T	0		x
	1	1	x

... T

K-Map for K

		Q(t)	
		0	1
T	0	x	
	1	x	1

... T



### 3. COUNTERS

- ⚙ Counter is a sequential circuit.
- ⚙ A digital circuit which is used for counting pulses is known as counter.
- ⚙ Counter is the widest application of flip-flops. It is a group of flip-flops with a clock signal applied.

Counters are of two types.

- ⚙ Asynchronous or ripple counters.
- ⚙ Synchronous counters.

#### Difference between Synchronous Counter Asynchronous Counter

Sl.No	Synchronous Counter	Asynchronous Counter
1	In synchronous counter, all flip flops are triggered with same clock simultaneously.	In asynchronous counter, different flip flops are triggered with different clock, not simultaneously.
2	Synchronous Counter is faster than asynchronous counter in operation.	Asynchronous Counter is slower than synchronous counter in operation.
3	Synchronous Counter is also called Parallel Counter.	Asynchronous Counter is also called Serial Counter.
4	Synchronous Counter will operate in any desired count sequence.	Asynchronous Counter will operate only in fixed count sequence (UP/DOWN).
5	Synchronous Counter examples are: <a href="#">Ring counter</a> , <a href="#">Johnson counter</a> .	Asynchronous Counter examples are: <a href="#">Ripple</a> UP counter, Ripple DOWN counter
6	In synchronous counter, propagation delay is less.	In asynchronous counter, there is high propagation delay.



## Asynchronous Counter/ Ripple Counter:

- ❁ Asynchronous counters are those counters which **do not operate on simultaneous clocking**.
- ❁ In asynchronous counter, only the first flip-flop is externally clocked using clock pulse while the clock input for the successive flip-flops will be the output from a previous flip-flop.
- ❁ As the data ripples between the output of one flip-flop to the input of the next, it is called as **Ripple Counter**.

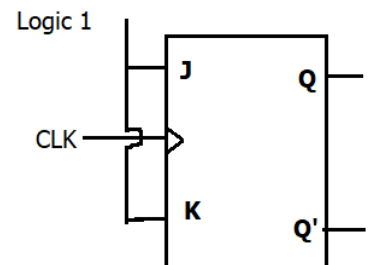
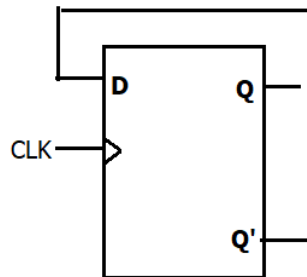
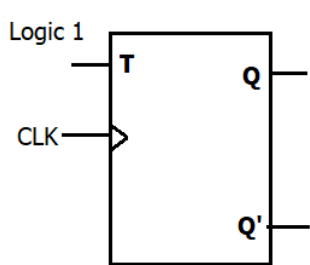
### Design of Asynchronous Counter:

- ❁ 3-bit up counter
- ❁ 3 bit Down Counter
- ❁ MOD-5 counter
- ❁ 4-bit up counter
- ❁ Decade counter/ MOD-10 Counter/BCD Counter

Triggering	Up counter	Down Counter
<b>Positive Edge Triggering</b>	<b>Q'</b> of previous FF is connected as <u>Clk</u> input for next FF	<b>Q</b> of previous FF is connected as <u>Clk</u> input for next FF
<b>Negative Edge Triggering</b>	Q of previous FF is connected as <u>Clk</u> input for next FF	Q' of previous FF is connected as <u>Clk</u> input for next FF

- ❁ Asynchronous counter is implemented in toggle mode
- ❁ The toggle flip-flop changes state when the clock input is applied,  $T = 1$
- ❁ The T-type flip-flop is not available commercially but can be constructed from a JK flip-flop (or D-type flip-flop)
- ❁ By connecting the J input with the K input and both to logic level "1". With J and K HIGH, the flip-flop changes state every time it is triggered at its clock input. This clock input is now called the "toggle input" as the output becomes "1" if it was "0", and a "0" if it was "1", that is it toggles.

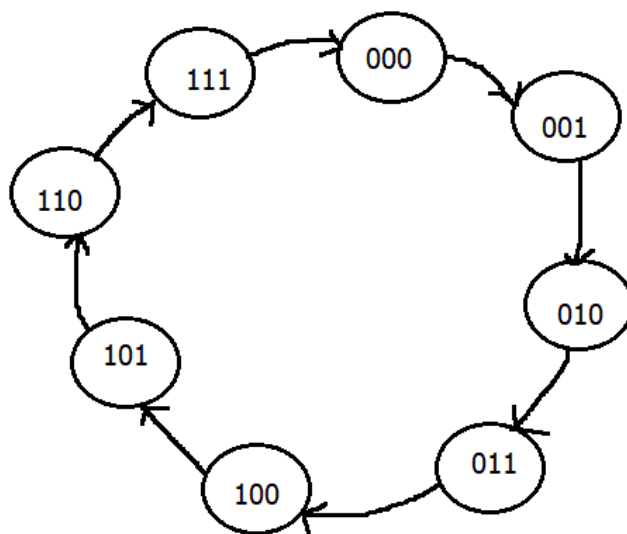
- ✿ The Data/Delay D-type flip-flop can just like the JK flip-flop be converted to perform as a toggle flip-flop by connecting the Q output directly to the D-input



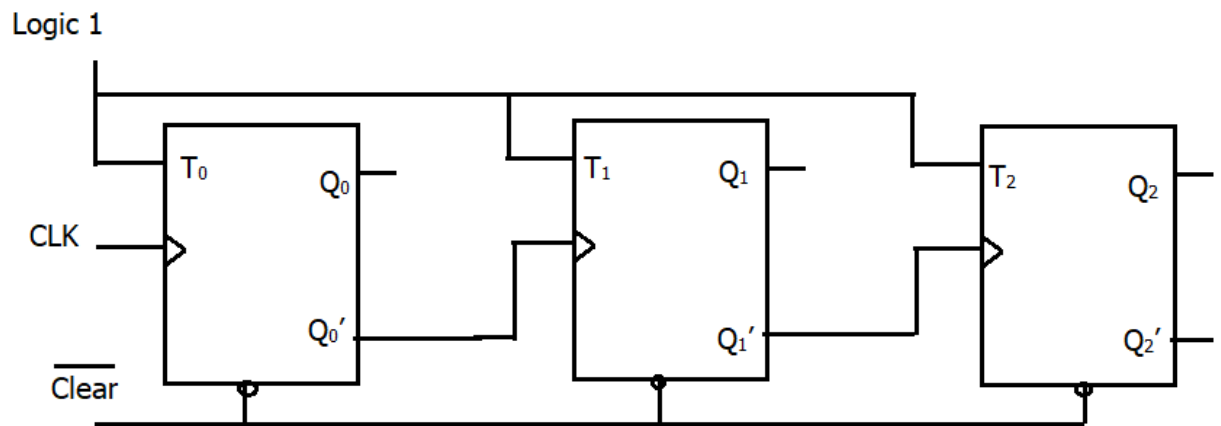
## ✿ Design of 3-bit up counter :

- ✿ Step-1 : State Diagram ( 0-1-2-3-4-5-6-7-0)

No.of FFs Required= 3



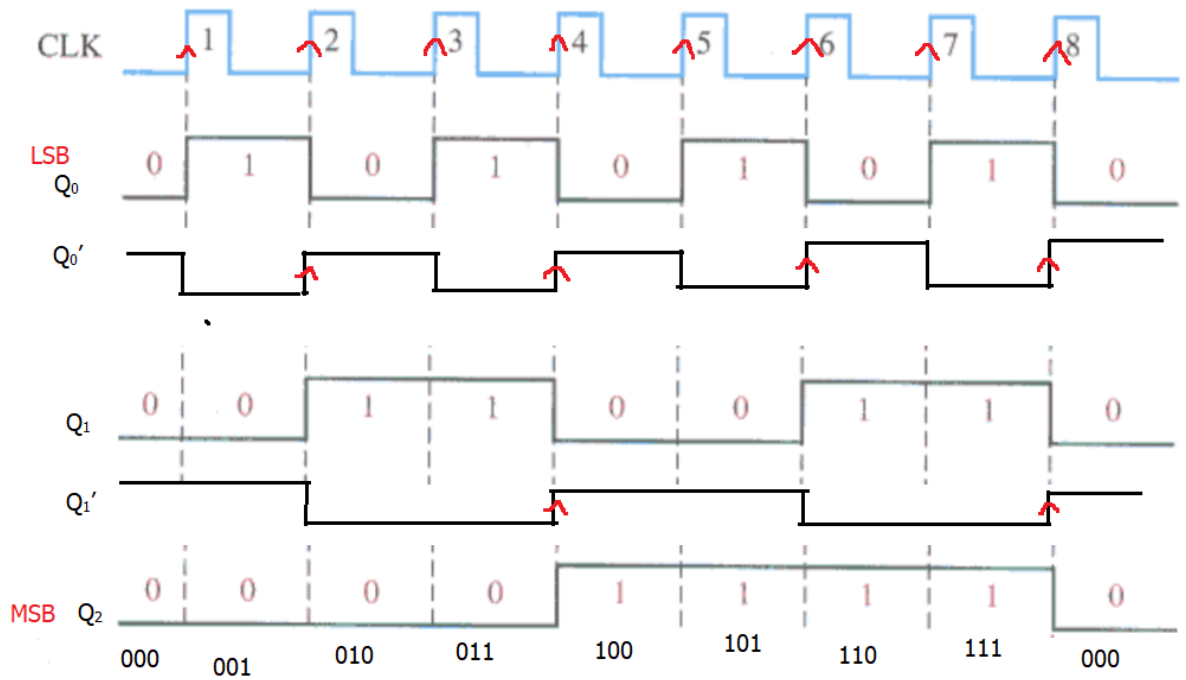
## ✿ Block Diagram:



## ✿ Truth Table

Outputs	$Q_2$	$Q_1$	$Q_0$
Clear	0	0	0
CK Pulse 1	0	0	1
CK Pulse 2	0	1	0
CK Pulse 3	0	1	1
CK Pulse 4	1	0	0
CK Pulse 5	1	0	1
CK Pulse 6	1	1	0
CK Pulse 7	1	1	1
CK Pulse 8	0	0	0

## ❁ Output Waveform

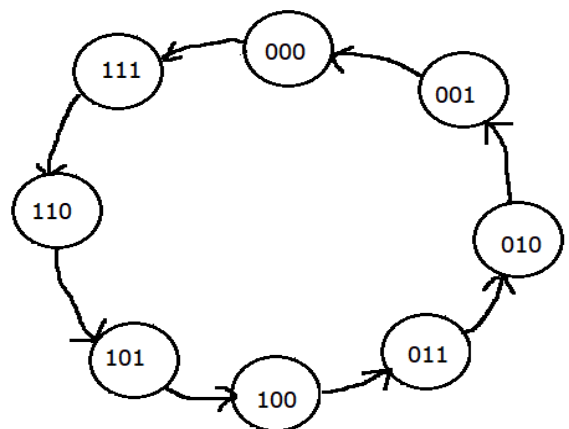


## ❁ Design of 3-bit Down counter :

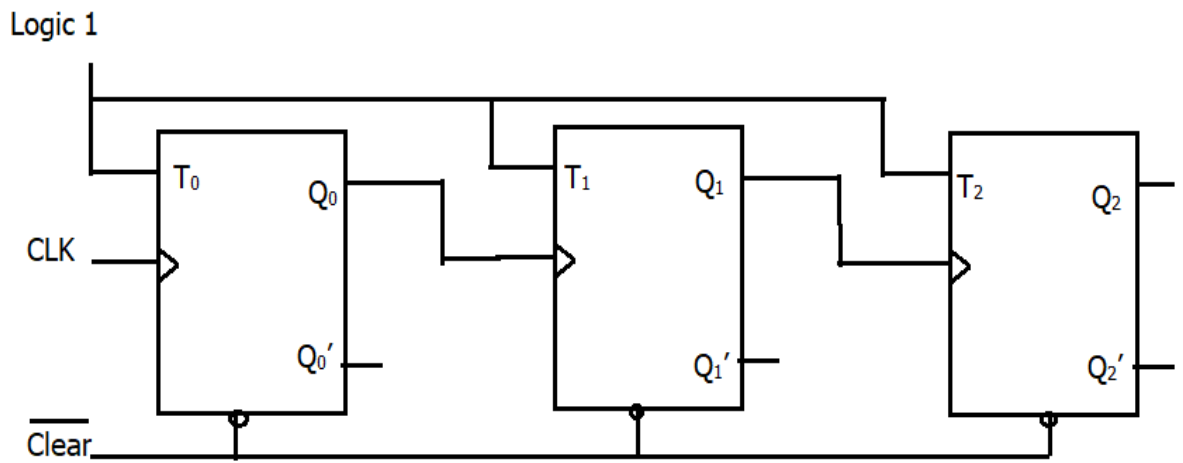
### Step-1 : State Diagram

Counting sequence (0-7-6-5-4-3-2-1-0)

No. of FFs Required=3



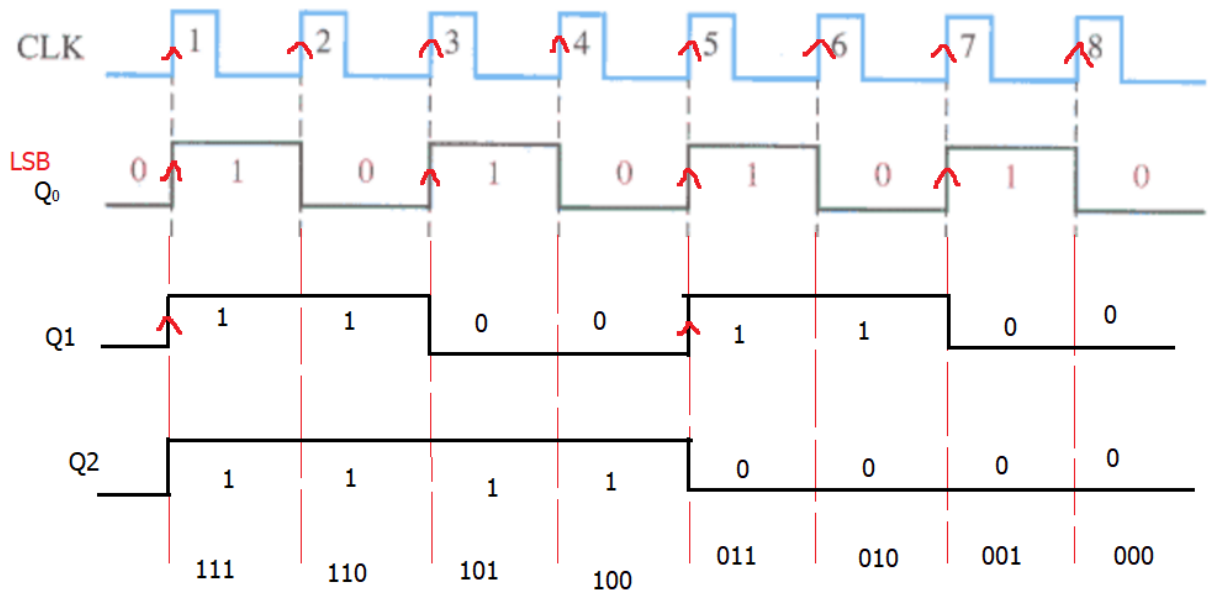
### ✿ Block Diagram:



### ✿ Truth Table:

Outputs	$Q_2$	$Q_1$	$Q_0$
Clear	0	0	0
CK Pulse 1	1	1	1
CK Pulse 2	1	1	0
CK Pulse 3	1	0	1
CK Pulse 4	1	0	0
CK Pulse 5	0	1	1
CK Pulse 6	0	1	0
CK Pulse 7	0	0	1
CK Pulse 8	0	0	0

## ❁ Output Waveform:



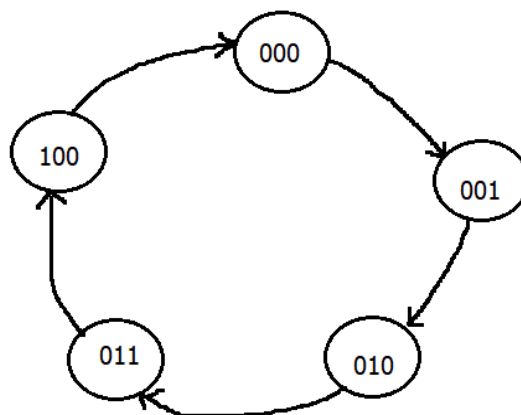
## ❁ Design of Mod-5 Counter

MOD 6 = 0-1-2-3-4-5-0

Step-1: State Diagram

Counting Sequence (0-1-2-3-4-0)

No. of FFs Required = 3

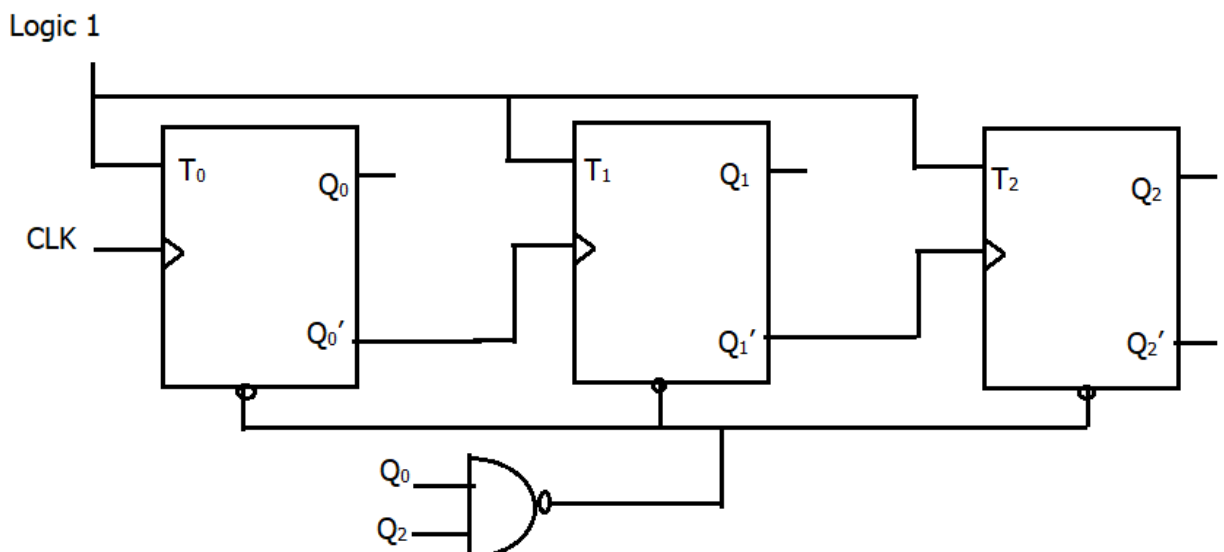


### ❁ Truth Table:

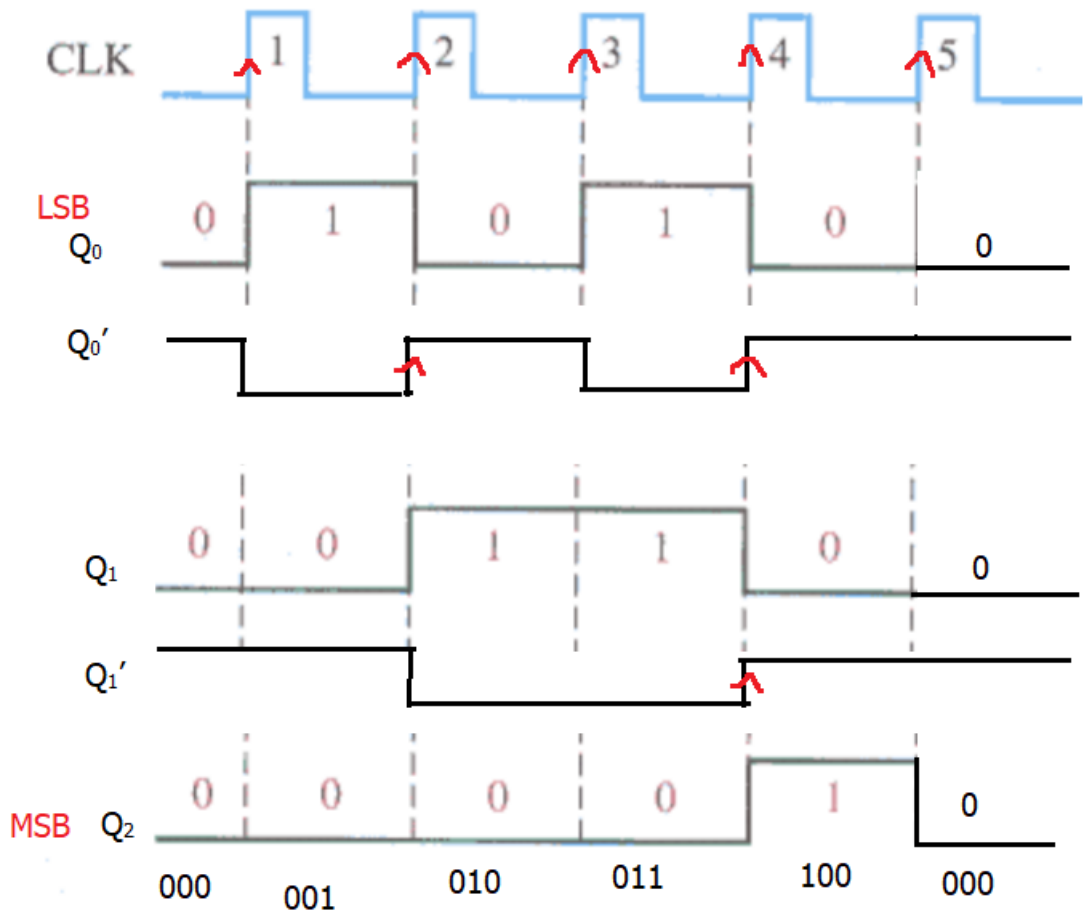
Outputs	$Q_2$	$Q_1$	$Q_0$
Clear	0	0	0
CK Pulse 1	0	0	1
CK Pulse 2	0	1	0
CK Pulse 3	0	1	1
CK Pulse 4	1	0	0
CK Pulse 5	0	0	0

- ❁ Instead 101 for clock pulse 5, the state of FFs should be 000. Hence  $Q_2=1$  &  $Q_0=1$  are taken from 101 and fed to input of NAND gate to give clear input to all FFs.

### ❁ Block Diagram:



❁ Output waveform:



## ❁ Design of BCD/Mod-10/Decade ripple counter

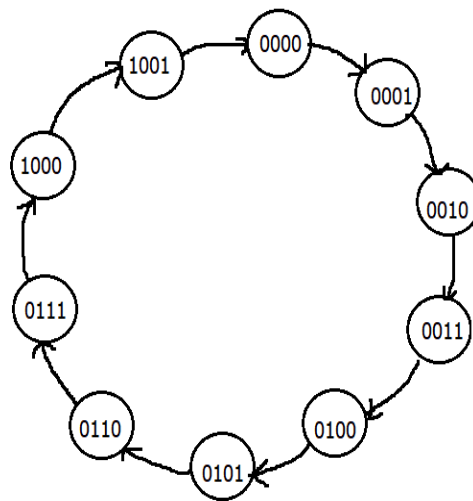
MOD 10= 0-1-2-3-4-5-6-7-8-9-0

Step-1: State Diagram

Counting Sequence (0-1-2-3-4-0)

No.of FFs Required=4



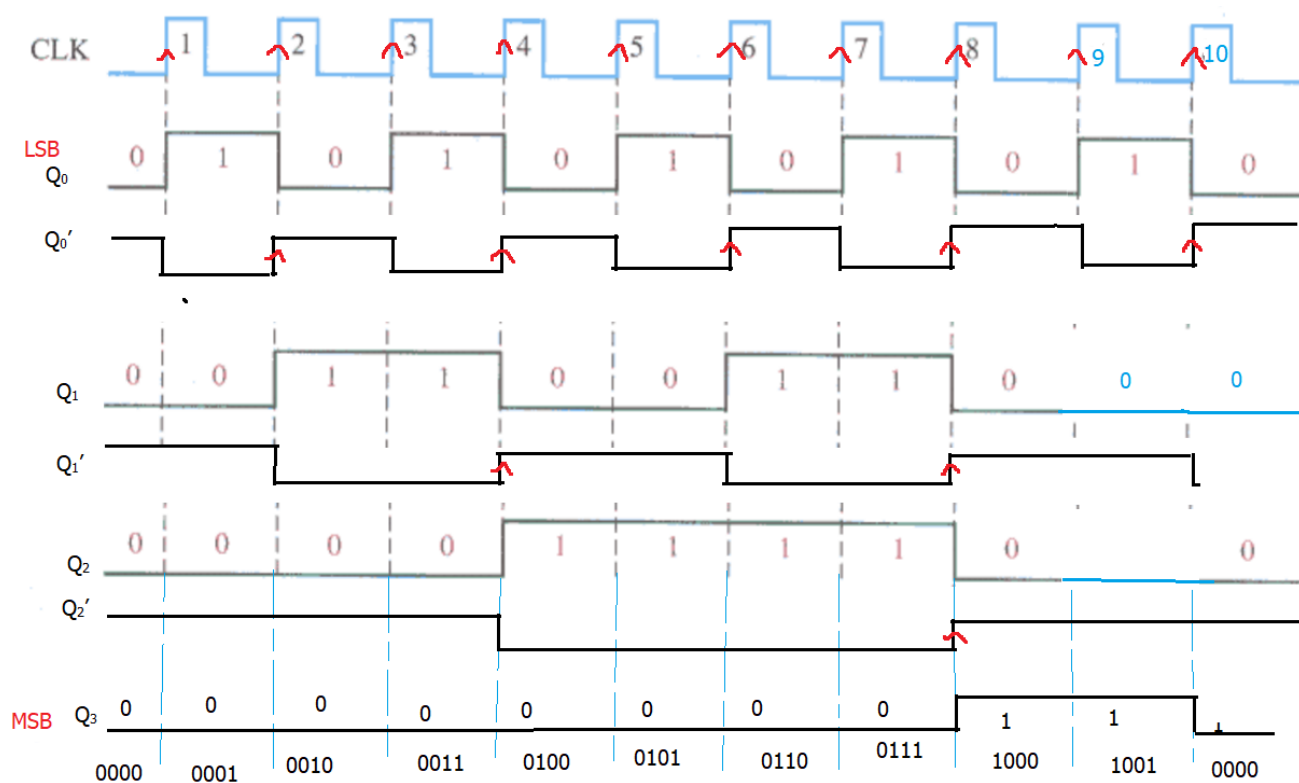


### ✿ Truth Table:

Outputs	Q <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>0</sub>
Clear	0	0	0	0
CK Pulse 1	0	0	0	1
CK Pulse 2	0	0	1	0
CK Pulse 3	0	0	1	1
CK Pulse 4	0	1	0	0
CK Pulse 5	0	1	0	1
CK Pulse 6	0	1	1	0
CK Pulse 7	0	1	1	1
CK Pulse 8	1	0	0	0
CK Pulse 9	1	0	0	1
CK Pulse 10	0	0	0	0

✿ Instead 1010 for clock pulse 10, the state of FFs should be 0000. Hence Q<sub>3</sub>=1 & Q<sub>1</sub>=1 are taken from 1010 and fed to input of NAND gate to give clear input to all FFs.

## ✿ Output waveform:



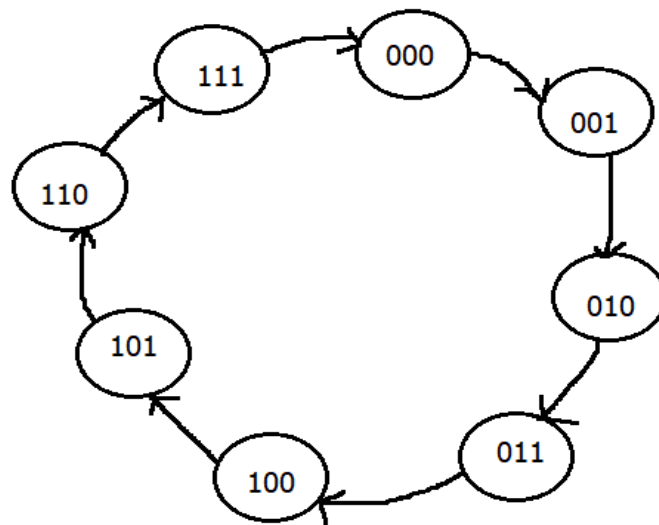
## Steps for Designing Synchronous Counter:

- ✿ Decide the number of FFs required.
- ✿ Draw State Diagram
- ✿ Form State Table (Present state, Next State, FF input)
- ✿ Using K –map , get the simplified expression for the input of FF
- ✿ Draw the design using given FF

### 1.Design a 3 bit counter (Up counter)

Step-1 : State Diagram ( 0-1-2-3-4-5-6-7-0)

No: of States =8, No: of FFs required= 3



## ✿ Step2: State Table

Present State			Next State			FF Input					
A	B	C	A(t+1)	B(t+1)	C(t+1)	J <sub>A</sub>	K <sub>A</sub>	J <sub>B</sub>	K <sub>B</sub>	J <sub>C</sub>	K <sub>C</sub>
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	0	0	0	X	1	X	1	X	1

Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

## ✿ Step-3: Finding Equation for FF input using K-map

K-Map for J<sub>A</sub>

A	BC			
	00	01	11	10
0			1	
1	X	X	X	X

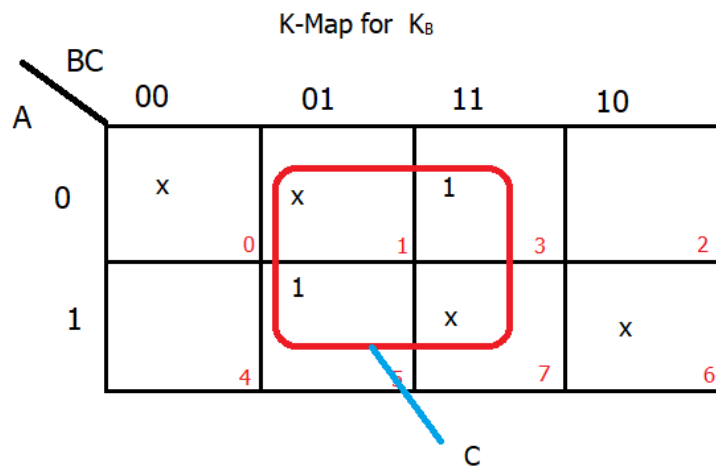
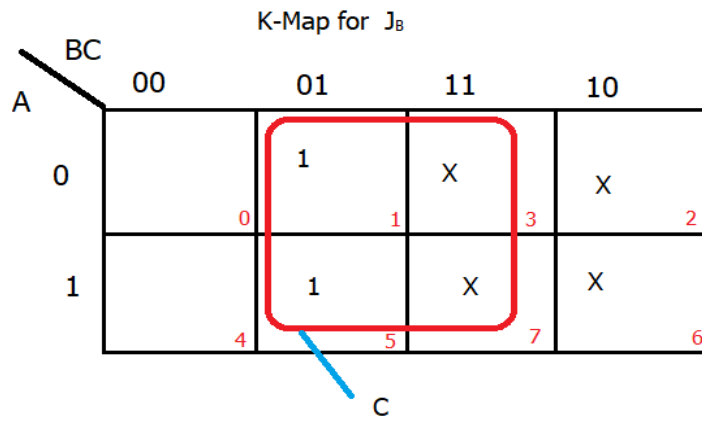
BC

K-Map for K<sub>A</sub>

A	BC			
	00	01	11	10
0	X	X	X	X
1			1	

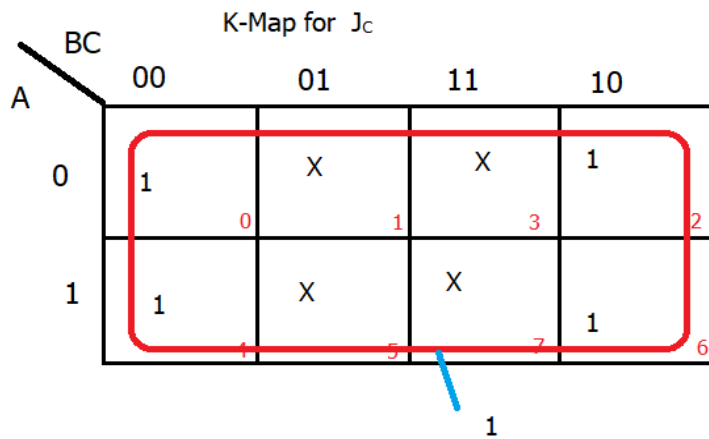
BC

✿  $J_A = K_A = BC$

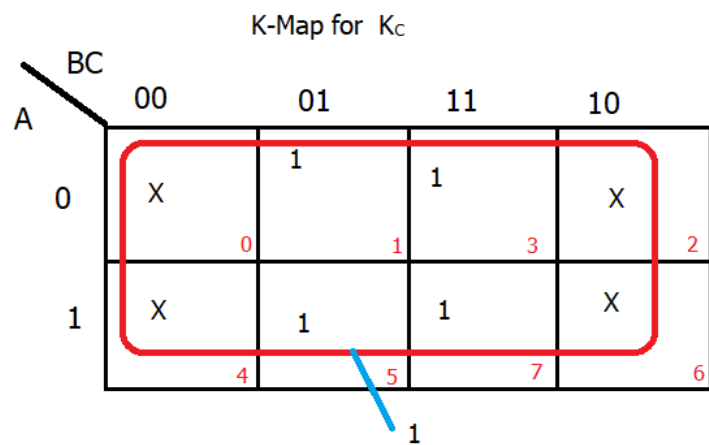


⊗  $J_B = K_B = C$

⊗ Let  $Q_A Q_B Q_C = A B C$



⊗  $J_C = K_C = 1$

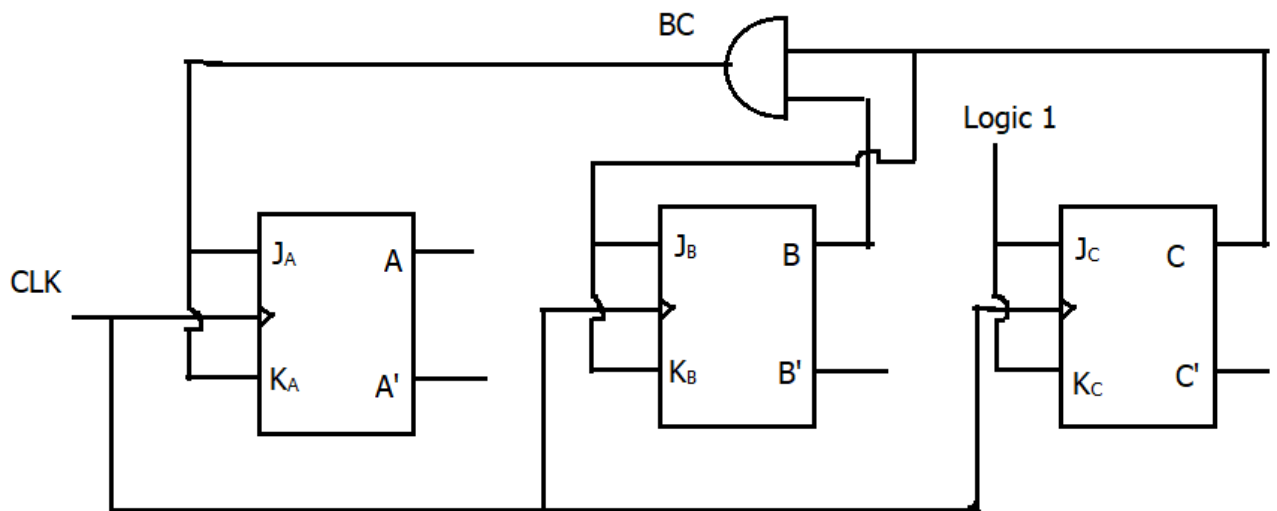


⚙️ Step: 4 Drawing the design

$$J_A = K_A = BC$$

$$J_B = K_B = C$$

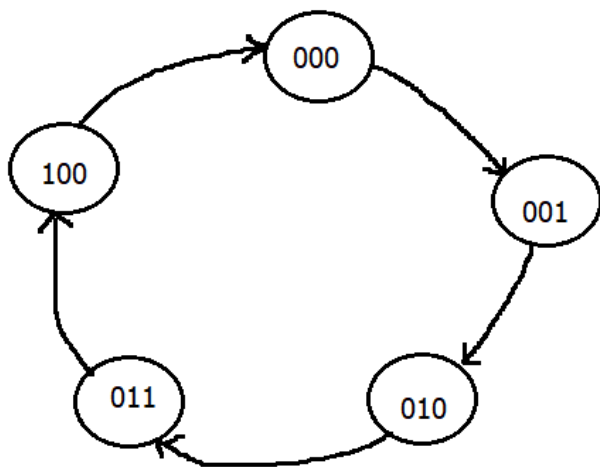
$$J_C = K_C = 1$$



## 2.Design of Mod-5 Counter using T FF

### Step-1: State Diagram

- ✿ Counting Sequence (0-1-2-3-4-0)
- ✿ No.of FFs Required=3



### ✿ Step-2 : Forming State Table

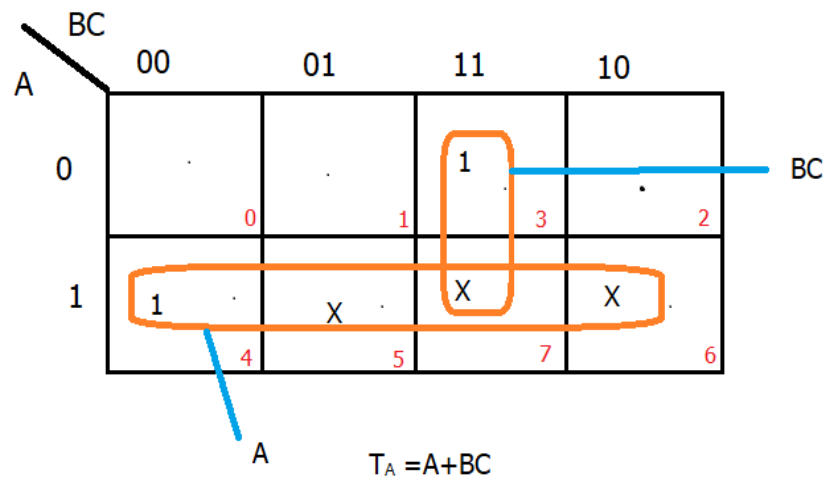
Present State			Next State			FF Input		
A	B	C	A(t+1)	B(t+1)	C(t+1)	T <sub>A</sub>	T <sub>B</sub>	T <sub>C</sub>
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	0	0	0	1	0	0
1	0	1	X	X	X	X	X	X
1	1	0	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

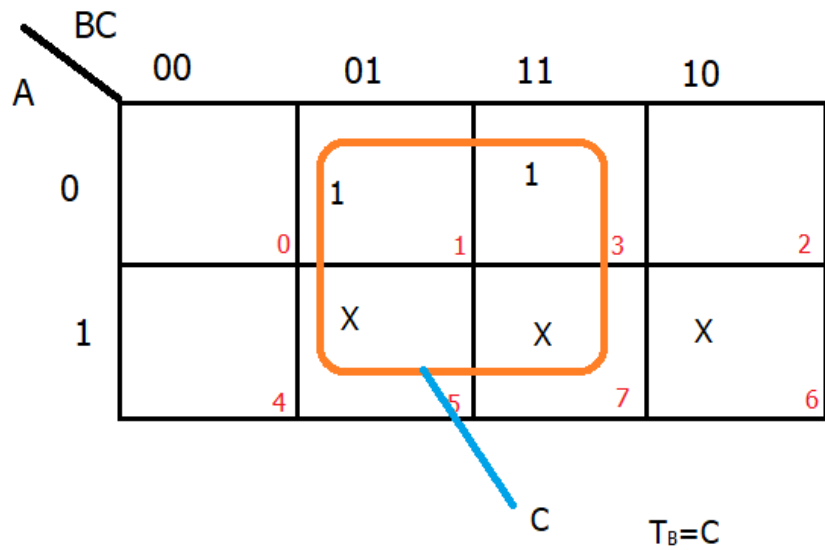
✿ ( Unused States will have don't cares in Next State and FF input)

❁ Step 3: Finding FF input equation using K-map

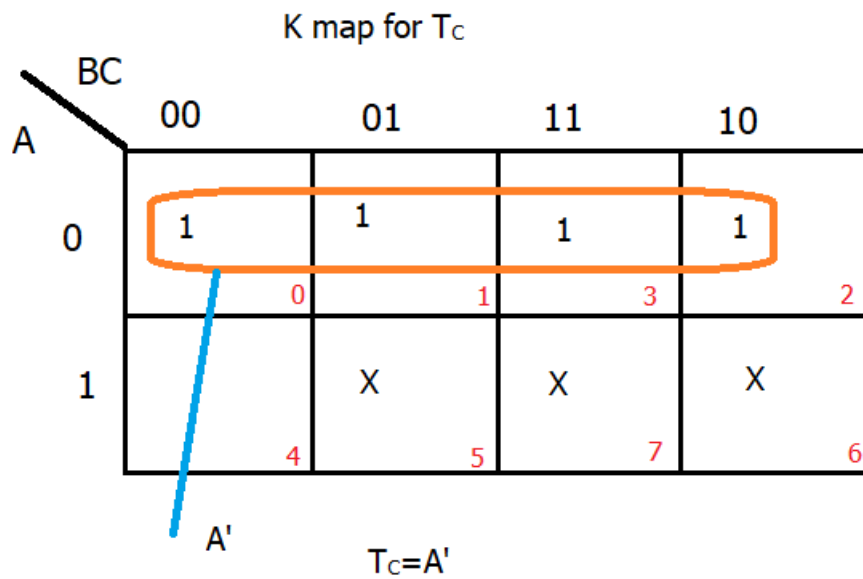
K map for  $T_A$



K map for  $T_B$





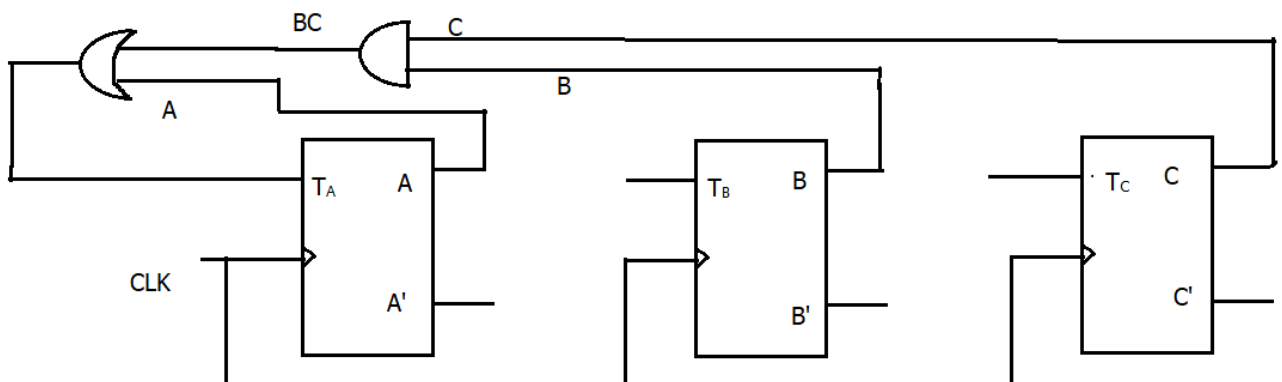


$$T_A = A + BC$$

$$T_B = C$$

$$T_C = A'$$

✿ Step: 4 Drawing the design

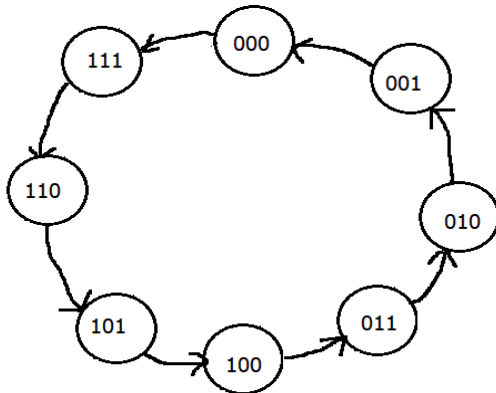


### 3. Design 3 bit down counter using D FF

Step-1 : State Diagram

✿ Counting sequence (0-7-6-5-4-3-2-1-0)

