UNIT I

NUMBER SYSTEMS AND DIGITAL LOGIC FAMILIES

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

Introduction

Basically there are two types of signals in electronics,

i)   Analog

ii) Digital

Digital systems

Advantages:

❖ The usual advantages of digital circuits when compared to analog circuits are:Digital systems interface well with computers and are easy to control with software. New features can often be added to a digital system without changing hardware.

❖ Often this can be done outside of the factory by updating the product's software. So, the product's design errors can be corrected after the product is in a customer's hands.

❖ Information storage can be easier in digital systems than in analog ones. The noise-immunity of digital systems permits data to be stored and retrieved without degradation.

❖ In an analog system, noise from aging and wear degrade the information stored.

❖ In a digital system, as long as the total noise is below a certain level, the information can be recovered perfectly.

Disadvantages:

❖ In some cases, digital circuits use more energy than analog circuits to accomplish the same tasks, thus producing more heat as well. In portable or battery-powered systems this can limit use of digital systems.

❖ Digital circuits are sometimes more expensive, especially in small quantities.The sensed world is analog, and signals from this world are analog quantities.

❖ Digital circuits are sometimes more expensive, especially in small quantities. The sensed world is analog, and signals from this world are analog quantities.

❖ For example, light, temperature, sound, electrical conductivity, electric and magnetic fields are analog.

REVIEW OFNUMBER SYSTEMS

Many number systems are in use in digital technology. The most common are the decimal, binary, octal, and hexadecimal systems. The decimal system is clearly the most familiar to us because it is tools that we use every day.

Types of Number Systems are

- ❖ Decimal Number system
- ❖ Binary Number system
- ❖ Octal Number system
- ❖ Hexadecimal Number system

Table: Types of Number Systems

DECIMAL	BINARY	OCTAL	HEXADECIMAL
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Table: Numbersystemandtheir Base value

Number Systems		
System	**Base**	**Digits**
Binary	2	0 1
Octal	8	0 1 2 3 4 5 6 7
Decimal	10	0 1 2 3 4 5 6 7 8 9
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 A B C D E F

❖ Convertingfromonecodeformtoanothercodeformiscalledcodeconversion,likeconvertingfrom binaryto decimal orconverting from hexadecimal to decimal.

Binary-To-DecimalConversion:

Anybinarynumbercanbeconvertedtoitsdecimalequivalent simplybysummingtogether theweights of the variouspositions in the binarynumber whichcontaina1.

Binary	Decimal
11011_2	
$=2^4+2^3+0^1+2^1+2^0$	$=16+8+0+2+1$
Result	27_{10}

Decimal to binary Conversion:

Division	Remainder	Binary
25/2	=12+remainder of1	1 (LeastSignificantBit)
12/2	=6 +remainder of0	0
6/2	=3 +remainder of0	0
3/2	=1 +remainder of1	1
½	=0 +remainder of1	1 (MostSignificantBit)
Result	25_{10}	$=11001_2$

Binary to octal:

Example: $100\ 111010_2=(100)(111)(010)_2=4\ 7\ 2_8$

Octal to Binary:
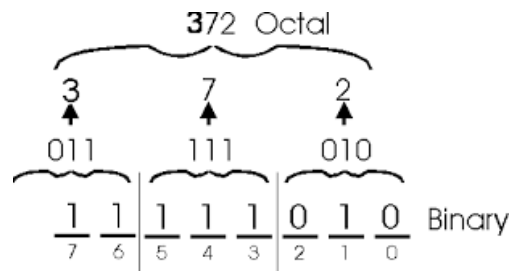
Decimal to octal:

Division	Result	Binary
177/8	=22+remainder of1	1 (LeastSignificantBit)
22/ 8	=2 +remainder of6	6
2 / 8	=0 +remainder of2	2 (Most Significant Bit)
Result	177_{10}	$=261_8$
Binary		$=010110001_2$

Octal to Decimal:
Example:
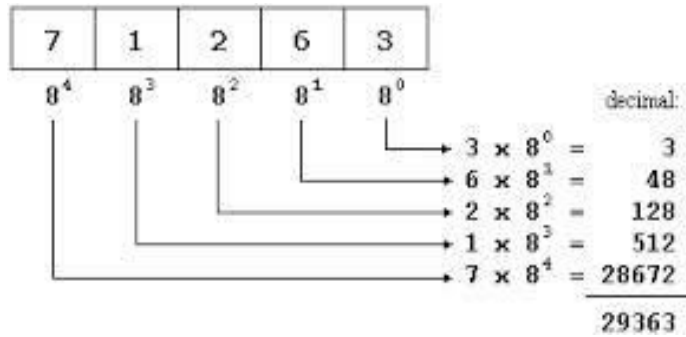
Decimal to Hexadecimal:

Division	Result	Hexadecimal
378/16	=23+remainder of10	A(LeastSignificantBit)23
23/16	=1 +remainder of7	7
1/16	=0 +remainder of1	1 (Most Significant Bit)
Result	37810	$=17A_{16}$
Binary		$=00010111\ 1010_2$

Binary-To-Hexadecimal:

Example: $1011\ 0010\ 1111_2 = (1011)\ (0010)\ (1111)_2 = B2F_{16}$

Hexadecimal to binary:
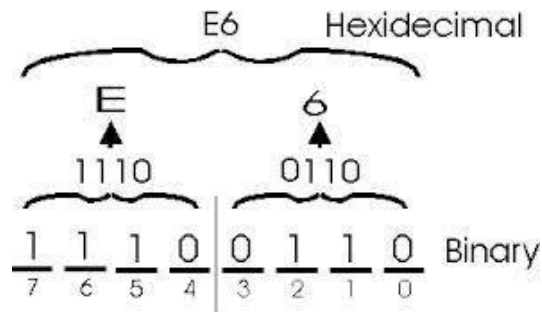
Octal-To-Hexadecimal / Hexadecimal-To-Octal Conversion:
 ❖ Convert Octal (Hexadecimal) to Binary first.
 ❖ Regroup the binary number by three bits per group starting from LSB if Octal is required.
 ❖ Regroup the binary number by four bits per group starting from LSB if Hexadecimal is required.

Octal to Hexadecimal: *(May 2014)*

Octal	Hexadecimal
=2 6 5 0	
= **010** 110**101**000	=**0101** 1010 **1000**(Binary)
Result	$=(5A8)_{16}$

Hexadecimal to octal:

Hexadecimal	Octal
$(5A8)_{16}$	=**0101** 1010 **1000**(Binary)
	=**010** 110**101**000(Binary)
Result	=2 6 5 0(Octal)

1's and2's complement:

❖ Complements are used in digital computers to simplify the subtraction operation and for logical manipulation.
❖ ThereareTWOtypesofcomplementsforeachbase-rsystem: theradixcomplementand the diminished radix complement.
❖ The first is referred to as there's complement and the second as the (r-1)'scomplement,whenthevalueofthebaserissubstitutedinthename.Thetwo typesarereferredtoasthe 2's complement and 1's complement for binary numbers and the 10's complement and 9's complement for decimal numbers.

Note:
  ▪ *The1'scomplementofabinarynumberisthenumberthatresultswhenwechangeall1'sto zeros and the zeros to ones.*
  ▪ *The2's complement is the binary number that results whenweadd1 to the1's complement.*
  ▪ *It is used to represent negativenumbers.*

2's complement=1'scomplement+1

Example 1) : Find 1's complement of $(1101)_2$
Sol: 1 1 0 1 ⬅ Number
              0 0 1 0 ⬅   1's complement

Example 2) : Find 2's complement of $(1001)_2$
Sol: 1 0 0 1   number

              0 1 1 0 ⬅ 1's complement
             +   1
             ――――――――
              0 1 1 1

Diminished Radix Complement:
    Given a number *N* in base *r* having *n* digits, the $(r - 1)$'s complement of *N,* i.e., its diminished radix complement, is defined as $(r^n - 1) - N$.
                The 9's complementof546700 is 999999 – 546700= 453299.

                The 9's complementof012398 is 999999 – 012398= 987601.

Radix Complement:

The r's complement of an n-digit number N in base r is defined as $r^n - N$ for $N \neq 0$ and as 0 for N = 0.

