DIGITAL ELECTRONICS

COURSE MATERIAL

B.TECH

(I YEAR – II SEM)

(2020-21)

Prepared by:

Mrs. Parveen MD, Assistant Professor

Department of Electrical and Electronics Engineering

MALLA REDDY ENGINEERING COLLEGE

(Autonomous)

2020-21 Onwards (MR-20)	MALLA REDDY ENGINEERING COLLEGE (Autonomous)	B.Tech. II Semester		
Code: A0401	DIGITAL ELECTRONICS	L	T	P
Credits: 3		2	1	-

Prerequisites: Nil

Course Objectives:

This course introduces various number systems and conversion from one number system to other and also to understand different binary codes, the theory of Boolean algebra and to study representation of switching functions using Boolean expressions and their minimization techniques. Understanding the combinational logic design of various logic and switching devices and their realization, the basic flip flops and sequential logic circuits design both in synchronous and Asynchronous modes for various complex logic and switching devices, their minimization techniques and their realizations and to analyze a given sequential circuit by using state tables and state diagrams.

MODULE-I: Number systems& Binary codes **8 Periods**

Number systems: Number Systems, Radix conversions, complement of numbers.

Binary codes: Binary codes, Weighted and non-Weighted codes, BCD code, gray code, excess 3 codes - Error detecting code, Error Correcting code, Hamming Code.

MODULE-II: Boolean Algebra&Boolean functions **10 Periods**

Boolean Algebra: Postulates and Theorems - Canonical and Standard forms: SOP and POS forms, Minterms and Maxterms –Logic gates: NOT, OR, AND, NOR, NAND, XOR, XNOR - Universal gates

Simplification of Boolean functions: Simplification of functions: Karnaughmap (2,3,4,5,6 Variables) and Quine McCluskey method (Tabular Method) - Prime implicants, essential prime implicants.

MODULE-III: Combinational Logic Circuits **10 Periods**

A:Arithmetic circuits: Half adder, full adder, half subtractor, full subtractor, binary adder, Carry look ahead adder, BCD adder

B:Code conversion circuits, Comparator, Decoder, Encoder, Priority Encoder, Multiplexers and Design, De – Multiplexers, ROM, PLA, PAL.

MODULE-IV: Sequential Logic Circuits - I **10 Periods**

Introduction –Latches and Flip flops: Basic Flip flop circuit, RS, D, JK and T Flip-flops – Triggering of Flip flops: Master Slave Flip flop, edge triggered flip flop – Conversion of one type of Flip flop to another,Setup time, hold time.

Registers and Counters: Shift Register, Universal Shift Register, Applications of Registers,Asynchronous counter, Synchronous counter, Mod-N Counter, binary up/down counter, Ripple counter, Johnson counter.

MODULE-V: Sequential Logic Circuits - II **10 Periods**

Analysis of Sequential Logic circuit: State Diagram, state table, reduction of state table, state Assignment – Design procedure of sequential circuits using state diagram, state table and Flip flops. Example design Sequence detector.

Finite State Machine: Introduction, FSM capabilities and Limitations, Mealy and Moore models – minimization of completely specified and incompletely specified sequential Machines. Partition techniques and Merger charts

Text Books

1. ZviKohavi, **"Switching and Finite Automata Theory"**,TMH, 2nd edition, 2006.
2. Morris Mano,**"Digital Design"**,PHI, 3rd Edition, 2009.
3. A.Anand Kumar,**"Switching Theory and Logic Design"**,PHI 2nd Edition, 2014.
4. John F.Wakerly, **"Digital Design Principles & Practices"**, PHI/ Pearson Education Asia, 3rd Ed., 2005.

References

1. Stephen Brown and ZvonkaVramesic, **"Fundamentals of Digital Logic with VHDL Design"**,McGraw Hill, 2nd Edition, 2008.
2. William I. Fletcher, **"An Engineering Approach to Digital Design"**, PHI, 1st Edition, 2009.

E-Resources:

1. https://www.researchgate.net/publication/264005171_Digital_Electronics
2. https://www.cl.cam.ac.uk/teaching/0708/DigElec/Digital_Electronics_pdf.pdf
3. http://ieeexplore.ieee.org/abstract/document/753678/
4. http://docshare01.docshare.tips/files/20257/202573063.pdf
5. http://nptel.ac.in/courses/117106086/1
6. http://nptel.ac.in/courses/117105080/
7. http://nptel.ac.in/courses/117106114/

Course Outcomes:

At the end of the course, students will be able to

1. Perform radix conversions
2. Minimize a given boolean function by using k-map or tabular method
3. Design a combinational circuit
4. Design a sequential circuit by using various flipflops
5. Analyze and minimize the circuitry of a given sequential circuit and will be able to design a sequence detector

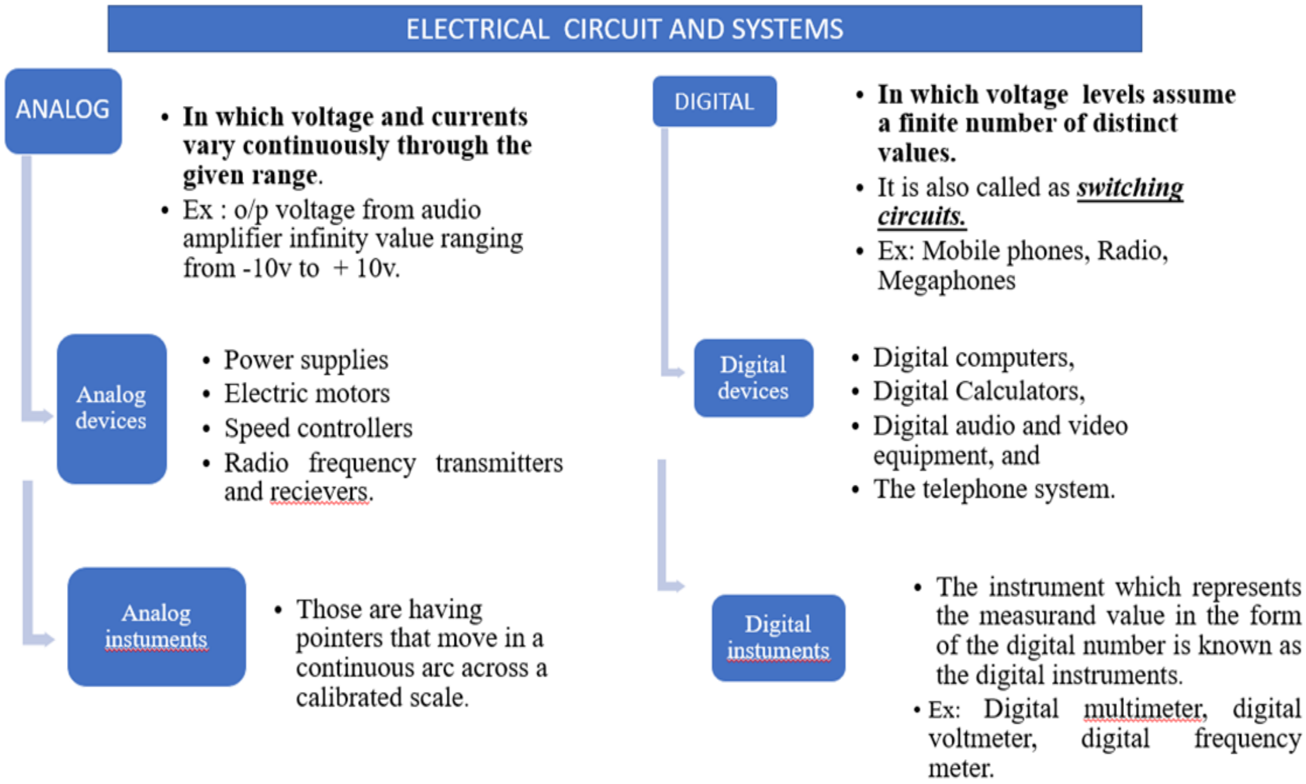
COS	Programme Outcomes(POs)														
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	3	3	3								3	1		1
CO2	3	3	3	3								3	1		1
CO3	3	3	3	3								3	1		1
CO4	3	3	3	3								3	1		1
CO5	3	3	3	3								3	1		1

MODULE 1

Number systems& Binary codes:

- Number systems: Number Systems, Radix conversions, complement of numbers.
- Binary codes: Binary codes, Weighted and non-Weighted codes, BCD code, gray code, excess 3codes - Error detecting code, Error Correcting code, Hamming Code

INTRODUCTION



DIGITAL CIRCUIT:

- Digital circuit is one in which the voltage levels assume a finite number of distinct values.
- Each voltage level in a practical digital system can actually be a narrow band or range of voltages.
- Also called as switching circuits, the voltage levels in a digital circuit are assumed to be switched from one value to another value instantaneously, that is the transition time is assumed to be zero.
  - A) COMBINATIONAL SWITCHING CIRCUITS:
    - The output depends only on the present inputs.
    - They have no memory.
  - B) SEQUENTIAL SWITCHING CIRCUITS
    - The output depends on the present inputs as well as the present state of the circuit, i.e., on the past values also.
    - These are combinational circuits with memory.

❖ SEQUENTIAL SWITCHING CIRCUITS:

a) SYNCHRONOUS SEQUENTIAL CIRCUITS.: Digital sequential circuits in which the feedback to the input for next output generation is governed by clock signals.

b) ASYNCHRONOUS SEQUENTIAL CIRCUITS: Digital sequential circuits in which the feedback to the input for next output generation is not governed by clock signals.

DIGITAL CIRCUIT is also called as Binary signals or Logic signals.

- The digital signals are represented by two voltage bands, one band which is near a reference value (generally 0), and the other band lies near the supply voltage.

- This is similar to the values, '0' and '1' or 'false' and 'true' of the Boolean domain.
- This means that at any particular time, a digital signal can represent only one binary digit.
- The manner in which a logic circuit responds to an input as referred to as the circuit logic.

Application:

- Thermometer, photocopies, landline telephones, audiotape recorders, television, computers, laptops, mobile phones, wristwatches, wall clocks, are all becoming digital nowadays.
- It increases the accuracy of the message as well as makes it easy to read.

Advantages of Digital system or signals:

- Because of the digital nature, the signals in the digital systems can travel significantly faster over digital lines as compared to the Analog signals
- As compared to Analog signals, digital signals can transfer more data.
- The digital systems are less expensive, more reliable, easy to manipulate, and more flexible as compared to the Analog system.
- A digital system can be made compatible with other digital systems to which is not possible in the Analog system.

1. THE DECIMAL SYSTEM

The decimal number system comprises digits from 0-9 that are 0, 1, 2, 3, 4, 5, 6, 7, 8 & 9. The base or radix of the decimal number system is 10 because the total number of digits available in the decimal number system is 10. All the other digits can be expressed with the help of these 10 digit numbers.

number **345** represents:

$$\begin{aligned} &= 3 * 10^2 + 4 * 10^1 + 5 * 10^0 \\ &= 3 * 100 + 4 * 10 + 5 \\ &= 300 + 40 + 5 \\ &= 34 \end{aligned}$$

the value **123.456** means:

$$\begin{aligned} &= 1 * 10^2 + 2 * 10^1 + 3 * 10^0 + 4 * 10^{-1} + 5 * 10^{-2} + 6 * 10^{-3} \\ &= 100 + 20 + 3 + 0.4 + 0.05 + 0.006 \end{aligned}$$

2. THE BINARY SYSTEM

Binary number system can be said to be the simplest one in the number system. It uses only two digits (0 and 1) to represent a number. Thus, as the 'bi' in its name suggests, the system uses 2 as a base. The entire number system can be represented through the binary system. For example, fractions, real numbers, as well as large numbers, can be represented through binary numbers.

BINARY TO DECIMAL CONVERSION

The binary numbering system works just like the decimal numbering system, with two exceptions:

- binary only allows the digits 0 and 1 (rather than 0–9), and
- binary uses powers of two rather than powers of ten.

Therefore, it is very easy to convert a binary number to decimal. For each "1" in the binary string, add 2n where "n" is the bit position in the binary string (0 to n–1 for n bit binary string).

For example, the binary value 1010_2 represents the decimal 10 which can be obtained through the procedure shown in the table 1:

Table 1

Binary No.	1	0	1	0
Bit Position (n)	3rd	2nd	1st	0th
Weight Factor (2n)	2^3	2^2	2^1	2^0
bit * 2n	$1*2^3$	$0*2^2$	$1*2^1$	$0*2^0$
Decimal Value	8	0	2	0

Decimal Number 8 + 0 + 2 + 0 = 10

All the steps in above procedure can be summarized in short as

$1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 = 8 + 0 + 2 + 0 = 1010$

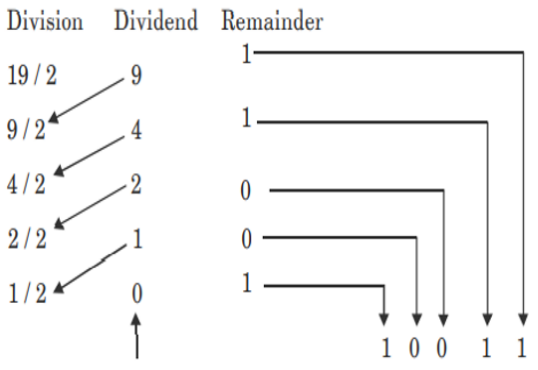
i.e.,

1. Multiply each digit of the binary number by its positional weight and then add up the result.

2. If any digit is 0, its positional weight is not to be taken into account.

DECIMAL TO BINARY CONVERSION

let us find out binary of 19_{10} (decimal 19).



Dividend is 0, stop the procedure.

1. The right most bit in a binary number is bit position zero.
2. Each bit to the left is given the next successive bit number.

- An eight-bit binary value uses bits zero through seven:
- X7 X6 X5 X4 X3 X2 X1 X0
- A 16-bit binary value uses bit positions zero through fifteen:

X15 X14 X13 X12 X11 X10 X9 X8 X7 X6 X5 X4 X3 X2 X1 X0

- Bit zero is usually referred to as the ***low order bit.***

or

Least significant bit (LSB).

- The left-most bit is typically called the ***high order bit.***

or

Most significant bit (msb)

3. OCTAL NUMBERING SYSTEM:

- The octal number system uses base 8 instead of base 10 or base 2.
- This is sometimes convenient since many computer operations are based on bytes (8 bits). In octal, we have 8 digits at our disposal, 0–7.

DECIMAL OCTAL

0 0

1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	10
9	11
10	12
11	13
12	14
13	15
14	16
15	17
16	20

Octal to Decimal,

Converting octal to decimal is just like converting binary to decimal, except instead of powers of 2, we use powers of 8.

To convert 172 in octal to decimal:

$$\begin{array}{lcl} & & 1 \quad 7 \quad 2 \\ & & 8^2 \quad 8^1 \quad 8^0 \\ \text{Weight} & = & 1*8^2 + 7*8^1 + 2*8^0 \\ & = & 1*64 + 7*8 + 2*1 \\ & = & 122_{10} \end{array}$$

Decimal to Octal Conversion,

Converting decimal to octal is just like converting decimal to binary, except instead of dividing by 2, we divide by 8.

To convert 122 to octal:

$$\begin{array}{rcl} 122/8 & = & 15 \text{ remainder } 2 \\ 15/8 & = & 1 \text{ remainder } 7 \\ 1/8 & = & 0 \text{ remainder } 1 \\ & = & 172_8 \end{array}$$

Octal to binary

Convert $(145056)_8$ to binary.

To convert from octal to binary and vice versa we will need this conversion table.

value $(145056)_8$ can be converted to binary as $(001\ 100\ 101\ .\ 101\ 110)_2$

OCTAL SYMBOL	BINARY CODE
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Binary to Octal:

We can use the same table to convert a binary number to octal number. And for that, we first have to group the binary number into a group of three bits and write the octal equivalent of it.

Convert the binary number $(11001111)_2$ to octal
The three bit group of binary numbers can be written as 011,001,111 because we have to add a zero before each number to complete the in the form of three binary digits. Therefore, the octal numbers will be 3, 1, 7 i.e., $(317)_8$

3. HEXADECIMAL NUMBERING SYSTEM

- Hexadecimal uses a base 16 numbering system. This means that we have 16 symbols to use for digits. Consequently, we must invent new digits beyond 9.
- The digits used in hex are the letters A, B, C, D, E, and F.

Hexa decimal to Decimal

Converting hex to decimal is just like converting binary to decimal, except instead of powers of 2, we use powers of 16.

To convert 15E in hex to decimal:

$$\begin{array}{ccc} 1 & 5 & E \\ 16^2 & 16^1 & 16^0 \end{array}$$

$$\begin{aligned} \text{Weight} &= 1*16^2 + 5*16^1 + 14*16^0 \\ &= 1*256 + 5*16 + 14*1 \\ &= 350_{10} \end{aligned}$$

Decimal to Hex Conversion

Converting decimal to hex is just like converting decimal to binary, except instead of dividing by 2, we divide by 16. To convert 350 to hex:

$$\begin{aligned} 350/16 &= 21 \text{ remainder } 14 = E \\ 21/16 &= 1 \text{ remainder } 5 \\ 1/16 &= 0 \text{ remainder } 1 \end{aligned}$$

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

Hexa to Octal

A group of 4-bits represent a hexadecimal digit and a group of 3-bits represent an octal digit.

1. Convert the given hexadecimal number into binary.
2. Starting from right make groups of 3-bits and designate each group an octal digit.

. *Convert $(1A3)_{16}$ into octal.*

1. Converting hex to binary

$$(1\ A\ 3)_{16} = \underbrace{0001}_{1}\ \underbrace{1010}_{A}\ \underbrace{0011}_{3}$$

2. Grouping of 3-bits

$$(1A3)_{16} = \underset{0}{\underset{\downarrow}{000}}\ \underset{6}{\underset{\downarrow}{110}}\ \underset{4}{\underset{\downarrow}{100}}\ \underset{3}{\underset{\downarrow}{011}}$$

so $(1A3)_{16} = (0643)_8 \equiv (643)_8$

Octal to Hex Conversion

1. Convert the given octal number into binary.
2. Starting from right make groups of 4-bits and designate each group as a Hexadecimal digit.

Convert $(76)_8$ into hexadecimal.

Solution. 1. Converting octal to binary

$$(76)_8 = \underbrace{111}_{7}\ \underbrace{110}_{6}$$

2. Grouping of 4-bits

$$(76)_8 = \underset{3}{\underset{\downarrow}{11}}\ \underset{E}{\underset{\downarrow}{1110}} \equiv \underset{3}{\underset{\downarrow}{0011}}\ \underset{E}{\underset{\downarrow}{1110}}$$

$\therefore$ $(76)_8 = (3E)_{16}$

THE BINARY ARITHMETIC OPERATIONS

- Binary arithmetic's are simpler than decimal because they involve only two digits (bits) 1 and 0.
- Addition, subtraction, multiplication and division.

Binary Addition

Augend	Addend	Sum	Carry	Result
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	0	1	10

Binary subtraction

Minuend	Subtrahend	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

(i) Add 1010 and 0011 (ii) Add 0101 and 1111

$$\begin{array}{r l} 111\ & \leftarrow \text{Carry} \\ 0101\ & \\ +\ \underline{1111}\ & \\ 10100\ & \\ \uparrow \text{Carry} & \end{array}$$

$$\begin{array}{r l} 111\ & \leftarrow \text{Carry} \\ 0101\ & \\ +\ \underline{1111}\ & \\ 10100\ & \\ \uparrow \text{Carry} & \end{array}$$

(i) Subtract 0100 from 1011 (ii) Subtract 0110 from 1001

$$\begin{array}{cccc l} 1 & & & & \leftarrow \text{Borrow} \\ 1 & 0 & 1 & 1 & \leftarrow \text{Minuend} \\ -0 & 1 & 0 & 0 & \leftarrow \text{Subtrahend} \\ \hline 0 & 1 & 1 & 1 & \leftarrow \text{Difference} \\ \uparrow & \uparrow & \uparrow & \uparrow & \\ C_3 & C_2 & C_1 & C_0 & \end{array}$$

$$\begin{array}{cccc l} 1 & 1 & & & \leftarrow \text{Borrow} \\ 1 & 0 & 0 & 1 & \leftarrow \text{Minuend} \\ -0 & 1 & 1 & 0 & \leftarrow \text{Subtrahend} \\ \hline 0 & 0 & 1 & 1 & \leftarrow \text{Difference} \\ \uparrow & \uparrow & \uparrow & \uparrow & \\ C_3 & C_2 & C_1 & C_0 & \end{array}$$

The rules are still the same as in decimal, except that the borrow in a given significant position adds 2 to a minuend digit.

Binary multiplication

Multiplicand	Multiplier	product
0	0	0
1	0	0
1	1	1
0	1	0

(i) Multiply 1001 with 101

```
    1 0 0 1  ← MULTIPLICAND
      1 0 1  ← MULTIPLIER
    -------
    1 0 0 1  ← Partial Product when multiplier bit = 1
  0 0 0 0 x  ← Partial Product when multiplier bit = 0
1 0 0 1 x x
-----------
1 0 1 1 0 1  ← FINAL PRODUCT
```

Binary division.

Divisor	dividend	Q
X	Y	IF X< Y, Q = 1
X	Y	IF X>Y, Q = 0

Binary division is also similar to decimal division

```
                     1 0 1
Divisor → 1 0 0 1 ) 1 0 1 1 0 1 ← Dividend
                    1 0 0 1
                    -------
                    x x 1 0 0 1
                        1 0 0 1
                        -------
                        x x x x
```

NEGATIVE NUMBERS AND THEIR ARITHMETIC

So far, we have discussed straight forward number representation which are nothing but positive number. The negative numbers have got two representation

- **Complement representation**. In digital computers to simplify the subtraction operation & for logical manipulation complements are used. There are two types of complements used in each radix system.
  - The radix complement or r's complement
  - The diminished radix complements or (r-1)'s complement

r's Complement and (r – 1)'s Complement

The r's and (r – 1)'s complements are generalized representation of the complements. r stands for radix or base of the number system; thus, r's complement is referred as radix complement and (r – 1)'s complement is referred as diminished radix complement. Examples of r's complements are 2's complement and 10's complement. Examples of (r – 1)'s complement are 1's complement and 9's

In a base-r system, the r's and $(r - 1)$'s complement of the number N having n digits, can be defined as:

$$\boxed{(r - 1)\text{'s complement of } N = (r^n - 1) - N}$$

and

$$\boxed{\begin{aligned} r\text{'s complement of } N &= r^n - N \\ &= (r - 1)\text{'s complement of } N + 1 \end{aligned}}$$

The $(r - 1)$'s complement can also be obtained by subtracting each digit of N from $r-1$. Using the above methodology we can also define the 7's and 8's complement for octal system and 15's and 16's complement for hexadecimal system.

- **Sign magnitude representation** : Representation of signed no's binary arithmetic in computers: Two ways of representation of signed no's Sign Magnitude for Complemented form, Two complimented forms: 1's compliment form, & 2's compliment form

  1's and 2's Complement: These are the complements used for binary numbers. Their representation are very important as digital systems work on binary numbers only.

1's Complement

bit	Actual binary	complement

1's Complement	1	0
	0	1

1's complement of a binary number is obtained simply by replacing each 1 by 0 and each 0 by 1. Alternately, 1's complement of a binary can be obtained by subtracting each bit from 1.

EX: Find 1's complement of (i) 011001 (ii) 00100111

Sol: (i) Replace each 1 by 0 and each 0 by 1

0 1 1 0 0 1

↓ ↓ ↓ ↓ ↓ ↓

1 0 0 1 1 0

So, 1's complement of 011001 is 100110.

2's Complement: 2's complement of a binary number can be obtained by adding 1 to its 1's complement.

EX: Find 2's complement of (i) 011001 (ii) 010110016

Solution.

(*i*)

0	1	1	0	0	1		← Number
1	0	0	1	1	0		← 1's complement
				+	1		← Add 1 to 1's complement
1	0	0	1	1	1		← 2's complement

(*ii*)

0	1	0	1	1	0	0	← Number
1	0	1	0	0	1	1	← 1's complement
					+	1	← Add 1 to 1's complement
1	0	1	0	1	0	0	← 2's complement

Subtraction Using 1's and 2's Complement

Before using any complement method for subtraction equate the length of both minuend and subtrahend by introducing leading zeros.

1's complement subtraction following are the rules for subtraction using 1's complement.

1. To do the subtraction (M-S), represent the M&S in equal no. of digits.

2. Add 1's complement of subtrahend to minuend.

3. If a carry is produced by addition, then add this carry to the LSB of result. This is called as end around carry (EAC).

4. If carry is generated from MSB in step 2 then result is positive. If no carry generated result is negative, and is in 1's complement form.

EX: Perform binary subtraction for $(23)_{10}$-$(11)_{10}$

Sol: M= 23, 10111

S= 11, 1011

Step 1: represent the M&S in equal no. of digits. 10111=23

01011=11

Step 2: 1's complement of subtrahend (01011) = 10100

Add 1's complement of subtrahend to minuend 10111= M

\+ 10100=1'S Comp of S

Carry $\boxed{101011}$

Step3: If a carry is produced by addition, then add this carry to the LSB of result.

01011

\+ 1

01100 =12

2's complement Subtraction:

Method of 2's complement is similar to 1's complement subtraction except the end around carry (EAC). The rules are listed below:

1. To do the subtraction (M-S), represent the M&S in equal no. of digits.
2. Take 2's complement of subtrahend. Add 2's complement of subtrahend to minuend.

3. If a carry is produced, then discard the carry and the result is positive. If no carry is produced result is negative and is in 2's compliment form.

EX: Perform binary subtraction for $(22)_{10}$-$(12)_{10}$ Using 2's complement

Sol: M= 22, 10110

S= 12, 1100

Step 1: represent the M&S in equal no. of digits. 10110=22

01100=12

Step 2: 2's complement of subtrahend (01100) = 10100

Add 2's complement of subtrahend to minuend 10110= M

\+ 10100=1'S Comp of S

Carry $\boxed{101010}$

(Neglected)

Step 3: If a carry is produced, then discard the carry and the result is positive = (01010) = $(10)_{10}$

Signed Binary Representation

Untill now we have discussed representation of unsigned (or positive) numbers, except one or two places. In computer systems sign (+ve or –ve) of a number should also be represented by binary bits.

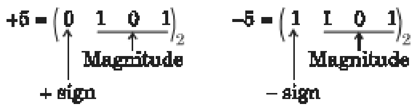
The accepted convention is to use 1 for negative sign and 0 for positive sign. In signed representation MSB of the given binary string represents the sign of the number, in all types of representation. We have two types of signed representation:

1. Signed Magnitude Representation

2. Signed Complement Representation

sign magnitude representation.

In a signed-magnitude representation, the MSB represent the sign and rest of the bits represent the magnitude. e.g.,



Note that positive number is represented similar to unsigned number. From the example it is also evident that out of 4-bits, only 3-bits are used to represent the magnitude.

What is sign magnitude of +5 and -7?

Sign bit	Actual binary number
+ is (0)	X
- Is (1)	X

Sol: actual number 5 is 0101 in binary number system. But to represent signed number in computer it has to represent in 8-bit binary number then

5 → 0000101 → 8 bit binary

+5 → 00000101 → signed magnitude for positive

-5 → 10000101 → signed magnitude for negative

Complement of signed magnitude representation

In a signed-complement representation the positive numbers are represented in true binary form with MSB as 0. Whereas the negative numbers are represented by taking appropriate complement of equivalent positive number, including the sign bit. Both 1's and 2's complements can be used for this purpose e.g.,

$+5 = (0101)_2$

$-5 = (1010)_2 \leftarrow$ in 1's complement

$= (1011)_2 \leftarrow$ in 2's complement

What is sign magnitude of +5 and -7 in 1's complement and 2's complement form

Sign bit	*1's complement* Of actualnumber	*2's complement* Of actual number
+ is (0)	X	X +1
- Is (1)	X	X+1

+5 → 00000101 → signed magnitude number

→ 11111010 → 1's complement of signed magnitude number

+1

→ 11111011 → 2's complement of signed magnitude number

<u>9's and 10's Complement</u>

9's and 10's complements are the methods used for the representation of decimal numbers. They are identical to the 1's and 2's complements used for binary numbers.

<u>9's complement:</u> 9's complement of a decimal number is defined as (10n – 1) – N, where n is no. of digits and N is given decimal numbers. Alternately, 9's complement of a decimal number can be obtained by subtracting each digit from 9.

9's complement of N = $(10^n-1) - N$

EX: Find out the 9's complement of $(36)_{10}$.

Sol: By using $(10^n-1) - N$; n = 2. So, $(10^n-1) - N = (100 - 1) - 36 = 63$

By subtracting each digit from 9

$$\begin{array}{r} 9\ 9 \\ -3\ 6 \\ \hline 6\ 3 \\ \hline \end{array}$$

So, 9's complement of 36 is 63.

10's complement: 10's complement of a decimal number is defined as $10^n - N$. 10's complement of N = $10^n - N$ (or)

$10^n - N = (10^n - 1) - N + 1 =$ 9's complement of N + 1. Thus, 10's complement of a decimal number can also be obtained by adding 1 to its 9's complement.

EX: Find out the 10's complement of $(36)_{10}$.

Solution. By adding 1 to 9's complement

$$\begin{aligned} \text{9's complement of 36} &= 99 - 36 \\ &= 63 \\ \text{Hence, 10's complement of 36} &= 63 + 1 \\ &= 64 \end{aligned}$$

CODES

Coding and encoding is the process of assigning a group of binary digits, commonly referred to as 'bits', to represent, identify, or relate to a multivalued items of information. In short, a code is a symbolic representation of an information transform. The bit combination is referred to as 'CODEWORDS'.

In a broad sense we can classify the codes into five groups:

(i) Weighted Binary codes (ii) Non-weighted codes (iii) sequential codes(iv) Error–detecting codes (v) Error–correcting codes (vi) Alphanumeric codes

i) Weighted Binary Codes

In weighted binary codes, each position of a number represents a specific weight. The bits are multiplied by the weights indicated; and the sum of these weighted bits gives the equivalent decimal digit.

a) **Straight Binary coding:** is a method of representing a decimal number by its binary equivalent. A straight binary code representing decimal 0 through 7

Decimal	*Three bit straight Binary Code*	*Weights MOI* 2^2	2^1	2^0	*Sum*
0	000	0	0	0	0
1	001	0	0	1	1
2	010	0	2	0	2
3	011	0	2	1	3
4	100	4	0	0	4
5	101	4	0	1	5
6	110	4	2	0	6
7	111	4	2	1	7

b) **Binary Codes Decimal Codes (BCD codes).** In BCD codes, individual decimal digits are coded in binary notation and are operated upon singly. Thus, binary codes representing 0 to 9 decimal digits are allowed. Therefore, all BCD codes have at least four bits (∵ min. no. of bits required to encode to decimal digits = 4) For example, decimal 364 in BCD
$3 \rightarrow 0011$

6 → 0110

4 → 0100

364 → 0011 0110 0100

However, we should realize that with 4 bits, total 16 combinations are possible (0000, 0001, ..., 1111) but only 10 are used (0 to 9). The remaining 6 combinations are invalid and commonly referred to as 'UNUSED CODES'

ii) Non weighted codes

Non weighted codes are codes that are not positionally weighted. That is, each position within the binary number is not assigned a fixed value. Ex: Excess-3 code, Gray code.

Excess-3 Code

Excess-3 is a non-weighted code used to express decimal numbers. The code derives its name from the fact that each binary code is the corresponding 8421 code plus 0011(3).

$[643]_{10}$ into XS3 code

Decimal	6	4	3
Add 3 to each	3	3	3
Sum	9	7	6

Converting the sum into BCD code we have

9	7	6
↓	↓	↓
1001	0111	0110

Hence, XS3 for $[643]_{10}$ = 1001 0111 0110

Gray Code

The Gray code belongs to a class of codes called minimum change codes, in which only one bit in the code changes when moving from one code to the next. The Gray code is non-weighted code, as the position of bit does not contain any weight. The Gray code is a reflective digital code which has the special property that any two subsequent numbers codes differ by only one bit. This is also called a unit-distance code. In digital Gray code has got a special place.

Gray codes*

Decimal Digit	*Three bit Gray code*	*Four bit Gray code*	*Decimal Digit*	*Three bit Gray code*	*Four bit Gray code*
0	0 0 0	0 0 0 0	8	–	1 1 0 0
1	0 0 1	0 0 0 1	9	–	1 1 0 1
2	0 1 1	0 0 1 1	10	–	1 1 1 1
3	0 1 0	0 0 1 0	11	–	1 1 1 0
4	1 1 0	0 1 1 0	12	–	1 0 1 0
5	1 1 1	0 1 1 1	13	–	1 0 1 1
6	1 0 1	0 1 0 1	14	–	1 0 0 1
7	1 0 0	0 1 0 0	15	–	1 0 0 0

Binary to Gray conversion:

N bit binary no is rep by $B_n\ B_{n-1}$ ----------- B_1

Gray code equivalent is by $G_n\ G_{n-1}$ ----------- G_1

B_n, G_n are the MSB's then the gray code bits are obtaind from the binary code as

$G_n=B_n$	$G_{n-1}=B_n \oplus B_{n-1}$	$G_{n-2}=B_{n-1} \oplus B_{n-2}$	-----------	$G_1=B_2 \oplus B_1$	

$\oplus$ →EX-or symbol

Procedure: ex-or the bits of the binary no with those of the binary no shifted one position to the right . The LSB of the shifted no. is discarded & the MSB of the gray code no.is the same as the MSB of the original binaryno.

EX: 10001 $\oplus$ $\oplus$ $\oplus$

(a). Binary : 1 →0 →0 →1

Gray : 1 1 0 1

(b). Binary: 1 0 0 1

Shifted binary: 1 0 0 (1)

——————————

1 1 0 1→gray

Gray to Binary Conversion:

If an n bit gray no. is rep by $G_n\ G_{n-1}$ ------------ G_1

its binary equivalent by $B_n\ B_{n-1}$ ------------ B_1 then the binary bits are obtained from gray bits as

$B_n= G_n$	$B_{n-1}=B_n \oplus G_{n-1}$	$B_{n-2}= \oplus\ G_{n-2}$	-----------	$B_1 = B_2 \oplus G_1$

To convert no. in any system into given no. first convert it into binary & then binary to gray. To convert gray no into binary no & convert binary no into require no system.

Ex:10110010(gray) = 11011100_2= DC_{16}=334_8=220_{10}

EX:1101

Gray: 1 1 0 1

↓ $\oplus$ $\oplus$ $\oplus$

Binary:1 0 0 1

Ex: $3A7_{16}$=$0011,1010,0111_2$=1001110100(gray)

527_8=$101,011,011_2$=111110110(gray)

652_{10}=1010001100_2= 1111001010(gray)

XS-3 gray code:

In a normal gray code , the bit patterns for 0(0000) & 9(1101) do not have a unit distance between them i.e, they differ in more than one position.In xs-3 gray code , each decimal digit is encoded with gray code patter of the decimal digit that is greater by 3. It has a unit distance between the patterns for 0 & 9.

XS-3 gray code for decimal digits 0 through 9

Decimal digit	Xs-3 gray code	Decimal digit	Xs-3 gray code
0	0010	5	1100
1	0110	6	1101
2	0111	7	1111
3	0101	8	1110
4	0100	9	1010

iii) Sequential Codes

A code is said to be sequential when two subsequent codes, seen as numbers in binary representation, differ by one. This greatly aids mathematical manipulation of data. The 8421 and

Excess-3 codes are sequential, whereas the 2421 and 5211 codes are not.

Binary coded decimal (bcd) and its arithmetic:

The BCD is a group of four binary bits that represent a decimal digit. In this representation each digit of a decimal number is replaced by a 4-bit binary number (i.e., a nibble). Since a decimal digit is a number from 0 to 9, a nibble representing a number greater than 9 is invalid BCD. For example (1010)2 is invalid BCD as it represents a number greater than 9.

Decimal Number	*Binary Representation*	*BCD Representation*
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 0
3	0 0 1 1	0 0 1 1
4	0 1 0 0	0 1 0 0
5	0 1 0 1	0 1 0 1
6	0 1 1 0	0 1 1 0
7	0 1 1 1	0 1 1 1
8	1 0 0 0	1 0 0 0
9	1 0 0 1	1 0 0 1
10	1 0 1 0	0 0 0 1 0 0 0 0
11	1 0 1 1	0 0 0 1 0 0 0 1
12	1 1 0 0	0 0 0 1 0 0 1 0
13	1 1 0 1	0 0 0 1 0 0 1 1
14	1 1 1 0	0 0 0 1 0 1 0 0
15	1 1 1 1	0 0 0 1 0 1 0 1

BCD Addition: In many application it is required to add two BCD numbers. But the adder circuits used are simple binary adders, which does not take care of peculiarity of BCD representation. Thus one must verify the result for valid BCD by using following rules:

1. If Nibble (i.e., group of 4-bits) is less than or equal to 9, it is a valid BCD number.

2. If Nibble is greater than 9, it is invalid. Add 6 (0110) to the nibble, to make it valid. Or If a carry was generated from the nibble during the addition, it is invalid. Add 6 (0110) to the nibble, to make it valid.

3. If a carry is generated when 6 is added, add this carry to next nibble.

EX: Add the BCD numbers i)1000 and 0101 and ii) 00011001 and 00011000

Solution.

```
    1 0 0 0  →    8
  + 0 1 0 1  →  + 5
    1 1 0 1      13
```

Since, $(1101)_2 > (9)_{10}$ add 6 (0110) to it

So,

```
      1 1 0 1
      0 1 1 0
    1 0 0 1 1
    1    3
```

So, result = 00010011

(ii)

```
          1                ← Carry generated from nibble
  0 0 0 1   1 0 0 1  →   19
  0 0 0 1   1 0 0 0  →  +18
  0 0 1 1   0 0 0 1      37
```

Since, a carry is generated from right most nibble we must add 6 (0110) to it.

So,

```
  0 0 1 1   0 0 0 1
            0 1 1 0
  0 0 1 1   0 1 1 1  →  (37)10
```

So, result = 00110111

BCD Subtraction:

The best way to cary out the BCD subtraction is to use complements. e. Although any of the two complements can be used, we prefer 10's complement for subtraction. Following are the steps to be followed for BCD subtraction using 10's complement:

1. Add the 10's complement of subtrahend to minuend.
2. Apply the rules of BCD addition to verify that result of addition is valid BCD.
3. Apply the rules of 10's complement on the result obtained in step 2, to declare the final result i.e., to declare the result of subtraction.

Ex: Subtract 61 from 68 using BCD.

Solution. To illustrate the process first we perform the subtraction using 10's comple ment in decimal system. After that we go for BCD subtraction.

we have, D = 68 – 61

So, 10's complement of 61 = 99 – 61 + 1 = 39

So, 10's complement of 61 = 99 – 61 + 1 = 39

So,

```
   6 8
 + 3 9
 1 0 7
 ↑
 Carry
```

In 10's complement if an end carry is produced then it is discarded and result is declared positive. So,

D = +07

by using BCD

1.

```
                         1              ← Carry generated
BCD of 68 =     0 1 1 0  1 0 0 0          from nibble
BCD of 39 =  +  0 0 1 1  1 0 0 1
                1 0 1 0  0 0 0 1
```

2. Check for valid BCD– since a carry is generated from right most nibble, we must add 6 (0110) to it. Since the left most nibble is greater than 9, we must add 6(0110) to it.

Thus,

```
     1 0 1 0   0 0 0 1
  +  0 1 1 0   0 1 1 0
```

(a) 8421 BCD code, sometimes referred to as the Natural Binary Coded Decimal Code (NBCD);

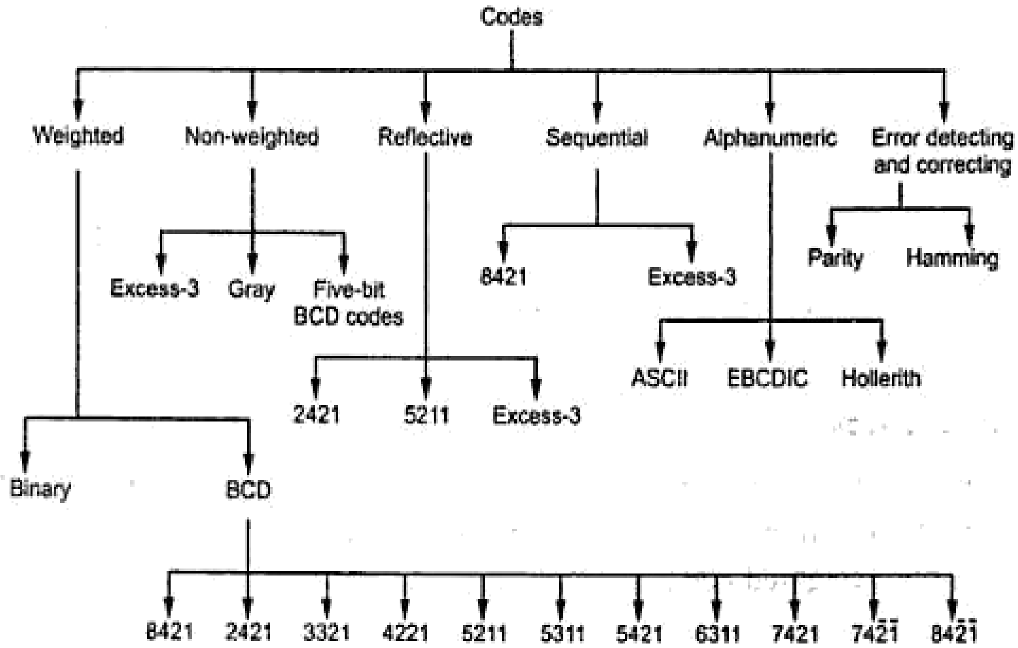
(b)* Excess-3 code (XS3); adding 3 to BCD gives the Excess -3 code.

(c)** 84 –2 –1 code (+8, +4, –2, –1);

(d) 2 4 2 1 code

Table BCD codes

Decimal Digit	*8421 (NBCD)*	*Excess-3 code (XS3)*	*84–2–1 code*	*2421 code*
0	0000	0011	0000	0000
1	0001	0100	0111	0001
2	0010	0101	0110	0010
3	0011	0110	0101	0011
4	0100	0111	0100	0100
5	0101	1000	1011	1011
6	0110	1001	1010	1100
7	0111	1010	1001	1101
8	1000	1011	1000	1110
9	1001	1100	1111	1111



Binary codes block diagram

Error – Detecting codes: When binary data is transmitted & processed,it is susceptible to noise that can alter or distort its contents. The 1‘s may get changed to 0‘s & 1‘s .because digital systems must be accurate to the digit, error can pose a problem. Several schemes have been devised to detect the occurrence of a single bit error in a binary word, so that whenever such an error occurs the concerned binary word can be corrected & retransmitted.

Parity: The simplest techniques for detecting errors is that of adding an extra bit known as parity bit to each word being transmitted.Two types of parity: Oddparity, evenparity forodd parity, the parity bit is set to a ‗0‘ or a ‗1‘ at the transmitter such that the total no. of 1 bit in the word including the parity bit is an odd no.For even parity, the parity bit is set to a ‗0‘ or a ‗1‘ at the transmitter such that the parity bit is an even no.

Decimal	8421 code	Odd parity	Even parity
0	0000 1		0
1	0001 0		1
2	0010 0		1
3	0011 1		0
4	0100 0		1
5	0100 1		0
6	0110 1		0
7	0111 0		1
8	1000 0		1
9	1001 1		0

When the digit data is received . a parity checking circuit generates an error signal if the total no of 1's is even in an odd parity system or odd in an even parity system. This parity check can always detect a single bit error but cannot detect 2 or more errors with in the same word.Odd parity is used more often than even parity does not detect the situation. Where all 0's are created by a short ckt or some other fault condition.

Ex: Even parity scheme
(a) 10101010 (b) 11110110 (c)10111001
Ans:

(a) No. of 1's in the word is even is 4 so there is no error
(b) No. of 1's in the word is even is 6 so there is no error
(c) No. of 1's in the word is odd is 5 so there is error

Ex: odd parity
(a)10110111 (b) 10011010 (c)11101010

Ans:

(a) No. of 1's in the word is even is 6 so word has error
(b) No. of 1's in the word is even is 4 so word has error
(c) No. of 1's in the word is odd is 5 so there is no error

Checksums:

Simple parity can't detect two errors within the same word. To overcome this, use a sort of 2 dimensional parity. As each word is transmitted, it is added to the sum of the previously transmitted words, and the sum retained at the transmitter end. At the end of transmission, the sum called the check sum. Up to that time sent to the receiver. The receiver can check its sum with the transmitted sum. If the two sums are the same, then no errors were detected at the receiver end. If there is an error, the receiving location can ask for retransmission of the entire data, used in teleprocessing systems.