✿ No. of FFs Required=3



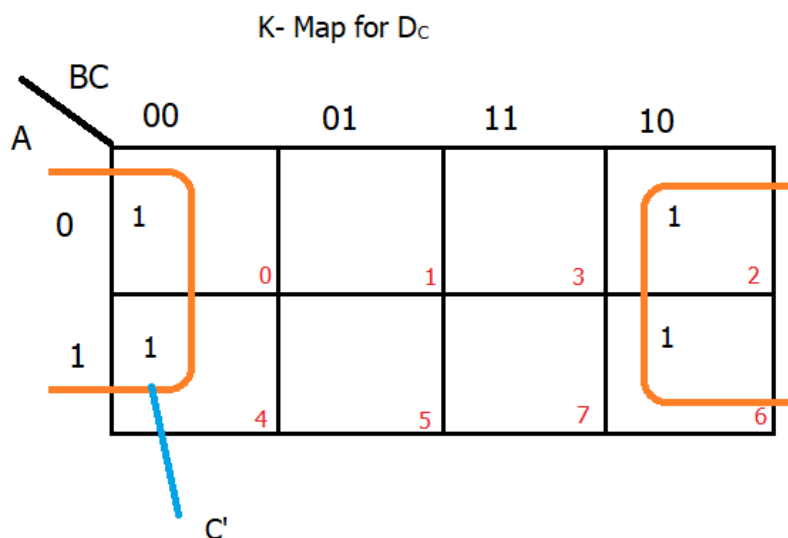
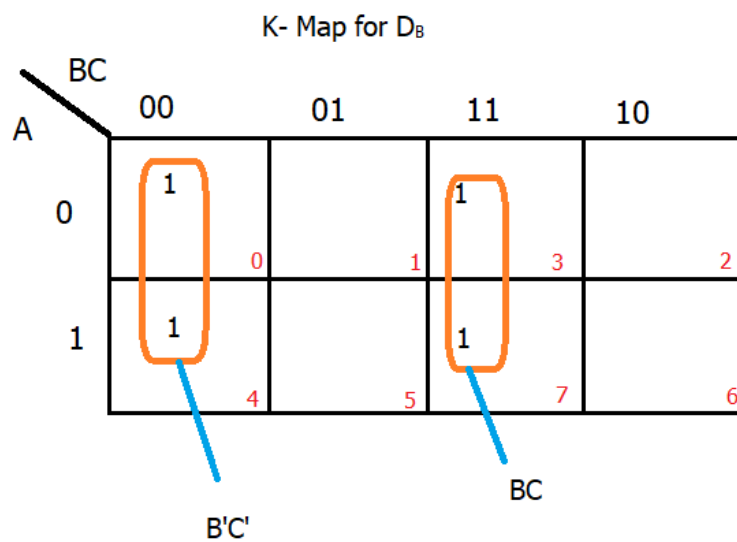
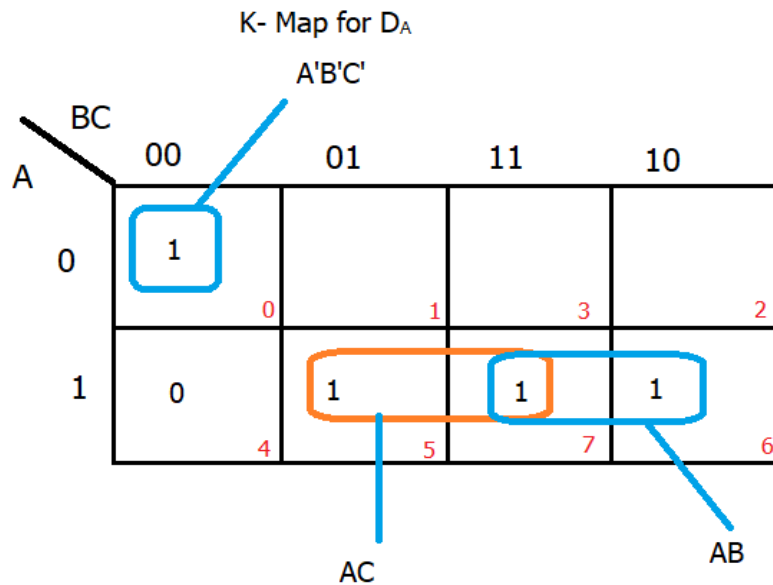
✿ Step-2 : Forming State Table

Present State			Next State			FF Input		
A	B	C	A(t+1)	B(t+1)	C(t+1)	D <sub>A</sub>	D <sub>B</sub>	D <sub>C</sub>
0	0	0	1	1	1	1	1	1
0	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	1
0	1	1	0	1	0	0	1	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	0	1	0	0
1	1	0	1	0	1	1	0	1
1	1	1	1	1	0	1	1	0

Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

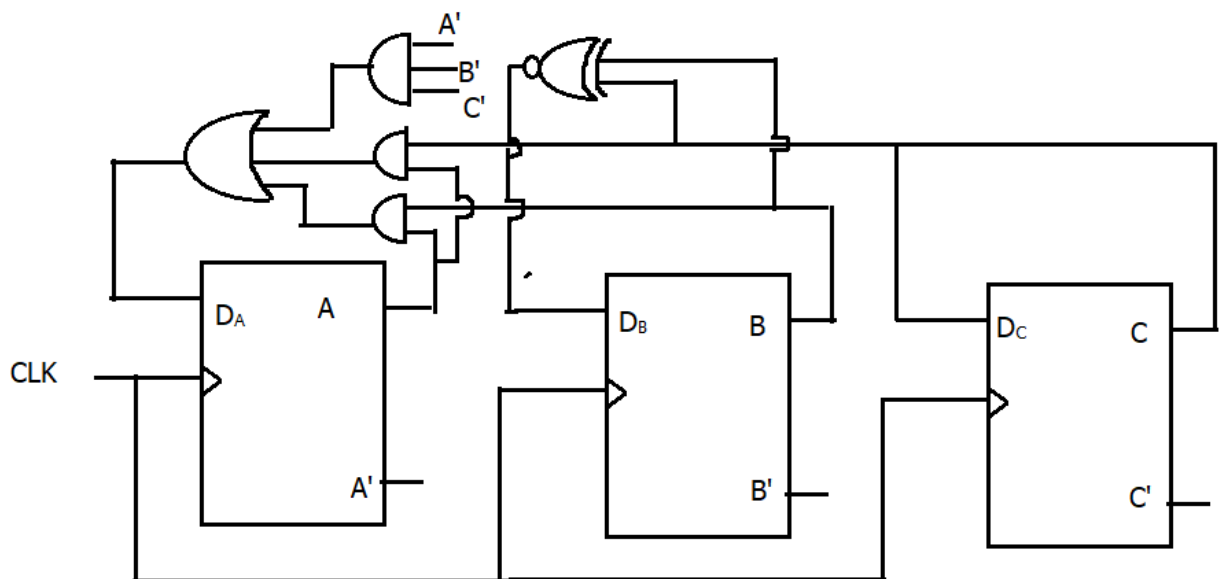
✿ (For D-FF  $D=Q(t+1)$  )

### Step 3: Finding FF input equation using K-map



- ✿  $D_A = A'B'C' + AB + AC$
- ✿  $D_B = B'C' + BC = B \oplus C$
- ✿  $D_C = C$

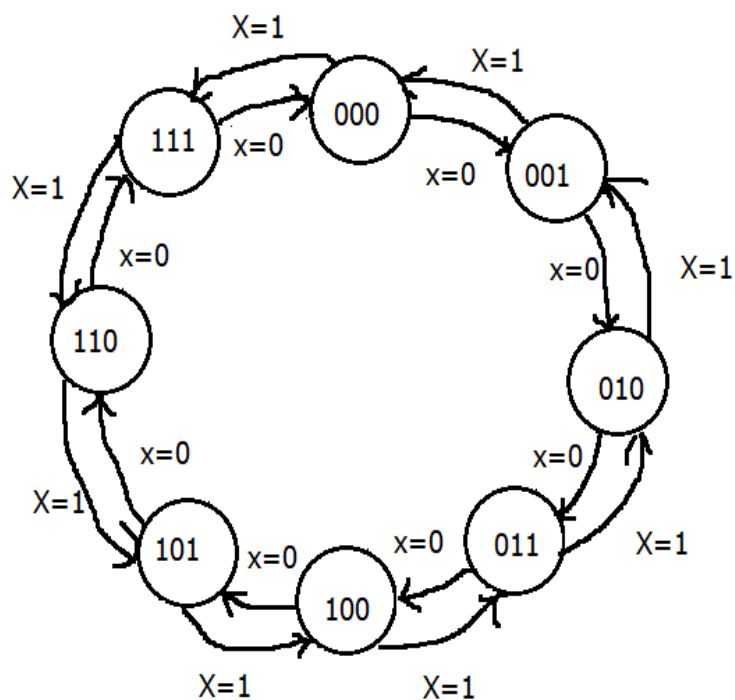
## Step: 4 Drawing the design



## 4. Design 3 bit Bidirectional (Up/Down) 3 bit Counter

### Step-1 : State Diagram

- ✿ Counting sequence :
- ✿ X=0 Up counting ( 0-1-2-3-4-5-6-7-0)
- ✿ X=1 Down counting (0-7-6-5-4-3-2-1-0)
- ✿ No.of FFs Required=3

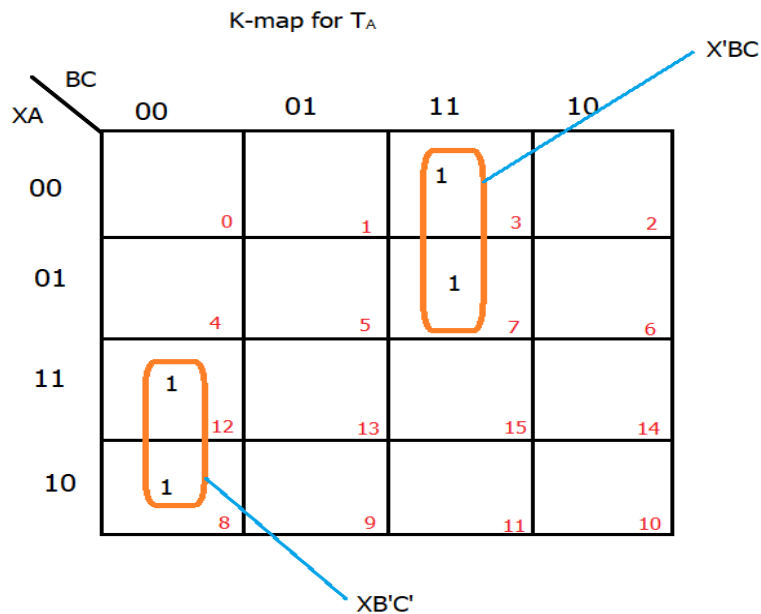


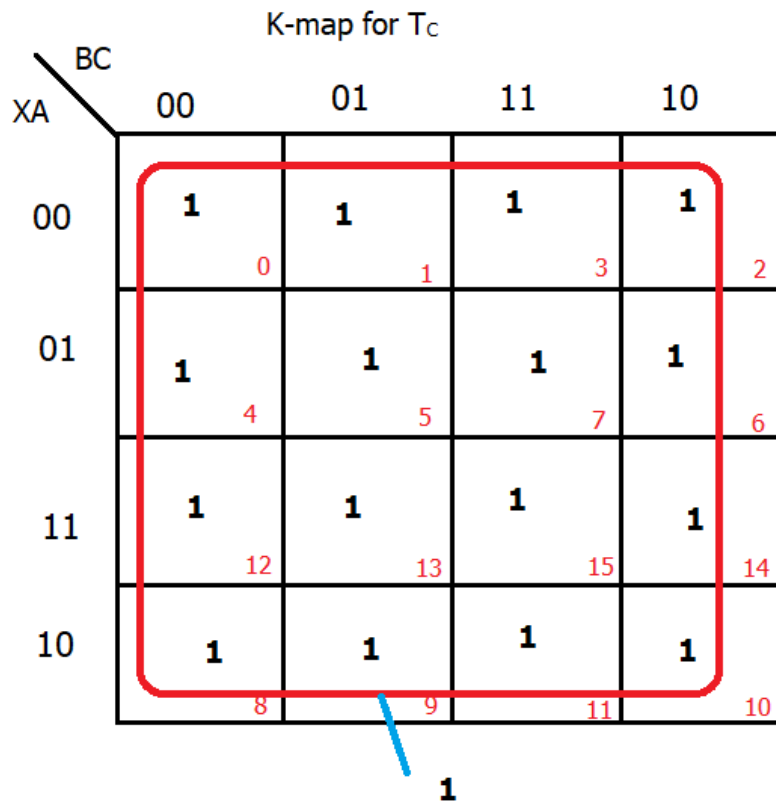
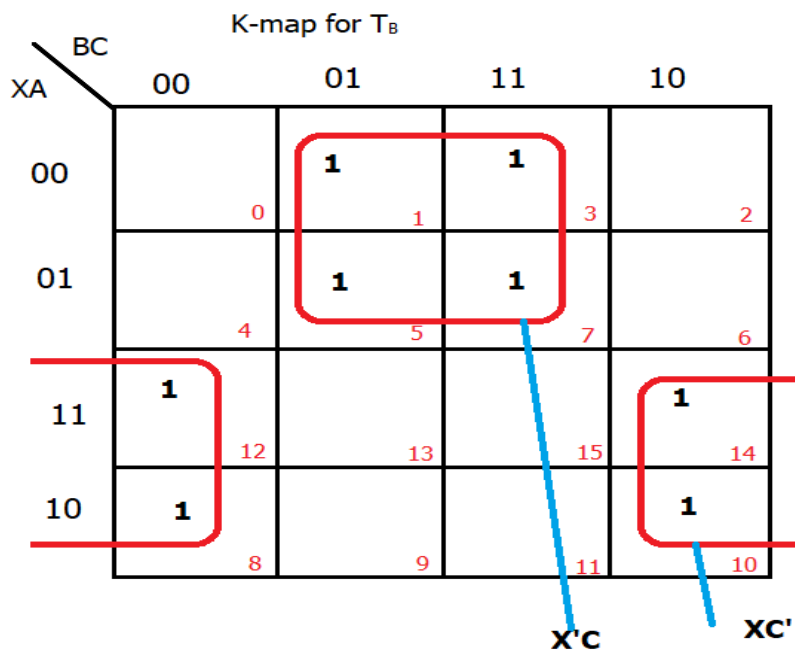
## ❁ Step-2 : Forming State Table

External I/P	Present State			Next State			FF Input		
	A	B	C	A(t+1)	B(t+1)	C(t+1)	T <sub>A</sub>	T <sub>B</sub>	T <sub>C</sub>
0	0	0	0	0	0	1	0	0	1
0	0	0	1	0	1	0	0	1	1
0	0	1	0	0	1	1	0	0	1
0	0	1	1	1	0	0	1	1	1
0	1	0	0	1	0	1	0	0	1
0	1	0	1	1	1	0	0	1	1
0	1	1	0	1	1	1	0	0	1
0	1	1	1	0	0	0	1	1	1
1	0	0	0	1	1	1	1	1	1
1	0	0	1	0	0	0	0	0	1
1	0	1	0	0	0	1	0	1	1
1	0	1	1	0	1	0	0	0	1
1	1	0	0	0	1	1	1	1	1
1	1	0	1	1	0	0	0	0	1
1	1	1	0	1	0	1	0	1	1
1	1	1	1	1	1	0	0	0	1

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

## ❁ Step 3: Finding FF input equation using K-map



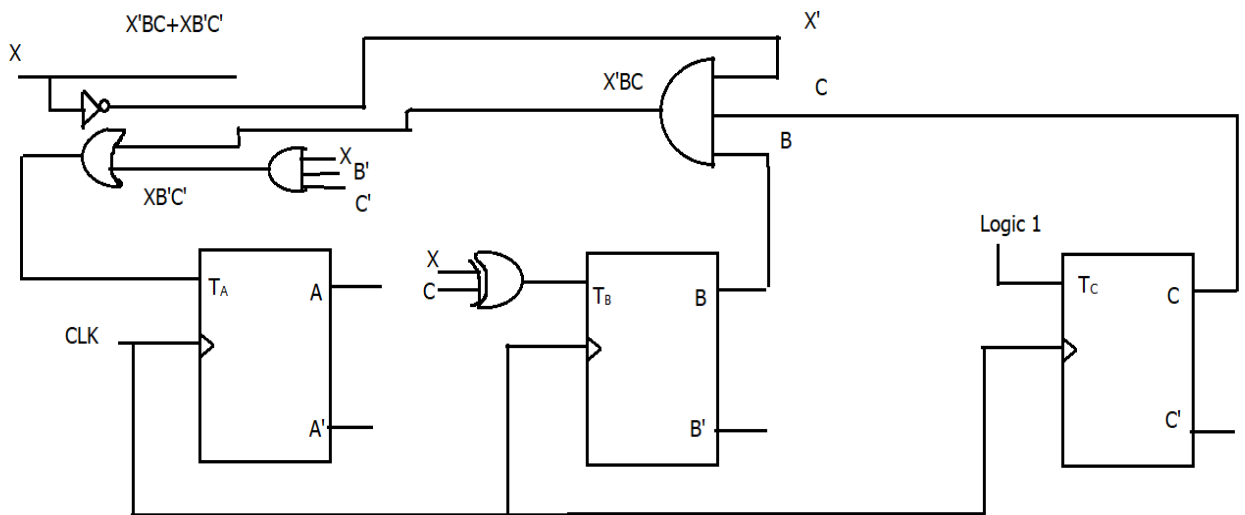


✿  $T_A = XB'C' + X'BC$

✿  $T_B = X'C + XC' = X \oplus C$

✿  $T_C = 1$

## ✿ Step: 4 Drawing the design



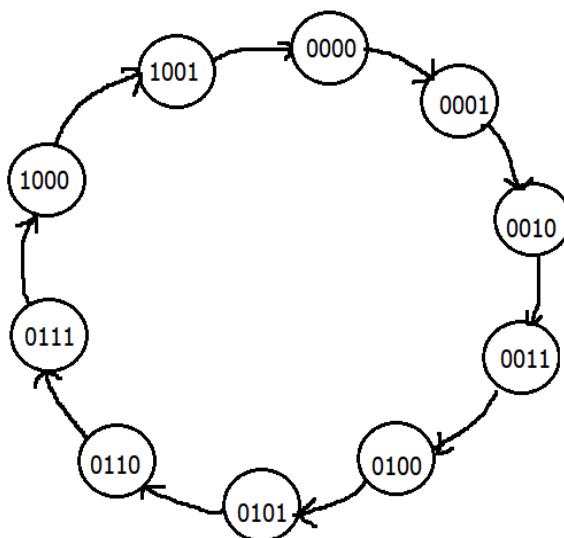
## 5. Design Decade Counter/ MOD-10 Counter/BCD Counter

### ✿ Step-1 : State Diagram

✿ Counting sequence :

✿ ( 0-1-2-3-4-5-6-7-8-9-0)

✿ No.of FFs Required=4

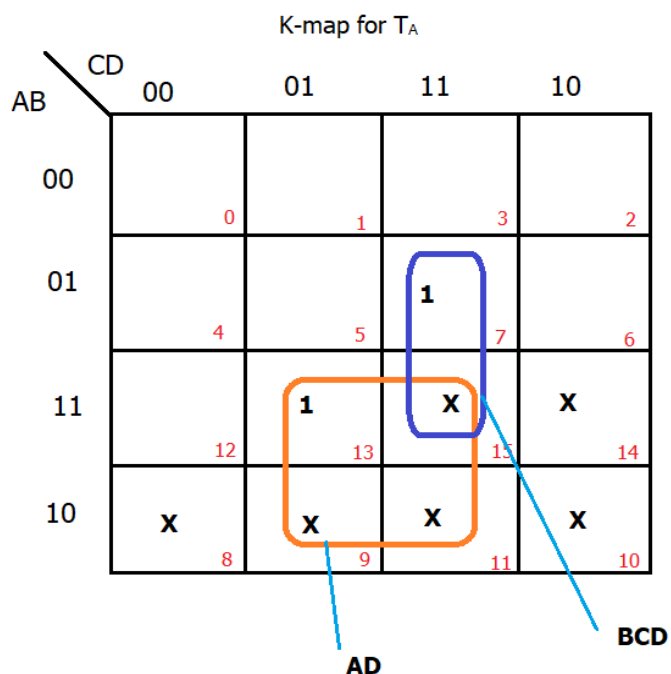




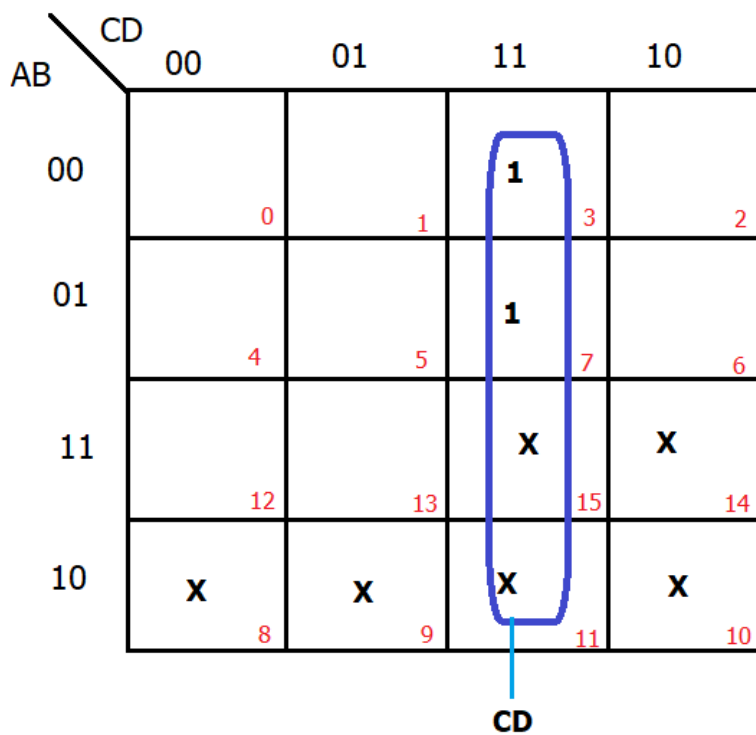
## ❁ Step-2 : Forming State Table

Present State				Next State				FF Input			
A	B	C	D	A(t+1)	B(t+1)	C(t+1)	D(t+1)	T <sub>A</sub>	T <sub>B</sub>	T <sub>C</sub>	T <sub>D</sub>
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	1
1	0	0	1	0	0	0	0	1	0	0	1
1	0	1	0	X	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X

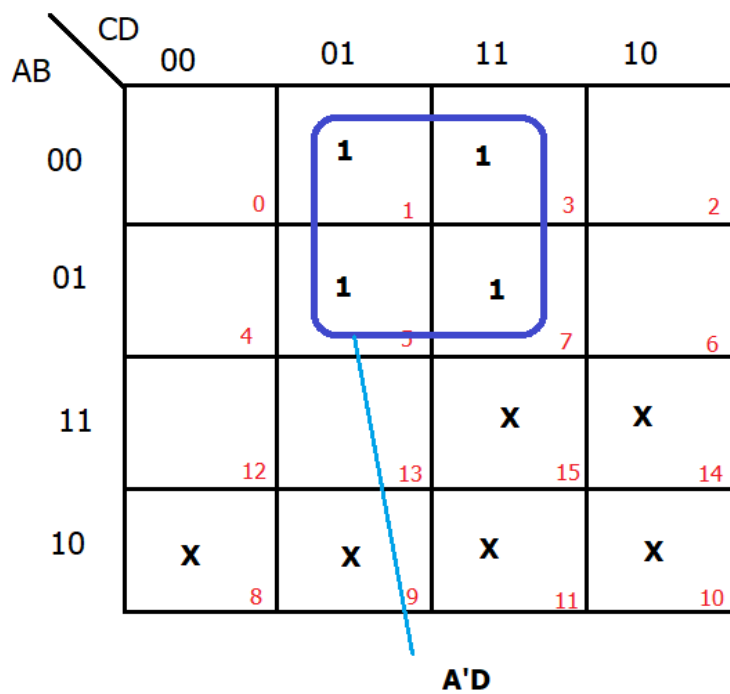
## ❁ Step 3: Finding FF input equation using K-map

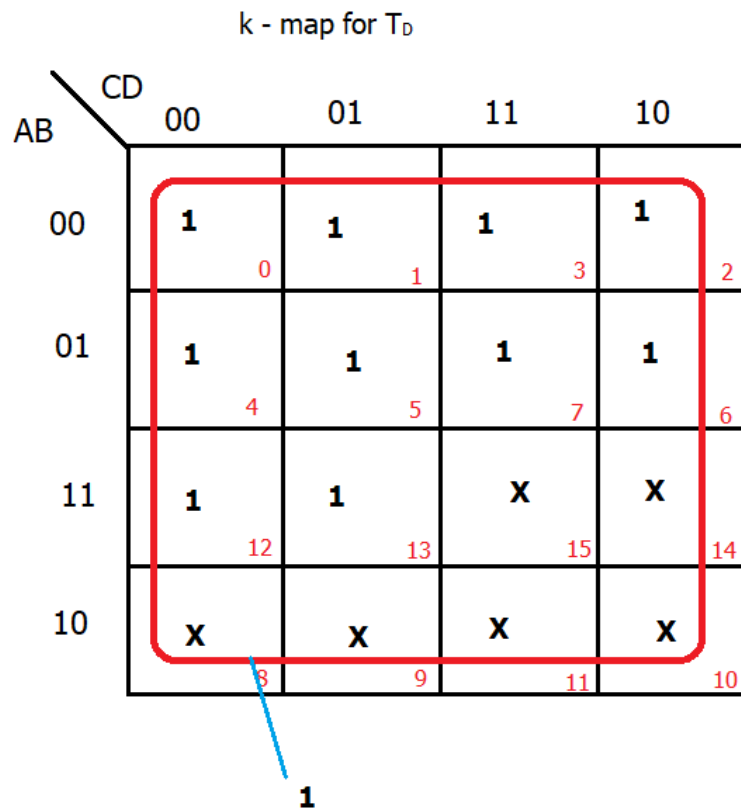


k - map for  $T_B$



k - map for  $T_C$





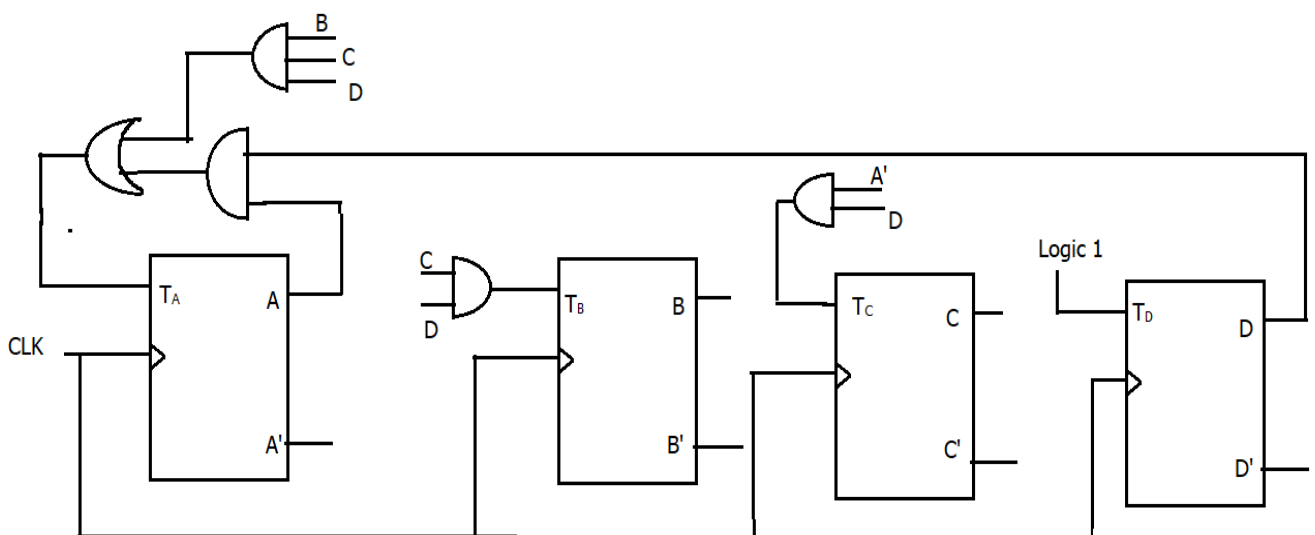
✿  $T_A = AD + BCD$

✿  $T_B = CD$

✿  $T_C = A'D$

✿  $T_D = 1$

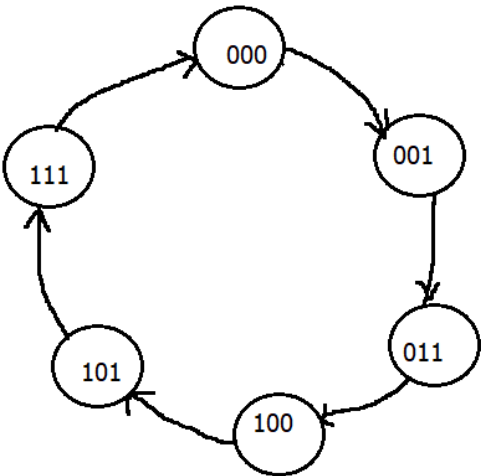
Step: 4 Drawing the design



Design synchronous counter for sequence: 0 → 1 → 3 → 4 → 5 → 7 → 0, using T flip-flop. Check whether it is self starting (Check lock out condition)

Step-1 : State Diagram

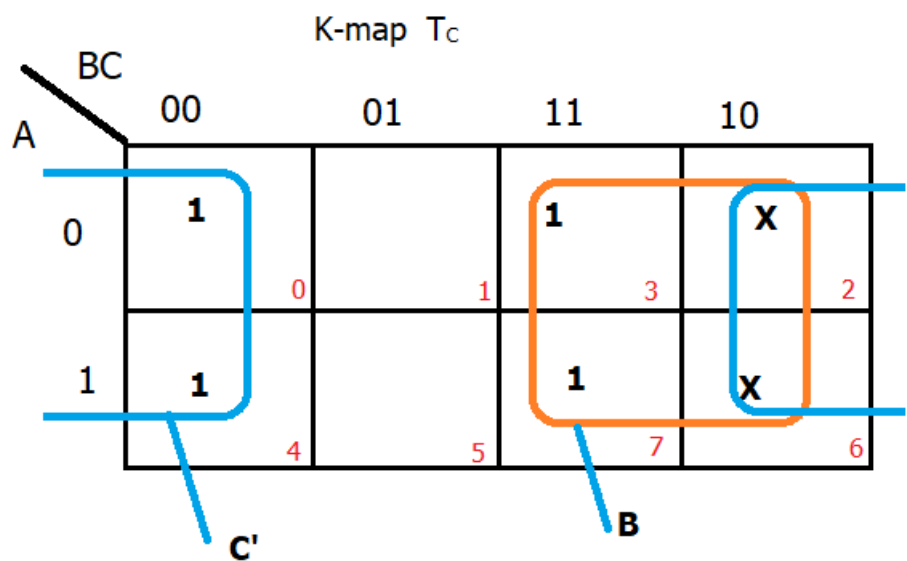
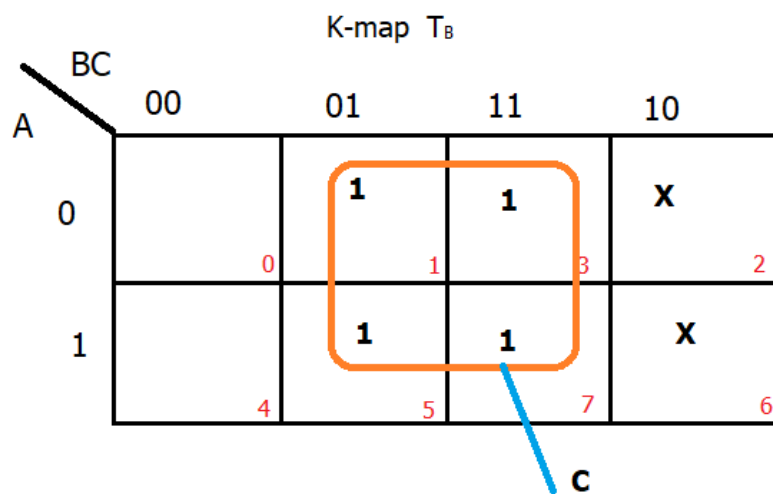
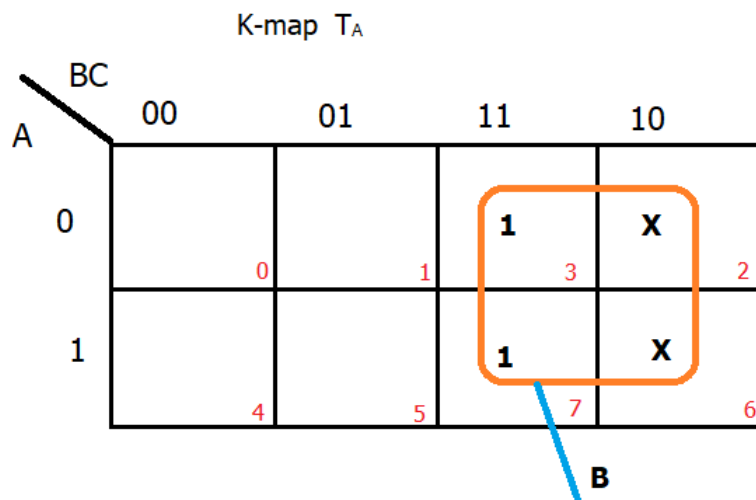
- ✿ Counting sequence :
- ✿ ( 0-1-3-4-5-7-0)
- ✿ No.of FFs Required=3



✿ Step-2 : Forming State Table 0-1-3-4-5-7-0

Present State			Next State			FF Input		
A	B	C	A(t+1)	B(t+1)	C(t+1)	T <sub>A</sub>	T <sub>B</sub>	T <sub>C</sub>
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	0
0	1	0	X	X	X	X	X	X
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	1	0	1	0
1	1	0	X	X	X	X	X	X
1	1	1	0	0	0	1	1	1

### ❁ Step 3: Finding FF input equation using K-map

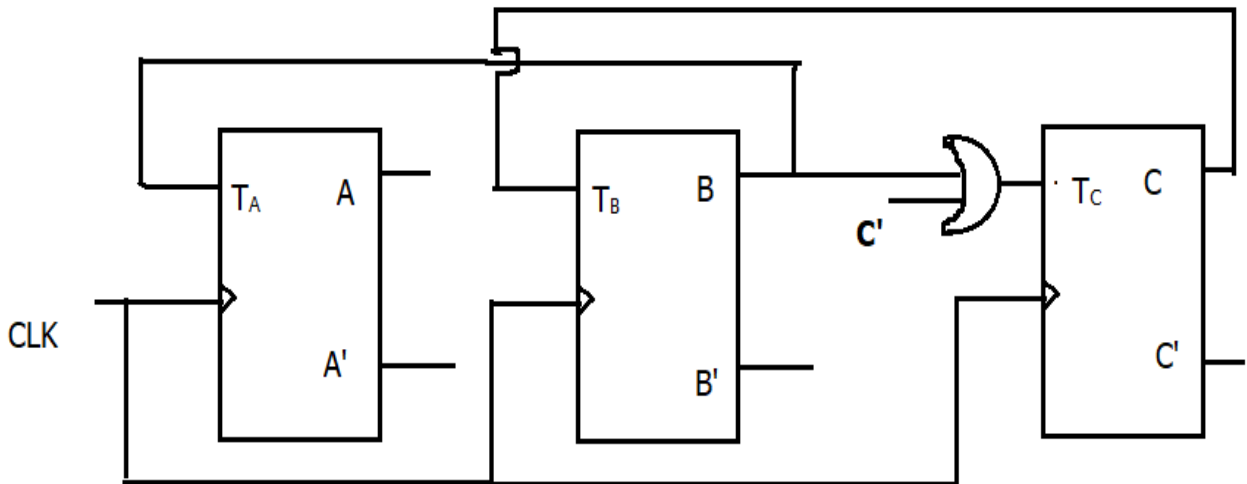


✿  $T_A = B$

✿  $T_B = C$

✿  $T_C = B + C'$

✿ Step: 4 Drawing the design



✿ Step: 5 Checking lock out /self starting:

( From the FF i/p equation, find next state of Unused state )

Unused Present State			FF Input			Next State		
A	B	C	$T_A$	$T_B$	$T_C$	$A(t+1)$	$B(t+1)$	$C(t+1)$
0	1	0	1	0	1	1	1	1
1	1	0	1	0	1	0	1	1

✿ 2-7

✿ 6-3

✿ As the next state of 2 is 7 (used state) and the next state of 6 is 3 (used state), the counter is self starting.

## 4. SHIFT REGISTERS

- ✿ A shift register consists of a group of flip-flops arranged such that the output of one feeds the input of the next so that the binary numbers stored shift from one flip-flop to the next controlled by a clock pulse.
- ✿ This implementation is a 4-bit shift register utilising D-type flip-flops. In this type of circuit, the clock inputs of all the flip-flops connect to a common line, so they receive clock inputs simultaneously.
- ✿ With a D-type flip-flop, the value at the input D transfers to the output Q on the rising edge of every clock pulse. Since they all receive the clock pulse simultaneously, they all do this operation together on the rising edge.
- ✿ A shift register “shifts” its output once every clock cycle

### ✿ **Types of Shift Register**

#### ✿ **Serial-in, Serial-out (SISO)**

In this type of shift register, the binary data is input, shifted and output serially in either left or right direction one bit at a time under a common clock signal.

#### ✿ **Serial-in, Parallel-out (SIPO)**

In this type of shift register, the binary data is input and shifted serially in either left or right direction one bit at a time but is output parallel all together under a clock signal.

#### ✿ **Parallel-in, Serial-out (PISO)**

In this type of shift register, the binary data is input all together in parallel mode but is shifted and output serially in either left or right direction one bit at a time under a clock signal.

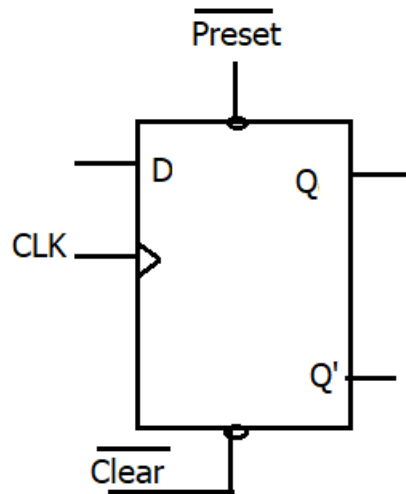
#### ✿ **Parallel-in, Parallel-out (PIPO)**

In this type of shift register, the binary data is input as well as shifted and output in parallel mode all together under a clock signal.