For examples:

The 10's complement of 012398 is 987602

The 10's complement of 246700 is 753300

Model 1:   *(Dec 2009)*

Using 10's complement, subtract 72532 – 3250.

$$
\begin{array}{rr}
M = & 72532 \\
\text{10's complement of } N = & +\underline{96750} \\
Sum = & 169282 \\
\text{Discard end carry } 10^5 = & -\underline{100000} \\
Answer = & 69282
\end{array}
$$

Model 2:

Using 10's complement, subtract 3250 – 72532.

$$
\begin{array}{rr}
M = & 03250 \\
\text{10's complement of } N = & +\underline{27468} \\
Sum = & 30718
\end{array}
$$

Model 3:

Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction (a) $X - Y$ and (b) $Y - X$ by using 2's complements. [NOV – 2019]

(a)
$$
\begin{array}{rr}
X = & 1010100 \\
\text{2's complement of } Y = + & \underline{0111101} \\
Sum = & 10010001 \\
\text{Discard end carry } 2^7 = - & \underline{10000000} \\
Answer: X - Y = & 0010001
\end{array}
$$

(b)
$$
\begin{array}{rr}
Y = & 1000011 \\
\text{2's complement of } X = & \underline{0101100} \\
Sum = & 1101111
\end{array}
$$

There is no end carry. Therefore, the answer is $Y - X = -(2$'s complement of $1101111) = -0010001$.

Model 4:

Given the two binary numbers X=1010100 and Y= 1000011, perform the subtraction (a) X-Y and (b) Y-X by using 1's complements. *(Dec 2009)*

(a) *X - Y* = 1010100 – 1000011

$$X= \quad 1010100$$
$$1\text{'s complement of } Y= +\underline{0111100}$$
$$\text{Sum} = \quad 10010000$$
$$\text{End around carry} = + \underline{1}$$
$$Answer: X - Y = \quad 0010001$$

(b) *Y - X* = 1000011 – 1010100

$$Y= \quad 1000011$$
$$1\text{'s complement of } X= +\underline{0101011}$$
$$\text{Sum} = \quad 1101110$$

There is no end carry. Therefore, the answer is *Y - X* = –(1's complement of 1101110) = –0010001.

****

ARITHMETIC OPERATIONS

Binary Addition:

Rules of Binary Addition

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 0, and carry 1 to the next most significant bit

Example:

Add: 00011010+00001100=00100110

```
              1   1

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

Binary Subtraction:

Rules of Binary Subtraction

- 0 -0 =0
- 0 -1 =1,and borrow 1 from the next more significant bit
- 1 -0 =1
- 1 -1 =0

Example:

   Sub: 00100101-00010001= 00010100

$$
\begin{array}{ccccccccc}
 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
- & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\hline
 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
\end{array}
$$

Binary Multiplication:

RulesofBinaryMultiplication

-     0 x 0 =0
-     0 x 1 =0
-     1 x 0 =0
-     1 x 1 =1,andnocarryorborrowbits

Example: Multiply the following binary numbers:

        (a) 0111 and 1101   (b) 1.011 and 10.01.

(a)   0111 × 1101

				0	1	1	1	Multiplicand
			×	1	1	0	1	Multiplier
				0	1	1	1	
			0	0	0	0		Partial
		0	1	1	1			Products
	0	1	1	1				
	1	0	1	1	0	1	1	Final Product

(b)   1.011 × 10.01

				1.	0	1	1	Multiplicand
			× 1	0.	0	1		Multiplier
				1	0	1	1	
			0	0	0	0		Partial
		0	0	0	0			Products
	1	0	1	1				
	1	1 .	0	0	0	1	1	Final Product

Binary Division:

Binary division is the repeated process of subtraction, just as in decimal division.

Example: Divide the following

(a) $11001 \div 101$

```
                1    0    1
      1    0    1 | 1    1    0    0    1
                   1    0    1
                   0    0    1    0    1
                             1    0    1
                             0    0    0
```

(b) $11110 \div 1001$

```
                     1   1.   0    1    0
      1   0   0   1 | 1    1    1    1    0
                     1    0    0    1
                     0    1    1    0    0
                     1    0    0    1
                          1    0    0    0    0
                          1    0    0    1
                               1    1    0
                          1    0    0    1
                               1    0    1
```

****

BINARYCODES

Explain the various codes used in digital systems with an example.(or)Explain in detail about Binary codes with an example

➢ In digital systems a variety of codes are used to serve different purposes, such as data entry, arithmetic operation, error detection and correction, etc.

➢ Selection of a particular code depends on the requirement.

➢ Binary codes are codes which are represented in binary system with modification from the original ones.

➢ Codes can be broadly classified into five groups.

     (i) Weighted Binary Codes

     (ii) Non-weighted Codes

     (iii) Error-detection Codes

     (iv) Error-correcting Codes

     (v) Alphanumeric Codes

Weighted Binary Codes

➢ If each position of a number represents a specific weight then the coding scheme is called weighted binary code.

BCD Code or 8421 Code:

➢ The full form of BCD is 'Binary-Coded Decimal'. Since this is a coding scheme relating decimal and binary numbers, *four bits are required* to code each decimal number.

- A decimal number in BCD (8421) is the same as its equivalent binary number only when the number is between 0 and 9. A BCD number greater than 10 looks different from its equivalent binary number, even though both contain 1's and 0's. Moreover, the binary combinations 1010 through 1111 are not used and have no meaning in BCD.
- Consider decimal 185 and its corresponding value in BCD and binary:

$$(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (10111001)_2$$

- For example, $(35)_{10}$ is represented as 0011 0101 using BCD code, rather than $(100011)_2$
- *Example*: *Give the BCD equivalent for the decimal number 589.*

    The decimal number is   5   8   9
    BCD code is   0101 1000 1001
    Hence, $(589)_{10} = (010110001001)_{BCD}$

2421 Code:

- Another weighted code is 2421 code. The weights assigned to the four digits are 2, 4, 2, and 1.
- The 2421 code is the same as that in BCD from 0 to 4. However, it varies from 5 to 9.
- For example, in this case the bit combination 0100 represents decimal 4; whereas the bit combination 1101 is interpreted as the decimal 7, as obtained from $2 \times 1 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 7$.
- This is also a self-complementary code.

BCD Addition:

Examples:

❖ Consider the addition of   184 + 576 = 760   in BCD:

```
BCD                 1       1
                0001    1000    0100        184
               +0101    0111    0110       +576
Binary sum      0111   10000    1010
Add 6                   0110    0110
BCD sum         0111    0110    0000        760
```

❖ Add the following BCD numbers: (a) 1001 and 0100, (b) 00011001 and 00010100