Block parity:

Block of data shown is create the row & column parity bits for the data using odd parity. The parity bit 0 or 1 is added column wise & row wise such that the total no. of 1's in each column & row including the data bits & parity bit is odd as

Data	Parity bit
10110	0
10001	1
10101	0
00010	0
11000	1
00000	1
11010	0

data
10110
10001
10101
00010
11000
00000
11010

Error –Correcting Codes:

A code is said to be an error –correcting code, if the code word can always be deduced from an erroneous word. For a code to be a single bit error correcting code, the minimum distance of that code must be three. The minimum distance of that code is the smallest no. of bits by which any two code words must differ. A code with minimum distance of 3 can‘t only correct single bit errors but also detect (can‘t correct) two bit errors, The key to error correction is that it must be possible to detect & locate erroneous that it must be possible to detect & locate erroneous digits. If the location of an error has been determined. Then by complementing the erroneous digit, the message can be corrected , error correcting , code is the Hamming code , In this , to each group of m information or message or data bits, K parity checking bits denoted by P1,P2,----------pk located at positions 2^{k-1} from left are added to form an (m+k) bit code word.

To correct the error, k parity checks are performed on selected digits of each code word, & the position of the error bit is located by forming an error word, & the error bit is then complemented. The k bit error word is generated by putting a 0 or a 1 in the 2^{k-1}th position depending upon whether the check for parity involving the parity bit P_k is satisfied or not.Error positions & their corresponding values :

Error Position	For 15 bit code C_4 C_3 C_2 C_1	For 12 bit code C_4 C_3 C_2 C_1	For 7 bit code C_3 C_2 C_1
0	0000	0000	0 0 0
1	0001	0001	0 0 1
2	0010	0010	0 1 0
3	0011	0011	0 1 1
4	0100	0100	1 0 0
5	0101	0101	1 0 1
6	0 11 0	0 11 0	1 1 0
7	0 11 1	0 11 1	1 1 1
8	1 00 0	1 00 0	
9	1 00 1	1 00 1	
10	1 01 0	1 01 0	
11	1 01 1	1 01 1	
12	1 10 0	1 10 0	
13	1 10 1		
14	1 11 0		
15	1 11 1		

7- bit Hamming code:

To transmit four data bits, 3 parity bits located at positions 2^0 21&2^2 from left are added to make a 7 bit codeword which is then transmitted.

The word format

P_1	P_2	D_3	P_4	D_5	D_6	D_7

D—Data bits P-
Parity bits

Decimal Digit	For BCD P1P2D3P4D5D6D7	For Excess-3 P1P2D3P4D5D6D7
0	0 0 0 0 00 0	1 0 0 0 0 1 1
1	1 1 0 1 00 1	1 0 0 1 1 0 0
2	0 1 0 1 01 1	0 1 0 0 1 0 1
3	1 0 0 0 01 1	1 1 0 0 1 1 0
4	1 0 0 1 10 0	0 0 0 1 1 1 1
5	0 1 0 0 10 1	1 1 1 0 0 0 0
6	1 1 0 0 11 0	0 0 1 1 0 0 1
7	0 0 0 1 11 1	1 0 1 1 0 1 0

8	1 1 1 0 0 0 0	0 1 1 0 0 1 1
9	0 0 1 1 0 0 1	0 1 1 1 1 0 0

Ex: Encode the data bits 1101 into the 7 bit even parity Hamming Code
The bit pattern is
P1P2D3P4D5D6D7

1 1 0 1

Bits 1,3,5,7 (P_1 111) must have even parity, so P_1 =1
Bits 2, 3, 6, 7(P_2 101) must have even parity, so P_2 =0
Bits 4,5,6,7 (P_4 101)must have even parity, so P_4 =0
The final code is 1010101

EX: Code word is 1001001

Bits 1,3,5,7 (C_1 1001) →no error →put a 0 in the 1's position→C1=0
Bits 2, 3, 6, 7(C_2 0001)) → error →put a 1 in the 2's position→C2=1
Bits 4,5,6,7 (C_4 1001)) →no error →put a 0 in the 4's position→C3=0

15-bit Hamming Code: It transmit 11 data bits, 4 parity bits located 2^0 2^1 2^2 2^3 Word format is

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D10	D11	D12	D13	D14	D15

12-Bit Hamming Code:It transmit 8 data bits, 4 parity bits located at position 2^0 2^1 2^2 2^3 Word format is

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D10	D11	D12

Alphanumeric Codes:

These codes are used to encode the characteristics of alphabet in addition to the decimal digits. It is used for transmitting data between computers & its I/O device such as printers, keyboards & video display terminals.Popular modern alphanumeric codes are ASCII code & EBCDIC code.

MODULE -II
Boolean Algebra& Boolean functions

Boolean algebra

In 1854, George Boole developed an algebraic system now called Boolean algebra. In 1938, Claude E. Shannon introduced a two - valued Boolean algebra called switching algebra that represented the properties of bistable electrical switching circuits. For the formal definition of Boolean algebra, we shall employ the postulates formulated by E. V. Huntington in 1904.

Boolean algebra is a system of mathematical logic. It is an algebraic system consisting of the set of elements (0, 1), two binary operators called OR, AND, and one unary operator NOT. It is the basic mathematical tool in the analysis and synthesis of switching circuits. It is a way to express logic functions algebraically.

Axioms and laws of Boolean algebra

Axioms or Postulates of Boolean algebra are a set of logical expressions that we accept without proof and upon which we can build a set of useful theorems.

	AND Operation	OR Operation	NOT Operation
Axiom1 :	0.0=0	0+0=0	0=1
Axiom2:	0.1=0	0+1=1	1=0
Axiom3:	1.0=0	1+0=1	
Axiom4:	1.1=1	1+1=1	

Complementation law

Law1: 0=1 Law3: if A=0,then A=1

Law2: 1=0 **Law4: if A=1,then =0**

Law5: if =A (double inversion law)

AND Law	OR Law
Law1: A.0=0 (Null law)	Law1: A+0=A
Law2: A.1=A (Identity law)	Law2: A+1=1
Law3: A.A=A (Impotence law)	Law3:A+A=A(Impotencelaw)
Law4: A. A=0	Law4: A+ A=1

Basic Theorems and Properties of Boolean algebra

Commutative law

Law1: A+B=B+A **Law2: A.B=B.A**

Associative law

Law1: A + (B +C) = (A +B) +C **Law2: A(B.C) = (A.B)C**

Distributive law

Law1: A.(B + C) = AB+ AC **Law2: A + BC = (A + B).(A +C)**

Absorption law

Law1: A +AB =A **Law2: A(A +B) = A**

Solution: A(1+B)
A

Solution: A.A+A.B
A+A.B
A(1+B)
A

DeMorgan Theorems

Theorem1: $\overline{(A+B)} = \bar{A}.\bar{B}$ **Theorem2: $\overline{(A.B)} = \bar{A}+\bar{B}$**

Redundant Literal Rule

Rule1: $A+\bar{A}.B=A+B$

Solution: $A+\bar{A}.B$

$(A+\bar{A}).(A+B)$ ∴ A+BC=(A +B).(A +C) A+B ∴ $A+\bar{A}=1$

Rule2: $A.(\bar{A}+B)=AB$

Solution: $A.(\bar{A}+B)$

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

AB

Consensus Theorem

Theorem1. AB+ A’C + BC = AB + A’C Theorem2. (A+B). (A’+C).(B+C) =(A+B).(A’+C)

The BC term is called the consensus term and is redundant. The consensus term is formed from a PAIR OF TERMS in which a variable (A) and its complement (A’) are present; the consensus term is formed by multiplying the two terms and leaving out the selected variable and its complement

Consensus Theorem1 Proof:

AB+A'C+BC=AB+A'C+(A+A')BC

=AB+A'C+ABC+A'BC

=AB(1+C)+A'C(1+B)

= AB+ A'C

Principle of Duality

Each postulate consists of two expressions statement one expression is transformed into the other by interchanging the operations (+) and (·) as well as the identity elements 0 and 1. Such expressions are known as duals of each other.

If some equivalence is proved, then its dual is also immediately true. E.g. If we prove: (x.x)+(x'+x')=1, then we have by duality: (x+x)·(x'.x')=0

The Huntington postulates were listed in pairs and designated by part (a) and part (b) in below table.

Table for Postulates and Theorems of Boolean algebra

Part-A	Part-B
A+0=A	A.0=0
A+1=1	A.1=A
A+A=A (Impotence law)	A.A=A (Impotence law)
A+ A=1	A. A=0
=A (double inversion law)	- -
Commutative law: A+B=B+A	A.B=B.A
Associativelaw: A + (B +C) = (A +B) +C	A(B.C) = (A.B)C
Distributivelaw: A.(B + C) = AB+ AC	A + BC = (A + B).(A +C)
Absorptionlaw: A +AB =A	A(A +B)= A
DeMorgan Theorem: A+B =A.B	(A.B)=A+ B
Redundant Literal Rule: A+ A.B=A+B	A.(A+B)=AB
ConsensusTheorem:AB+A'C+BC= AB+A'C	(A+B). (A'+C).(B+C) =(A+B).(A'+C)

Boolean Function

Boolean algebra is an algebra that deals with binary variables and logic operations. A Boolean function described by an algebraic expression consists of binary variables, the constants 0 and 1, and the logic operation symbols.
For a given value of the binary variables, the function can be equal to either 1 or 0.

F(vars) = expression

Set of binary Variables → **Operators (+, •, ‘)**
Constants (0, 1)
Groupings (parenthesis)
Variables

Consider an example for the Boolean function

F1 = x + y’z

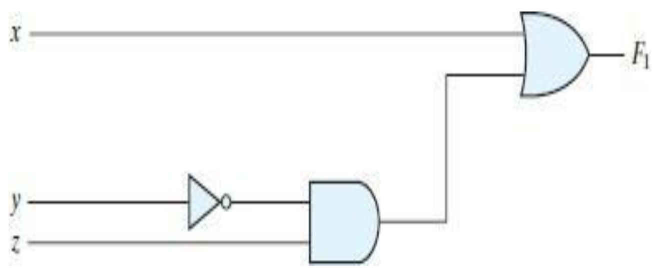
The function F1 is equal to 1 if x is equal to 1 or if both y’ and z are equal to 1. F1 is equal to 0 otherwise. The complement operation dictates that when y’ = 1, y = 0. Therefore, F1 = 1 if x = 1 or if y = 0 and z = 1.

A Boolean function expresses the logical relationship between binary variables and is evaluated by determining the binary value of the expression for all possible values of the variables.

A Boolean function can be represented in a truth table. The number of rows in the truth table is 2^n, where n is the number of variables in the function. The binary combinations for the truth table are obtained from the binary numbers by counting from 0 through $2^n - 1$.

Truth Table for F1

x	y	z	F_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	1



Gate Implementation of F1 = x + y’z

Note:

Q:Let a function F() depend on n variables. How many rows are there in the truth table of F() ?
A:2^n rows, since there are 2^n possible binary patterns/combinations for the n variables.

Truth Tables

Enumerates all possible combinations of variable values and the corresponding function value
Truth tables for some arbitrary functions
F1(x,y,z), F2(x,y,z), and F3(x,y,z) are shown to the below.

x	y	z	F_1	F_2	F_3
0	0	0	0	1	1
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	0	1	1
1	0	0	0	1	0
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	1	0	1

Truth table: a unique representation of a Boolean function
If two functions have identical truth tables, the functions are equivalent (and vice - versa).
Truth tables can be used to prove equality theorems.
However, the size of a truth table grows exponentially with the number of variables involved, hence unwieldy. This motivates the use of Boolean Algebra.

Boolean expressions-NOT unique

Unlike truth tables, expressions epresenting a Boolean function are NOT unique.

- **Example:**

– F(x,y,z) = x'•y'•z' + x'•y•z' + x•y•z'
– G(x,y,z) = x'•y'•z' + y•z'

- **The corresponding truth tables for F() and G() are to the right. They are identical.**
- **Thus, F() = G()**

x	y	z	F	G
0	0	0	1	1
0	0	1	0	0
0	1	0	1	1
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

Algebraic Manipulation (Minimization of Boolean function)

- **Boolean algebra is a useful tool for simplifying digital circuits.**
- **Why do it? Simpler can mean cheaper, smaller, faster.**
- **Example: Simplify F = x'yz + x'yz' + xz.**

F= x'yz + x'yz' + xz

=x'y(z+z') + xz

= x'y•1 + xz

= x'y + xz

- **Example: Prove**

x'y'z' + x'yz' + xyz' = x'z' + yz'

- **Proof:**

x'y'z'+ x'yz'+ xyz'

= x'y'z' + x'yz' + x'yz' + xyz'

= x'z'(y'+y) + yz'(x'+x)

= x'z'•1 + yz'•1

= x'z' + yz'

Complement of a Function

The complement of a function is derived by interchanging (• and +), and (1 and 0), and complementing each variable.

Otherwise, interchange 1s to 0s in the truth table column showing F. The complement of a function IS NOT THE SAME as the dual of a function.

Example

- **Find G(x,y,z), the complement of F(x,y,z) = xy'z' + x'yz Ans: G = F' = (xy'z' + x'yz)'**

= (xy'z')' • (x'yz)' DeMorgan

= (x'+y+z) • (x+y'+z') DeMorgan again

Note: The complement of a function can also be derived by finding the function's dual, and then complementing all of the literals

Canonical and Standard Forms

We need to consider formal techniques for the simplification of Boolean functions.
Identical functions will have exactly the same canonical form.
Minterms and Maxterms
Sum - of - Minterms and Product - of -
Maxterms Product and Sum terms
Sum - of - Products (SOP) and Product - of - Sums (POS)

Definitions

Literal: A variable or its complement
Product term: literals connected by •
Sum term: literals connected by +
Minterm: a product term in which all the variables appear exactly once, either complemented or uncomplemented.
Maxterm: a sum term in which all the variables appear exactly once, either complemented or uncomplemented.
Canonical form: Boolean functions expressed as a sum of Minterms or product of Maxterms are said to be in canonical form.

Minterm

Represents exactly one combination in the truth table.
Denoted by m_j, where j is the decimal equivalent of the minterm's corresponding binary combination (b_j).
A variable in m_j is complemented if its value in b_j is 0, otherwise is uncomplemented.

Example: Assume 3 variables (A, B, C), and j=3. Then, b_j = 011 and its corresponding minterm is denoted by m_j = A'BC

Maxterm

Represents exactly one combination in the truth table.
Denoted by M_j, where j is the decimal equivalent of the maxterm's corresponding binary combination (b_j).
A variable in M_j is complemented if its value in b_j is 1, otherwise is uncomplemented.

Example: Assume 3 variables (A, B, C), and j=3. Then, b_j = 011 and its corresponding maxterm is denoted by M_j = A+B'+C'

Truth Table notation for Minterms and Maxterms

- **Minterms and Maxterms are easy to denote using a truth table. Example: Assume 3 variables x,y,z (order is fixed)**

x	y	z	Minterm	Maxterm
0	0	0	$x'y'z' = m_0$	$x+y+z = M_0$
0	0	1	$x'y'z = m_1$	$x+y+z' = M_1$
0	1	0	$x'yz' = m_2$	$x+y'+z = M_2$
0	1	1	$x'yz = m_3$	$x+y'+z' = M_3$
1	0	0	$xy'z' = m_4$	$x'+y+z = M_4$
1	0	1	$xy'z = m_5$	$x'+y+z' = M_5$
1	1	0	$xyz' = m_6$	$x'+y'+z = M_6$
1	1	1	$xyz = m_7$	$x'+y'+z' = M_7$

Canonical Forms

- **Every function F() has two canonical forms:**

– Canonical Sum - Of - Products (sum of minterms)

– Canonical Product - Of - Sums (product of maxterms)

Canonical Sum - Of - Products:

The minterms included are those m_j such that F( ) = 1 in row j of the truth table for F( ).

Canonical Product - Of - Sums:

The maxterms included are those M_j such that F( ) = 0 in row j of the truth table for F( ).

Example

Consider a Truth table for $f_1(a,b,c)$ at right The canonical sum - of - products form for f_1 is $f_1(a,b,c) = m_1 + m_2 + m_4 + m_6$

$$= a'b'c + a'bc' + ab'c' + abc'$$

The canonical product - of - sums form for f_1 is $f_1(a,b,c) = M_0 \bullet M_3 \bullet M_5 \bullet M_7$

$$= (a+b+c)\bullet(a+b'+c')\bullet (a'+b+c')\bullet(a'+b'+c').$$

- **Observe that: $m_j = M_j'$**

a	b	c	f_1
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	1
1	1	1	0

Shorthand: ∑ and ∏

- **$f_1(a,b,c) = \sum m(1,2,4,6)$, where ∑ indicates that this is a sum - of - products form, and m(1,2,4,6) indicates that the minterms to be included are m_1, m_2, m_4, and m_6.**
- **$f_1(a,b,c) = \prod M(0,3,5,7)$, where ∏ indicates that this is a product - of - sums form, and M(0,3,5,7) indicates that the maxterms to be included are M_0, M_3, M_5, and M_7.**
- **Since $m_j = M_j'$ for any j,**
  $\sum m(1,2,4,6) = \prod M(0,3,5,7) = f_1(a,b,c)$
-

Conversion between Canonical Forms

- **Replace ∑ with ∏ (or vice versa) and replace those j's that appeared in the original form with those that do not.**
  - **Example:**

$f_1(a,b,c)= a'b'c + a'bc' + ab'c' + abc'$

$$= m_1 + m_2 + m_4 + m_6$$
$$= \sum(1,2,4,6)$$
$$= \prod(0,3,5,7)$$
$$= (a+b+c)\bullet(a+b'+c')\bullet(a'+b+c')\bullet(a'+b'+c')$$

Standard Forms

Another way to express Boolean functions is in standard form. In this configuration, the terms that form the function may contain one, two, or any number of literals.
There are two types of standard forms: the sum of products and products of sums.
The sum of products is a Boolean expression containing AND terms, called product terms, with one or more literals each. The sum denotes the ORing of these terms. An example of a function expressed as a sum of products is
$F1 = y' + xy + x'yz'$
The expression has three product terms, with one, two, and three literals. Their sum is, in effect, an OR operation.
A product of sums is a Boolean expression containing OR terms, called sum terms. Each term may have any number of literals. The product denotes the ANDing of these terms. An example of a function expressed as a product of sums is
$F2 = x(y' + z)(x' + y + z')$
This expression has three sum terms, with one, two, and three literals. The product is an AND operation.

Conversion of SOP from standard to canonical form Example-1.

Express the Boolean function F = A + B'C as a sum of minterms.

Solution: The function has three variables: A, B, and C. The first term A is missing two variables; therefore, A=A(B+B')=AB+AB'

This function is still missing one variable, so

A = AB(C + C') + AB' (C + C')

= ABC + ABC' + AB'C + AB'C'

The second term B'C is missing one variable; hence,

B'C = B'C(A + A') = AB'C + A'B'C

Combining all terms, we have

F=A+B'C

= ABC + ABC' + AB'C + AB'C'+ A'B'C

But AB'C appears twice, and according to theorem (x + x = x), it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

F = A'B'C + AB'C + AB'C + ABC' + ABC

= m1 + m4 + m5 + m6 + m7

When a Boolean function is in its sum - of - minterms form, it is sometimes convenient to express the function in the following brief notation:

F(A, B, C) = ∑m (1, 4, 5, 6, 7)

Example-2.

Express the Boolean function F = xy + x'z as a product of maxterms.

Solution: First, convert the function into OR terms by using the distributive law:

F = xy + x'z = (xy + x')(xy + z)

=(x + x')(y + x')(x + z)(y + z)

= (x'+ y)(x + z)(y + z)

The function has three variables: x, y, and z. Each OR term is missing one variable; therefore, x'+ y = x' + y + zz' = (x' + y + z)(x' + y + z')

x + z = x + z + yy' = (x + y + z)(x + y' + z)

y + z = y + z + xx' = (x + y + z)(x' + y + z)

Combining all the terms and removing those which appear more than once, we finally obtain F = (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z)

F= M0M2M4M5

A convenient way to express this function is as follows:

F(x, y, z) = πM(0, 2, 4, 5)

The product symbol, π, denotes the ANDing of maxterms; the numbers are the indices of the maxterms of the function.

Digital Logic Gates

Boolean functions are expressed in terms of AND, OR, and NOT operations, it is easier to implement a Boolean function with these type of gates.

Name	Graphic symbol	Algebraic function	Truth table
AND		$F = x \cdot y$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$F = x'$	x | F 0 | 1 1 | 0
Buffer		$F = x$	x | F 0 | 0 1 | 1
NAND		$F = (xy)'$	x y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = (x + y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Properties of XOR Gates

- **XOR (also) : the “not ‐ equal” function**
- **XOR(X,Y) = X Y = X’Y + XY’**
- **Identities:**

–X0=X
– X 1=X’
–XX=0
– X X’=1

- **Properties:**

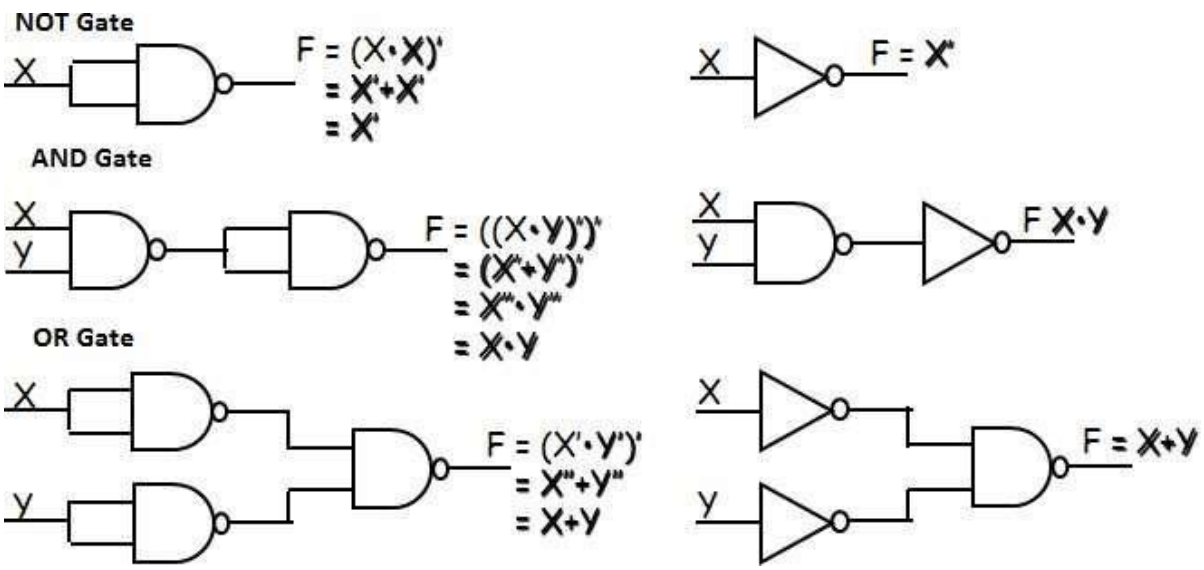
–XY=YX
– (X Y) W=X (Y W)

Universal Logic Gates

NAND and NOR gates are called Universal gates. All fundamental gates (NOT, AND, OR) can be realized by using either only NAND or only NOR gate. A universal gate provides flexibility and offers enormous advantage to logic designers.

NAND as a Universal Gate

NAND Known as a “universal” gate because ANY digital circuit can be implemented with NAND gates alone.
To prove the above, it suffices to show that AND, OR, and NOT can be implemented using NAND gates only.



Two-variable k-map:

A two-variable k-map can have 2^2=4 possible combinations of the input variables A and B. Each of these combinations, , B,A ,AB(in the SOP form) is called a minterm. The minterm may be represented in terms of their decimal designations – m0 for , m1 for B,m2 for A and m3 for AB, assuming that A represents the MSB. The letter m stands for minterm and the subscript represents the decimal designation of the minterm. The presence or absence of a minterm in the expression indicates that the output of the logic circuit assumes logic 1 or logic 0 level for that combination of input variables.

The expression f= ,+ B+A +AB , it can be expressed using min

term as F= m0+m2+m3=∑m(0,2,3)

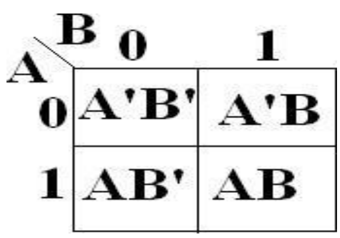
Using Truth Table:

Minterm	Inputs		Output	
	A	B	F	
0	0	0	1	
1	0	1	0	
2	1	0	1	
3	1	1	1	

A 1 in the output contains that particular minterm in its sum and a 0 in that column indicates that the particular mintermdoes not appear in the expression for output . this information can also be indicated by a two-variable k-map.

Mapping of SOP Expresions:

A two-variable k-map has 22=4 squares .These squares are called cells. Each square on the k-map represents a unique minterm. The minterm designation of the squares are placed in any square, indicates that the corresponding minterm does output expressions. And a 0 or no entry in any square indicates that the corresponding minterm does not appear in the expression for output.



The minterms of a two-variable k-map

The mapping of the expressions =∑m(0,2,3)is

A \ B	0	1
0	1 (0)	0 (1)
1	1 (2)	1 (3)

k-map of ∑m(0,2,3)

EX: Map the expressions f= B+A

F= m_1+m_2=∑m(1,2)The k-map is

A \ B	0	1
0	0 (0)	1 (1)
1	1 (2)	0 (3)

Minimizations of SOP expressions:

To minimize Boolean expressions given in the SOP form by using the k-map, look for adjacent adjacent squares having 1's minterms adjacent to each other, and combine them to form larger squares to eliminate some variables. Two squares are said to be adjacent to each other, if their minterms differ in only one variable. (i.e, B & A differ only in one variable. so they may be combined to form a 2-square to eliminate the variable B.similarly all other.

The necessary condition for adjacency of minterms is that their decimal designations must differ by a power of 2. A minterm can be combined with any number of minterms adjacent to it to form larger squares. Two minterms which are adjacent to each other can be combined to form a bigger square called a 2-square or a pair. This eliminates one variable – the variable that is not common to both the minterms. For EX:

m0 and m1 can be combined to yield,

f_1 = m0+m1= + B= (B+

)= m0 and m2 can be combined to yield,

f_2 = m0+m2= + = (+)=

m1 and m3 can be combined to yield,

f_3= m1+m3= B+AB=B(+)=B

m2 and m3 can be combined to yield,

f_4 = m2+m3=A +AB=A(B+)=A

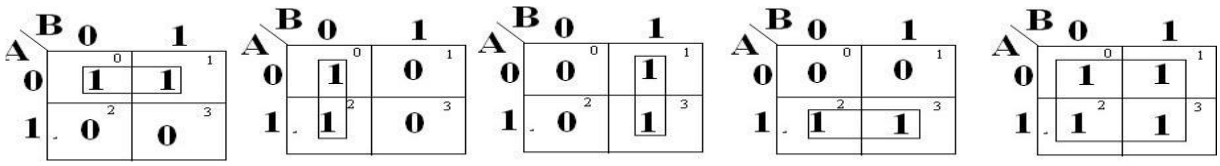
m_0 ,m_1 ,m_2 and m_3 can be combined to yield,

= **+ +A +AB**

= **(B+) +A(B+)**

= **+A**

=1

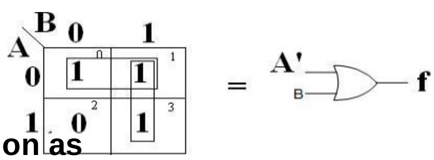


f1= f2= f3=B f4=A f5=1

The possible minterm groupings in a two-variable k-map.

Two 2-squares adjacent to each other can be combined to form a 4-square. A 4-square eliminates 2 variables. A 4-square is called a quad. To read the squares on the map after minimization, consider only those variables which remain constant through the square, and ignore the variables which are varying. Write the non complemented variable if the variable is remaining constant as a 1, and the complemented variable if the variable is remaining constant as a 0, and write the variables as a product term. In the above figure f_1 read as , because, along the square , A remains constant as a 0, that is , as , where as B is changing from 0 to 1.

EX: Reduce the minterm f= +A +AB using mapping Expressed in terms of minterms, the given expression is F=m_0+m_1+m_2+ m_3=m∑(0,1,3)& the figure shows the k-map for f and its reduction . In one 2-square, A is constant as a 0 but B varies from a 0 to a 1, and in the other 2-square, B is constant as a 1 but A varies from a 0 to a 1. So, the reduced expressions is +B.



It requires two gate inputs for realization as
f= +B (k-map in SOP form, and logic diagram.)

The main criterion in the design of a digital circuit is that its cost should be as low as possible. For that the expression used to realize that circuit must be minimal.Since the cost is proportional to number of gate inputs in the circuit in the circuit, an expression is considered minimal only if it corresponds to the least possible number of gate inputs. & there is no guarantee for that k-map in SOP is the real minimal. To obtain real minimal expression, obtain the minimal expression both in SOP & POS form form by using k-maps and take the minimal of these two minimals.

The 1's on the k-map indicate the presence of minterms in the output expressions, where as the 0s indicate the absence of minterms .Since the absence of a minterm in the SOP expression means the presense of the corresponding maxterm in the POS expression of the same .when a SOP expression is plotted on the k-map, 0s or no entries on the k-map represent the maxterms. To obtain the minimal expression in the POS form, consider the 0s on the k-map and follow the procedure used for combining 1s. Also, since the absence of a maxterm in the POS expression means the presence of the corresponding minterm in the SOP expression of the same , when a POS expression is plotted on the k-map, 1s or no entries on the k-map represent the minterms.

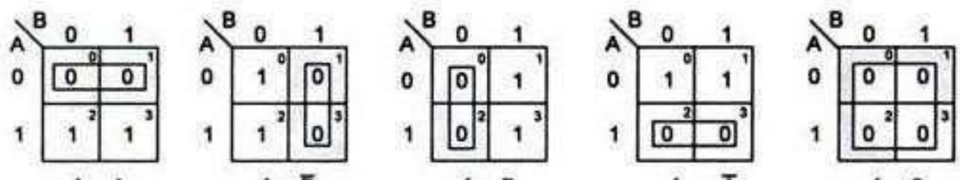
Mapping of POS expressions:

Each sum term in the standard POS expression is called a maxterm. A function in two variables (A, B) has four possible maxterms, A+B,A+ , +B, +

. They are represented as M_0, M_1, M2, and M3respectively. The uppercase letter M stands for maxterm and its subscript denotes the decimal designation of that maxterm obtained by treating the non-complemented variable as a 0 and the complemented variable as a 1 and putting them side by side for reading the decimal equivalent of the binary number so formed.

For mapping a POS expression on to the k-map, 0s are placed in the squares corresponding to the maxterms which are presented in the expression an d1s are placed in the squares corresponding to the maxterm which are not present in the expression. The decimal designation of the squares of the squares for maxterms is the same as that for the minterms. A two-variable k-map & the associated maxterms are asthe maxterms of a two-variable k-map

The possible maxterm groupings in a two-variable k-map



$f_1 = A$ $f_2 = \bar{B}$ $f_3 = B$ $f_4 = \bar{A}$ $f_5 = 0$

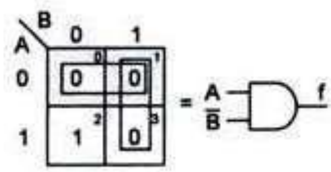
Minimization of POS Expressions:

To obtain the minimal expression in POS form, map the given POS expression on to the K-map and combine the adjacent 0s into as large squares as possible. Read the squares putting the complemented variable if its value remains constant as a 1 and the non-complemented variable if its value remains constant as a 0 along the entire square (ignoring the variables which do not remain constant throughout the square) and then write them as a sum term.

Various maxterm combinations and the corresponding reduced expressions are shown in figure. In this f_1 read as A because A remains constant as a 0 throughout the square and B changes from a 0 to a 1. f_2 is read as B' because B remains constant along the square as a 1 and A changes from a 0 to a 1. f_5 Is read as a 0 because both the variables are changing along the square.

Ex: Reduce the expression f=(A+B)(A+B')(A'+B') using mapping.

The given expression in terms of maxterms is f=πM(0,1,3). It requires two gates inputs for realization of the reduced expression as



F=AB'

K-map in POS form and logic diagram

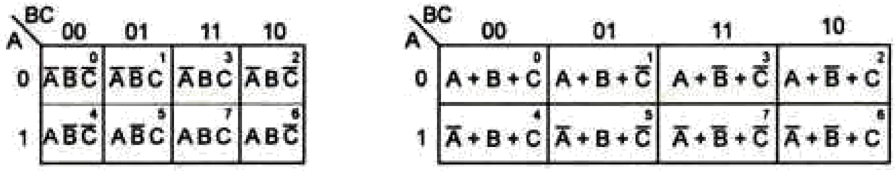
In this given expression ,the maxterm M_2 is absent. This is indicated by a 1 on the k-map. The corresponding SOP expression is $\sum m_2$ or AB'. This realization is the same as that for the POS form.

Three-variable K-map:

A function in three variables (A, B, C) expressed in the standard SOP form can have eight possible combinations: A B C , AB C,A BC ,A BC,AB C ,AB C,ABC , and ABC. Each one of these combinations designate d by m0,m1,m2,m3,m4,m5,m6, and m7, respectively, is called a minterm. A is the MSB of the minterm designator and C is the LSB.

In the standard POS form, the eight possible combinations are:A+B+C, A+B+C , A+B +C,A+B + C ,A + B+ C,A + B + C ,A + B + C,A + B + C . Each oneof these combinations designated by M_0, M1, M2, M3, M4, M5, M6, and M7respectively is called a maxterm. A is the MSB of the maxterm designator and C is the LSB.

A **three-variable k-map has, therefore, 8(=2^3) squares or cells, and each square on the map represents a minterm or maxterm as shown in figure. The small number on the top right corner of each cell indicates the minterm or maxterm designation.**



(a) Minterms (b) Maxterms

The three-variable k-map.

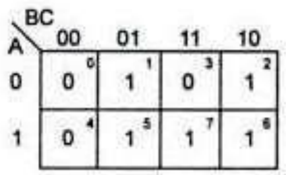
The binary numbers along the top of the map indicate the condition of B and C for each column. The binary number along the left side of the map against each row indicates the condition of A for that row. For example, the binary number 01 on top of the second column in fig indicates that the variable B appears in complemented form and the variable C in non-complemented form in all the minterms in that column. The binary number 0 on the left of the first row indicates that the variable A appears in complemented form in all the minterms in that row, the binary numbers along the top of the k-map are not in normal binary order. They are, infact, in the Gray code. This is to ensure that twophysically adjacent squares are really adjacent, i.e., their minterms or maxterms differ by only one variable.

Ex: Map the expression f=: C+ + + +ABC

In the given expression , the minterms are : C=001=m_1 ; =101=m_5; =010=m_2;

=110=m_6;ABC=111=m_7.

So the expression is f=∑m(1,5,2,6,7)= ∑m(1,2,5,6,7). The corresponding k-map is



K-map in SOP form

Ex: Map the expression f= (A+B+C),(+ +) (+ +)(A + +)(+ +)

In the given expression the maxterms are :A+B+C=000=M_0; + + =101=M_5; + + = 111=M_7; A + + =011=M_3; + + =110=M_6.

So the expression is f = π M (0,5,7,3,6)= π M (0,3,5,6,7). The mapping of the expression is

A \ BC	00	01	11	10
0	0 [0]	1 [1]	0 [3]	1 [2]
1	1 [4]	0 [5]	0 [7]	0 [6]

K-map in POS form.

Minimization of SOP and POS expressions:

For reducing the Boolean expressions in SOP (POS) form plotted on the k-map, look at the 1s (0s) present on the map. These represent the minterms (maxterms). Look for the minterms (maxterms) adjacent to each other, in order to combine them into larger squares. Combining of adjacent squares in a k-map containing 1s (or 0s) for the purpose of simplification of a SOP (or POS)expression is called looping. Some of the minterms (maxterms) may have many adjacencies. Always start with the minterms (maxterm) with the least number of adjacencies and try to form as large as large a square as possible. The larger must form a geometric square or rectangle. They can be formed even by wrapping around, but cannot be formed by using diagonal configurations. Next consider the minterm (maxterm) with next to the least number of adjacencies and form as large a square as possible. Continue this till all the minterms (maxterms) are taken care of . A minterm (maxterm) can be part of any number of squares if it is helpful in reduction. Read the minimal expression from the k-map, corresponding to the squares formed. There can be more than one minimal expression.

Two squares are said to be adjacent to each other (since the binary designations along the top of the map and those along the left side of the map are in Gray code), if they are physically adjacent to each other, or can be made adjacent to each other by wrapping around. For squares to be combinable into bigger squares it is essential but not sufficient that their minterm designations must differ by a power of two.

General procedure to simplify the Boolean expressions:

1. **Plot the k-map and place 1s(0s) corresponding to the minterms (maxterms) of the SOP (POS) expression.**
2. **Check the k-map for 1s(0s) which are not adjacent to any other 1(0). They are isolated minterms(maxterms) . They are to be read as they are because they cannot be combined even into a 2-square.**
3. **Check for those 1s(0S) which are adjacent to only one other 1(0) and make them pairs (2 squares).**
4. **Check for quads (4 squares) and octets (8 squares) of adjacent 1s (0s) even if they contain some 1s(0s) which have already been combined. They must geometrically form a square or a rectangle.**
5. **Check for any 1s(0s) that have not been combined yet and combine them into bigger squares if possible.**
6. **Form the minimal expression by summing (multiplying) the product the product (sum) terms of all the groups.**

Reading the K-maps:

While reading the reduced k-map in SOP (POS) form, the variable which remains constant as 0 along the square is written as the complemented (non-complemented) variable and the one which remains constant as 1 along the square is written as non-complemented (complemented) variable and the term as a product (sum) term. All the product (sum) terms are added (multiplied).

Some possible combinations of minterms and the corresponding minimal expressions readfrom the k-maps are shown in fig: Here f_6 is read as 1, because along the 8-square no variable remains constant. F_5 is read as , because, along the 4-square formed by0,m_1,m_2 and m_3 , the variables B and C are changing, and A remains constant as a 0. Algebraically,

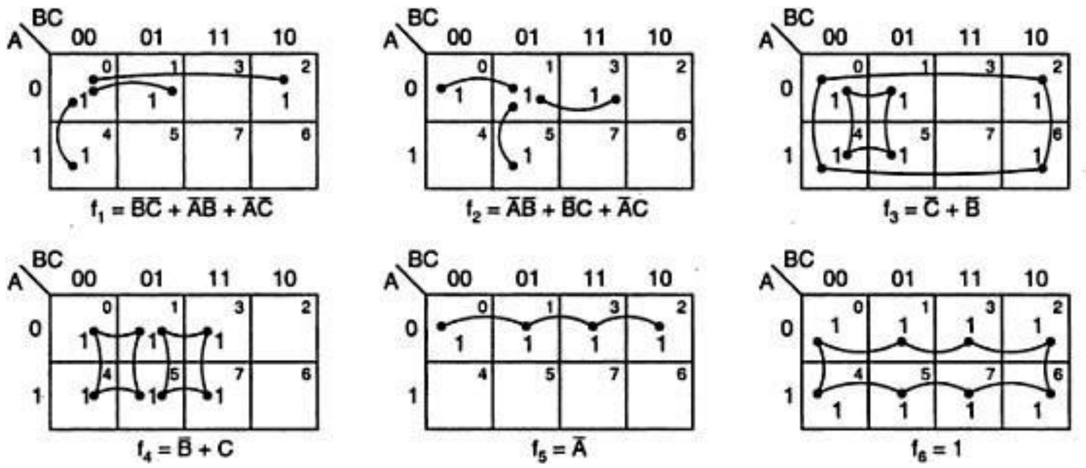
f_5= m_0+m_1+m_2+m_3

= **+C++**

= **(+C)+ B(C+)**

= **+ B**

= **(+B)=**

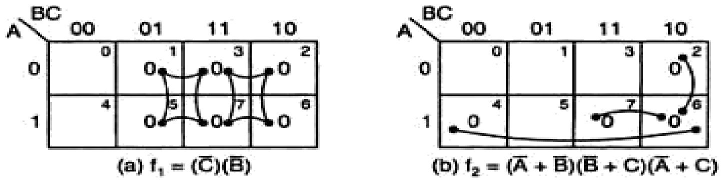


f_3 is read as + , because in the 4-square formed by m0,m2,m6, and m4, the variable A and B are changing , where as the variable C remains constant as a 0. So it is read as . In the 4-square formed by m_0, m1, m4, m_5, A and C are changing but B remains constant as a 0. So it is read as . So, the resultant expression for f_3 is the sum of these two, i.e., + .

f_1 is read as + + ,because in the 2-square formed by m_0 and m_4 , A is changing from a 0 to a 1. Whereas B and C remain constant as a 0. So it s read as . In the 2-square formed by m_0 and m_1, C is changing from a 0 to a 1, whereas A and B remain constant as a 0. So it is read as .In the 2-square formed by m_0 and m_2 , B is changing from a 0 to a 1 whereas A and C remain constant as a 0. So, it is read as . Therefore, the resultant SOP expression is

+ +

Some possible maxterm groupings and the corresponding minimal POS expressions read from the k-map are



In this figure, along the 4-square formed by M_1, M3, M7, M5, A and B are changing from a 0 to a 1, where as C remains constant as a 1. SO it is read as . Along the 4-squad formed by M_3, M2, M7, and M_6, variables A and C are changing from a 0 to a 1. But B remains constant as a 1. So it is read as . The minimal expression is the product of these two terms , i.e., f_1 = ( )( ).also in this figure, along the 2-square formed by M_4 and M_6 , variable B is changing from a 0 to a 1, while variable A remains constant as a 1 and variable C remains constant as a 0. SO, read it as

+C. Similarly, the 2-square formed by M_7 and M_6 is read as + , while the 2-square formed by M_2 and M_6 is read as +C. The minimal expression is the product of these sum terms, i.e, f_2

=(+)+(+)+(+C)

Ex:Reduce the expression f=∑m(0,2,3,4,5,6) using mapping and implement it in AOI logic as well as in NAND logic.The Sop k-map and its reduction , and the implementation of the minimal expression using AOI logic and the corresponding NAND logic are shown in figures below

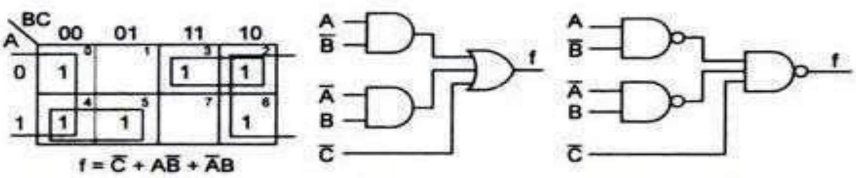
In SOP k-map, the reduction is done as:

1. **m_5 has only one adjacency m_4 , so combine m_5 and m_4 into a square. Along this 2-square A remains constant as 1 and B remains constant as 0 but C varies from 0 to 1. So read it as A .**
2. **m_3 has only one adjacency m_2 , so combine m_3 and m_2 into a square. Along this 2-square A remains constant as 0 and B remains constant as 1 but C varies from 1 to 0. So read it as B.**

.3 m_6 can form a 2-square with m_2 and m_4 can form a 2-square with m_0, but observe that by wrapping the map from left to right m_0, m_4 ,m_2 ,m_6 can form a 4-square. Out of these m_2 and m4 have already been combined but they can be utilized again. So make it. Along this 4-square, A is changing from 0 to 1 and B is also changing from 0 to 1 but C is remaining constant as 0. so read it as .

4. Write all the product terms in SOP form. So the minimal SOP expression is

f_{min}=



k-map AOI logic NAND logic

Four variable k-maps:

Four variable k-map expressions can have 2^4=16 possible combinations of input variables such as , ,------------ABCD with minterm designations m_0,m_1 ------------------ m_{15} respectively in SOP form & A+B+C+D, A+B+C+ ,---------- + + + with maxterms M_0,M_1, ---------

-

-M_{15} respectively in POS form. It has 2^4=16 squares or cells.The binary number designations of rows & columns are in the gray code. Here follows 01 & 10 follows 11 called Adjacency ordering.

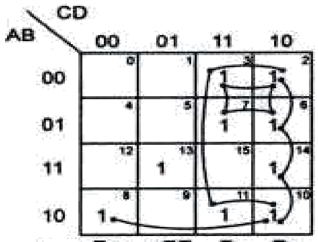
AB \ CD	00	01	11	10
00	0: $\bar{A}\bar{B}\bar{C}\bar{D}$	1: $\bar{A}\bar{B}\bar{C}D$	3: $\bar{A}\bar{B}CD$	2: $\bar{A}\bar{B}C\bar{D}$
01	4: $\bar{A}B\bar{C}\bar{D}$	5: $\bar{A}B\bar{C}D$	7: $\bar{A}BCD$	6: $\bar{A}BC\bar{D}$
11	12: $AB\bar{C}\bar{D}$	13: $AB\bar{C}D$	15: $ABCD$	14: $ABC\bar{D}$
10	8: $A\bar{B}\bar{C}\bar{D}$	9: $A\bar{B}\bar{C}D$	11: $A\bar{B}CD$	10: $A\bar{B}C\bar{D}$

AB \ CD	00	01	11	10
00	0: $A+B+C+D$	1: $A+B+C+\bar{D}$	3: $A+B+\bar{C}+\bar{D}$	2: $A+B+\bar{C}+D$
01	4: $A+\bar{B}+C+D$	5: $A+\bar{B}+C+\bar{D}$	7: $A+\bar{B}+\bar{C}+\bar{D}$	6: $A+\bar{B}+\bar{C}+D$
11	12: $\bar{A}+\bar{B}+C+D$	13: $\bar{A}+\bar{B}+C+\bar{D}$	15: $\bar{A}+\bar{B}+\bar{C}+\bar{D}$	14: $\bar{A}+\bar{B}+\bar{C}+D$
10	8: $\bar{A}+B+C+D$	9: $\bar{A}+B+C+\bar{D}$	11: $\bar{A}+B+\bar{C}+\bar{D}$	10: $\bar{A}+B+\bar{C}+D$

SOP form **POS form**

EX: Reduce using mapping the expression Σ m(2, 3, 6, 7, 8, 10, 11, 13, 14).