<https://www.petervis.com/dictionary-of-digital-terms/4-bit-shift-register/4-bit-shift-register-animation.html>

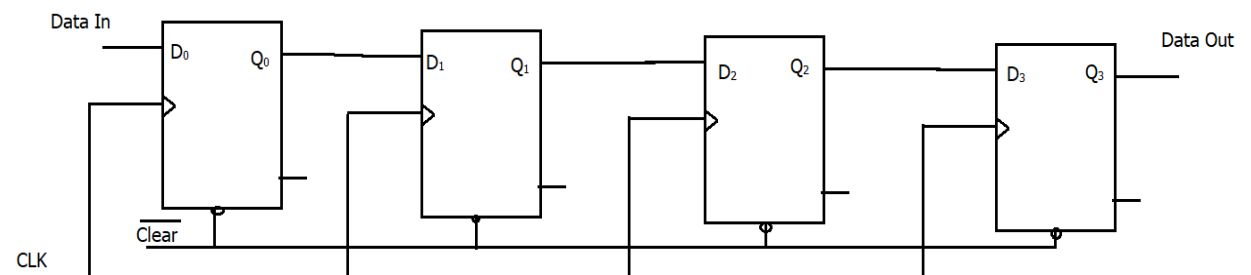
## Asynchronous Inputs:

1. Preset -Sets the FF as 1
2. Clear -Clear the state of the FF



## Serial-in, Serial-out (SISO)

Block Diagram:





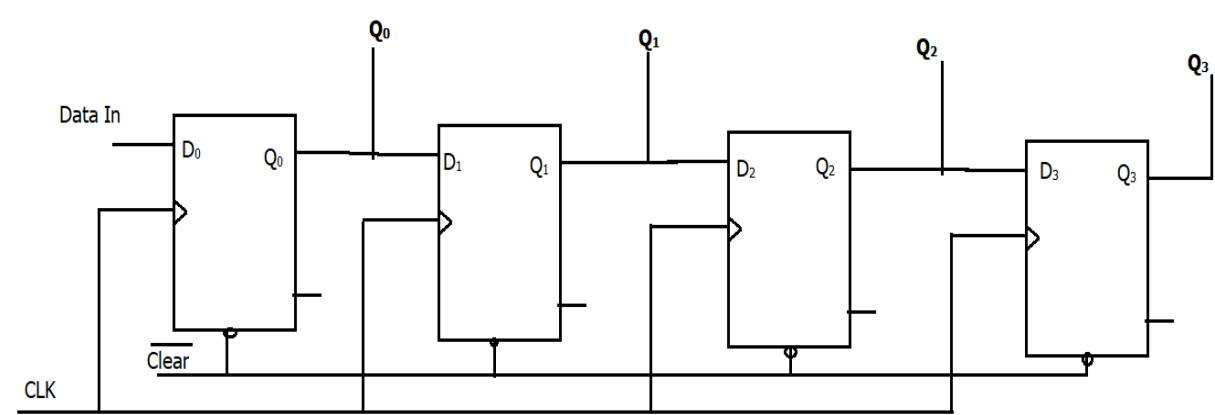
**Data: 1011**

Shift Register Truth Table

Outputs	Q <sub>0</sub>	Q <sub>1</sub>	Q <sub>2</sub>	Q <sub>3</sub>
Clear	0	0	0	0
CK Pulse 1	1	0	0	0
CK Pulse 2	1	1	0	0
CK Pulse 3	0	1	1	0
CK Pulse 4	1	0	1	1
CK Pulse 5	x	1	0	1
CK Pulse 6	x	x	1	0
CK Pulse 7	x	x	x	1

**Serial-in, Parallel-out (SIPO)**

Block Diagram:



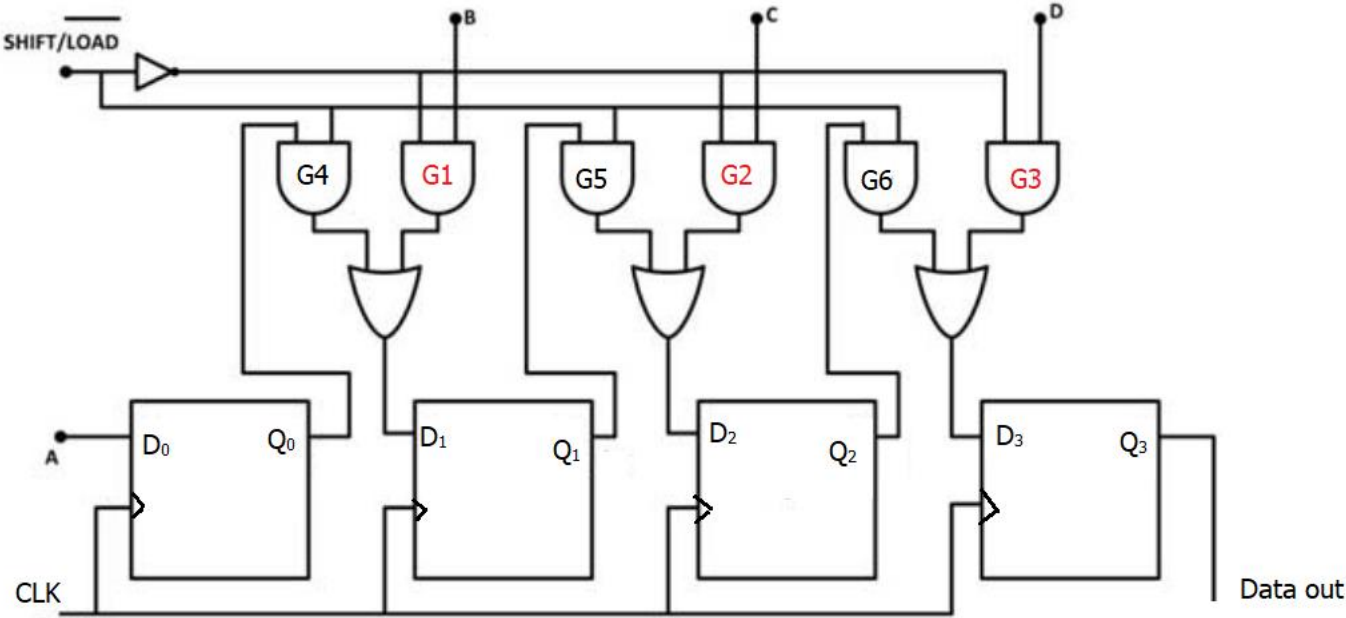
Data :1001

Shift Register Truth Table

Outputs	Q <sub>0</sub>	Q <sub>1</sub>	Q <sub>2</sub>	Q <sub>3</sub>
Clear	0	0	0	0
CK Pulse 1	1	0	0	0
CK Pulse 2	0	1	0	0
CK Pulse 3	0	0	1	0
CK Pulse 4	1	0	0	1

Parallel-in, Serial-out (PISO)

Block Diagram:



- ❁ SHIFT/LOAD is a control input that allows the four bit of data at A, B, C and D inputs to enter into the register in parallel or shift the data in serial.
- ❁ When SHIFT/LOAD is LOW, AND gates G1 ,G2, G3 are enabled, allowing the data at parallel inputs i.e. A, B, C and D to the Data inputs of the respective flip-flops.
- ❁ When SHIFT/LOAD is HIGH, AND gates G1, G2, G3 are disabled and the remaining AND gates G4 G5 G6 are enabled, allowing the data bits to shift right from one stage to the next.
- ❁ The OR gates allow either the normal shifting operation or the parallel data-entry operation, depending on which the AND gates are enabled by the level on the SHIFT/LOAD input.

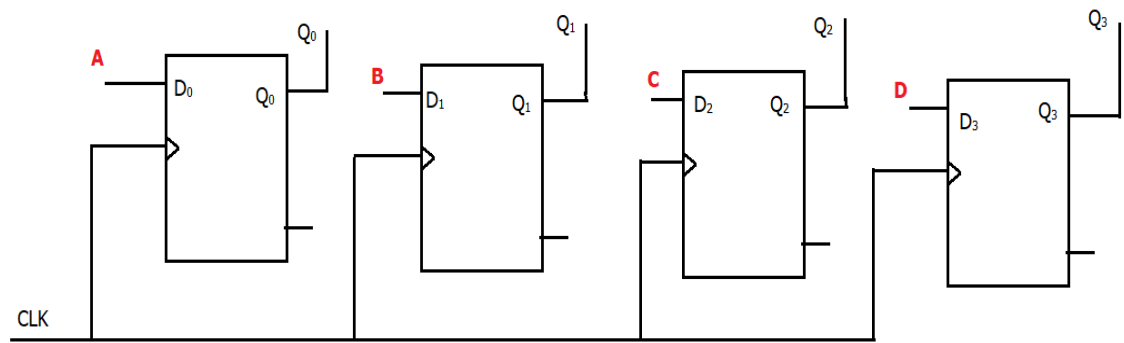
❁ **Data: 1101**

❁ **Shift Register Truth Table**

Outputs	Q <sub>0</sub>	Q <sub>1</sub>	Q <sub>2</sub>	Q <sub>3</sub>
Reset	0	0	0	0
CK Pulse 1	1	1	0	1
CK Pulse 2	X	1	1	0
CK Pulse 3	x	x	1	1
CK Pulse 4	x	x	x	1

# Parallel-in, Parallel-out (PIPO)

Block Diagram:



❁ **Data: 1101**

❁ **Shift Register Truth Table**

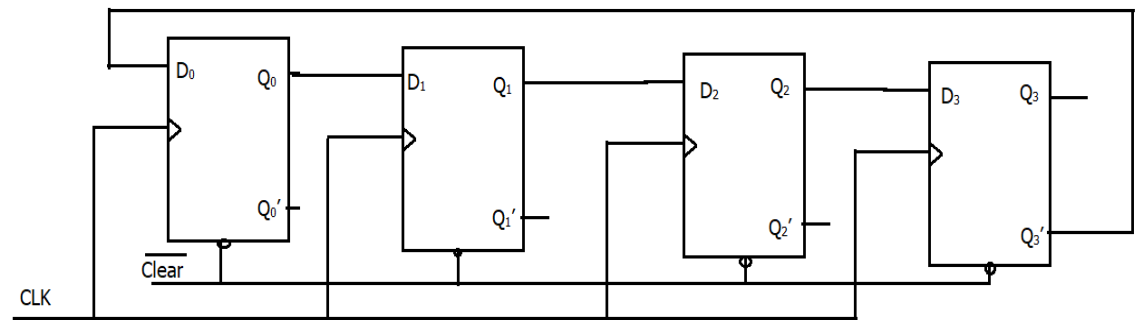
Outputs	Q <sub>0</sub>	Q <sub>1</sub>	Q <sub>2</sub>	Q <sub>3</sub>
Reset	0	0	0	0
CK Pulse 1	1	1	0	1

❁ **Application of Shift Registers:**

❁ **Johnson Counter/Twisted Ring Counter:**

❁ Block Diagram

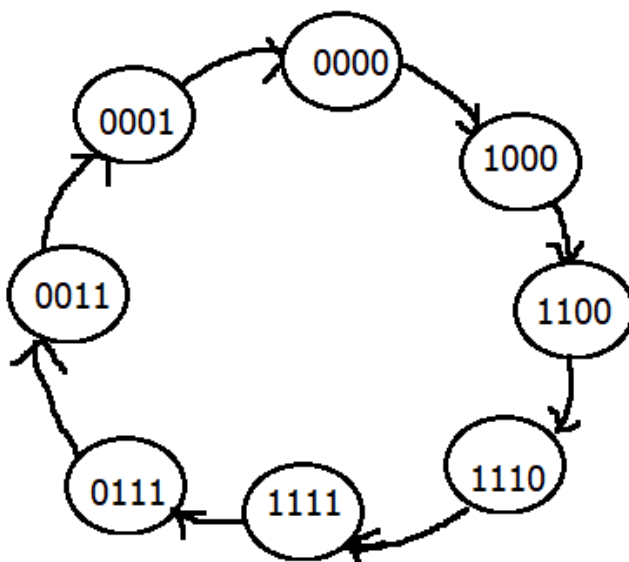
❁ Complement of the output of the last flip-flop is connected back to the *D* input of the first flip-flop



- ✿ If the counter starts at 0, this feedback arrangement produces a characteristic sequence of states,
- ✿ 4-bit sequence has a total of eight states
- ✿ 5-bit sequence has a total of ten states
- ✿ Johnson counter will produce a modulus of  $2n$ , where  $n$  is the number of stages in the counter
- ✿ Truth Table:

Outputs	$Q_0$	$Q_1$	$Q_2$	$Q_3$
Clear	0	0	0	0
CK Pulse 1	1	0	0	0
CK Pulse 2	1	1	0	0
CK Pulse 3	1	1	1	0
CK Pulse 4	1	1	1	1
CK Pulse 5	0	1	1	1
CK Pulse 6	0	0	1	1
CK Pulse 7	0	0	0	1
CK Pulse 8	0	0	0	0

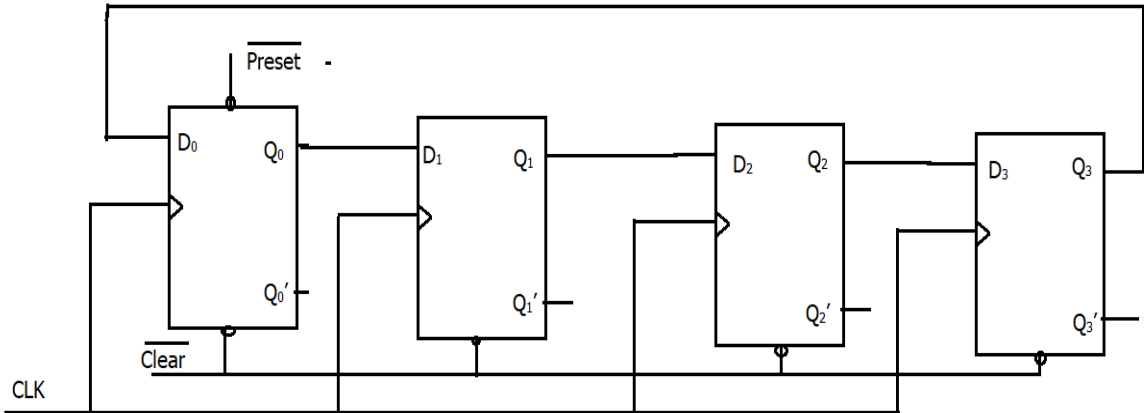
- ✿ State Diagram:



## ❁ Ring Counter:

- ❁ Ring counter has one flip-flop for each state in its sequence
- ❁ Initially, a 1 is preset into the first flip-flop
- ❁ Remaining flip-flops are cleared

## ❁ Block Diagram

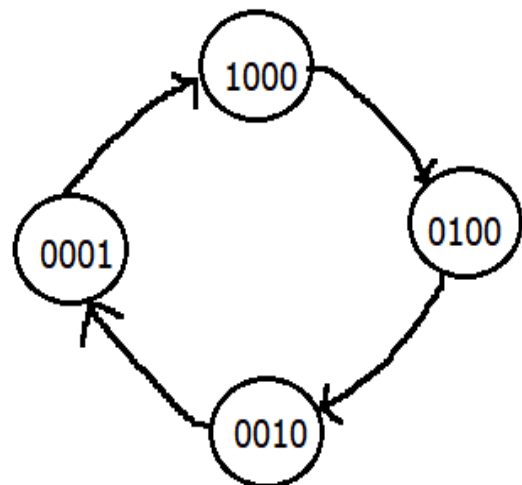


- ❁ Inter stage connections are the same as Johnson counter
- ❁ But  $Q$  rather than  $Q'$  is fed back from the last stage
- ❁ Outputs of the counter indicate directly the decimal count of the clock pulse
- ❁ 1 is always retained in the counter and simply shifted "around the ring," advancing one stage for each clock pulse

## ❁ Truth Table

Outputs	$Q_0$	$Q_1$	$Q_2$	$Q_3$
Clear	0	0	0	0
Preset	1	0	0	0
CK Pulse 1	0	1	0	0
CK Pulse 2	0	0	1	0
CK Pulse 3	0	0	0	1
CK Pulse 4	1	0	0	0

## State Diagram



## 5. DESIGN OF SYNCHRONOUS SEQUENTIAL CIRCUITS

### Synchronous sequential circuits:

There are two types.

1. Moore Circuit

2. Mealy Circuit

Moore Circuit: The output depends only on the present state of the flip-flops.

Mealy Circuit: The output depends on both the present state and the external inputs of the flip-flops.

Moore Circuit	Mealy Circuit
<ul style="list-style-type: none"><li>▪It's output is a function of present state only.</li><li>▪Input changes does not affect the output.</li><li>▪It requires more number of states for implementing the same function.</li></ul>	<ul style="list-style-type: none"><li>▪It's output is a function of present state as well as external inputs.</li><li>▪Input changes may affect the output of the circuit.</li><li>▪It requires less number of states for implementing same function.</li></ul>

Problems:

1. A sequential circuit with 2 D Flip-flops A&B has 2 inputs x &y and 1 output Z is specified by following next state and output equations. Draw state diagram and logical diagram.

$$\begin{aligned}A(t+1) &= \overline{x}y + xa \\ B(t+1) &= \overline{x}b + xa \\ Z &= b\end{aligned}$$

**Solution:**

Here the output doesn't depend upon the external inputs x &y.

Number of inputs = 2

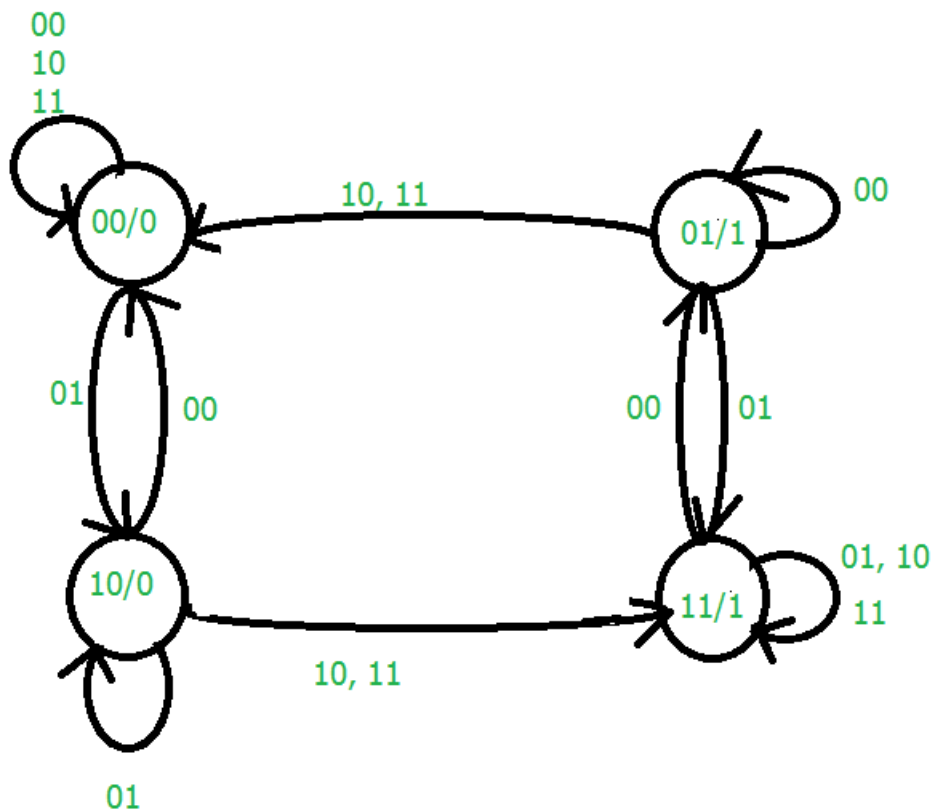
Number of outputs = 1

Number of flip-flops = 2





### Step 3: Derive the State diagram



2. A sequential circuit with 2 – D flip-flops A & B and input x and output Y is specified by the following next state and output equations.

$$A(t+1) = Ax + Bx$$

$$B(t+1) = A'x$$

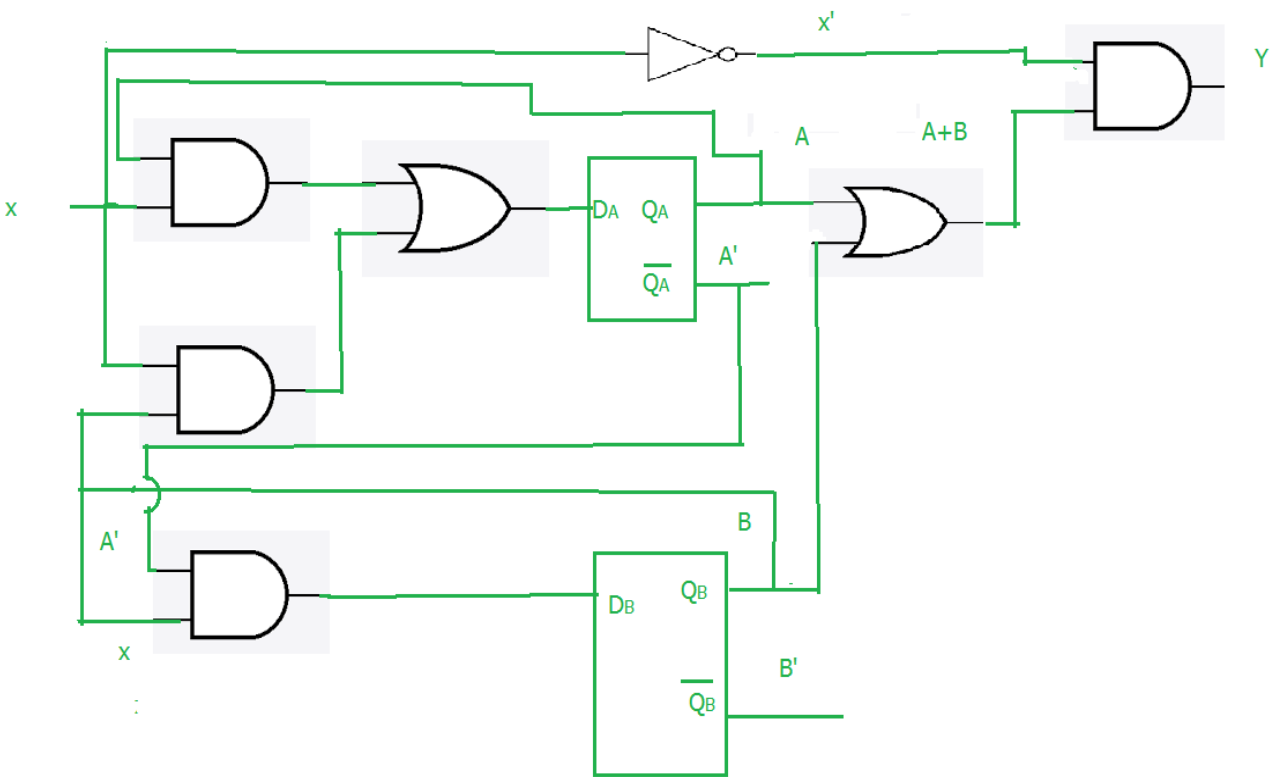
$$Y = (A+B) X'$$

- Draw the logic diagram of the circuit.
- Derive the state table.
- Derive the state diagram.

**Solution:**

Number of inputs – 1, Number of flip-flops – 2, Number of output - 1

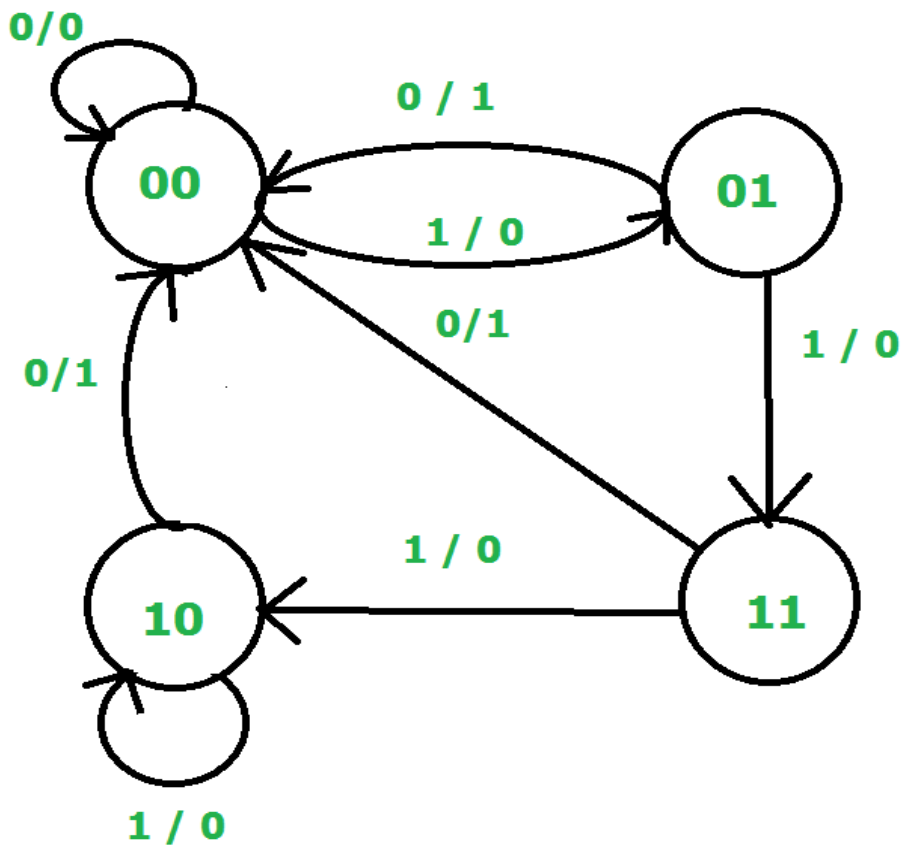
Step1: Draw the Logic diagram of the circuit



Step 2: Derive the state table

Present State		Next State		Output Y	
A	B	$x = 0$ A+ B+	$x = 1$ A+ B+	$x = 0$	$x = 1$
0	0	0 0	0 1	0	0
0	1	0 0	1 1	1	0
1	0	0 0	1 0	1	0
1	1	0 0	1 0	1	0

### Step 3: Derive the state diagram



### State Diagram:

- ❑ The information available in a state table may be represented graphically in a state diagram.
- ❑ A state is represented by a circle and the transition between states is indicated by directed lines connecting the circles.
- ❑ The directed lines are labeled with two binary numbers separated by a /. The input value that causes the state transition is labeled first and the output after the symbol / gives the value of the output during the present state.
- ❑ This is pictorial view of a state table derived from the given logic diagram.

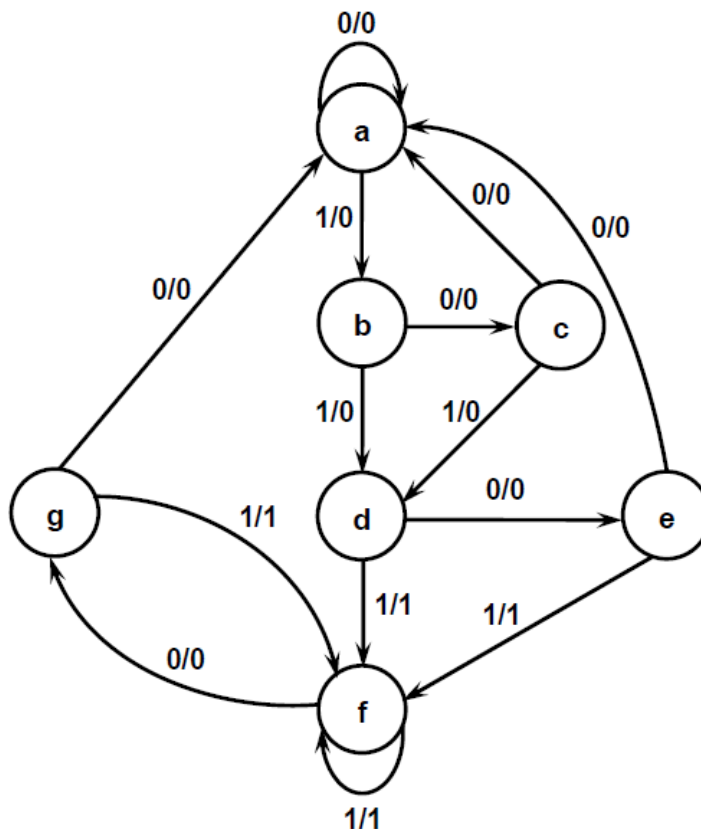
## State Reduction:

The reduction of the number of flip-flops in a sequential circuit is referred to as the state reduction problem. State-reduction algorithms are concerned with procedures for reducing the number of states in a state table, while keeping the external input-output requirements unchanged.

Since  $(N)$  flip-flops produce  $(2^N)$  states, a reduction in the number of states may (or may not) result in a reduction in the number of flip-flops. An unpredictable effect in reducing the number of flip-flops is that sometimes the equivalent circuit (with fewer flip-flops) may require more combinational gates.

We will illustrate the state reduction procedure with an example. We start with a sequential circuit whose specification is given in the state diagram shown in Fig. In this example, only the input-output sequences are important; the internal states are used merely to provide the required sequences.

For this reason, the states marked inside the circles are denoted by letter symbols instead of their binary values. This is in constant to a binary counter, where the binary value sequence of the state themselves is taken as the outputs.



- ❁ There are an infinite number of input sequences that may be applied to the circuit; each results in a unique output sequence. As an example, consider the input sequence [01010110100] starting from the initial state (a). Each input of 0 or 1 produces an output of 0 or 1 and causes the circuit to go to the next state. the output and state sequence for the given input sequence as follows: With the circuit in initial state (a), an input of 0 produces an output of 0 and the circuit remains in state (a). With present state (a) and input of 1, the output is 0 and the next state is (b). With present state (b) and input of 0, the output is 0 and next state is (c). Continuing this process, we find the complete sequence to be as follows:

State	a	a	b	c	d	e	f	f	g	f	g	a
Input	0	1	0	1	0	1	1	0	1	0	0	
Output	0	0	0	0	0	1	1	0	1	0	0	

- ❁ In each column, we have the present state, input value, and output value. The next state is written on top of the next column. It is important to realize that in this circuit, the states themselves are of secondary importance because we are interested only in output sequences caused by input sequences.
- ❁ Now let us assume that we have found a sequential circuit whose state diagram has less than seven states and we wish to compare it with the circuit whose state diagram is given by Fig. (1).
- ❁ If identical input sequences are applied to the two circuits and identical outputs occur for all input sequences, then the two circuits are said to be equivalent (as far as the input-output is concerned) and one may be replaced by the other. The problem of state reduction is to find ways of reducing the number of states in a sequential circuit without altering the input-output relationships.
- ❁ We now proceed to reduce the number of states for this example. First, we need the state table; it is more convenient to apply procedures for state reduction using a table rather than a diagram. The state table of the circuit is listed in Table (1) and is obtained directly from the state diagram.

State table:

Present State	Next State		Output	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

e, g are Equal states

**Equal states:** Two states are said to be equivalent if, for each member of the set of inputs, they give exactly the same output and send the circuit either to the same state or to an equivalent state.

**Reduced State table -1** [ Note 'e' and 'g' are equal states. So in the reduced table 'g' replaced by 'e'

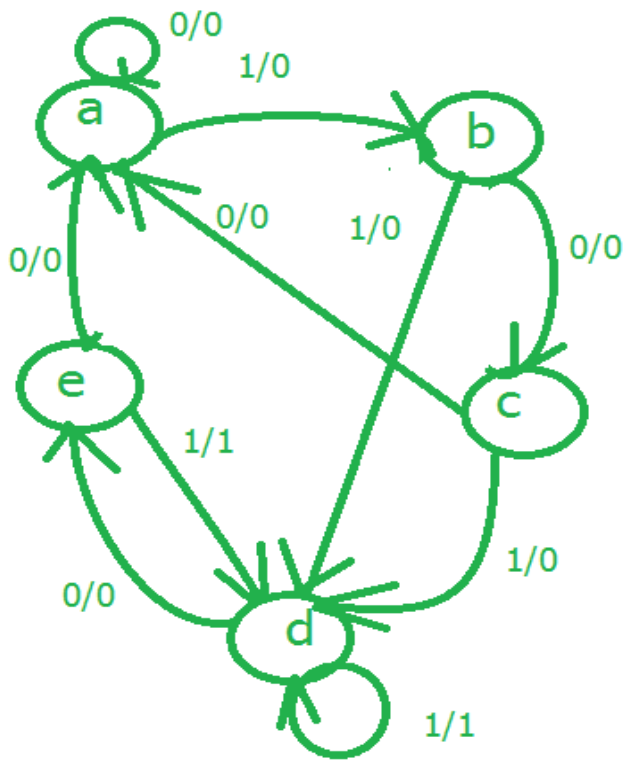
Present State	Next State		Output	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	e	f	0	1

d, f are Equal states

Reduced State table -2 [ Note 'd' and 'f' are equal states. So in the reduced table 'f' replaced by 'd'

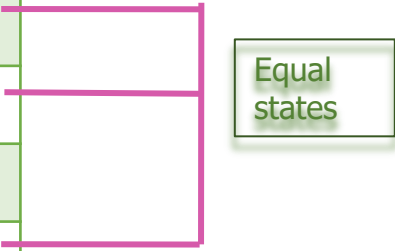
Present State	Next State		Output	
	x=0	x=1	x=0	x=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

Reduced State diagram:



1.A sequential circuit has one input and one output, Minimize the following state table and then draw the state diagram.

Present state	Next state, Z(output), X(Input)	
	X=0	X=1
A	A,0	D,1
B	C,1	D,0
C	B,0	A,1
D	D,1	A,1
E	D,1	A,1
F	D,0	A,0
G	D,1	A,1
H	D,1	C,1



Equal states

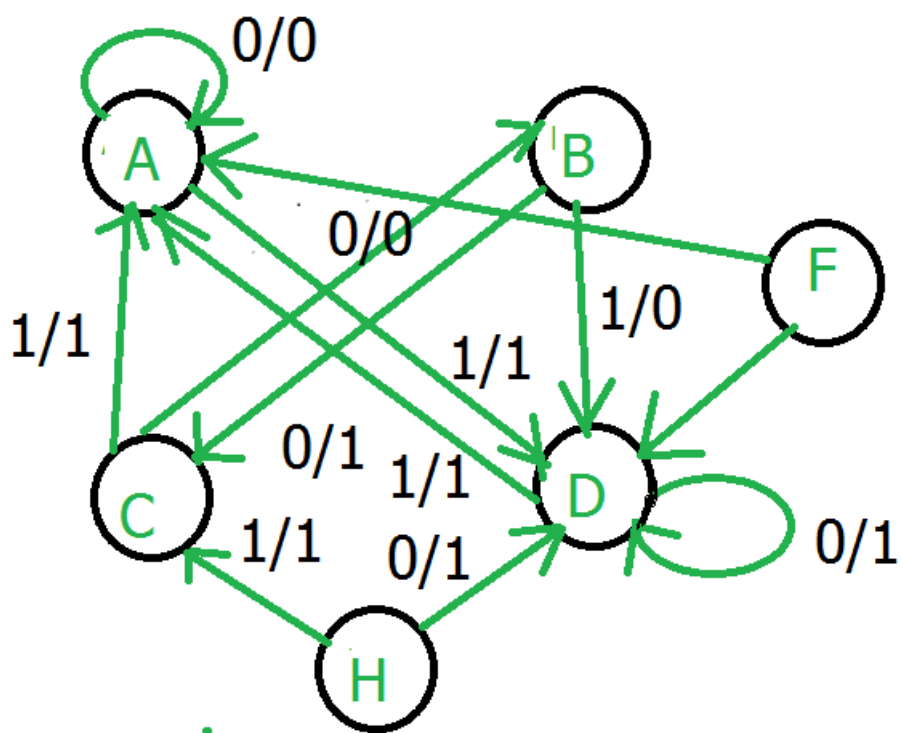
Solution:

Step1: Derive the reduced state table

Present state	Next state, Z(output), X(Input)	
	X = 0	X = 1
A	A,0	D,1
B	C,1	D,0
C	B,0	A,1
D	D,1	A,1
F	D,0	A,0
H	D,1	C,1



Step2: Derive the state diagram



## 9. Lecture Notes

### ✿ E-Books

Digital Fundamentals\_ Global Ed - Thomas L Floyd

Digital Principles And Application - Leach & Malvino

Digital Design - M. Morris Mano and Michael D. Ciletti

Fundamentals of Digital Logic with Verilog Design-Stephen Brown and Zonko Vranesic

### ✿ ONLINE LEARNING MATERIALS:

**1:** [http://nptel.iitm.ac.in/video.php? subject Id=117106086](http://nptel.iitm.ac.in/video.php?subject%20Id=117106086)

**2:** <http://nptel.iitm.ac.in/courses/117101001>

**3:** <https://youtu.be/C-oAyXibnJU>

**4** <https://youtu.be/oYRMYSIVj1o>

**5:** <https://www.youtube.com/watch?v=XZmGGAbHqa0>

**6:** <https://www.youtube.com/watch?v=KymIDyQiXZI>

### ✿ Video Links

<https://drive.google.com/open?id=1qCP5dBvi1LZxd6S7FAUxt788iMmLLpbq>

<https://drive.google.com/open?id=1hrXWfXKWnhidFbnDPDtd75hQt9OzOG6O>

## ❁ QUIZ QUESTIONS:

**1. What is/are the crucial function/s of memory elements used in the sequential circuits?**

- a. Storage of binary information
- b. Specify the state of sequential
- c. Both a & b
- d. None of the above

**2. The behaviour of synchronous sequential circuit can be predicted by defining the signals at \_\_\_\_\_.**

- a. discrete instants of time
- b. continuous instants of time
- c. sampling instants of time
- d. at any instant of time

**3. In Moore models, output are the function of only**

- a.** present state
- b.** input state
- c. next state
- d.** both a and b

**4. Memory elements in clocked sequential circuits are called**

- a.**latches
- b.**flip-flop
- c.signals
- d.**gates

**5.The state of flip-flop can be switched by changing its**

- a.**input signal
- b.**output signal
- c.momentary signals
- d.**all signals

6.The flip-flops can be constructed with two

- a.**NAND gates
- b.**XOR gates
- c.**AND gates
- d.**NOT gates

7.Unused states are treated as Don't cares conditions during the

- a.**Design of a circuit
- b.**Execution
- c.**Pulse trigger
- d.**None

8. .A synchronous sequential circuit is made up of

- a.**combinational gates
- b.**flip-flops
- c.**latches
- d.**both a and b

9. In the last step of design procedure we

- a.**draw map
- b.**draw circuit
- c.**draw table
- d.**draw a logic diagram

10. In T flip-flop when state of the T flip-flop has to be complemented the T must be

- a.**0
- b.**1
- c.**t
- d.**t+1

## 10. Assignments

✿ Design a sequential circuit with two *JK* flip-flops, *A* and *B*, and two inputs, *E* and *x*. If *E* = 0, the circuit remains in the same state regardless of the value of *x*. When *E* = 1 and *x* = 1, the circuit goes through the state transitions from 00 to 01 to 10 to 11 back to 00, and repeats. When *E* = 1 and *x* = 0, the circuit goes through the state transitions from 00 to 11 to 10 to 01 back to 00, and repeats.

✿ A sequential circuit has three *D* flip-flops, *A*, *B*, and *C*, and one input, *X*. It is described by the following flip-flop input functions:

$$DA = (BC' + B'C)x + (Bx + B'C')x', \quad DB = A, \quad DC = B$$

(a) Derive *the* state table for the circuit.

(b) Draw two state diagrams: *one* for *x* = 0 and the other for *x* = 1.

✿ A sequential circuit has two *JK* flip-flops, *A* and *B*; two inputs, *x* and *y*; and one output, *z*.

The flip-flop input functions and the circuit output function are as follows:

$$JA = Bx + B'y', \quad KA = B'xy', \quad JB = A'x, \quad z = Axy + Bx'y'$$

(a) Draw the logic diagram of the circuit.

(b) Tabulate the state table.

(c) Derive the next-state equations for *A* and *B*.

✿ Convert a *D* flip-flop to a *JK* flip-flop by including input gates to the *D* flip-flop. The gates needed for the input of the *D* flip-flop can be determined by means of sequential-circuit design procedures. The sequential circuit to be considered will have one *D* flip-flop and two inputs, *J* and *K*.

✿ Design a sequential circuit with two *D* flip-flops, *A* and *B*, and one input, *x*. When *x* = 0, the state of the circuit remains the same. When *x* = 1, the circuit goes through the state transitions from 00 to 01 to 11 to 10 back to 00, and repeats.