```
Solution
  (a)                 1  0  0  1
                    + 0  1  0  0
                      1  1  0  1   → Invalid BCD number           9
                    + 0  1  1  0   → Add 6                       +4
              0  0  0  1  0  0  1  1 → Valid BCD number.          13₁₀
                        1        3
  (b)         0  0  0  1    1  0  0  1
            + 0  0  0  1    0  1  0  0
              0  0  1  0    1  1  0  1 → Right group is invalid   19
                         + 0  1  1  0 → Add 6                     +14
              0  0  1  1    0  0  1  1 → Valid BCD number         33₁₀
                    3          3
```

Four Different Binary Codes for the Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
	1010	0101	0000	0001
Unused	1011	0110	0001	0010
bit	1100	0111	0010	0011
combi-	1101	1000	1101	1100
nations	1110	1001	1110	1101
	1111	1010	1111	1110

Non-weighted Codes

➢ It basically means that each position of the binary number is not assigned a fixed value.

➢ Excess-3 codes and Gray codes are such non-weighted codes.

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).

Example: 1000 of 8421 (BCD)=1011 in Excess-3

❖ *Convert (367)₁₀ into its Excess-3 code.*

Solution.   The decimal number is   3   6   7

Add 3 to each bit   +3   +3   +3

Sum   6   9   10

Converting the above sum into 4-bit binary equivalent, we have a

4-bit binary equivalent of   0110   1001   1010

Hence, the Excess-3 code for $(367)_{10}$ = 0110 1001 1010

Graycode:

- ❖ 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. In digital Gray code has got a special place.

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

- ❖ 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.*
- ❖ Important when an analog quantity must be converted to a digital representation. Only one bit changes between two successive integers which are being coded.

Example:
Binary to Gray Code Conversion:
Any binary number can be converted into equivalent Gray code by the following steps:

- i) the MSB of the Gray code is the same as the MSB of the binary number;
- ii) the second bit next to the MSB of the Gray code equals the Ex-OR of the MSB and second bit of the binary number; it will be 0 if there are same binary bits or it will be 1 for different binary bits;
- iii) the third bit for Gray code equals the exclusive-OR of the second and third bits of the binary number, and similarly all the next lower order bits follow the same mechanism.
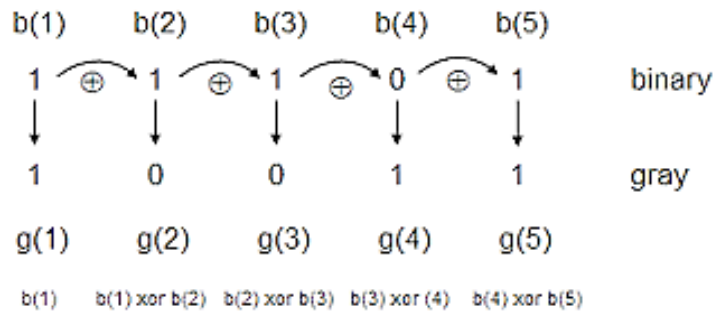
GrayCode to Binary Code Conversion:

Any Gray code can be converted into an equivalent binary number by the following steps:

  i.   The MSB of the binary number is the same as the MSB of the Gray code.
  ii. the second bit next to the MSB of the binary number equals the Ex-OR of the MSB of the binary number and second bit of the Gray code; it will be 0 if there are samebinary bits or it will be 1 for different binary bits;
  iii. the third bit for the binary number equals the exclusive-OR of the second bit of the binary number and third bit of the Gray code, and similarly all the next lower orderbits follow the same mechanism.
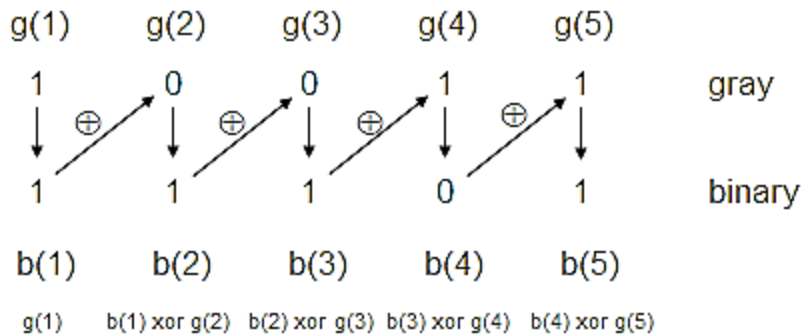
Errordetectingcodes

➢ Whendataistransmittedfromonepointtoanother,likeinwirelesstransmission,　　　　　o r itisjuststored, likeinharddisksandmemories,therearechancesthatdata ma ygetcorrupted.

➢ Todetectthesedata errors,weusespecialcodes,whichareerrordetection codes.

Twotypes ofparity

  ➢ **Evenparity:**Checksifthereisanevennumberofones;ifso,paritybitiszero.Whenthenumberof one'sisoddthenparitybitissetto 1.
  ➢ **OddParity:**Checksifthereisanoddnumberofones;ifso,paritybitiszero.　　　　　Whenthenumberof one'siseventhenparitybitis set to 1.

Errorcorrectingcode

➢ Error-correctingcodesnotonlydetecterrors,butalsocorrectthem.

➢ Thisisused 　　　　normallyinSatellite 　　　communication,whereturn-arounddelayisveryhighasisthe

probabilityofdata gettingcorrupt.

Hamming codes

- ➢ Hammingcodeaddsaminimumnumberofbitstothedatatransmitted inanoisychannel,tobeableto correct everypossible one-bit error.
- ➢ It candetect(not correct)two-biterrorsandcannotdistinguish between1-bitand2-bits inconsistencies. Itcan't-ingeneral-detect 3(ormore)-bits errors.

Alphanumeric Codes

- ➢ An alphanumeric code is a binary code of a group of elements consisting of ten decimal digits, the 26 letters of the alphabet (both in uppercase and lowercase), and a certain number of special symbols such as #, /, &, %, etc.

ASCII(AmericanStandardCode for InformationInterchange)

- ➢ It is actually a 7-bit code, where a character is represented with seven bits.
- ➢ The character is stored as one byte with one bit remainingunused.
- ➢ But often the extra bit is used to extend the ASCII to represent an additional128 characters.

EBCDIC codes

- ➢ EBCDICstandsfor*ExtendedBinary CodedDecimalInterchange*.
- ➢ It is also an alphanumeric code generally used in IBM equipment and in large computersfor communicating alphanumeric data.
- ➢ For the different alphanumeric characters the code grouping in this code is different from the ASCII code. It is actually an 8-bit code and a ninth bit is added as the parity bit.

****

Explain various theorems of Boolean algebra. (Nov – 2018)
Definition:

  Boolean algebra is an algebraic structure defined by a set of elements B, together with two binary operators. '+' and '- ', provided that the following (Huntington) postulates are satisfied;

Theorems of Boolean algebra:

The theorems of Boolean algebra can be used to simplify many a complex Boolean expression and also to transform the given expression into a more useful and meaningful equivalent expression.

T1: Commutative Law
(a) $A + B = B + A$
(b) $A B = B A$

T2: Associative Law
(a) $(A + B) + C = A + (B + C)$

(b) $(A B) C = A (B C)$

T3: Distributive Law
(a) $A (B + C) = A B + A C$
(b) $A + (B C) = (A + B) (A + C)$

T4: Identity Law
(a) $A + A = A$
(b) $A A = A$

T5: Negation Law
$(\bar{A}) = \bar{A}$ and $(\bar{\bar{A}}) = A$

T6: Redundancy
(a) $A + A B = A$
(b) $A (A + B) = A$

T7: Operations with '0' & '1'
(a) $0 + A = A$
(b) $1 A = A$
(c) $1 + A = 1$
(d) $0 A = 0$

T8 : Complement laws
   (a) $\bar{A} + A = 1$
   (b) $\bar{A}.A = 0$

T9 :   (a) $A + \bar{A}B = A + B$
    (b) $A. (\bar{A} + B) = A.B$

Postulates of Boolean algebra:

  The postulates of a mathematical system form the basic assumptions from which it is possible to deduce the rules, theorems, and properties of the system. The following are the important postulates of Boolean algebra:

1. $1.1 = 1, 0+0 = 0$.
2. $1.0 = 0.1 = 0, 0+1 = 1+0 = 1$.
3. $0.0 = 0, 1+1 = 1$
4. $1' = 0$ and $0' = 1$.

  Many theorems of Boolean algebra are based on these postulates, which can be used to simplifyBoolean expressions.

  The operators and postulates have the following meanings:
- ✓ The binary operator + defines addition.
- ✓ The additive identity is 0.
- ✓ The additive inverse defines subtraction.
- ✓ The binary operator **.**(dot) defines multiplication.
- ✓ The multiplicative identity is 1.
- ✓ The only distributive law applicable is that of **.**(dot) over +:

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

Two-Valued Boolean Algebra:

A two-valued Boolean algebra is defined on a set of two elements, $B = \{0, 1\}$, with rules for the two binary operators + and .(dot) as shown in the following operator tables.

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

Duality Principle:

The *duality principle* states that every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged. If the *dual* of an algebraic expression is desired, we simply interchange OR and AND operators and replace 1's by 0's and 0's by 1's.

DeMorgan's theorem:
1. The complement of product is equal to the sum of their complements. $(X.Y)'=X'+Y'$
2. The complement of sum is equal to the product of their complements. $(X+Y)' = X'.Y'$

Basic Theorems:

State and prove postulates and theorems of Boolean algebra.

Postulates and Theorems of Boolean Algebra

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$	
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$	
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$	
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$	
Theorem 3, involution		$(x')' = x$			
Postulate 3, commutative	(a)	$x + y = y + x$	(b)	$xy = yx$	
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$	
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$	
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$	
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$	

THEOREM 1(a): $x + x = x.$

Statement	Justification
$x + x = (x + x) \cdot 1$	postulate 2(b)
$= (x + x)(x + x')$	5(a)
$= x + xx'$	4(b)
$= x + 0$	5(b)
$= x$	2(a)

THEOREM 1(b): $x \cdot x = x.$

Statement	Justification
$x \cdot x = xx + 0$	postulate 2(a)
$= xx + xx'$	5(b)
$= x(x + x')$	4(a)
$= x \cdot 1$	5(a)
$= x$	2(b)

THEOREM 2(a): $x + 1 = 1.$

Statement	Justification
$x + 1 = 1 \cdot (x + 1)$	postulate 2(b)
$= (x + x')(x + 1)$	5(a)
$= x + x' \cdot 1$	4(b)
$= x + x'$	2(b)
$= 1$	5(a)

THEOREM 2(b): $x \cdot 0 = 0$ by duality.

THEOREM 3: $(x')' = x.$ From postulate 5, we have $x + x' = 1$ and $x \cdot x' = 0$, which together define the complement of x. The complement of x' is x and is also $(x')'$.

THEOREM 6(a): $x + xy = x.$

Statement	Justification
$x + xy = x \cdot 1 + xy$	postulate 2(b)
$= x(1 + y)$	4(a)
$= x(y + 1)$	3(a)
$= x \cdot 1$	2(a)
$= x$	2(b)

THEOREM 6(b): $x(x + y) = x$ by duality.

****

Boolean Functions

❖ 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.

Example, consider 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. The gate implementation of F1 is shown below.
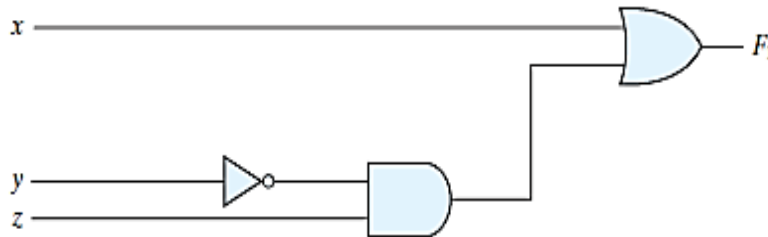
Example: Consensus Law: (function 4)

Simplify the following Boolean functions to a minimum number of literals.

1. $x(x' + y) = xx' + xy = 0 + xy = xy.$
2. $x + x'y = (x + x')(x + y) = 1(x + y) = x + y.$
3. $(x + y)(x + y') = x + xy + xy' + yy' = x(1 + y + y') = x.$
4. $xy + x'z + yz = xy + x'z + yz(x + x')$
$$= xy + x'z + xyz + x'yz$$
$$= xy(1 + z) + x'z(1 + y)$$
$$= xy + x'z.$$
5. $(x + y)(x' + z)(y + z) = (x + y)(x' + z)$, by duality from function 4.

Complement of a function:

The complement of a function F is obtained from an interchange of 0's for 1's and 1's for 0's in the value of F.

Example:

1.

$$(A + B + C)' = (A + x)' \qquad \text{let } B + C = x$$
$$= A'x' \qquad \text{by theorem 5(a) (DeMorgan)}$$
$$= A'(B + C)' \qquad \text{substitute } B + C = x$$
$$= A'(B'C') \qquad \text{by theorem 5(a) (DeMorgan)}$$
$$= A'B'C' \qquad \text{by theorem 4(b) (associative)}$$

2. **Find the complement of the functions $F1 = x'yz' + x'y'z$ and $F2 = x(y'z' + yz)$.**

   By applying DeMorgan's theorems as many times as necessary, the complements are obtained as follows:

   $$F_1' = (x'yz' + x'y'z)' = (x'yz')'(x'y'z)' = (x + y' + z)(x + y + z')$$
   $$F_2' = [x(y'z' + yz)]' = x' + (y'z' + yz)' = x' + (y'z')'(yz)'$$
   $$= x' + (y + z)(y' + z')$$
   $$= x' + yz' + y'z$$