Start with the minterm with the least number of adjacencies. The minterm m_{13} has no adjacency. Keep it as it is. The m_8 has only one adjacency, m_{10}. Expand m_8 into a 2-square with m_{10}. The m_7 has two adjacencies, m_6 and m_3. Hence m_7 can be expanded into a 4-square with m_6, m_3 and m_2. Observe that, m_7, m_6, m_2, and m_3 form a geometric square. The m_{11} has 2 adjacencies, m_{10} and m_3. Observe that, m_{11}, m_{10}, m_3, and m_2 form a geometric square on wrapping the K-map. So expand m_{11} into a 4-square with m_{10}, m_3 and m_2. Note that, m_2 and m_3, have already become a part of the 4-square m_7, m_6, m_2, and m_3. But if m_{11} is expanded only into a 2-square with m_{10}, only one variable is eliminated. So m_2 and m_3 are used again to make another 4-square with m_{11} and m_{10} to eliminate two variables. Now only m_6 and m_{14} are left uncovered. They can form a 2-square that eliminates only one variable. Don't do that. See whether they can be expanded into a larger square. Observe that, m_2, m_6, m_{14}, and m_{10} form a rectangle. So m_6 and m_{14} can be expanded into a 4-square with m_2 and m_{10}. This eliminates two variables.



$f = AB\overline{C}D + A\overline{B}\overline{D} + \overline{A}C + \overline{B}C + C\overline{D}$

Five variable k-map:

Five variable k-map can have 2^5 =32 possible combinations of input variable as , E,--------ABCDE with minterms m_0, m_1-----m_{31} respectively in SOP & A+B+C+D+E, A+B+C+ ,---------- + + + + with maxterms M_0,M_1, ----------- M_{31} respectively in POS form. It has 2^5=32 squares or cells of the k-map are divided into 2 blocks of 16 squares each.The left block represents minterms from m_0 to m_{15} in which A is a 0, and the right block represents minterms from m_{16} to m_{31} in which A is 1.The 5-variable k-map may contain 2-squares, 4-squares , 8-squares , 16-squares or 32-squares involving these two blocks. Squares are also considered adjacent in these two blocks, if when superimposing one block on top of another, the squares coincide with one another.

Some possible 2-squares in a five-variable map are m_0, m_{16}; m_2, m_{18}; m_5, m_{21}; m_{15}, m_{31}; m_{11}, m_{27}.

Some possible 4-squares are m_0, m_2, m_{16}, m_{18}; m_0, m_1, m_{16}, m_{17}; m_0, m_4, m_{16}, m_{20}; m_{13}, m_{15}, m_{29}, m_{31}; m_5, m_{13}, m_{21}, m_{29}.

Some possible 8-squares are m_0, m_1, m_3, m_2, m_{16}, m_{17}, m_{19}, m_{18}; m_0, m_4, m_{12}, m_8, m_{16}, m_{20}, m_{28}, m_{24}; m_5, m_7, m_{13}, m_{15}, m_{21}, m_{23}, m_{29}, m_{31}.

The squares are read by dropping out the variables which change. Some possible

Grouping s is

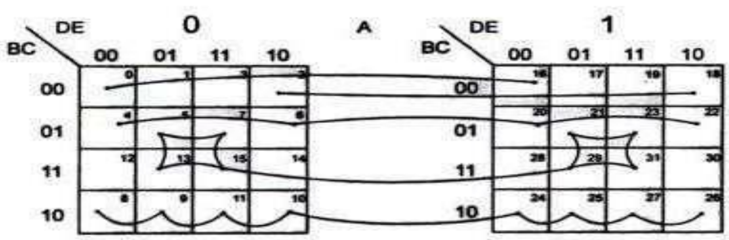
(a) m_0, m_{16} = $\overline{B}\overline{C}\overline{D}\overline{E}$ M_0, M_{16} = B + C + D + E

(b) m_2, m_{18} = $\overline{B}\overline{C}D\overline{E}$ M_2, M_{18} = B + C + $\overline{D}$ + E

(c) m_4, m_6, m_{20}, m_{22} = $\overline{B}C\overline{E}$ M_4, M_6, M_{20}, M_{22} = B + $\overline{C}$ + E

(d) m_5, m_7, m_{13}, m_{15}, m_{21}, m_{23}, m_{29}, m_{31} = CE M_5, M_7, M_{13}, M_{15}, M_{21}, M_{23}, M_{29}, M_{31} = $\overline{C}$ + $\overline{E}$

(e) m_8, m_9, m_{10}, m_{11}, m_{24}, m_{25}, m_{26}, m_{27} = $B\overline{C}$ M_8, M_9, M_{10}, M_{11}, M_{24}, M_{25}, M_{26}, M_{27} = $\overline{B}$ + C



Ex: F=∑m(0,1,4,5,6,13,14,15,22,24,25,28,29,30,31) is SOP

POS is F=πM(2,3,7,8,9,10,11,12,16,17,18,19,20,21,23,26,27)

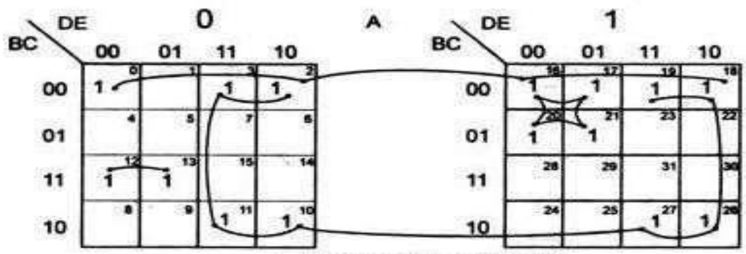
The real minimal expression is the minimal of the SOP and POS forms.

The reduction is done as

1. **There is no isolated 1s**
2. **M_{12} can go only with m_{13}. Form a 2-square which is read as A'BCD'**
3. **M_0 can go with m_2,m_{16} and m_{18} . so form a 4-square which is read as B'C'E'**
4. **M_{20},m_{21},m_{17} and m_{16} form a 4-square which is read as AB'D'**
5. **M2,m3,m18,m19,m10,m11,m26 and m27 form an 8-square which is read as C'd**
6. **Write all the product terms in SOP form.**

So the minimal expression is

F_{min}= A'BCD'+B'C'E'+AB'D'+C'D(16 inputs)



$f = \bar{A}BC\bar{D} + \bar{B}\bar{C}\bar{E} + A\bar{B}\bar{D} + \bar{C}D$

In the POS k-map ,the reduction is done as:

1. There are no isolated 0s

M_1 can go only with M_5. So, make a 2-square, which is read as $(A + B + D + \bar{E})$.

3. M_4 can go with M_5, M_7, and M_6 to form a 4-square, which is read as $(A + B + \bar{C})$.

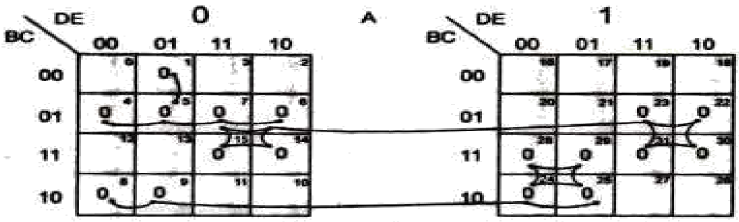
4.M_8

5. M_{28}

6.M_{30}

7. **Sum terms in POS form. So the minimal expression**

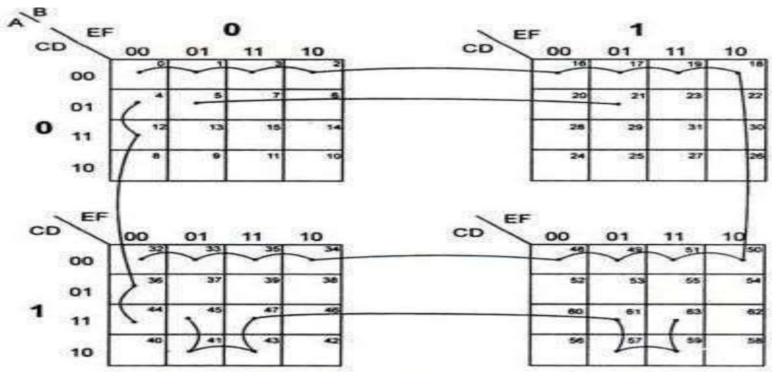
   in POS is F_{min}= A'BcD'+B'C'E'+AB'D'+C'D



$f = (A + B + D + \bar{E})(A + B + \bar{C})(\bar{B} + C + D)(\bar{A} + \bar{B} + D)(\bar{C} + \bar{D})$

Six variable k-map:

Six variable k-map can have 2^6 =64 combinations as , ,------------ABCDEF with minterms m_0, m_1-----m_{63} respectively in SOP & (A+B+C+D+E+F), ---------- (+ + + + +) with maxterms M_0,M_1, -----------M_{63} respectively in POS form. It has 2^6=64 squares or cells of the k-map are divided into 4 blocks of 16 squares each.



Some possible groupings in a six variable k-map

Don’t care combinations:For certain input combinations, the value of the output is unspecified either because the input combinations are invalid or because the precise value of the output is of no consequence. The combinations for which the value of experiments are not specified are called don‘t care combinations are invalid or because the precise value of the output is of no consequence. The combinations for which the value of expressions is not specified are called don‘t care combinations or Optional Combinations, such expressions stand incompletely specified. The output is a don‘t care for these invalid combinations.

Ex:In XS-3 code system, the binary states 0000, 0001, 0010,1101,1110,1111 are unspecified. & never occur called don‘t cares.

A standard SOP expression with don‘t cares can be converted into a standard POS form by keeping the don‘t cares as they are & writing the missing minterms of the SOP form as the maxterms of the POS form viceversa.

Don‘t cares denoted by ‗X‘ or ‗φ‘

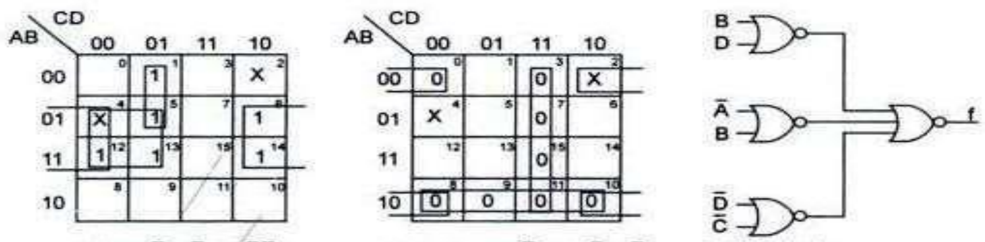
Ex:f=∑m(1,5,6,12,13,14)+d(2,4)

Or f=π M(0,3,7,9,10,11,15).πd(2,4)

SOP minimal form f_{min}= +B +

POS minimal form f_{min}=(B+D)(+B)(+D)

= **++ ++(+**



(a) $f = B\bar{C} + \bar{B}D + \bar{A}\bar{C}D$ (b) $f = (B + D)(\bar{A} + B)(\bar{C} + \bar{D})$ (c) NOR logic

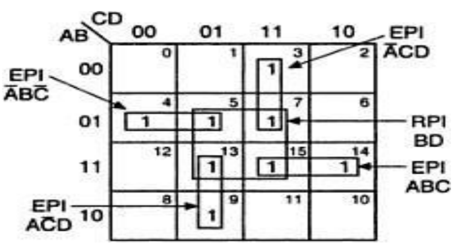
Prime implicants, Essential Prime implicants, Redundant prime implicants:

Each square or rectangle made up of the bunch of adjacent minterms is called a subcube. Each of these subcubes is called a Prime implicant (PI). The PI which contains at leastone which cannot be covered by any other prime implicants is called as Essential Prime implicant (EPI).The PI whose each 1 is covered at least by one EPI is called a Redundant Prime implicant (RPI). A PI which is neither an EPI nor a RPI is called a Selective Prime implicant (SPI).

The function has unique MSP comprising EPI is

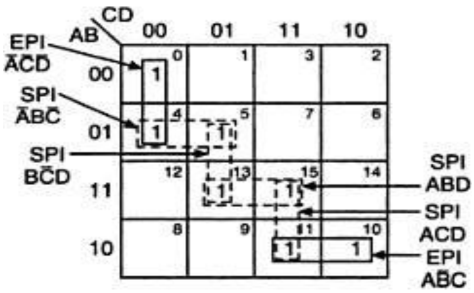
F(A,B,C,D)= CD+ABC+A D + B

The RPI ‗BD' may be included without changing the function but the resulting expression would not be in minimal SOP(MSP) form.



Essential and Redundant Prime Implicants

F(A,B,C,D)=∑m(0,4,5,10,11,13,15) SPI are marked by dotted squares, shows MSP form of a function need not be unique.



Essential and Selective Prime Implicants

Here, the MSP form is obtained by including two EPI‘s & selecting a set of SPI‘s to cover remaining uncovered minterms 5,13,15. & these can be covered as

(A) (4,5) &(13,15) ----------B +ABD
(B) (5,13) & (13,15) --------B D+ABD
(C) (5,13) & (15,11) -------B D+ACD

F(A,B,C,D)= +A C---------EPI‘s + B +ABD

(OR) F(A,B,C,D)= +A C---------EPI‘s + B D+ABD

(OR) F(A,B,C,D)= +A C---------EPI‘s + B D+ACD

False PI's Essential False PI's, Redundant False PI's & Selective False PI's:

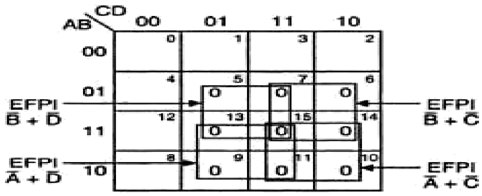
The maxterms are called falseminterms. The PI‘s is obtained by using the maxterms are called False PI‘s (FPI). The FPI which contains at least one ‗0‘ which can‘t be covered by only other FPI is called an Essential False Prime implicant (ESPI)

F(A,B,C,D)= ∑m(0,1,2,3,4,8,12)

=π M(5,6,7,9,10,11,13,14,15)

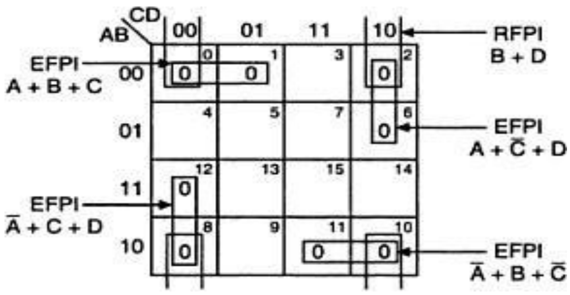
F_{min}= (+)(+)(+)(+)

All the FPI, EFPI‘s as each of them contain atleast one ‗0‘ which can‘t be covered by any other FPI



Essential False Prime implicants

Consider Function F(A,B,C,D)= π M(0,1,2,6,8,10,11,12)



Essential and Redundant False Prime Implicants

Mapping when the function is not expressed in minterms (maxterms):

An expression in k-map must be available as a sum (product) of minterms (maxterms). However if not so expressed, it is not necessary to expand the expression algebraically into its minterms (maxterms). Instead, expansion into minterms (maxterms) can be accomplished in the process of entering the terms of the expression on the k-map.

Limitations of Karnaugh maps:

Convenient as long as the number of variables does not exceed six.
Manual technique, simplification process is heavily dependent on the human abilities.

Quine-Mccluskey Method:

It also known as Tabular method. It is more systematic method of minimizing expressions of even larger number of variables. It is suitable for hand computation as well as computation by machines i.e., programmable. . The procedure is based on repeated application of the combining theorem.

PA+P =P (P is set of literals) on all adjacent pairs of terms, yields the set of all PI's from which a minimal sum may be selected.

Consider expression

∑m(0,1,4,5)= + C+A +A C

First, second terms & third, fourth terms can be combined

(+)+ (C+)= +A

Reduced to

(+)=

The same result can be obtained by combining m_0& m_4 & m_1&m_5 in first step & resulting terms in the second step .

Procedure:

Decimal Representation Don't

cares

PI chart EPI

Dominating Rows & Columns

Determination of Minimal expressions in comples cases.

Branching Method:

EXAMPLE 3.29 Obtain the set of prime implicants for the Boolean expression

$f = \Sigma\, m(0, 1, 6, 7, 8, 9, 13, 14, 15)$ using the tabular method.

Solution

Group the minterms in terms of the number of 1s present in them and write their binary designations. The procedure to obtain the prime implicants is shown in Table 3.3.

Table 3.3 Example 3.29

Column 1			Column 2		Column 3
	Minterm	**Binary designation**		**A B C D**	**A B C D**
Index 0	0	0 0 0 0 ✓	0, 1 (1)	0 0 0 – ✓	0, 1, 8, 9 (1, 8) – 0 0 – Q
Index 1	1	0 0 0 1 ✓	0, 8 (8)	– 0 0 0 ✓	... ...
	8	1 0 0 0 ✓	1, 9 (8)	– 0 0 1 ✓	... ...
Index 2	6	0 1 1 0 ✓	8, 9 (1)	1 0 0 – ✓	6, 7, 14, 15 (1, 8) – 1 1 – P
	9	1 0 0 1 ✓	6, 7 (1)	0 1 1 – ✓	
Index 3	7	0 1 1 1 ✓	6, 14 (8)	– 1 1 0 ✓	
	13	1 1 0 1 ✓	9, 13 (4)	1 – 0 1 S	
	14	1 1 1 0 ✓	7, 15 (8)	– 1 1 1 ✓	
Index 4	15	1 1 1 1 ✓	13, 15 (2)	1 1 – 1 R	
			14, 15 (1)	1 1 1 – ✓	

Comparing the terms of group 1 with the terms of group 2 in column 2, the terms 0, 1 (1), i.e. 000– and 8, 9 (1), i.e. 100– are combined to form 0, 1, 8, 9 (1, 8), i.e. –00–. Record it in group 1 of column 3 and check off 0, 1 (1), i.e. 000–, and 8, 9 (1), i.e. 100– of column 2. The terms 0, 8 (8), i.e. –000 and 1, 9 (8), i.e. –001 are combined to form 0, 1, 8, 9 (1, 8), i.e. –00–. This has already been recorded in column 3. So, no need to record again. Check off 0, 8 (8), i.e. –000 and 1, 9 (8), i.e. –001 of column 2. Draw a line below 0, 1, 8, 9 (1, 8), i.e. –00–. This is the only term in group 1 of column 3. No term of group 2 of column 2 can be combined with any term of group 3 of column 2. So, no entries are made in group 2 of column 2.

Comparing the terms of group 3 of column 2 with the terms of group 4 of column 2, the terms 6, 7 (1), i.e. 011–, and 14, 15 (1), i.e. 111– are combined to form 6, 7, 14, 15 (1, 8), i.e. –11–. Record it in group 3 of column 3 and check off 6, 7 (1), i.e. 011– and 14, 15 (1), i.e. 111– of column 2. The terms 6, 14 (8), i.e. –110 and 7, 15 (8), i.e. –111 are combined to form 6, 7, 14, 15 (1, 8), i.e. –11–. This has already been recorded in column 3; so, check off 6, 14 (8), i.e. –110 and 7, 15 (8), i.e. –111 of column 2.

Observe that the terms 9, 13 (4), i.e. 1–01 and 13, 15 (2), i.e. 11–1 cannot be combined with any other terms. Similarly in column 3, the terms 0, 1, 8, 9 (1, 8), i.e. –00– and 6, 7, 14, 15 (1, 8), i.e. –11– cannot also be combined with any other terms. So, these 4 terms are the prime implicants.

The terms, which cannot be combined further, are labelled as P, Q, R, and S. These form the set of prime implicants.

EX:

Obtain the minimal expression for f = Σ m(1, 2, 3, 5, 6, 7, 8, 9, 12, 13, 15) using the tabular method.

Solution

The procedure to obtain the set of prime implicants is illustrated in Table 3.4.

Table 3.4 Example 3.30

Step 1		Step 2	Step 3	
Index 1	1 ✓	1, 3 (2) ✓	1, 3, 5, 7 (2, 4)	T
	2 ✓	1, 5 (4) ✓	1, 5, 9, 13 (4, 8)	S
	8 ✓	1, 9 (8) ✓	2, 3, 6, 7 (1, 4)	R
Index 2	3 ✓	2, 3 (1) ✓	8, 9, 12, 13 (1, 4)	Q
	5 ✓	2, 6 (4) ✓	5, 7, 13, 15 (2, 8)	P
	6 ✓	8, 9 (1) ✓		
	9 ✓	8, 12 (4) ✓		
	12 ✓	3, 7 (4) ✓		
Index 3	7 ✓	5, 7 (2) ✓		
	13 ✓	5, 13 (8) ✓		
Index 4	15 ✓	6, 7 (1) ✓		
		9, 13 (4) ✓		
		12, 13 (1) ✓		
		7, 15 (8) ✓		
		13, 15 (2) ✓		

Tow nail-combinablc terms P. Q. R. S and T arc accorded as prime implicants.

P—+5.7.13,IS(2.8)- X I X1=BD

(Liberals with weijzhcs 2 anJ 8, i.e. C auct A are deleted. The lowesE minterm is mg(3 = 4 + I). 5o,]item1s wijh weights 4 ance I , i.e. B and D aze present in non-comp1<menled form. So, zead it as BD.)

p+s,s, iz, nn,4)= i x n* =AC

{Liberals with weights 1 aztd 4, i.e. D and B arc rlr\etrd. The lowest mintcrm ts mg. So, literal with weight 8 is present in non-complemented form and literal wit:ft woighl 2 is present in couipJcmcnicd form. Sn, reed it as AC.j

R —+ 2. ?, d. 7 (I, 4} = O X J X = AC

(Litertits with w'eights I and 4, i.e. D and B are rlel«irrt- The lnwest miniemi is mg, So, literal with weighl 2 is present in non-complemented form and literal with weight 8 is present in complcnicnted form. So, rrad it as AC.I

S—+ I,:2,9, 13(4,B) XX0I=CD

(Li\era\s wi\h weights 4 end 6. i.e. B and A are deleted. The lowest mintcrm is m,. SD. lltCF8l with wcigh\ 1 is pream in non-complemented form and literal witb weight 2 is present in complemented form. So, read it cs CD.)

T—+1.3,3.7{2.4}=0XX J =AD

fLitcrals with weigfit,s 2 and 4. i.e. C and B arc deieted. The lowest Minicrni is I . Sri. literal with weight 1 is present in non-complemented form und literal with weight 8 is present in complemented form. So. read ii as AD. I

Thc primc implicant chart of- the expression

F — Z m(l. 2, 3, 5. 6, 7, 8, V. I2, J 3. I 'i)

is cx shown in Table 3.5. It cnnsisti of 11 columns corresponding tn the number of minterms rind J rows corresponding iu the prime implicants P, Q. R. S, and T teneraied. Row R contains four xs nt the intersections with col umns 2, 3, 6, and 7. becau se these minicrms as covered b3• the prime implicant ft. A ruw ix said to cover the columns in whic h it hits xu. The problem now is to xclect a minimo) subset of prime implicants, such thni each cnlumn contains at lcast one x in the rows corresponding io the sr lr twfl suti.set and the tntnl numher of titerols in the prince iniplicants selected is a.s .small a.s prissible. Tnem equirements guaronicc ihat the nuriibc r nf unions of the selected prime impl icanks is et}ual in the nriginal number r›f ni inicrrris and thai, nn uther exprc+sinn cnnluiriiog tier liberals can be You nd.

Talile 3J **Eerie iplc .3.3tl. Prime implicani chin**

	1	2	3	fi	6	7	s	9	I*	I3	15
”P —• 1. 7. J 3. I3 12.6›				x		x				n	x
“Q —+ 8. 9. 12. TI (1.4)							×	×	×	×	
’R -+ 2. T 6. 7 N . 4j											
S 1, 5. 9, I? f4. F•›	×			×				×		×	
T + 1. 3. 5. 7 I 2, 4 I	×		×	×		×					

In the prime implicant chart of Table 3.5, m_2 and m_6 are covered by R only. So, R is an essential prime implicant. So, check off all the minterms covered by it, i.e. m_2, m_3, m_6, and m_7. Q is also an essential prime implicant because only Q covers m_8 and m_{12}. Check off all the minterms covered by it, i.e. m_8, m_9, m_{12}, and m_{13}. P is also an essential prime implicant, because m_{15} is covered only by P. So check off m_{15}, m_5, m_7, and m_{13} covered by it. Thus, only minterm 1 is not covered. Either row S or row T can cover it and both have the same number of literals. Thus, two minimal expressions are possible.

$$P + Q + R + S = BD + A\bar{C} + \bar{A}C + \bar{C}D$$

or

$$P + Q + R + T = BD + A\bar{C} + \bar{A}C + \bar{A}D$$

MODULE-III:

Combinational Logic Circuits

Combinational Logic Design

Logic circuits for digital systems may be combinational or sequential. The output of a combinational circuit depends on its present inputs only .Combinational circuit processing operation fully specified logically by a set of Boolean functions .A combinational circuit consists of input variables, logic gates and output variables.Both input and output data are represented by signals, i.e., they exists in two possible values. One is logic –1 and the other logic 0.

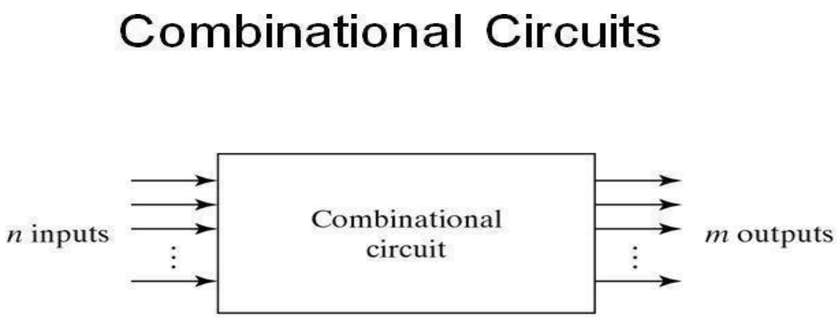


Fig. Block Diagram of Combinational Circuit

For n input variables,there are 2^n possible combinations of binary input variables .For each possible input Combination ,there is one and only one possible output combination.A combinational circuit can be described by m Boolean functions one for each output variables.Usually the input s comes from flip-flops and outputs goto flip-flops.

Design Procedure:

1.The problem is stated
2. The number of available input variables and required output variables is determined. 3.The input and output variables are assigned letter symbols.
4.The truth table that defines the required relationship between inputs and outputs is derived.
5.The simplified Boolean function for each output is obtained.
6.The logic diagram is drawn.

Adders:

Digital computers perform variety of information processing tasks,the one is arithmetic operations.And the most basic arithmetic operation is the addition of two binary digits.i.e, 4 basic possible operations are:

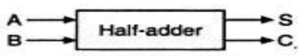
0+0=0,0+1=1,1+0=1,1+1=10

The first three operations produce a sum whose length is one digit, but when augends and addend bits are equal to 1,the binary sum consists of two digits.The higher significant bit of this result is called a carry.A combinational circuit that performs the addition of two bits is called a half-adder. One that performs the addition of 3 bits (two significant bits & previous carry) is called a full adder.& 2 half adder can employ as a full-adder.

The Half Adder: A Half Adder is a combinational circuit with two binary inputs (augends and addend bits and two binary outputs (sum and carry bits.) It adds the two inputs (A and B) and produces the sum (S) and the carry (C) bits. It is an arithmetic operation of addition of two single bit words.

Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(a) Truth table



(b) Block diagram

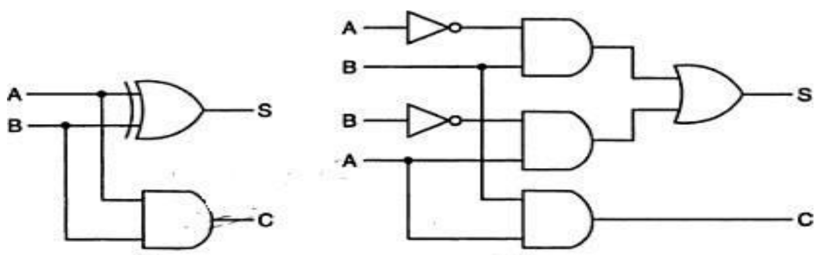
The Sum(S) bit and the carry (C) bit, according to the rules of binary addition, the sum (S) is the X-OR of A and B (It represents the LSB of the sum). Therefore,

S=A + B= $A \oplus B$

The carry (C) is the AND of A and B (it is 0 unless both the inputs are 1).Therefore,

C=AB

A half-adder can be realized by using one X-OR gate and one AND gate a



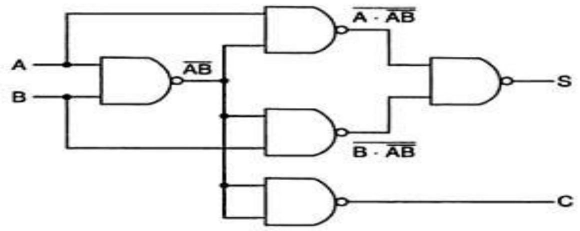
(a) (b)

Logic diagrams of half-adder

NAND LOGIC:

$$
\begin{aligned}
S &= A\overline{B} + \overline{A}B = A\overline{B} + A\overline{A} + \overline{A}B + B\overline{B} \\
&= A(\overline{A} + \overline{B}) + B(\overline{A} + \overline{B}) \\
&= A \cdot \overline{AB} + B \cdot \overline{AB} \\
&= \overline{\overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}}
\end{aligned}
$$

$$C = AB = \overline{\overline{AB}}$$

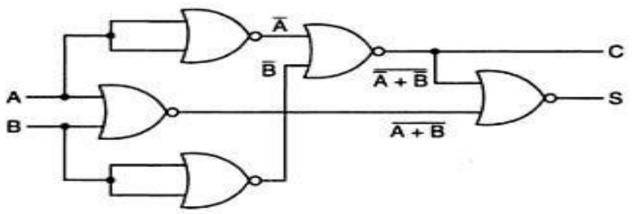


Logic diagram of a half-adder using only 2-input NAND gates.

NOR Logic:

$$
\begin{aligned}
S &= A\overline{B} + \overline{A}B = A\overline{B} + A\overline{A} + \overline{A}B + B\overline{B} \\
&= A(\overline{A} + \overline{B}) + B(\overline{A} + \overline{B}) \\
&= (A + B)(\overline{A} + \overline{B}) \\
&= \overline{\overline{A + B} + \overline{\overline{A} + \overline{B}}}
\end{aligned}
$$

$$C = AB = \overline{\overline{AB}} = \overline{\overline{A} + \overline{B}}$$



Logic diagram of a half-adder using only 2-input NOR gates.

The Full Adder:

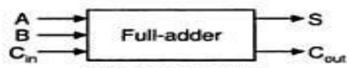
A Full-adder is a combinational circuit that adds two bits and a carry and outputs a sum bit and a carry bit. To add two binary numbers, each having two or more bits, the LSBs can be added by using a half-adder. The carry resulted from the addition of the LSBs is carried over to the next significant column and added to the two bits in that column. So, in the second and higher columns, the two data bits of that column and the carry bit generated from the addition in the previous column need to be added.

The full-adder adds the bits A and B and the carry from the previous column called the carry-in C_{in} and outputs the sum bit S and the carry bit called the carry-out C_{out} . The variable S gives the value of the least significant bit of the sum. The variable C_{out} gives the output carry.The

eight rows under the input variables designate all possible combinations of 1s and 0s that these variables may have. The 1s and 0s for the output variables are determined from the arithmetic sum of the input bits. When all the bits are 0s , the output is 0. The S output is equal to 1 when only 1 input is equal to 1 or when all the inputs are equal to 1. The C_{out} has a carry of 1 if two or three inputs are equal to 1.

Inputs			Sum	Carry
A	B	C_{in}	S	C_{out}
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

(a) Truth table



(b) Block diagram

Full-adder.

From the truth table, a circuit that will produce the correct sum and carry bits in response to every possible combination of A,B and C_{in} is described by

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

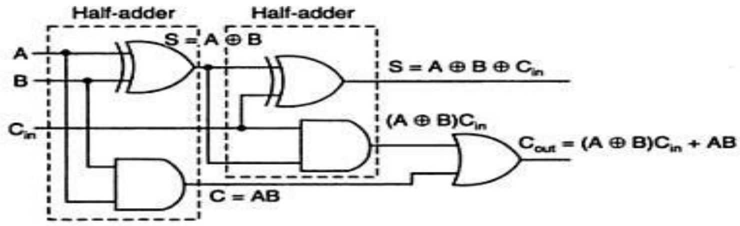
$$C_{out} = \bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in}$$

and

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

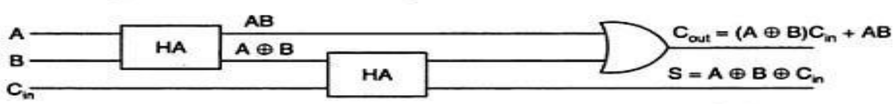
$$C_{out} = AC_{in} + BC_{in} + AB$$

The sum term of the full-adder is the X-OR of A,B, and C_{in}, i.e, the sum bit the modulo sum of the data bits in that column and the carry from the previous column. The logic diagram of the full-adder using two X-OR gates and two AND gates (i.e, Two half adders) and one OR gate is



Logic diagram of a full-adder using two half-adders.

The block diagram of a full-adder using two half-adders is



Block diagram of a full-adder using two half-adders.

Even though a full-adder can be constructed using two half-adders, the disadvantage is that the bits must propagate through several gates in accession, which makes the total propagation delay greater than that of the full-adder circuit using AOI logic.

The Full-adder neither can also be realized using universal logic, i.e., either only NAND gates or only NOR gates as

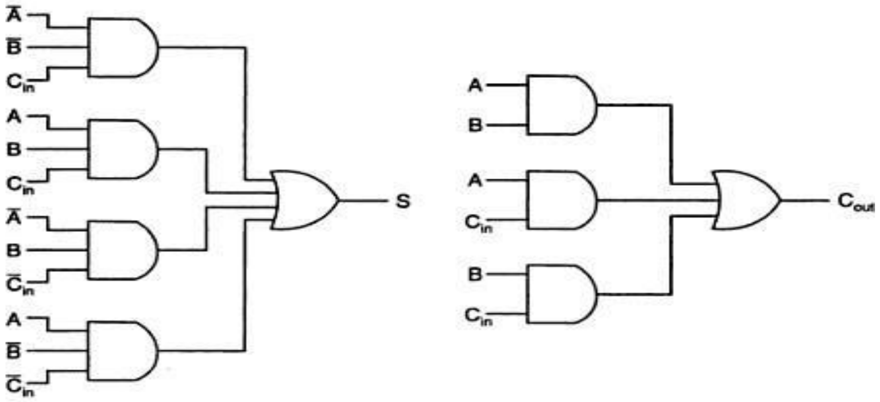
$$A \oplus B = \overline{\overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}}$$

Then

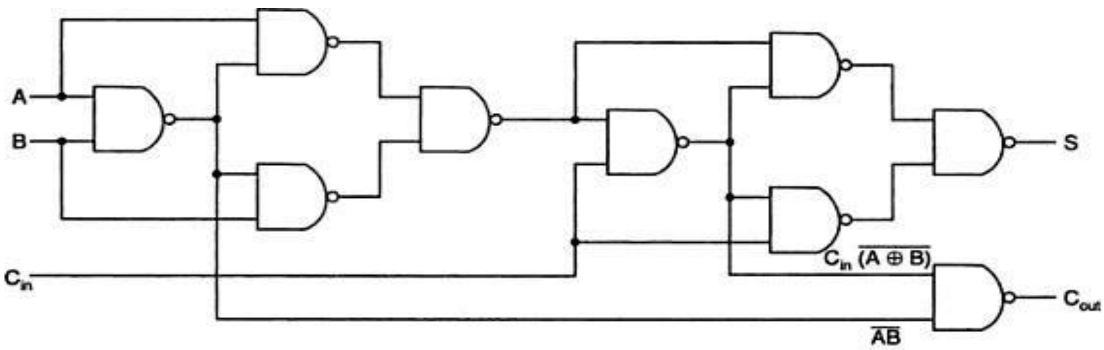
$$S = A \oplus B \oplus C_{in} = \overline{\overline{(A \oplus B) \cdot \overline{(A \oplus B)C_{in}}} \cdot \overline{C_{in} \cdot \overline{(A \oplus B)C_{in}}}}$$

NAND Logic:

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



Sum and carry bits of a full-adder using AOI logic.



Logic diagram of a full-adder using only 2-input NAND gates.

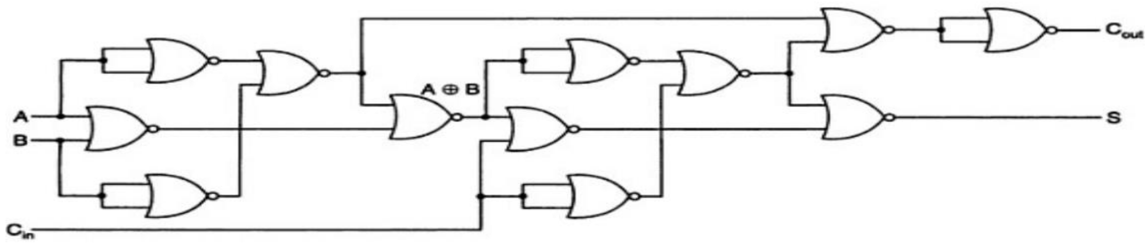
NOR Logic:

$$A \oplus B = \overline{\overline{(A + B)} + \overline{\overline{A} + \overline{B}}}$$

Then

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

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



Logic diagram of a full-adder using only 2-input NOR gates.

Subtractors:

The subtraction of two binary numbers may be accomplished by taking the complement of the subtrahend and adding it to the minuend. By this, the subtraction operation becomes an addition operation and instead of having a separate circuit for subtraction, the adder itself can be used to perform subtraction. This results in reduction of hardware. In subtraction, each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a difference bit. If the minuend bit is smaller than the subtrahend bit, a 1 is borrowed from the next significant position., that has been borrowed must be conveyed to the next higher pair of bits by means of a signal coming out (output) of a given stage and going into (input) the next higher stage.

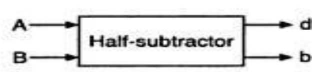
The Half-Subtractor:

A Half-subtractor is a combinational circuit that subtracts one bit from the other and produces the difference. It also has an output to specify if a 1 has been borrowed. . It is used to subtract the LSB of the subtrahend from the LSB of the minuend when one binary number is subtracted from the other.

A Half-subtractor is a combinational circuit with two inputs A and B and two outputs d and b. d indicates the difference and b is the output signal generated that informs the next stage that a 1 has been borrowed. When a bit B is subtracted from another bit A, a difference bit (d) and a borrow bit (b) result according to the rules given as

Inputs		Outputs	
A	B	d	b
0	0	0	0
1	0	1	0
1	1	0	0
0	1	1	1

(a) Truth table



(b) Block diagram

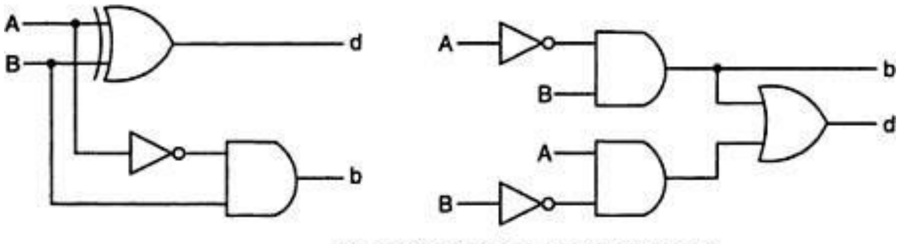
Half-subtractor.

The output borrow b is a 0 as long as A≥B. It is a 1 for A=0 and B=1. The d output is the result of the arithmetic operation 2b+A-B.

A circuit that produces the correct difference and borrow bits in response to every possible combination of the two 1-bit numbers is , therefore ,

d=A + B= A ⊕ B and b= B

That is, the difference bit is obtained by X-OR ing the two inputs, and the borrow bit is obtained by ANDing the complement of the minuend with the subtrahend.Note that logic for this exactly the same as the logic for output S in the half-adder.



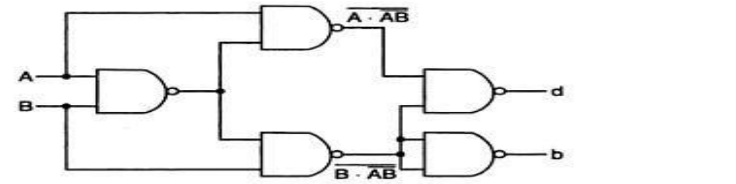
Logic diagrams of a half-subtractor.

A half-substractor can also be realized using universal logic either using only NAND gates or using NOR gates as:

NAND Logic:

$$d = A \oplus B = \overline{\overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}}$$

$$b = \overline{A}B = B(\overline{A} + \overline{B}) = B(\overline{AB}) = \overline{\overline{B \cdot \overline{AB}}}$$



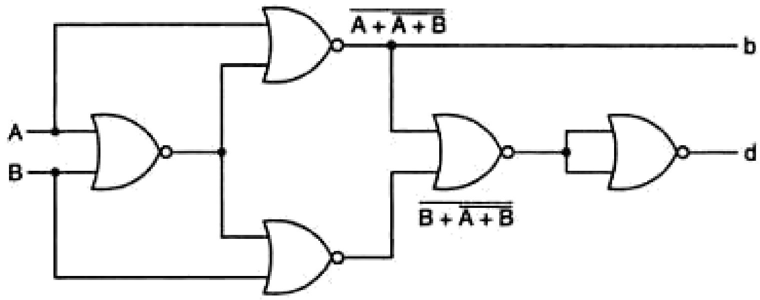
Logic diagram of a half-subtractor using only 2-input NAND gates.

NOR Logic:

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

$$= \overline{B}(A + B) + \overline{A}(A + B) = \overline{B + \overline{A + B}} + \overline{A + \overline{A + B}}$$

$$d = \overline{A}B = \overline{A}(A + B) = \overline{\overline{\overline{A}(A + B)}} = \overline{A + \overline{(A + B)}}$$



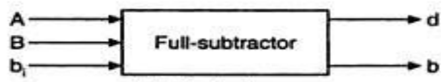
Logic diagram of a half-subtractor using only 2-input NOR gates.

The Full-Subtractor:

The half-subtractor can be only for LSB subtraction. IF there is a borrow during the subtraction of the LSBs, it affects the subtraction in the next higher column; the subtrahend bit is subtracted from the minuend bit, considering the borrow from that column used for the subtraction in the preceding column. Such a subtraction is performed by a full-subtractor. It subtracts one bit (B) from another bit (A) , when already there is a borrow b_i from this column for the subtraction in the preceding column, and outputs the difference bit (d) and the borrow bit(b) required from the next d and b. The two outputs present the difference and output borrow. The 1s and 0s for the output variables are determined from the subtraction of A-B-b_i.

Inputs			Difference	Borrow
A	B	b_i	d	b
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

(a) Truth table



(b) Block diagram

Full-subtractor.

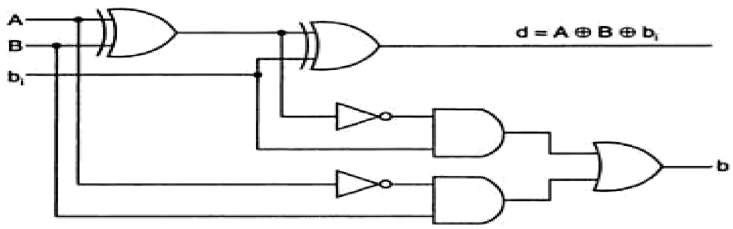
From the truth table, a circuit that will produce the correct difference and borrow bits in response to every possiblecombinations of A,B and b_i is

$$d = \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + A\bar{B}\bar{b}_i + ABb_i$$
$$= b_i(AB + \bar{A}\bar{B}) + \bar{b}_i(A\bar{B} + \bar{A}B)$$
$$= b_i(\overline{A \oplus B}) + \bar{b}_i(A \oplus B) = A \oplus B \oplus b_i$$

and

$$b = \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + \bar{A}Bb_i + ABb_i = \bar{A}B(b_i + \bar{b}_i) + (AB + \bar{A}\bar{B})b_i$$
$$= \bar{A}B + (\overline{A \oplus B})b_i$$

A full-subtractor can be realized using X-OR gates and AOI gates as



Logic diagram of a full-subtractor.