## 11. Part A Q & A (with K level and CO)

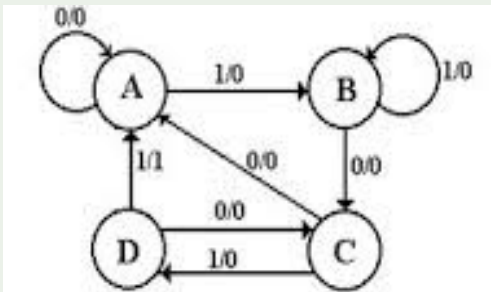
PART A		
Questions and Answers	Blooms Level	COs
<p>1. What is a sequential circuit?</p> <p>A sequential circuit is one in which the output variables dependent not only on the present input variables but they also depend up on the past history of the input variables.</p> <p>.</p>	K1	CO3
<p>2. What are the types of sequential circuits?</p> <p>Synchronous sequential circuit: Change in input signals can affect the memory elements only at discrete instants of time.</p> <p>Asynchronous sequential circuit: Change in input signals can affect memory element at any instant of time.</p>	K1	CO3
<p>3. Define flip flop.</p> <p>Flip flop is a device with two stable states 0 or 1. A flip flop maintains its output state until directed by an input signal to change its state. Since it can store 1-bit of information, it is also called 1-bit memory unit.</p>	K1	CO3
<p>4. What are the types of flip flops?</p> <p>SR Flip flop JK Flip flop Delay (D) Flip flop Toggle (T) Flip flop</p>	K1	CO3
<p>5. What is a latch?</p> <p>A latch is a memory device without clock signal.</p> <p>.</p>	K1	CO3
<p>6. What is a characteristic table?</p> <p>A characteristic table defines the logical property of the flip-flop and completely characteristic its operation.</p>	K1	CO3
<p>7. What is the difference between truth table and excitation table.</p> <p>i) An excitation table is a table that lists the required inputs for a given change of state.</p> <p>ii) A truth table is a table indicating the output of a logic circuit for various input states.</p>	K1	CO3

<p>8. What are the types of triggering a flip flop? The types of triggering a flip flop are, Level triggering Edge triggering.</p>	K1	CO3
<p>9. What is edge triggering in flip flops? Edge triggering means that the flip flop changes state either at the positive edge(rising edge) or at the negative edge (falling edge) of the clock pulse and is sensitive to its inputs only at this transition of the clock.</p>	K1	CO3
<p>10. What is meant by level triggering? In level triggering the output of the flip-flop changes state or responds only when the clock pulse is present.</p>	K1	CO3
<p>11. What is race around condition? In JK flip flop output is fed back to the input and therefore changes in the output results change in the input. Due to this, in the positive half of the clock pulse if J and K are both high then output toggles continuously. This condition is known as race around condition.</p>	K1	CO3
<p>12. What are shift registers? A register is a group of flip flops to store a word. The binary information in a register can be moved from stage to stage upon application of clock pulses. This gives rise to group of registers called shift registers. This type of bit shifting is essential for certain arithmetic and logic operations used in microprocessors.</p>	K1	CO3
<p>13. What is a counter? A counter is a register capable of counting the number of clock pulses arriving at its clock input. Count represents the number of clock pulses arrived.</p>	K1	CO3
<p>14. What is ripple or asynchronous counter? Ripple counter is one in which first flip flop is clocked by the external clock pulse and then each successive flip flop is clocked by the output of previous flip flop.</p>	K1	CO3

15. A counter has 14 stable states 0000 through 1101. If the input frequency is 50KHz what will be its output frequency? Output frequency = $50 / 14 = 3.57$ KHz.	K1	CO3
16. What are the two models of synchronous sequential circuits? The two models of synchronous sequential circuits are, Moore model Melay model	K1	CO3
17. What is a ring counter? A counter formed by circulating a 'bit' in a shift register whose serial output has been connected to its serial input.	K1	CO3
18. What is Johnson counter? It is a ring counter in which the inverted output is fed into the input. It is also known as a twisted ring counter.	K1	CO3
19. What is meant by modulus of a counter? By the term modulus of a counter we say it is the number of states through which a counter can progress.	K1	CO3
20. Give the characteristic equation of a JK and T flip-flop. JK characteristic equation $Q(t+1)=JQ'+K'Q$ T characteristic equation $Q(t+1)=TQ'+T'Q$	K1	CO3



## 12. Part B Qs (with K level and CO)

PART B																								
1. (i) Draw the circuit of SR flip flop and explain the operation (ii) Realize JK flip flop using T flip flop	K3	CO3																						
2. (i) Explain the operation of Master Slave JK flip flop (ii) Explain edge triggering in flip flops	K3	CO3																						
3. Explain the operations of D flip flop and T flip flop, Give their characteristic equation and draw state diagram.	K3	CO3																						
4. A sequential circuit with two D flip flops A and B, two inputs X and Y and one output Z is specified by the following input equations. $TA = X'Y + XA$ , $TB = X'B + XA$ , $Z = XB$ . i. Draw the logic diagram of the circuit ii. Define the state table iii. Derive the state diagram.	K3	CO3																						
5. Explain various types of shift registers with neat diagrams	K3	CO3																						
6. Explain the operation of universal shift register	K3	CO3																						
7. Minimize the state table and design the clocked sequential circuit	K3	CO3																						
<table border="1"> <thead> <tr> <th rowspan="3">Present State</th><th colspan="2">Next State</th></tr> <tr> <th colspan="2">X(Input), Z(Output)</th></tr> <tr> <th>0</th><th>1</th></tr> </thead> <tbody> <tr> <td>A</td><td>B,0</td><td>C,0</td></tr> <tr> <td>B</td><td>B,0</td><td>D,0</td></tr> <tr> <td>C</td><td>E,1</td><td>C,0</td></tr> <tr> <td>D</td><td>E,1</td><td>C,0</td></tr> <tr> <td>E</td><td>B,0</td><td>D,0</td></tr> </tbody> </table>			Present State	Next State		X(Input), Z(Output)		0	1	A	B,0	C,0	B	B,0	D,0	C	E,1	C,0	D	E,1	C,0	E	B,0	D,0
Present State	Next State																							
	X(Input), Z(Output)																							
	0	1																						
A	B,0	C,0																						
B	B,0	D,0																						
C	E,1	C,0																						
D	E,1	C,0																						
E	B,0	D,0																						
8. Design synchronous sequential circuit using JK flip flop for the given state diagram.	K3	CO3																						
																								

9. Design synchronous BCD counter	K3	CO3
10. Design MOD 5 counter using T flip flop	K3	CO3

### 13.Supportive online Certification courses (**NPTEL, Swayam, Coursera, Udemy, etc.,**)

#### ✿ Swayam

Digital Circuits: [https://swayam.gov.in/nd1\\_noc20\\_ee70/preview](https://swayam.gov.in/nd1_noc20_ee70/preview)

Duration: 12 weeks, Start Date: 14 Sep 2020, End Date: 04 Dec 2020

#### ✿ Udemy

Digital Electric Circuits & Intelligent Electrical Devices

<https://www.udemy.com/course/digital-electric-circuits-intelligent-electrical-devices/>

#### ✿ Coursera

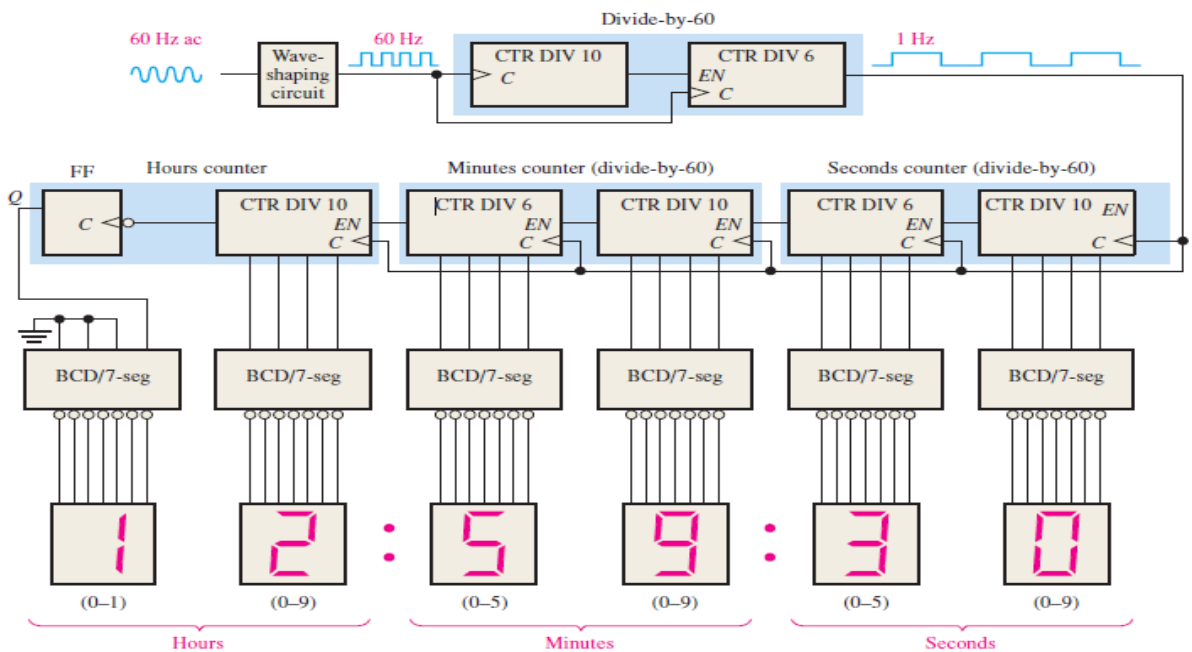
Build a Modern Computer from First Principles: From Nand to Tetris (Project-Centered Course)

<https://www.coursera.org/learn/build-a-computer>

# 14. Real time Applications in day to day life and to Industry

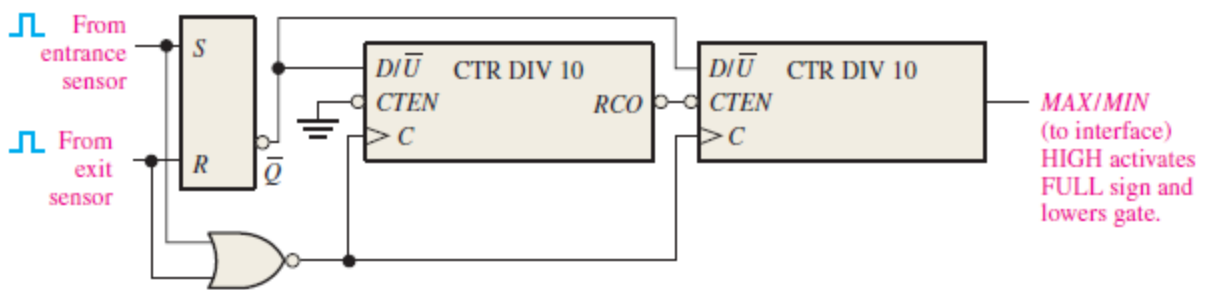
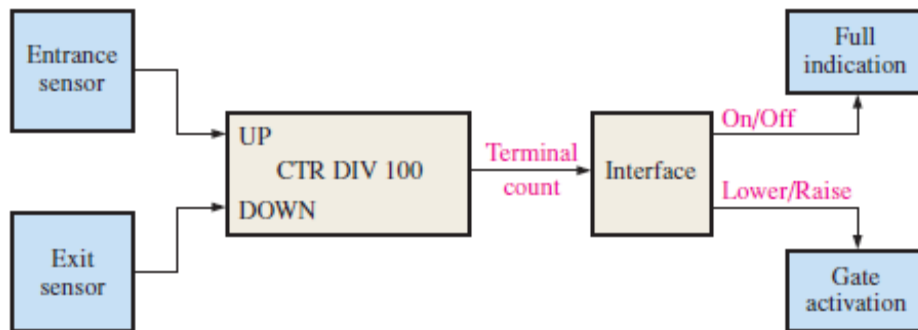
## ❁ DIGITAL CLOCK

- ❁ A digital clock displays seconds, minutes, and hours
- ❁ AC voltage is converted to pulse waveform and divided down to a 1 Hz pulse waveform by divide counters.
- ❁ Both the *seconds* and *minutes* counts are produced by divide counters
- ❁ Synchronous decade counters are used for implementation a truncated sequence achieved by using the decoder count 6 to asynchronously clear the



## ❁ Automobile Parking Control

- ❁ An up/down counter is used as a means of monitoring available spaces in a parking garage and provide indication of a full condition by illuminating a display sign and lowering a gate bar at the entrance



- ❁ Each automobile entering the garage breaks a light beam, activating a sensor that produces an electrical pulse.
- ❁ This positive pulse sets the S-R latch on its leading edge.
- ❁ The LOW on the  $Q$  output of the latch puts the counter in the UP mode. Also, the sensor pulse goes through the NOR gate and clocks the counter on the LOW-to-HIGH transition of its trailing edge.

- ✿ Each time an automobile enters the garage, the counter is advanced by one **(incremented)**.
- ✿ When the one-hundredth automobile enters, the counter goes to its last state (10010).
- ✿ The *MAX/MIN* output goes HIGH and activates the interface circuit (no detail), which lights the FULL sign and lowers the gate bar to prevent further entry.
- ✿ When an automobile exits, an optoelectronic sensor produces a positive pulse, which resets the S-R latch and puts the counter in the DOWN mode. The trailing edge of the clock decreases the count by one **(decremented)**.
- ✿ If the garage is full and an automobile leaves, the *MAX/MIN* output of the counter goes LOW, turning off the FULL sign and raising the gate.

## 15. Contents beyond the Syllabus ( COE related Value added courses)

### ❁ Timers and Counters in Microcontrollers – Contents for Embedded Systems CoE.

#### ❁ **PROGRAMMABLE DIGITAL TIMER SWITCH USING A PIC MICROCONTROLLER**

- ❁ Digital timer switches are used to control the operation of electrical devices based on a programmed schedule.
- ❁ This project describes a programmable digital timer based on the PIC16F628A microcontroller that can be programmed to schedule the on and off operation of an electrical appliance.
- ❁ The appliance is controlled through a relay switch.
- ❁ This timer switch allows you to set both on and off time.

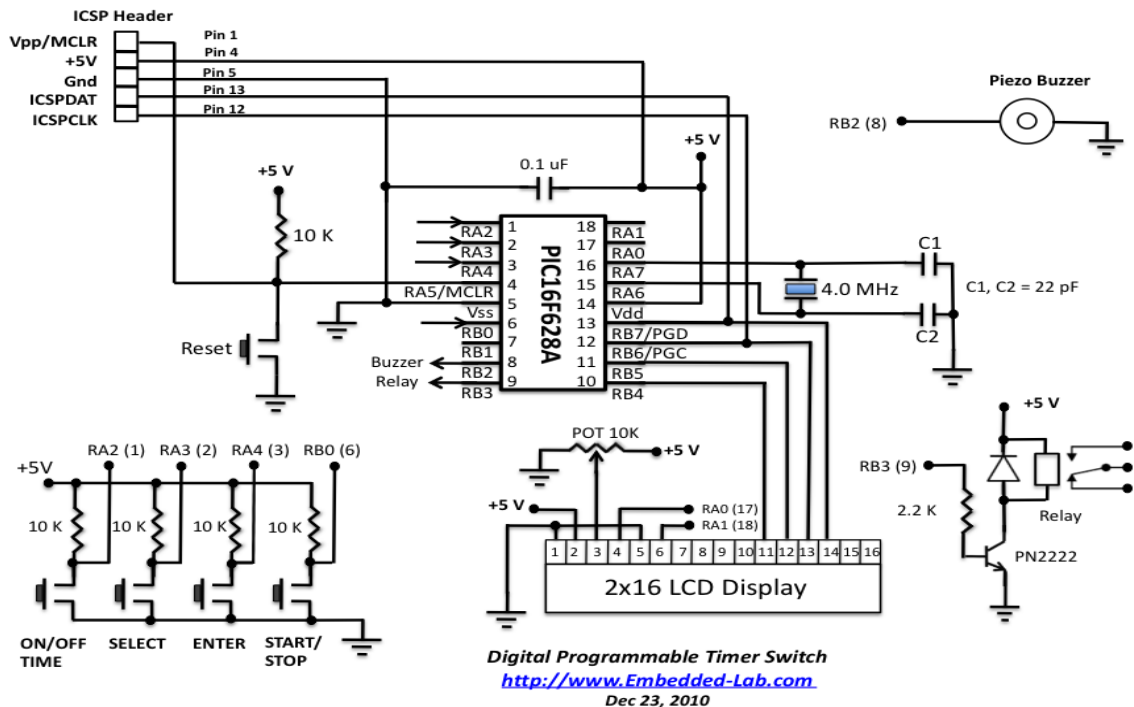
#### ❁ **Operation of the timer**

- ❁ The timer gets inputs from the 4 push buttons. Their functions are described as follows:
- ❁ **ON/OFF TIME** : This timer device allows you to set both on and off time. When the timer is initially powered on, the device is in off condition and both on and off times are 0. Pressing this button, you can switch between the on and off time on the display.
- ❁ **SELECT** : This allows you to select between the on and off time settings as well as hour and minute digits. The selected digit is incremented by pressing the ON/OFF TIME button.
- ❁ **ENTER** : When the appropriate hour and minutes are selected, pressing ENTER finalize the corresponding on or off time.
- ❁ **START/STOP** is to start or stop the timer. If the timer is already on, you can stop it at anytime during its operation by pressing this button.

<http://embedded-lab.com/blog/programmable-digital-timer-switch-using-a-pic-microcontroller/>

### Contents beyond the Syllabus ( COE related Value added courses)

- ✿ The device connected to the relay switch is needed to be turned on after 2 minutes.
- ✿ Further, once it is turned on, it is required to be on for next 20 minutes. In this case, the off time is 00:02 and the on time is 00:20, in hh:mm format.



- ✿ Once the timer is started, the device will be turned on after 2 minutes and remained on for 20 minutes.
- ✿ After that it will be turned off again.
- ✿ A standard 16×2 character LCD is used in the project to display the device status, program menu and time.
- ✿ A piezoelectric buzzer provides audible tone when the timer is started and stopped. It also beeps when the device is turned on or off.



## 17. Prescribed Text Books & Reference Books

### ✿ TEXT BOOKS:

1. James W. Bignel, Digital Electronics, Cengage learning, 5th Edition, 2007.
2. M. Morris Mano, 'Digital Design with an introduction to the VHDL', Pearson Education, 2013.
3. Comer "Digital Logic & State Machine Design, Oxford, 2012.

### ✿ REFERENCES

1. Mandal, "Digital Electronics Principles & Application, McGraw Hill Edu, 2013.
2. William Keitz, Digital Electronics-A Practical Approach with VHDL, Pearson, 2013.
3. Thomas L. Floyd, 'Digital Fundamentals', 11th edition, Pearson Education, 2015.
4. Charles H. Roth, Jr, Lizy Lizy Kurian John, 'Digital System Design using VHDL, Cengage, 2013.

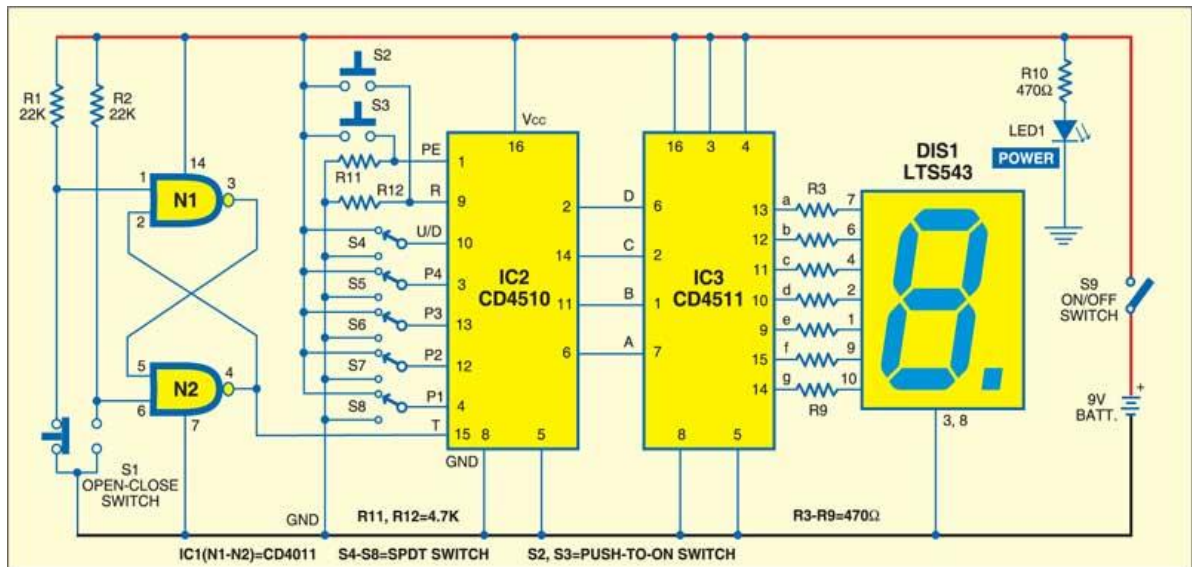
## 18. MINI PROJECT

### ❁ Digital counter circuit

❁ Circuit operation:

❁ Momentarily press of micro-switch S1 once, sends one clock pulse.

❁ The display shows '1.'



❁ Continuous pressing and releasing switch S1 increments the 7-segment display by one digit for each clock pulse.

❁ Continued pressing is done until the display shows '9.'

❁ The next press-and-release operation will change the display to '0,' indicating that the counter has been reset and that it has completed its one cycle.

❁ On changing the position of switch S4 to GND and giving a clock pulse from switch S1, the display will show '9'.

❁ This indicates that the counter has now started counting downwards. Reverse counting is justified as the down-count mode is selected.

❁ For parallel data loading, the parallel data by switches S5 through S8 (either GND or Vcc) is set and then switch S3 is pressed.

❁ The set parallel data appears on the 7-segment display.

<https://www.electronicsforu.com/electronics-projects/digital-counter>

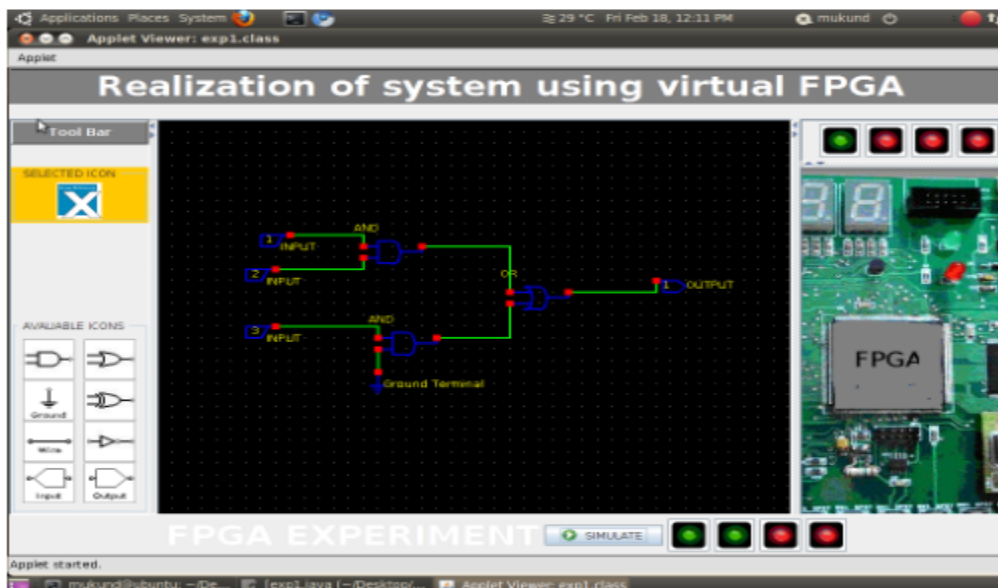
# Digital Logic Circuits

## Unit 4

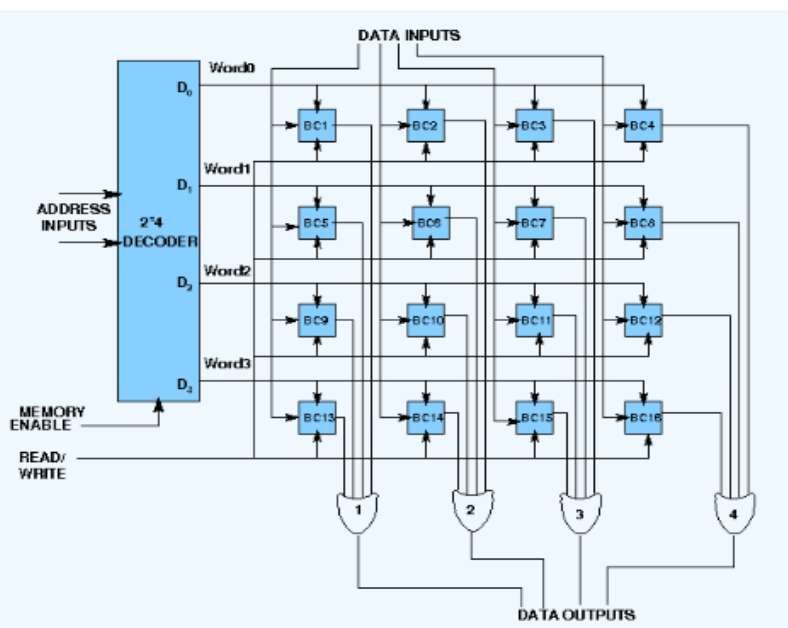
## 8. Activity based learning

- Understanding the working of the PLDs and FPGA through simulation using Virtual Labs

<http://vlab.amrita.edu/?sub=3&brch=66&sim=543&cnt=1>



<https://cse.iitkgp.ac.in/~chitta/coldvl/memory.html>



Practice using open source DEEDS: <https://www.digitalelectronicsdeeds.com/>

## Table of Contents

### ✿ UNIT IV ASYNCHRONOUS SEQUENTIAL CIRCUITS AND PROGRAMMABILITY LOGIC DEVICES

Page No:

**1. Asynchronous sequential logic circuits -----16**

✿ Transition stability, flow stability

**2. Analysis of asynchronous sequential logic circuits -----25**

**3. Race conditions-----35**

**4. Hazards & errors in digital circuits-----43**

**5. Introduction to Programmability Logic Devices-----51**

✿ PROM

✿ PLA

✿ PAL

✿ CPLD

✿ FPGA.

## 9. Lecture Notes

### ✿ E-Books

Digital Fundamentals\_ Global Ed - Thomas L Floyd

Digital Principles And Application - Leach & Malvino

Digital Design - M. Morris Mano and Michael D. Ciletti

Fundamentals of Digital Logic with Verilog Design-Stephen Brown and Zonko Vranesic

### ✿ ONLINE LEARNING MATERIALS:

**1:** [http://nptel.iitm.ac.in/video.php? subject Id=117106086](http://nptel.iitm.ac.in/video.php?subject%20Id=117106086)

**2:** <http://nptel.iitm.ac.in/courses/117101001>

**3:** <https://youtu.be/C-oAyXibnJU>

**4** <https://youtu.be/oYRMYSIVj1o>

**5:** <https://www.youtube.com/watch?v=XZmGGAbHqa0>

**6:** <https://www.youtube.com/watch?v=KymIDyQiXZI>

### ✿ Video Links

<https://drive.google.com/open?id=1qCP5dBvi1LZxd6S7FAUxt788iMmLLpbq>

<https://drive.google.com/open?id=1hrXWfXKWnhidFbnDPDtd75hQt9OzOG6O>

# 1. ASYNCHRONOUS SEQUENTIAL CIRCUITS

## ❁ Asynchronous Sequential Circuits:

### ❁ Introduction:

- ❁ Asynchronous sequential circuits do not use clock signals as synchronous circuits do. Instead, the circuit is driven by the pulses of the inputs which means the state of the circuit changes when the inputs change. Also, they don't use clock pulses. The change of internal state occurs when there is a change in the input variable. Their memory elements are either un-clocked flip-flops or time-delay elements. They are similar to combinational circuits with feedback.

### ❁ Advantages:

- ❁ No clock signal, hence no waiting for a clock pulse to begin processing inputs, therefore fast. Their speed is faster and theoretically limited only by propagation delays of the logic gates.
- ❁ Robust handling. Higher performance function units, which provide average-case completion rather than worst-case completion. Lower power consumption because no transistor transitions when it is not performing a useful computation. Absence of clock drivers reduce power consumption. Less severe electromagnetic interference (EMI).
- ❁ More tolerant to process variations and external voltage fluctuations. Achieve high performance while gracefully handling variable input and output rates and mismatched pipeline stage delays. Freedom from difficulties of distributing a high-fan-out, timing-sensitive clock signal. Better modularity.

- ✿ Less assumptions about the manufacturing process. Circuit speed adapts to changing temperature and voltage conditions. Immunity to transistor-to-transistor variability in the manufacturing process, which is one of the most serious problems faced by the semiconductor industry.

## ✿ **Disadvantages:**

- ✿ Some asynchronous circuits may require extra power for certain operations.
- ✿ More difficult to design and subject to problems like sensitivity to the relative arrival times of inputs at gates. If transitions on two inputs arrive at almost the same time, the circuit can go into the wrong state depending on slight differences in the propagation delays of the gates which is known as race condition.
- ✿ Number of circuit elements (transistors) maybe double that of synchronous circuits. Fewer people are trained in this style compared to synchronous design. Difficult to test and debug. Their output is uncertain.
- ✿ Performance of asynchronous circuits may be reduced in architectures that have a complex data path.

## ✿ **Modes of Asynchronous Sequential Machines:**

- ✿ Depending on the type of input variables, the way they are allowed to change etc., the asynchronous sequential circuits are classified into two categories:
  - Fundamental Mode
  - Pulse Mode



## ❁ Fundamental Mode Asynchronous Circuits:

- ❁ The fundamental mode asynchronous circuit design is based on the following assumptions:
  - The inputs (I) to the synchronous circuits change only when the circuit is stable, that means when the state variables (S) are not in their transition state.
  - Another assumption is that the inputs are levels and not pulses.
  - The state variables in these circuits are characterized as delay elements. Delay may be introduced by a latch or simply the propagation delays inherent in the logic gates used for realizing the asynchronous circuits.

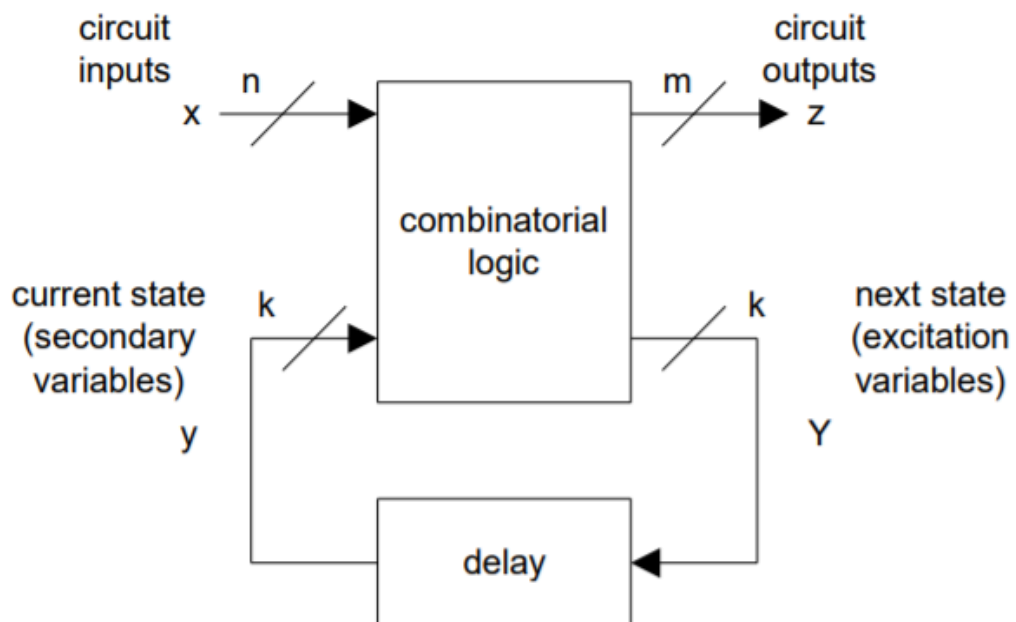
## ❁ Pulse Mode Asynchronous Circuits:

- ❁ In the pulsed mode, the input variables are allowed to be applied in the form of pulses, rather than in the form of levels.
- ❁ But the width of input pulses is a critical parameter.
- ❁ There are two restrictions on the width of the input pulses.
  - The first requirement is that the pulses should be long enough so that the circuit can respond to them.
  - The second requirement is that the pulses should not be too long so that they are still present after the new secondary state is reached.
- ❁ The second requirement is that the pulses should not be too long so that they are still present after the new secondary state is reached.
- ❁ The base of calculating the minimum pulse width is the propagation delay of the excitation logic.

- ✿ The maximum pulse width is calculated based on the total propagation delay through the excitation logic and the memory.

## ✿ Block Diagram:

- ✿ Type of circuit without clocks, but with the concept of memory.
- ✿ Concept of memory is obtained via un-clocked latches and/or circuit delay.
- ✿ Changes in input variables cause changes in states.
- ✿ Asynchronous sequential circuits resemble combinatorial circuits with feedback paths.
- ✿ The design of synchronous circuits is more difficult than synchronous circuits using flip-flops and clocks.



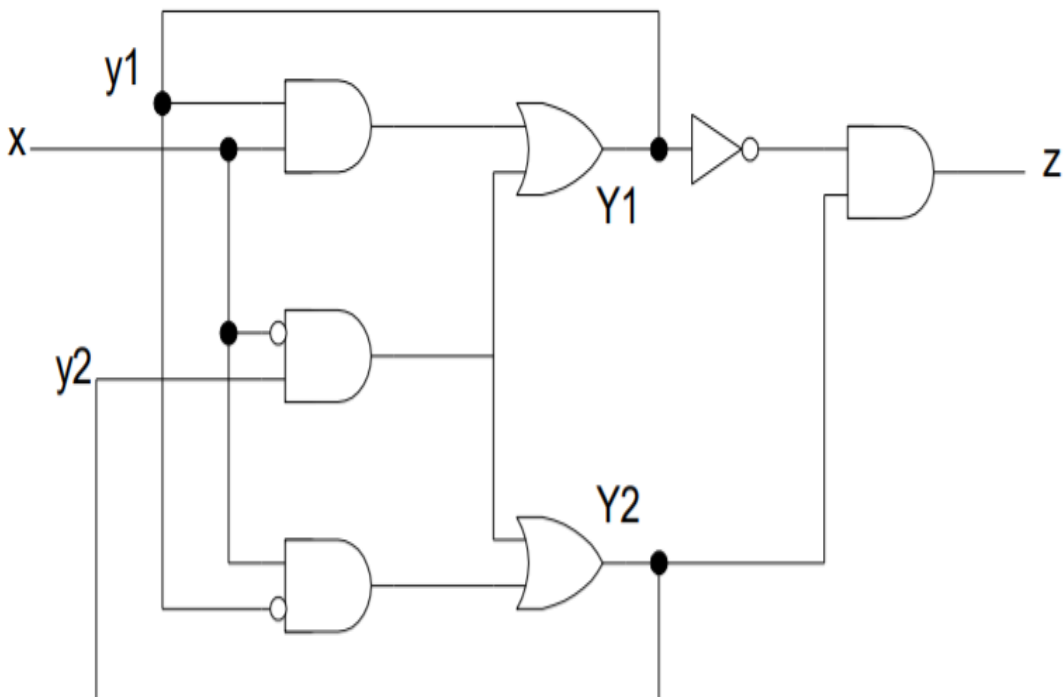
- ⚙ Note difference in “little  $y$ ” secondary variables and “capital  $Y$ ” excitation variables.
- ⚙ Delay elements are hypothetical, and typically are a result of gate delays.
- ⚙ Note: When inputs change, excitation variables  $Y$  change. It takes additional delay for the secondary variables (current state) to assume the values of the excitation variables (next state).
- ⚙ Asynchronous circuits are identified by:
  - The presence of combinatorial feedback paths, and/or
  - The presence of un-clocked storage elements (i.e., latches).

⚙ **Stability:**

- For a given set of inputs (i.e., values), the system is STABLE if the circuit eventually reaches steady state and the excitation variables and secondary variables are equal and unchanging (little  $y$  = capital  $Y$ ), otherwise the circuit is UNSTABLE.

## ✿ Transition Table:

- ✿ Transition table shows excitation variables and outputs as a function of inputs and secondary variables.
- ✿ Example.: For the circuit, transition table is obtained as follows:



- Write logic equations for the excitation variables in terms of the circuit inputs and secondary variables:

$$Y_1 = xy_1 + \bar{x}y_2$$

$$Y_2 = \bar{x}y_2 + x\bar{y}_1$$

- Write logic equations for circuit outputs in terms of the circuit inputs and secondary variables:

$$z = \bar{y}_1 y_2$$

- Using these equations, we can write a transition table that shows excitation variables and outputs as a function of inputs and secondary variables:

curr state  y <sub>2</sub> y <sub>1</sub>	next state		output	
	x=0 Y <sub>2</sub> Y <sub>1</sub>	x=1 Y <sub>2</sub> Y <sub>1</sub>	x=0 z	x=1 z
00	00	10	0	0
01	00	01	0	0
10	11	10	1	1
11	11	01	0	0

- Note that stable states (secondary variables equal to excitation variables) are circled.

## ❁ Flow Table:

❁ Flow Table is just the transition table with binary numbers replaced with symbols (e.g., let  $a = 00$ ,  $b = 01$ ,  $c = 10$  and  $d = 11$ ).

❁ Example.: For the previous circuit, flow table is obtained as follows:

curr state y2y1	next state		output	
	x=0 Y2Y1	x=1 Y2Y1	x=0 z	x=1 z
00	00	10	0	0
01	00	01	0	0
10	11	10	1	1
11	11	01	0	0

curr state y2y1	next state		output	
	x=0 Y2Y1	x=1 Y2Y1	x=0 z	x=1 z
a	a	c	0	0
b	a	b	0	0
c	d	c	1	1
d	d	b	0	0

❁ Another way to draw a flow table:

	x=0	x=1
a	a , 0	c , 0
b	a , 0	b , 0
c	d , 1	c , 1
d	d , 0	b , 0

- Left-most column shows current state (secondary variables), and the inputs are listed across the top.
- Entries in the matrix show the next state (excitation variables) and output values.

### ✿ Primitive Flow Table:

✿ Flow table with only one stable state per row is called a primitive flow table.

✿ Example: A Flow Table which is a Primitive Flow Table

	X	
	0	1
a	a	b
b	c	b
c	c	d
d	a	d

✿ Example: A Flow Table which is not a Primitive Flow Table

	x1x2			
	00	01	11	10
a	a	a	a	b
b	a	a	b	b

## 2. Analysis of Asynchronous Sequential Logic Circuits:

⚙ Procedure to determine transition table and/or flow table from a circuit with combinational feedback paths:

- Determine feedback paths.
- Label  $Y$  (excitation variables) at output and  $y$  (secondary variables at input).
- Derive logic expressions for  $Y$  (excitation variables) in terms of circuit inputs and secondary variables. Do the same for circuit outputs.
- Create a transition table and flow table.
- Circle stable states where  $Y$  (excitation variables) are equal to  $y$  (secondary variables).

## ⚙ Design of Asynchronous Sequential Logic Circuits:

⚙ Similar procedure to synchronous circuit design, but with some added complexities due to the asynchronous part:

- Obtain a primitive flow table (one stable state per row) from problem description.
- Reduce the flow table to get a smaller flow table with less states.
- Perform state assignment (need to avoid race conditions) to obtain a transition table.
- Obtain next state and output equations (need to avoid hazards and glitches).
- Draw circuit (with or without latches).



### ✿ Example Problem:

✿ Design a circuit with two inputs, D and G and one output, Q. Output Q follows D with  $G=1$ , otherwise Q holds its value.

✿ Assume fundamental mode operation – only one input changes at a time.

state	Inputs D      G		Output Q	Behavior : trnsfer D to o/p if G is 1; retain D value if $G \rightarrow 0$
a	0	1	0	Transfer D to Q
b	1	1	1	Transfer D to Q
c	0	0	0	Keep previous Q=0; after a or d
d	1	0	0	Keep previous Q=0; after c
e	1	0	1	Keep previous Q=1; after b or f
f	0	0	1	Keep previous Q=1; after e

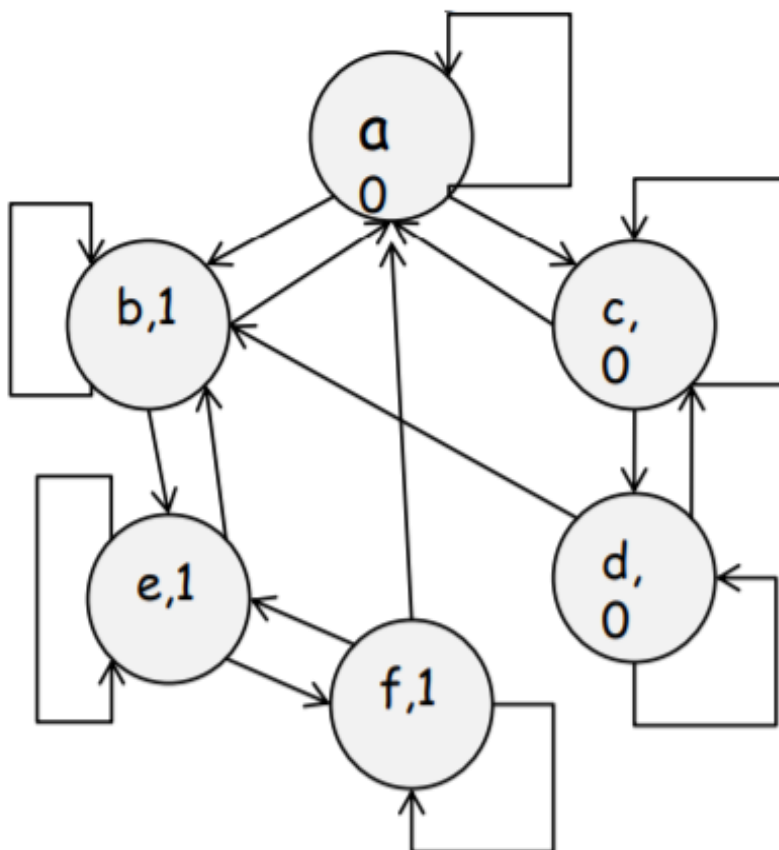
### ✿ Primitive Flow Table:

curr state	next state				output Q
	DG=00	DG=01	DG=10	DG=11	
a	c	a	-	b	0
b	-	a	e	b	1
c	c	a	d	-	0
d	c	-	d	b	0
e	f	-	e	b	1
f	f	a	e	-	1

✿ Note: Some unspecified entries due to the fundamental mode assumption (e.g., in state a,  $DG=01$ , so we never go from  $DG=01 \rightarrow DG=10$ )...

## ❁ Reduced Flow Table:

- ❁ For the moment, assume that the following flow table will also work for the verbal problem description – assume (a,c,d) and (b,e,f) can be merged.



curr state	next state				output Q
	DG=00	DG=01	DG=10	DG=11	
a	a	a	a	b	0
b	b	a	b	b	1

### ❁ State Assignment and Transition Table:

❁ We only have two states, so we can let  $a=0$ , and  $b=1$ .

❁ Our transition table becomes:

curr state (y)	next state (Y)				output Q
	DG=00	DG=01	DG=10	DG=11	
0	0	0	0	1	0
1	1	0	1	1	1

### ❁ Logic Equations:

❁ We can make K-Maps to determine excitation variables (Y) and output (Z) in terms of circuit inputs and secondary variables (y):

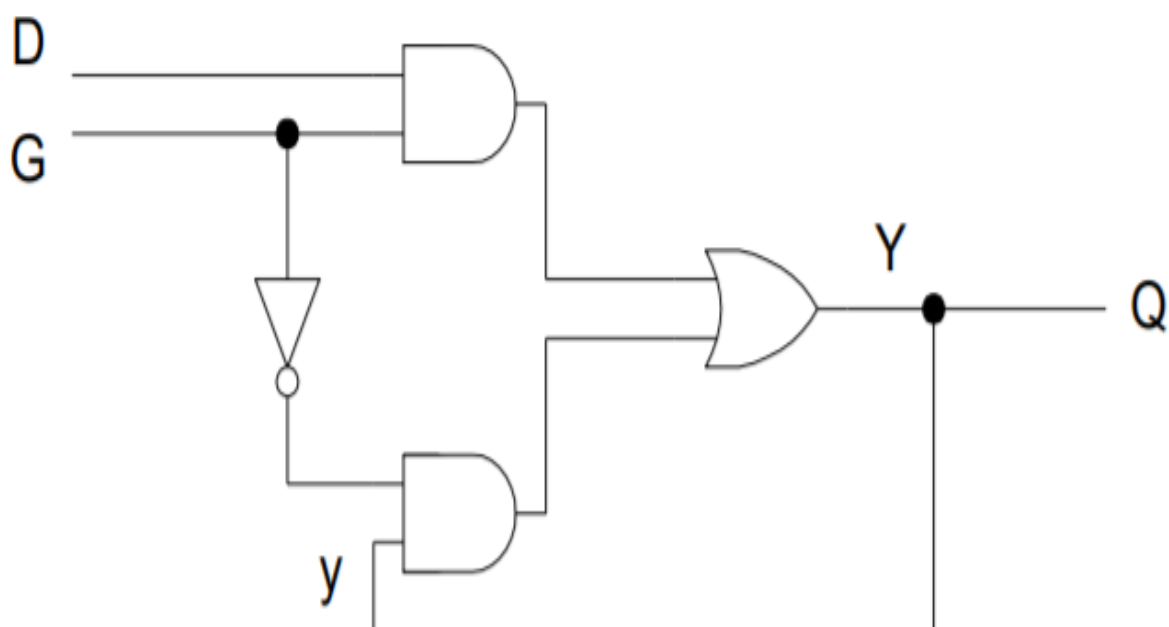
DG

y		00	01	11	10
0		0	0	1	0
1		1	0	1	1

$$Y = DG + G'y$$

❁ Output equal to the secondary (state) variable.

❁ Logic Circuit:







## ❁ Output Assignment:

- ❁ Flow and transition tables might have unspecified entries for circuit outputs.
  - This might be a result of the fundamental mode assumption.
  - This might be a result of unstable states.
- ❁ Note: output values always assigned for stable states!
- ❁ We should think about the correctness of these unspecified don't care output values...
  - We might temporarily pass through these values while transitioning from one stable state to another stable state.
- ❁ Example: Consider the following flow table with don't cares at some outputs (circuit has one input and one output):

curr state	next state		output	
	x=0	x=1	x=0	x=1
a	(a)	b	0	-
b	c	(b)	-	0
c	(c)	d	1	-
d	a	(d)	-	1

- We might consider using the un-specified output values as don't cares in order to minimize the logic function for the output...
- We need to be careful with output don't cares in asynchronous design.
- Consider start and stop STABLE STATES due to a change in input value.
  - If both stable states produce a 0 output, make output 0 instead of a don't care.
  - If both stable states produce a 1 output, make output 1 instead of a don't care.
  - If stable states produce different outputs, the output can remain a don't care and be used to find a smaller output circuit.
- We do this to avoid GLITCHES in the output (e.g., if the output should go 0->0 (or 1->1), it should remain 0 (or 1) during the transition through an unstable state.
- Recall the flow table... If we consider possible transitions, we see that some of the output don't cares should be changed to 0 or 1 to avoid GLITCHES.

curr state	next state		output	
	x=0	x=1	x=0	x=1
a	(a)	b	0	-
b	c	(b)	-	0
c	(c)	d	1	-
d	a	(d)	-	1

curr state	next state		output	
	x=0	x=1	x=0	x=1
a	(a) 	b	0	0
b	c	(b) 	-	0
c	(c) 	d	1	1
d	a	(d) 	-	1

- The above changes will avoid temporary glitches at the outputs during transitions where the output should not change.