3. **Find the complement of the functions $F1 = x'yz' + x'y'z$ and $F2 = x(y'z' + yz)$ by taking their duals and complementing each literals.**

   Solution:

   1. $F_1 = x'yz' + x'y'z$.
      The dual of F_1 is $(x' + y + z')(x' + y' + z)$.
      Complement each literal: $(x + y' + z)(x + y + z') = F_1'$.

   2. $F_2 = x(y'z' + yz)$.
      The dual of F_2 is $x + (y' + z')(y + z)$.
      Complement each literal: $x' + (y + z)(y' + z') = F_2'$.

   ****

Canonical and Standard forms:

Explain canonical SOP & POS form with suitable example.

- ➢ Binary logic values obtained by the logical functions and logic variables are in binary form. An arbitrary logic function can be expressed in the following forms.
  (i) Sum of the Products (SOP)
  (ii) Product of the Sums (POS)
- ➢ Boolean functions expressed as a sum of minterms or product of maxterms are said to be in *canonical form*.

Product term:

The AND function is referred to as a product. The variable in a product term can appear either in complementary or uncomplimentary form. **Example: ABC'**

Sum term:

The OR function is referred to as a Sum. The variable in a sum term can appear either in complementary or uncomplimentary form. **Example: A+B+C'**

Sum of Product (SOP):

The logical sum of two or more logical product terms is called sum of product expression. It is basically an OR operation of AND operated variables. **Example: Y=AB+BC+CA**

Product of Sum (POS):

The logical product of two or more logical sum terms is called product of sum expression. It is basically an AND operation of OR operated variables. **Example: Y=(A+B).(B+C).(C+A)**

Minterm:

  A product term containing all the K variables of the function in either complementary or uncomplimentary form is called Minterm or standard product.

Maxterm:

  A sum term containing all the K variables of the function in either complementary or uncomplimentary form is called Maxterm or standard sum.

Minterms and Maxterms for Three Binary Variables

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
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 SOP Expression:

  The minterms whosesum defines the Boolean function are those which give the 1's of the function in a truth table.

Procedure for obtaining Canonical SOP expression:

 ✓ Examine each term in a given logic function. Retain if it is a minterm, continue to examine the next term in the same manner.

 ✓ Check for the variables that are missing in each product which is not minterm. Multiply the product by $(X+X')$, for each variable X that is missing.

 ✓ Multiply all the products and omit the redundant terms.

Example:

  Express the Boolean function F = A + B'C as a sum of minterms. (May -10)(Nov – 2018)

Solution:

  The function hasthree 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,

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

B.ARUNKUMAR, Asst. Prof.,/ECE   Page 20

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 1 $(x + x = x)$, it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

$$F = A'B'C + AB'C' + AB'C + ABC' + ABC = m1 + m4 + m5 + m6 + m7$$
$$F(A, B, C) = \sum(1, 4, 5, 6, 7)$$

Example:*Obtain the canonical sum of product form of the following function.   (May 2014)*

$$F (A, B, C) = A + BC$$
$$= A (B + B') (C + C') + BC (A + A')$$
$$= (AB + AB') (C + C') + ABC + A'BC$$
$$= ABC + AB'C + ABC' + AB'C' + ABC + A'BC$$
$$= ABC + AB'C + ABC' + AB'C' + A'BC \text{ (as } ABC + ABC = ABC)$$

Hence the canonical sum of the product expression of the given function is

$$F (A, B) = ABC + AB'C + ABC' + AB'C' + A'BC.$$

Canonical POS Expression:

The Maxterms whose product defines the Boolean function are those which give the 1's of the function in a truth table.

Procedure for obtaining Canonical POS expression:

✓ Examine each term in a given logic function. Retain if it is a maxterm, continue to examine the next term in the same manner.

✓ Check for the variables that are missing in each sum which is not maxterm. Add (X.X'), for each variable X that is missing.

✓ Expand the expression using distributive property eliminate the redundant terms.

Example:

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

$$F = (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z')$$
$$= M_0 M_2 M_4 M_5$$

A convenient way to express this function is as follows:

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

Example:

Obtain the canonical product of the sum form of the following function.

F (A, B, C) = (A + B′) (B + C) (A + C′)   **(Dec 2012)**

Solution:

$$F (A, B, C) = (A + B') (B + C) (A + C')$$
$$= (A + B' + 0) (B + C + 0) (A + C' + 0)$$
$$= (A + B' + CC') (B + C + AA') (A + C' + BB')$$
$$= (A + B' + C) (A + B' + C') (A + B + C) (A' + B + C) (A + B + C')$$
$$(A + B' + C')$$

[using the distributive property, as X + YZ = (X + Y)(X + Z)]

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

[as (A + B′ + C′) (A + B′ + C′) = A + B′ + C′]

Hence the canonical product of the sum expression for the given function is

$$F (A, B, C) = (A + B' + C) (A + B' + C') (A + B + C) (A' + B + C) (A + B + C')$$

****

Karnaugh Map (K-map):

- ❖ Using Boolean algebra to simplify Boolean expressions can be difficult. The Karnaugh map provides a simple and straight-forward method of minimizing Boolean expressions which represent combinational logic circuits.
- ❖ A Karnaugh map is a pictorial method of grouping together expressions with common factors and then eliminating unwanted variables.