The full subtractor can also be realized using universal logic either using only NAND gates or using NOR gates as:

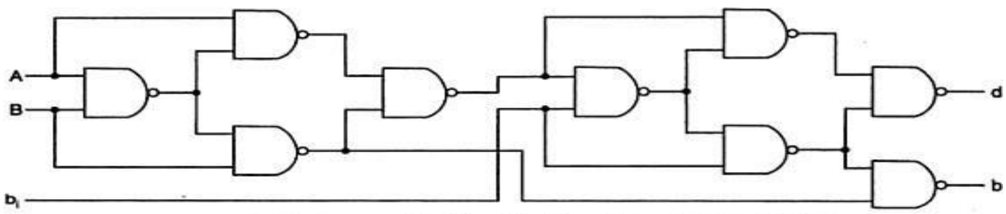
NAND Logic:

$$d = A \oplus B \oplus b_i = \overline{\overline{(A \oplus B) \oplus b_i}} = \overline{\overline{(A \oplus B)\overline{(A \oplus B)b_i}} \cdot \overline{b_i\overline{(A \oplus B)b_i}}}$$

$$b = \bar{A}B + b_i(\overline{A \oplus B}) = \overline{\overline{\bar{A}B + b_i(\overline{A \oplus B})}}$$

$$= \overline{\overline{\bar{A}B} \cdot \overline{b_i(\overline{A \oplus B})}} = \overline{\overline{B(\bar{A} + \bar{B})} \cdot \overline{b_i(\bar{b}_i + (\overline{A \oplus B}))}}$$

$$= \overline{\overline{B \cdot \overline{AB}} \cdot \overline{b_i[\overline{b_i \cdot (A \oplus B)}]}}$$



Logic diagram of a full-subtractor using only 2-input NAND gates.

NOR Logic:

$$d = A \oplus B \oplus b_i = \overline{\overline{(A \oplus B) \oplus b_i}}$$

$$= \overline{(A \oplus B)b_i + (\overline{A \oplus B})\bar{b}_i}$$

$$= \overline{[(A \oplus B) + (\overline{A \oplus B})\bar{b}_i][b_i + (\overline{A \oplus B})\bar{b}_i]}$$

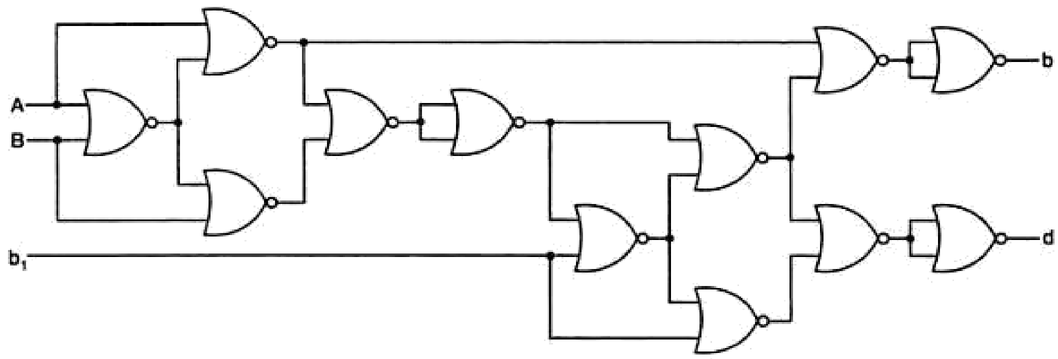
$$= \overline{\overline{(A \oplus B) + \overline{(A \oplus B) + b_i}} + \overline{b_i + \overline{(A \oplus B) + b_i}}}$$

$$= \overline{\overline{\overline{\overline{(A \oplus B) + \overline{(A \oplus B) + b_i}} + \overline{b_i + \overline{(A \oplus B) + b_i}}}}}$$

$$b = \bar{A}B + b_i(\overline{A \oplus B})$$

$$= \bar{A}(A + B) + (\overline{A \oplus B})[(A \oplus B) + b_i]$$

$$= \overline{\overline{\overline{\overline{A + \overline{(A + B)}} + \overline{(A \oplus B) + \overline{(A \oplus B) + b_i}}}}}$$

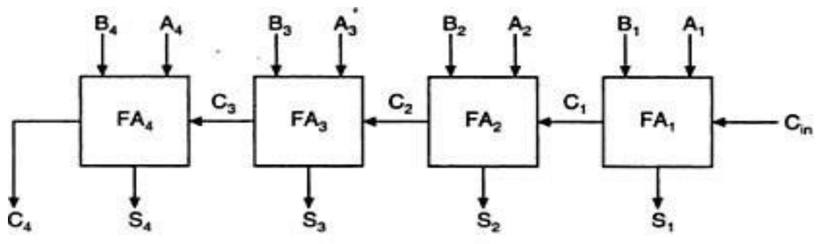


Logic diagram of a full subtractor using only 2-input NOR gates.

Binary Parallel Adder:

A binary parallel adder is a digital circuit that adds two binary numbers in parallel form and produces the arithmetic sum of those numbers in parallel form. It consists of full adders connected in a chain , with the output carry from each full-adder connected to the input carry of the next full-adder in the chain.

The interconnection of four full-adder (FA) circuits to provide a 4-bit parallel adder. The augends bits of A and addend bits of B are designated by subscript numbers from right to left, with subscript 1 denoting the lower –order bit. The carries are connected in a chain through the full-adders. The input carry to the adder is C_{in} and the output carry is C_4. The S output generates the required sum bits. When the 4-bit full-adder circuit is enclosed within an IC package, it has four terminals for the augends bits, four terminals for the addend bits, four terminals for the sum bits, and two terminals for the input and output carries. AN n-bit parallel adder requires n-full adders. It can be constructed from 4-bit, 2-bit and 1-bit full adder ICs by cascading several packages. The output carry from one package must be connected to the input carry of the one with the next higher –order bits. The 4-bit full adder is a typical example of an MSI function.



Logic diagram of a 4-bit binary parallel adder.

Ripple carry adder:

In the parallel adder, the carry –out of each stage is connected to the carry-in of the next stage. The sum and carry-out bits of any stage cannot be produced, until sometime after the carry-in of that stage occurs. This is due to the propagation delays in the logic circuitry,

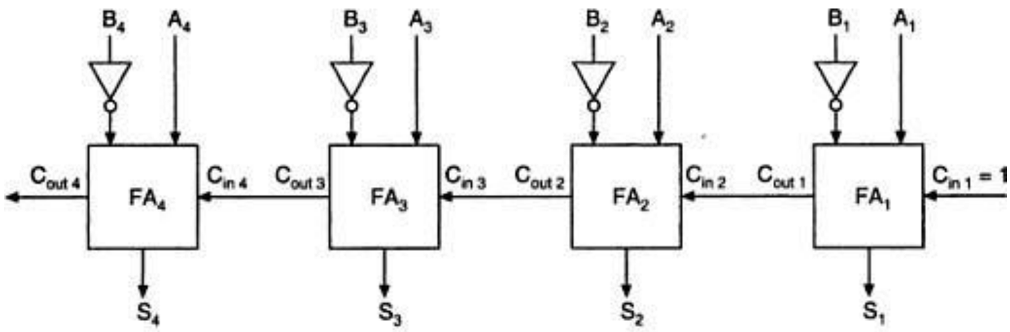
which lead to a time delay in the addition process. The carry propagation delay for each full-adder is the time between the application of the carry-in and the occurrence of the carry-out.

The 4-bit parallel adder, the sum (S_1) and carry-out (C_1) bits given by FA_1 are not valid, until after the propagation delay of FA_1. Similarly, the sum S_2 and carry-out (C_2) bits given by FA_2 are not valid until after the cumulative propagation delay of two full adders (FA_1 and FA_2) , and so on. At each stage ,the sum bit is not valid until after the carry bits in all the preceding stages are valid. Carry bits must propagate or ripple through all stages before the most significant sum bit is valid. Thus, the total sum (the parallel output) is not valid until after the cumulative delay of all the adders.

The parallel adder in which the carry-out of each full-adder is the carry-in to the next most significant adder is called a ripple carry adder.. The greater the number of bits that a ripple carry adder must add, the greater the time required for it to perform a valid addition. If two numbers are added such that no carries occur between stages, then the add time is simply the propagation time through a single full-adder.

4- Bit Parallel Subtractor:

The subtraction of binary numbers can be carried out most conveniently by means of complements , the subtraction A-B can be done by taking the 2's complement of B and adding it to A . The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits. The 1's complement can be implemented with inverters as

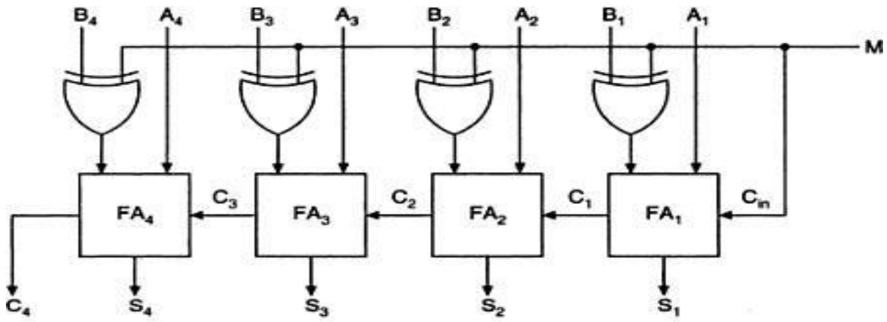


Logic diagram of a 4-bit parallel subtractor.

Binary-Adder Subtractor:

A 4-bit adder-subtractor, the addition and subtraction operations are combined into one circuit with one common binary adder. This is done by including an X-OR gate with each full-adder. The mode input M controls the operation. When M=0, the circuit is an adder, and when M=1, the circuit becomes a subtractor. Each X-OR gate receives input M and one of the inputs of B. When M=0, $B \oplus 0 = B$ .The full-adder receives the value of B , the input carry is 0

and the circuit performs A+B. when $B \oplus 1 = B'$ and C_1=1. The B inputs are complemented and a 1 is through the input carry. The circuit performs the operation A plus the 2's complement of B.



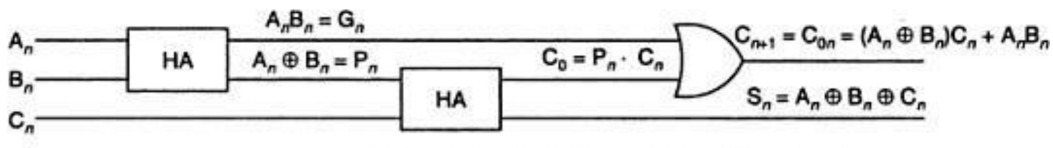
Logic diagram of a 4-bit binary adder-subtractor.

The Look-Ahead –Carry Adder:

In parallel-adder,the speed with which an addition can be performed is governed by the time required for the carries to propagate or ripple through all of the stages of the adder. The look-ahead carry adder speeds up the process by eliminating this ripple carry delay. It examines all the input bits simultaneously and also generates the carry-in bits for all the stages simultaneously.

The method of speeding up the addition process is based on the two additional functions of the full-adder, called the carry generate and carry propagate functions.

Consider one full adder stage; say the nth stage of a parallel adder as shown in fig. we know that is made by two half adders and that the half adder contains an X-OR gate to produce the sum and an AND gate to produce the carry. If both the bits A_n and B_n are 1s, a carry has to be generated in this stage regardless of whether the input carry C_{in} is a 0 or a 1. This is called generated carry, expressed as $G_n= A_n.B_n$ which has to appear at the output through the OR gate as shown in fig.



A full adder (nth stage of a parallel adder).

Thereis another possibility of producing a carry out. X-OR gate inside the half-adder

at the input produces an intermediary sum bit- call it P_n –which is expressed as .
Next P_n and C_n are added using the X-OR gate inside the second half adder to produce the final

sum bit and $S_n = P_n \oplus C_n$ where $P_n = A_n \oplus B_n$ **and output carry** $C_0 = P_n . C_n = (A_n \oplus B_n) C_n$ **which becomes carry for the (n+1) th stage.**

Consider the case of both P_n and C_n being 1. The input carry C_n has to be propagated to the output only if P_n is 1. If P_n is 0, even if C_n is 1, the and gate in the second half-adder will inhibit C_n . the carry out of the nth stage is 1 when either G_n=1 or $P_n.C_n$ =1 or both G_n and $P_n.C_n$ are equal to 1.

For the final sum and carry outputs of the nth stage, we get the following Boolean expressions.

$$S_n = P_n \oplus C_n \text{ where } P_n = A_n \oplus B_n$$
$$C_{on} = C_{n+1} = G_n + P_n C_n \text{ where } G_n = A_n \cdot B_n$$

Observe the recursive nature of the expression for the output carry at the nth stage which becomes the input carry for the (n+1)st stage .it is possible to express the output carry of a higher significant stage is the carry-out of the previous stage.

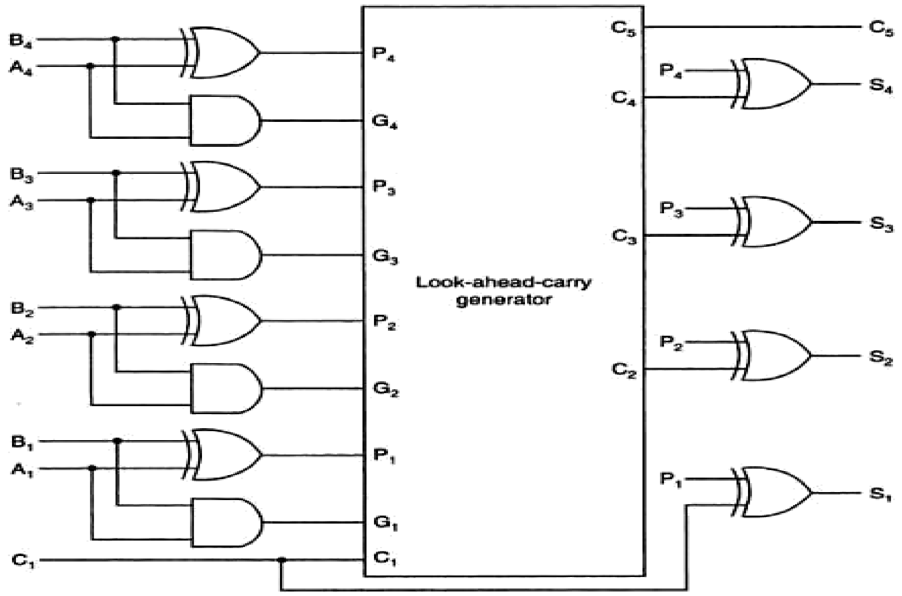
Based on these , the expression for the carry-outs of various full adders are as follows,

$$C_1 = G_0 + P_0 \cdot C_0$$
$$C_2 = G_1 + P_1 \cdot C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$$
$$C_3 = G_2 + P_2 \cdot C_2 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$$
$$C_4 = G_3 + P_3 \cdot C_3 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

The general expression for n stages designated as 0 through $(n-1)$ would be

$$C_n = G_{n-1} + P_{n-1} \cdot C_{n-1} = G_{n-1} + P_{n-1} \cdot G_{n-2} + P_{n-1} \cdot P_{n-2} \cdot G_{n-3} + \dots + P_{n-1} \cdot \dots P_0 \cdot C_0$$

Observe that the final output carry is expressed as a function of the input variables in SOP form. Which is two level AND-OR or equivalent NAND-NAND form. Observe that the full look-ahead scheme requires the use of OR gate with (n+1) inputs and AND gates with number of inputs varying from 2 to (n+1).



Logic diagram of a 4-bit look-ahead-carry adder.

2's complement Addition and Subtraction using Parallel Adders:

Most modern computers use the 2's complement system to represent negative numbers and to perform subtraction operations of signed numbers can be performed using only the addition operation ,if we use the 2's complement form to represent negative numbers.

The circuit shown can perform both addition and subtraction in the 2's complement. This adder/subtractor circuit is controlled by the control signal ADD/SUB'. When the ADD/SUB' level is HIGH, the circuit performs the addition of the numbers stored in registers A and B. When the ADD/Sub' level is LOW, the circuit subtract the number in register B from the number in register A. The operation is:

When ADD/SUB' is a 1:

1. **AND gates 1,3,5 and 7 are enabled , allowing B_0,B_1,$B_2$and B_3 to pass to the OR gates 9,10,11,12 . AND gates 2,4,6 and 8 are disabled , blocking B_0',B_1',B_2', and B_3' from reaching the OR gates 9,10,11 and 12.**
2. **The two levels B_0 to B_3 pass through the OR gates to the 4-bit parallel adder, to be added to the bits A_0 to A_3. The sum appears at the output S_0 to S_3**
3. **Add/SUB' =1 causes no carry into the adder.**

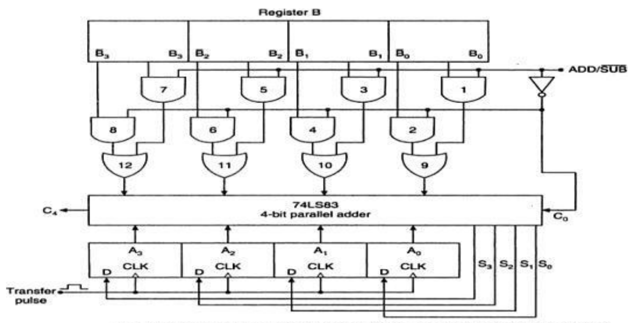
When ADD/SUB' is a 0:

1. **AND gates 1,3,5 and 7 are disabled , allowing B_0,B_1,$B_2$and B_3 from reaching the OR gates 9,10,11,12 . AND gates 2,4,6 and 8 are enabled , blocking B_0',B_1',B_2', and B_3' from reaching the OR gates.**

2. **The two levels B_0' to B_3' pass through the OR gates to the 4-bit parallel adder, to be added to the bits A_0 to A_3.The C_0 is now 1.thus the number in register B is converted to its 2's complement form.**

3. **The difference appears at the output S_0 to S_3.**

Adders/Subtractors used for adding and subtracting signed binary numbers. In computers , the output is transferred into the register A (accumulator) so that the result of the addition or subtraction always end up stored in the register A This is accomplished by applying a transfer pulse to the CLK inputs of register A.



Logic diagram of a parallel adder/subtractor using 2's complement system.

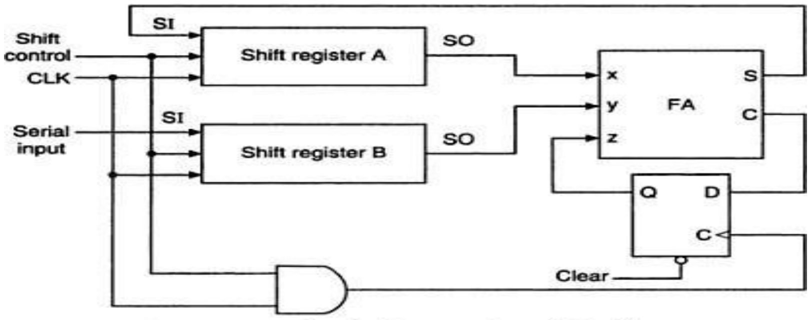
Serial Adder:

A serial adder is used to add binary numbers in serial form. The two binary numbers to be added serially are stored in two shift registers A and B. Bits are added one pair at a time through a single full adder (FA) circuit as shown. The carry out of the full-adder is transferred to a D flip-flop. The output of this flip-flop is then used as the carry input for the next pair of significant bits. The sum bit from the S output of the full-adder could be transferred to a third shift register. By shifting the sum into A while the bits of A are shifted out, it is possible to use one register for storing both augend and the sum bits. The serial input register B can be used to transfer a new binary number while the addend bits are shifted out during the addition.

The operation of the serial adder is:

Initially register A holds the augend, register B holds the addend and the carry flip-flop is cleared to 0. The outputs (SO) of A and B provide a pair of significant bits for the full-adder at x and y. The shift control enables both registers and carry flip-flop , so, at the clock pulse both registers are shifted once to the right, the sum bit from S enters the left most flip-flop of A , and the output carry is transferred into flip-flop Q . The shift control enables the registers for a number of clock pulses equal to the number of bits of the registers. For each succeeding clock pulse a new sum bit is transferred to A, a new carry is transferred to Q, and both registers are shifted once to the right. This process continues until the shift control is disabled. Thus the addition is accomplished by passing each pair of bits together with the previous carry through a single full adder circuit and transferring the sum, one bit at a time, into register A.

Initially, register A and the carry flip-flop are cleared to 0 and then the first number is added from B. While B is shifted through the full adder, a second number is transferred to it through its serial input. The second number is then added to the content of register A while a third number is transferred serially into register B. This can be repeated to form the addition of two, three, or more numbers and accumulate their sum in register A.



Logic diagram of a serial adder.

Difference between Serial and Parallel Adders:

The parallel adder registers with parallel load, whereas the serial adder uses shift registers. The number of full adder circuits in the parallel adder is equal to the number of bits in the binary numbers, whereas the serial adder requires only one full adder circuit and a carry flip-flop. Excluding the registers, the parallel adder is a combinational circuit, whereas the serial adder is a sequential circuit. The sequential circuit in the serial adder consists of a full-adder and a flip-flop that stores the output carry.

BCD Adder:

The BCD addition process:

1. **Add the 4-bit BCD code groups for each decimal digit position using ordinary binary addition.**
2. **For those positions where the sum is 9 or less, the sum is in proper BCD form and no correction is needed.**
3. **When the sum of two digits is greater than 9, a correction of 0110 should be added to that sum, to produce the proper BCD result. This will produce a carry to be added to the next decimal position.**

A BCD adder circuit must be able to operate in accordance with the above steps. In other words, the circuit must be able to do the following:

1. **Add two 4-bit BCD code groups, using straight binaryaddition.**

2. **Determine, if the sum of this addition is greater than 1101 (decimal 9); if it is , add 0110 (decimal 6) to this sum and generate a carry to the next decimalposition.**

The first requirement is easily met by using a 4- bit binary parallel adder such as the 74LS83 IC .For example , if the two BCD code groups $A_3A_2A_1A_0$and $B_3B_2B_1B_0$ are applied to a 4-bit parallel adder, the adder will output $S_4S_3S_2S_1S_0$, where S_4 is actually C_4 , the carry –out of the MSB bits.

The sum outputs $S_4S_3S_2S_1S_0$ can range anywhere from 00000 to 100109when both the BCD code groups are 1001=9). The circuitry for a BCD adder must include the logic needed to detect whenever the sum is greater than 01001, so that the correction can be added in. Those cases , where the sum is greater than 1001 are listed as:

S_4	S_3	S_2	S_1	S_0	Decimal number
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	0	0	0	0	16
1	0	0	0	1	17
1	0	0	1	0	18

Let us define a logic output X that will go HIGH only when the sum is greater than 01001 (i.e, for the cases in table). If examine these cases ,see that X will be HIGH for either of the following conditions:

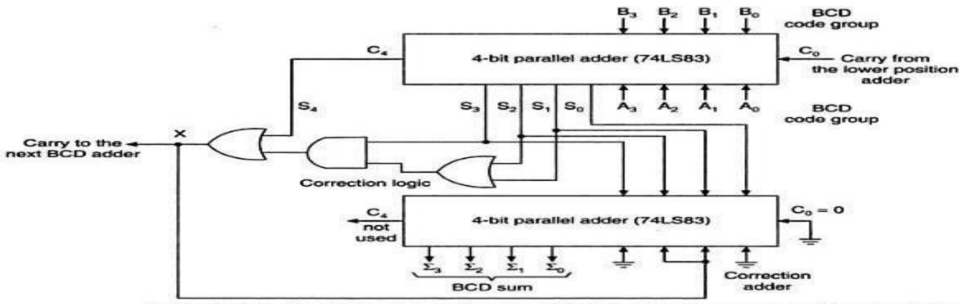
1. **Whenever S_4 =1(sum greater than 15)**
2. **Whenever S_3 =1 and either S_2 or S_1 or both are 1 (sum**

10 to 15) This condition can be expressed as

$X=S_4+S_3(S_2+S_1)$

Whenever X=1, it is necessary to add the correction factor 0110 to the sum bits, and to generate a carry. The circuit consists of three basic parts. The two BCD code groups $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are added together in the upper 4-bit adder, to produce the sum $S_4S_3S_2S_1S_0$. The logic gates shown implement the expression for X. The lower 4-bit adder will add the correction 0110 to the sum bits, only when X=1, producing the final BCD sum output represented by $\sum_3\sum_2\sum_1\sum_0$. The X is also the carry-out that is produced when the sum is greater than 01001. When X=0, there is no carry and no addition of 0110. In such cases, $\sum_3\sum_2\sum_1\sum_0= S_3S_2S_1S_0$.

Two or more BCD adders can be connected in cascade when two or more digit decimal numbers are to be added. The carry-out of the first BCD adder is connected as the carry-in of the second BCD adder, the carry-out of the second BCD adder is connected as the carry-in of the third BCD adder and so on.



Logic diagram of a BCD adder using two 4-bit adders and a correction-detector circuit.

EXCESS-3(XS-3) ADDER:

To perform Excess-3 additions,

1. **Add two xs-3 code groups**
2. **If carry=1, add 0011(3) to the sum of those two code groups**

If carry =0, subtract 0011(3) i.e., add 1101 (13 in decimal) to the sum of those two code groups.

Ex: Add 9 and 5

	1100	9 in Xs-3
	+1000	5 in xs-3
1	0100	there is a carry
+0011	0011	add 3 to each group
0100	0111	14 in xs-3
(1)	(4)	

EX:

(b)	0 1 1 1	4 in XS-3
	+ 0 1 1 0	3 in XS-3
	1 1 0 1	no carry
	+ 1 1 0 1	Subtract 3 (i.e. add 13)
Ignore carry	1 1 0 1 0	7 in XS-3
	(7)	

Implementation of xs-3 adder using 4-bit binary adders is shown. The augend ($A_3 A_2 A_1 A_0$) and addend ($B_3 B_2 B_1 B_0$) in xs-3 are added using the 4-bit parallel adder. If the carry is a 1, then 0011(3) is added to the sum bits $S_3 S_2 S_1 S_0$ of the upper adder in the lower 4-bit parallel

adder. If the carry is a 0, then 1101(3) is added to the sum bits (This is equivalent to subtracting 0011(3) from the sum bits. The correct sum in xs-3 is obtained

Excess-3 (XS-3) Subtractor:

To perform Excess-3 subtraction,

1. **Complement the subtrahend**
2. **Add the complemented subtrahend to the minuend.**
3. **If carry =1, result is positive. Add 3 and end around carry to the result . If carry=0, the result is negative. Subtract 3, i.e, and take the 1's complement of the result.**

Ex: Perform 9-4

	1100	9 in xs-3
	+1000	Complement of 4 n Xs-3
	- - - - - - - - - -	
(1)	0100	There is a carry
	+0011	Add 0011(3)
	- - - - - - - - - -	
0111		
	1	End around carry
	- - - - - - - - - -	
1000		5 in xs-3

The minuend and the 1's complement of the subtrahend in xs-3 are added in the upper 4-bit parallel adder. If the carry-out from the upper adder is a 0, then 1101 is added to the sum bits of the upper adder in the lower adder and the sum bits of the lower adder are complemented to get the result. If the carry-out from the upper adder is a 1, then 3=0011 is added to the sum bits of the lower adder and the sum bits of the lower adder give the result.

Binary Multipliers:

In binary multiplication by the paper and pencil method, is modified somewhat in digital machines because a binary adder can add only two binary numbers at a time. In a binary multiplier, instead of adding all the partial products at the end, they are added two at a time and their sum accumulated in a register (the accumulator register). In addition, when the multiplier bit is a 0,0s are not written down and added because it does not affect the final result. Instead, the multiplicand is shifted left by one bit.

The multiplication of 1110 by 1001 using this process is

Multiplicand 1110		
Multiplier	1001	
	1110	The LSB of the multiplier is a 1; write down the multiplicand; shift the multiplicand one position to the left (11100)
	1110	The second multiplier bit is a 0; write down the previous result 1110; shift the multiplicand to the left again (1 1 1 0 0 0)

0000 **The fourth multiplier bit is a 1 write down the new multiplicand add it to the first partial product to obtain the final product.**
1111110

This multiplication process can be performed by the serial multiplier circuit , which multiplies two 4-bit numbers to produce an 8-bit product. The circuit consists of following elements

X register: A 4-bit shift register that stores the multiplier --- it will shift right on the falling edge of the clock. Note that 0s are shifted in from the left.

B register: An 8-bit register that stores the multiplicand; it will shift left on the falling edge of the clock. Note that 0s are shifted in from the right.

A register: An 8-bit register, i.e, the accumulator that accumulates the partial products.

Adder:An 8-bit parallel adder that produces the sum of A and B registers. The adder outputs S_7 through S_0 are connected to the D inputs of the accumulator so that the sum can be transferred to the accumulator only when a clock pulse gets through the AND gate.

The circuit operation can be described by going through each step in the multiplication of 1110 by 1001. The complete process requires 4 clock cycles.

1. **Before the first clock pulse: Prior to the occurrence of the first clock pulse, the register A is loaded with 00000000, the register B with the multiplicand 00001110, and the register X with the multiplier 1001. Assume that each of these registers is loaded using its asynchronous inputs(i.e., PRESET and CLEAR). The output of the adder will be the sum of A and B,i.e., 00001110.**
2. **First Clock pulse:Since the LSB of the multiplier (X_0) is a 1, the first clock pulse gets through the AND gate and its positive going transition transfers the sum outputs into the accumulator. The subsequent negative going transition causes the X and B registers to shift right and left, respectively. This produces a new sum of A and B.**
3. **Second Clock Pulse: The second bit of the original multiplier is now in X_0 . Since this bit is a 0, the second clock pulse is inhibited from reaching the accumulator. Thus, the sum outputs are not transferred into the accumulator and the number in the accumulator does not change. The negative going transition of the clock pulse will again shift the X and B registers. Again a new sum is produced.**
4. **Third Clock Pulse:The third bit of the original multiplier is now in X_0;since this bit is a 0, the third clock pulse is inhibited from reaching the accumulator. Thus, the sum outputs are not transferred into the accumulator and the number in the accumulator does not change. The negative going transition of the clock pulse will again shift the X and B registers. Again a new sum is produced.**
5. **Fourth Clock Pulse: The last bit of the original multiplier is now in X_0 , and since it is a 1, the positive going transition of the fourth pulse transfers the sum into the accumulator. The accumulator now holds the final product. The negative going transition of the clock pulse shifts X and B again. Note that, X is now 0000, since all the multiplier bits have been shifted out.**

Code converters:

The availability of a large variety of codes for the same discrete elements of information results in the use of different codes by different digital systems. It is sometimes necessary to use the output of one system as the input to another. A conversion circuit must be inserted between the two systems if each uses different codes for the same information. Thus a

code converter is a logic circuit whose inputs are bit patterns representing numbers (or character) in one cod and whose outputs are the corresponding representation in a different code. Code converters are usually multiple output circuits.

To convert from binary code A to binary code B, the input lines must supply the bit combination of elements as specified by code A and the output lines must generate the corresponding bit combination of code B. A combinational circuit performs this transformation by means of logic gates.

For example, a binary –to-gray code converter has four binary input lines B_4, B_3,B_2,B_1 and four gray code output lines G_4,G_3,G_2,G_1. When the input is 0010, for instance, the output should be 0011 and so forth. To design a code converter, we use a code table treating it as a truth table to express each output as a Boolean algebraic function of all the inputs.

In this example, of binary –to-gray code conversion, we can treat the binary to the gray code table as four truth tables to derive expressions for G_4, G3, G2, and G1. Each of these four expressions would, in general, contain all the four input variables B_4, B3,B_2,and B1. Thus,this code converter is actually equivalent to four logic circuits, one for each of the truth tables.

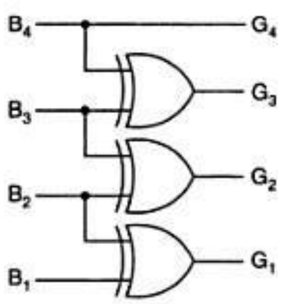
The logic expression derived for the code converter can be simplified using the usual techniques, including _don't cares' if present. Even if the input is an unweighted code, the same cell numbering method which we used earlier can be used, but the cell numbers -- must correspond to the input combinations as if they were an 8-4-2-1 weighted code. s

Design of a 4-bit binary to gray code converter:

$$G_4 = \Sigma m(8, 9, 10, 11, 12, 13, 14, 15) \qquad G_4 = B_4$$
$$G_3 = \Sigma m(4, 5, 6, 7, 8, 9, 10, 11) \qquad G_3 = \overline{B}_4 B_3 + B_4 \overline{B}_3 = B_4 \oplus B_3$$
$$G_2 = \Sigma m(2, 3, 4, 5, 10, 11, 12, 13) \qquad G_2 = \overline{B}_3 B_2 + B_3 \overline{B}_2 = B_3 \oplus B_2$$
$$G_1 = \Sigma m(1, 2, 5, 6, 9, 10, 13, 14) \qquad G_1 = \overline{B}_2 B_1 + B_2 \overline{B}_1 = B_2 \oplus B_1$$

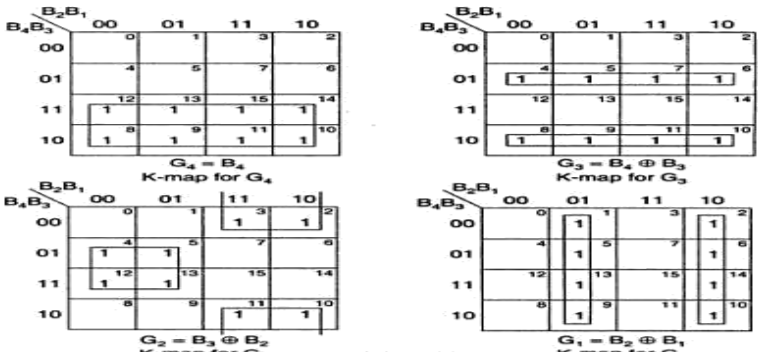
4-bit binary				4-bit Gray			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

(a) Conversion table



(c) Logic diagram

4-bit binary-to-Gray code converter



(b) K-maps

4-bit binary-to-Gray code converter.

Design of a 4-bit gray to Binary code converter:

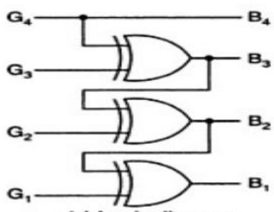
$$B_4 = \Sigma\, m(12, 13, 15, 14, 10, 11, 9, 8) = \Sigma\, m(8, 9, 10, 11, 12, 13, 14, 15)$$
$$B_3 = \Sigma\, m(6, 7, 5, 4, 10, 11, 9, 8) = \Sigma\, m(4, 5, 6, 7, 8, 9, 10, 11)$$
$$B_2 = \Sigma\, m(3, 2, 5, 4, 15, 14, 9, 8) = \Sigma\, m(2, 3, 4, 5, 8, 9, 14, 15)$$
$$B_1 = \Sigma\, m(1, 2, 7, 4, 13, 14, 11, 8) = \Sigma\, m(1, 2, 4, 7, 8, 11, 13, 14)$$

$$B_4 = G_4$$
$$B_3 = \bar{G}_4 G_3 + G_4 \bar{G}_3 = G_4 \oplus G_3$$
$$B_2 = \bar{G}_4 G_3 \bar{G}_2 + \bar{G}_4 \bar{G}_3 G_2 + G_4 \bar{G}_3 \bar{G}_2 + G_4 G_3 G_2$$
$$= \bar{G}_4 (G_3 \oplus G_2) + G_4 (\overline{G_3 \oplus G_2}) = G_4 \oplus G_3 \oplus G_2 = B_3 \oplus G_2$$
$$B_1 = \bar{G}_4 \bar{G}_3 \bar{G}_2 G_1 + \bar{G}_4 \bar{G}_3 G_2 \bar{G}_1 + \bar{G}_4 G_3 G_2 G_1 + \bar{G}_4 G_3 \bar{G}_2 \bar{G}_1 + G_4 G_3 \bar{G}_2 G_1 + G_4 G_3 G_2 \bar{G}_1 + G_4 \bar{G}_3 G_2 G_1 + G_4 \bar{G}_3 \bar{G}_2 \bar{G}_1$$

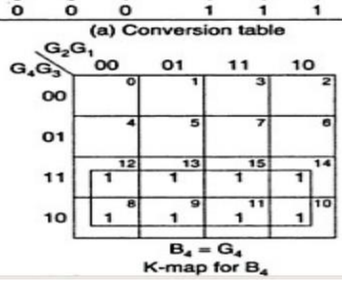
$$= \bar{G}_4 \bar{G}_3 (G_2 \oplus G_1) + G_4 G_3 (G_2 \oplus G_1) + \bar{G}_4 G_3 (\overline{G_2 \oplus G_1}) + G_4 \bar{G}_3 (\overline{G_2 \oplus G_1})$$
$$= (G_2 \oplus G_1)(\overline{G_4 \oplus G_3}) + (\overline{G_2 \oplus G_1})(G_4 \oplus G_3)$$
$$= G_4 \oplus G_3 \oplus G_2 \oplus G_1$$

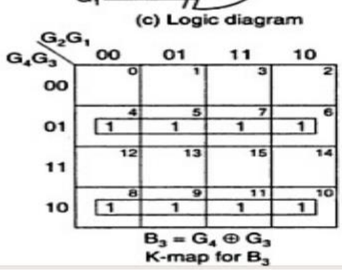
4-bit Gray				4-bit binary			
G_4	G_3	G_2	G_1	B_4	B_3	B_2	B_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

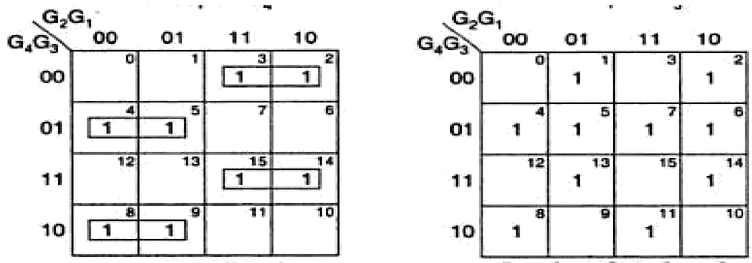
(a) Conversion table



(c) Logic diagram







$B_2 = G_4 \oplus G_3 \oplus G_2$ K-map for B_2

$B_1 = G_4 \oplus G_3 \oplus G_2 \oplus G_1$ K-map for B_1

(b) K-maps

4-bit Gray-to-binary code converter.

Design of a 4-bit BCD to XS-3 code converter:

8421 code				XS-3 code			
B_4	B_3	B_2	B_1	X_4	X_3	X_2	X_1
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

(a) Conversion table

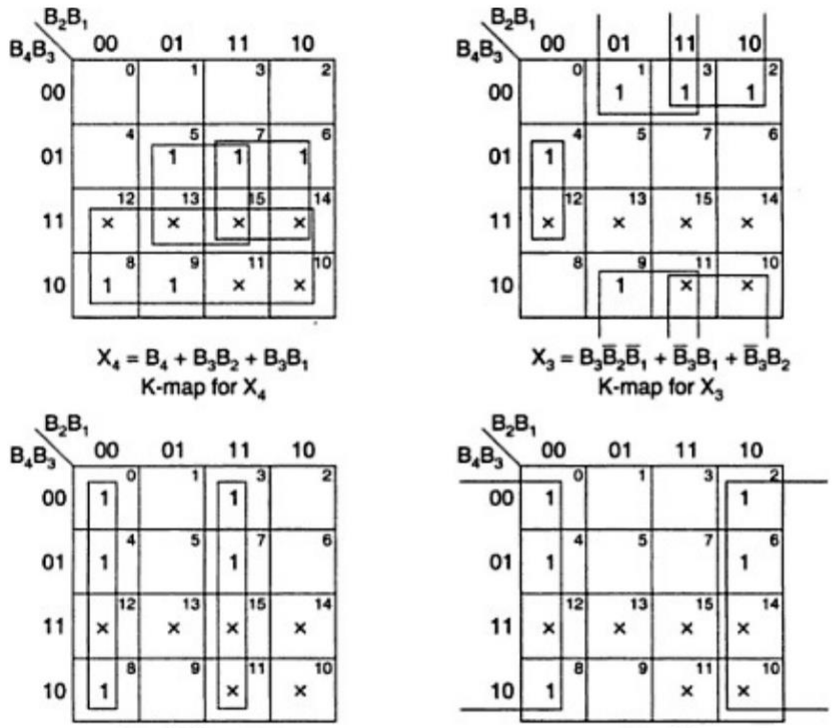
$X_4 = \Sigma\, m(5, 6, 7, 8, 9) + d(10, 11, 12, 13, 14, 15)$
$X_3 = \Sigma\, m(1, 2, 3, 4, 9) + d(10, 11, 12, 13, 14, 15)$
$X_2 = \Sigma\, m(0, 3, 4, 7, 8) + d(10, 11, 12, 13, 14, 15)$
$X_1 = \Sigma\, m(0, 2, 4, 6, 8) + d(10, 11, 12, 13, 14, 15)$

The minimal expressions are

$X_4 = B_4 + B_3B_2 + B_3B_1$
$X_3 = B_3\overline{B}_2\overline{B}_1 + \overline{B}_3B_1 + \overline{B}_3B_2$
$X_2 = \overline{B}_2\overline{B}_1 + B_2B_1$
$X_1 = \overline{B}_1$

(b) Minimal expressions

4-bit BCD-to-XS-3 code converter



$X_4 = B_4 + B_3B_2 + B_3B_1$ K-map for X_4

$X_3 = B_3\overline{B}_2\overline{B}_1 + \overline{B}_3B_1 + \overline{B}_3B_2$ K-map for X_3

$X_2 = \overline{B}_2\overline{B}_1 + B_2B_1$ K-map for X_2

$X_1 = \overline{B}_1$ K-map for X_1

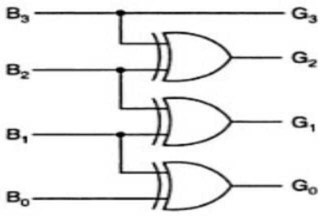
(c) K-maps

4-bit BCD-to-XS-3 code converter.

Design of a BCD to gray code converter:

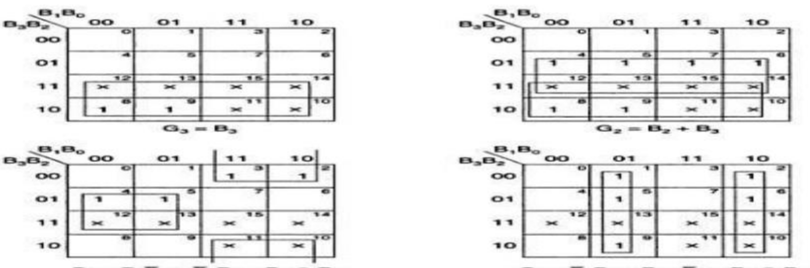
BCD code				Gray code			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1

(a) BCD-to-Gray code conversion table



(b) Logic diagram

BCD-to-Gray code converter.



$G_3 = B_3$ $G_2 = B_2 + B_3$

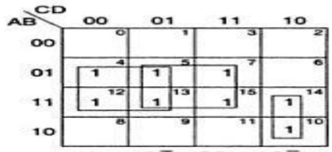
$G_1 = \bar{B}_2B_1 + B_2\bar{B}_1 = B_2 \oplus B_1$ $G_0 = \bar{B}_1B_0 + B_1 \cdot \bar{B}_0 = B_1 \oplus B_0$

K-maps for a BCD-to-Gray code converter.

Design of a SOP circuit to Detect the Decimal numbers 5 through 12 in a 4-bit gray code Input:

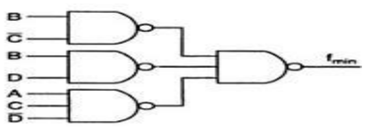
Decimal number	4-bit Gray code A	B	C	D	Output f
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	1	0
3	0	0	1	0	0
4	0	1	1	0	0
5	0	1	1	1	1
6	0	1	0	1	1
7	0	1	0	0	1
8	1	1	0	0	1
9	1	1	0	1	1
10	1	1	1	1	1
11	1	1	1	0	1
12	1	0	1	0	1
13	1	0	1	1	0
14	1	0	0	1	0
15	1	0	0	0	0

(a) Truth table



$f_{min} = B\bar{C} + BD + AC\bar{D}$

(b) K-map



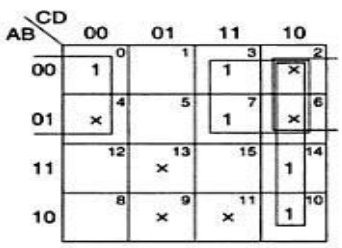
(c) NAND logic

Truth table, K-map and logic diagram for the SOP circuit.

Design of a SOP circuit to detect the decimal numbers 0,2,4,6,8 in a 4-bit 5211 BCD code input:

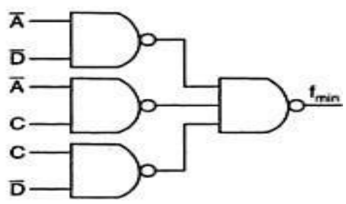
Decimal number	5211 code A	B	C	D	Output f
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	1	1
3	0	1	0	1	0
4	0	1	1	1	1
5	1	0	0	0	0
6	1	0	1	0	1
7	1	1	0	0	0
8	1	1	1	0	1
9	1	1	1	1	0

(a) Truth table



$f_{min} = \bar{A}\bar{D} + \bar{A}C + C\bar{D}$

(b) K-map



(c) Logic diagram

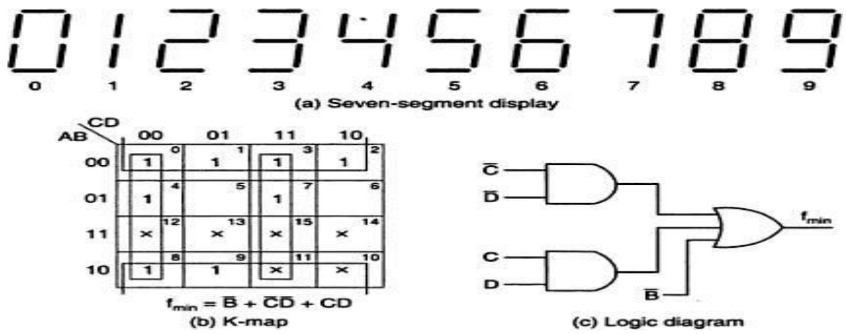
Truth table, K-map and logic diagram for the SOP circuit.