Analysis of asynchronous synchronous sequential circuit:

1. An asynchronous sequential circuit is described by the following excitation and output function.

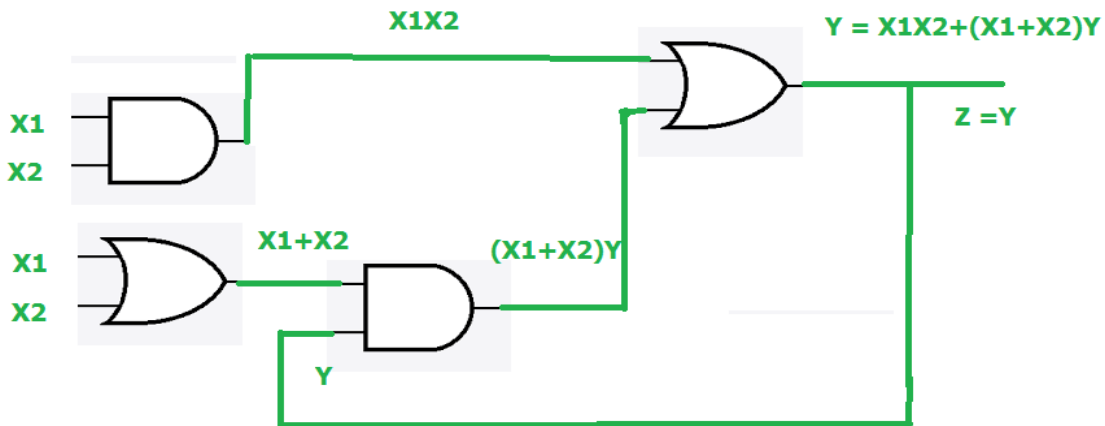
$$Y = X_1X_2 + (X_1 + X_2)Y$$

$$Z = y$$

- i) Draw the logic diagram of the circuit.
- ii) Derive the transition table and output map
- iii) Describe the behavior of the circuit.

Solution:

- i) Logic diagram of the circuit



- ii) Transition Table:

Transition table is useful to analyze an asynchronous circuit from the circuit diagram  
Procedure to obtain transition table:

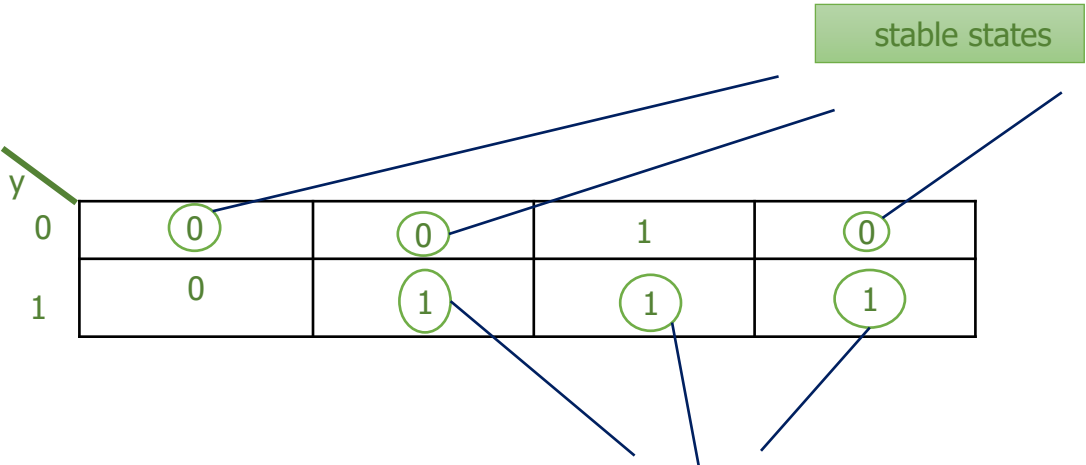
1. Determine all feedback loops in the circuits.
2. Mark the input ( $y_i$ ) and output ( $Y_i$ ) of each feedback loop.
3. Derive the Boolean functions of all  $Y$ 's.
4. Plot each  $Y$  function in a map and combine all maps into one table.
5. Circle those values of  $Y$  in each square that are equal to the value of  $y$  in the same row.

To derive the transition table, First we have to derive state table.

State table of the circuit:

State table:

External Inputs		Present State	Next State	Output
X1	X2	y	Y	Z
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1



Transition table:

X1X2	00	01	11	10
------	----	----	----	----



iii) Output map (Z)

		y	
		0	1
X1X2	00	0	1
	01	0	1
	11	0	1
	10	0	1

iv) Behavior of the circuit:

- The behavior of the circuit depends only on the present state.
- The output of the circuit is high only when the present state is high.

Analysis of asynchronous synchronous sequential circuit:

1. An asynchronous sequential circuit is described by the following excitation and output function.

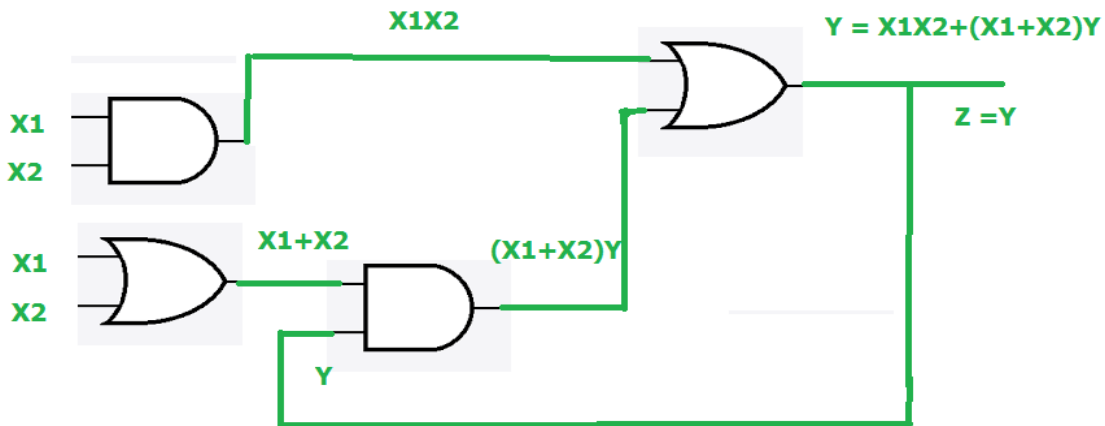
$$Y = X_1X_2 + (X_1 + X_2)Y$$

$$Z = y$$

- i) Draw the logic diagram of the circuit.
- ii) Derive the transition table and output map
- iii) Describe the behavior of the circuit.

Solution:

- i) Logic diagram of the circuit



- ii) Transition Table:

Transition table is useful to analyze an asynchronous circuit from the circuit diagram  
Procedure to obtain transition table:

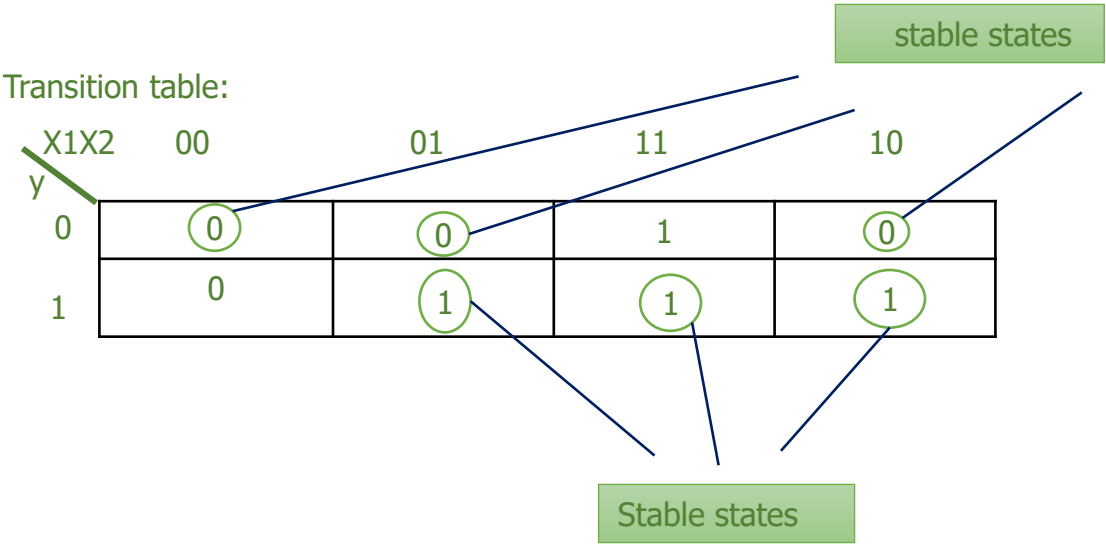
1. Determine all feedback loops in the circuits.
2. Mark the input ( $y_i$ ) and output ( $Y_i$ ) of each feedback loop.
3. Derive the Boolean functions of all  $Y$ 's.
4. Plot each  $Y$  function in a map and combine all maps into one table.
5. Circle those values of  $Y$  in each square that are equal to the value of  $y$  in the same row.

To derive the transition table, First we have to derive state table.

State table of the circuit:

State table:

External Inputs		Present State	Next State	Output
X1	X2	y	Y	Z
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1



iii) Output map (Z)

y x1x2		0	1
		0	1
00	0	1	
01	0	1	
11	0	1	
10	0	1	

iv) Behavior of the circuit:

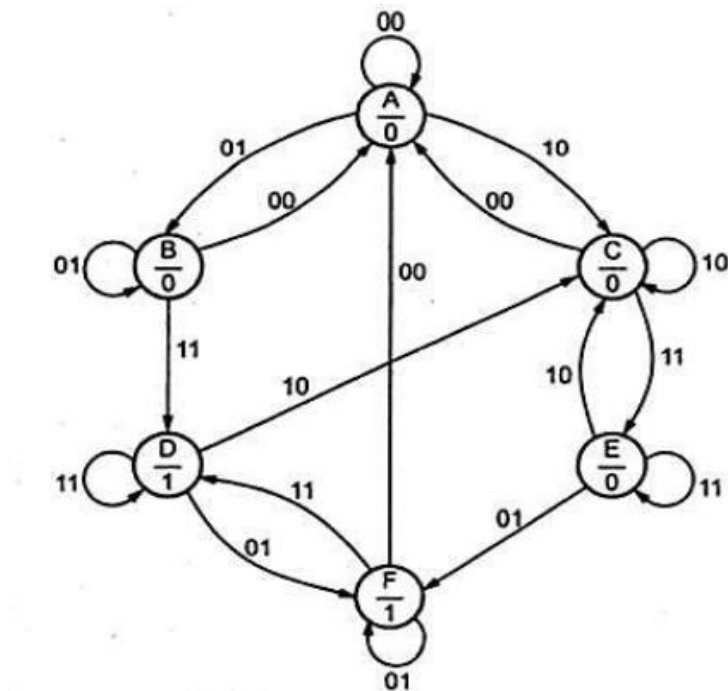
- The behavior of the circuit depends only on the present state.
- The output of the circuit is high only when the present state is high.

# Design of asynchronous sequential circuit

1. Design an asynchronous sequential circuit that has two inputs  $X_2$  and  $X_1$  and one output  $Z$ . When  $X_1 = 0$ , output  $Z = 0$ . The first change in  $X_2$  that occurs while  $X_1$  is 1 will cause output  $Z$  to be 1. The output  $Z$  will remain at 1 till  $X_1$  returns to 0.

**Solution:**

Step 1: Draw the state diagram



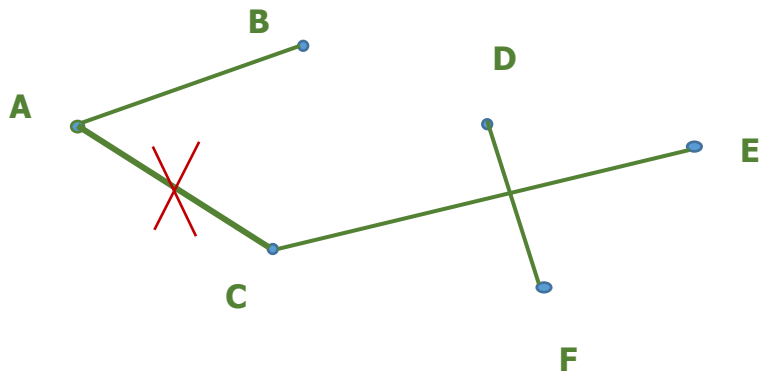
Step2: Primitive flow table constructed from state diagram

Present State	Next state, Output for $X_2 X_1$ inputs			
	00	01	11	10
A	<u>A</u> , 0	B, —	—, —	C, —
B	A, —	<u>B</u> , 0	D, —	—, —
C	A, —	—, —	E, —	<u>C</u> , 0
D	—, —	F, —	<u>D</u> , 1	C, —
E	—, —	F, —	<u>E</u> , 0	C, —
F	A, —	<u>F</u> , 1	D, —	—, —

### Step 3: Implication table

B	✓				
C	✓	X			
D	X	X	X		
E	X	X	✓	X	
F	X	X	X	✓	X
	A	B	C	D	E

### Step 4 : Merger Diagram



Equal states are (A,B), (C,E), (D,F)

### Step 5: State assignment

(A, B)  $\rightarrow$   $S_0$     (C,E)  $\rightarrow$   $S_1$     (D,F)  $\rightarrow$   $S_2$

State assignment: Assigning binary value to the states

$S_0 \rightarrow 00$ ,  $S_1 \rightarrow 01$ ,  $S_2 \rightarrow 10$

### Step 6: Reduced flow table with state assignment

Present State F2 F1	Next state, Output for $X_2 X_1$ inputs			
	00	01	11	10
$S_0$ 00	$S_0$ , 1	$S_0$ , 1	$S_2$ , —	$S_1$ , —
$S_1$ 01	$S_0$ , —	$S_2$ , —	$S_1$ , 0	$S_1$ , 0
$S_2$ 10	$S_0$ , —	$S_2$ , 1	$S_2$ , 1	$S_1$ , —

Note :  $S_1$  to  $S_2$ ,  $S_2$  to  $S_1$  – transition is not possible.  
(Because critical race) So it requires temporary  $S_3$

So, the reduced flow table becomes,

### Step 7: Reduced flow table with race free state assignment

Present State F2 F1	Next state, Output for $X_2 X_1$ inputs			
	00	01	11	10
$S_0$ 00	$S_0$ , 1	$S_0$ , 1	$S_2$ , —	$S_1$ , —
$S_1$ 01	$S_0$ , —	$S_3$ , —	$S_1$ , 0	$S_1$ , 0
$S_2$ 10	$S_0$ , —	$S_2$ , 1	$S_2$ , 1	$S_3$ , —
$S_3$ 11	—, —	$S_2$ , —	—, —	$S_1$ , —

Step 8: Transition table

Present State  F <sub>2</sub> F <sub>1</sub>	Next state, Output for X <sub>2</sub> X <sub>1</sub> inputs			
	00	01	11	10
00	00, 1	00, 1	10, —	01, —
01	00, —	11, —	01, 0	01, 0
10	00, —	10, 1	10, 1	11, —
11	—, —	10, —	—, —	01, —

Step 9 : K- map Simplification

For  $F_2^+$

$F_2 F_1 \backslash X_2 X_1$	00	01	11	10
00	0	0	1	0
01	0	1	0	0
11	X	1	X	0
10	0	1	1	1

For  $F_1^+$

$F_2 F_1 \backslash X_2 X_1$	00	01	11	10
00	0	0	0	1
01	0	1	1	1
11	X	0	X	1
10	0	0	0	1

$$F_2^+ = \overline{F_1} X_2 X_1 + \overline{F_1} \overline{X_2} X_1 + F_2 X_1 + F_2 \overline{F_1} X_2$$

$$F_1^+ = \overline{F_2} F_1 X_1 + X_2 \overline{X_1}$$



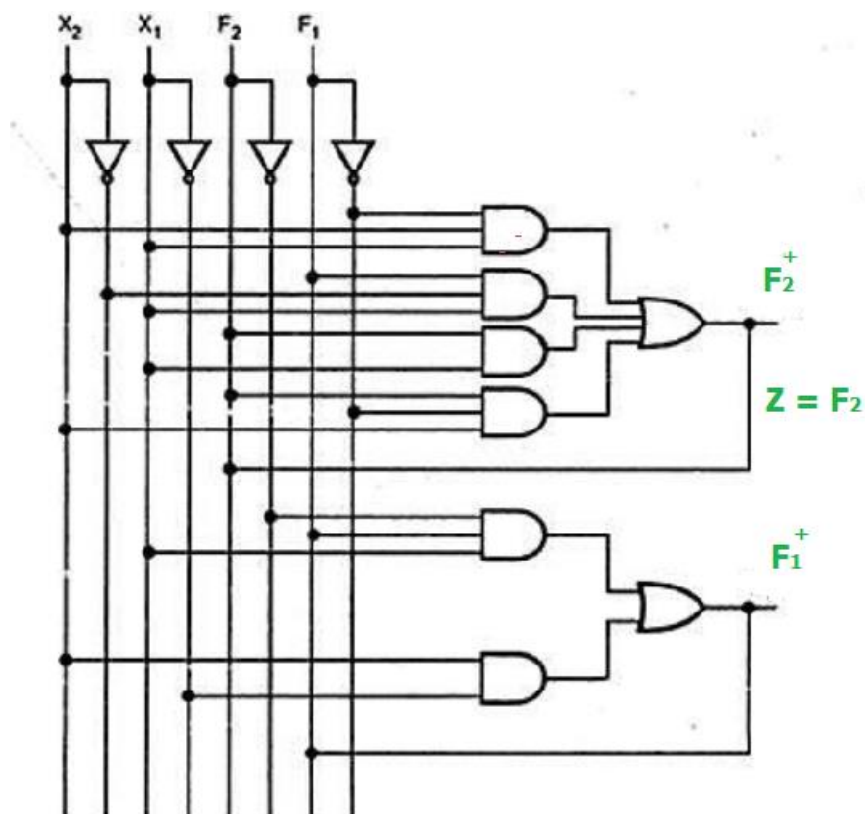
K-map for output Z:

**For Z**

$F_2F_1 \backslash X_2X_1$	00	01	11	10
00	0	0	X	X
01	X	X	0	0
11	X	X	X	X
10	X	1	1	X

$$Z = F_2$$

Step 10: Logic diagram



### 3. Race Conditions:

- ✿ A race condition exists in an asynchronous circuit when two or more binary state variables change value in response to a change in an input variable, when unequal delays are encountered a race condition may cause the state variable to change in an unpredictable manner.
- ✿ A race condition or race hazard is the behaviour of an electronic, software, or other system where the output is dependent on the sequence or timing of other uncontrollable events.
- ✿ A race condition may occur in a system of logic gates where inputs vary.
- ✿ If a given output depends on the state of the inputs it may only be defined for steady-state signals.
- ✿ As the inputs change state a small delay will occur before the output changes due to the physical nature of the electronic system.
- ✿ The output may, for a brief period, change to an unwanted state before settling back to the designed state.
- ✿ Certain systems can tolerate such glitches but if this output functions as a clock signal for further systems that contain memory, for example, the system can rapidly depart from its designed behaviour.
- ✿ Types:
  - Critical and non-critical forms
    - A critical race condition occurs when the order in which internal variables are changed determines the eventual state that the state machine will end up in. Race condition: Two or more binary state variables will change value when one input variable changes. Cannot predict state sequence if unequal delay is encountered.

	$x$	
	0	1
$y_1 y_2$		
00	00	11
01		01
11		11
10		10

(a) Possible transitions:

$00 \rightarrow 11$   
 $00 \rightarrow 01$   
 $00 \rightarrow 10$

	$x$	
	0	1
$y_1 y_2$		
00	00	11
01		11
11		11
10		10

(b) Possible transitions:

$00 \rightarrow 11$   
 $00 \rightarrow 01 \rightarrow 11$   
 $00 \rightarrow 10$

- A non-critical race condition occurs when the order in which internal variables are changed does not determine the eventual state that the state machine will end up in. The final stable state does not depend on the change order of state variables.

	$x$	
	0	1
$y_1 y_2$		
00	00	11
01		11
11		11
10		11

(a) Possible transitions:

$00 \rightarrow 11$   
 $00 \rightarrow 01 \rightarrow 11$   
 $00 \rightarrow 10 \rightarrow 11$

	$x$	
	0	1
$y_1 y_2$		
00	00	11
01		01
11		01
10		11

(b) Possible transitions:

$00 \rightarrow 11 \rightarrow 01$   
 $00 \rightarrow 01$   
 $00 \rightarrow 10 \rightarrow 11 \rightarrow 01$

	$x$	
	0	1
$y_1 y_2$		
00	00	11
01		01
11		11
10		10

(a) Possible transitions:

$00 \rightarrow 11$   
 $00 \rightarrow 01$   
 $00 \rightarrow 10$

	$x$	
	0	1
$y_1 y_2$		
00	00	11
01		11
11		11
10		10

(b) Possible transitions:

$00 \rightarrow 11$   
 $00 \rightarrow 01 \rightarrow 11$   
 $00 \rightarrow 10$

- A non-critical race condition occurs when the order in which internal variables are changed does not determine the eventual state that the state machine will end up in. The final stable state does not depend on the change order of state variables.

	$x$	
	0	1
$y_1 y_2$		
00	00	11
01		11
11		11
10		11

(a) Possible transitions:

$00 \rightarrow 11$   
 $00 \rightarrow 01 \rightarrow 11$   
 $00 \rightarrow 10 \rightarrow 11$

	$x$	
	0	1
$y_1 y_2$		
00	00	11
01		01
11		01
10		11

(b) Possible transitions:

$00 \rightarrow 11 \rightarrow 01$   
 $00 \rightarrow 01$   
 $00 \rightarrow 10 \rightarrow 11 \rightarrow 01$

- Design techniques such as Karnaugh maps encourage designers to recognize and eliminate race conditions before they cause problems. Often logic redundancy can be added to eliminate some kinds of races.

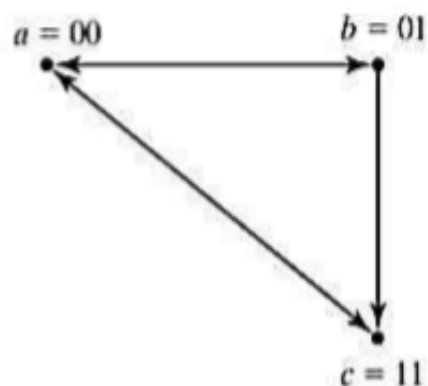
### ❁ Race -Free State Assignment:

- ❁ Once a reduced flow table has been derived for an asynchronous sequential circuit, the next step in the design is to assign binary variables to each stable state. This assignment results in the transformation of the flow table into its equivalent transition table.
- ❁ The primary objective in choosing a proper binary state assignment is the prevention of critical races. Critical races can be avoided by making a binary state assignment in such a way that only one variable changes at any given time when a state transition occurs in the flow table.

### ❁ Three-Row Flow-Table Example:

	$x_1 x_2$			
	00	01	11	10
$a$	$a$	$b$	$c$	$a$
$b$	$a$	$b$	$b$	$c$
$c$	$a$	$c$	$c$	$c$

(a) Flow table

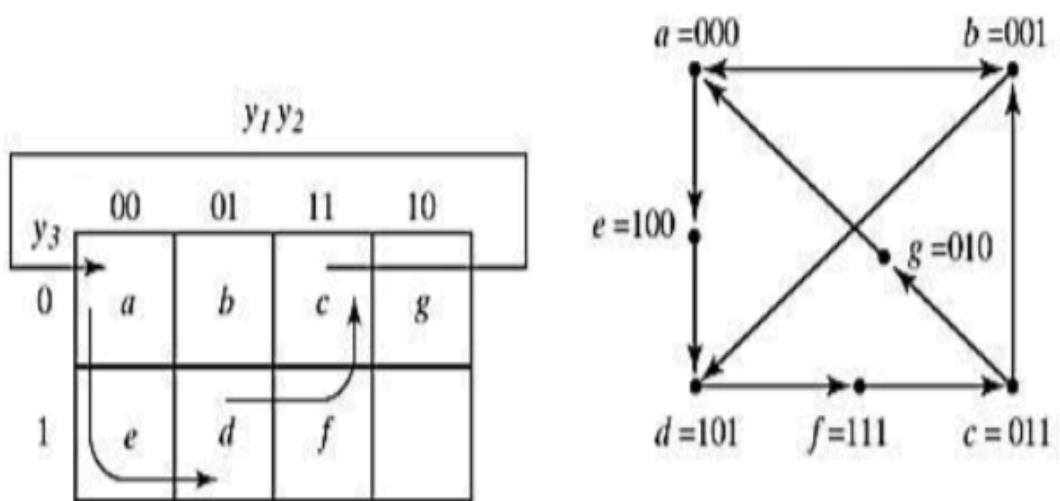


(b) Transition diagram

- To avoid critical races, we must find a binary state assignment such that only one binary variable change during each state transition.
- An attempt to find such an assignment is shown in the transition diagram.
- State a is assigned binary 00, and state c is assigned binary 11.
- This assignment will cause a critical race during the transition from a to c because there are two changes in the binary state variables and the transition from a to c may occur directly or pass through b.
- Note that the transition from c to a also causes a race condition, but it is noncritical because the transition does not pass through other states.
- A race-free assignment can be obtained if we add an extra row to the flow table. The use of a fourth row does not increase the number of binary state variables, but it allows the formation of cycles between two stable states.
- The transition table corresponding to the flow table with the indicated binary state assignment is shown. The two dashes in row d represent unspecified states that can be considered don't care conditions. However, care must be taken not to assign 10 to these squares, in order to avoid the possibility of an unwanted stable state being established in the fourth row.

### ❁ Four-Row Flow-Table Example:

- A flow table with four rows requires a minimum of two state variables.
- Although a race-free assignment is sometimes possible with only two binary state variables, in many cases the requirement of extra rows to avoid critical races will dictate the use of three binary state variables.



- The above figure shows a state assignment map that is suitable for any four-row flow table.
- States a, b, c and d are the original states and e, f and g are extra states.
- The transition from a to d must be directed through the extra state e to produce a cycle so that only one binary variable changes at a time.
- Similarly, the transition from c to a is directed through g and the transition from d to c goes through f.
- By using the assignment given by the map, the four-row table can be expanded to a seven-row table that is free of critical races.

	00	01	11	10
000 = $a$	$b$	$a$	$e$	$a$
001 = $b$	$b$	$d$	$b$	$a$
011 = $c$	$c$	$g$	$b$	$c$
010 = $g$	-	$a$	-	-
110 -	-	-	-	-
111 = $f$	$c$	-	-	$c$
101 = $d$	$f$	$d$	$d$	$f$
100 = $e$	-	-	$d$	-

- Note that although the flow table has seven rows there are only four stable states. The uncircled states in the three extra rows are there merely to provide a race-free transition between the stable states.



## ❁ Cycle:

- A cycle occurs when an asynchronous circuit makes a transition through a series of unstable states. If a cycle does not contain a stable state, the circuit will go from one unstable to stable to another, until the inputs are changed.
- When a state assignment is made so that it introduces cycles.
- Care must be taken so that each cycle terminates on a stable state.
- If a cycle does not contain a stable state, the circuit will go from one unstable state to another, until the inputs are changed.

$y_1y_2 \backslash x$	0	1
00	00	01
01		11
11		10
10		10

(a) State transition:  
00 → 01 → 11 → 10

$y_1y_2 \backslash x$	0	1
00	00	01
01		11
11		11
10		10

(b) State transition:  
00 → 01 → 11

$y_1y_2 \backslash x$	0	1
00	00	01
01		11
11		10
10		01

(c) Unstable  
01 → 11 → 10 → 01

## 4. Hazards & Errors in Digital Circuits:

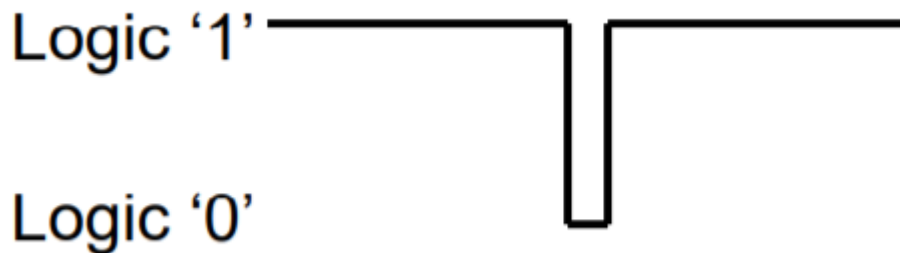
### ❁ Introduction:

1. A hazard, if exists, in a digital circuit causes a temporary fluctuation in output of the circuit. In other words, a hazard in a digital circuit is a temporary disturbance in ideal operation of the circuit which if given some time, gets resolved itself.
2. These disturbances or fluctuations occur when different paths from the input to output have different delays and due to this fact, changes in input variables do not change the output instantly but do appear at output after a small delay caused by the circuit building elements, i.e., logic gates.
3. A hazard is a momentary unwanted switching transient at a logic function output (i.e., a glitch).
4. Hazards/glitches occur due to unequal propagation delays along different paths in a combinational circuit.
5. Hazards in combinational circuits:
  - a. Static Hazard: Static 0 Hazard & Static 1 Hazard.
  - b. Dynamic Hazard
6. Hazards in sequential circuits:
  - a. Essential Hazard

## ❁ Static Hazard:

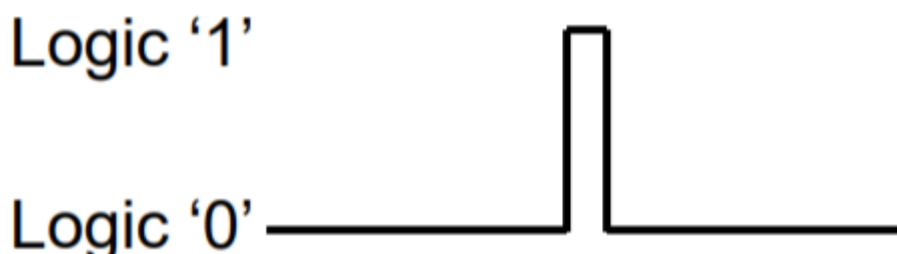
❁ A Static Hazard takes place when change in an input causes the output to change momentarily before stabilizing to its correct value. Based on what is the correct value, there are two types of static hazards:

1. Static-1 Hazard: If the output is currently at logic state 1 and after the input changes its state, the output momentarily changes to 0 before settling on 1, then it is a Static-1 Hazard.



❁ When the output is to remain at the value 1, but a momentary 0 output is produced during the transition between two input states, then the hazard is Static-1 Hazard.

2. Static-0 Hazard: If the output is currently at logic state 0 and after the input changes its state, the output momentarily changes to 1 before settling on 0, then it is a Static-0 Hazard.



- ✿ When the output is to remain at the value 0, but a momentary 1 output is produced during the transition between two input states, then the hazard is Static-0 hazard.

### ✿ Static Hazards Elimination:

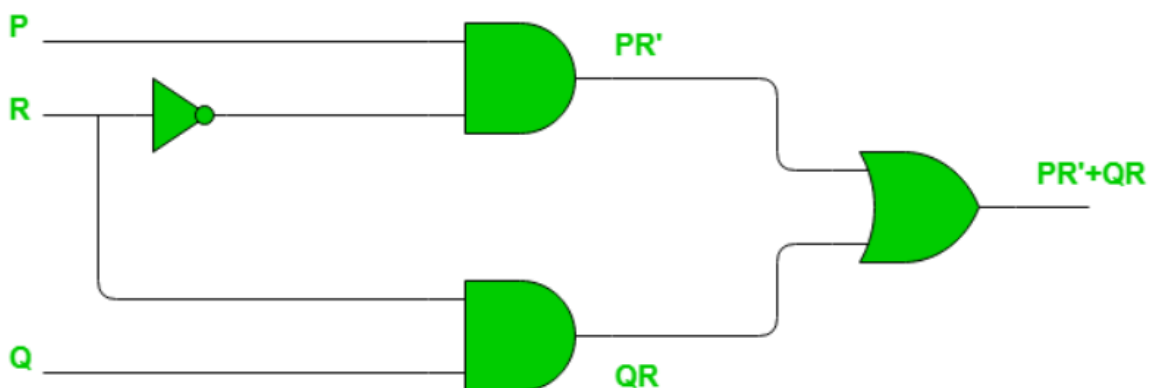
- ✿ A static hazard can be removed by covering the adjacent cells with a redundant grouping that overlap both groupings.

#### ✿ Detection of Static-1 Hazard:

- ✿ Let's consider static-1 hazard first. To detect a static-1 hazard for a digital circuit following steps are used:

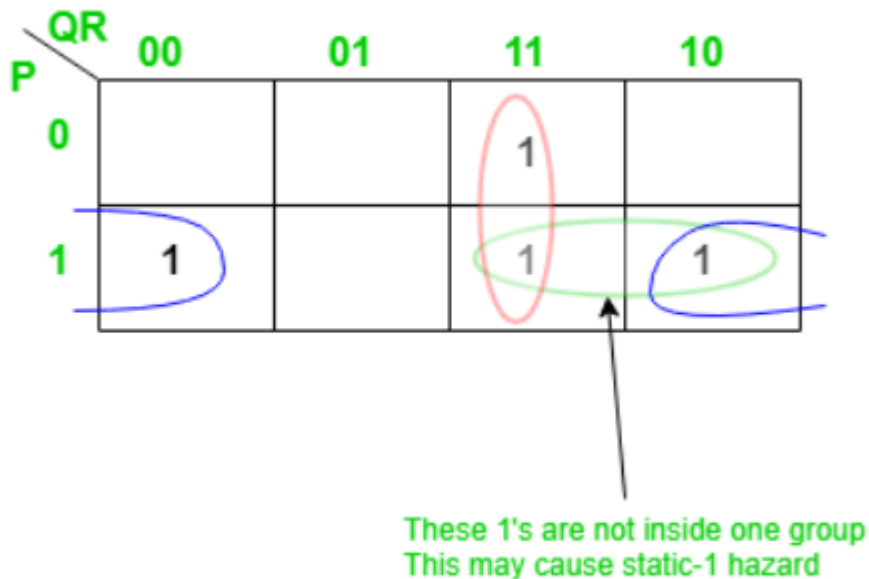
- ✿ Step-1: Write down the output of the digital circuit, say Y.
- ✿ Step-2: Draw the K-map for this function Y and note all adjacent 1's.
- ✿ Step-3: If there exists any pair of cells with 1's which do not occur to be in the same group (i.e. prime implicant), it indicates the presence of a static-1 hazard. Each such pair is a static-1 hazard.

- ✿ Example – Consider the circuit shown below.



$$F(P, Q, R) = QR + P\bar{R} = \sum m\{3, 4, 6, 7\}$$

- Let's draw the K-map for this Boolean function as follows:



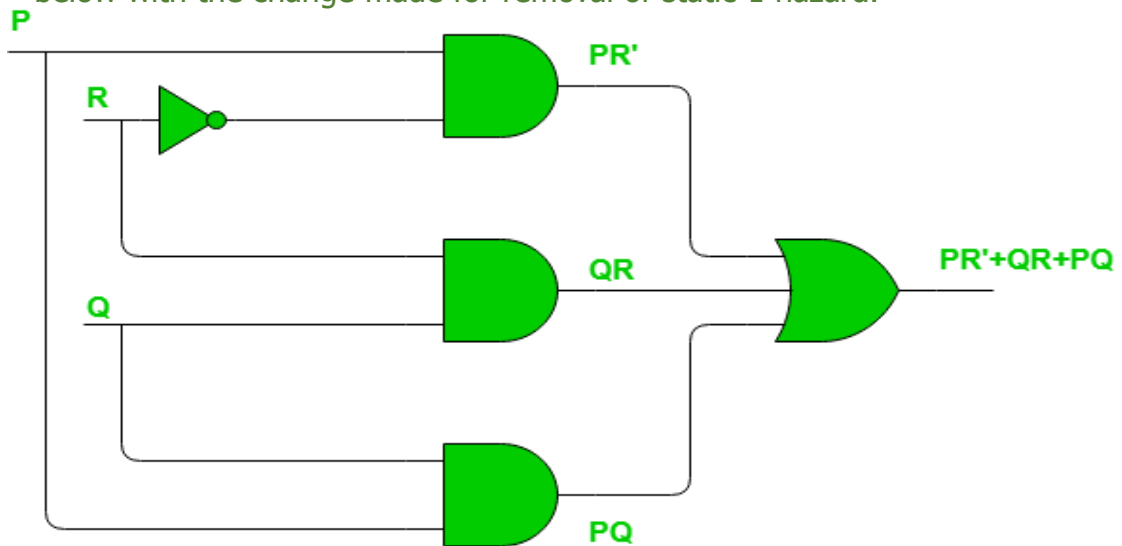
- The pair of 1's encircled as green are not part of the grouping/pairing provided by the output of this Boolean function. This will cause a static-1 hazard in this circuit.

### ❁ Removal of Static-1 Hazard:

- ❁ Once detected, a static-1 hazard can be easily removed by introducing some more terms (logic gates) to the function (circuit). The most common idea is to add the missing group in the existing Boolean function, as adding this term would not affect the function by any mean but it will remove the hazard. Since in above example the pair of 1's encircled with blue color causes the static-1 hazard, we just add this as a prime implicant to the existing function as follows:

$$F(P, Q, R) = QR + P\bar{R} + PQ = \sum m\{3, 4, 6, 7\}$$

- Note that there is no difference in number of minterms of this function. The reason is that the static-1 hazards are based on how we group 1's (or 0's for static-0 hazard) for a given set of 1's in K-map. Thus, it does not make any difference in number of 1's in K-map. The circuit would look like as shown below with the change made for removal of static-1 hazard.

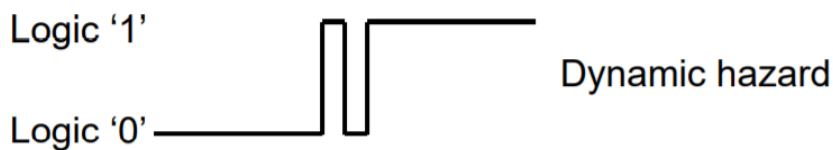


### ❁ Removal of Static-0 Hazard:

- ❁ Similarly, for Static-0 Hazards we need to consider 0's instead of 1's and if any adjacent 0's in K-map are not grouped into same group that may cause a static-0 hazard. The method to detect and resolve the static-0 hazard is completely same as the one we followed for static-1 hazard except that instead of SOP, POS will be used as we are dealing with 0's in this case.

## ❁ Dynamic Hazard:

- A dynamic hazard is defined as a transient change occurring three or more times at an output terminal of a logic network when the output is supposed to change only once during a transition between two inputs states.



- Dynamic Hazards occurs when an input changes and a circuit output should change 0 to 1 or 1 to 0, but temporarily flips between values.
- Dynamic Hazards occur when the output of a network is to change between its two logic states, but a momentary false output signal occurs during the transient behaviour.
- A dynamic hazard is the possibility of an output changing more than once as a result of a single input change. Dynamic hazards often occur in larger logic circuits where there are different routes to the output (from the input). If each route has a different delay, then it quickly becomes clear that there is the potential for changing output values that differ from the required / expected output.

- Example: A logic circuit is meant to change output state from '1' to '0', but instead changes from '1' to '0' then '1' and finally rests at the correct value '0'. This is a dynamic hazard.

### ❁ **Dynamic Hazard Elimination:**

- Dynamic hazard takes a more complex method to resolve (which we shall not cover).
- As a final note on dynamic hazards, it should be noted that if all static hazards have been eliminated from a circuit, then dynamic hazards cannot occur.

### ❁ **Essential Hazard:**

- Essential hazard is a type of hazard that exists only in asynchronous sequential circuits with two or more feedbacks.
- An essential hazard is caused by unequal delays along two or more paths that originate from the same input.
- An excessive delay through an inverter circuits in comparison to the delay associated with the feedback path may cause essential hazard.



## ❁ Essential Hazard Elimination:

- Essential hazard can be eliminated by adding redundant gates as in static hazards.
- They can be eliminated by adjusting the amount of delay in the affected path.
- For this, each feedback loop must be designed with extra care to ensure that the delay in the feedback path is long enough compared to the delay other signals that originate from the input terminals.

## 5.PROGRAMMABLE LOGIC DEVICES

A memory unit is a device to which binary information is transferred for storage and from which information is retrieved when needed for processing. When data processing takes place, information from memory is transferred to selected registers in the processing unit. Intermediate and final results obtained in the processing unit are transferred back to be stored in memory. Binary information received from an input device is stored in memory, and information transferred to an output device is taken from memory. A memory unit is a collection of cells capable of storing a large quantity of binary information.

There are two types of memories that are used in digital systems:

Random-access memory (RAM)

Read-only memory (ROM)

RAM stores new information for later use. The process of storing new information into memory is referred to as a memory write operation. The process of transferring the stored information out of memory is referred to as a memory read operation. RAM can perform both write and read operations. ROM can perform only the read operation. This mean that suitable binary information is already stored inside memory and can be retrieved or read at any time. However, that information cannot be altered by writing.

ROM is a programmable logic device (PLD). The binary information that is stored within such a device is specified in some fashion and then embedded within the hardware in a process is referred to as programming the device. The word “programming” here refers to a hardware procedure which specifies the bits that are inserted into the hardware configuration of the device.

ROM is one example of a PLD. Other such units are the programmable logic array (PLA), programmable array logic (PAL), and the field-programmable gate array (FPGA). A PLD is an integrated circuit with internal logic gates connected through electronic paths that behave similarly to fuses. In the original state of the device, all the fuses are intact. Programming the device involves blowing those fuses along the paths that must be removed in order to obtain the particular configuration of the desired logic function.



The above diagrams are used to represent multiple logic OR gate in conventional symbol and array logic symbol.

A typical PLD may have hundreds to millions of gates interconnected through hundreds to thousands of internal paths. In order to show the internal logic diagram of such a device in a concise form, it is necessary to employ a special gate symbol applicable to array logic. Figure shows the conventional and array logic symbols for a multiple-input OR gate. Instead of having multiple input lines into the gate, we draw a single line entering the gate. The input lines are drawn perpendicular to this single line and are connected to the gate through internal fuses. In a similar fashion, we can draw the array logic for an AND gate.

# PROGRAMMABLE READ -ONLY MEMORY

A ROM is essentially a memory device in which permanent binary information is stored. The binary information must be specified by the designer and is then embedded in the unit to form the required interconnection pattern. Once the pattern is established, it stays within the unit even when power is turned off and on again.