- ❖ A Karnaugh map is a two-dimensional truth-table. Note that the squares are numbered so that the binary representations for the numbers of two adjacent squares differ in exactly one position.

Rules for Grouping together adjacent cells containing 1's:
- Groups must contain 1, 2, 4, 8, 16 (2^n) cells.
- Groups must contain only 1 (and X if don't care is allowed).
- Groups may be horizontal or vertical, but not diagonal.
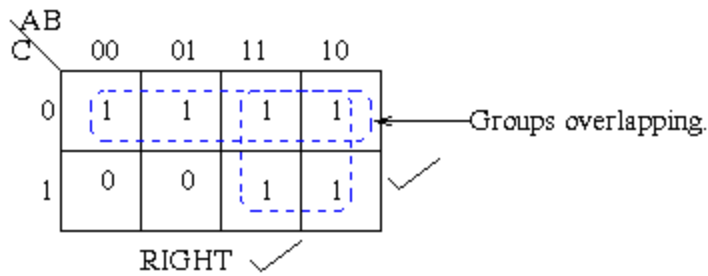- Groups should be as large as possible.
- Each cell containing a 1 must be in at least one group.
- Groups may overlap.
- Groups may wrap around the table. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.
- There should be as few groups as possible.

Obtaining Product Terms

- If A is a variable that has value 0 in all of the squares in the grouping, then the complemented form A is in the product term.
- If A is a variable that has value 1 in all of the squares in the grouping, then the true form A is in the product term.

- If A is a variable that has value 0 for some squares in the grouping and value 1 for others, then it is not in the product term

The Format of K-Maps:

K-Maps of 2 Variables:

(a) xy   (b) x + y
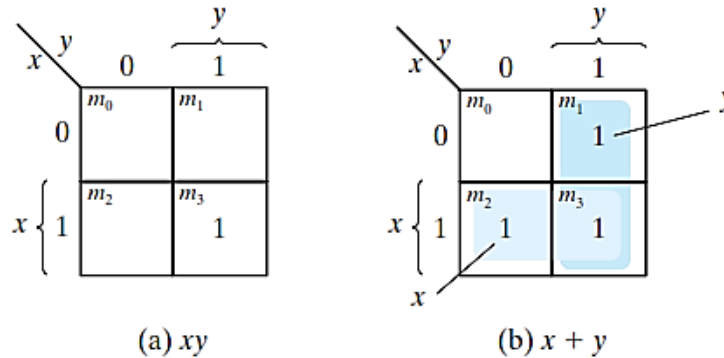
K-Maps of 3 Variables:
- ❖ Simplify the boolean function $F(x, y, z) = \Sigma(2, 3, 4, 5)$
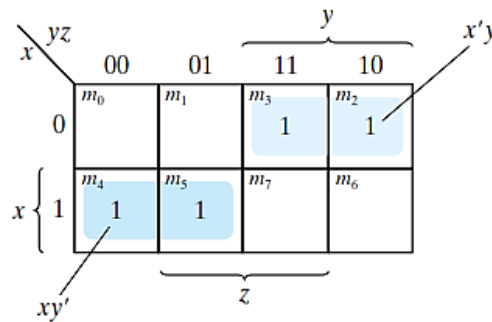
$$F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$$

- ❖ Simplify the boolean function $F(x, y, z) = \Sigma 3, 4, 6, 7$
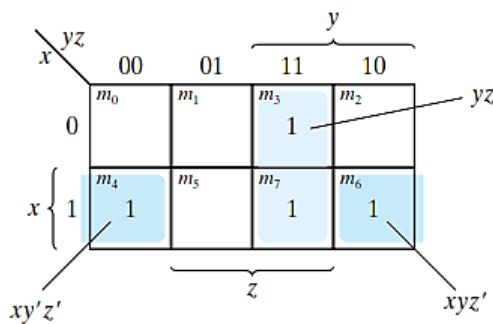
Note: $xy'z' + xyz' = xz'$

$$F(x, y, z) = \Sigma(3, 4, 6, 7) = yz + xz'$$

K-Maps of 4 Variables:

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

(a)

wx \ yz	00	01	11	10
00	$w'x'y'z'$	$w'x'y'z$	$w'x'yz$	$w'x'yz'$
01	$w'xy'z'$	$w'xy'z$	$w'xyz$	$w'xyz'$
11	$wxy'z'$	$wxy'z$	$wxyz$	$wxyz'$
10	$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$

(b)

❖ Simplify the boolean function $F(w,x,y,z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

Note: $w'y'z' + w'yz' = w'z'$
$xy'z' + xyz' = xz'$

$F(w, x, y, z) = \Sigma(0,1, 2, 4, 5, 6, 8, 9, 12, 13, 14) = \quad y' + w'z' + xz'$

❖ Simplify the Boolean function

$$F(w, x, y, z) = \Sigma(1, 3, 7, 11, 15)$$

which has the don't-care conditions

$$d(w, x, y, z) = \Sigma(0, 2, 5)$$

(a) K-map

	yz 00	01	11	10
00	X (m_0)	1 (m_1)	1 (m_3)	X (m_2)
01	0 (m_4)	X (m_5)	1 (m_7)	0 (m_6)
11	0 (m_{12})	0 (m_{13})	1 (m_{15})	0 (m_{14})
10	0 (m_8)	0 (m_9)	1 (m_{11})	0 (m_{10})

(a) $F = yz + w'x'$

(b) K-map

	yz 00	01	11	10
00	X (m_0)	1 (m_1)	1 (m_3)	X (m_2)
01	0 (m_4)	X (m_5)	1 (m_7)	0 (m_6)
11	0 (m_{12})	0 (m_{13})	1 (m_{15})	0 (m_{14})
10	0 (m_8)	0 (m_9)	1 (m_{11})	0 (m_{10})
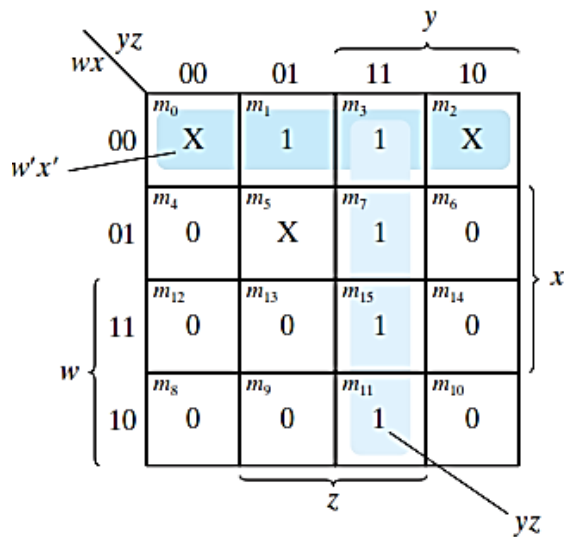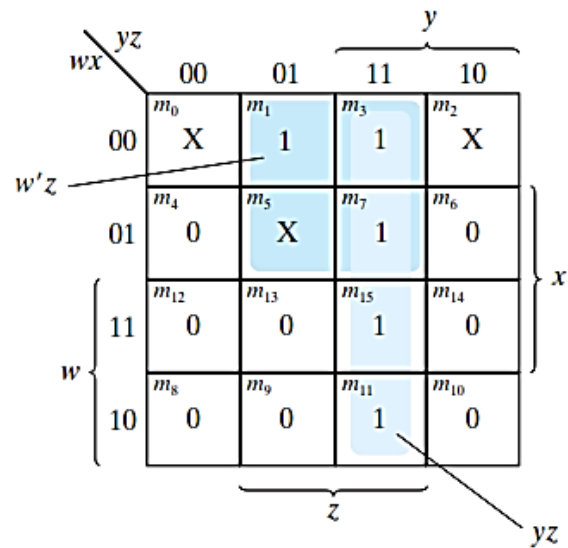
(b) $F = yz + w'z$

Note:

Karnaugh Maps - Rules of Simplification

The Karnaugh map uses the following rules for the simplification of expressions by *grouping* together adjacent cells containing *ones*

- **Groups may not include any cell containing a zero**

	A 0	1
0	0	
1	1	

WRONG ✕

	A 0	1
0	0	
1	1	1

RIGHT ✓

- **Groups may be horizontal or vertical, but not diagonal.**

	A 0	1
0	0	1
1	1	0

WRONG ✕

	A 0	1
0	0	1
1	1	1

RIGHT ✓
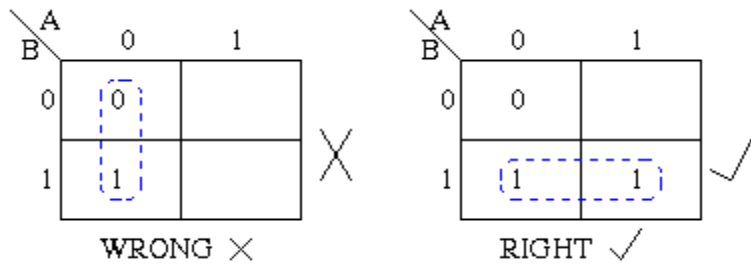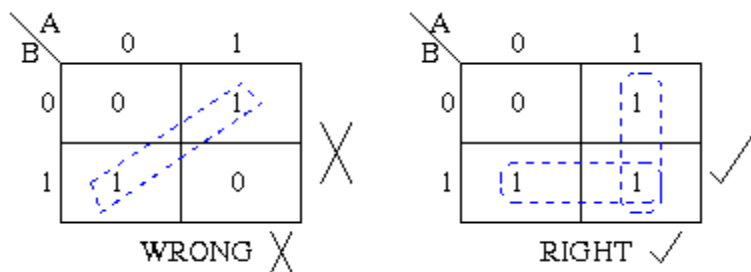
- **Groups must contain 1, 2, 4, 8, or in general 2^n cells. That is if n = 1, a group will contain two 1's since $2^1 = 2$. If n = 2, a group will contain four 1's since $2^2 = 4$.**
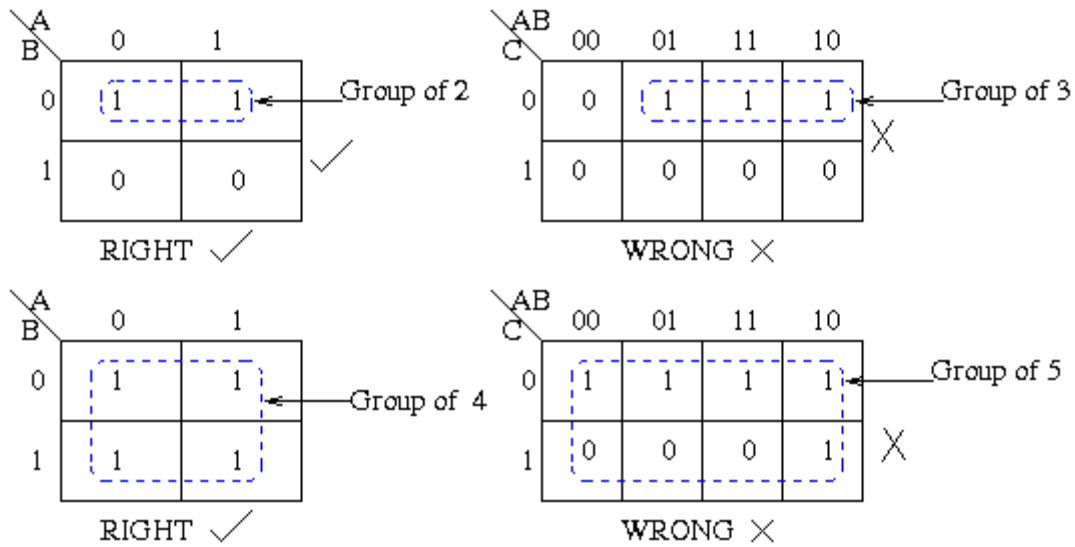
- **Each group should be as large as possible.**
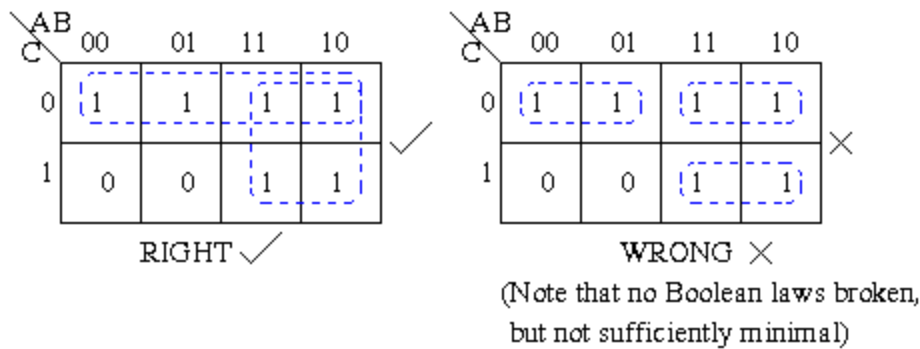
- **Each cell containing a *one* must be in at least one group.**
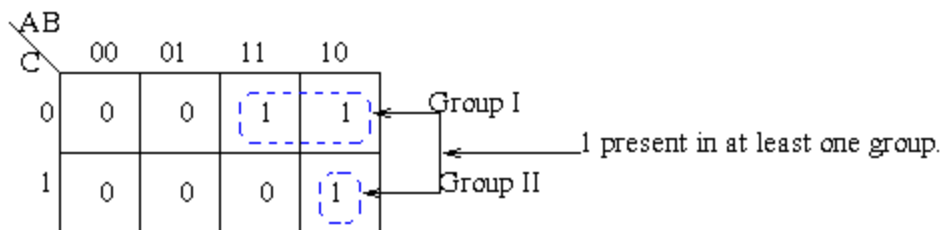
- **Groups may overlap.**
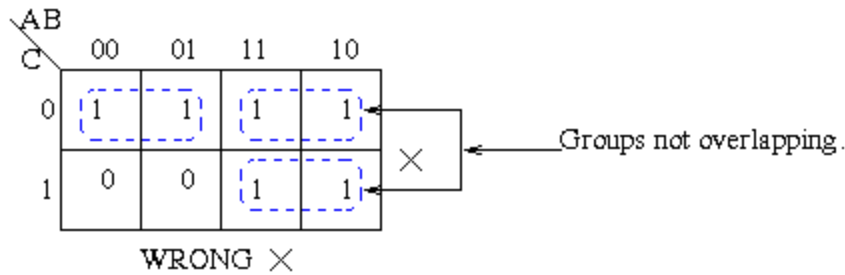
- **There should be as few groups as possible, as long as this does not contradict any of the previous rules.**
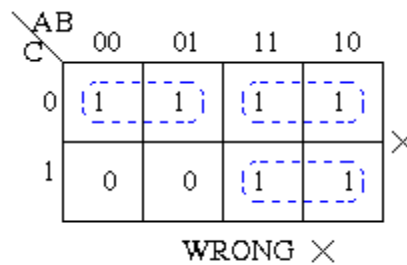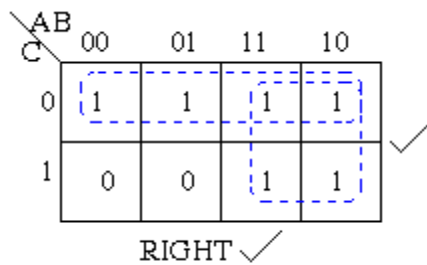
Summmary:

1. No zeros allowed.
2. No diagonals.
3. Only power of 2 numbers of cells in each group.
4. Groups should be as large as possible.
5. Everyone must be in at least one group.
6. Overlapping allowed.
7. Wrap around allowed.
8. Fewest numbers of groups possible.

Don't care combination:

In certain digital systems, some input combinations never occur during the process of normal operation because those input conditions are guaranteed never to occur. Such input combinations are don't care conditions.

Completely specified functions:

If a function is completely specified, it assumes the value 1 for some input combinations and the value 0 for others.

Incompletely specified functions:

There are functions which assume the value 1 for some combinations and 0 for some other and either 0 or 1 for the remaining combinations. Such a functions are called incompletely specified .

Prime Implicants:

A primeimplicant is a product term obtained by combining the maximum possible number ofadjacent squares in the map. If a minterm in a square is covered by only one primeimplicant, that prime implicant is said to be essential.

****

Quine-McCluskey (or) Tabulation Method

Minimization of Logic functions:
Steps:

- ✓ A set of all prime implicants of the function must be obtained.

- ✓ From the set of prime implicants, a set of essential implicants must be determined by preparing a prime implicant chart.

- ✓ The minterm which are not covered by the essential implicants are taken into consideration and a minimum cover is obtained from the remaining prime implicants.

Example: *(Nov-06,07,10,May- 10,08)*