Design of a Combinational circuit to produce the 2's complement of a 4-bit binary number:

Input				Output			
A	B	C	D	E	F	G	H
0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	0
0	0	1	1	1	1	0	1
0	1	0	0	1	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	0	1	0
0	1	1	1	1	0	0	1
1	0	0	0	1	0	0	0
1	0	0	1	0	1	1	1
1	0	1	0	0	1	1	0
1	0	1	1	0	1	0	1
1	1	0	0	0	1	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	0	1	0
1	1	1	1	0	0	0	1

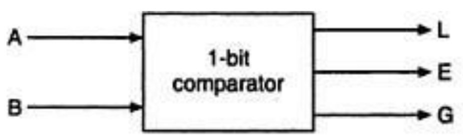
(a) Conversion table

Conversion table and K-maps for the circuit



Comparators:

$$\text{EQUALITY} = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$



Block diagram of a 1-bit comparator.

1. Magnitude Comparator:

The logic for a 1-bit magnitude comparator: Let the 1-bit numbers be $A = A_0$ and $B = B_0$.
If $A_0 = 1$ and $B_0 = 0$, then $A > B$.
Therefore,

$$A > B: G = A_0\bar{B}_0$$

If $A_0 = 0$ and $B_0 = 1$, then $A < B$.
Therefore,

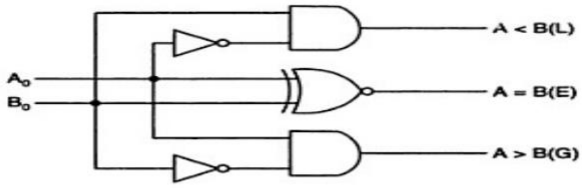
$$A < B: L = \bar{A}_0 B_0$$

If A_0 and B_0 coincide, i.e. $A_0 = B_0 = 0$ or if $A_0 = B_0 = 1$, then $A = B$.
Therefore,

$$A = B : E = A_0 \odot B_0$$

A_0	B_0	L	E	G
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

(a) Truth table



(b) Logic diagram

1-bit comparator.

1- bit Magnitude Comparator:

The logic for a 2-bit magnitude comparator: Let the two 2-bit numbers be $A = A_1 A_0$ and $B = B_1 B_0$.

1. If $A_1 = 1$ and $B_1 = 0$, then $A > B$ or
2. If A_1 and B_1 coincide and $A_0 = 1$ and $B_0 = 0$, then $A > B$. So the logic expression for $A > B$ is

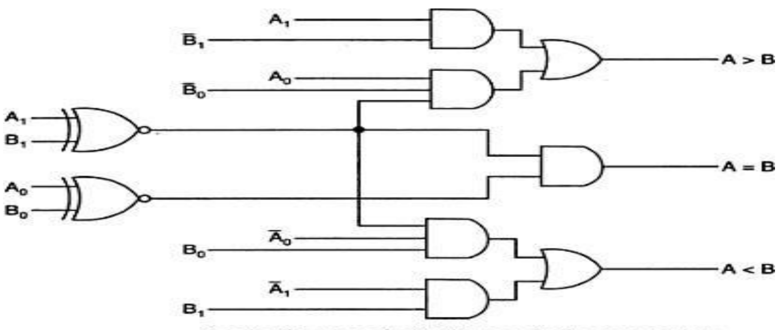
$$A > B : G = A_1\bar{B}_1 + (A_1 \odot B_1)A_0\bar{B}_0$$

1. If $A_1 = 0$ and $B_1 = 1$, then $A < B$ or
2. If A_1 and B_1 coincide and $A_0 = 0$ and $B_0 = 1$, then $A< B$. So the expression for $A < B$ is

$$A < B : L = \bar{A}_1 B_1 + (A_1 \odot B_1)\bar{A}_0 B_0$$

If A_1 and B_1 coincide and if A_0 and B_0 coincide then $A = B$. So the expression for $A = B$ is

$$A = B : E = (A_1 \odot B_1)(A_0 \odot B_0)$$



Logic diagram of a 2-bit magnitude comparator.

4- Bit Magnitude Comparator:

The logic for a 4-bit magnitude comparator: Let the two 4-bit numbers be $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$.

1. If $A_3 = 1$ and $B_3 = 0$, then $A > B$. Or
2. If A_3 and B_3 coincide, and if $A_2 = 1$ and $B_2 = 0$, then $A > B$. Or
3. If A_3 and B_3 coincide, and if A_2 and B_2 coincide, and if $A_1 = 1$ and $B_1 = 0$, then $A > B$. Or
4. If A_3 and B_3 coincide, and if A_2 and B_2 coincide, and if A_1 and B_1 coincide, and if $A_0 = 1$ and $B_0 = 0$, then $A > B$.

From these statements, we see that the logic expression for $A > B$ can be written as

$$(A > B) = A_3\overline{B}_3 + (A_3 \odot B_3)A_2\overline{B}_2 + (A_3 \odot B_3)(A_2 \odot B_2)A_1\overline{B}_1 + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)A_0\overline{B}_0$$

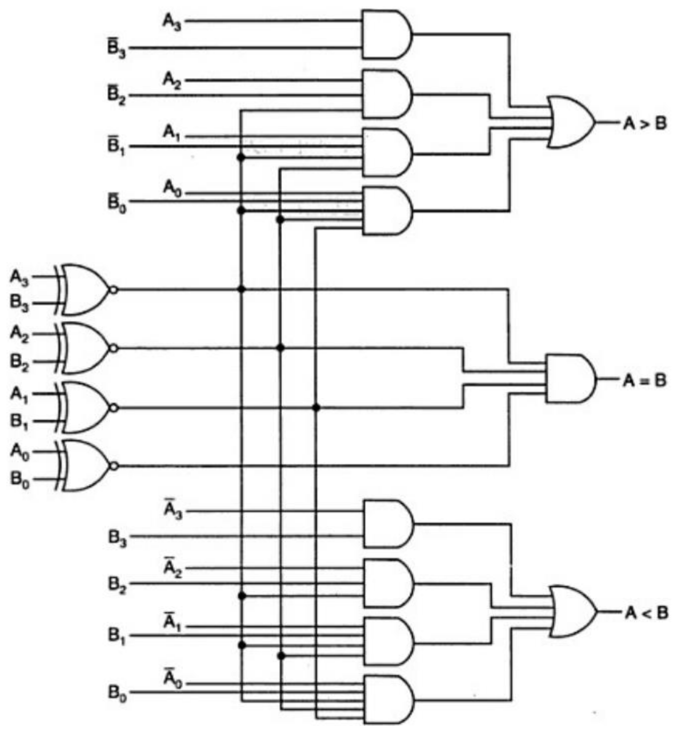
Similarly, the logic expression for $A < B$ can be written as

$$A < B = \overline{A}_3B_3 + (A_3 \odot B_3)\overline{A}_2B_2 + (A_3 \odot B_3)(A_2 \odot B_2)\overline{A}_1B_1 + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)\overline{A}_0B_0$$

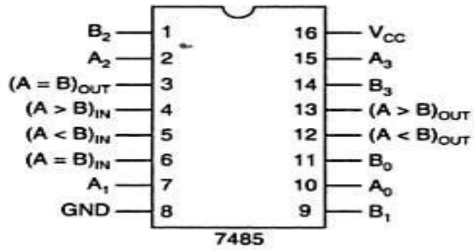
If A_3 and B_3 coincide and if A_2 and B_2 coincide and if A_1 and B_1 coincide and if A_0 and B_0 coincide, then $A = B$.

So the expression for $A = B$ can be written as

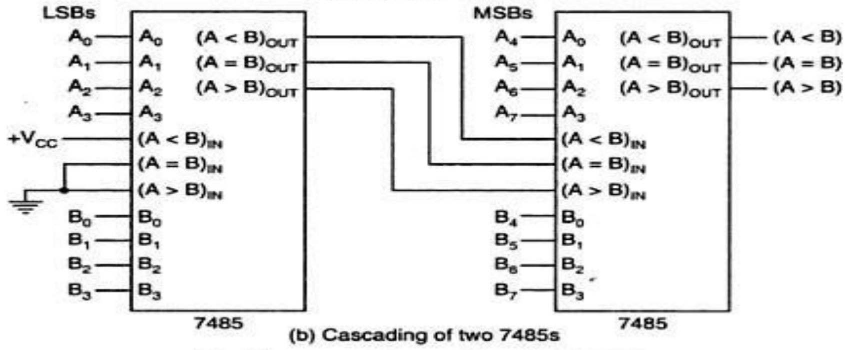
$$(A = B) = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$



IC Comparator:



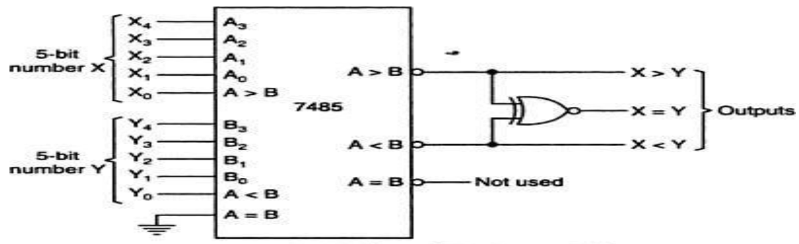
(a) Pin diagram of 7485



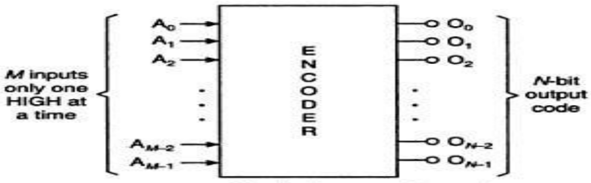
(b) Cascading of two 7485s

Pin diagram and cascading of 7485 4-bit comparators.

ENCODERS:



Use of 7485 as a 5-bit comparator.

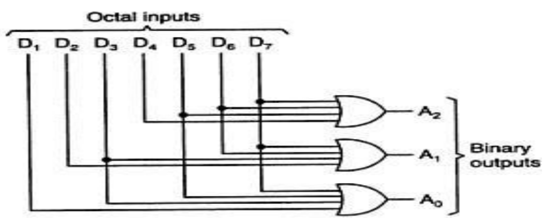


Block diagram of encoder.

Octal to Binary Encoder:

Octal digits		Binary A_2	A_1	A_0
D_0	0	0	0	0
D_1	1	0	0	1
D_2	2	0	1	0
D_3	3	0	1	1
D_4	4	1	0	0
D_5	5	1	0	1
D_6	6	1	1	0
D_7	7	1	1	1

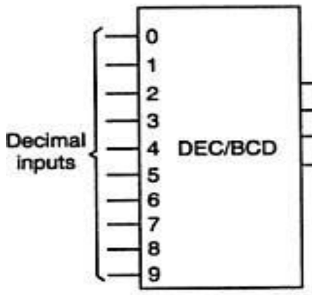
(a) Truth table



(b) Logic diagram

Octal-to-binary encoder.

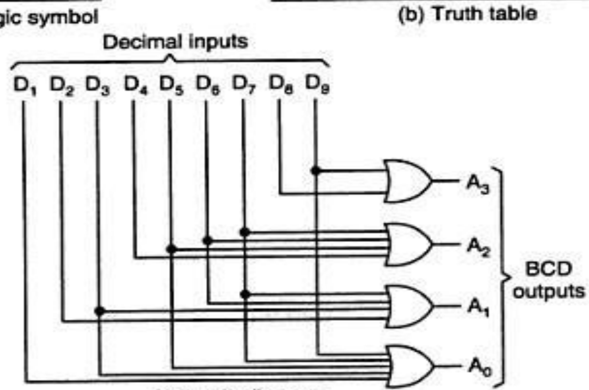
Decimal to BCD Encoder:



(a) Logic symbol

Decimal inputs		A_3	A_2	A_1	A_0
		Binary			
D_0	0	0	0	0	0
D_1	1	0	0	0	1
D_2	2	0	0	1	0
D_3	3	0	0	1	1
D_4	4	0	1	0	0
D_5	5	0	1	0	1
D_6	6	0	1	1	0
D_7	7	0	1	1	1
D_8	8	1	0	0	0
D_9	9	1	0	0	1

(b) Truth table



(c) Logic diagram

Decimal-to-BCD encoder.

Tristate bus system:

In three-state, tri-state, or 3-statelogic allows an output port to assume a high impedance state in addition to the 0 and 1 logic levels, effectively removing the output from the circuit.

This allows multiple circuits to share the same output line or lines (such as a bus which cannot listen to more than one device at a time).

Three-state outputs are implemented in many registers, bus drivers, and flip-flops in the 7400 and 4000 series as well as in other types, but also internally in many integrated circuits. Other typical uses are internal and external buses in microprocessors, computer memory, and peripherals. Many devices are controlled by an active-low input called OE (Output Enable) which dictates whether the outputs should be held in a high-impedance state or drive their respective loads (to either 0- or 1-level).

A tristate buffer can be thought of as a switch. If *B* is on, the switch is closed. If B is off, the switch is open.

INPUT		OUTPUT
A	B	C
0	1	0
1		1
X	0	Z (high impedance)

MODULE-IV:

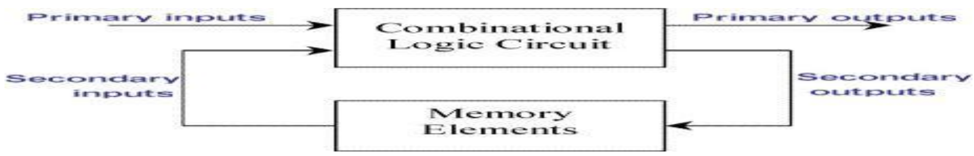
Sequential Logic Circuits - I

Sequential circuits

Classification of sequential circuits: Sequential circuits may be classified as two types.

1. **Synchronous sequential circuits**
2. **Asynchronous sequential circuits**

Combinational logic refers to circuits whose output is strictly depended on the present value of the inputs. As soon as inputs are changed, the information about the previous inputs is lost, that is, combinational logics circuits have no memory. Although every digital system is likely to have combinational circuits, most systems encountered in practice also include memory elements, which require that the system be described in terms of sequential logic. Circuits whose output depends not only on the present input value but also the past input value are known as sequential logic circuits. The mathematical model of a sequential circuit is usually referred to as a sequential machine.



Comparison between combinational and sequential circuits

Combinational circuit	Sequential circuit
1. In combinational circuits, the output variables at any instant of time are dependent only on the present input variables	1. in sequential circuits the output variables at any instant of time are dependent not only on the present input variables, but also on the present state
2.memory unit is not requires in combinational circuit	2.memory unit is required to store the past history of the input variables
3. these circuits are faster because the delay between the i/p and o/p due to propagation delay of gates only	3. sequential circuits are slower than combinational circuits
4. easy to design	4. comparatively hard to design

Level mode and pulse mode asynchronous sequential circuits:

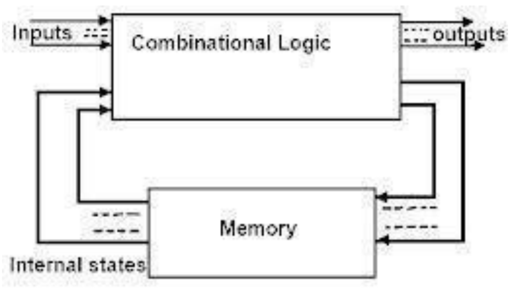


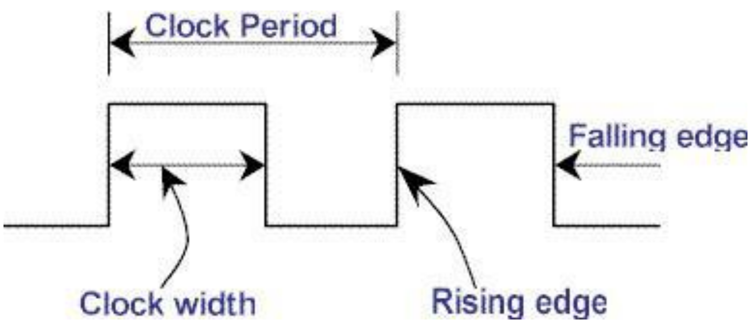
Figure 1: Asynchronous Sequential Circuit

Fig shows a block diagram of an asynchronous sequential circuit. It consists of a combinational circuit and delay elements connected to form the feedbackloops. The present state and next state variables in asynchronous sequential circuits called secondary variables and excitation variables respectively..

There are two types of asynchronous circuits: fundamental mode circuits and pulse mode circuits.

Synchronous and Asynchronous Operation:

Sequential circuits are divided into two main types: synchronous and asynchronous. Their classification depends on the timing of their signals.Synchronous sequential circuits change their states and output values at discrete instants of time, which are specified by the rising and falling edge of a free-running clock signal. The clock signal is generally some form of square wave as shown in Figure below.



From the diagram you can see that the clock period is the time between successive transitions in the same direction, that is, between two rising or two falling edges. State transitions in synchronous sequential circuits are made to take place at times when the clock is making a transition from 0 to 1 (rising edge) or from 1 to 0 (falling edge). Between successive clock pulses there is no change in the information stored in memory.

The reciprocal of the clock period is referred to as the clock frequency. The clock width is defined as the time during which the value of the clock signal is equal to 1. The ratio of the clock width and clock period is referred to as the duty cycle. A clock signal is said to

be active high if the state changes occur at the clock's rising edge or during the clock width. Otherwise, the clock is said to be active low. Synchronous sequential circuits are also known as clocked sequential circuits.

The memory elements used in synchronous sequential circuits are usually flip-flops. These circuits are binary cells capable of storing one bit of information. A flip-flop circuit has two outputs, one for the normal value and one for the complement value of the bit stored in it. Binary information can enter a flip-flop in a variety of ways, a fact which give rise to the different types of flip-flops. For information on the different types of basic flip-flop circuits and their logical properties, see the previous tutorial on flip-flops.

In asynchronous sequential circuits, the transition from one state to another is initiated by the change in the primary inputs; there is no external synchronization. The memory commonly used in asynchronous sequential circuits are time-delayed devices, usually implemented by feedback among logic gates. Thus, asynchronous sequential circuits may be regarded as combinational circuits with feedback. Because of the feedback among logic gates, asynchronous sequential circuits may, at times, become unstable due to transient conditions. The instability problem imposes many difficulties on the designer. Hence, they are not as commonly used as synchronous systems.

Fundamental Mode Circuits assumes that:

1. The input variables change only when the circuit is stable
2. Only one input variable can change at a given time
3. Inputs are levels are not pulses

A pulse mode circuit assumes that:

1. The input variables are pulses instead of levels
2. The width of the pulses is long enough for the circuit to respond to the input
3. The pulse width must not be so long that is still present after the new state is reached.

Latches and flip-flops

Latches and flip-flops are the basic elements for storing information. One latch or flip-flop can store one bit of information. The main difference between latches and flip-flops is that for latches, their outputs are constantly affected by their inputs as long as the enable signal is asserted. In other words, when they are enabled, their content changes immediately when their inputs change. Flip-flops, on the other hand, have their content change only either at the rising or falling edge of the enable signal. This enable signal is usually the controlling clock signal. After the rising or falling edge of the clock, the flip-flop content remains constant even if the input changes.

There are basically four main types of latches and flip-flops: SR, D, JK, and T. The major differences in these flip-flop types are the number of inputs they have and how they change state. For each type, there are also different variations that enhance their operations. In this chapter, we

will look at the operations of the various latches and flip-flops.the flip-flops has two outputs, labeled Q and Q‘. the Q output is the normal output of the flip flop and Q‘ is the inverted output.

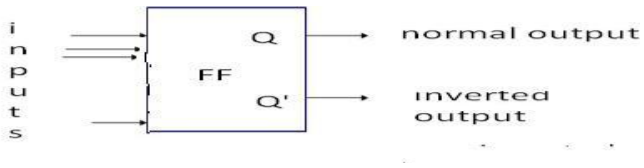


Figure: basic symbol of flipflop

A latch may be an active-high input latch or an active –LOW input latch.active –HIGH means that the SET and RESET inputs are normally resting in the low state and one of them will be pulsed high whenever we want to change latch outputs.

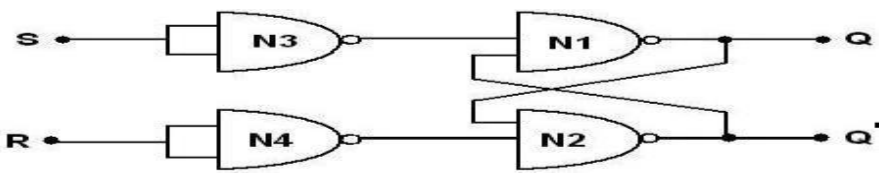
SR latch:

The latch has two outputs Q and Q‘. When the circuit is switched on the latch may enter into any state. If Q=1, then Q‘=0, which is called SET state. If Q=0, then Q‘=1, which is called RESET state. Whether the latch is in SET state or RESET state, it will continue to remain in the same state, as long as the power is not switched off. But the latch is not an useful circuit, since there is no way of entering the desired input. It is the fundamental building block in constructing flip-flops, as explained in the following sections

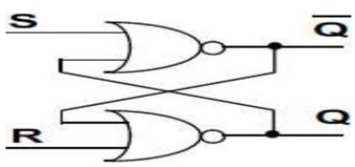
NAND latch

NAND latch is the fundamental building block in constructing a flip-flop. It has the property of holding on to any previous output, as long as it is not disturbed.

The opration of NAND latch is the reverse of the operation of NOR latch.if 0‘s are replaced by 1‘s and 1‘s are replaced by 0‘s we get the same truth table as that of the NOR latch shown



NOR latch



S	R	Q	$\overline{Q}$	Function
0	0	Q^+	$\overline{Q}^+$	Storage State
0	1	0	1	Reset
1	0	1	0	Set
1	1	0-?	0-?	Indeterminate State

The analysis of the operation of the active-HIGHNOR latch can be summarized as follows.

1. **SET=0, RESET=0: this is normal resting state of the NOR latch and it has no effect on the output state. Q and Q‘ will remain in whatever stste they were prior to the occurrence of this input condition.**
2. **SET=1, RESET=0: this will always set Q=1, where it will remain even after SET returns to 0**
3. **SET=0, RESET=1: this will always reset Q=0, where it will remain even after RESET returns to 0**
4. **SET=1,RESET=1; this condition tries to SET and RESET the latch at the same time, and it produces Q=Q‘=0. If the inputs are returned to zero simultaneously, the resulting output stste is erratic and unpredictable. This input condition should not be used.**

The SET and RESET inputs are normally in the LOW state and one of them will be pulsed HIGH. Whenever we want to change the latch outputs..

RS Flip-flop:

The basic flip-flop is a one bit memory cell that gives the fundamental idea of memory device. It constructed using two NAND gates. The two NAND gates N1 andN2 are connected such that, output of N1 is connected to input of N2 and output of N2 to input of N1. These form the feedback path the inputs are S and R, and outputs are Q and Q‘. The logic diagram and the block diagram of R-S flip-flop with clocked input

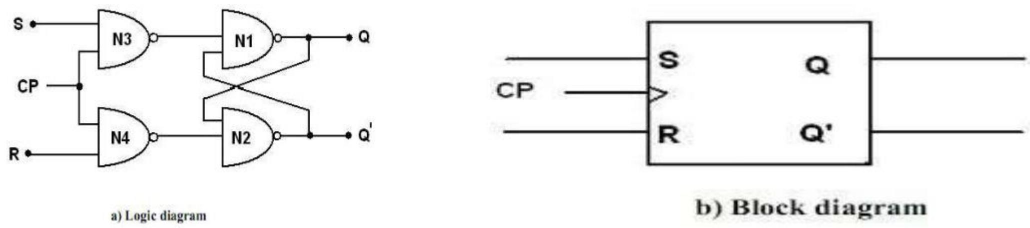


Figure: RS Flip-flop

The flip-flop can be made to respond only during the occurrence of clock pulse by adding two NAND gates to the input latch. So synchronization is achieved. i.e., flip-flops are allowed to change their states only at particular instant of time. The clock pulses are generated by a clock pulse generator. The flip-flops are affected only with the arrival of clock pulse.

Operation:

1. When CP=0 the output of N3 and N4 are 1 regardless of the value of S and R. This is given as input to N1 and N2. This makes the previous value of Q and Q‘unchanged.

2. When CP=1 the information at S and R inputs are allowed to reach the latch and change of state in flip-flop takes place.

3. CP=1, S=1, R=0 gives the SET state i.e., Q=1, Q‘=0.

4. CP=1, S=0, R=1 gives the RESET state i.e., Q=0, Q'=1.

5. CP=1, S=0, R=0 does not affect the state of flip-flop.

6. CP=1, S=1, R=1 is not allowed, because it is not able to determine the next state. This condition is said to be a ―race condition‖.

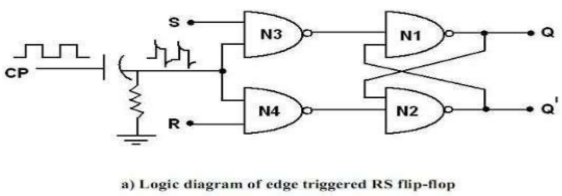
In the logic symbol CP input is marked with a triangle. It indicates the circuit responds to an input change from 0 to 1. The characteristic table gives the operation conditions of flip-flop. Q(t) is the present state maintained in the flip-flop at time ‗t'. Q(t+1) is the state after the occurrence of clock pulse.

Truth table

S	R	$Q_{(t+1)}$	Comments
0	0	Q_t	No change
0	1	0	Reset / clear
1	0	1	Set
1	1	*	Not allowed

Edge triggered RS flip-flop:

Some flip-flops have an RC circuit at the input next to the clock pulse. By the design of the circuit the R-C time constant is much smaller than the width of the clock pulse. So the output changes will occur only at specific level of clock pulse. The capacitor gets fully charged when clock pulse goes from low to high. This change produces a narrow positive spike. Later at the trailing edge it produces narrow negative spike. This operation is called edge triggering, as the flip-flop responds only at the changing state of clock pulse. If output transition occurs at rising edge of clock pulse (0□1),**it is called positively edge triggering. If it occurs at trailing edge (** 1□ **0) it is called negative edge triggering. Figure shows the logic and block diagram.**



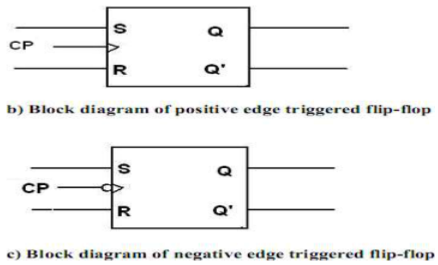
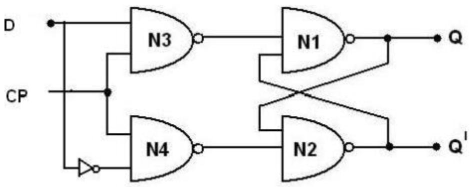


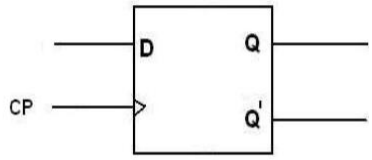
Figure: Edge triggered RS flip-flop

D flip-flop:

The D flip-flop is the modified form of R-S flip-flop. R-S flip-flop is converted to D flip-flop by adding an inverter between S and R and only one input D is taken instead of S and R. So one input is D and complement of D is given as another input. The logic diagram and the block diagram of D flip-flop with clocked input



a) Logic diagram



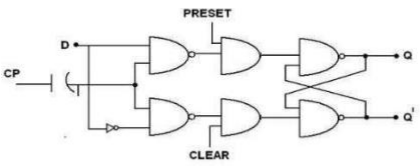
b) Block diagram

When the clock is low both the NAND gates (N1 and N2) are disabled and Q retains its last value. When clock is high both the gates are enabled and the input value at D is transferred to its output Q. D flip-flop is also called —Data flip-flop‖.

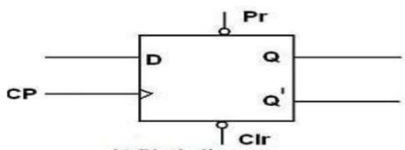
Truth table

CP	D	Q
0	x	Previous state
1	0	0
1	1	1

Edge Triggered D Flip-flop:



a) Logic diagram



b) Block diagram

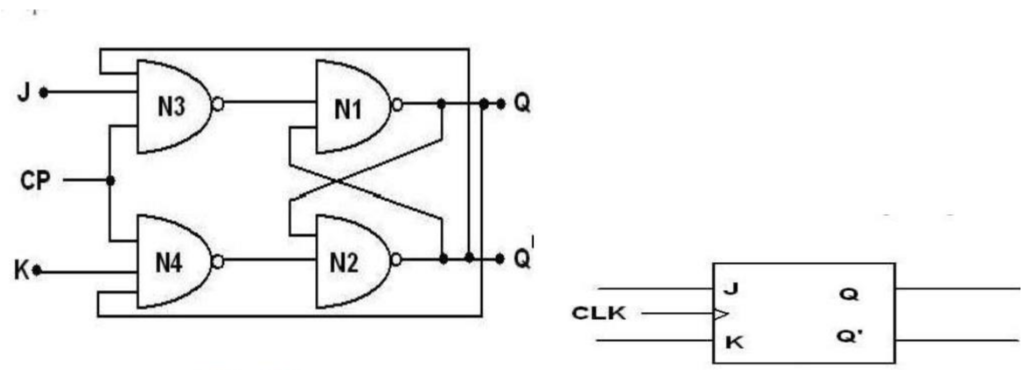
Truth table

PRESET	CLEAR	CP	D	Q
0	0	X	X	*(forbidden)
0	1	X	X	1
1	0	X	X	0
1	0	0	X	NC
1	1	1	X	NC
1	1	↓	X	NC
1	1	↑	0	0
1	1	↑	1	1

Figure: truth table, block diagram, logic diagram of edge triggered

flip-flop JK flip-flop (edge triggered JK flip-flop)

The race condition in RS flip-flop, when R=S=1 is eliminated in J-K flip-flop. There is a feedback from the output to the inputs. Figure 3.4 represents one way of building a JK flip-flop.



a) Logic diagram b) Block diagram

Truth table

J	K	$Q_{(t+1)}$	Comments
0	0	Q_t	No change
0	1	0	Reset / clear
1	0	1	Set
1	1	Q'_t	Complement/ toggle.

Figure: JK flip-flop

The J and K are called control inputs, because they determine what the flip-flop does when a positive clock edge arrives.

Operation:

1. When J=0, K=0 then both N3 and N4 will produce high output and the previous value of Q and Q‘ retained as it is.

2. When J=0, K=1, N3 will get an output as 1 and output of N4 depends on the value of Q. The final output is Q=0, Q‘=1 i.e., reset state

3. When J=1, K=0 the output of N4 is 1 and N3 depends on the value of Q‘. The final output is Q=1 and Q‘=0 i.e., set state

4. When J=1, K=1 it is possible to set (or) reset the flip-flop depending on the current state of output. If Q=1, Q‘=0 then N4 passes ‘0‘to N2 which produces Q‘=1, Q=0 which is reset state. When J=1, K=1, Q changes to the complement of the last state. The flip-flop is said to be in the toggle state.

The characteristic equation of the JK flip-flop is:

$$Q_{next} = J\overline{Q} + \overline{K}Q$$

JK flip-flop operation[28]

Characteristic table				Excitation table				
J	**K**	**Q_{next}**	**Comment**	**Q**	**Q_{next}**	**J**	**K**	**Comment**
0	**0**	**Q**	**hold state**	**0**	**0**	**0**	**X**	**No change**
0	**1**	**0**	**reset**	**0**	**1**	**1**	**X**	**Set**
1	**0**	**1**	**set**	**1**	**0**	**X**	**1**	**Reset**
1	**1**	**Q**	**toggle**	**1**	**1**	**X**	**0**	**No change**

T flip-flop:

If the T input is high, the T flip-flop changes state ("toggles") whenever the clock input is strobed. If the T input is low, the flip-flop holds the previous value. This behavior is described by the characteristic equation

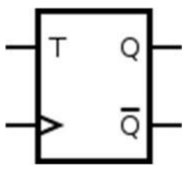


Figure : symbol for T flip flop

$Q_{next} = T \oplus Q = T\overline{Q} + \overline{T}Q$ **(expanding the XOR operator**

When T is held high, the toggle flip-flop divides the clock frequency by two; that is, if clock frequency is 4 MHz, the output frequency obtained from the flip-flop will be 2 MHz This "divide by" feature has application in various types of digital counters. A T flip-flop can also be built using a JK flip-flop (J & K pins are connected together and act as T) or D flip-flop (T input and $P_{revious}$ is connected to the D input through an XOR gate).

T flip-flop operation[28]

Characteristic table				Excitation table			
T	Q	Q_{next}	Comment	Q	Q_{next}	T	Comment
0	0	0	hold state (no clk)	0	0	0	No change
0	1	1	hold state (no clk)	1	1	0	No change
1	0	1	toggle	0	1	1	Complement
1	1	0	toggle	1	0	1	Complement

Flip flop operating characteristics:

The operation characteristics specify the performance, operating requirements, and operating limitations of the circuits. The operation characteristics mentions here apply to all flip-flops regardless of the particular form of the circuit.

Propagation Delay Time: is the interval of time required after an input signal has been applied for the resulting output change to occur.

Set-up Time: is the minimum interval required for the logic levels to be maintained constantly on the inputs (J and K, or S and R, or D) prior to the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop.

Hold Time: is the minimum interval required for the logic levels to remain on the inputs after the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop.

Maximum Clock Frequency: is the highest rate that a flip-flop can be reliably triggered. Power Dissipation: is the total power consumption of the device. It is equal to product of supply voltage (Vcc) and the current (Icc).

$P=V_{cc}.I_{cc}$

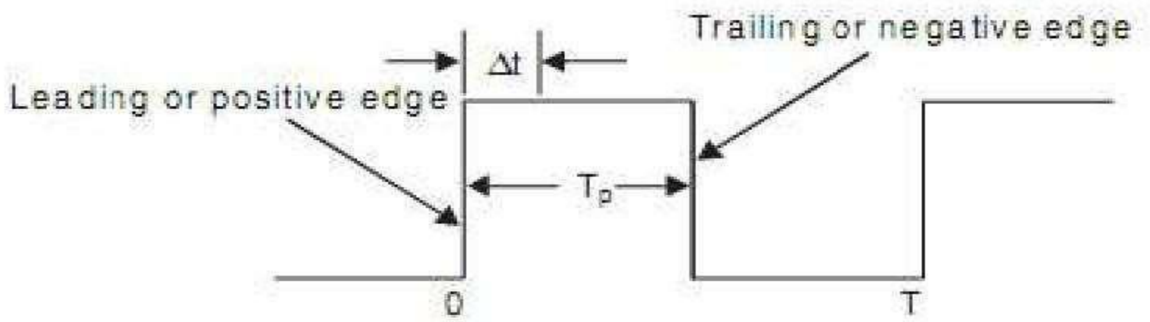
The power dissipation of a flip flop is usually in mW.

Pulse Widths: are the minimum pulse widths specified by the manufacturer for the Clock, SET and CLEAR inputs.

Clock transition times: for reliable triggering, the clock waveform transition times should be kept very short. If the clock signal takes too long to make the transitions from one level to other, the flip flop may either triggering erratically or not trigger at all.

Race around Condition

The inherent difficulty of an S-R flip-flop (i.e., S = R = 1) is eliminated by using the feedback connections from the outputs to the inputs of gate 1 and gate 2 as shown in Figure. Truth tables in figure were formed with the assumption that the inputs do not change during the clock pulse (CLK = 1). But the consideration is not true because of the feedback connections



Consider, for example, that the inputs are J = K = 1 and Q = 1, and a pulse as shown in Figure is applied at the clock input.

After a time interval t equal to the propagation delay through two NAND gates in series, the outputs will change to Q = 0. So now we have J = K = 1 and Q = 0.

After another time interval of t the output will change back to Q = 1. Hence, we conclude that for the time duration of tP of the clock pulse, the output will oscillate between 0 and 1. Hence, at the end of the clock pulse, the value of the output is not certain. This situation is referred to as a race-around condition.

Generally, the propagation delay of TTL gates is of the order of nanoseconds. So if the clock pulse is of the order of microseconds, then the output will change thousands of times within the clock pulse.

This race-around condition can be avoided if tp< t < T. Due to the small propagation delay of the ICs it may be difficult to satisfy the above condition.

A more practical way to avoid the problem is to use the master-slave (M-S) configuration as discussed below.

Applications of flip-flops:

Frequency Division: When a pulse waveform is applied to the clock input of a J-K flip-flop that is connected to toggle, the Q output is a square wave with half the frequency of the clock input. If more flip-flops are connected together as shown in the figure below, further division of the clock frequency can be achieved

. Parallel data storage: a group of flip-flops is called register. To store data of N bits, N flip-flops are required. Since the data is available in parallel form. When a clock pulse is applied to all flip-flops simultaneously, these bits will transfer will be transferred to the Q outputs of the flip flops.

Serial data storage: to store data of N bits available in serial form, N number of D-flip-flops is connected in cascade. The clock signal is connected to all the flip-flops. The serial data is applied to the D input terminal of the first flip-flop.

Transfer of data: data stored in flip-flops may be transferred out in a serial fashion, i.e., bit-by-bit from the output of one flip-flops or may be transferred out in parallel form.

Excitation Tables:

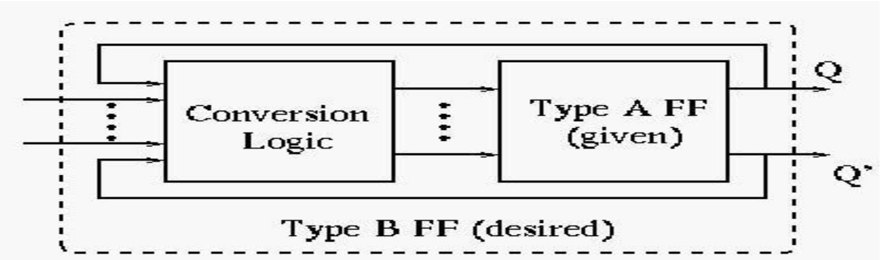
Previous State -> Present State	D
0 -> 0	0
0 -> 1	1
1 -> 0	0
1 -> 1	1

Previous State -> Present State	J	K
0 -> 0	0	X
0 -> 1	1	X
1 -> 0	X	1
1 -> 1	X	0

Previous State -> Present State	S	R
0 -> 0	0	X
0 -> 1	1	0
1 -> 0	0	1
1 -> 1	X	0

Previous State -> Present State	T
0 -> 0	0
0 -> 1	1
1 -> 0	1
1 -> 1	0

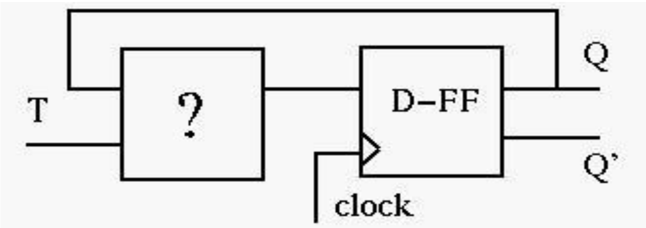
Conversions of flip-flops:



The key here is to use the excitation table, which shows the necessary triggering signal (S,R,J,K, D and T) for a desired flip-flop state transition :

Q_t	Q_{t+1}	S	R	J	K	D	T
0	0	0	x	0	x	0	0
0	1	1	0	1	x	1	1
1	0	0	1	x	1	0	1
1	1	x	0	x	0	1	0

Convert a D-FF to a T-FF:



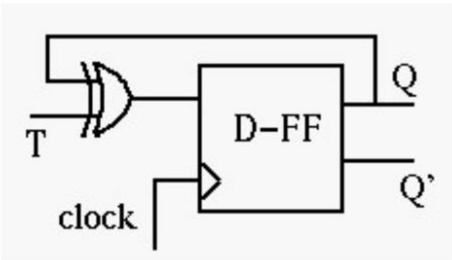
We need to design the circuit to generate the triggering signal D as a function of T and Q: . Consider the excitation table:

$D = f(T, Q).$

Q_t	Q_{t+1}	T	D
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

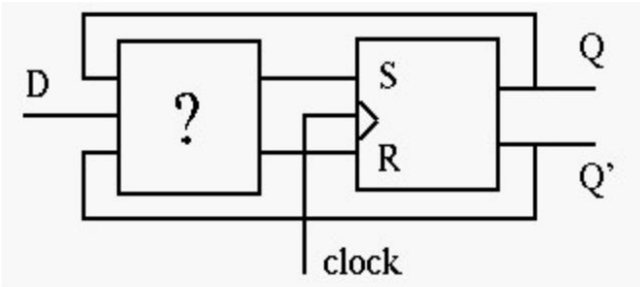
Treating as a function of and current FF state , we have

$$D = T'Q + TQ' = T \oplus Q$$



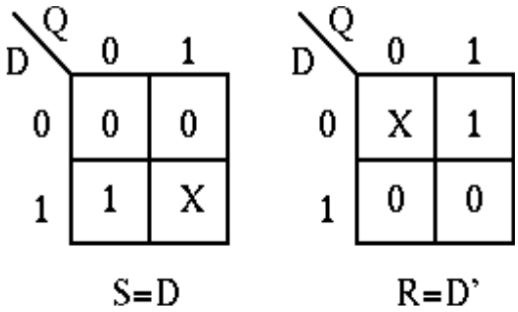
Convert a RS-FF to a D-FF:

We need to design the circuit to generate the triggering signals S and R as functions of and consider the excitation table:

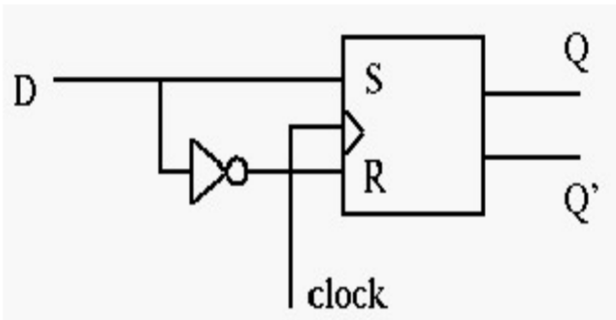


Q_t	Q_{t+1}	D	S	R
0	0	0	0	x
0	1	1	1	0
1	0	0	0	1
1	1	1	x	0

The desired signal and can be obtained as functions of and current FF state from the Karnaugh maps:



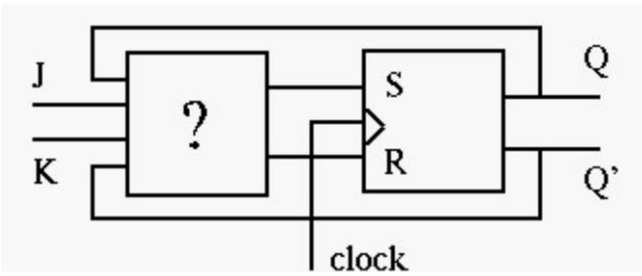
$S = D, \quad R = D'$



Convert a RS-FF to a JK-FF:

We need to design the circuit to generate the triggering signals S and R as functions of, J, K.

Consider the excitation table: The desired signal and as functions of, and current FF state can be obtained from the Karnaugh maps:



Q_t	Q_{t+1}	J	K	S	R
0	0	0	x	0	x
0	1	1	x	1	0
1	0	x	1	0	1
1	1	x	0	x	0

K-maps:

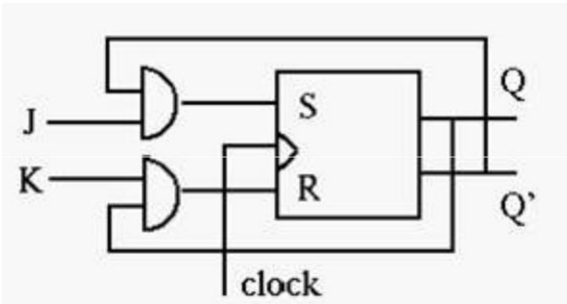
K \ QJ	00	01	11	10
0	0	1	X	X
1	0	1	0	0

S=Q'J

K \ QJ	00	01	11	10
0	X	0	0	0
1	X	0	1	1

R=QK

$$S = Q'J, \quad R = QK$$



The Master-Slave JK Flip-flop:

The Master-Slave Flip-Flop is basically two gated SR flip-flops connected together in a series configuration with the slave having an inverted clock pulse. The outputs from Q and Q from the "Slave" flip-flop are fed back to the inputs of the "Master" with the outputs of the "Master" flip-flop being connected to the two inputs of the "Slave" flip-flop. This feedback configuration from the slave's output to the master's input gives the characteristic toggle of the JK flip-flop as shown below.