A block diagram of a ROM consisting of  $k$  inputs and  $n$  outputs is shown in figure. The inputs provide the address for memory, and the outputs give the data bits of the stored word that is selected by the address. The number of words in a ROM is determined from the fact that  $k$  address input lines are needed to specify  $2^k$  words. Note that ROM does not have data inputs, because it does not have a write operation. Integrated circuit ROM chips have one or more enable inputs and sometimes come with three-state outputs to facilitate the construction of large arrays of ROM.

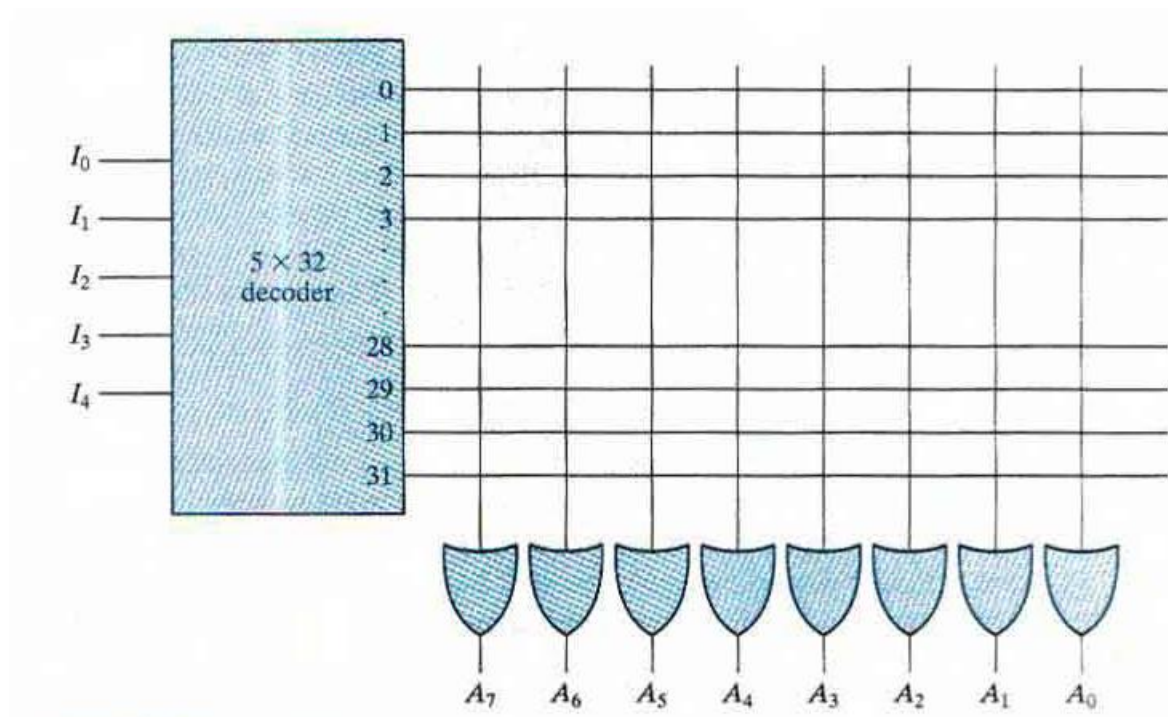


Block Diagram Representation of ROM

## EXAMPLE: 32X8 PROM

Consider, for example, a 32 X 8 ROM. The unit consists of 32 words of 8 bits each. There are five input lines that from the binary numbers from 0 through 31 for the address. Figure 7.10 shows the internal logic construction of this ROM. The five inputs are decoded into 32 distinct outputs by means of a 5 X 32 decoder. Each output of the decoder represents a memory address.

The 32 outputs of the decoder are connected to each of the eight OR gates. The diagram shows the array logic convention user in complex circuits. Each OR gate must be considered as having 32 inputs. Each output of the decoder is connected to one of the inputs of each OR gate. Since each OR gate has 32 input connections and there are 8 OR gates, the ROM contains  $32 \times 8 = 256$  internal connections. In general a  $2^k \times n$  ROM will have an internal  $k \times 2^k$  decoder and  $n$  OR gates. Each OR gate has  $2^k$  inputs, which are connected to each of the outputs of the decoder



# PROGRAMMING THE ROM

The hardware procedure that programs the ROM blows fuse links in accordance with a given truth table. For example, programming the ROM according to the truth table given by Table results in the configuration shown in figure.

Every 0 listed in the truth table specifies the absence of a connection, and every 1 listed specifies a path that is obtained by a connection. For example, the table specifies the eight-bit word 10110010 for permanent storage at address 3.

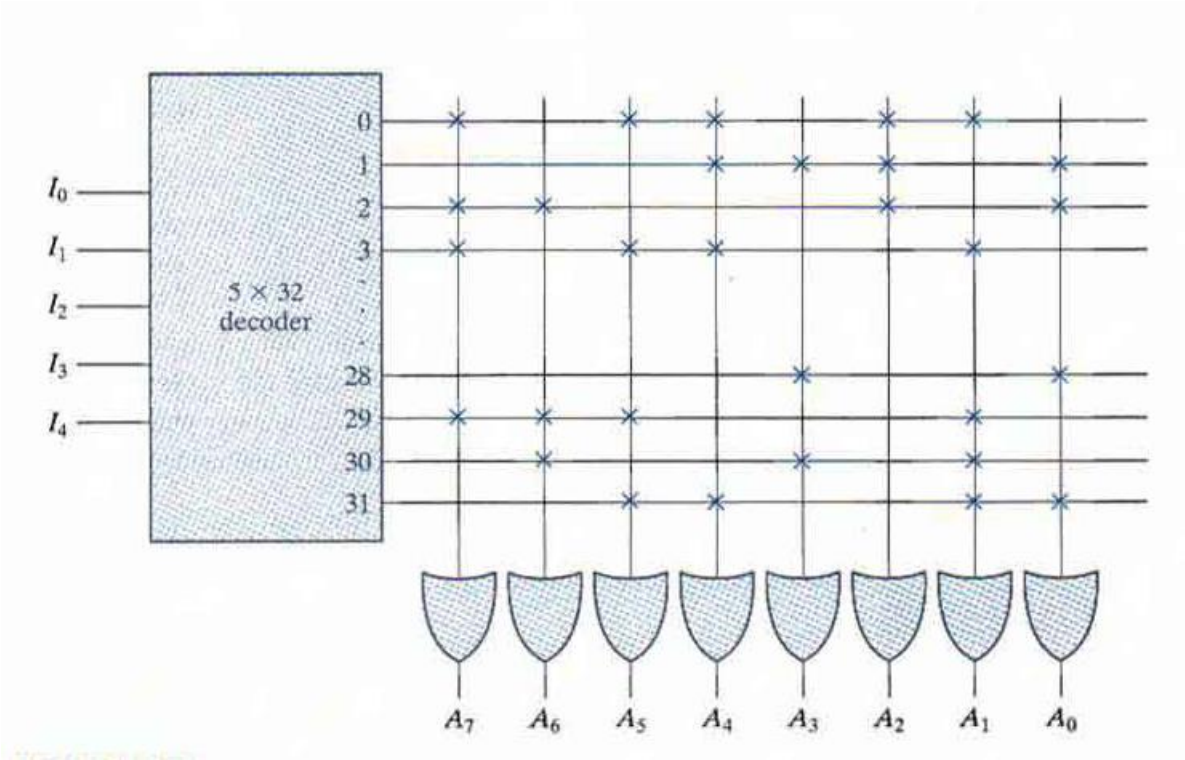
The four 0's in the word are programmed by blowing the fuse links between output 3 of the decoder and the inputs of the OR gates associated with outputs A6, A3, A2, and

in

Inputs					Outputs							
$I_4$	$I_3$	$I_2$	$I_1$	$I_0$	$A_7$	$A_6$	$A_5$	$A_4$	$A_3$	$A_2$	$A_1$	$A_0$
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
		⋮						⋮				
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

## PROM IMPLEMENTATION

When the input of the ROM is 00011, all the outputs of the decoder are 0 except for output 3, which is a logic 1, the signal equivalent to logic 1 at decoder output 3 propagates through the connections to the OR gate outputs of A7, A5, A4, and A1. The other four outputs remain at 0. The result is that the stored word 10110010 is applied to the eight data outputs.

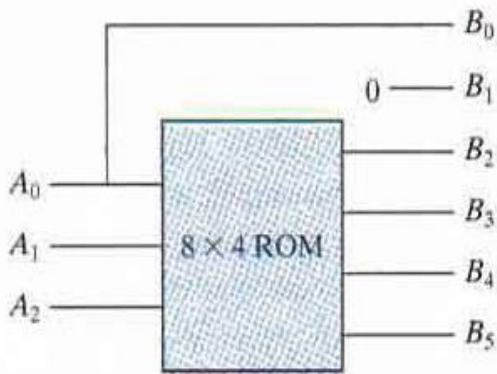


Example 2

CIRCUIT IMPLEMENTATION

Truth Table

Inputs			Outputs						Decimal
A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	B <sub>5</sub>	B <sub>4</sub>	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49



(a) Block diagram

A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	B <sub>5</sub>	B <sub>4</sub>	B <sub>3</sub>	B <sub>2</sub>
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

(b) ROM truth table

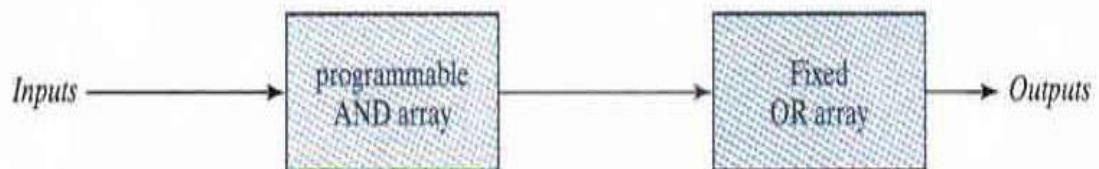


## BASIC CONFIGURATION OF THREE PLD'S

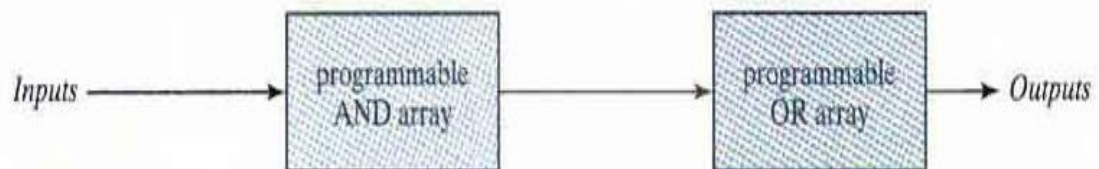
The block diagram representation of three programmable logic devices.



(a) Programmable read-only memory (PROM)



(b) Programmable array logic (PAL)



(c) Programmable logic array (PLA)

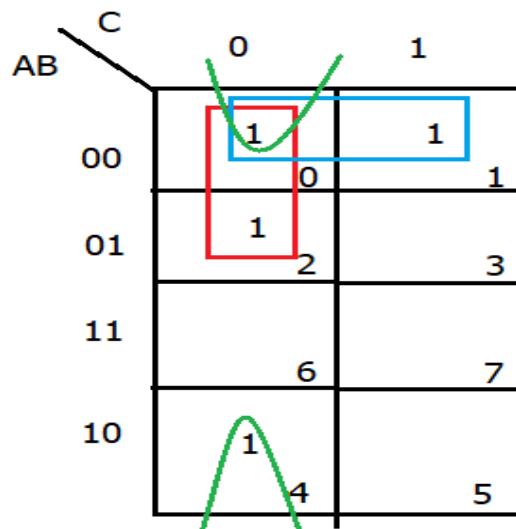
## PROGRAMMABLE LOGIC ARRAY(PLA)

Implement the following Boolean functions with a PLA

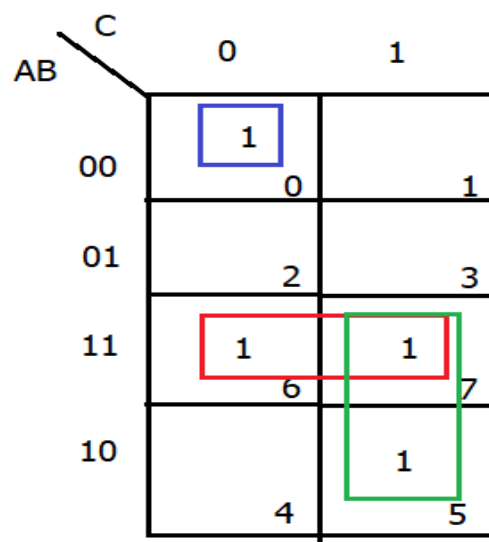
$$F1(A,B,C) = \Sigma(0,1,2,4)$$

$$F2(A,B,C) = \Sigma(0,5,6,7)$$

KMAP



$$F1 = A'C' + A'B' + B'C'$$



$$F2 = A'B'C' + AB + AC$$

KMAP to find F1 complement

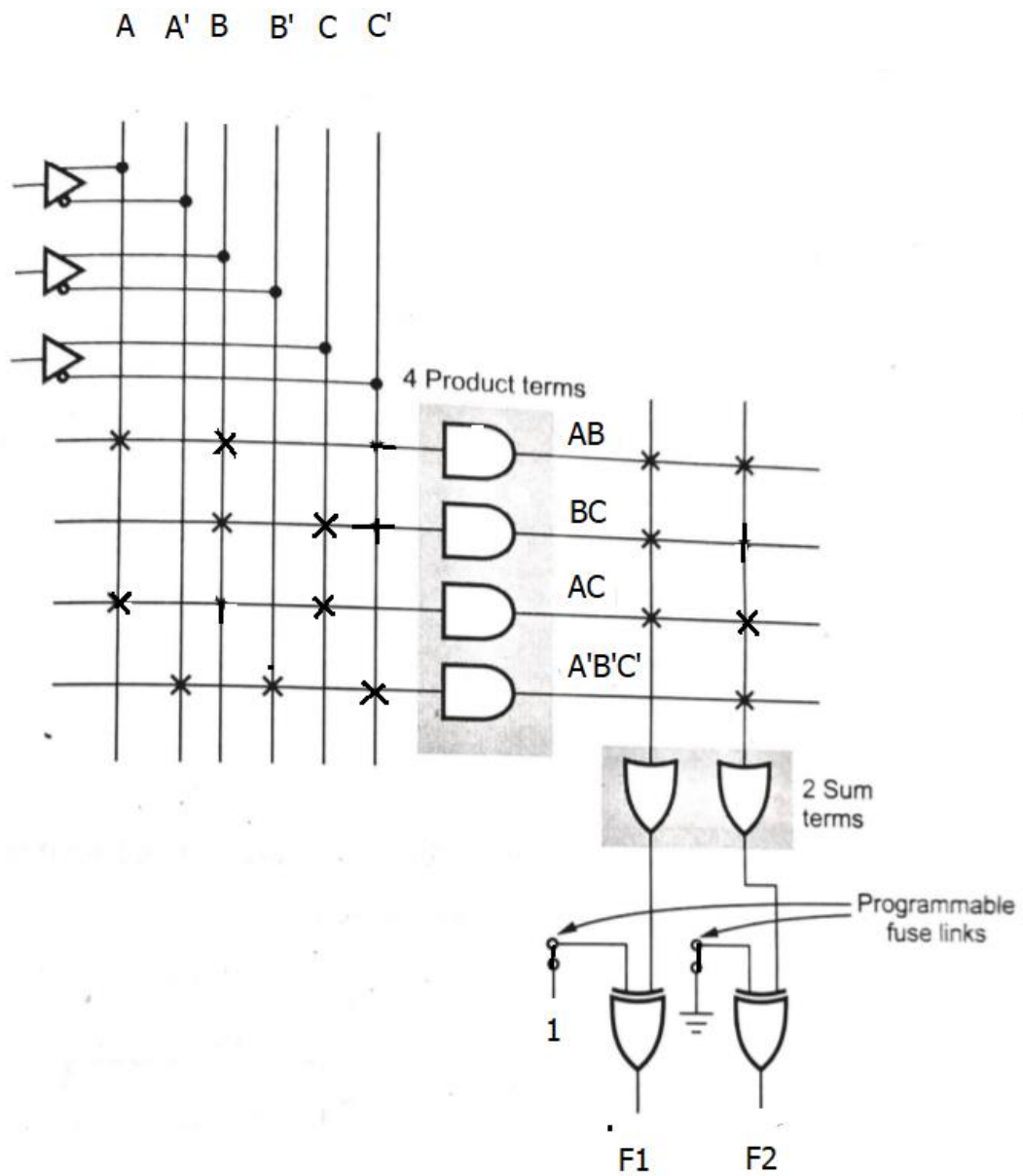
		C	
		0	1
AB	00	0	1
	01	2	3
	11	6	7
	10	4	5

$$F1' = AB+BC+AC$$

PLA Program Table:

PLA programming table						
		Inputs			Outputs	
Product term					(C)	(T)
		A	B	C	$F_1$	$F_2$
AB	1	1	1	-	1	1
AC	2	1	-	1	1	1
BC	3	-	1	1	1	-
$A'B'C'$	4	0	0	0	-	1

## PLA Implementation



Example 2

Design BCD to Excess 3 code converter with a PLA

Truth table

B3	B2	B1	B0	E3	E2	E1	E0
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

For  $E_3$ 

$B_3B_2 \backslash B_1B_0$		00	01	11	10
		00	01	11	10
00	00	0	0	0	0
01	01	0	1	1	1
11	11	X	X	X	X
10	10	1	1	X	X

$$E_3 = B_3 + B_2B_0 + B_2B_1$$

For  $E_2$ 

$B_3B_2 \backslash B_1B_0$		00	01	11	10
		00	01	11	10
00	00	0	1	1	1
01	01	1	0	0	0
11	11	X	X	X	X
10	10	0	1	X	X

$$E_2 = B_2\bar{B}_1\bar{B}_0 + \bar{B}_2B_0 + \bar{B}_2B_1$$

For  $E_1$ 

$B_3B_2 \backslash B_1B_0$		00	01	11	10
		00	01	11	10
00	00	1	0	1	0
01	01	1	0	1	0
11	11	X	X	X	X
10	10	1	0	X	X

$$E_1 = \bar{B}_1\bar{B}_0 + B_1B_0$$

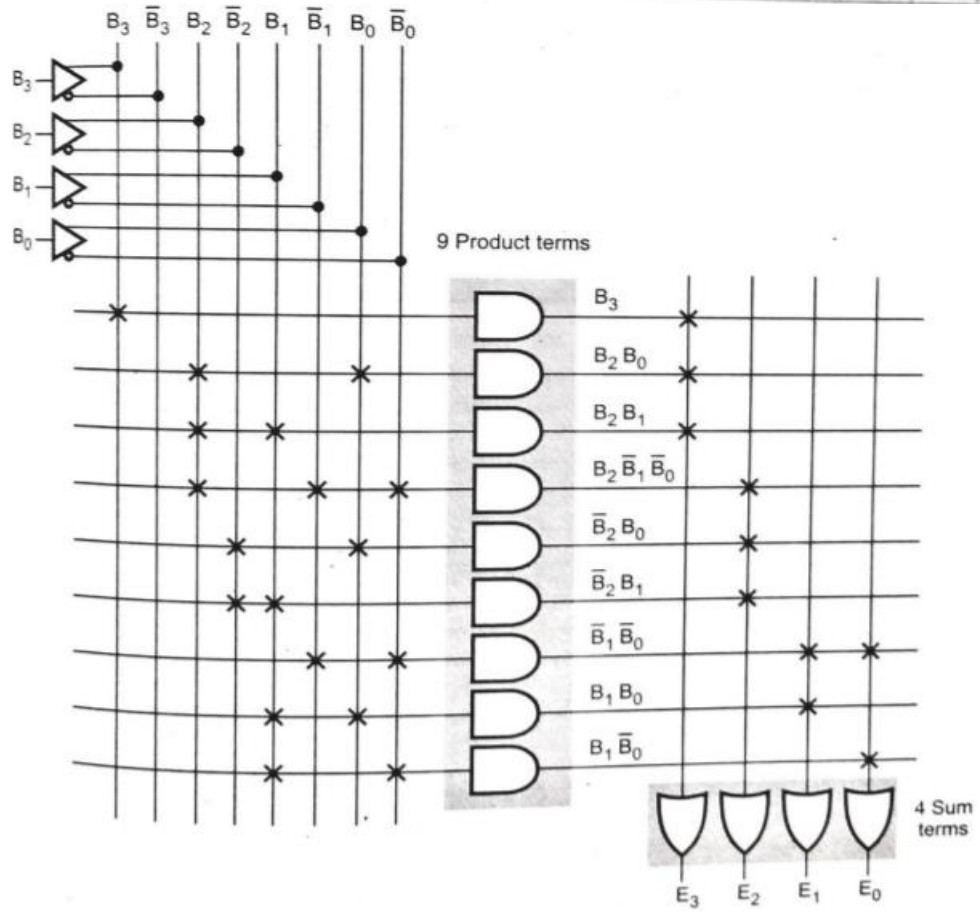
For  $E_0$ 

$B_3B_2 \backslash B_1B_0$		00	01	11	10
		00	01	11	10
00	00	1	0	0	1
01	01	1	0	0	1
11	11	X	X	X	X
10	10	1	0	X	X

$$E_0 = \bar{B}_1\bar{B}_0 + B_1\bar{B}_0$$

PLA Program Table and Circuit

Product terms		Inputs				Outputs			
		B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	E <sub>3</sub>	E <sub>2</sub>	E <sub>1</sub>	E <sub>0</sub>
B <sub>3</sub>	1	1	-	-	-	1	-	-	-
B <sub>2</sub> B <sub>0</sub>	2	-	1	-	1	1	-	-	-
B <sub>2</sub> B <sub>1</sub>	3	-	1	1	-	1	-	-	-
B <sub>2</sub> $\overline{B_1}$ $\overline{B_0}$	4	-	1	0	0	-	1	-	-
$\overline{B_2}$ B <sub>0</sub>	5	-	0	-	1	-	1	-	-
$\overline{B_2}$ B <sub>1</sub>	6	-	0	1	-	-	1	-	-
$\overline{B_1}$ $\overline{B_0}$	7	-	-	0	0	-	-	1	1
B <sub>1</sub> B <sub>0</sub>	8	-	-	1	1	-	-	1	-
B <sub>1</sub> $\overline{B_0}$	9	-	-	1	0	-	-	-	1
						T	T	T	T
						T/C			



## COMBINATIONAL PLD'S

The PROM is a combinational programmable logic device(PLD) – an integrated circuit with programmable gates divided into an AND array and an OR array to provide an AND-OR implementation. There are three major types of combinational PLD's differing in the placement of programmable connection in the AND-OR array. The most flexible PLD is PLA in which both AND and OR arrays can be programmed.

Programmable Array Logic:

The PAL has a programmable AND array and a fixed OR array. The AND gates programmed to provide the product terms of the Boolean function, which are logically summed in each OR gate

Example1

Design a PAL with following Boolean functions

$$w(A, B, C, D) = \sum(2, 12, 13)$$

$$x(A, B, C, D) = \sum(7, 8, 9, 10, 11, 12, 13, 14, 15)$$

$$y(A, B, C, D) = \sum(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$$

$$z(A, B, C, D) = \sum(1, 2, 8, 12, 13)$$



PAL PROGRAM TABLE

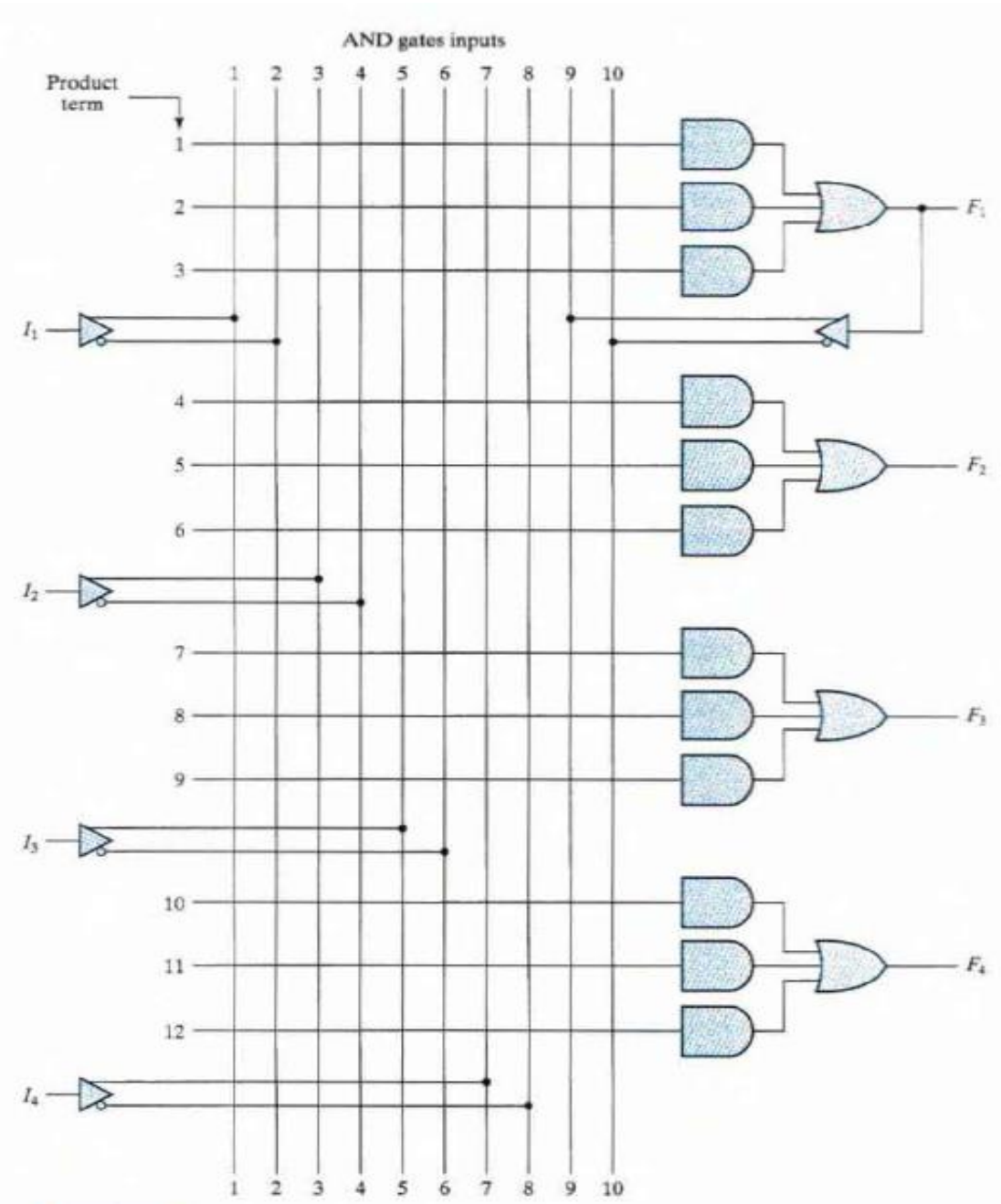
Simplifying the four functions to a minimum number of terms results in the following Boolean functions:

$$w = ABC' + A'B'CD'$$
$$x = A + BCD$$
$$y = A'B + CD + B'D'$$
$$z = ABC' + A'B'CD' + AC'D' + A'B'C'D$$
$$= w + AC'D' + A'B'C'D$$

PAL Programming Table

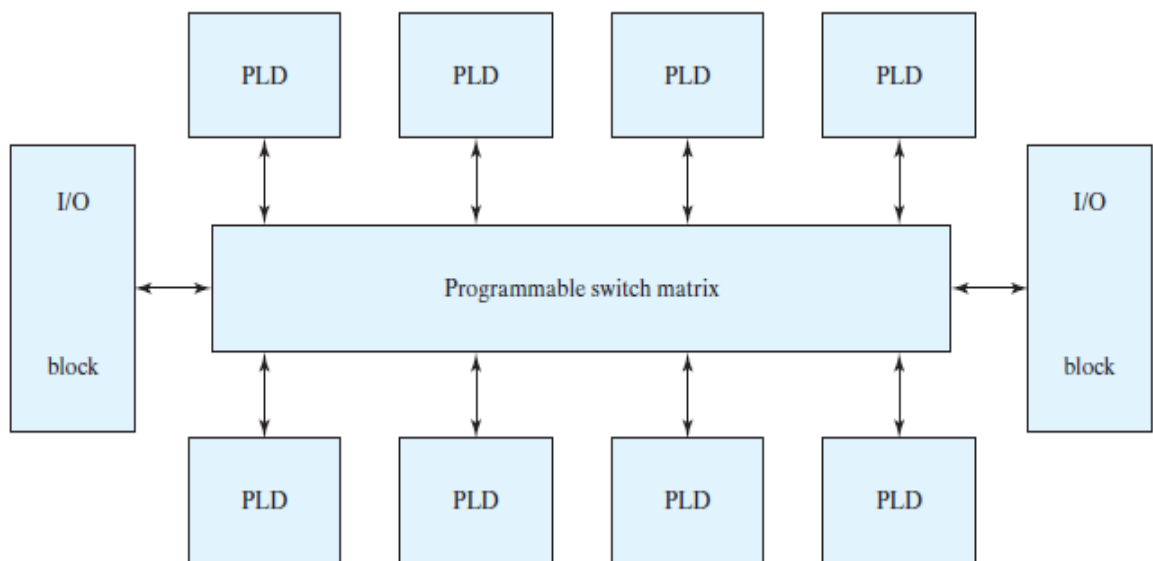
Product Term	AND Inputs					Outputs
	A	B	C	D	w	
1	1	1	0	—	—	$w = ABC' + A'B'CD'$
2	0	0	1	0	—	
3	—	—	—	—	—	
4	1	—	—	—	—	$x = A + BCD$
5	—	1	1	1	—	
6	—	—	—	—	—	
7	0	1	—	—	—	$y = A'B + CD + B'D'$
8	—	—	1	1	—	
9	—	0	—	0	—	
10	—	—	—	—	1	$z = w + AC'D' + A'B'C'D$
11	1	—	0	0	—	
12	0	0	0	1	—	

PAL CIRCUIT



## SEQUENTIAL PROGRAMMABLE DEVICES

- ❁ Sequential programmable devices include both gates and flip-flops.
- ❁ There are three major types
  1. Sequential (or simple) programmable logic device (SPLD)
  2. Complex programmable logic device (CPLD)
  3. Field-programmable gate array (FPGA)
- ❁ CPLD
  - ❁ Complex Programmable Logic Device is a single device containing multiple Simple Programmable Logic Device SPLDs
  - ❁ SPLD includes flip-flops, in addition to the AND–OR array, Each section of an SPLD is called a *macrocell*.
  - ❁ CPLD consists of multiple PLDs interconnected through a programmable switch matrix.
  - ❁ Input–output (I/O) blocks provide the connections to IC pins. The pins are driven by a three state buffer and programmed as input or output.
  - ❁ Configuration of CPLD is shown below:



- ❁ The switch matrix receives inputs from the I/O block and directs them to the individual macrocells and the outputs selected from macrocells are given as outputs. 8 to 16 macrocells are present in each PLD.
- ❁ Unused product terms in PLDs can be used by other nearby macrocells.

## ❁ FPGA

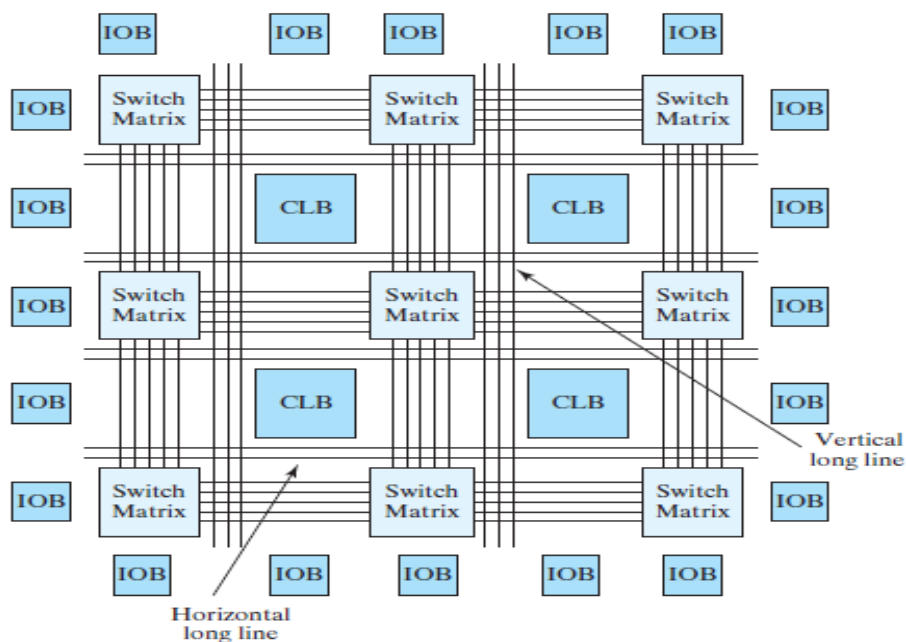
❁ Field-Programmable Gate Array (FPGA) is a user programmable VLSI circuit.

❁ FPGA consists of an array of millions of logic blocks, with multiple programmable input and output blocks and connected through programmable interconnections.

❁ FPGA logic block consists of lookup tables, multiplexers, gates, and flip-flops.

❁ It is reprogrammed every time power is turned on and makes it suitable for various applications using different logic implementations in the program.

❁ The basic architecture is given as follows (Xilinx Spartan):



❁ It consists of an array of configurable logic blocks (CLBs), a variety of local and global routing resources, and input-output (I/O) blocks (IOBs), programmable I/O buffers, and an SRAM-based configuration memory

❁ CLBs consists of a programmable lookup table, multiplexers, registers, and paths for control signals

❁ CLB have two storage devices that can be configured as edge-triggered flip-flops with a common clock

❁ Programmable interconnect resources of the device connect CLBs and IOBs, either directly or through switch boxes

## 10. Assignments

✿ A ROM chip of 4,096 \* 8 bits has two chip select inputs and operates from a 5-V power supply. How many pins are needed for the integrated circuit package?  
Draw a block diagram, and label all input and output terminals in the ROM.

✿ Tabulate the PLA programming table for the four Boolean functions listed below.  
Minimize the numbers of product terms.

$$A(x, y, z) = (1, 3, 5, 6) \quad B(x, y, z) = (0, 1, 6, 7)$$

$$C(x, y, z) = (3, 5) \quad D(x, y, z) = (1, 2, 4, 5, 7)$$

✿ Draw a PLA circuit to implement the functions

$$F1 = AB + AC + ABC$$

$$F2 = (AC + AB + BC)$$

✿ Tabulate the truth table for an 8 \* 4 ROM that implements the Boolean functions

$$A(x, y, z) = (0, 3, 4, 6) \quad B(x, y, z) = (0, 1, 4, 7)$$

$$C(x, y, z) = (1, 5) \quad D(x, y, z) = (0, 1, 3, 5, 7)$$

✿ An asynchronous sequential circuit has two internal states and one output. The excitation and output functions describing the circuit are as follows:

$$Y1 = X1X2 + X1Y2' + X2'Y1$$

$$Y2 = X2 + X1Y1'Y2 + X1'Y1$$

$$Z = X2 + Y1$$

(a) Draw the logic diagram of the circuit.

(b) Derive the transition table and output map.

(c) Obtain a flow table for the circuit.

## 11. Part A Q & A (with K level and CO)

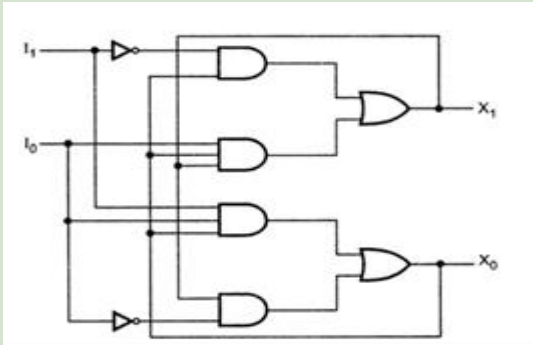
PART A		
Questions and Answers	Blooms Level	COs
<p>1. List basic types of programmable logic devices.</p> <p>Read only memory Programmable logic Array Programmable Array Logic</p>	K1	CO5
<p>2. Define ROM</p> <p>A read only memory is a device that includes both the decoder and the OR gates within a single IC package.</p>	K1	CO5
<p>3. Define PAL.</p> <p>PAL programmable logic device with fixed OR array and a programmable AND array. Because only AND gates are programmable, PAL is easier to program, but it is not as flexible as PLA.</p>	K1	CO5
<p>4. Define address and word.</p> <p>In a ROM, each bit combination of the input variable is called on address. Each bit combination that comes out of the output lines is called a word.</p>	K1	CO5
<p>5. What are the types of ROM?</p> <p>Masked ROM. Programmable Read only Memory Erasable Programmable Read only memory. Electrically Erasable Programmable Read only Memory.</p>	K1	CO5
<p>6. What is programmable logic array? How does it differ from ROM?</p> <p>PLA is a programmable logic device which consists of programmable AND and OR array. A PLA is similar to a ROM in concept; however it does not provide full decoding of the variables and does not generates all the minterms as in the ROM.</p>	K1	CO5
<p>7. What is deadlock condition?</p> <p>A condition resulting when one task is waiting to access a resource that another is holding, and vice versa</p>	K1	CO4

<p>8. State the difference between static – 0 and static – 1 hazard.</p> <p>Static-1 Hazard: the output is currently 1 and after the inputs change, the output momentarily changes to 0 before settling on 1</p> <p>Static-0 Hazard: the output is currently 0 and after the inputs change, the output momentarily changes to 1 before settling on 0</p>	K1	CO4
<p>9.What are the two types of asynchronous sequential circuits?</p> <p>The two types of asynchronous sequential circuits are:</p> <p>a) Fundamental mode circuits</p> <p>b) Pulse mode circuits</p>	K1	CO4
<p>10. Define races in asynchronous sequential circuits.</p> <p>When 2 or more binary state variables change their value in response to a change in an input variable, race condition occurs in an asynchronous sequential circuit. In case of unequal delays, a race condition may cause the state variables to change in an unpredictable manner.</p>	K1	CO4
<p>11. Define Dynamic Hazard.</p> <p>A dynamic hazard is the possibility of an output changing more than once as a result of a single input change .</p>	K1	CO4
<p>12. Define Static Hazards</p> <p>When one input variable changes, the output changes momentarily when it shouldn't</p>	K1	CO4
<p>13. What is a critical race condition?</p> <p>A critical race condition occurs when the order in which internal variables are changed determines the eventual state that the state machine will end up in.</p>	K1	CO4
<p>14. What is a non-critical race condition?</p> <p>A non-critical race condition occurs when the order in which internal variables are changed does not determine the eventual state that the state machine will end up in.</p>	K1	CO4

<p>15. What is a flow table ?</p> <p>A state transition table with its internal state being symbolised with letters.</p>	K1	CO4
<p>16. What is a implication table ?</p> <p>The rows in the primitive flow table are merged by first obtaining all compatible pairs of states by means of the implication table.</p>	K1	CO4
<p>17. How is a primitive table derived?</p> <p>The derivation of the primitive flow table can be facilitated by deriving a table that lists all the possible total states in the circuit.</p>	K1	CO4
<p>18. What are the steps in the design of asynchronous sequential circuits?</p> <ol style="list-style-type: none"> <li>1.State the design specifications.</li> <li>2. Derive a primitive flow table.</li> <li>3. Reduce the flow table by merging the rows.</li> <li>4. Make a race-free binary state assignment.</li> <li>5. Obtain the transition table and output map.</li> <li>6. Obtain the logic diagram</li> </ol>	K1	CO4
<p>19. What is multiple row method?</p> <p>In the multiple-row assignment, each state in the original flow table is replaced by two or more combinations of state variables.</p>	K1	CO4
<p>20. What is shared row method?</p> <p>The method for making race-free stale assignments by adding extra rows in the flow table is referred to as the shared-row method.</p>	K1	CO4



## 12. Part B Qs (with K level and CO)

PART B		
1. Analyze the given fundamental mode asynchronous sequential circuit	K2	CO4
		
2. An asynchronous sequential circuit is described by the following excitation and output function $Y_1 = X_1X_2 + X_1Y_2 + X_2Y_1$ $Y_2 = X_2 + X_1Y_1Y_2 + X_1Y_1$ $Z = X_2 + Y_1$ Draw logic diagram and derive transition table.	K2	CO4
3. Design an asynchronous sequential circuit with two inputs X and Y and with one output Whenever Y is 1, input X is transferred to Z. When Y is 0, the output does not change for any change in X.	K3	CO4
4. Explain races and cycles in asynchronous sequential circuits.	K2	CO4
5. Explain various types of hazards with examples	K2	CO4
6. Obtain hazard free realization for the given functions $F(A,B,C,D) = \sum m(0,5,7,8,10,11,12,14)$ $F(A,B,C) = \sum m(1,2,3,6,7)$	K2	CO4
7. Implement the following functions using PLA and PROM $F(x,y,z) = \sum m(0,1,3,5,7)$	K2	CO5
8. (i) Draw the configuration of 32X5 PROM (ii) Implement full adder using PROM	K2	CO5
9. Design binary to gray converter using PLA	K3	CO5
10. Design BCD to excess converter using PAL	K3	CO5

### 13.Supportive online Certification courses (**NPTEL, Swayam, Coursera, Udemy, etc.,**)

#### ✿ Swayam

Digital Circuits: [https://swayam.gov.in/nd1\\_noc20\\_ee70/preview](https://swayam.gov.in/nd1_noc20_ee70/preview)

Duration: 12 weeks, Start Date: 14 Sep 2020, End Date: 04 Dec 2020

#### ✿ Udemy

Digital Electric Circuits & Intelligent Electrical Devices

<https://www.udemy.com/course/digital-electric-circuits-intelligent-electrical-devices/>

#### ✿ Coursera

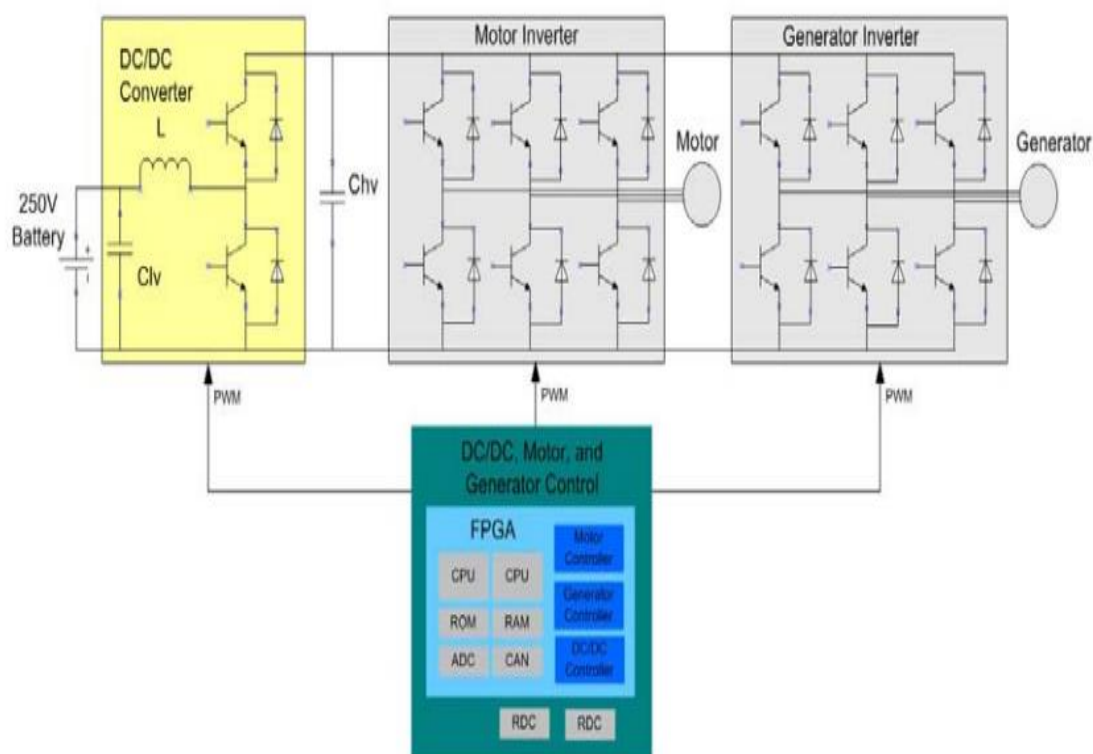
Build a Modern Computer from First Principles: From Nand to Tetris (Project-Centered Course)

<https://www.coursera.org/learn/build-a-computer>

## 14. Real time Applications in day to day life and to Industry

### ❁ Simplified Hybrid EV Power Control Architecture with Single FPGA

- ❁ FPGA technology allows multiple control functions to run in parallel on one device without the bottleneck of a single processor.
- ❁ A new architecture that integrates MG and VVC (DC-DC) control functions into a single FPGA is shown in the figure.