Simplify the boolean function F(A,B,C,D)= ∑m (1,3,6,7,8,9,10,12,14,15) + ∑d (11,13) using Quine McClusky method. *(Apr 2017)*

Step:1

Minterms	Binary representation	Minterms	Binary representation
m_1	0 0 0 1	m_1	0 0 0 1 ✓
m_3	0 0 1 1	m_8	1 0 0 0 ✓
m_6	0 1 1 0	m_3	0 0 1 1 ✓
m_7	0 1 1 1	m_6	0 1 1 0 ✓
m_8	1 0 0 0	m_9	1 0 0 1 ✓
m_9	1 0 0 1	m_{10}	1 0 1 0 ✓
m_{10}	1 0 1 0	m_{12}	1 1 0 0 ✓
m_{12}	1 1 0 0	m_7	0 1 1 1 ✓
m_{14}	1 1 1 0	m_{14}	1 1 1 0 ✓
m_{15}	1 1 1 1	dm_{11}	1 0 1 1 ✓
dm_{11}	1 0 1 1	dm_{13}	1 1 0 1 ✓
dm_{13}	1 1 0 1	m_{15}	1 1 1 1 ✓

Step:2

Minterms	Binary representation	Minterms	Binary representation
1, 3	0 0 − 1 ✓	1, 3, 9, 11	− 0 − 1
1, 9	− 0 0 1 ✓	8, 9, 10, 11 ✓	1 0 − −
8, 9	1 0 0 − ✓	8, 10, 12, 14	1 − − 0
8, 10	1 0 − 0 ✓		
8, 12	1 − 0 0 ✓	6, 7, 14, 15 ✓	− 1 1 −
3, 7	0 − 1 1 ✓		
3, 11	− 0 1 1 ✓	12, 13, 14, 15	1 1 − −
6, 7	0 1 1 − ✓		
6, 14	− 1 1 0 ✓		
9, 11	1 0 − 1 ✓		
9, 13	1 − 0 1 ✓		
10, 14	1 − 1 0 ✓		
10, 11	1 0 1 − ✓		
12, 14	1 1 − 0 ✓		
12, 13	1 1 0 − ✓		
7, 15	− 1 1 1 ✓		
14, 15	1 1 1 − ✓		

Step:3

Prime implicants	Binary representation
1, 3, 9, 11 ($\overline{B}D$)	$-\,0\,-\,1$
8, 9, 10, 11, 12, 13, 14, 15 (A)	$1\,-\,-\,-$
6, 7, 14, 15 (BC)	$-\,1\,1\,-$

Step:4

Prime implicants	m_1	m_3	m_6	m_7	m_8	m_9	m_{10}	m_{12}	m_{14}	m_{15}	dm_{11}	dm_{13}
1, 3, 9, 11 ($\overline{B}D$)	⊙	⊙				⊙					⊙	
8, 9, 10, 11, 12, 13, 14, 15					⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙
6, 7, 14, 15 (BC)			⊙	⊙					⊙	⊙		

$\therefore\ F(A, B, C, D) = \overline{B}D + A + BC$

****

Logic gates

Explain about different types of logic gates.(OR) What are Universal gates? Construct any four basic gates using only NOR gates and using only NAND gates.   *(May 2011)[NOV – 2019]*

❖ A **logic gate** is an idealized or physical device implementing a Boolean function; that is, it performs a logical operation on one or more logical inputs, and produces a single logical output.

Positive and Negative Logic
❖ The binary variables two states, i.e. the logic '0' state or the logic '1' state. These logic states in digital systems such as computers.
❖ These are represented by two different voltage levels or two different current levels.
❖ If the more positive of the two voltage or current levels represents a logic '1' and the less positive of the two levels represents a logic '0', then the logic system is referred to as **a *positive logic system.***
❖ **If** the more positive of the two voltage or current levels represents a logic '0' and the less positive of the two levels represents a logic '1', then the logic system is referred to as **a *negative logic system.***

Truth Table

A truth table lists all possible combinations of input binary variables and the corresponding outputs of a logic system.

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

Universal Gates

❖ The OR, AND and NOT gates are the three basic logic gates as they together can be used to construct the logic circuit for any given Boolean expression.

❖ The NOR and NAND gates have the property that they individually can be used to hardware-implement a logic circuit corresponding to any given Boolean expression.

❖ That is, it is possible to use either only NAND gates or only NOR gates to implement any Boolean expression. This is so because a combination of NAND gates or a combination of NOR gates can be used to perform functions of any of the basic logic gates. It is for this reason that NAND and NOR gates are universal gates.

1. What is digital logic family? Give the classification of digital logic family.

DIGITAL LOGIC FAMILIES:

- A digital family is a group of compatible devices with the same logic levels and supply voltages.
- According to components used in the logic family, digital logic families are classified into
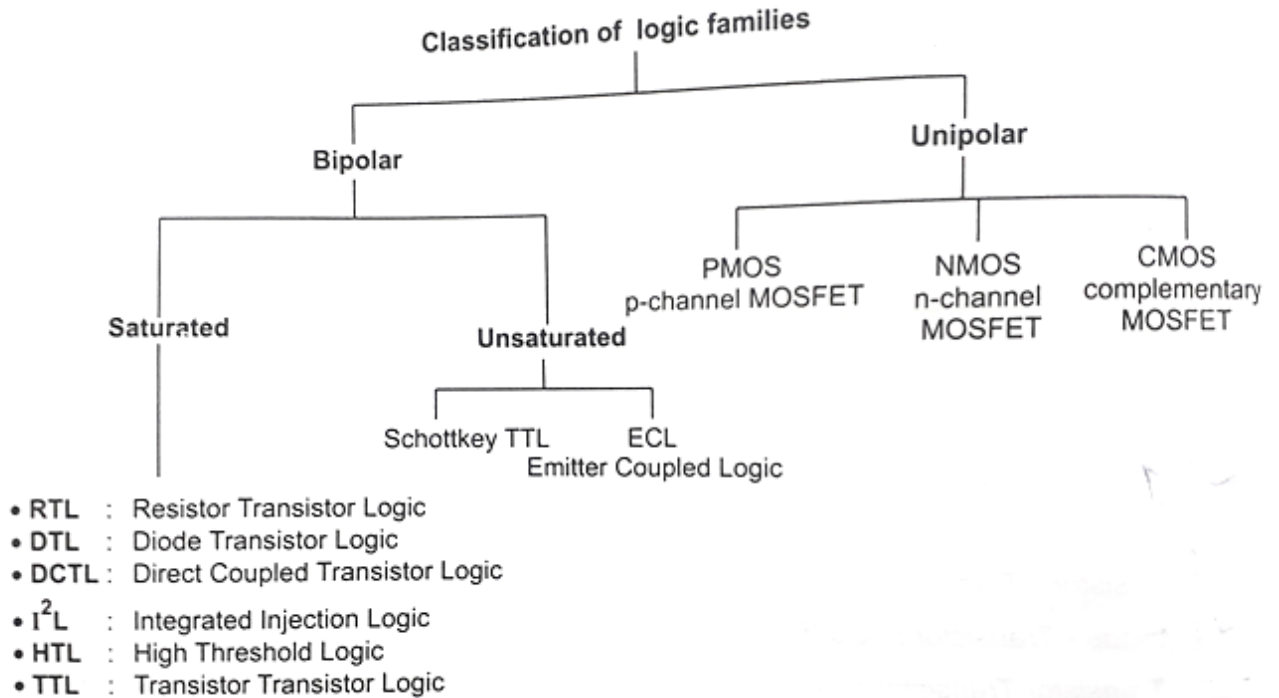

- RTL   :   Resistor Transistor Logic
- DTL   :   Diode Transistor Logic
- DCTL :   Direct Coupled Transistor Logic

- I^2L :   Integrated Injection Logic
- HTL   :   High Threshold Logic
- TTL   :   Transistor Transistor Logic

Figure 1.1 : classification of logic families

2. What are the characteristics of digital logic family? Explain its characteristics.   (Dec-06,08; May-08,11,14,18)

Characteristics of digital logic families:

Propagation delay:
- The propagation delay of a gate is basically the time interval between the application of the input pulse and the occurrence of the resulting output pulse.
- The propagation delay is very important characteristic of logic circuits because it limits the speed at which they can operate.
- The shorter the propagation delay, higher the speed of the circuit and vice-versa.
- The propagation delay is determined using two basic time intervals:
    1. t_{PLH} : It is the delay time measured when output is changing from logic 0 to logic 1 state (LOW to HIGH)
    2. t_{PHL} : It is the delay time measured when output is changing from logic 1 to logic 0 state (HIGH to LOW).
- When the t_{PHL} and t_{PLH} are not equal, the larger value is considered as a propagation delay time for that logic gate, i.e.

t_p = max (t_{PLH}, t_{PHL})

Power dissipation:

- The amount of power that an IC dissipates is determined by the average supply current, I_{CC} that it draws from the V_{CC} supply. It is the product of I_{CC} and V_{CC}.
- For ICs, the value of I_{CC} for a LOW gate output (I_{CCL}) is different from a HIGH output (I_{CCH}).
- Therefore, average I_{CC} is determined based on the 50% duty cycle operation of the gate (LOW half of the time and HIGH half of the time).