The input signals J and K are connected to the gated "master" SR flip-flop which "locks" the input condition while the clock (Clk) input is "HIGH" at logic level "1". As the clock input of the "slave" flip-flop is the inverse (complement) of the "master" clock input, the "slave" SR flip-flop does not toggle. The outputs from the "master" flip-flop are only "seen" by the gated "slave" flip-flop when the clock input goes "LOW" to logic level "0". When the clock is "LOW", the outputs from the "master" flip-flop are latched and any additional changes to its inputs are ignored. The gated "slave" flip-flop now responds to the state of its inputs passed over by the "master" section. Then on the "Low-to-High" transition of the clock pulse the inputs of the "master" flip-flop are fed through to the gated inputs of the "slave" flip-flop and on the "High-to-Low" transition the same inputs are reflected on the output of the "slave" making this type of flip-flop edge or pulse-triggered. Then, the circuit accepts input data when the clock signal is "HIGH", and passes the data to the output on the falling-edge of the clock signal. In other words, the Master-Slave JK Flip-flop is a "Synchronous" device as it only passes data with the timing of the clock signal.

MODULE-V:
Sequential Logic Circuits - II

Sequential Circuit Design

Steps in the design process for sequential circuits
State Diagrams and State Tables
Examples

Steps in Design of a Sequential Circuit

1. Specification – A description of the sequential circuit. Should include a detailing of the inputs, the outputs, and the operation. Possibly assumes that you have knowledge of digital system basics.
2. Formulation: Generate a state diagram and/or a state table from the statement of the problem.
3. State Assignment: From a state table assign binary codes to the states.
4. Flip-flop Input Equation Generation: Select the type of flip-flop for the circuit and generate the needed input for the required state transitions
5. Output Equation Generation: Derive output logic equations for generation of the output from the inputs and current state.
6. Optimization: Optimize the input and output equations. Today, CAD systems are typically used for this in real systems.
7. Technology Mapping: Generate a logic diagram of the circuit using ANDs, ORs, Inverters, and F/Fs.
8. Verification: Use a HDL to verify the design.

Sequential machines are typically classified as either a Mealy machine or a Moore machine implementation.

Moore machine: The outputs of the circuit depend only upon the current state of the circuit.

Mealy machine: The outputs of the circuit depend upon both the current state of the circuit and the inputs.

An example to go through the steps

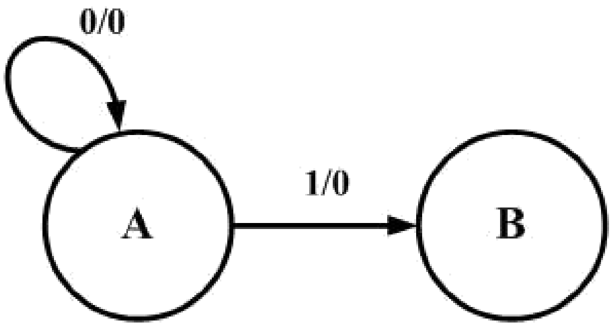
The specification: The circuit will have one input, X, and one output, Z. The output Z will be 0 except when the input sequence 1101 are the last 4 inputs received on X. In that case it will be a 1

Generation of a state diagram

Create states and meaning for them.

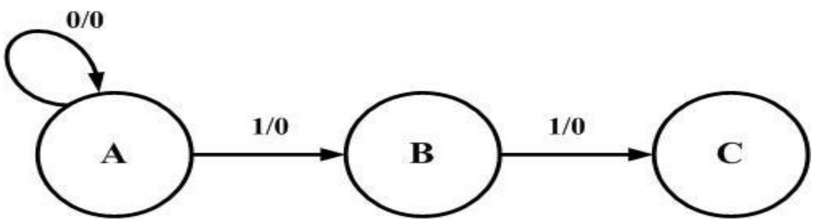
State A – the last input was a 0 and previous inputs unknown. Can also be the reset state. State B – the last input was a 1 and the previous input was a 0. The start of a new sequence possibly.

Capture this in a state diagram



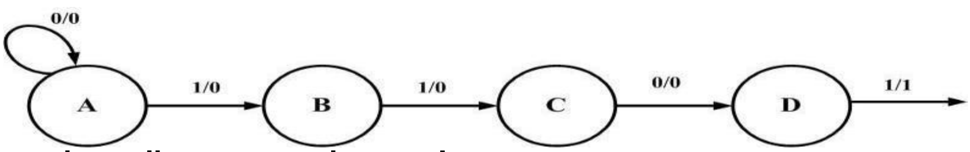
- Capture this in a state diagram

Circles represent the states
Lines and arcs represent the transition between states.
The notation Input/output on the line or arc specifies the input that causes this transition and the output for this change of state.
Add a state C – Have detected the input sequence 11 which is the start of the sequence

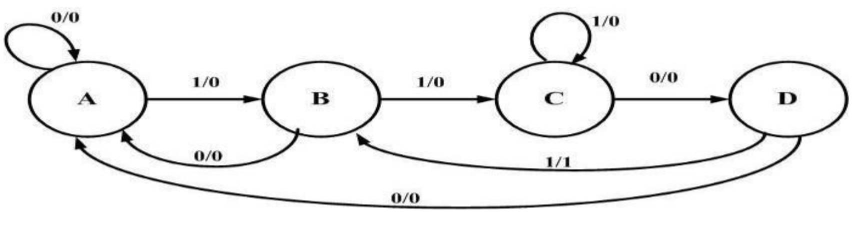


- Add a state D

State D – have detected the 3rd input in the start of a sequence, a 0, now having 110. From State D, if the next input is a 1 the sequence has been detected and a 1 is output.



- The previous diagram was incomplete.
- In each state the next input could be a 0 or a 1. This must be included



The state table

This can be done directly from the state diagram

Prresent State	Next State X=0	Next State X=1	Output X=0	Output X=1
A	A	B	0	0
B	A	C	0	0
C	D	C	0	0
D	A	B	0	1

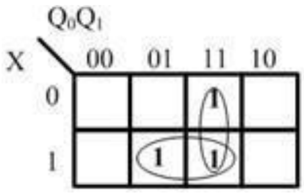
Now need to do a state assignment

Will select a gray encoding
For this state A will be encoded 00, state B 01, state C 11 and state D 10

Prresent State	Next State X=0	Next State X=1	Output X=0	Output X=1
00	00	01	0	0
01	00	11	0	0
11	10	11	0	0
10	00	01	0	1

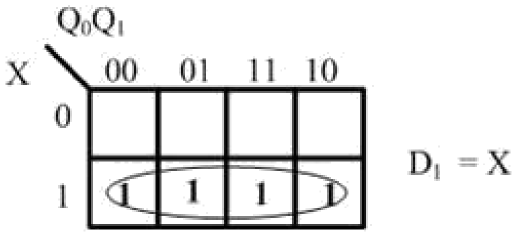
Flip-flop input equations

Generate the equations for the flip-flop inputs Generate the D_0 equation



$$D_0 = Q_0 Q_1 + X Q_1$$

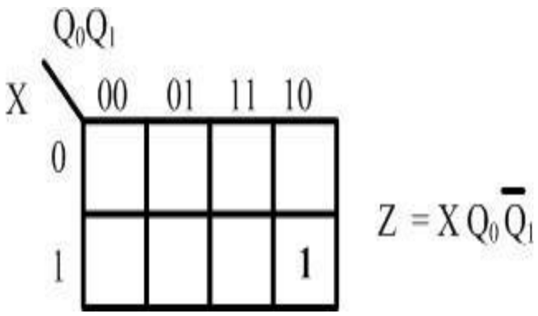
Generate the D_1 equation



The output equation

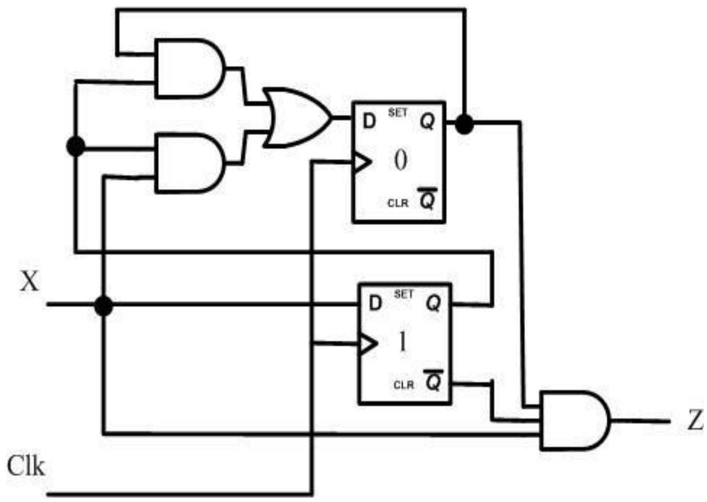
The next step is to generate the equation for the output Z and what is needed to generate it.

Create a K-map from the truth table.



Now map to a circuit

The circuit has 2 D type F/Fs



Shift registers:

In digital circuits, a shift register is a cascade of flip-flops sharing the same clock, in which the output of each flip-flop is connected to the "data" input of the next flip-flop in the chain, resulting in a circuit that shifts by one position the "bit array" stored in it, shifting in the data present at its input and shifting out the last bit in the array, at each transition of the clock input. More generally, a shift register may be multidimensional, such that its "data in" and stage outputs are themselves bit arrays: this is implemented simply by running several shift registers of the same bit-length in parallel.

Shift registers can have both parallel and serial inputs and outputs. These are often configured as serial-in, parallel-out (SIPO) or as parallel-in, serial-out (PISO). There are also types that have both serial and parallel input and types with serial and parallel output. There are also bi-directional shift registers which allow shifting in both directions: L→R or R→L. The serial input and last output of a shift register can also be connected to create a circular shift register

Shift registers are a type of logic circuits closely related to counters. They are basically for the storage and transfer of digital data.

Buffer register:

The buffer register is the simple set of registers. It is simply stores the binary word. The buffer may be controlled buffer. Most of the buffer registers used D Flip-flops.

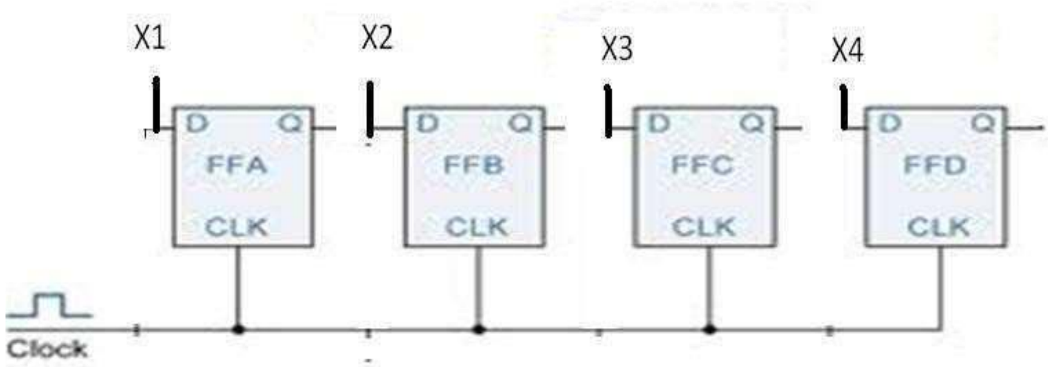


Figure: logic diagram of 4-bit buffer register

The figure shows a 4-bit buffer register. The binary word to be stored is applied to the data terminals. On the application of clock pulse, the output word becomes the same as the word applied at the terminals. i.e., the input word is loaded into the register by the application of clock pulse.

When the positive clock edge arrives, the stored word becomes:

$Q_4Q_3Q_2Q_1=X_4X_3X_2X_1$

$Q=X$

Controlled buffer register:

If goes LOW, all the FFs are RESET and the output becomes, Q=0000.

When is HIGH, the register is ready for action. LOAD is the control input.

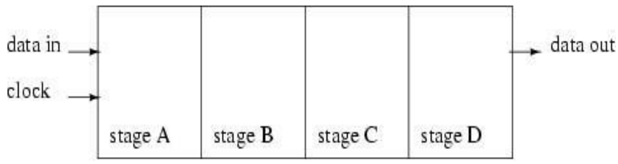
When LOAD is HIGH, the data bits X can reach the D inputs of FF's.

$Q_4Q_3Q_2Q_1=X_4X_3X_2X_1$

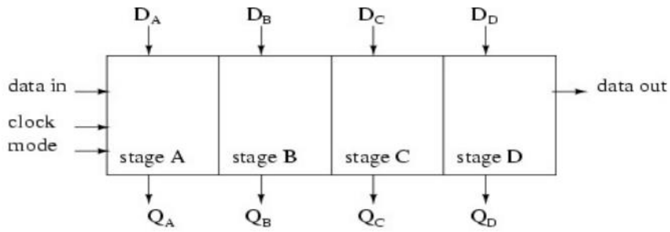
$Q=X$

When load is low, the X bits cannot reach the FF's.

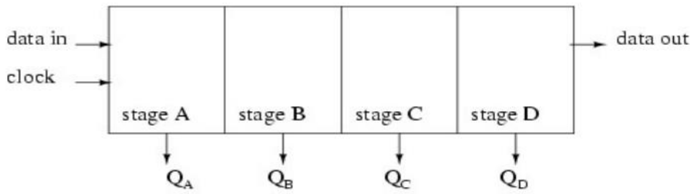
Data transmission in shift registers:



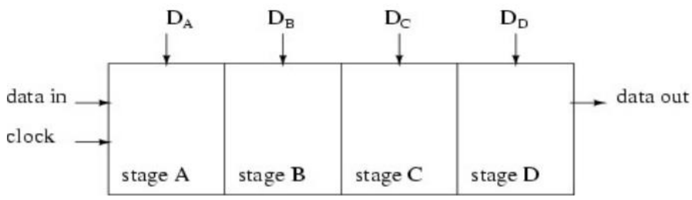
Serial-in, serial-out shift register with 4-stages



Parallel-in, parallel-out shift register with 4-stages



Serial-in, parallel-out shift register with 4-stages



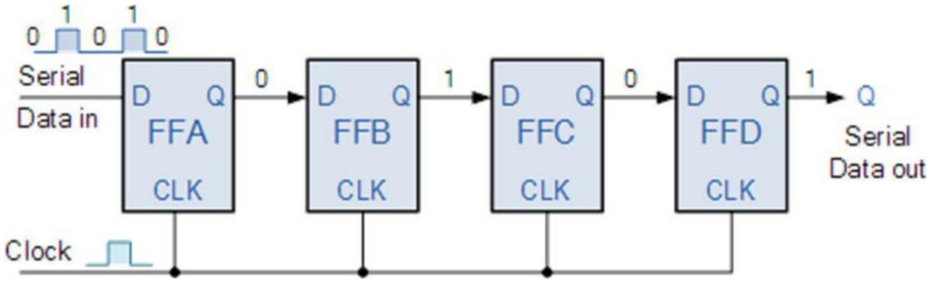
Parallel-in, serial-out shift register with 4-stages

A number of ff's connected together such that data may be shifted into and shifted out of them is called shift register. data may be shifted into or out of the register in serial form or in parallel form. There are four basic types of shift registers.

1. **Serial in, serial out, shift right, shift registers**
2. **Serial in, serial out, shift left, shift registers**
3. **Parallel in, serial out shift registers**
4. **Parallel in, parallel out shift registers**

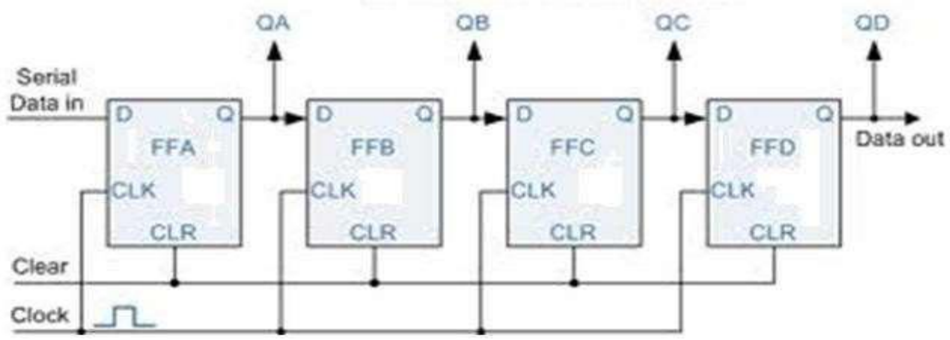
Serial IN, serial OUT, shift right, shift left register:

The logic diagram of 4-bit serial in serial out, right shift register with four stages. The register can store four bits of data. Serial data is applied at the input D of the first FF. the Q output of the first FF is connected to the D input of another FF. the data is outputted from the Q terminal of the last FF.



When serial data is transferred into a register, each new bit is clocked into the first FF at the positive going edge of each clock pulse. The bit that was previously stored by the first FF is transferred to the second FF. the bit that was stored by the Second FF is transferred to the third FF.

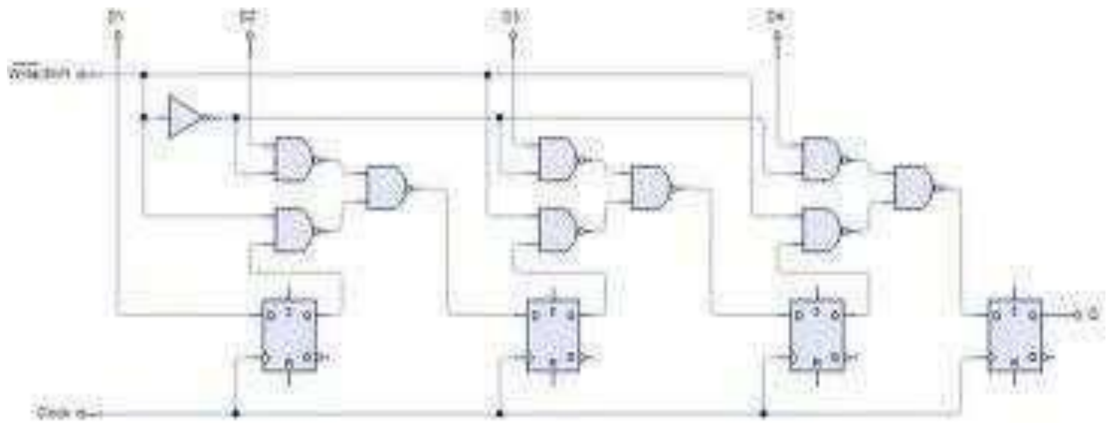
Serial-in, parallel-out, shift register:



In this type of register, the data bits are entered into the register serially, but the data stored in the register is shifted out in parallel form.

Once the data bits are stored, each bit appears on its respective output line and all bits are available simultaneously, rather than on a bit-by-bit basis with the serial output. The serial-in, parallel out, shift register can be used as serial-in, serial out, shift register if the output is taken from the Q terminal of the last FF.

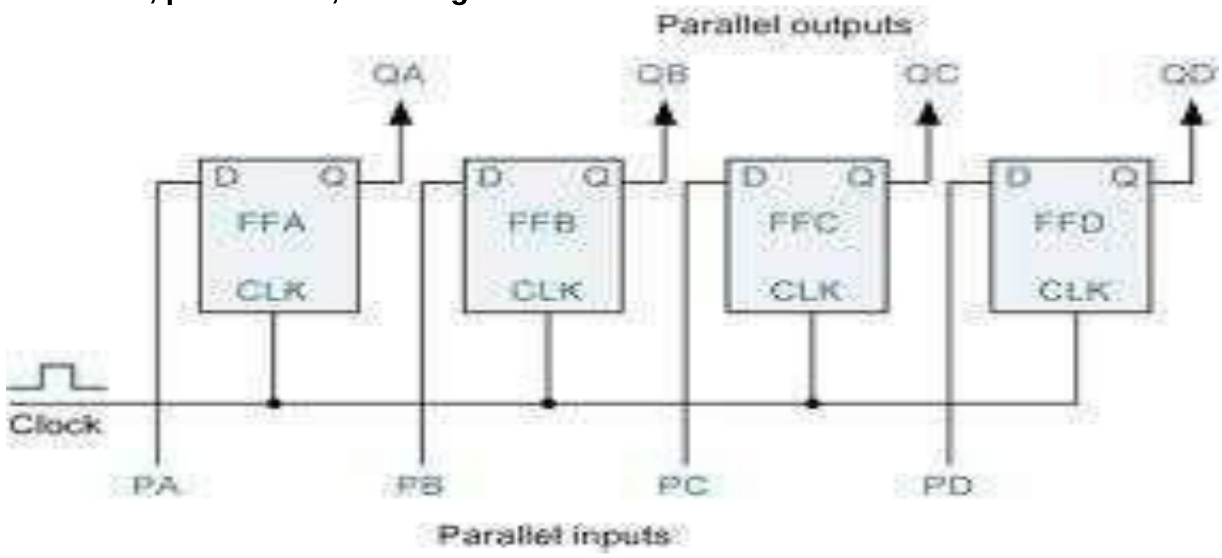
Parallel-in, serial-out, shift register:



For a parallel-in, serial out, shift register, the data bits are entered simultaneously into their respective stages on parallel lines, rather than on a bit-by-bit basis on one line as with serial data bits are transferred out of the register serially. On a bit-by-bit basis over a single line.

There are four data lines A,B,C,D through which the data is entered into the register in parallel form. The signal shift/ load allows the data to be entered in parallel form into the register and the data is shifted out serially from terminalQ4

Parallel-in, parallel-out, shift register



In a parallel-in, parallel-out shift register, the data is entered into the register in parallel form, and also the data is taken out of the register in parallel form. Data is applied to the D input terminals of the FF's. When a clock pulse is applied, at the positive going edge of the pulse, the D inputs are shifted into the Q outputs of the FFs. The register now stores the data. The stored data is available instantaneously for shifting out in parallel form.

Bidirectional shift register:

A bidirectional shift register is one which the data bits can be shifted from left to right or from right to left. A fig shows the logic diagram of a 4-bit serial-in, serial out, bidirectional shift register. Right/left is the mode signal, when right /left is a 1, the logic circuit works as a shift-register.the bidirectional operation is achieved by using the mode signal and two NAND gates and one OR gate for each stage.

A HIGH on the right/left control input enables the AND gates G1, G2, G3 and G4 and disables the AND gates G5,G6,G7 and G8, and the state of Q output of each FF is passed through the gate to the D input of the following FF. when a clock pulse occurs, the data bits are then effectively shifted one place to the right. A LOW on the right/left control inputs enables the AND gates G5, G6, G7 and G8 and disables the And gates G1, G2, G3 and G4 and the Q output of each FF is passed to the D input of the preceding FF. when a clock pulse occurs, the data bits are then effectively shifted one place to the left. Hence, the circuit works as a bidirectional shift register

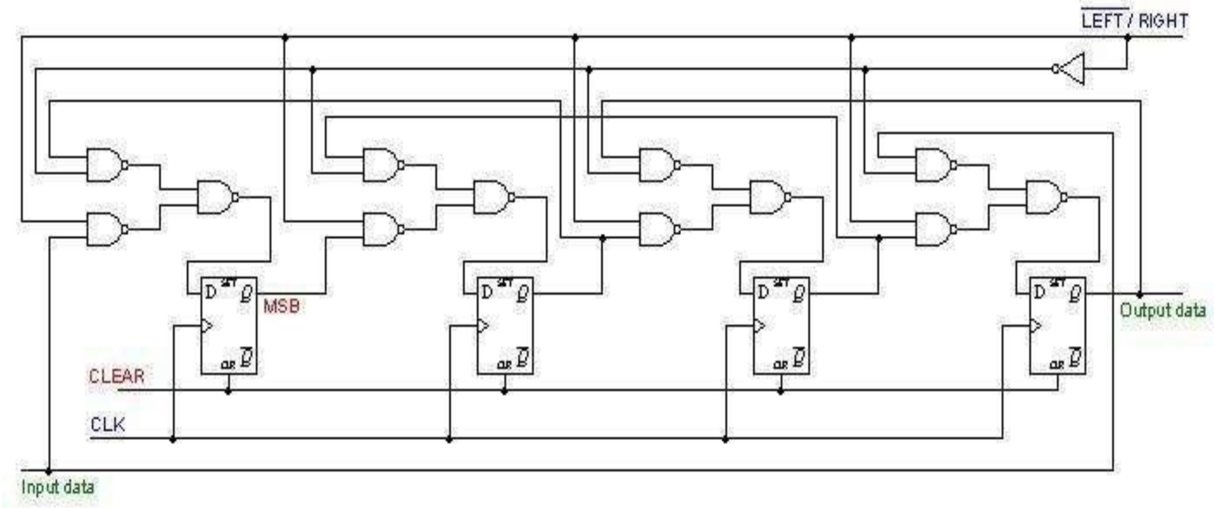


Figure: logic diagram of a 4-bit bidirectional shift register

Universal shift register:

A register is capable of shifting in one direction only is a unidirectional shift register. One that can shift both directions is a bidirectional shift register. If the register has both shifts and parallel load capabilities, it is referred to as a universal shift registers. Universal shift register is a bidirectional register, whose input can be either in serial form or in parallel form and whose output also can be in serial form or I parallel form.
The most general shift register has the following capabilities.

1. **A clear control to clear the register to 0**
2. **A clock input to synchronize the operations**
3. **A shift-right control to enable the shift-right operation and serial input and output lines associated with the shift-right**

4. **A shift-left control to enable the shift-left operation and serial input and output lines associated with the shift-left**
5. **A parallel loads control to enable a parallel transfer and the n input lines associated with the parallel transfer**
6. **N parallel output lines**
7. **A control state that leaves the information in the register unchanged in the presence of the clock.**

A universal shift register can be realized using multiplexers. The below fig shows the logic diagram of a 4-bit universal shift register that has all capabilities. It consists of 4 D flip-flops and four multiplexers. The four multiplexers have two common selection inputs s1 and s0. Input 0 in each multiplexer is selected when S1S0=00, input 1 is selected when S1S0=01 and input 2 is selected when S1S0=10 and input 4 is selected when S1S0=11. The selection inputs control the mode of operation of the register according to the functions entries. When S1S0=0, the present value of the register is applied to the D inputs of flip-flops. The condition forms a path from the output of each flip-flop into the input of the same flip-flop. The next clock edge transfers into each flip-flop the binary value it held previously, and no change of state occurs. When S1S0=01, terminal 1 of the multiplexer inputs have a path to the D inputs of the flip-flop. This causes a shift-right operation, with serial input transferred into flip-flopA4. When S1S0=10, a shift left operation results with the other serial input going into flip-flop A1. Finally when S1S0=11, the binary information on the parallel input lines is transferred into the register simultaneously during the next clock cycle

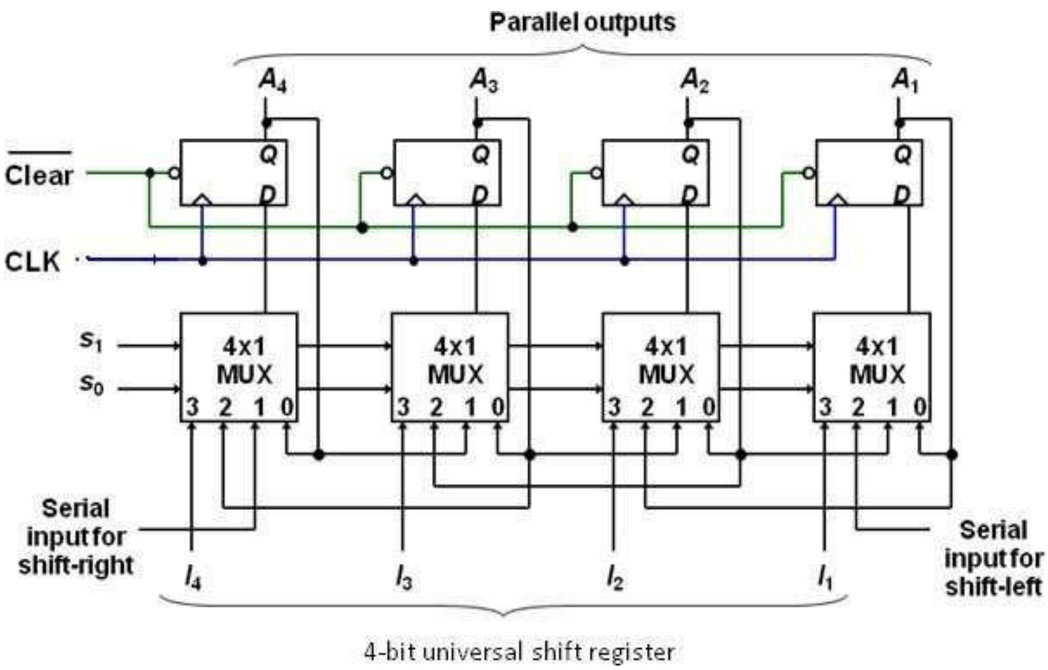


Figure: logic diagram 4-bit universal shift register

Function table for theregister

		mode control
S0	**S1**	**register operation**
0	**0**	**No change**
0	**1**	**Shift Right**
1	**0**	**Shift left**
1	**1**	**Parallel load**

Counters:

Counter is a device which stores (and sometimes displays) the number of times particular event or process has occurred, often in relationship to a clock signal. A Digital counter is a set of flip flops whose state change in response to pulses applied at the input to the counter. Counters may be asynchronous counters or synchronous counters. Asynchronous counters are also called ripple counters

In electronics counters can be implemented quite easily using register-type circuits such as the flip-flops and a wide variety of classifications exist:

Asynchronous (ripple) counter – changing state bits are used as clocks to subsequent state flip-flops
Synchronous counter – all state bits change under control of a
singleclock Decade counter – counts through ten states per stage
Up/down counter – counts both up and down, under command of a control input Ring
counter – formed by a shift register with feedback connection in a ring
Johnson counter – a twisted ring counter

- **Cascaded counter**
- **Modulus counter.**

Each is useful for different applications. Usually, counter circuits are digital in nature, and count in natural binary Many types of counter circuits are available as digital building blocks, for example a number of chips in the 4000 series implement different counters.

Occasionally there are advantages to using a counting sequence other than the natural binary sequence such as the binary coded decimal counter, a linear feed-back shift register counter, or a gray-code counter.

Counters are useful for digital clocks and timers, and in oven timers, VCR clocks, etc.

Asynchronous counters:

An asynchronous (ripple) counter is a single [JK-type flip-flop](), with its J (data) input fed from its own inverted output. This circuit can store one bit, and hence can count from zero to one before it overflows (starts over from 0). This counter will increment once for every clock cycle and takes two clock cycles to overflow, so every cycle it will alternate between a transition from 0 to 1 and a transition from 1 to 0. Notice that this creates a new clock with a 50% [duty cycle]() at exactly half the frequency of the input clock. If this output is then used as the clock signal for a similarly arranged D flip-flop (remembering to invert the output to the input), one will get another 1 bit counter that counts half as fast. Putting them together yields a two-bit counter:

Two-bit ripple up-counter using negative edge triggered flip flop:

Two bit ripple counter used two flip-flops. There are four possible states from 2 – bit up-counting I.e. 00, 01, 10 and 11.

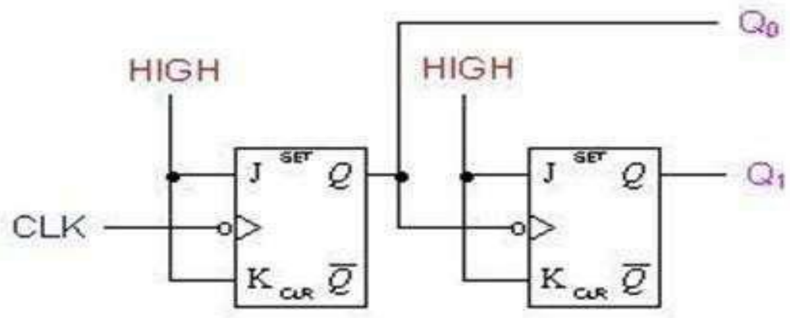
- **The counter is initially assumed to be at a state 00 where the outputs of the tow flip-flops are noted as Q_1Q_0. Where Q_1 forms the MSB and Q_0 forms the LSB.**

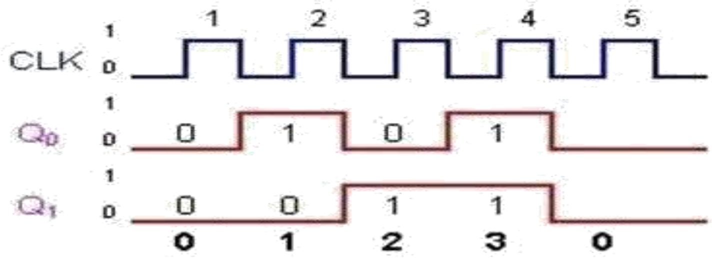
- **For the negative edge of the first clock pulse, output of the first flip-flop FF_1 toggles its state. Thus Q_1 remains at 0 and Q_0 toggles to 1 and the counter state are now read as 01.**

- **During the next negative edge of the input clock pulse FF_1 toggles and $Q_0 = 0$. The output Q0 being a clock signal for the second flip-flop FF_2 and the present transition acts as a negative edge for FF_2 thus toggles its state $Q_1 = 1$. The counter state is now read as 10.**

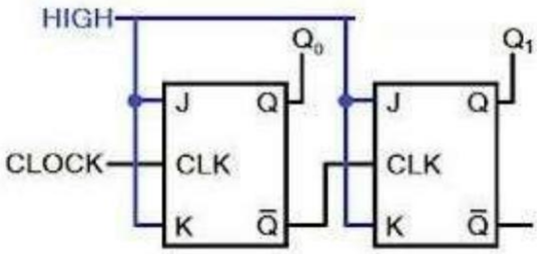
- **For the next negative edge of the input clock to FF_1 output Q0 toggles to 1. But this transition from 0 to 1 being a positive edge for FF_2 output Q_1 remains at 1. The counter state is now read as 11.**

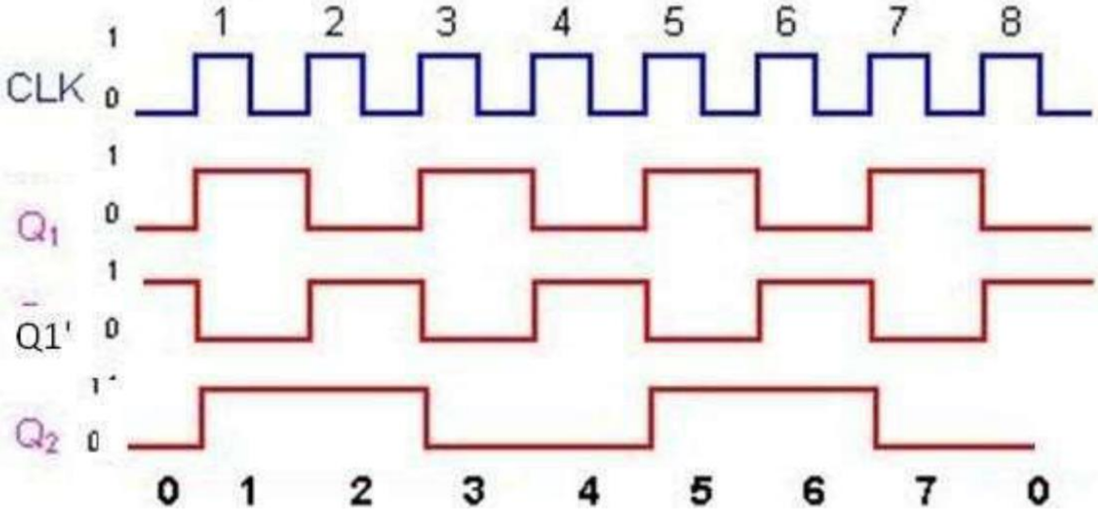
- **For the next negative edge of the input clock, Q_0 toggles to 0. This transition from 1 to 0 acts as a negative edge clock for FF_2 and its output Q_1 toggles to 0. Thus the starting state 00 is attained. Figure shown below**





Two-bit ripple down-counter using negative edge triggered flip flop:





A 2-bit down-counter counts in the order 0,3,2,1,0,1…….,i.e, 00,11,10,01,00,11 …..,etc. the above fig. shows ripple down counter, using negative edge triggered J-K FFs and its timing diagram.
For down counting, Q1‘ of FF1 is connected to the clock of Ff2. Let initially all the FF1 toggles, so, Q1 goes from a 0 to a 1 and Q1‘ goes from a 1 to a 0.

The negative-going signal at Q1' is applied to the clock input of FF2, toggles Ff2 and, therefore, Q2 goes from a 0 to a 1.so, after one clock pulse Q2=1 and Q1=1, I.e., the state of the counter is 11.

At the negative-going edge of the second clock pulse, Q1 changes from a 1 to a 0 and Q1' from a 0 to a 1.

This positive-going signal at Q1' does not affect FF2 and, therefore, Q2 remains at a 1. Hence , the state of the counter after second clock pulse is 10

At the negative going edge of the third clock pulse, FF1 toggles. So Q1, goes from a 0 to a 1 and Q1' from 1 to 0. This negative going signal at Q1' toggles FF2 and, so, Q2 changes from 1 to 0, hence, the state of the counter after the third clock pulse is 01.

At the negative going edge of the fourth clock pulse, FF1 toggles. So Q1, goes from a 1 to a 0 and Q1' from 0 to 1. . This positive going signal at Q1' does not affect FF2 and, so, Q2 remains at 0, hence, the state of the counter after the fourth clock pulse is 00.

Two-bit ripple up-down counter using negative edge triggered flip flop:

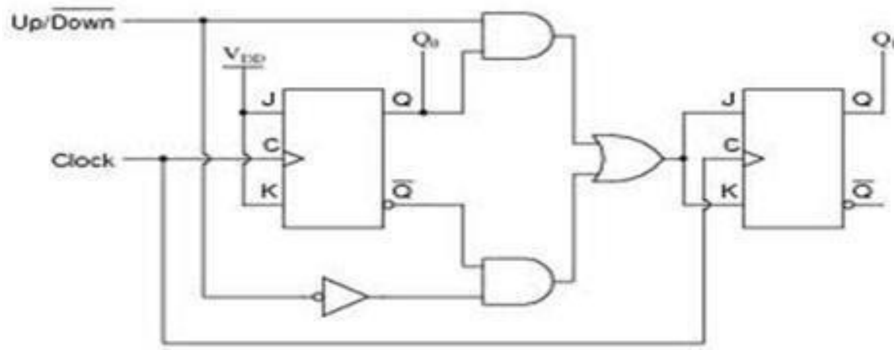


Figure: asynchronous 2-bit ripple up-down counter using negative edge triggered flip flop:

As the name indicates an up-down counter is a counter which can count both in upward and downward directions. An up-down counter is also called a forward/backward counter or a bidirectional counter. So, a control signal or a mode signal M is required to choose the direction of count. When M=1 for up counting, Q1 is transmitted to clock of FF2 and when M=0 for down counting, Q1' is transmitted to clock of FF2. This is achieved by using two AND gates and one OR gates. The external clock signal is applied to FF1.

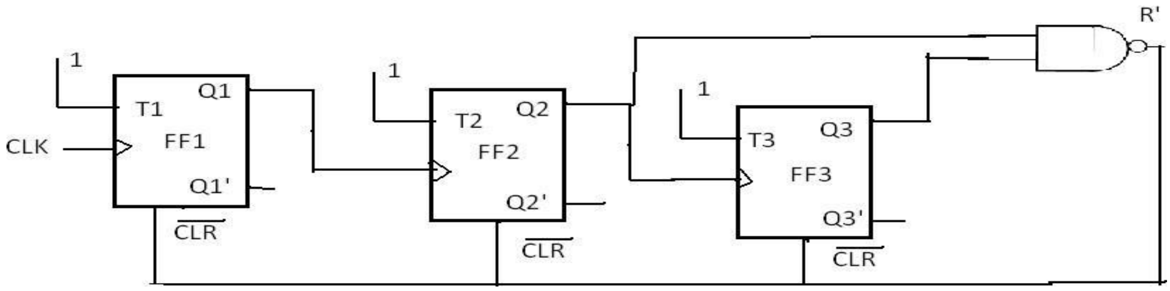
Clock signal to FF2= (Q1.Up)+(Q1'. Down)= Q1m+Q1'M'

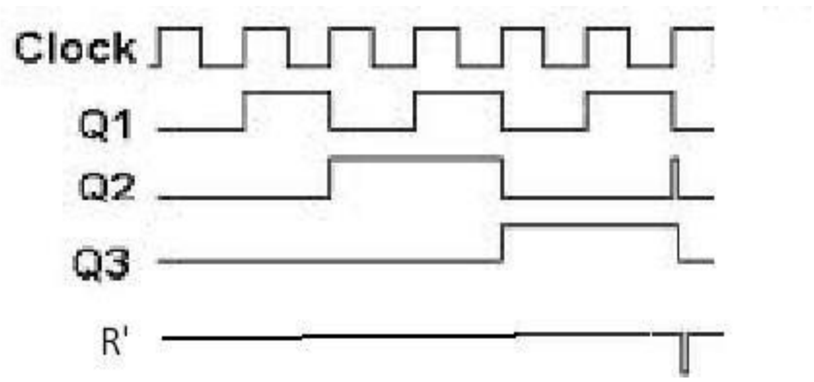
Design of Asynchronous counters:

To design a asynchronous counter, first we write the sequence , then tabulate the values of reset signal R for various states of the counter and obtain the minimal expression for R and R' using K-Map or any other method. Provide a feedback such that R and R' resets all the FF's after the desired count

Design of a Mod-6 asynchronous counter using T FFs:

A mod-6 counter has six stable states 000, 001, 010, 011, 100, and 101. When the sixth clock pulse is applied, the counter temporarily goes to 110 state, but immediately resets to 000 because of the feedback provided. it is —divide by-6-counter‖, in the sense that it divides the input clock frequency by 6.it requires three FFs, because the smallest value of n satisfying the condition$N \le 2^n$ is n=3; three FFs can have 8 possible states, out of which only six are utilized and the remaining two states 110and 111, are invalid. If initially the counter is in 000 state, then after the sixth clock pulse, it goes to 001, after the second clock pulse, it goes to 010, and so on.





After sixth clock pulse it goes to 000. For the design, write the truth table with present state outputs Q3, Q2 and Q1 as the variables, and reset R as the output and obtain an expression for R in terms of Q3, Q2, and Q1that decides the feedback into be provided. From the truth table, R=Q3Q2. For active-low Reset, R‘ is used. The reset pulse is of very short duration, of the order of nanoseconds and it is equal to the propagation delay time of the NAND gate used. The expression for R can also be determined as follows.

R=0 for 000 to 101, R=1 for 110, and R=X=for111
Therefore,
R=Q3Q2Q1‘+Q3Q2Q1=Q3Q2

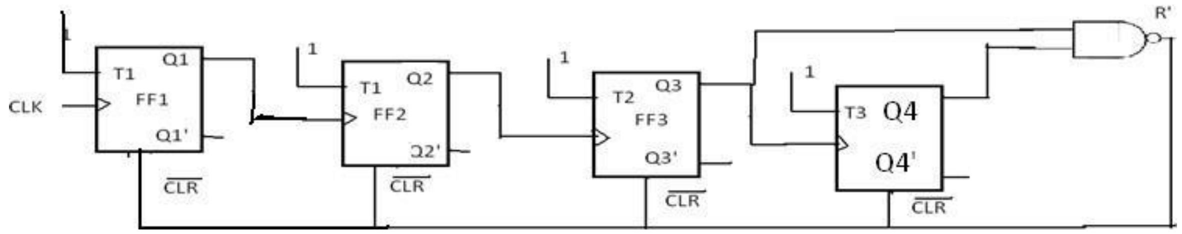
The logic diagram and timing diagram of Mod-6 counter is shown in the above fig.

The truth table is as shown in below.

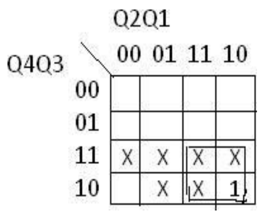
After pulses	States			
	Q3	Q2	Q1	R
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
	↓	↓	↓	
	0	0	0	0
7	0	0	0	0

Design of a mod-10 asynchronous counter using T-flip-flops:

A mod-10 counter is a decade counter. It also called a BCD counter or a divide-by-10 counter. It requires four flip-flops (condition 10 ≤2^n is n=4). So, there are 16 possible states, out of which ten are valid and remaining six are invalid. The counter has ten stable state, 0000 through 1001, i.e., it counts from 0 to 9. The initial state is 0000 and after nine clock pulses it goes to 1001. When the tenth clock pulse is applied, the counter goes to state 1010 temporarily, but because of the feedback provided, it resets to initial state 0000. So, there will be a glitch in the waveform of Q2. The state 1010 is a temporary state for which the reset signal R=1, R=0 for 0000 to 1001, and R=C for 1011 to 1111.