- ❁ In addition to the benefits of a reduced number of parts, the new architecture reduces the number of hardware and firmware interfaces.
- ❁ It also provides opportunities for complete system simulation and auto-code generation not possible with the existing architecture.

## 15. Contents beyond the Syllabus ( COE related Value added courses)

### ✿ Creating and Exporting a PLD Schematic – Simulations for Embedded Systems CoE

✿ Multisim is used to create specialized schematics that describe the logic of a PLD, such as the FPGA on the NI Digital Electronics FPGA Board (DEFB).

✿ Multisim also includes configurations to automate creating synthesizable and implementable designs for the DEFB.

✿ Steps Involved:

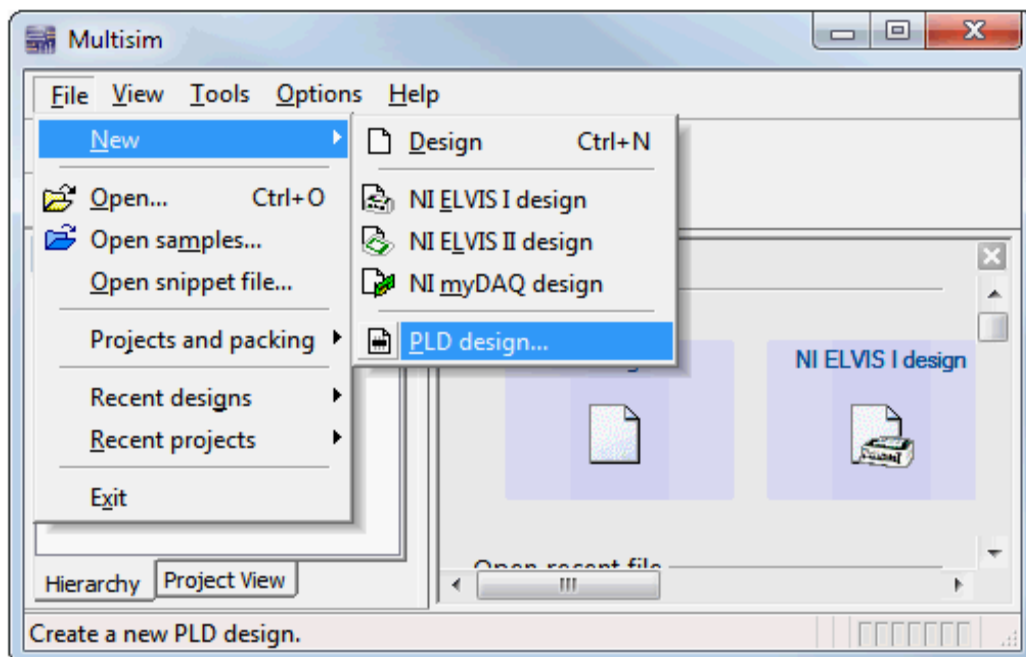
Create a new PLD design in Multisim

Describe the logic in Multisim

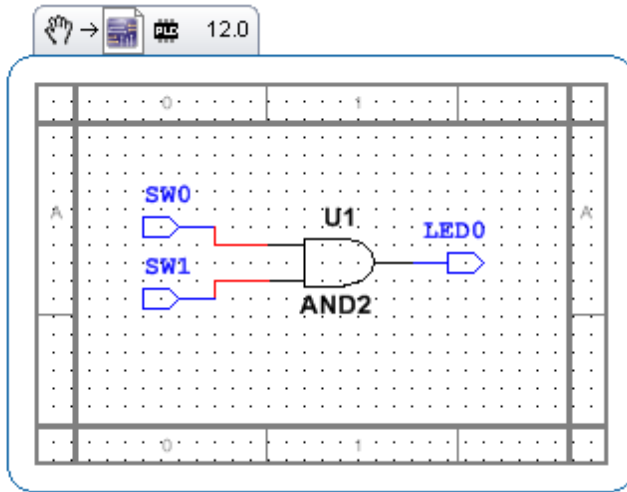
Export digital logic

✿ 1. Step 1: Create a New PLD Design in Multisim

The first step is to create a new PLD design in Multisim



## ⚙ Step 2: Describe the Logic in Multisim



## ⚙ Step 3: Export PLD Logic

Select Program the connected PLD.

Choose to Save the generated program file.

Select Next to proceed.

<http://www.ni.com/tutorial/10289/en/>

## 17. Prescribed Text Books & Reference Books

### ✿ TEXT BOOKS:

1. James W. Bignel, Digital Electronics, Cengage learning, 5th Edition, 2007.
2. M. Morris Mano, 'Digital Design with an introduction to the VHDL', Pearson Education, 2013.
3. Comer "Digital Logic & State Machine Design, Oxford, 2012.

### ✿ REFERENCES

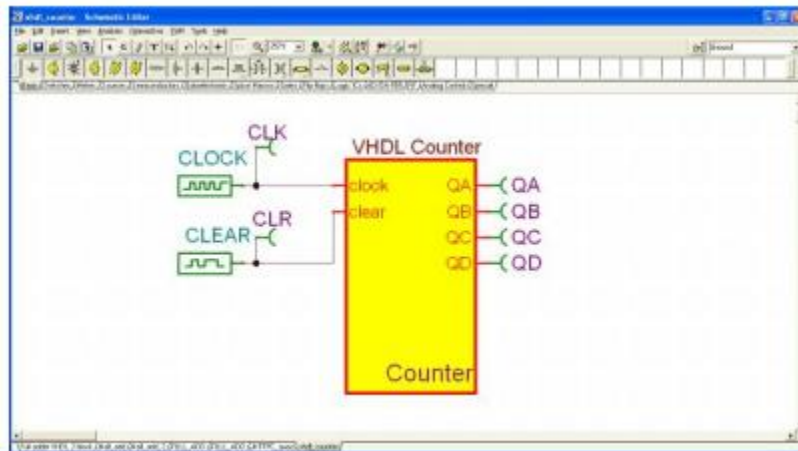
1. Mandal, "Digital Electronics Principles & Application, McGraw Hill Edu, 2013.
2. William Keitz, Digital Electronics-A Practical Approach with VHDL, Pearson, 2013.
3. Thomas L. Floyd, 'Digital Fundamentals', 11th edition, Pearson Education, 2015.
4. Charles H. Roth, Jr, Lizy Lizy Kurian John, 'Digital System Design using VHDL, Cengage, 2013.

# Digital Logic Circuits

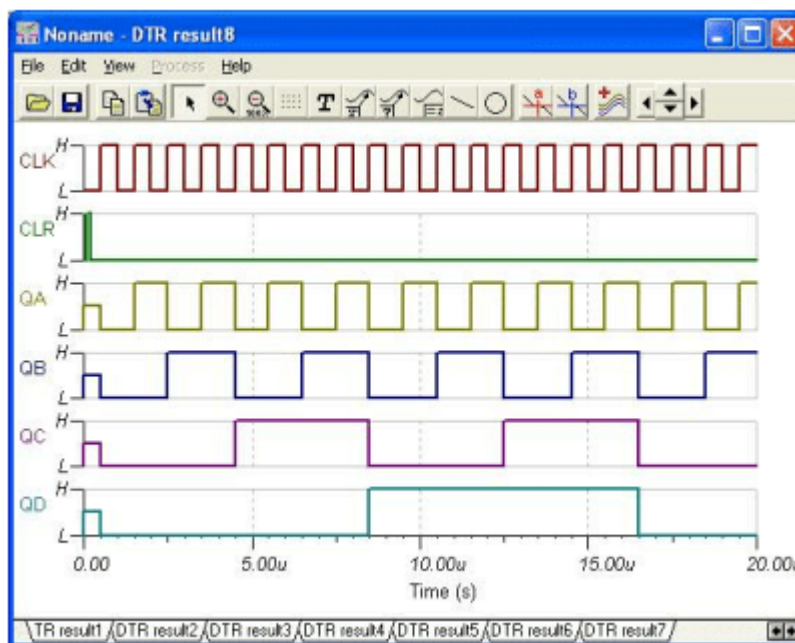
## Unit 5

## 8. Activity Based Learning

- ✿ Understanding and learning VHDL by generating synthesizable VHDL code using TINA software by Design soft. <https://www.tina.com/digital-vhdl-simulation/>
- ✿ Circuit for a counter, defined in VHDL using TINA software



- ✿ Output for the counter in TINA on Running Analysis / Digital VHDL simulation



- ✿ The VHDL circuit can be tested and also modified by changing the VHDL code and the effect can be seen immediately
- ✿ Practice using open source DEEDS: <https://www.digitalelectronicsdeeds.com/>
- ✿ <https://www.edaplayground.com/>



## Table of Contents

### ❁ UNIT V VHDL

Page No:

1. RTL Design -----	17
❁ VHDL -----	21
Operators	
❁ Combinational logic	
Data flow modelling -----	38
Structural modelling -----	54
❁ Sequential circuit -----	60
2. Introduction to	
❁ Packages -----	68
❁ Subprograms-----	70
❁ Test bench -----	73
3. Simulation / Tutorial -----	79
(Examples: Adders, Counters, Flip flops, Multiplexers & De multiplexers)	

# Register Transfer Level (RTL) Design

The Register Transfer Level (RTL) description specifies the digital system in terms of the registers, the operations performed, and the control that sequences the operations.

This type of description simplifies the design process because it consists of procedural statements that determine the relationship between the various operations of the design without reference to any specific structure.

The RTL description implies a certain hardware configuration among the registers, allowing the designer to create a design that can be synthesized automatically, rather than manually, into standard digital components.

A digital system is represented at the *register transfer level* (RTL) when it is specified by the following three components:

1. The set of registers in the system.
2. The operations that are performed on the data stored in the registers.
3. The control that supervises the sequence of operations in the system.

The statement  $R2 \leftarrow R1$  , denotes a transfer of the contents of register  $R1$  into register  $R2$ —that is, a replacement of the contents of register  $R2$  by the contents of register  $R1$  .

The operations in a digital system are controlled by signals that sequence the operations in a prescribed manner. Certain conditions that depend on results of previous operations may determine the sequence of future operations. The outputs of the control logic of a digital system are binary variables that initiate the various operations in the system's registers.

A control signal would determine when the operation actually executes. The controller in a digital system is a finite state machine whose outputs are the control signals governing the register operations.

A statement that specifies a register transfer operation implies that a data path (set of circuit connections) is available from the outputs of the source register to the inputs of the destination register. Data can be transferred serially or in parallel between registers.

## ⚙️ **Type of Operations:**

The type of operations most often encountered in digital systems can be classified into four categories:

1. Transfer operations, which transfer (i.e., copy) data from one register to another.
2. Arithmetic operations, which perform arithmetic (e.g., multiplication) on data in registers.
3. Logic operations, which perform bit manipulation (e.g., logical OR) of non numeric data in registers.
4. Shift operations, which shift data between registers.

⚙️ The transfer operation does not change the information content of the data being moved from the source register to the destination register unless the source and destination are the same.

⚙️ The other three operations change the information content during the transfer.

⚙️ The register transfer notation and the symbols used to represent the various register transfer operations are not standardized.

## ⚙️ **RTL Design is composed of**

1) Registers and combinational function blocks (e.g. adders and multiplexers) called the datapath and

2) A finite state machine, called the controller that controls the transfer of data through the function blocks and between the registers.

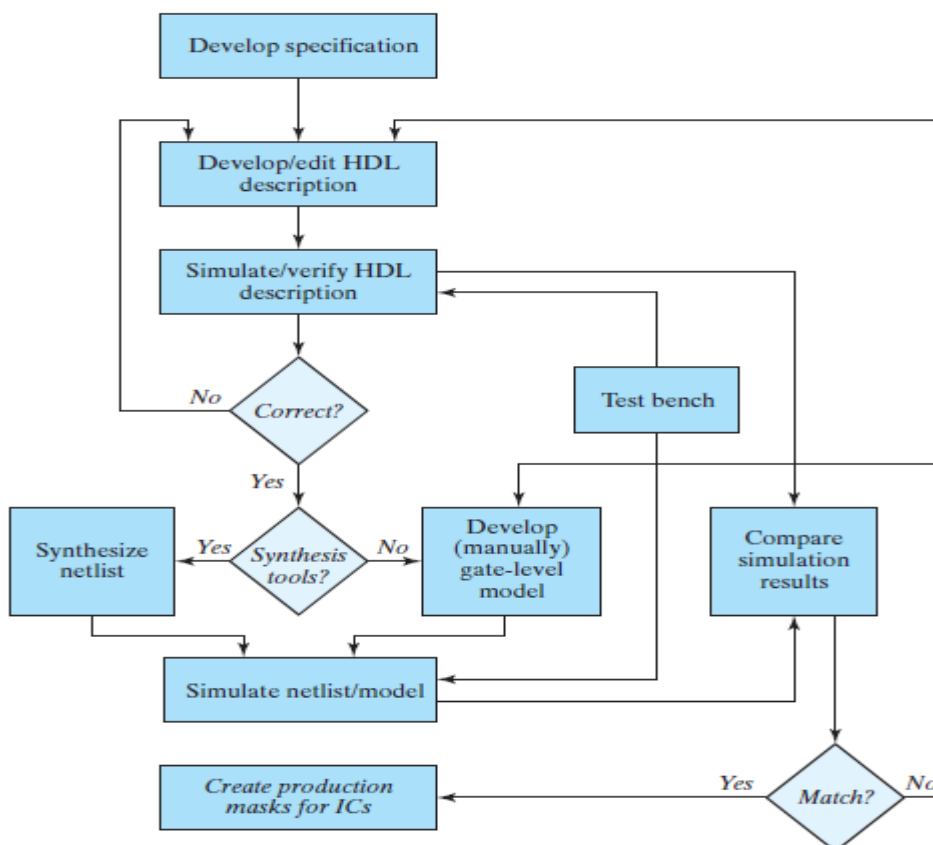
## ⚙️ **Steps in RTL Design :**

The steps in RTL design are:

- 1) Determine the number and sizes of registers needed to hold the data used by the device,
- 2) Determine the logic and arithmetic operations that need to be performed on these register contents, and
- 3) Design a state machine whose outputs determine how these register contents are updated with the results of those operations in order to obtain the desired results.

## RTL BY MEANS OF HDL

- ❁ Digital systems can be described at the RTL by means of a HDL.
- ❁ VHDL - **V**ery **H**igh-speed Integrated Circuit **H**ardware **D**escription **L**anguage description of RTL operations use combination of dataflow and behavioural constructs to specify the combinational logic functions and register operations implemented by hardware.
- ❁ Combinational circuit functions in RTL level are specified by means of concurrent or sequential signal assignment statements within a level sensitive process.
- ❁ Register transfers in RTL are specified by procedural statements within an edge – sensitive process
- ❁ Two types of assignments may be made in a process: Variable assignment (:= symbol) and signal assignment ( <= symbol)
- ❁ A simplified flowchart for HDL-based modeling, verification, and synthesis is shown below.



Simplified flowchart for HDL-based modeling, verification, and synthesis

- ✿ The RTL description of the HDL design is simulated and checked for proper operation.
- ✿ Its operational features must match those given in the specification for the behavior of the circuit.
- ✿ The test bench provides the stimulus signals to the simulator.
- ✿ If the result of the simulation is not satisfactory, the HDL description is corrected and checked again.
- ✿ After the simulation run shows a valid design, the RTL description is ready to be compiled by the logic synthesizer.
- ✿ All errors (syntax and functional) in the description must be eliminated before synthesis.
- ✿ The synthesis tool generates a netlist equivalent to a gate-level description of the design as it is represented by the model.
- ✿ If the model fails to express the functionality of the specification, the circuit will fail to do so also.
- ✿ The gate-level circuit is simulated with the same set of stimuli used to check the RTL design.
- ✿ If any corrections are needed, the process is repeated until a satisfactory simulation is achieved.
- ✿ The results of the two simulations are compared to see if they match.
- ✿ If they do not, the designer must change the RTL description to correct any errors in the design.
- ✿ Then the description is compiled again by the logic synthesizer to generate a new gate-level description.
- ✿ Once the designer is satisfied with the results of all simulation tests, the design of the circuit is ready for physical implementation in a technology.
- ✿ In practice, additional testing will be performed to verify that the timing specifications of the circuit can be met in the chosen hardware technology

# VHDL

## ❁ Introduction:

- ❁ VHDL stands for **V**ery High-speed Integrated Circuit **H**ardware **D**escription Language.
- ❁ It is an IEEE (Institute of Electrical and Electronics Engineers) standard hardware description language that is used to describe and simulate the behavior of complex digital circuits.
- ❁ This language was first introduced in 1981 for the Department of Defense (DoD) under the VHSIC program.
- ❁ VHDL (VHSIC Hardware Description Language) is becoming increasingly popular as a way to capture complex digital electronic circuits for both simulation and synthesis.
- ❁ VHDL is a programming language that has been designed and optimized for describing the behavior of digital circuits and systems.
- ❁ VHDL is different from languages like C in that it is intended to describe hardware. A C program is usually compiled to a platform-specific assembly language and run as a binary. VHDL, on the other hand, is usually simulated using a simulator.

## ❁ Purpose of VHDL:

VHDL is used for the following purposes:

- ❁ For describing hardware
- ❁ As a modeling language
- ❁ For simulation of hardware
- ❁ For early performance estimation of system architecture
- ❁ For the synthesis of hardware

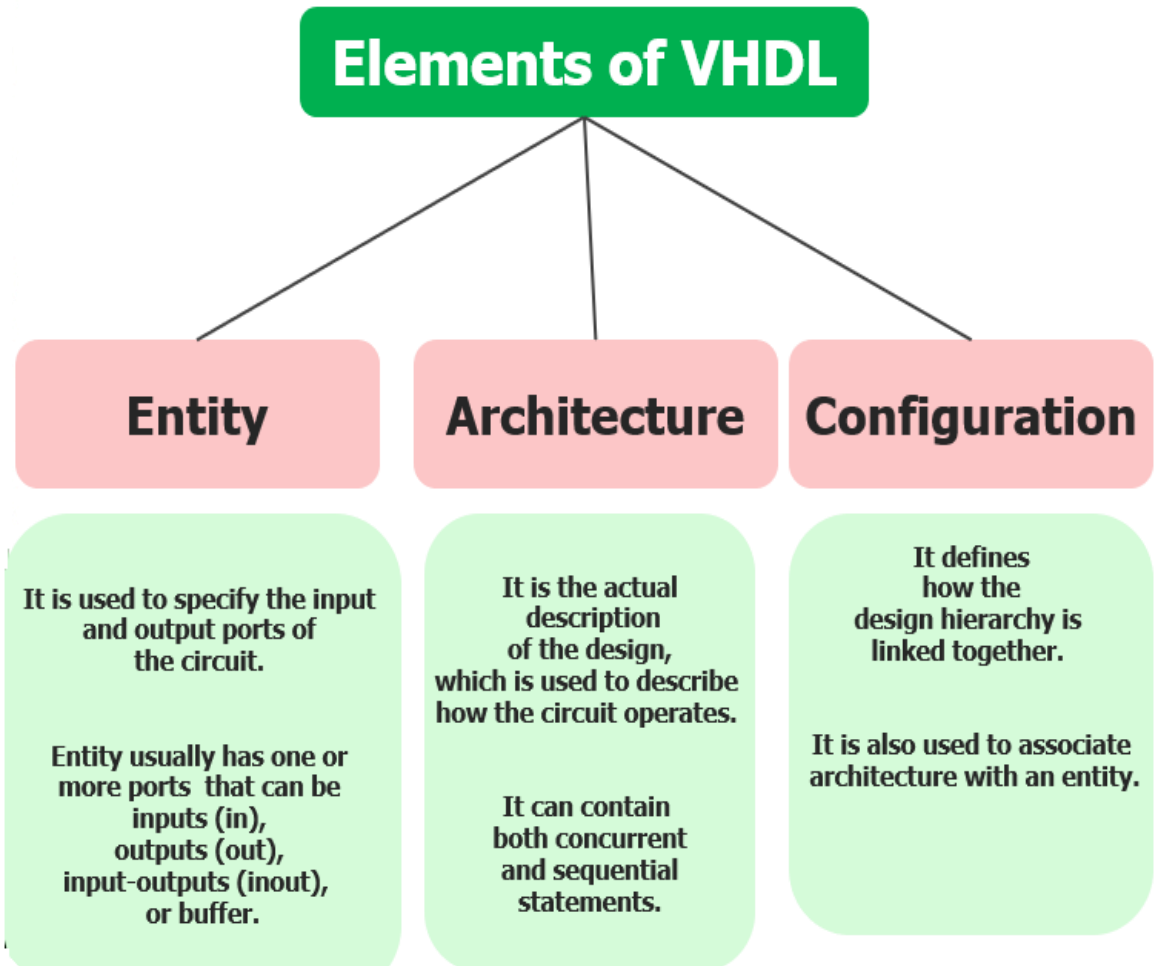
## ✿ Advantages of VHDL:

- ✿ It supports various design methodologies.
- ✿ It provides a flexible design language.
- ✿ It allows better design management.
- ✿ It allows detailed implementations.
- ✿ It supports a multi-level abstraction.
- ✿ It provides tight coupling to lower levels of design.
- ✿ It supports all CAD tools.
- ✿ It strongly supports code reusability and code sharing.

## ✿ Disadvantages of VHDL:

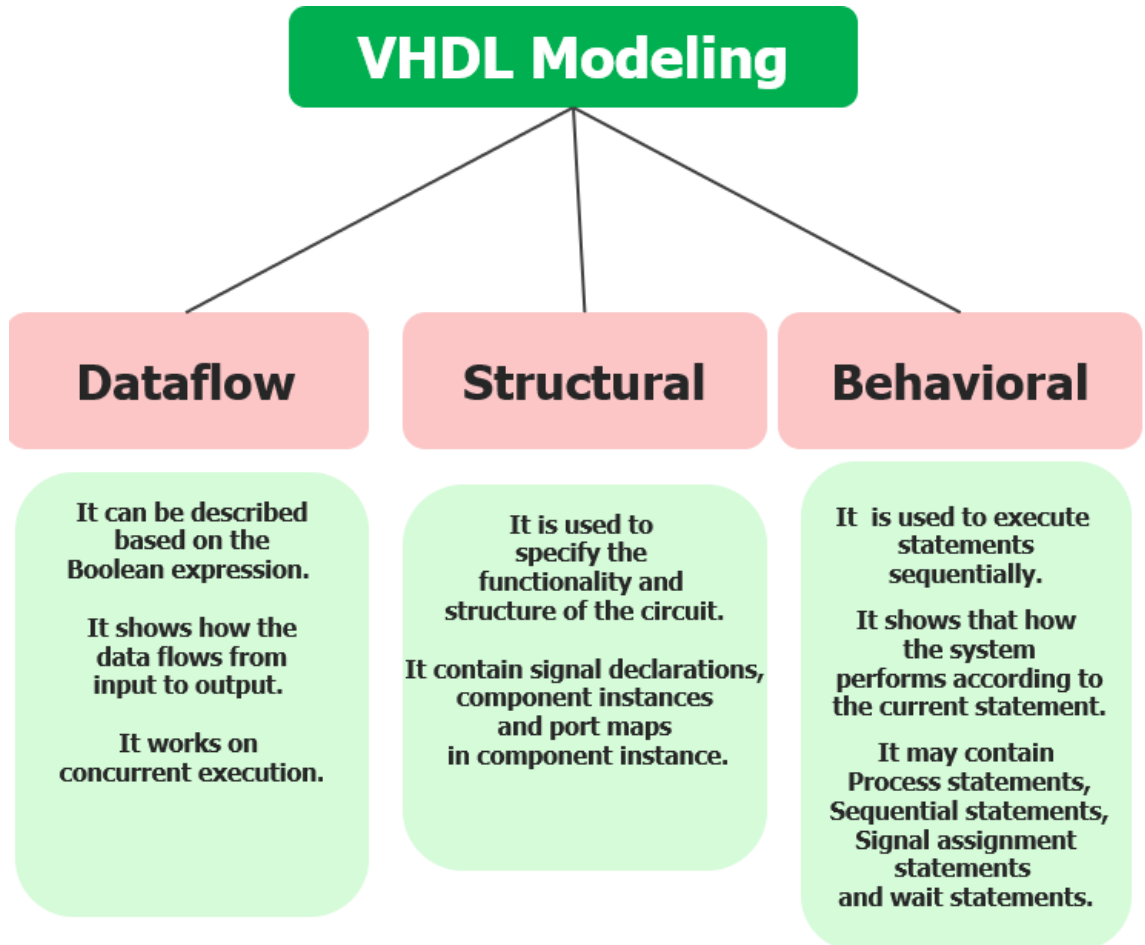
- ✿ It requires specific knowledge of the structure and syntax of the language.
- ✿ It is more difficult to visualize and troubleshoot a design.
- ✿ Some VHDL programs cannot be synthesized.
- ✿ VHDL is more difficult to learn.

## ✿ Basic Elements of VHDL:

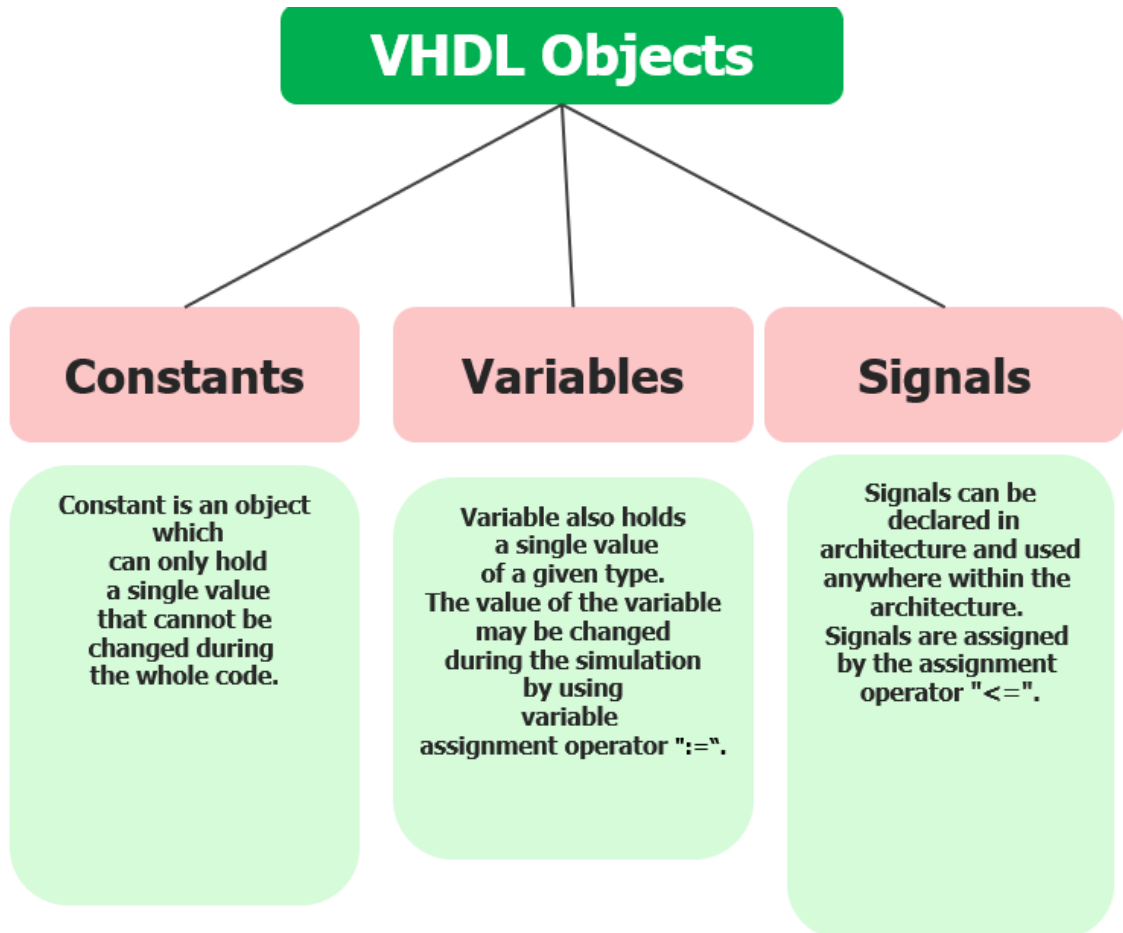




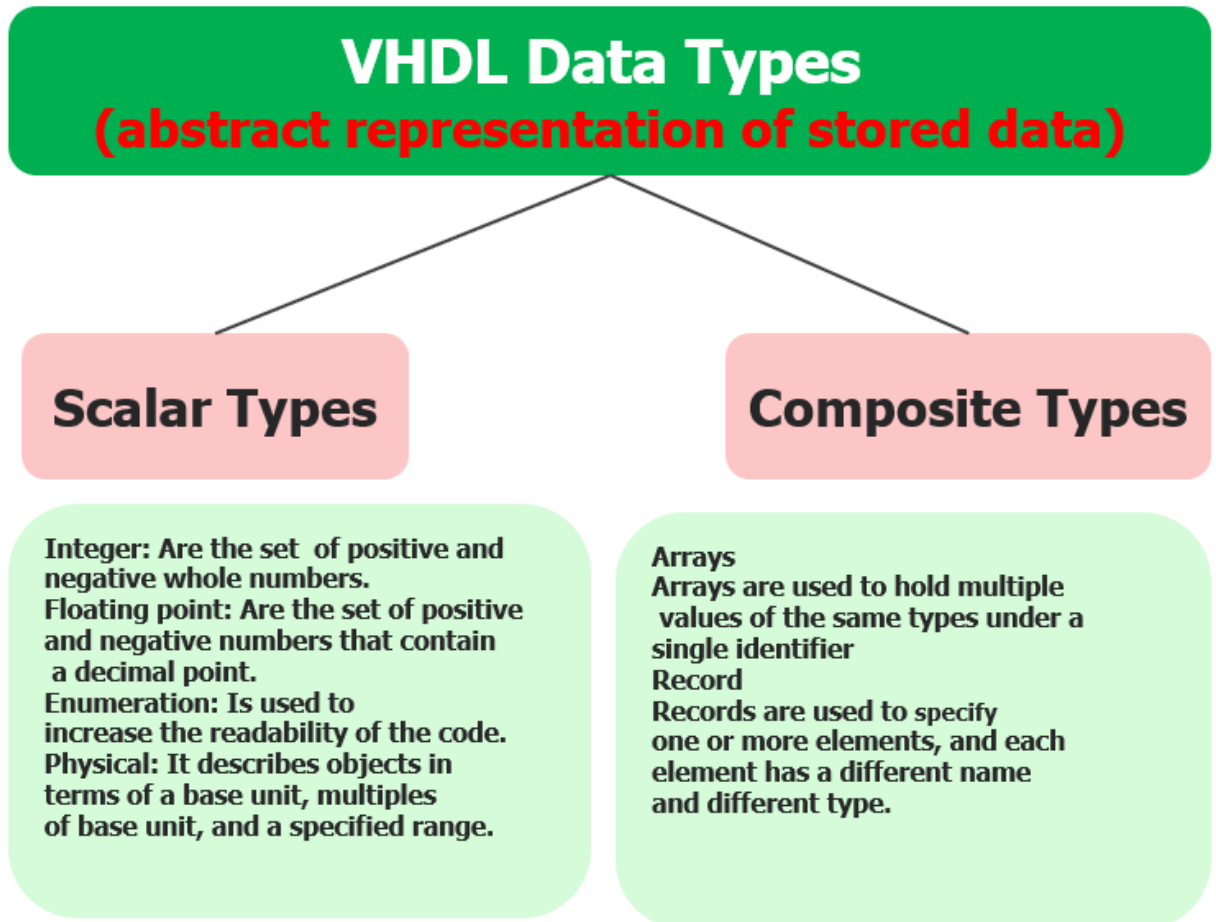
## ✿ Types of VHDL Modeling:



## ✿ Types of VHDL Objects:



## ✿ Types of VHDL Data Types:



## ✿ Types of VHDL Operators:

### VHDL Operators

**Logical**

**Relational**

**Arithmetic**

**Shift**

**Logical**

Logical Operators are used to control the program flow. When the logical operators combined with signals or variables, then it is used to create combinational logic.

**and  
or  
nand  
nor  
xor  
xnor  
not**

## Relational

Relational operators are used to compare two operands of the same data type and the received result is always of the Boolean type.

**= Equal to**  
**/= Not Equal to**  
**< Less than**  
**> Greater than**  
**<= Less than or equal to**  
**>= Greater than or equal to**

## Arithmetic

Arithmetic Operators are used to perform arithmetic operations. These operators are numeric types, such as integer and real.

**+ Addition**  
**- Subtraction**  
**\* Multiplication**  
**/ Division**  
**& Concatenation**  
**mod Modulus**  
**rem Remainder**  
**abs Absolute Value**  
**\*\* Exponentiation**

---

## Shift

shift operator is used to perform the bit manipulation on the data by shifting and rotating the bits of its first operand right or left.

**Sll shift logical left**  
**Srl shift logical right**  
**Sla shift arithmetic left**  
**Sra shift arithmetic right**  
**Rol rotate left**  
**Ror rotate right**

## ❁ **BASIC STRUCTURES IN VHDL:**

❁ Basic building blocks of a VHDL description can be classified into five groups:

- ❁ Entity
- ❁ Architecture
- ❁ Package
- ❁ Configuration
- ❁ Library

❁ A digital system is usually designed as a hierarchical collection modules. Each module corresponds to a design entity in VHDL. Each design entity has two parts:

- ❁ Entity declaration
- ❁ Architecture bodies

❁ An entity declaration describes a component's external interface (input and output ports etc.), whereas architecture bodies describe its internal implementations. Packages define global information that can be used by several entities. A configuration binds component instances of a structure design into entity architecture pairs. It allows a designer to experiment with different variations of a design by selecting different implementations. A VHDL design consists of several library units, each of which is compiled and saved in a design library.

## ❁ ENTITY DECLARATIONS:

- ❁ The entity declaration provides an external view of a component but does not provide information about how a component is implemented. The syntax is ;

```
entity entity_name is
    [generic (generic_declarations);]
    [port (port_declarations);]
    {entity_declarative_item
    {constants, types, signals};}
end [entity_name];
```

[ ] : square bracket denotes optional parameters.

| : vertical bar indicates a choice among alternatives.

{ } : a choice of none, one or more items can be made.

## ❁ GENERIC DECLARATIONS:

- ❁ The generic\_declaration declares constants that can be used to control the structure or behavior of the entity. The syntax is ;

```
generic (
    constant_name : type [:=init_value]
    {;constant_name : type [:=init_value]}
);
```

- ❁ where constant\_name specifies the name of a generic constant, type specifies the data type of the constant, and init\_value specifies an initial value for the constant.



## ❁ PORT DECLARATIONS:

- ❁ The port\_declaration specifies the input and output ports of the entity.

```
port (  
    port_name : [mode] type [:=init_value]  
    {; port_name : [mode] type [:=init_value]}  
);
```

- ❁ where port\_name specifies the name of a port, mode specifies the direction of a port signal, type specifies the data type of a port, and init\_value specifies an initial value for a port.

- ❁ VHDL is not case sensitive, so xyz=xYz=XYZ !!!

- ❁ There are four port modes :

- ❁ • in : can only be read. It is used for input only (can be only on the right side of the assignment).
- ❁ • out : can only be assigned a value. It is used for output only (can be only on the left side of the assignment).
- ❁ • inout : can be read and assigned a value. It can have more than one driver (can be both on the right and left side of the assignment).
- ❁ • buffer : can be read and assigned a value. It can have only one driver (can be both on the right and left side of the assignment).

- ❁ Inout is a bidirectional port whereas buffer is a unidirectional one. The entity\_declarative\_item declares some constants, types or signals that can be used in the implementation of the entity.

## ✿ ARCHITECTURES:

✿ An architecture provides an “internal” view of an entity. An entity may have more than one architecture. It defines the relationships between the inputs and the outputs of a design entity which may be expressed in terms of :

- ✿ Behavioral Style

- ✿ Dataflow Style

- ✿ Structural Style

✿ An architecture determines the function of an entity. It consists of a declaration section where signals, types, constants, components, and subprograms are declared, followed by a collection of concurrent statements.

✿ An architecture is declared using the following syntax :

```
architecture architecture_name of entity_name is
    {architecture_declarative_part
    }
begin
    {concurrent_statement
    }
end [architecture_name] ;
```

## ❁ **BEHAVIORAL STYLE ARCHITECTURES:**

- ❁ A behavioral style specifies what a particular system does in a program like description using processes, but provides no details as to how a design is to be implemented. The primary unit of a behavior description in VHDL is the process.

## ❁ **DATAFLOW STYLE ARCHITECTURES:**

- ❁ A dataflow style specifies a system as a concurrent representation of the flow of control and movement of data. It models the information flow or dataflow behavior, over time, of combinational logic functions such as adders, comparators, decoders, and primitive logic gates.

## ❁ **STRUCTURAL STYLE ARCHITECTURES:**

- ❁ A structural style defines the structural implementation using component declarations and component instantiations.

## ❁ PACKAGES:

- ❁ The primary purpose of a package is to collect elements that can be shared (globally) among two or more design units. It contains some common data types, constants, and subprogram specifications. A package may consist of two separate design units : a package declaration and a package body. A package declaration declares all the names of items that will be seen by the design units that use the package. A package body contains the implementation details of the subprograms declared in the package declaration. A package body is not required if no subprograms are declared in a package declaration. The separation between package declaration and package body serves the same purpose as the separation between the entity declaration and architecture body.

- ❁ The package syntax is :

```
package package_name is
    {package_declarative_item}
end [package_name] ;

package body package_name is
    {package_declarative_item}
end [package_name] ;
```

## ❁ CONFIGURATIONS:

❁ An entity may have several architectures. During the design process, a designer may want to experiment with different variations of a design by selecting different architectures. Configurations can be used to provide fast substitutions of component instances of a structural design.

❁ The syntax is :

```
configuration configuration_name of entity_name is
{configuration_declarative_part
}
for block_specification
{use_clause
}
{configuration_item
}
End for ;
```

## ✿ **DESIGN LIBRARIES:**

✿ The results of a VHDL compilation (analyze) are kept inside of a library for subsequent simulation, for use as a component in other designs. A design library can contain the following library units :

✿ • Packages • Entities • Architectures • Configurations

✿ To open a library to access a compiled entity as a part of a new V H D L design, you first need to declare the library name.