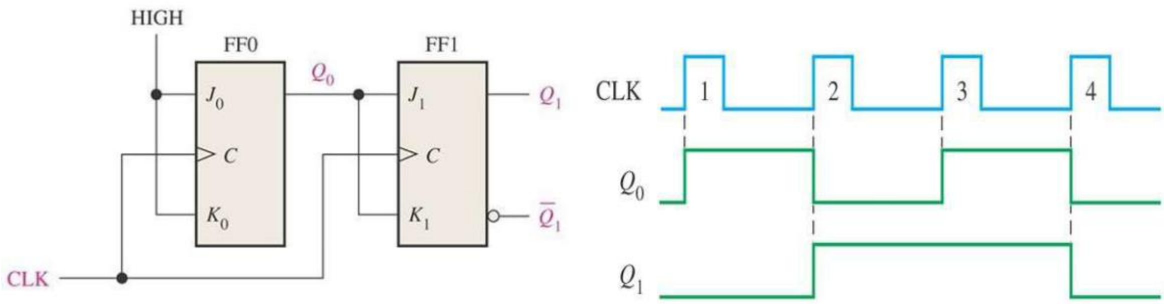
The count table and the K-Map for reset are shown in fig. from the K-Map R=Q4Q2. So, feedback is provided from second and fourth FFs. For active –HIGH reset, Q4Q2 is applied to the clear terminal. For active-LOW reset 4 2 is connected isof all Flip=flops.



After pulses	Count Q4	Q3	Q2	Q1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	0	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	0	1	0	1
10	0	0	0	0

Synchronous counters:

Asynchronous counters are serial counters. They are slow because each FF can change state only if all the preceding FFs have changed their state. if the clock frequency is very high, the asynchronous counter may skip some of the states. This problem is overcome in synchronous counters or parallel counters. Synchronous counters are counters in which all the flip flops are triggered simultaneously by the clock pulses Synchronous counters have a common clock pulse applied simultaneously to all flip flops. A 2-Bit Synchronous Binary Counter



Design of synchronous counters:

For a systematic design of synchronous counters. The following procedure is used.

Step 1:State Diagram: draw the state diagram showing all the possible states state diagram which also be called nth transition diagrams, is a graphical means of depicting the sequence of states through which the counter progresses.

Step2: number of flip-flops: based on the description of the problem, determine the required number n of the flip-flops- the smallest value of n is such that the number of states N≤2^n--- and the desired counting sequence.

Step3: choice of flip-flops excitation table: select the type of flip-flop to be used and write the excitation table. An excitation table is a table that lists the present state (ps) , the next state(ns) and required excitations.

Step4: minimal expressions for excitations: obtain the minimal expressions for the excitations of the FF using K-maps drawn for the excitation of the flip-flops in terms of the present states and inputs.

Step5: logic diagram: draw a logic diagram based on the minimal expressions

Design of a synchronous 3-bit up-down counter using JK flip-flops:

Step1: determine the number of flip-flops required. A 3-bit counter requires three FFs. It has 8 states (000,001,010,011,101,110,111) and all the states are valid. Hence no don‘t cares. For selecting up and down modes, a control or mode signal M is required. When the mode signal M=1 and counts down when M=0. The clock signal is applied to all the FFs simultaneously.

Step2: draw the state diagrams: the state diagram of the 3-bit up-down counter is drawn as

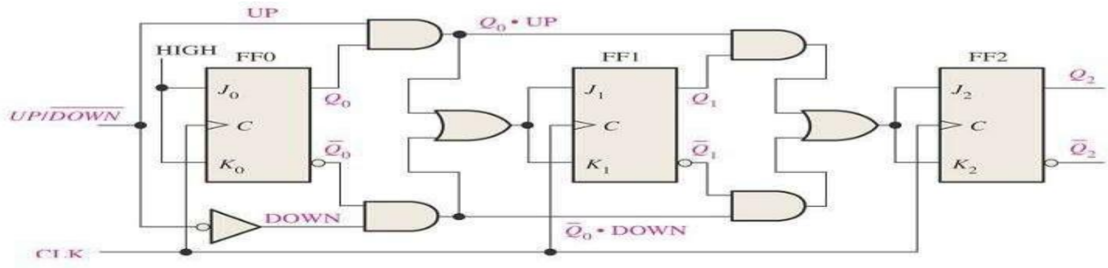
Step3: select the type of flip flop and draw the excitation table: JK flip-flops are selected and the excitation table of a 3-bit up-down counter using JK flip-flops is drawn as shown in fig.

PS			mode	NS			required excitations					
Q3	Q2	Q1	M	Q3	Q2	Q1	J3	K3	J2	K2	J1	K1
0	0	0	0	1	1	1	1	x	1	x	1	x
0	0	0	1	0	0	1	0	x	0	x	1	x
0	0	1	0	0	0	0	0	x	0	x	x	1
0	0	1	1	0	1	0	0	x	1	x	x	1
0	1	0	0	0	0	1	0	x	x	1	1	x
0	1	0	1	0	1	1	0	x	x	0	1	x
0	1	1	0	0	1	0	0	x	x	0	x	1
0	1	1	1	1	0	0	1	x	x	1	x	1
1	0	0	0	0	1	1	x	1	1	x	1	x
1	0	0	1	1	0	1	x	0	0	x	1	x
1	0	1	0	1	0	0	x	0	0	x	x	1
1	0	1	1	1	1	0	x	0	1	x	x	1
1	1	0	0	1	0	1	x	0	x	1	1	x
1	1	0	1	1	1	1	x	0	x	0	1	x
1	1	1	0	1	1	0	x	0	x	0	x	1
1	1	1	1	0	0	0	x	1	x	1	x	1

Step4: obtain the minimal expressions: From the excitation table we can conclude that J1=1 and K1=1, because all the entries for J1and K1 are either X or 1. The K-maps for J3, K3,J2 and K2 based on the excitation table and the minimal expression obtained from them are shown in fig.

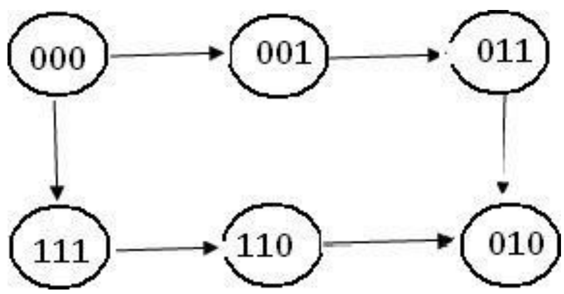
Q3Q2 \ Q1M	00	01	11	10
	1			
			1	
	X	X	X	X
	X	X	X	X

Step5: draw the logic diagram: a logic diagram using those minimal expressions can be drawn as shown in fig.



Design of a synchronous modulo-6 gray cod counter:

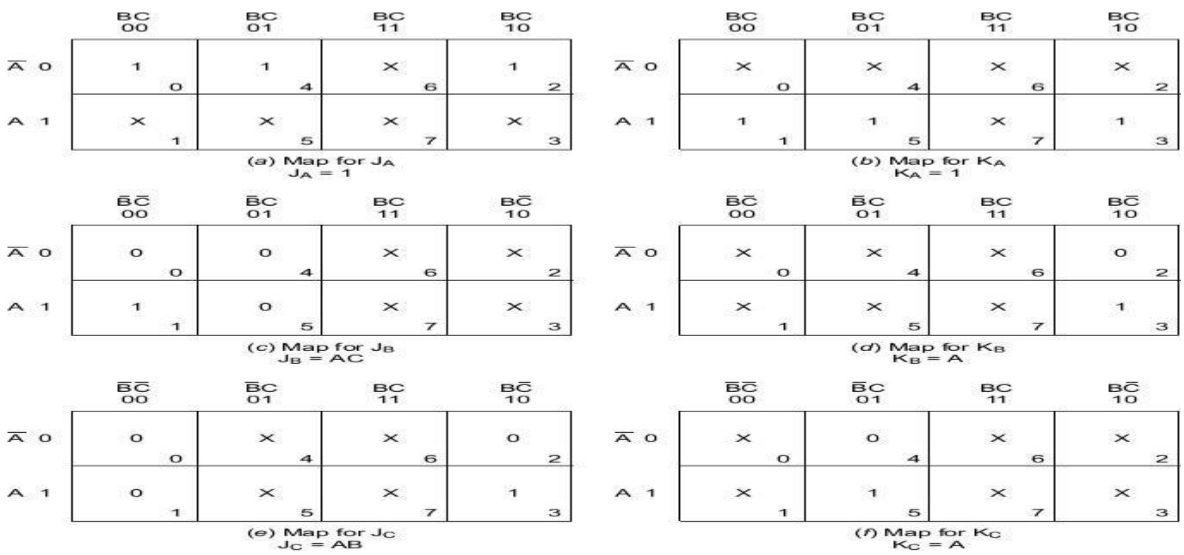
Step 1: the number of flip-flops: we know that the counting sequence for a modulo-6 gray code counter is 000, 001, 011, 010, 110, and 111. It requires n=3FFs ($N \leq 2^n$, i.e., $6 \leq 2^3$). 3 FFs can have 8 states. So the remaining two states 101 and 100 are invalid. The entries for excitation corresponding to invalid states are don't cares.
Step2: the state diagram: the state diagram of the mod-6 gray code converter is drawn as shown in fig.



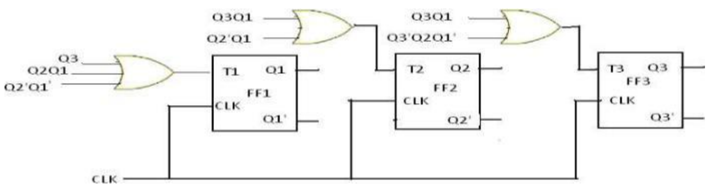
Step3: type of flip-flop and the excitation table: T flip-flops are selected and the excitation table of the mod-6 gray code counter using T-flip-flops is written as shown in fig.

PS			NS			required excitations		
Q3	Q2	Q1	Q3	Q2	Q1	T3	T2	T1
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	0
0	1	1	0	1	0	0	0	1
0	1	0	1	1	0	1	0	0
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

Step4: The minimal expressions: the K-maps for excitations of FFs T3,T2,and T1 in terms of outputs of FFs Q3,Q2, and Q1, their minimization and the minimal expressions for excitations obtained from them are shown if fig



Step5: the logic diagram: the logic diagram based on those minimal expressions is drawn as shown in fig.



Design of a synchronous BCD Up-Down counter using FFs:

Step1: the number of flip-flops: a BCD counter is a mod-10 counter has 10 states (0000 through 1001) and so it requires n=4FFs($N \leq 2^n$,, i.e., $10 \leq 2^4$). 4 FFS can have 16 states. So out of 16 states, six states (1010 through 1111) are invalid. For selecting up and down mode, a control or mode signal M is required. , it counts up when M=1 and counts down when M=0. The clock signal is applied to all FFs.

Step2: the state diagram: The state diagram of the mod-10 up-down counter is drawn as shown in fig.

Step3: types of flip-flops and excitation table: T flip-flops are selected and the excitation table of the modulo-10 up down counter using T flip-flops is drawn as shown in fig.

The remaining minterms are don't cares(∑d(20,21,22,23,24,25,26,37,28,29,30,31)) from the excitation table we can see that T1=1 and the expression for T4,T3,T2 are asfollows.

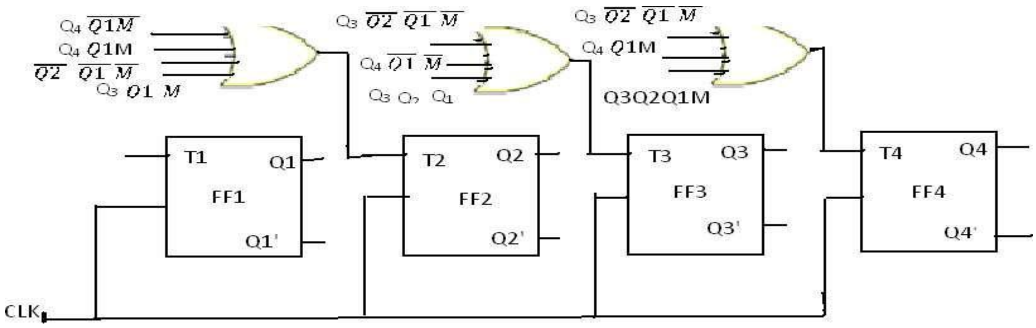
T4=∑m(0,15,16,19)+d(20,21,22,23,24,25,26,27,28,29,30,31)
T3=∑m(7,15,16,8)+d(20,21,22,23,24,25,26,27,28,29,30,31)
T2=∑m(3,4,7,8,11,12,15,16)+d(20,21,22,23,24,25,26,27,28,29,30,31)

PS				mode	NS				required excitations			
Q4	Q3	Q2	Q1	M	Q4	Q3	Q2	Q1	T4	T3	T2	T1
0	0	0	0	0	1	0	0	1	1	0	0	1
0	0	0	0	1	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	0	0	1	0	0	1	1
0	0	1	0	1	0	0	1	1	0	0	0	1
0	0	1	1	0	0	0	1	0	0	0	0	1
0	0	1	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	0	0	1	1	0	1	1	1
0	1	0	0	1	0	1	0	1	0	0	0	1
0	1	0	1	0	0	1	0	0	0	0	0	1
0	1	0	1	1	0	1	1	0	0	0	1	1
0	1	1	0	0	0	1	0	1	0	0	1	1
0	1	1	0	1	0	1	1	1	0	0	0	1
0	1	1	1	0	0	1	1	0	0	0	0	1
0	1	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	1	1	0	0	1	0	0	0	1
1	0	0	1	0	1	0	0	0	0	0	0	1
1	0	0	1	1	0	0	0	0	1	0	0	1

Step4: The minimal expression: since there are 4 state variables and a mode signal, we require 5 variable kmaps. 20 conditions of Q4Q3Q2Q1M are valid and the remaining 12 combinations are invalid. So the entries for excitations corresponding to those invalid combinations are don‘t cares. Minimizing K-maps for T2 we get

T 2= Q4Q1‘M+Q4‘Q1M+Q2Q1‘M‘+Q3Q1‘M‘

Step5: the logic diagram: the logic diagram based on the above equation is shown in fig.



Shift register counters:

One of the applications of shift register is that they can be arranged to form several types of counters. The most widely used shift register counter is ring counter as well as the twisted ring counter.

Ring counter: this is the simplest shift register counter. The basic ring counter using D flip-flops is shown in fig. the realization of this counter using JK FFs. The Q output of each stage is connected to the D flip-flop connected back to the ring counter.

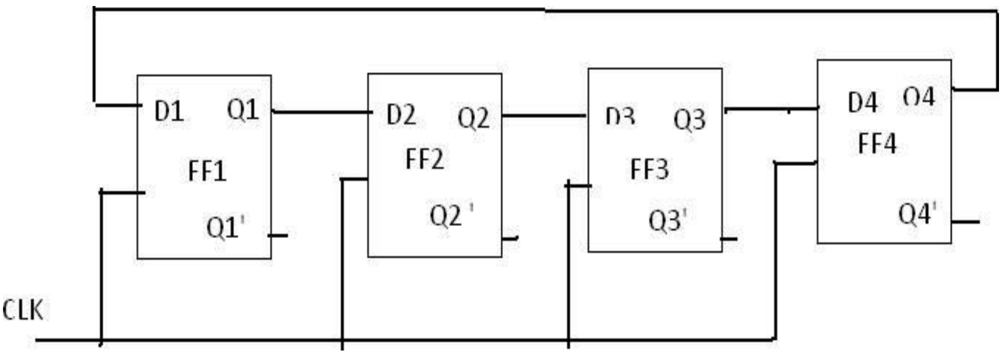
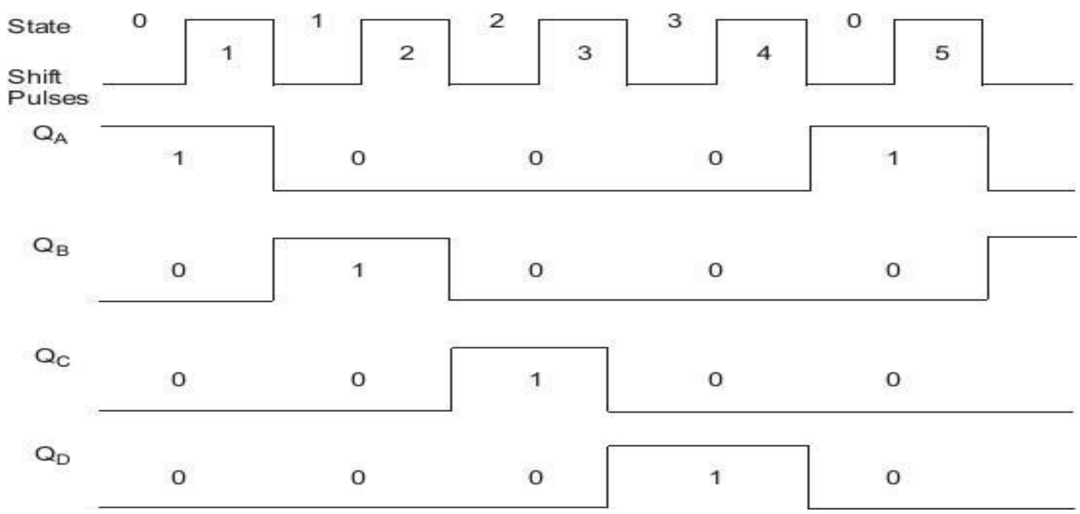


FIGURE: logic diagram of 4-bit ring counter using D flip-flops

Only a single 1 is in the register and is made to circulate around the register as long as clock pulses are applied. Initially the first FF is present to a 1. So, the initial state is 1000, i.e., Q1=1, Q2=0,Q3=0,Q4=0. After each clock pulse, the contents of the register are shifted to the right by one bit and Q4 is shifted back to Q1. The sequence repeats after four clock pulses. The number

of distinct states in the ring counter, i.e., the mod of the ring counter is equal to number of FFs used in the counter. An n-bit ring counter can count only n bits, where as n-bit ripple counter can count 2^n bits. So, the ring counter is uneconomical compared to a ripple counter but has advantage of requiring no decoder, since we can read the count by simply noting which FF is set. Since it is entirely a synchronous operation and requires no gates external FFs, it has the further advantage of being very fast.

Timing diagram:



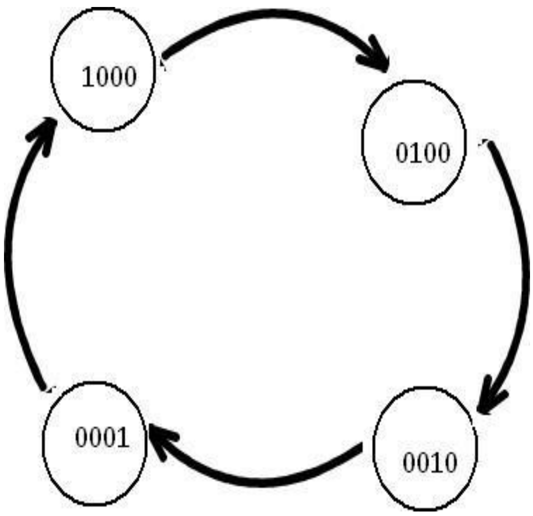


Figure: state diagram

Twisted Ring counter (Johnson counter):

This counter is obtained from a serial-in, serial-out shift register by providing feedback from the inverted output of the last FF to the D input of the first FF. the Q output of each is connected to the D input of the next stage, but the Q' output of the last stage is connected to the D input of the first stage, therefore, the name twisted ring counter. This feedback arrangement produces a unique sequence of states.

The logic diagram of a 4-bit Johnson counter using D FF is shown in fig. the realization of the same using J-K FFs is shown in fig.. The state diagram and the sequence table are shown in figure. The timing diagram of a Johnson counter is shown in figure.

Let initially all the FFs be reset, i.e., the state of the counter be 0000. After each clock pulse, the level of Q1 is shifted to Q2, the level of Q2to Q3, Q3 to Q4 and the level of Q4'to Q1 and the sequences given in fig.

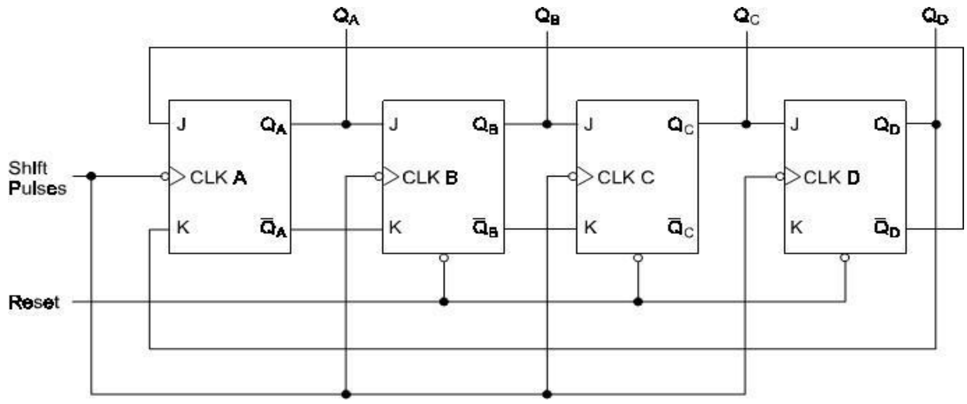


Figure: Johnson counter with JK flip-flops

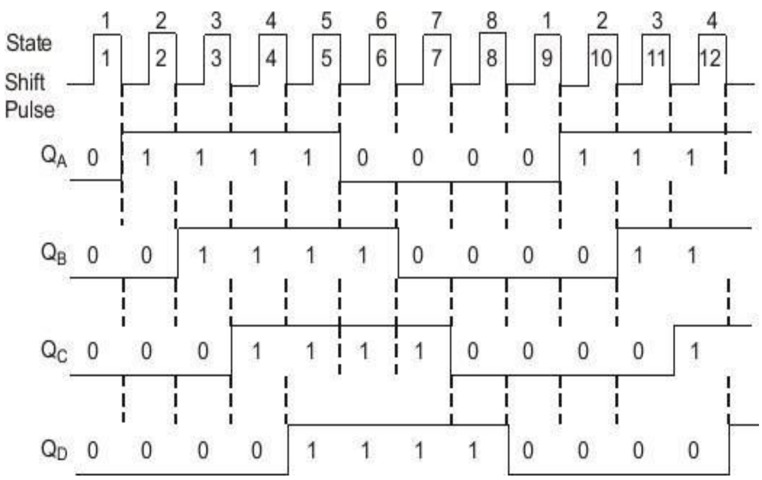
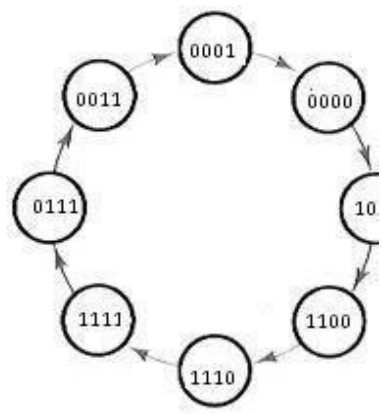


Figure: timing diagram

State diagram:



Q1	Q2	Q3	Q4	after clock pulse
0	0	0	0	0
1	0	0	0	1
1	1	0	0	2
1	1	1	0	3
1	1	1	1	4
0	1	1	1	5
0	0	1	1	6
0	0	0	1	7
0	0	0	0	8
1	0	0	0	9

Excitation table

Synthesis of sequential circuits:

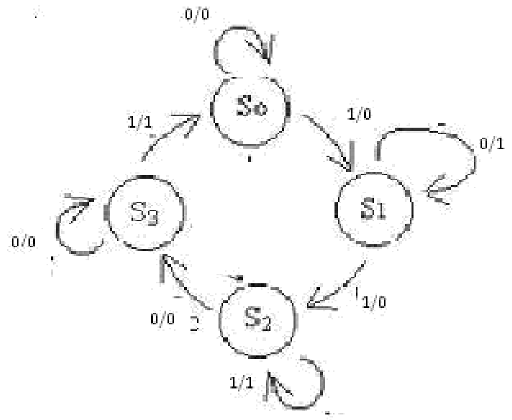
The synchronous or clocked sequential circuits are represented by two models.

1. **Moore circuit: in this model, the output depends only on the present state of the flip-flops**
2. **Meelay circuit: in this model, the output depends on both present state of the flip-flop. And the inputs.**

Sequential circuits are also called finite state machines (FSMs). This name is due to the fast that the functional behavior of these circuits can be represented using a finite number of states.

State diagram: the state diagram or state graph is a pictorial representation of the relationships between the present state, the input, the next state, and the output of a sequential circuit. The state diagram is a pictorial representation of the behavior of a sequential circuit.

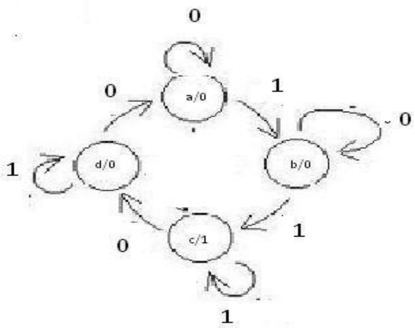
The state represented by a circle also called the node or vertex and the transition between states is indicated by directed lines connecting circle. a directed line connecting a circle with itself indicates that the next state is the same as the present state. The binary number inside each circle identifies the state represented by the circle. The direct lines are labeled with two binary numbers separated by a symbol. The input value is applied during the present state is labeled after the symbol.



PS	NS,O/P INPUT X X=0	X=1
a	a,0	b,0
b	b,1	c,0
c	d,0	c,1
d	d,0	a,1

Fig :a) state diagram (meelay circuit) **fig: b) state table**

In case of moore circuit ,the directed lines are labeled with only one binary number representing the input that causes the state transition. The output is indicated with in the circle below the present state, because the output depends only on the present state and not on the input.



PS	NS INPUT X X=0	X=1	O/P
a	a	b	0
b	b	c	0
c	d	c	1
d	a	d	0

Fig: a) state diagram (moore circuit) **fig:b) state table**

Serial binary adder:

Step1: word statement of the problem: the block diagram of a serial binary adder is shown in fig. it is a synchronous circuit with two input terminals designated X1and X2 which carry the two binary numbers to be added and one output terminal Z which represents the sum. The inputs and outputs consist of fixed-length sequences 0s and 1s.the output of the serial Z_i at time $t_i$is a function of the inputs $X_1(t_i)$ and $X_2(t_i)$ at that time t_{i-1} and of carry which had been generated at t_{i-1}. The carry which represent the past history of the serial adder may be a 0 or 1. The circuit has two states. If one state indicates that carry from the previous addition is a 0, the other state indicates that the carry from the previous addition is a 1

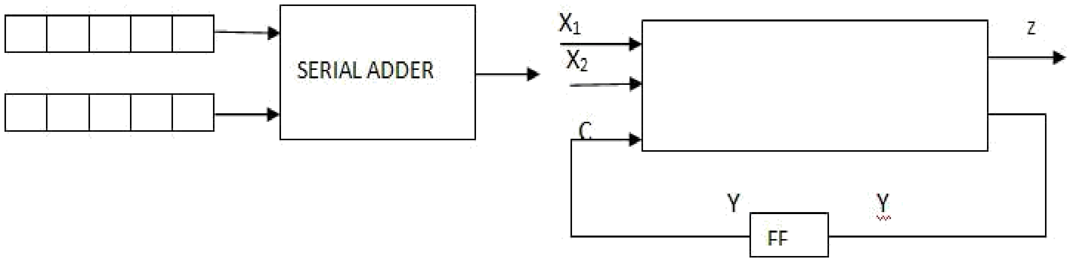
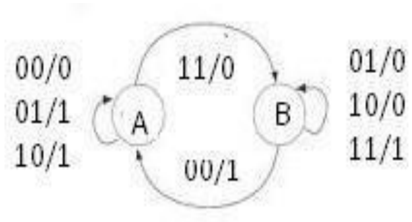


Figure: block diagram of serial binary adder

Step2 and 3: state diagram and state table: let a designate the state of the serial adder at t_i if a carry 0 was generated at t_{i-1}, and let b designate the state of the serial adder at t_i if carry 1 was generated at t_{i-1} .the state of the adder at that time when the present inputs are applied is referred to as the present state(PS) and the state to which the adder goes as a result of the new carry value is referred to as next state(NS).
The behavior of serial adder may be described by the state diagram and state table.



PS	NS ,O/P X1 X2			
	0 0	0 1	1 0	1 1
A	A,0	B,0	B,1	B,0
B	A,1	B,0	B,0	B,1

Figures: serial adder state diagram and state table

If the machine is in state B, i.e., carry from the previous addition is a 1, inputs X_1=0 and X_2=1 gives sum, 0 and carry 1. So the machine remains in state B and outputs a 0. Inputs X_1=1 and X_2=0 gives sum, 0 and carry 1. So the machine remains in state B and outputs a 0. Inputs X_1=1 and X_2=1 gives sum, 1 and carry 0. So the machine remains in state B and outputs a 1. Inputs X_1=0 and X_2=0 gives sum, 1 and carry 0. So the machine goes to state A and outputs a 1. The state table also gives the same information.

Setp4: reduced standard from state table: the machine is already in this form. So no need to do anything

Step5: state assignment and transition and output table:
The states, A=0 and B=1 have already been assigned. So, the transition and output table is as shown.

PS	NS				O/P			
	00	01	10	11	00	01	10	11
0	0	0	0	1	0	1	1	1
1	0	1	1	1	1	0	0	1

STEP6: choose type of FF and excitation table: to write table, select the memory element the excitation table is as shown in fig.

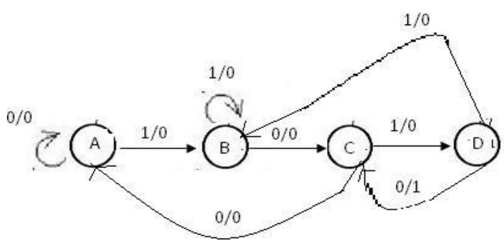
PS	I/P		NS	I/P-FF	O/P
y	x1	x2	Y	D	Z
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	1	1

Sequence detector:

Step1: word statement of the problem: a sequence detector is a sequential machine which produces an output 1 every time the desired sequence is detected and an output 0 at all other times

Suppose we want to design a sequence detector to detect the sequence 1010 and say that overlapping is permitted i.e., for example, if the input sequence is 01101010 the corresponding output sequence is 00000101.

Step2 and 3: state diagram and state table: the state diagram and the state table of the sequence detector. At the time t_1, the machine is assumed to be in the initial state designed arbitrarily as A. while in this state, the machine can receive first bit input, either a 0 o r a 1. If the input bit is 0, the machine does not start the detection process because the first bit in the desired sequence is a 1. If the input bit is a 1 the detection process starts.



PS	NS,Z	
	X=0	X=1
A	A,0	B,0
B	C,0	B,0
C	A,0	D,0
D	C,1	B,0

Figure: state diagram and state table of sequence detector

So, the machine goes to state B and outputs a 0. While in state B, the machinery may receive 0 or 1 bit. If the bit is 0, the machine goes to the next state, say state c, because the previous two bits are 10 which are a part of the valid sequence, and outputs 0.. if the bit is a 1, the two bits become 11 and this not a part of the valid sequence

Step4: reduced standard form state table: the machine is already in this form. So no need to do anything.

Step5: state assignment and transition and output table: there are four states therefore two states variables are required. Two state variables can have a maximum of four states, so, all states are utilized and thus there are no invalid states. Hence, there are no don‘t cares. Let a=00, B=01, C=10 and D=11 be the state assignment.

	NS(Y1Y2)		O/P(z)	
PS(y1y2	X=0	X=1	X=0	X=1
A=0 0	0	0 0	1 0	0
B=0 1	1	0 0	1 0	0
C=1 0	0	0 1	1 0	0
D=1 1 1	1 0	1 1		0

Step6: choose type of flip-flops and form the excitation table: select the D flip-flops as memory elements and draw the excitation table.

PS y1	Y2	I/P X	NS Y1	Y2	INPUTS - FFS D1	D2	O/P Z
0	0	0	0	0 0	0	0	
0	0	1	0	1 0	1	0	
0	1	0	1	0 1	0	0	
0	1	1	0	1 0	1	0	
1	0	0	0	0 0	0	0	
1	0	1	1	1 1	1	0	
1	1	0	1	0 1	0	1	
1	1	1	0	1 0	1	0	

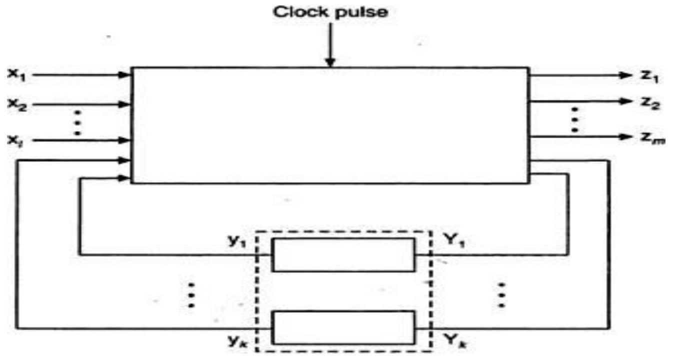
Step7: K-maps and minimal functions: based on the contents of the excitation table , draw the k-map and simplify them to obtain the minimal expressions for D1 and D2 in terms of y1, y2 and x as shown in fig. The expression for z (z=y1,y2) can be obtained directly from table

Step8: implementation: the logic diagram based on these minimal expressions

Finite State Machine:

Finite state machine can be defined as a type of machine whose past histories can affect its future behavior in a finite number of ways. To clarify, consider for example of binary full adder. Its output depends on the present input and the carry generated from the previous input. It may have a large number of previous input histories but they can be divided into two types: (i) Input

The most general model of a sequential circuit has inputs, outputs and internal states. A sequential circuit is referred to as a finite state machine (FSM). A finite state machine is abstract model that describes the synchronous sequential machine. The fig. shows the block diagram of a finite state model. X_1, X2,….., X_l, are inputs. Z_1, Z2,….,Z_m are outputs. Y_1,Y_2,….Y_k are state variables, and Y_1,Y_2,….Y_k represent the next state.



Capabilities and limitations of finite-state machine

Let a finite state machine have n states. Let a long sequence of input be given to the machine. The machine will progress starting from its beginning state to the next states according to the state transitions. However, after some time the input string may be longer than n, the number of states. As there are only n states in the machine, it must come to a state it was previously been in and from this phase if the input remains the same the machine will function in a periodically repeating fashion. From here a conclusion that ‗for a n state machine the output will become periodic after a number of clock pulses less than equal to n can be drawn. States are memory elements. As for a finite state machine the number of states is finite, so finite number of memory elements are required to design a finite state machine.

Limitations:

1. **Periodic sequence and limitations of finite states: with n-state machines, we can generate periodic sequences of n states are smaller than n states. For example, in a 6-state machine, we can have a maximum periodic sequence as 0,1,2,3,4,5,0,1….**

2. **No infinite sequence: consider an infinite sequence such that the output is 1 when and only when the number of inputs received so far is equal to P(P+1)/2 for P=1,2,3….,i.e., the desired input-output sequence has the following form:**

Input: x x x x x x x x x x x x xx x x xx x xx x
Output: 1 0 1 0 0 1 0 0 0 01 0 0 0 0 1 0 0 00 01

Such an infinite sequence cannot be produced by a finite state machine.

3. **Limited memory: the finite state machine has a limited memory and due to limited memory it cannot produce certain outputs. Consider a binary multiplier circuit for multiplying two arbitrarily large binary numbers. The memory is not sufficient to store arbitrarily large partial products resulted duringmultiplication.**

Finite state machines are two types. They differ in the way the output is generate they are:

1. **Mealy type model: in this model, the output is a function of the present state and the present input.**
2. **Moore type model: in this model, the output is a function of the present state only.**

Mathematical representation of synchronous sequential machine:

The relation between the present state S(t), present input X(t), and next state s(t+1) can be given as

$S(t+1)= f\{S(t),X(t)\}$

The value of output Z(t) can be given as

$Z(t)= g\{S(t),X(t)\}$ for mealy model

$Z(t)= G\{S(t)\}$ for Moore model

Because, in a mealy machine, the output depends on the present state and input, where as in a Moore machine, the output depends only on the present state.

Comparison between the Moore machine and mealy machine:

Moore machine	mealy machine
1. its output is a function of present state only $Z(t)= g\{S(t)\}$	1. its output is a function of present state as well as present input $Z(t)=g\{S(t),X(t)\}$
2. input changes do not affect the output	2. input changes may affect the output of the circuit
3. it requires more number of states for implementing same function	3. it requires less number of states for implementing same function

Mealy model:

When the output of the sequential circuit depends on the both the present state of the flip-flops and on the inputs, the sequential circuit is referred to as mealy circuit or mealy machine.

The fig. shows the logic diagram of the mealy model. Notice that the output depends up on the present state as well as the present inputs. We can easily realize that changes in the input during the clock pulse cannot affect the state of the flip-flop. They can affect the output of the circuit. If the input variations are not synchronized with a clock, he derived output will also not be synchronized with the clock and we get false output. The false outputs can be eliminated by allowing input to change only at the active transition of the clock.

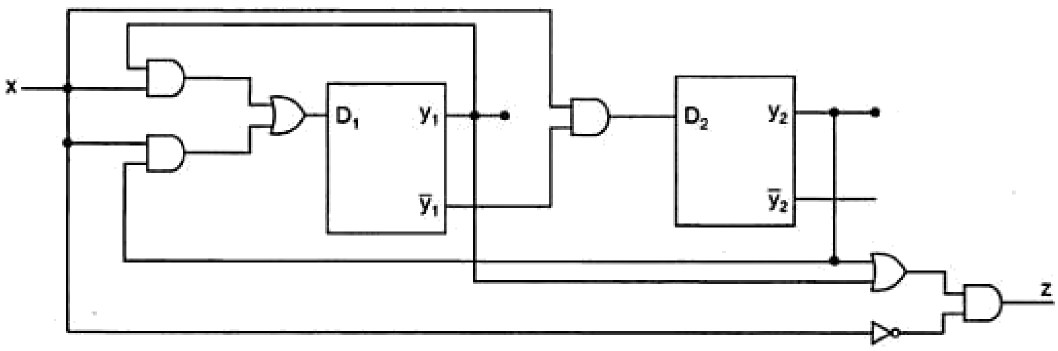


Fig: logic diagram of a mealy model

The behavior of a clocked sequential circuit can be described algebraically by means of state equations. A state equation specifies the next state as a function of the present state and inputs. The mealy model shown in fig. consists of two D flip-flops, an input x and an output z. since the D input of a flip-flop determines the value of the next state, the state equations for the model can be written as

Y_1 (t+1)=y_1(t)x(t)+y_2(t)x(t)

Y_2(t+1)= 1(t)x(t)

And the output equation is

Z(t)={ y_1(t)+y_2(t)} X'(t)

Where y(t+1) is the next state of the flip-flop one clock edge later, x(t) is the present input, and z(t) is the present output. If y1(t+1) are represented by y1(t) and y2(t) , in more compact form, the equations are

Y1(t+1)=y1=y1x+y2x

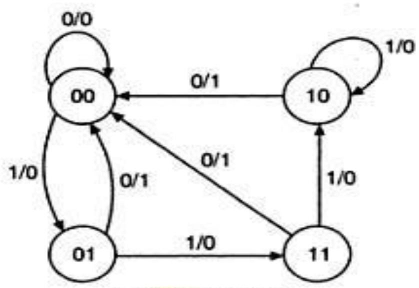
Y2(t+1)=y2=y1'x

Z=(y1+y2)x'

The stable table of the mealy model based on the above state equations and output equation is shown in fig. the state diagram based on the state table is shown in fig.

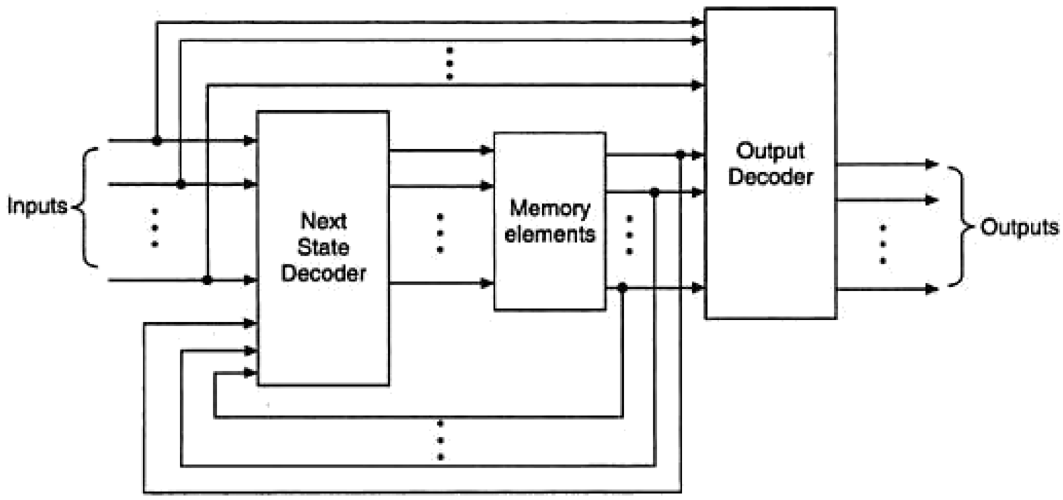
PS		NS				O/P	
		x = 0		x = 1		x = 0	x = 1
y_1	y_2	Y_1	Y_2	Y_1	Y_2	z	z
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

(a) State table



(b) State diagram

In general form, the mealy circuit can be represented with its block schematic as shown in below fig.



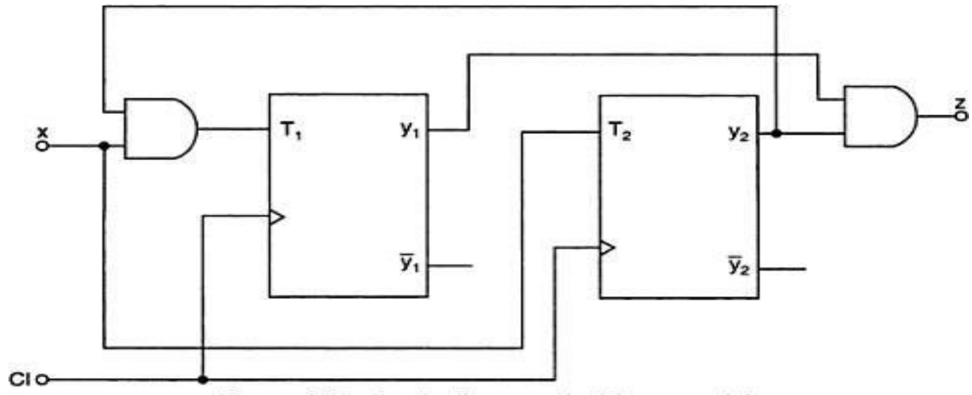
Moore model: when the output of the sequential circuit depends up only on the present state of the flip-flop, the sequential circuit is referred as to as the Moore circuit or the Moore machine.

Notice that the output depend only on the present state. It does not depend upon the input at all. The input is used only to determine the inputs of flip-flops. It is not used to determine the output. The circuit shown has two T flip-flops, one input x, and one output z. it can be described algebraically by two input equations an output equation.

$T_1=y_2x$

$T_2=x$

$Z=y_1y_2$



n of a T-flip-flop is $Q(t+1)=TQ'+T'Q$

The values for the next state can be derived from the state equations by substituting T_1 and T_2 in the characteristic equation yielding

$Y_1(t+1)=Y_1=(y_2x) \oplus =(2)y1+(y2x)1$

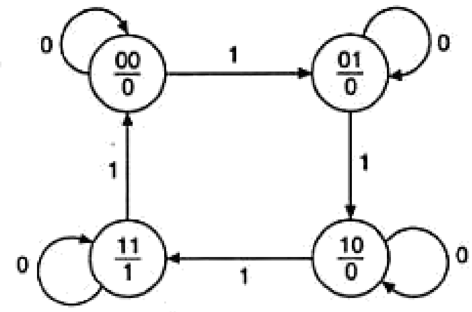
$= y1\ 2+ y1 + 1y2x$

$= y2\ (t+1)= x \oplus y2= x\ 2+ y2$

The state table of the Moore model based on the above state equations and output equation is shown in fig.

PS		NS				O/P
		x = 0		x = 1		
y_1	y_2	Y_1	Y_2	Y_1	Y_2	z
0	0	0	0	0	1	0
0	1	0	1	1	0	0
1	0	1	0	1	1	0
1	1	1	1	0	0	1

(a) State table



(b) State diagram

In general form , the Moore circuit can be represented with its block schematic as shown in below fig.

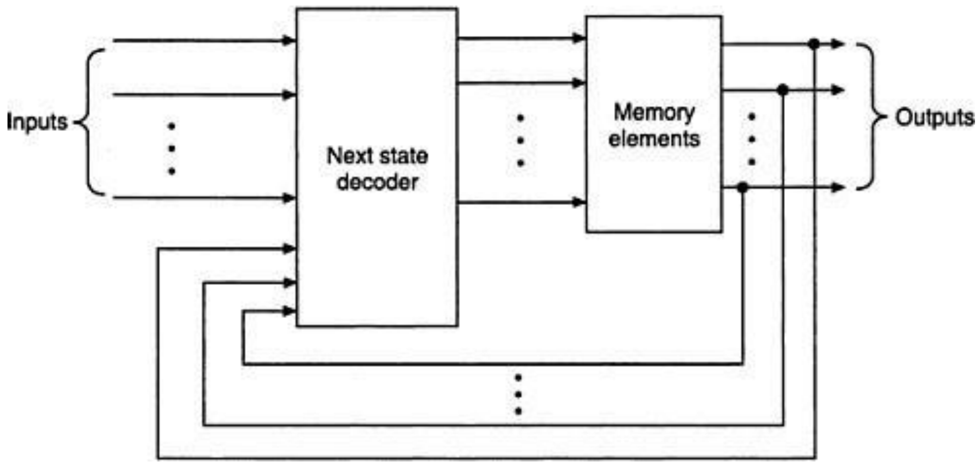


Figure: moore circuit model:

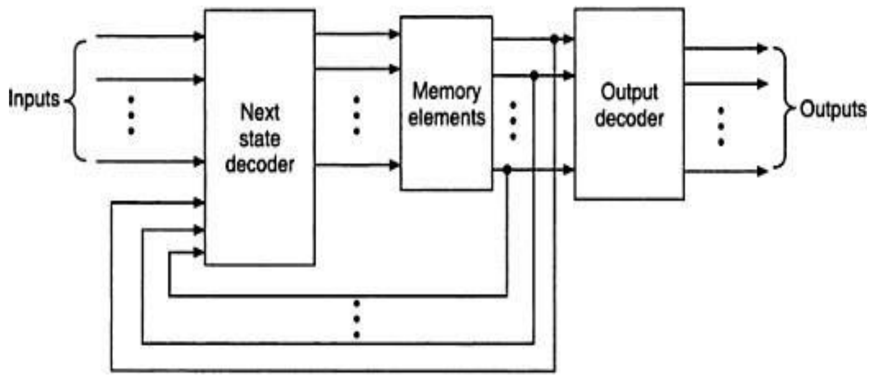
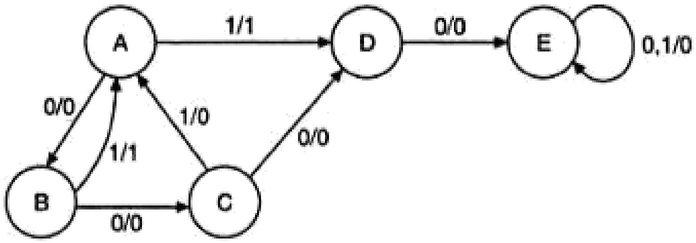


Figure: moore circuit model with an output decoder

Important definitions and theorems:

A). Finite state machine-definitions:

Consider the state diagram of a finite state machine shown in fig. it is five-state machine with one input variable and one output variable.



Successor: looking at the state diagram when present state is A and input is 1, the next state is D. this condition is specified as D is the successor of A. similarly we can say that A is the 1 successor of B, and C,D is the 11 successor of B and C, C is the 00 successor of A and D, D is the 000 successor of A,E, is the 10 successor of A or 0000 successor of A and so on.

Terminal state: looking at the state diagram , we observe that no such input sequence exists which can take the sequential machine out of state E and thus state E is said to be a terminal state.

Strongly-connected machine: in sequential machines many times certain subsets of states may not be reachable from other subsets of states. Even if the machine does not contain any terminal state. If for every pair of states s_i, s_j, of a sequential machine there exists an input sequence which takes the machine M from s_i to sj, then the sequential machine is said to be strongly connected.

B). state equivalence and machine minimization:

In realizing the logic diagram from a stat table or state diagram many times we come across redundant states. Redundant states are states whose functions can be accomplished by other states. The elimination of redundant states reduces the total number of states of the machines which in turn results in reduction of the number of flip-flops and logic gates, reducing the cost of the final circuit.

Two states are said to be equivalent. When two states are equivalent, one of them can be removed without altering the input output relationship.

State equivalence theorem: it states that two states s_1, and s_2 are equivalent if for every possible input sequence applied. The machine goes to the same next state and generates the same output. That is

If $S_1(t+1)= s_2(t+1)$ and $z_1=z_2$, then $s_1=s_2$

C). distinguishable states and distinguishing sequences:

Two states s_a, and s_b of a sequential machine are distinguishable, if and only if there exists at least one finite input sequence which when applied to the sequential machine causes different outputs sequences depending on weather s_a or s_b is the initial state.

Consider states A and B in the state table, when input X=0, their outputs are 0 and 1 respectively and therefore, states A and B are called 1-distinguishabke. Now consider states A and E . the output sequence is as follows.

X=0 A C,0 and E D, 0 ; outputs are the same

C → E,0 and D→ b,1 ; outputs are different

Here the outputs are different after 2-state transition and hence states A and E are 2-distungishable. Again consider states A and C . the output sequence is as follows:

X=0 A→ C,0 and C → E, 0; outputs are the same

C → E,0 and E → D,0 ; outputs are the

same E → D,0 – and D B,1 ; outputs are

different

Here the outputs are different after 3- transition and hence states A and B are 3-distuingshable. the concept of K- distuingshable leads directly to the definition of K-equivalence. States that are not K-distinguishable are said to be K-equivalent.

Truth table for Distunigshable states:

PS	NS,Z	
	X=0	X=1
A	C,0	F,0
B	D,1	F,0
C	E,0	B,0
D	B,1	E,0
E	D,0	B,0
F	D,1	B,0

Merger Chart Methods:

Merger graphs:

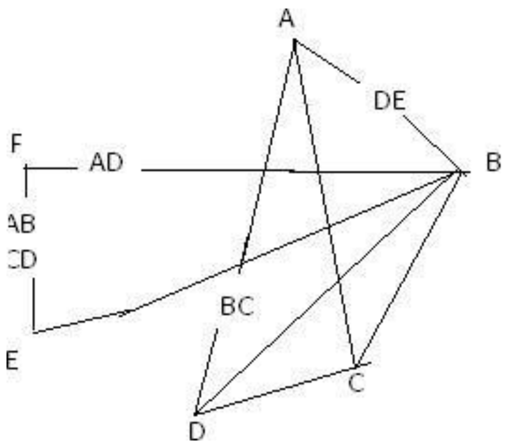
The merger graph is a state reducing tool used to reduce states in the incompletely specified machine. The merger graph is defined as follows.

1. **Each state in the state table is represented by a vertex in the merger graph. So it contains the same number of vertices as the state table contains states.**
2. **Each compatible state pair is indicated by an unbroken line draw between the two state vertices**
3. **Every potentially compatible state pair with non-conflicting outputs but with different next states is connected by a broken line. The implied states are written in theline break between the two potentially compatible states.**
4. **If two states are incompatible no connecting line is drawn.**

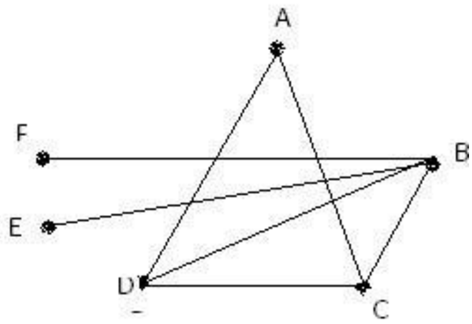
Consider a state table of an incompletely specified machine shown in fig. the corresponding merger graph shown in fig.

State table:

PS	NS,Z			
	I1	I2	I3	I4
A	…	E,1	B,1	….
B	…	D,1	…	F,1
C	F,1	…	…	…
D	…	…	C,1	…
E	C,0	…	A,0	F,1
F	D,0	A,1	B,0	…



a) Merger graph



b) simplified merger graph

States A and B have non-conflicting outputs, but the successor under input $I_2$are compatible only if implied states D and E are compatible. So, draw a broken line from A to B with DE written in between states A and C are compatible because the next states and output entries of states A and C are not conflicting. Therefore, a line is drawn between nodes A and C. states A and D have non-conflicting outputs but the successor under input I_3 are B and C. hence join A and D by a broken line with BC entered In between.

Two states are said to be incompatible if no line is drawn between them. If implied states are incompatible, they are crossed and the corresponding line is ignored. Like, implied states D and E are incompatible, so states A and B are also incompatible. Next, it is necessary to check whether the incompatibility of A and B does not invalidate any other broken line. Observe that states E and F also become incompatible because the implied pair AB is incompatible. The broken lines which remain in the graph after all the implied pairs have been verified to be compatible are regarded as complete lines. After checking all possibilities of incompatibility, the merger graph gives the following seven compatible pairs.

(A, C) (A, D) (B, C) (B, D) (C, D) (B, E) (B, F)

These compatible pairs are further checked for further compatibility. For example, pairs (B,C)(B,D)(C,D) are compatible. So (B, C, D) is also compatible. Also pairs (A,c)(A,D)(C,D) are compatible. So (A,C,D) is also compatible. . In this way the entire set of compatibles of sequential machine can be generated from its compatible pairs.

To find the minimal set of compatibles for state reduction, it is useful to find what are called the maximal compatibles. A set of compatibles state pairs is said to be maximal, if it is not completely covered by any other set of compatible state pairs. The maximum compatible can be found by looking at the merger graph for polygons which are not contained within any higher order complete polygons. For example only triangles (A, C,D) and (B,C,D) are of higher order. The set of maximal compatibles for this sequential machine given as

(A, C, D) (B, C, D) (B, E) (B, F)

Example:

Draw the merger graph and obtain the set of maximal compatibles for the incompletely specified sequential machine whose state table is given in Table 7.24.

Table 7.24 Example 7.9: State table

PS	NS, Z	
	I_1	I_2
A	E, 0	B, 0
B	F, 0	A, 0
C	E, –	C, 0
D	F, 1	D, 0
E	C, 1	C, 0
F	D, –	B, 0

mark × in the corresponding cell. For example, states B and C are incompatible because their outputs are conflicting and hence the cell corresponding to them contains a cross mark ×. Similarly states B, E; D, E; E, F are incompatible. Hence put a × mark in the corresponding cells. On the other hand, states A and B are compatible and hence the cell corresponding to them contains the check mark ✓. Similarly, cells corresponding to states A, D; A ,E; A, G; B, G; C, F; D, F ; D, G are also compatible. So a check mark is put in those cells also. The implied pairs or pairs corresponding to the state pair are written within the cell as shown in Table 7.26. For example, states A and C are compatible only when implied states E and F are compatible. Therefore, EF is written in the cell corresponding to states A and C. States C and E are compatible only when implied states A and B, and D and F are compatible. So AB and DF are written in the cell corresponding to states C and E. In a similar way, the entire merger table is written. Now it is necessary to check whether the implied pairs are compatible or not by observing the merger table. The implied states are incompatible if the corresponding cell contains a ×. For example, implied pair E, F is incompatible because cell EF contains a ×. Similarly, implied pairs EF, AF are incompatible because EF contains a ×. It is indicated by a ×.

PS	NS, Z			
	00	01	11	10
A	E, 0	–	–	–
B	–	F, 1	E, 1	A, 1
C	F, 0	–	A, 0	F, 1
D	–	–	A, 1	–
E	–	C, 0	B, 0	D, 1
F	C, 0	C, 1	–	–
G	E, 0	–	–	A, 1

Figure: state table

B	✓					
C	~~DF~~	×				
D	✓	AE	×			
E	✓	×	AB DF	×		
F	CE	CF	✓	✓	×	
G	✓	✓	~~EF AF~~	✓	AD	CE

State Minimization:
Completely Specified Machines

Two states, s_i and s_j of machine M are distinguishable if and only if there exists a finite input sequence which when applied to M causes different output sequences depending on whether M started in s_i or s_j.
Such a sequence is called a distinguishing sequence for (s_i, s_j).
If there exists a distinguishing sequence of length k for (s_i, s_j), they are said to be k-distinguishable.

EXAMPLE:

PS	NS, z	
	x=0	x=1
A	E, 0	D, 1
B	F, 0	D, 0
C	E, 0	B, 1
D	F, 0	B, 0
E	C, 0	F, 1
F	B, 0	C, 0

- **states A and B are 1-distinguishable, since a 1 input applied to A yields an output 1, versus an output 0 from B.**
- **states A and E are 3-distinguishable, since input sequence 111 applied to A yields output 100, versus an output 101 from E.**
- **States s_i and s_j ($s_i \sim s_j$) are said to be equivalent iff no distinguishing sequence exists for (s_i, s_j).**
- **If $s_i \sim s_j$ and $s_j \sim s_k$, then $s_i \sim s_k$. So state equivalence is an equivalence relation (i.e. it is a reflexive, symmetric and transitive relation).**
- **An equivalence relation partitions the elements of a set into equivalence classes.**
- **Property: If $s_i \sim s_j$, their corresponding X-successors, for all inputs X, are also equivalent.**
- **Procedure: Group states of M so that two states are in the same group iff they are equivalent (forms a partition of the states).**

Completely Specified Machines

PS	NS, z x=0	NS, z x=1
A	E, 0	D, 1
B	F, 0	D, 0
C	E, 0	B, 1
D	F, 0	B, 0
E	C, 0	F, 1
F	B, 0	C, 0

P_i : partition using distinguishing sequences of length i.

Partition:	Distinguishing Sequence:
P_0=(ABCDEF)	
P_1= (A C E)(BD F)	x =1
P_2= (A C E)(B D)(F)	x =1; x =1
P_3= (A C)(E)(B D)(F)	x =1; x =1; x =1

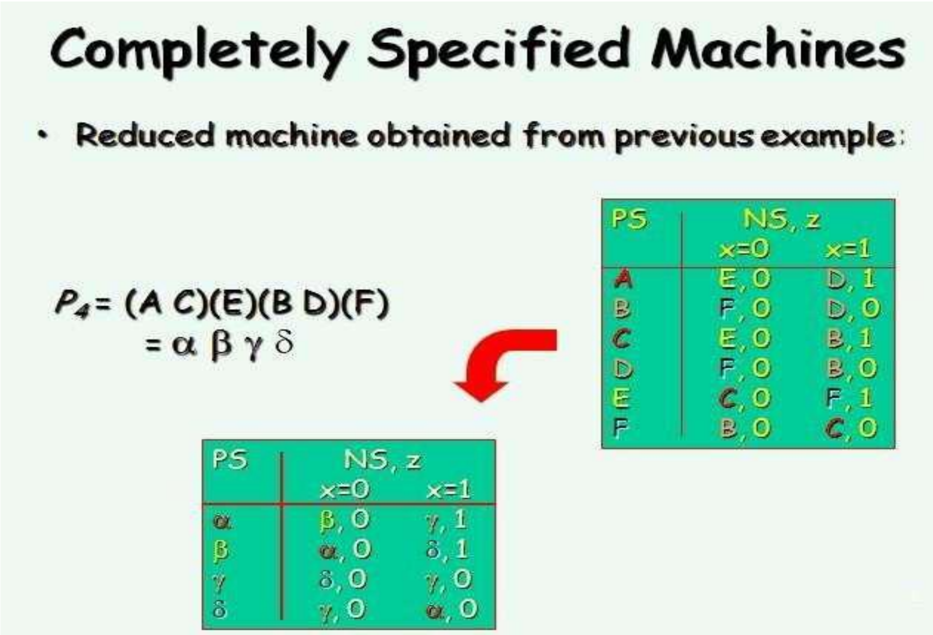
- **All states equivalent to each other form an equivalence class. These may becombined into one state in the reduced (quotient) machine.**
- **Start an initial partition of a single block. Iteratively refine this partition by separating the 1-distinguishable states, 2-distinguishable states and so on.**
- **To obtain P_{k+1}, for each block B_i of P_k, create one block of states that not 1-distinguishable within B_i , and create different blocks states that are 1-distinguishable within B_i .**

Theorem: The equivalence partition is unique.

Theorem: If two states, s_i and s_j, of machine M are distinguishable, then they are (n-1)-distinguishable, where n is the number of states in M.

Definition: Two machines, M_1 and M_2, are equivalent ($M_1 \sim M_2$) if, for every state in M_1 there is a corresponding equivalent state in M_2 and vice versa.

Theorem. For every machine M there is a minimum machine M_{red} ~ M. M_{red} is unique up to isomorphism.



State Minimization of CSMs: Complexity
Algorithm DFA ~ DFA_{min}
Input: A finite automaton M = (Q, , , q_0, F) with no unreachable states.
Output: A minimum finite automaton M' = (Q', , ', q'_0, F').

Method:

1. **t :=2; Q_0:= { undefined }; Q_1:=F; Q_2:= Q\F.**
2. **while there is 0 < i t, awith (Q_i,a) Q_j, for all j t**

do (a) Choose such an i, a , and jt with (Q_i,a) Q_j .

(b) Q_{t+1} := {q Q_i | (q,a) Q_j }; Q_i
:= Q_i \ Q_{t+1};

t := t +1.
end.

3. **(* Denote [q] the equivalence class of state q , and {Q_i } the set of all equivalence classes. *)**

q'_0 := [q_0].

' ([q], a) := [(q,a)] for all q Q, a.

Standard implementation: O (kn ^{2}), where n =|Q| and k = | |
Modification of the body of the while loop:

1. **Choose such an i, a, and choose j_1,j_2 t with j_1 j_2, (Q_i,a) Q_{j1}, and (Q_i,a) Q_{j2} .**
2. **If |{q Q_i | (q,a) Q_{j1}}| |{q Q_i | (q,a) Q_{j2}}|**

then Q_{t+1} := {q Q_i | (q,a) Q_{j1} else Q_{t+1} := {q }
Q_i | (q,a) Q_{j2} } fI;
Q_i := Q_i \ Q_{t+1};
t := t +1.
(i.e. put smallest set in t +1)
Note: |Q_{t+1}| 1/2|Q_i|. Therefore, for all q Q, the name of the class which contains a given state q changes at most log(n) times.

Goal: Develop an implementation such that all computations can be assigned to transitions containing a state for which the name of the corresponding class is changed.

Suitable data structures achieve an O (kn log n) implementation.

State Minimization:

Incompletely Specified Machines

Statement of the problem: given an incompletely specified machine M, find a machine M' such that:

– **on any input sequence, M' produces the same outputs as M, whenever M is specified.**
– **there does not exist a machine M'' with fewer states than M' which has the same property**

Machine M:

PS	NS, z x=0	x=1
s1	s3, 0	s2, 0
s2	s2, -	s3, 0
s3	s3, 1	s2, 0

Attempt to reduce this case to usual state minimization of completely specified machines.

Brute Force Method: Force the don't cares to all their possible values and choose the smallest of the completely specified machines soobtained.

In this example, it means to state minimize two completely specified machines obtained from M, by setting the don't care to either 0 and 1.

Suppose that the - is set to be a 0.

PS	NS, z x=0	x=1
s1	s3, 0	s2, 0
s2	s2, 0	s3, 0
s3	s3, 1	s2, 0

States s1 and s2 are equivalent if s3 and s2 are equivalent, but s3 and s2 assert different outputs under input 0, so s1 and s2 are not equivalent.

States s1 and s3 are not equivalent either.

So this completely specified machine cannot be reduced further (3 states is the minimum).

Suppose that the - is set to be a 1.

PS	NS, z	
	x=0	x=1
s1	s3, 0	s2, 0
s2	s2, 1	s3, 0
s3	s3, 1	s2, 0

States s1 is incompatible with both s2 and s3.
States s3 and s2 are equivalent.
So number of states is reduced from 3 to 2.

Machine M''$_{red}$:

PS	NS, z	
	x=0	x=1
A	A, 1	A, 0
B	B, 0	A, 0

Can this always be done?
Machine M:

PS	NS, z	
	x=0	x=1
s1	s3, 0	s2, 0
s2	s2, -	s1, 0
s3	s1, 1	s2, 0

Machine M_2:

PS	NS, z	
	x=0	x=1
s1	s3, 0	s2, 0
s2	s2, 0	s1, 0
s3	s1, 1	s2, 0

Machine M_3:

PS	NS, z	
	x=0	x=1
s1	s3, 0	s2, 0
s2	s2, 1	s1, 0
s3	s1, 1	s2, 0

Machine M_2 and M_3 are formed by filling in the unspecified entry in M with 0 and 1, respectively.

Both machines M_2 and M_3 cannot be reduced.
Conclusion?: M cannot be minimized further!
But is it a correct conclusion?
Note: that we want to ‗merge' two states when, for any input sequence, they generate the same output sequence, but only where both outputs are specified.
Definition: A set of states is compatible if they agree on the outputs where they are all specified.

Machine M'' :

PS	NS, z	
	x=0	x=1
s1	s3, 0	s2, 0
s2	s2, -	s1, 0
s3	s1, 1	s2, 0

In this case we have two compatible sets: A = (s1, s2) and B = (s3, s2). A reduced machine M_{red} can be built as follows.

Machine M_{red}

PS	NS, z	
	x=0	x=1
A	A, 1	A, 0
B	B, 0	A, 0

PS	NS, z			
	I1	I2	I3	I4
s1	s3,0	s1,-	-	-
s2	s6,-	s2,0	s1,-	-
s3	-,1	-,-	s4,0	-
s4	s1,0	-,-	-	s5,1
s5	-,-	s5,-	s2,1	s1,1
s6	-,-	s2,1	s6,-	s4,1

A set of compatibles that cover all states is: (s3s6), (s4s6), (s1s6), (s4s5), (s2s5).
But (s3s6) requires (s4s6),
(s4s6) requires(s4s5), **(s4s5) requires (s1s5),**
(s1s6) requires (s1s2), **(s1s2) requires (s3s6),**
(s2s5) requires (s1s2).
So, this selection of compatibles requires too many other compatibles...

PS	NS, z			
	I1	I2	I3	I4
s1	s3,0	s1,-	-	-
s2	s6,-	s2,0	s1,-	-
s3	-,1	-,-	s4,0	-
s4	s1,0	-,-	-	s5,1
s5	-,-	s5,-	s2,1	s1,1
s6	-,-	s2,1	s6,-	s4,1

Another set of compatibles that covers all states is (s1s2s5), (s3s6), (s4s5). But (s1s2s5) requires (s3s6) (s3s6) requires (s4s6)
(s4s6) requires (s4s5) **(s4s5) requires (s1s5).**
So must select also (s4s6) and (s1s5).
Selection of minimum set is a binate covering problem

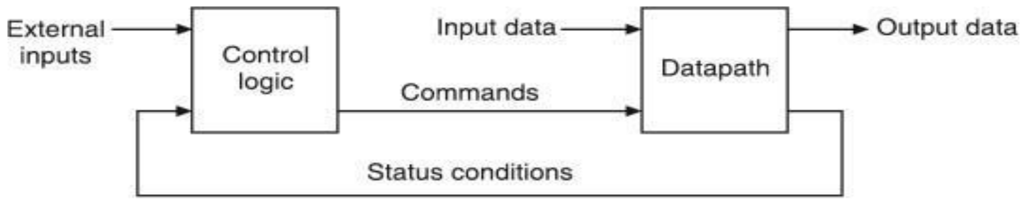
When a next state is unspecified, the future behavior of the machine is unpredictable. This suggests the definition of admissible input sequence.
Definition. An input sequence is admissible, for a starting state of a machine if no unspecified next state is encountered, except possibly at the final step.
Definition. State s_i of machine M_1 is said to cover, or contain, state s_j of M_2 provided

1. **every input sequence admissible to s_j is also admissible to s_i , and**
2. **its application to both M_1 and M_2 (initially is s_i and s_j, respectively) results in identical output sequences whenever the outputs of M_2 are specified.**

Definition. Machine M_1 is said to cover machine M_2 if for every state s_j in M_2, there is a corresponding state s_i in M_1 such that s_i covers s_j.

Algorithmic State Machines:

The binary information stored in the digital system can be classified as either data or control information.
The data information is manipulated by performing arithmetic, logic, shift and other data processing tasks.
The control information provides the command signals that controls the various operations on the data in order to accomplish the desired data processing task.
Design a digital system we have to design two subsystems data path subsystem and control subsystem.



Interaction between control logic and datapath.

ASM CHART:

A special flow chart that has been developed specifically to define digital hardware algorithms is called ASM chart.
A hardware algorithm is a step by step procedure to implement the desire task.

Difference b/n conventional flow chart and ASM chart:

conventional flow chart describes the sequence of procedural steps and decision paths for an algorithm without concern for their time relationship
An ASM chart describes the sequence of events as well as the timing relationship b/n the states of sequential controller and the events that occur while going from one state to the next

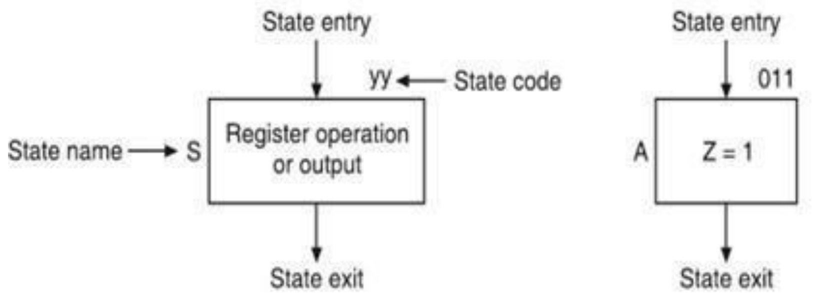
1. State box: A state of a clocked sequential circuit is represented by a rectangle called *state box*. It is equivalent to a node in the state diagram or a row in the state table. The name of the state is written to the left of the box. The binary code assigned to the state is indicated outside on the top right-side of the box. A list of unconditional outputs if any associated with the state are written within the box.

2. Decision box: The decision box or condition box is represented by a diamond-shaped symbol with one input and two or more output paths. The output branches are true and false branches. The decision box describes the effect of an input on the control subsystem. A Boolean variable or input or expression written inside the diamond indicates a condition which is evaluated to determine which branch to take.

ASM consists of

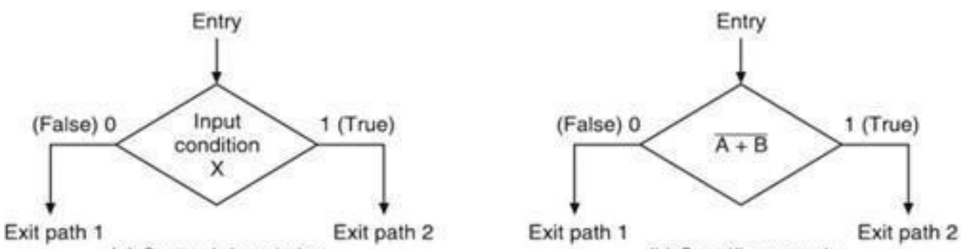
1. **State box**
2. **Decision box**
3. **Conditional box**

State box



(a) General description (b) Specific example

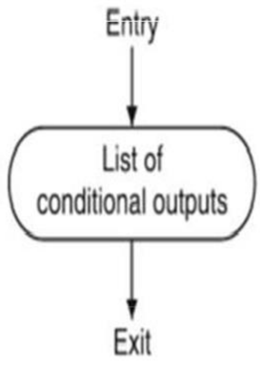
Decision box



(a) General description (b) Specific example

Decision box.

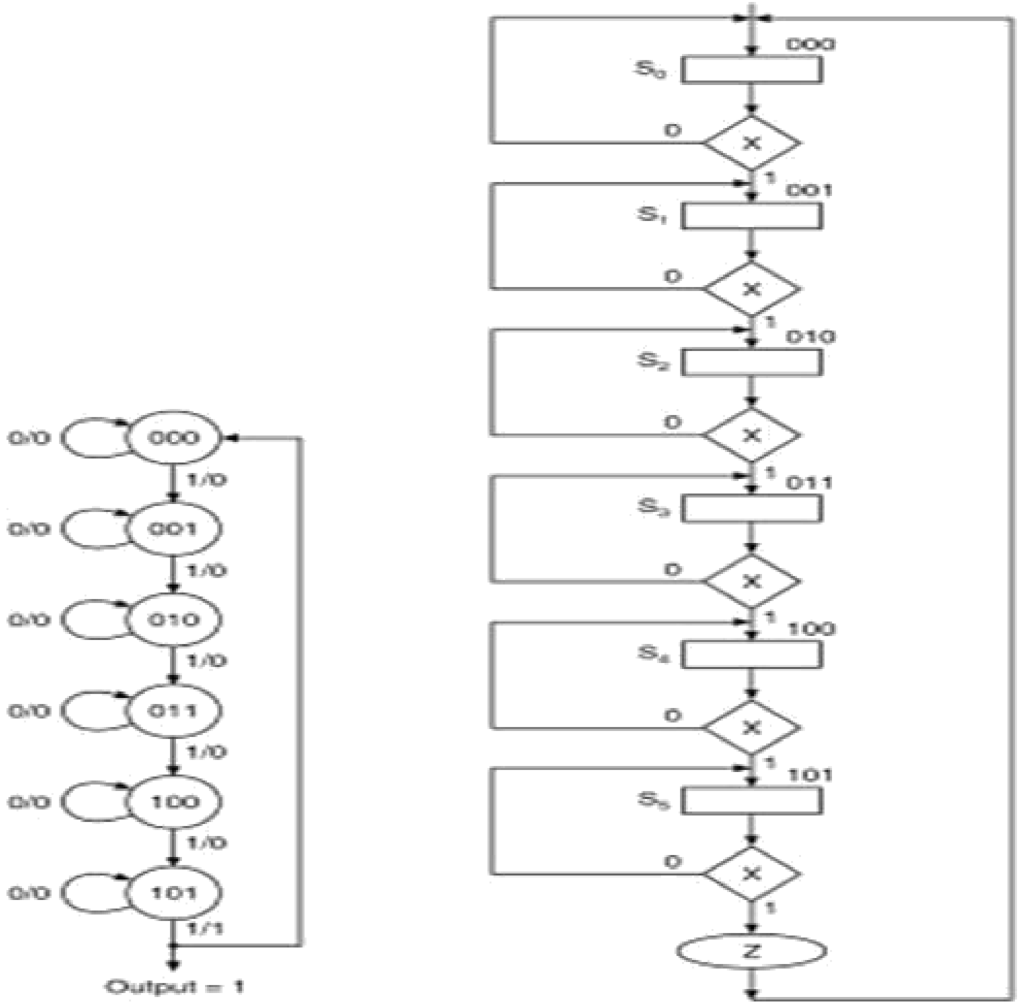
3. Conditional output box: The conditional output box is represented by a rectangle with rounded corners or by an oval with one input line and one output line. The outputs that depend on both the state of the system and the inputs are indicated inside the box.



Conditional output box.

SALIENT FEATURES OF ASM CHARTS

1. An ASM chart describes the sequence of events as well as the timing relationship between the states of a sequential controller and the events that occur while going from one state to the next.
2. An ASM chart contains one or more interconnected ASM blocks.
3. Each ASM block contains exactly one state box together with the decision boxes and conditional output boxes associated with that state.
4. Every block in an ASM chart specifies the operations that are to be performed during one common clock pulse.
5. An ASM block has exactly one entrance path and one or more exit paths represented by the structure of the decision boxes.
6. A path through an ASM block from entrance to exit is referred to as a link path.
7. The operations specified within the state and conditional output boxes in the block are performed in the datapath subsystem.
8. Internal feedback within an ASM block is not permitted. Even so, following a decision box or conditional output boxes, the machine may reenter the same state.
9. Each block in the ASM chart describes the state of the system during one clock pulse interval. When a digital system enters the state associated with a given ASM block, the outputs indicated within the state box become true. The conditions associated with the decision boxes are evaluated to determine which path or paths to be followed to enter the next ASM block.



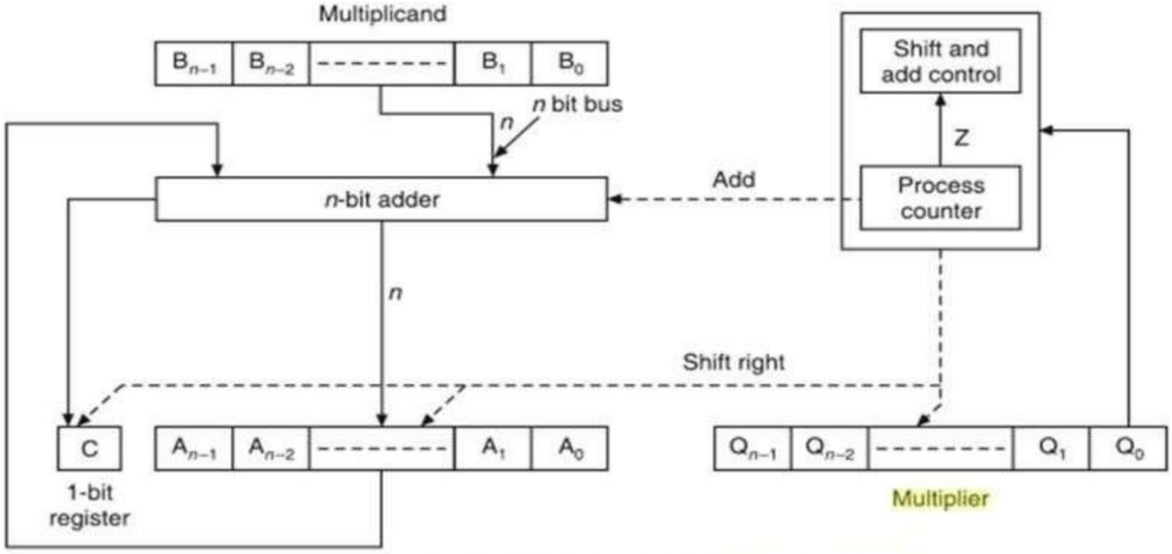
(a) State diagram (b) ASM chart

State diagram and ASM chart for mod-6 counter.

BINARY MULTIPLIER

1 1 0 1	←	13_{10} … Multiplicand
1 0 1 0	←	10_{10} … Multiplier
0 0 0 0	←	Partial product 1
1 1 0 1	←	Partial product 2
0 0 0 0	←	Partial product 3
1 1 0 1	←	Partial product 4
1 0 0 0 0 0 0 1 0	←	130_{10} … Product

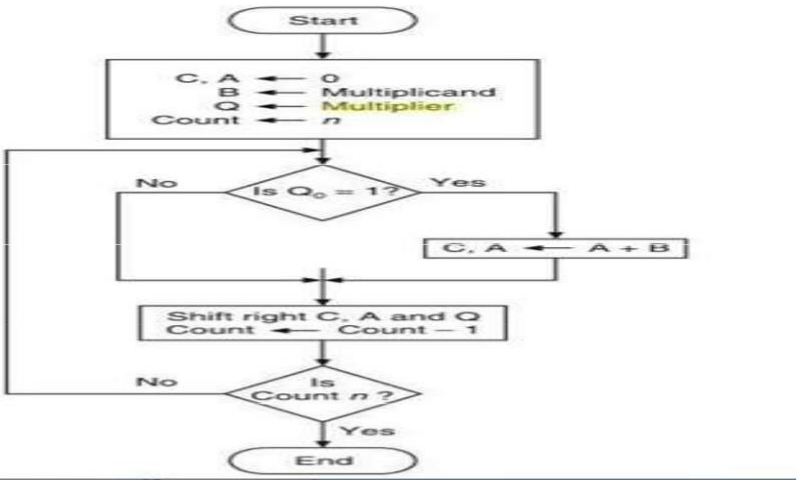
Data path subsystem for binary multiplier



Datapath subsystem for binary multiplier.

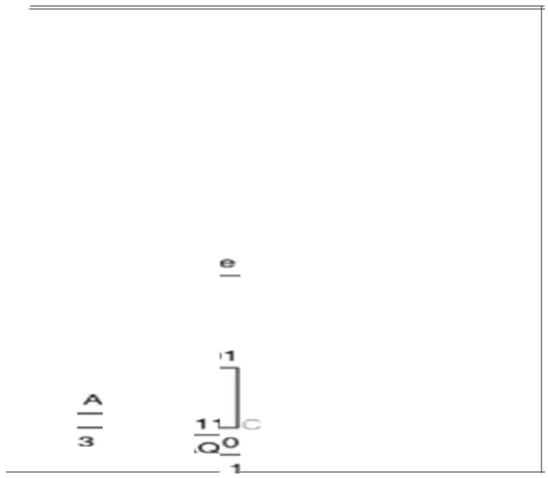
Multiplication Operation Steps

1. Bit 0 of multiplier operand (Q_0 of Q register) is checked.
2. If bit 0 (Q_0) is one then multiplicand and partial product are added and all bits of C, A and Q registers are shifted to the right one bit, so that the C bit goes into A_{n-1}, A_0 goes into Q_{n-1}, and $Q_0$is lost. If bit 0 (Q_0) is 0, then no addition is performed, only shift operation is carried out.
3. Steps 1 and 2 are repeated n times to get the desired result in the A and Q registers.



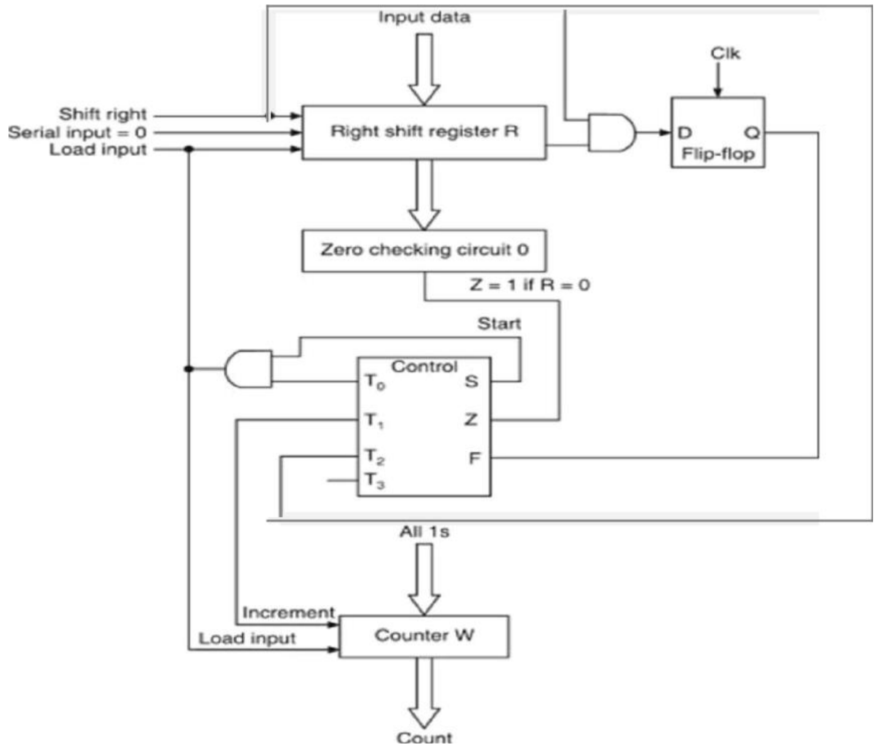
B	C	A	Q	Components	Count P
1101	O	O OOO	IOIO	B ← Multiplicand I OO t4 Q ← Multiplier $A \leftarrow 0, C \leftarrow 0, P \leftarrow n$	
1IO1	O	O OOO	IOIO	1	
	O	OOOO 0	I O I	C A Q shifted right	
				O 1 O i 21	
JIOI	O	OIIO	IOIO	$Q_0 = 1, A \leftarrow A + B$ C A P ← P − J right	
	t›	‹› ‹› I i u	i t› i	„=O. C A Q shifted right	001 (1)
1 1 0 1	1	0 0 0 0 0	1 0 I	$P \leftarrow P - 1$	
	0	1 0 0 0 0	0 1 0	C A @ .shi ftoct right	

flow cf-iart f<>r mwltipficati<>n in a corripwter.

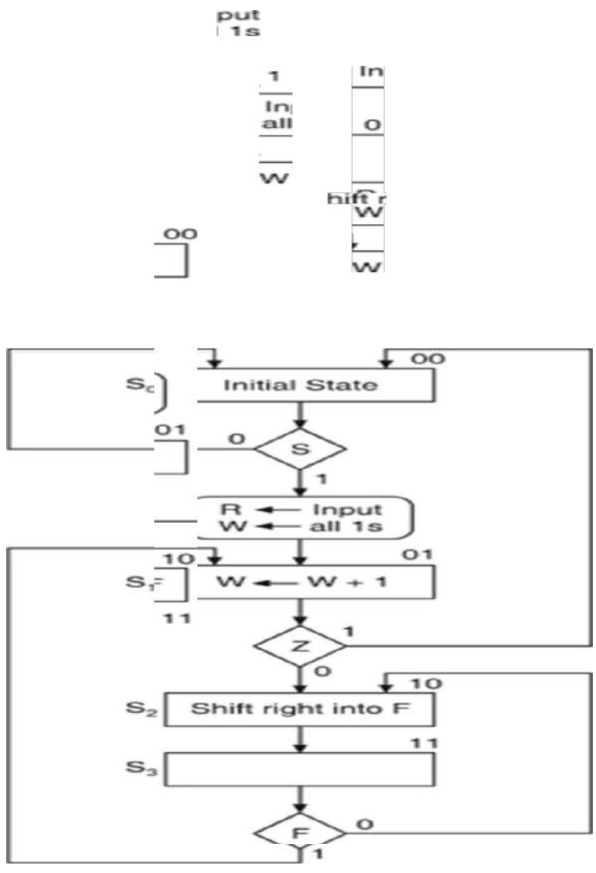


ASM FOR WEIGHING MACHINE

In tte a[$oriOin for tabolir minimization of Boolean expressions, we have to arrange be ininnnns in thc ascending order of their weights. This is only one of the iniuiy situations when we have tn examine the 1» of a given binary word. The wsight of a binary number is defined as the number of 5 }7f0 S9ltt 1f1 lls blf1d£\ fe]7fed nldtlOn.



Oatapeth **subsystem** lor **weighing** machine.



6r / S . Ln*ti atJy lite weighing stack ine i s in state Sq. one weighing process stacts wficn suu-t (S } signal becomes I . Whi le in state fi if *> i s I , the clock pulse cauws three jobs to be done sinTu ltaneousfy."

1. Bi n ay n u m ber is loader4 i nEo xug isCer R.
2. W regJ ster is set to all G s.
3. The xtachine is rransFerred to sta re S , .

State S_1: While in state S_1, the clock pulse causes two jobs to be done simultaneously:

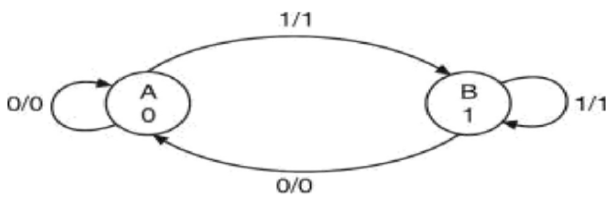
. COuntcr W in income rated by' 1{in thJz First councJ, all I s b+>c:Om &J1 US).

2. IT Z i s 0. the in ash ine gc• s la Lhc scale fi : i I Z i s I . tMe machinc gc•es to state fi .

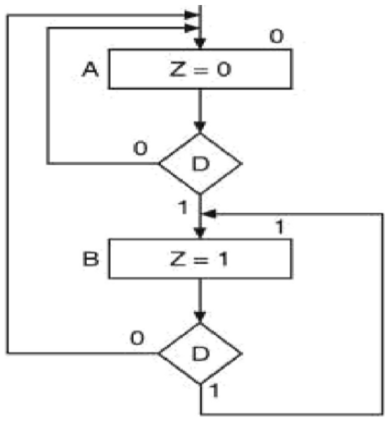
crore $ $_2$• In this state. regiMer R ix lifted righthyI hitin thaE L SC y ‹ws int‹t F and M $ B is 1naded with 0

fiinm Sy: In iii is siate, rhe value of F is chicken. Jf iz is 0, the machine is rransf'erred to the state S,, othew'ise the machine i3 transferred to state S,. Thus. when F - I. W is incremented.

All the operations occur in coincidence with the clock pulse while in the corresponding state. Also nokcc ‹w the rugi star k should eventually contain all 0s when the last I is stiir in into .

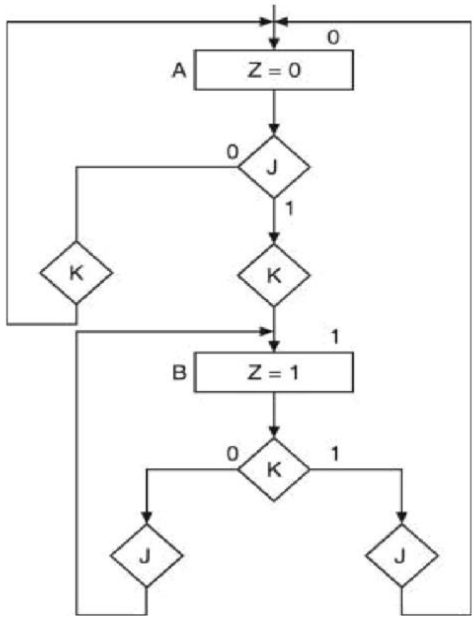


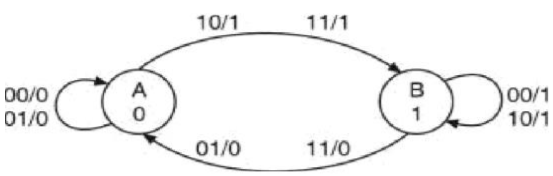
(a) State diagram



PS	NS, O/P	
	Input0	
	D - D	D - 1
A	A, D	B, 1
B	A, D	B, 1

(D) State table (c) ASM chart





(a) State diagram

PS	NS,OGP			
	Input J-K			
	OD	D1	10	11
A	A, O	A, O	B, 1	B, 1
B	B, 1	A, O	B, 1	A, 0

‹b› s‹ata lable (c) ADM chart