✿ The syntax is :

```
library library_name : [path / directory_name] ;
```

# VHDL – Combinational Circuits – Data Flow Modeling

## ✿ AND Gate

```
Library ieee;
use ieee.std_logic_1164.all;

entity and1 is
    port(x,y:in bit ; z:out bit);
end and1;

architecture working of and1 is
begin
    z<=x and y;
end working;
```

## ✿ OR Gate

```
Library ieee;
use ieee.std_logic_1164.all;

entity or1 is
    port(x,y:in bit ; z:out bit);
end or1;

architecture working of or1 is
begin
    z<=x or y;
end working;
```

## ❁ NOT Gate

```
Library ieee;
use ieee.std_logic_1164.all;

entity not1 is
    port(x:in bit ; y:out bit);
end not1;

architecture working of not1 is
begin
    y<=not x;
end working;
```

## ❁ NAND Gate

```
Library ieee;
use ieee.std_logic_1164.all;

entity nand1 is
    port(a,b:in bit ; c:out bit);
end nand1;

architecture working of nand1 is
begin
    c<=a nand b;
end working;
```



## ❁ **NOR Gate**

```
Library ieee;
use ieee.std_logic_1164.all;

entity nor1 is
    port(a,b:in bit ; c:out bit);
end nor1;

architecture working of nor1 is
begin
    c<=a nor b;
end working;
```

## ❁ **EXOR Gate**

```
Library ieee;
use ieee.std_logic_1164.all;

entity xor1 is
    port(a,b:in bit ; c:out bit);
end xor1;

architecture working of xor1 is
begin
    c<=a xor b;
end working;
```

## ❁ EXNOR Gate

```
Library ieee;
use ieee.std_logic_1164.all;

entity xnor1 is
    port(a,b:in bit ; c:out bit);
end xnor1;

architecture working of xnor1 is
begin
    c<=not(a xor b);
end working;
```

## ❁ HALF ADDER

```
Library ieee;
use ieee.std_logic_1164.all;

entity half_adder is
    port(a,b:in bit; sum,carry:out bit);
end half_adder;

architecture data of half_adder is
begin
    sum<= a xor b;
    carry <= a and b;
end data;
```

## ❁ FULL ADDER

```
Library ieee;

use ieee.std_logic_1164.all;

entity full_adder is port(a,b,c:in bit; sum,carry:out bit);
end full_adder;

architecture data of full_adder is
begin

    sum<= a xor b xor c;

    carry <= ((a and b) or (b and c) or (a and c));

end data;
```

## ❁ HALF SUBTRACTOR

```
Library ieee;

use ieee.std_logic_1164.all;

entity half_sub is

    port(a,c:in bit; d,b:out bit);

end half_sub;

architecture data of half_sub is
begin

    d<= a xor c;

    b<= (a and (not c));

end data;
```

## ⚙️ **FULL SUBTRACTOR**

```
Library ieee;
use ieee.std_logic_1164.all;

entity full_sub is
    port(a,b,c:in bit; sub,borrow:out bit);
end full_sub;

architecture data of full_sub is
begin
    sub<= a xor b xor c;
    borrow <= ((b xor c) and (not a)) or (b and c);
end data;
```

## ⚙️ **MULTIPLEXER (4:1)**

```
Library ieee;
use ieee.std_logic_1164.all;

entity mux is
    port(S1,S0,D0,D1,D2,D3:in bit; Y:out bit);
end mux;

architecture data of mux is
begin
    Y<= (not S0 and not S1 and D0) or
        (S0 and not S1 and D1) or
        (not S0 and S1 and D2) or
        (S0 and S1 and D3);
end data;
```

## ❁ DEMULTIPLEXER (1:4)

```
Library ieee;
use ieee.std_logic_1164.all;

entity demux is
    port(S1,S0,D:in bit; Y0,Y1,Y2,Y3:out bit);
end demux;

architecture data of demux is
begin
    Y0<= ((Not S0) and (Not S1) and D);
    Y1<= ((Not S0) and S1 and D);
    Y2<= (S0 and (Not S1) and D);
    Y3<= (S0 and S1 and D);
end data;
```

## ❁ ENCODER (8X3)

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

entity enc is
    port(i0,i1,i2,i3,i4,i5,i6,i7:in bit; o0,o1,o2: out bit);
end enc;

architecture working of enc is
begin
    o0<=i4 or i5 or i6 or i7;
    o1<=i2 or i3 or i6 or i7;
    o2<=i1 or i3 or i5 or i7;
end working;
```

## ❁ **DECODER (3X8)**

```
library ieee;

use ieee.std_logic_1164.all;

entity dec is
    port(i0,i1,i2:in bit; o0,o1,o2,o3,o4,o5,o6,o7: out bit);
end dec;

architecture working of dec is
begin
    o0<=(not i0) and (not i1) and (not i2);
    o1<=(not i0) and (not i1) and i2;
    o2<=(not i0) and i1 and (not i2);
    o3<=(not i0) and i1 and i2;
    o4<=i0 and (not i1) and (not i2);
    o5<=i0 and (not i1) and i2;
    o6<=i0 and i1 and (not i2);
    o7<=i0 and i1 and i2;
end working;
```

# VHDL – Combinational Circuits – Behavioral Modeling

## ❁ HALF ADDER

```
library ieee;

use ieee.std_logic_1164.all;

entity halfadder2 is
port(a, b : in bit;
      s, c : out bit);
end halfadder2;

architecture behavioral of halfadder2 is
begin
p1: process(a,b)
begin
if a & b = "00" then
s <= '0';
c <= '0';
elsif a & b = "01" or a & b = "10" then
s <= '1';
c <= '0';
else
s <= '0';
c <= '1';
end if;
end process;
end behavioral;
```

## ❁ FULL ADDER

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.STD_LOGIC_ARITH.ALL;  
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

entity fab is

```
    Port ( a : in std_logic;  
          b : in std_logic;  
          c : in std_logic;  
          s : out std_logic;  
          cr : out std_logic);
```

end fab;

architecture Behavioral of fab is

```
begin  
    process(a,b,c)  
    begin  
        if(a='0' and b='0' and c='0')then  
            s<='0';  
            cr<='0';  
        elsif( a='0' and b='0' and c='1')then  
            s<='1';  
            cr<='0';  
        elsif( a='0' and b='1' and c='0')then  
            s<='1';  
            cr<='0';
```



```
    elsif( a='0' and b='1' and c='1')then
        s<='0';
        cr<='1';
    elsif( a='1' and b='0' and c='0')then
        s<='1';
        cr<='0';
        elsif( a='1' and b='0' and c='1')then
            s<='0';
            cr<='1';
            elsif( a='1' and b='1' and c='0')then
                s<='0';
                cr<='1';
            else
                s<='1';
                cr<='1';
            end if;
        end process;
    end Behavioral;
```

## ❁ HALF SUBTRACTOR

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity HS is
    Port (
        A : in STD_LOGIC;
        B : in STD_LOGIC;
        Diff : out STD_LOGIC;
        Borrow : out STD_LOGIC
    );
end HS;

architecture Behavioral of HS is
begin
    process(A,B)
    begin
        if (A = '0' and B='0') THEN
            Diff <= '0';
            Borrow <= '0';
        elsif (A = '0' and B='1') THEN
            Diff <= '1';
            Borrow <= '1';
        elsif (A = '1' and B='0') THEN
            Diff <= '1';
            Borrow <= '0';
        end if;
    end process;
end;
```

```
elsif (A = '1' and B='1') THEN
```

```
Diff <= '0';
```

```
Borrow <= '0';
```

```
else
```

```
Diff <= 'Z';
```

```
Borrow <='Z';
```

```
end if;
```

```
end process;
```

```
end Behavioral;
```

## ❁ FULL SUBTRACTOR

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FULLSUBTRACTOR_BEHAVIORAL_SOURCE is
Port ( A : in  STD_LOGIC_VECTOR (2 downto 0);
      Y : out STD_LOGIC_VECTOR (1 downto 0));
end FULLSUBTRACTOR_BEHAVIORAL_SOURCE;

architecture Behavioral of FULLSUBTRACTOR_BEHAVIORAL_SOURCE is
begin
    process (A)
    begin
        if (A = "001" or A = "010" or A = "111") then
            Y <= "11";
        elsif (A = "011") then
            Y <= "01";
        elsif (A = "100") then
            Y <= "10";
        else
            Y <= "00";
        end if;
    end process;
end Behavioral;
```

## ❁ **DECODER (2X4)**

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

architecture Behavioral of DECODER_SOURCE is
begin
    process (I)
    begin
        case I is
            when "00" => Y <= "0001" ;
            when "01" => Y <= "0010" ;
            when "10" => Y <= "0100" ;
            when others => Y <= "1000" ;
        end case;
    end process;
end Behavioral;
```

## ❁ ENCODER (4X2)

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

architecture Behavioral of ENCODER_SOURCE is
begin
    process (I)
    begin
        if I = "0001" then Y <= "00";
        elsif I = "0010" then Y <= "01";
        elsif I = "0100" then Y <= "10";
        else Y <= "11";
        end if;
    end process;
end Behavioral;
```

# VHDL – Combinational Circuits – Structural Modeling

## ✿ FULL ADDER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fulladd is
    Port ( x : in  STD_LOGIC;
           y : in  STD_LOGIC;
           cin : in  STD_LOGIC;
           sum : out STD_LOGIC;
           cout : out STD_LOGIC);
end fulladd;

architecture Structural of fulladd is
    component or1 is
        Port ( a : in  STD_LOGIC;
              b : in  STD_LOGIC;
              c : out STD_LOGIC);
    end component;

    component xor1 is
        Port ( a : in  STD_LOGIC;
              b : in  STD_LOGIC;
              c : out STD_LOGIC);
    end component;
```

component and1 is

Port ( a : in STD\_LOGIC;

      b : in STD\_LOGIC;

      c : out STD\_LOGIC);

end component;

signal s1, s2, s3:STD\_LOGIC;

begin

g1:xor1 port map(x,y,s1);

g2:xor1 port map(s1, cin, sum);

g3:and1 port map(x, y, s2);

g4:and1 port map(s1, cin, s3);

g5:or1 port map(s2, s3, cout);

end Structural;



## ❁ HALF SUBTRACTOR

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity sub2 is

    Port ( x : in  STD_LOGIC;
          y : in  STD_LOGIC;
          dif : out  STD_LOGIC;
          bor : out  STD_LOGIC);

end sub2;

architecture Structural of sub2 is

    component xorg is

        Port ( a : in  STD_LOGIC;
              b : in  STD_LOGIC;
              c : out  STD_LOGIC);

    end component;

    component andg is

        Port ( a : in  STD_LOGIC;
              b : in  STD_LOGIC;
              c : out  STD_LOGIC);

    end component;

    component notg is

        Port ( a : in  STD_LOGIC;
              c : out  STD_LOGIC);

    end component;
```

```
signal s1:STD_LOGIC;  
begin  
g1: xorg port map(x, y, dif);  
g2: notg port map(x, s1);  
g3: andg port map(s1, y, bor);  
end Structural;
```

## ❁ DECODER

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity deco1 is
    Port ( x : in  STD_LOGIC;
          y : in  STD_LOGIC;
          z : in  STD_LOGIC;
          d0 : out STD_LOGIC;
          d1 : out STD_LOGIC;
          d2 : out STD_LOGIC;
          d3 : out STD_LOGIC;
          d4 : out STD_LOGIC;
          d5 : out STD_LOGIC;
          d6 : out STD_LOGIC;
          d7 : out STD_LOGIC);
end deco1;

architecture Structural of deco1 is
    component decand1 is
        Port ( a : in  STD_LOGIC;
              b : in  STD_LOGIC;
              c : in  STD_LOGIC;
              d : out STD_LOGIC);
    end component;
```

```
component decnot1 is
Port ( a : in  STD_LOGIC;
      c : out STD_LOGIC);
end component;

signal s2, s1, s3: STD_LOGIC;

begin

g1: decnot1 port map(x,s1);
g2: decnot1 port map(y,s2);
g3: decnot1 port map(z,s3);
g4: decand1 port map(s1, s2, s3, d0);
g5: decand1 port map(s1, s2, z, d1);
g6: decand1 port map(s1, y, s3, d2);
g7: decand1 port map(s1, y, z, d3);
g8: decand1 port map(x, s2, s3, d4);
g9: decand1 port map(x, s2, z, d5);
g10: decand1 port map(x, y, s3, d6);
g11: decand1 port map(x, y, z, d7);

end Structural;
```

# VHDL Programs for sequential circuits

## D Flip-Flop (Behavioral approach)

```
❁ Library ieee;

❁ Use ieee.std_logic_1164.all;

❁ Entity dff is
❁   Port(D,CLK,reset:in std_logic;
❁         Q:out std_logic);
❁ End dff;

❁ Architecture dff_arch of dff is
❁   begin
❁     process (CLK)
❁     Begin
❁       If (CLK'event and CLK='1')then
❁       If reset='0' then
❁         Q<='0';
❁       Else
❁         Q<='D';
❁       end if;
❁     end if;
❁   end process;
❁ end;
```

## JK Flip-Flop (Behavioral approach)

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity jkff is
Port(J,K,CLK:in std_logic;
      Q,NQ:inout std_logic);
End jkff;

Architecture jkff_arch of jkff is
begin
    process(CLK)
    Begin
        If(CLK'event and CLK='1')then
            If (j='0' and K='0')then
                Q<=Q;
                NQ=NQ;
            elseif (j='1' and K='0')then
                Q<=1;
                NQ=0;
            elseif (j='0' and K='1')then
                Q<=0;
                NQ=1;
            elseif (j='1' and K='1')then
                Q<=not Q;
                NQ=not NQ;
            end if;
        end if;
    end process;
end;
```

## 4-bit Serial in Serial out Shift Register: (Structural approach)

```
library ieee;  
  
use ieee.std_logic_1164.all;  
  
entity siso is  
  
    port(D:in std_logic;  
  
          reset,CLK:in std_logic);  
  
          Q:out std_logic);
```

```
end siso;
```

```
architecture arch_siso of siso is
```

```
    signal QA,QB,QC,QD:std_logic;
```

```
    component dff1 is
```

```
        port(D,CLK,reset:in std_logic;
```

*– D Flip-Flop program is*

*used as*

```
            Q:out std_logic);
```

*– Component*

```
end component;
```

```
begin
```

```
    a1:diff port map(D,CLK,reset,QA);
```

```
    a2:diff port map(QA,CLK,reset,QB);
```

```
    a3:diff port map(QB,CLK,reset,QC);
```

```
    a4:diff port map(QC,CLK,reset,QD);
```

```
end;
```

## 4-bit Parallel in Shift out Shift Register:

The VHDL program for the *4-bit Parallel in Serial out Shift Register* using *behavioral approach* can be written as follows.

```
library ieee;

use ieee.std_logic_1164 all;

entity dpiso is

port(CLK,load; in std_logic;

d: in std_logic_vector (3 downto 0);

dout: out std_logic);

end dpiso;

architecture arch_dpiso of dpiso is

signal reg: std_logic_vector (3 downto 0);

begin

process (CLK)

begin

if (CLK `event and CLK= `1') then

if (load = `1') then reg <=d;

else reg <= reg (2 downto 0) & `0';

end if;

end if;

end process;

dout <= reg (3);

end;
```



## 4-bit Parallel in Parallel out Shift Register:

The VHDL program for the *4-bit Parallel in Parallel out Shift Register* can be written as follows. This program follows *structural approach*.

```
library ieee;

use ieee.std_logic_1164.all;

entity pipo is
port(D:in std_logic_vector(0 to 3);
reset,CLK:in std_logic;

      Q:out std_logic_vector(0 to 3));

end pipo;

architecture arch_pipo of pipo is
component dff1 is
port(D,CLK,reset:in std_logic;

      Q:out std_logic);
end component;

begin

      a1:diff port map(D(0),CLK,reset,Q(0));  -- Delay FF component
is used at

      a2:diff port map(Q(1),CLK,reset,Q(1));  -- instances a1, a2, a3
      a3:diff port map(Q(2),CLK,reset,Q(2));
      a4:diff port map(Q(3),CLK,reset,Q(3));

end;
```

# Counters

## 4-bit Asynchronous/Ripple Counter:

The VHDL program for the 4-bit *Asynchronous/Ripple Counter* shown in Fig. 5.1 can be written as follows. This program follows *structural approach*.

```
library ieee;

use ieee.std_logic_1164.all;

entity ripple_counter is
    port(Vcc,CLK,reset:in std_logic;

          Q, NQ:input std_logic_vector(0 to 3));

end ripple_counter;

architecture arch_ripple_counter of ripple_counter is

    component jkff1 is
        port (J,K,CLK,reset: in std_logic;

              Q, NQ: inout std_logic);
    end component;

begin

    a:jkff1 port map (Vcc, Vcc,CLK, reset,Q(0), NQ (0));

    b:jkff1 port map (Vcc, Vcc,CLK, reset,Q(1), NQ (1));

    c:jkff1 port map (Vcc, Vcc,CLK, reset,Q(2), NQ (2));

    d:jkff1 port map (Vcc, Vcc,CLK, reset,Q(3), NQ (3));

end;
```

The external clock pulse is applied to the first flip-flop only and, '1' signal (Vcc) is given to JK inputs of all the flip-flops. The output of one flip=flop is connected to clock input of the next flip-flop.

## 4-bit synsynchronous Binary Counter :

The VHDL program for the 4-bit Synchronous Binary Counter shown in Fig. 5.11 can be written as follows. This program follows structural approach.

```
library ieee;

use ieee.std_logic_1164.all;

entity binary counter is
    port (Vcc,CLK:in std_logic;
          Q,NQ:inout std_logic_vector(0 to 3));
end binarycounter;

architecture arch_binarycounter of binarycounter is
    signal X1,Y1:std_logic;

    component jkff1 is
        port (J,K,CLK: in std_logic;
              Q,NQ: inout std_logic);
    end component;

    component andgate is
        port(A,B:in std_logic;
              Y:out std_logic);
    end component;

    component andgates is
        port(A,B,C:in std_logic;
              Y:out std_logic);
    end component;
```

```
begin
```

```
  a: jkff1 port map (Vcc, Vcc, CLK, Q(0),NQ(0));
```

```
  b: jkff1 port map (Q(0),Q(0), CLK, Q(1),NQ(1));
```

```
  c: andgate port map (Q(1),Q(0),X1);
```

```
  d: jkff1 port map (X1,X1,CLK, Q(2),NQ(2));
```

```
  f: andgates port map (Q(0),Q(1),Q(2),Y1);
```

```
  g: jkff1 port map (Y1,Y1,CLK, Q(3),NQ(3));
```

```
end;
```

## PACKAGES

- ✿ A package in VHDL is a collection of functions, procedures, shared variables, constants, files, aliases, types, subtypes, attributes, and components.
- ✿ Packages are most often used to group together all of the code specific to a Library.
- ✿ Packages can have two parts: a **declaration** and a **body**, (body may not be necessarily required).

The declarations section contains the prototypes for the functions and procedures that are defined.

The body section contains the actual implementation of the functions and procedures (similar to a .h file in C).

- ✿ A package body is not required if no subprograms are declared in a package declaration.
- ✿ Packages group functionality that belongs together and also makes reusable.
- ✿ All component definitions are put in a single package file.
- ✿ Constants and types that appear repeatedly throughout the code should be grouped together in a package file.
- ✿ Functions and procedures exist both in the declaration section as well as the body section. The declaration contains the prototype for the function or procedure and the body contains the actual implementation of the code.
- ✿ The package syntax is :

```
package package_name is
{package_declarative_item}
end [package_name] ;

package body package_name is
{package_declarative_item}
end [package_name] ;
```

## -- Package Declaration Section

```
package example_package is
    constant c_PIXELS : integer := 65536;
    type t_FROM_FIFO is record
        wr_full : std_logic;
        rd_empty : std_logic;
    end record t_FROM_FIFO;
    component example_component is
        port (
            i_data : in std_logic;
            o_result : out std_logic);
    end component example_component;
    function Bitwise_AND (
        i_vector : in std_logic_vector(3 downto 0))
        return std_logic;
end package example_package;
```

## -- Package Body Section

```
package body example_package is
    function Bitwise_AND (
        i_vector : in std_logic_vector(3 downto 0)
    )
        return std_logic is
    begin
        return (i_vector (0) and i_vector (1) and i_vector (2) and i_vector (3));
    end;
end package body example_package;
```

## Using Packages

- ✿ In order to use the constants, components, functions, etc created in the package file, it is needed to specify the other code where to look for these things.
- ✿ This is done with the *use* clause. The .all at the end of the use clause tells the file to use everything inside of the package file, as opposed to one particular function for example.

## SUBPROGRAMS

- ✿ Design units may contain a context clause as the initial part. The context clause of a primary unit applies to all of the primary units corresponding secondary units.
- ✿ Architectures and package bodies are the secondary units.
- ✿ Subprograms are not library units and must be inside entities, architectures or packages.

### Subprograms

- ✿ There are two kinds of subprograms: procedures and functions.
- ✿ Both procedures and functions written in VHDL must have a body and may have declarations.
- ✿ Procedures perform sequential computations and return values in global objects or by storing values into formal parameters.
- ✿ Functions perform sequential computations and return a value as the value of the function.
- ✿ Functions do not change their formal parameters.
- ✿ Subprograms may exist as just a procedure body or a function body.
- ✿ Subprograms may also have a procedure declarations or a function declaration.
- ✿ When subprograms are provided in a package, the subprogram declaration is placed in the package declaration and the subprogram body is placed in the package body.

### Procedure Declaration

- ✿ Used to declare the calling interface to a procedure.

```
procedure identifier [ ( formal parameter list ) ] ;  
procedure print_header ;  
procedure build ( A : in constant integer;  
                 B : inout signal bit_vector;  
                 C : out variable real;  
                 D : file ) ;
```

## Procedure Body

✿ Used to define the implementation of the procedure.

```
procedure identifier [ ( formal parameter list ) ] is
    [ declarations]
begin
    sequential statement(s)
end procedure identifier ;

procedure print_header is
    use STD.textio.all;
    variable my_line : line;
begin
    write ( my_line, string("A   B   C"));
    writeline ( output, my_line );
end procedure print_header ;
```

## Function Declaration

Used to declare the calling and return interface to a function.

```
function identifier [ ( formal parameter list ) ] return a_type ;

function random return float ;

function is_even ( A : integer) return boolean ;
```

## Function Body

Used to define the implementation of the function.

```
function identifier [ ( formal parameter list ) ]
    return a_type is
    [ declarations]
```



```
begin
    sequential statement(s)
    return some_value; -- of type a_type
end function identifier ;

function random return float is
    variable X : float;
begin
    -- compute X
    return X;
end function random ;
```

## TEST BENCH

✿ VHDL test bench is a VHDL code to verify the functional correctness of the HDL model.

✿ The main objectives of test bench are:

- Instantiate the design under test (DUT)
- Generate stimulus waveforms for design under test
- Generate reference outputs and compare them with the outputs of design under test
- Automatically provide a pass or fail indication

✿ Test bench is a part of the circuits specification.

✿ Stimulus generation for test bench

The three ways are:

- Generate them “on-the-fly”
- Read vectors stored as constants in an array
- Read vectors stored in a separate system file

Response is produced in the test bench. Response can be stored into file for further processing.

### **Example:**

- Stimulus can be generated with Matlab and test bench feeds it into design under test.
- Design under test (DUT) generates the response and test bench stores it into file.
- Result can be compared to Matlab simulations.

## ⚙ Example

### ⚙ AND GATE CODE

```
use IEEE.std_logic_1164.all;

entity andgate is
Port( A : in std_logic;
      B : in std_logic;
      Y : out std_logic
);
end andgate;

architecture Behavioral of andgate is
begin
Y<= A and B ;
end Behavioral;
```

### ⚙ AND GATE TEST BENCH

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity and_tb is

-- Port ( );

end and_tb;

architecture Behavioral of and_tb is

--Component name and entity's name must be same

--ports must be same

component andgate is

    Port (A,B:in std_logic;

          C: out std_logic );

end component;
```

```
--inputs
signal a: std_logic:= '0';
signal b: std_logic:= '0';
--outputs
signal c : std_logic;
begin
-- uut unit under test
uut: andgate PORT MAP(a=>A,b=>B,c=>C);
```

```
--Stimulus Process
```

```
stim_proc:process
```

```
begin
```

```
wait for 10ns;
```

```
a<='1';
```

```
b<='0';
```

```
wait for 10ns;
```

```
a<='0';
```

```
b<='1';
```

```
wait for 10ns;
```

```
a<='0';
```

```
b<='0';
```

```
wait for 10ns;
```

```
a<='1';
```

```
b<='1';
```

```
wait for 10ns;
```

```
end process;
```

```
end Behavioral;
```

✿ Example : Design all LOGIC GATES in VHDL and verify

### ✿ **CODE**

```
LIBRARY ieee;

USE ieee.std_logic_1164.ALL;

ENTITY logic_gates IS

PORT( a,b: IN std_logic;

      c: OUT std_logic_vector(0 to 6));

END logic_gates;

ARCHITECTURE logic_gates OF logic_gates IS

BEGIN

    c(0)<=a and b;

    c(1)<=a or b;

    c(2)<=not a;

    c(3)<=a nand b;

    c(4)<=a nor b;

    c(5)<=a xor b;

    c(6)<=a xnor b;

END logic_gates;
```

### ✿ **TEST BENCH CODE**

```
LIBRARY ieee;

USE ieee.std_logic_1164.ALL;

ENTITY logic_gates_tb IS

END logic_gates_tb;
```

ARCHITECTURE logic\_gates\_tb OF logic\_gates\_tb IS

COMPONENT logic\_gates

PORT(

a : IN std\_logic;

b : IN std\_logic;

c : OUT std\_logic\_vector(0 to 6)

);

END COMPONENT;

signal a : std\_logic := '0';

signal b : std\_logic := '0';

signal c : std\_logic\_vector(0 to 6);

BEGIN

uut: logic\_gates PORT MAP (

a => a,

b => b,

c => c

);

```
stim_proc: process
```

```
begin
```

```
a <= '0';
```

```
b <= '0';
```

```
    wait for 50 ns;
```

```
a <= '0';
```

```
b <= '1';
```

```
    wait for 50 ns;
```

```
a <= '1';
```

```
b <= '0';
```

```
    wait for 50 ns;
```

```
a <= '1';
```

```
b <= '1';
```

```
    wait;
```

```
end process;
```

```
END;
```

# SIMULATION AND TUTORIALS

## ADDER

### VHDL CODE:

```
-- function of adder:  
-- A plus B to get n-bit sum and 1 bit carry  
-- we may use generic statement to set the parameter  
-- n of the adder.
```

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

```
entity ADDER is
```

```
generic(n: natural :=2);  
port(  A:    in std_logic_vector(n-1 downto 0);  
      B:    in std_logic_vector(n-1 downto 0);  
      carry: out std_logic;  
      sum:   out std_logic_vector(n-1 downto 0)  
);
```

```
end ADDER;
```

```
architecture behv of ADDER is
```

```
-- define a temporary signal to store the result
```

```
signal result: std_logic_vector(n downto 0);
```

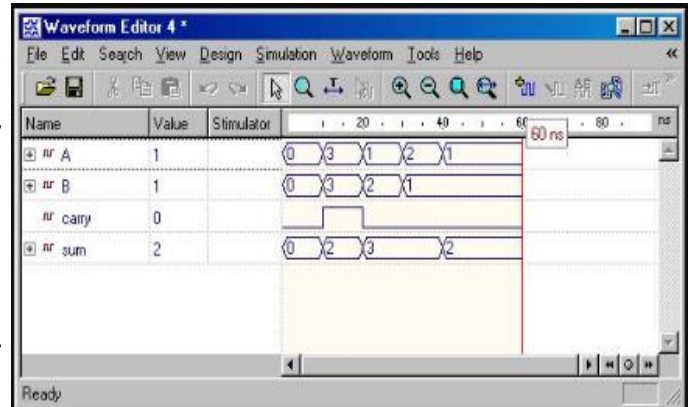
```
begin
```

```
-- the 3rd bit should be carry
```

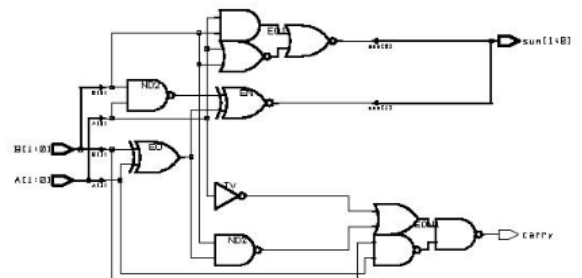
```
result <= ('0' & A)+('0' & B);  
sum <= result(n-1 downto 0);  
carry <= result(n);
```

```
end behv;
```

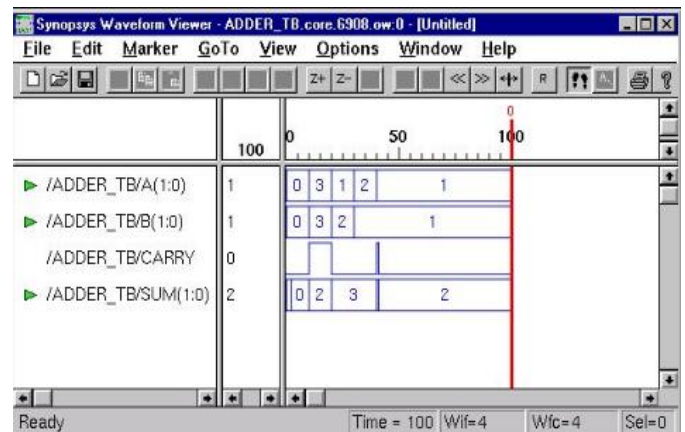
### Behaviour Simulation



### Synthesis Schematic



### Gate - Level Simulation



Ref: <http://esd.cs.ucr.edu/labs/tutorial/>



## 9. Lecture Notes

### ✿ E-Books

Digital Fundamentals\_ Global Ed - Thomas L Floyd

Digital Principles And Application - Leach & Malvino

Digital Design - M. Morris Mano and Michael D. Ciletti

Fundamentals of Digital Logic with Verilog Design-Stephen Brown and Zonko Vranesic

### ✿ ONLINE LEARNING MATERIALS:

1. <https://nptel.ac.in/courses/117/108/117108040/>
2. <https://nptel.ac.in/courses/106/102/106102181/>
- 3: <http://esd.cs.ucr.edu/labs/tutorial/>
4. <https://www.ics.uci.edu/~alexv/154/VHDL-Cookbook.pdf>
- 5: <https://www.csee.umbc.edu/portal/help/VHDL/VHDL-Handbook.pdf>

### ✿ Video Links

<https://www.youtube.com/watch?v=mwJ3uMWvJX0>

[https://www.youtube.com/watch?v=IZDgIg6cllw&feature=emb\\_logo](https://www.youtube.com/watch?v=IZDgIg6cllw&feature=emb_logo)

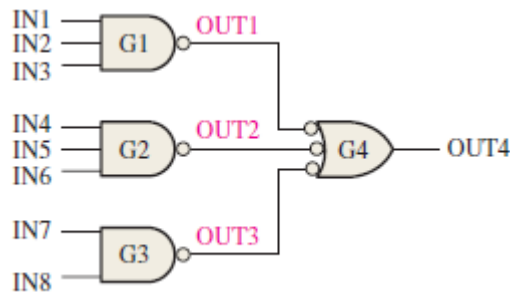
## 10. Assignments

- ✿ Design a system at the Register Transfer Level (RTL) that is to count the number of 1's in a word of data, in a register.

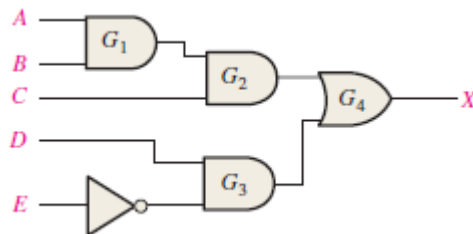
- ✿ Describe the 7-segment decoding logic using VHDL for implementation in a programmable logic device (PLD). The logic expressions for segments *a*, *b*, and *c* of the display are as follows:

$$a = H0 ; \quad b = H1H0 + H1H0 + H2H1 ; \quad c = H1H0 + H2H1$$

- ✿ Write a VHDL program for the SOP logic circuit below using the structural approach and compare with the data flow approach. Assume that VHDL component for a 3-input NAND gate and for a 2-input NAND are available.



- ✿ Develop a VHDL program for the logic circuit below, using both the data flow and the structural approach. Compare the resulting programs.



- ✿ A

## 11. Part A Q & A (with K level and CO)

PART A		
Questions and Answers	Blooms Level	COs
<b>1. What is VHDL?</b> <b>Vey high speed integrated circuit Hardware Description Language. It is a language for describing a hardware, which has to be readable for machines and humans at the same time.</b>	K1	CO6
<b>2. What are the VHDL structural elements?</b> <b>The main units in VHDL are Entity , Architecture , Configuration, Process , Package , Library</b>	K1	CO6
<b>3. Write VHDL program for an and gate</b> <b>entity andgate is</b> <b>Port ( a : in std_logic;</b> <b>b : in std_logic;</b> <b>c : out std_logic); end andgate;</b> <b>architecture dataflow of andgate is begin</b> <b>c&lt;= a and b; end andgate;</b>	K2	CO6
<b>4. List the operators in VHDL</b> <b>Logical operators</b> <b>Relational operators</b> <b>Shift operators</b> <b>Adding operators</b> <b>Multiply operators</b> <b>Miscellaneous operators</b>	K1	CO6
<b>5. What is test bench?</b> <b>A test bench is a model which is used to exercise and verify the correctness of a hardware model.</b>	K1	CO6
<b>6.What are packages?</b> <b>A package is used to provide a convenient method to store and share declarations that are common for many design units.</b>	K1	CO6
<b>7. What are the types of subprograms?</b> <b>Functions</b> <b>Procedure</b>	K2	CO6

<b>8. Define modularity.</b> It allows partitioning of big functional blocks into smaller units and to group closely related parts in self- contained sub blocks, so called module.	K1	CO6
<b>9. What are the data types available in VHDL?</b> Scalar type Composite type Access type File type	K1	CO6
<b>10. What is data flow modeling in VHDL?</b> A data flow model specifies the functionality of the entity without explicitly specifying its structure.	K1	CO6
<b>11. What is package declaration?</b> It contains a set of declarations that many be shared by various design units. It defines items which are made visible to other design units.	K1	CO6
<b>12. What is a Procedure?</b> These are used to partition large behavioral descriptions. Procedures can return zero or more values. It may or may not execute in zero simulation time. It depends whether the wait statement is used or not.	K1	CO6
<b>13. What is entity?</b> The interface between a module and its environment is described within the entity declaration which is initiated by the keyword entity.	K1	CO6
<b>14. What is a Function?</b> These are used for computing a single value. It executes in zero simulation time.	K1	CO6
<b>15. What are the RTL description processes?</b> The pure combinational process and the clocked process . All clocked processes infer flip-flops and can be described in terms of state machine syntax.	K1	CO6

## 12. Part B Qs (with K level and CO)

PART B		
1. Explain RTL design with an example	K3	CO6
2. Write VHDL program for full adder using data flow and structural approach	K3	CO6
3. What is test bench? Explain with an example	K3	CO6
4. Write VHDL program for 4x1 multiplexer using structural approach	K3	CO6
5. Write VHDL program for 4 bit ripple counter	K3	CO6
6. Write VHDL program for 3 bit synchronous down counter	K3	CO6
7. Write VHDL program for JK flip flop and T flip flop	K3	CO6
8. Write the VHDL code to realize a 2:1 multiplexer using data flow modeling	K3	CO6
9. Write the VHDL code for a logical gate which gives high output only when both the inputs are high	K3	CO6
10. Write the test bench for a FSM used to detect the sequence 1101	K3	CO6
11. Explain the operators and data types in VHDL	K3	CO6
12. Write short notes on package and subprograms	K3	CO6
13. Briefly explain simulation and synthesis	K3	CO6
14. Explain the different timing control available in VHDL with suitable examples	K3	CO6
15. Write VHDL program for a 1:8 de-multiplexer in all three modelling methods.	K3	CO6

### 13.Supportive online Certification courses (**NPTEL, Swayam, Coursera, Udemy, etc.,**)

#### ✿ Swayam

Digital Circuits: [https://swayam.gov.in/nd1\\_noc20\\_ee70/preview](https://swayam.gov.in/nd1_noc20_ee70/preview)

Duration: 12 weeks, Start Date: 14 Sep 2020, End Date: 04 Dec 2020

#### ✿ Udemy

Digital Electric Circuits & Intelligent Electrical Devices

<https://www.udemy.com/course/digital-electric-circuits-intelligent-electrical-devices/>

#### ✿ Coursera

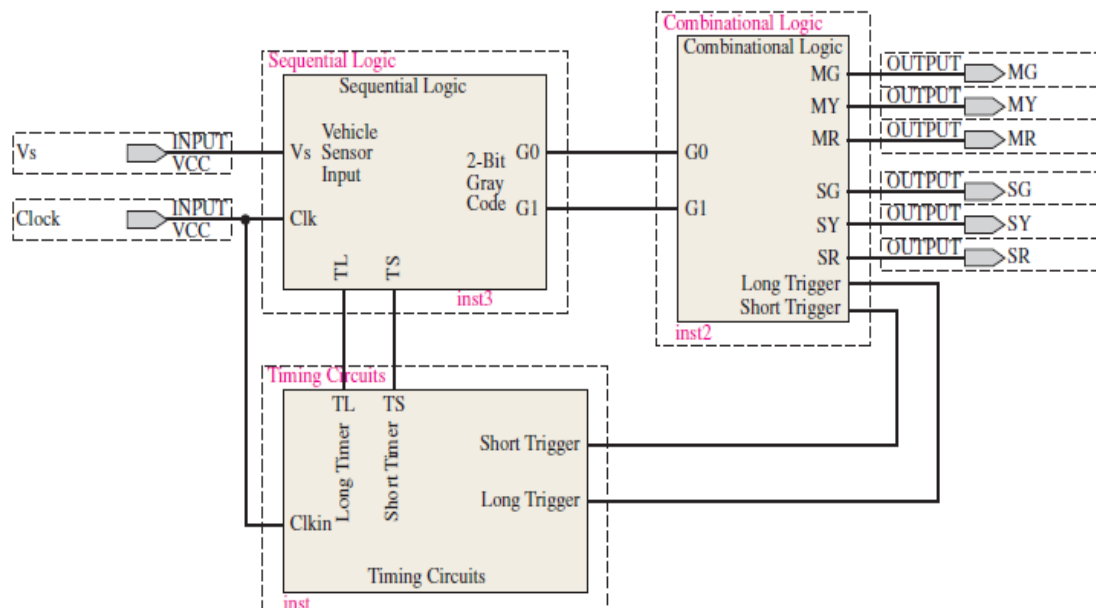
Build a Modern Computer from First Principles: From Nand to Tetris (Project-Centered Course)

<https://www.coursera.org/learn/build-a-computer>

# 14. Real time Applications in day to day life and to Industry

## TRAFFIC SIGNAL CONTROLLER

- A traffic signal controller is represented with its combinational logic and sequential



- The use of VHDL to create the sequential logic component of the system

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity SequentialLogic is
5 port(VS, TL, TS, Clk: in std_logic;
6      G0, G1: inout std_logic);
7 end entity SequentialLogic;
8
9 architecture SequenceBehavior of SequentialLogic is
10 component dff is
11 port(D, Clk: in std_logic;
12      Q: out std_logic);
13 end component dff;
14
15 signal D0, D1: std_logic;
16 begin
17     D1 <- (G0 and not TS) or
18          (G1 and TS);
19
20     D0 <- (not G1 and not TL and VS) or
21          (not G1 and G0) or
22          (G0 and TL and VS);
23
24     DFF0: dff port map(D => D0, Clk => Clk, Q => G0);
25     DFF1: dff port map(D => D1, Clk => Clk, Q => G1);
26 end architecture SequenceBehavior;
```

## 15. Contents beyond the Syllabus ( COE related Value added courses)

### ✿ **VHDL Coding for FPGAs – Contents for Embedded Systems CoE.**

#### DIGITAL LOGIC DESIGN

- ✿ DESIGN FLOW
- ✿ DATA TYPES
- ✿ LOGIC GATES IN VHDL
- ✿ TESTBENCH GENERATION
- ✿ XILINX: I/O ASSIGNMENT
- ✿ USE OF std\_logic\_vector

#### DESIGN FLOW

- ✿ Design Entry: The logic circuit is specified using a Hardware Description Language (e.g., VHDL, Verilog).
- ✿ Functional Simulation: Also called behavioural simulation. Here, we will only verify the logical operation of the circuit. Stimuli is provided to the logic circuit, so we can verify the outputs behave as we expect.
- ✿ Physical Mapping: The inputs/outputs of our logic circuit are mapped to specific pins of the FPGA.
- ✿ Timing Simulation: It simulates the circuit considering its timing behavior (delays between inputs and outputs)
- ✿ Implementation: A configuration file ('bitstream' file) is generated and then downloaded onto the FPGA
- ✿ <https://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html>



## 17. Prescribed Text Books & Reference Books

### ✿ TEXT BOOKS:

1. James W. Bignel, Digital Electronics, Cengage learning, 5th Edition, 2007.
2. M. Morris Mano, 'Digital Design with an introduction to the VHDL', Pearson Education, 2013.
3. Comer "Digital Logic & State Machine Design, Oxford, 2012.

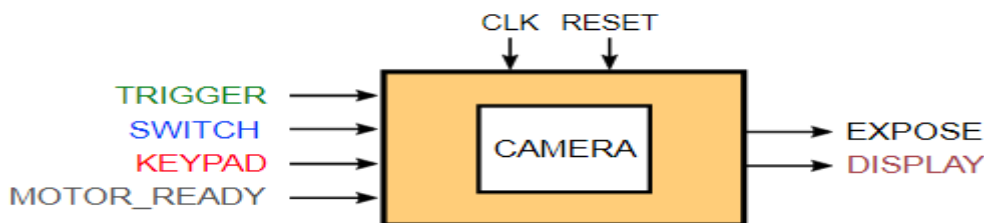
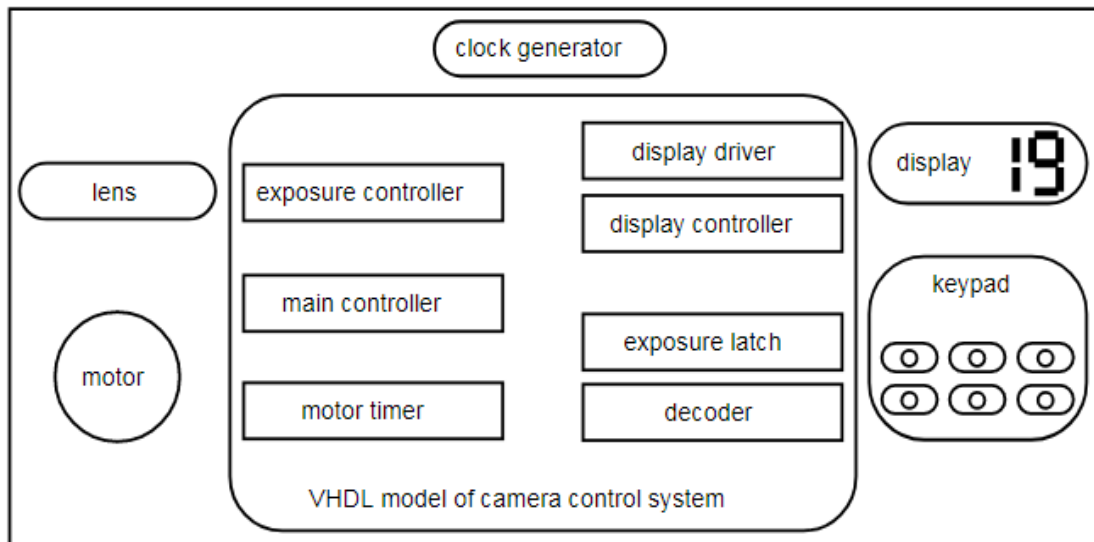
### ✿ REFERENCES

1. Mandal, "Digital Electronics Principles & Application, McGraw Hill Edu, 2013.
2. William Keitz, Digital Electronics-A Practical Approach with VHDL, Pearson, 2013.
3. Thomas L. Floyd, 'Digital Fundamentals', 11th edition, Pearson Education, 2015.
4. Charles H. Roth, Jr, Lizy Lizy Kurian John, 'Digital System Design using VHDL, Cengage, 2013.

## 18. MINI PROJECT

### Design of a camera control system

- ✿ The camera control system has to generate the appropriate signals for the lens shutter and the film transportation motor. It also generates the input signals for a small 7-segment display showing either the current exposure time or the total number of pictures taken so far. The user controls the camera operation via several keys.



- ✿ The central control system that is to be developed in this course is split into seven modules:
- ✿ A decoder (**decoder**) is used to pre process the signals from the input device for the controller.
- ✿ The exposure latch (**exp\_ff**) stores the selected exposure time.
- ✿ A timer for the film transport (**motor\_timer**) is used to detect errors during film transportation, e.g. if the film is torn apart.
- ✿ The exposure controller (**exp\_ctrl**) opens the lens shutter for the selected exposure time and counts the number pictures that have been taken.
- ✿ The photo controller (**main\_ctrl**) is the central control unit for the camera control system.
- ✿ A specialized display controller (**disp\_ctrl**) selects the appropriate data for the 7-segment display.
- ✿ The display driver (**disp\_drv**), finally, converts the internal data signals into a format suitable for the output device.
- ✿ [https://www.vhdl-online.de/vhdl\\_workshop/start](https://www.vhdl-online.de/vhdl_workshop/start)