$$I_{CC(avg)} = \frac{I_{CCH} + I_{CCL}}{2}$$

This can be used to calculate average power dissipation as,

$$P_{D(avg)} = I_{CC(avg)} X V_{CC}$$

Current and voltage parameter:

$V_{IL(min)}$ **– High-Level input voltage:** It is the minimum voltage level required for a logic 1 at an input. Any voltage below this level will not be accepted as a HIGH by the logic circuit.

$V_{IL(max)}$ **– Low-Level input voltage:** It is the maximum voltage level required for a logic 0 at an input. Any voltage above this level will not be accepted as a LOW by the logic circuit.

$V_{OH(min)}$ **– High-level output voltage:** It is the minimum voltage level at the logic circuit output in the logical 1 state under defined load conditions.

$V_{OL(max)}$ **– Low-level output voltage:** It is the maximum voltage level at the logic circuit output in the logical 0 state under defined load conditions.

I_{IH} **– High-level input current:** It is current that flows into an input when a specified high-level voltage is applied to that input.

I_{IL} **– Low-level input current:** It is current that flows into an input when a specified low-level voltage is applied to that input.
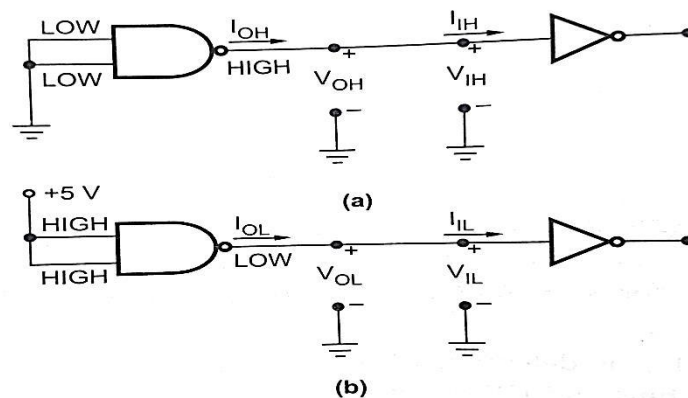
Figure 1.2 : currents and voltages in the two logic states

I$_{OH}$ – High-level output current: It is the current that flows from an output in the logical 1 state under specified load conditions.

I$_{OL}$ – Low-level output current: It is the current that flows from an output in the logical 0 state under specified load conditions.

Noise margin and logical voltages levels:

In digital circuits, the binary 0 and 1 are represented by a pair of voltage levels. Each logic family has a different standard which is shown in the table

Family	Logic 0	Logic 1
TTL	0V	+5V
ECL	-1.7V	-0.9V
CMOS	0V	3-15V

- The unwanted signals are called noise and can sometimes cause the voltage at the input to logic circuit to drop below $V_{IH(min)}$ or rise above $V_{IL(max)}$, which could produce unpredictable operation.
- The noise immunity at the logic circuit refers to the circuit's ability to tolerate the noise without causing spurious changes in the output voltage.
- To avoid this problem due to noise, voltage level $V_{IH(min)}$ is kept at a few fraction of volts below $V_{OH(min)}$ and voltage level $V_{IL(max)}$ is kept above $V_{OL(max)}$, at the design time.
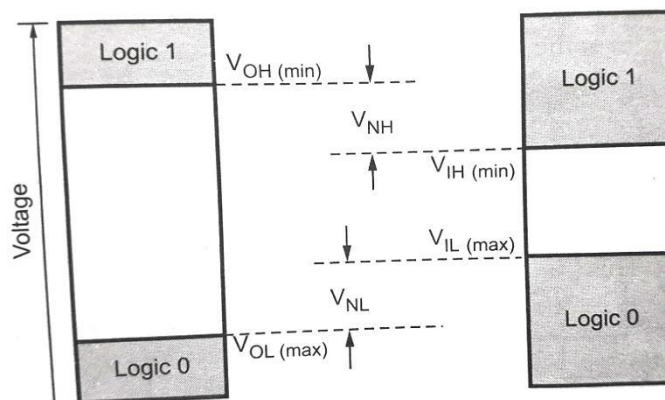
Figure 1.3 : Noise margins

- V_{NH} is the difference between the lowest possible HIGH output, $V_{oH(min)}$ and the minimum voltage, $V_{IH(min)}$ required for a HIGH input.
- This voltage difference, V_{NH} is called high-state noise margin. Similarly, we have low-state noise margin.
- It is the voltage difference between the largest possible low output, $V_{OL(max)}$ and the maximum voltage, $V_{IL(max)}$ required for a Low input.

- In short we can write as,

$V_{NH} = V_{OH(min)} - V_{IH(min)}$ and

$V_{NH} = V_{IL(max)} - V_{OL(max)}$

- The noise margin allows the digital circuit to function properly if noise voltages are within the limits of V_{NH} and V_{NL} for a particular logic family.

Fan-in and fan-out:

- The maximum number of inputs of several gates that can be driven by the output of a logic gate is decided by the parameter called fan-out.
- For example, a logic gate with fan-out 10 can drive maximum 10 logic inputs from the same family.
- The fan-in of a digital logic gate refers to the number of inputs. For example, an inverter has a fan-in of 1, a 2-input NOR gate has a fan-in of 2, a 4-input NAND gate has a fan-in of 4 and so-on.

Current sinking:

- A device output is said to sink current when current flows from the power supply, through the load and through the device output to ground.

Current sourcing:

- A device output is said to source current when the current from the power supply, out of the device output and through the load to ground.
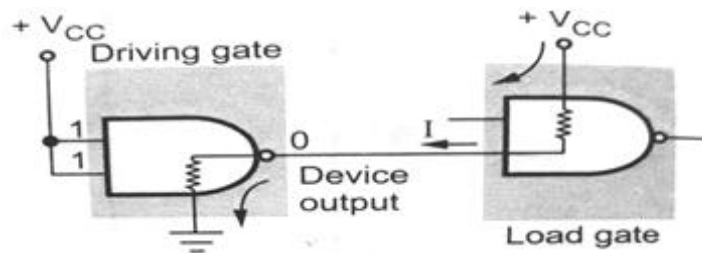
Figure 1.4(a) :Current sinking
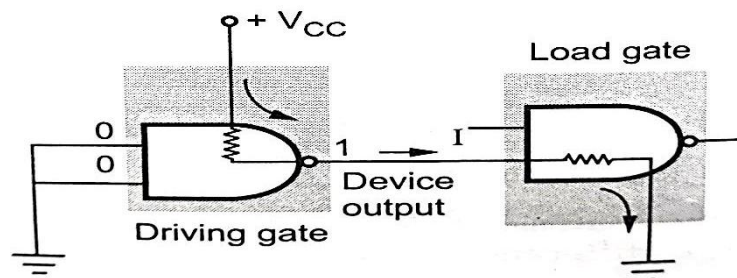
Figure 1.4(b) :Current sourcing

Speed power product:

- In general, for any digital IC, it is desirable to have shorter propagation delays (higher speed) and lower values of power dissipation.
- There is usually a trade-off between switching speed and power dissipation in the design of logic circuit i.e. speed is gained at the expense of increased power dissipation.

46

- Therefore, a common means of measuring and comparing the overall performance of an IC family is the speed-power product (SPP). It is also called Figure of Merit.

3. Write a note on RTL family? Explain the operation of 2-input RTL NOR gate. (Dec-17)

Resistor – Transistor Logic (RTL):
RTL circuit:
- RTL circuits consist of resistors and transistors. The figure 1.5 shows the 2-input RTL NOR gate.
- In this circuit, emitters of both the transistors are connected to a common ground and collectors of both transistors are tied through a common collector resistor R_C to a supply voltage V_{CC}.
- The resistor R_C is commonly known as passive pull-up resistor.
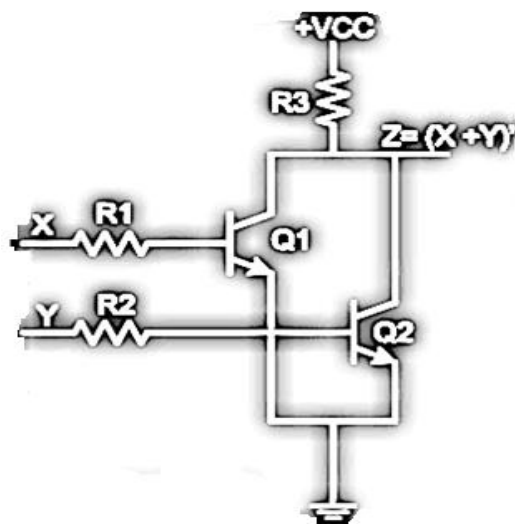
Figure 1.5: 2-input RTL NOR gate

Circuit operation:
- Inputs representing logic levels are applied at A and B terminals.
- When both the inputs are low, transistors Q1 and Q2 are cut-off and the output is HIGH.
- A HIGH level on any input devices the corresponding transistor to saturation causing the output to go LOW. The below table shows the truth table for 2-input NOR gate.

INPUT A	INPUT B	INPUT C
0	0	1
0	1	0
1	0	0
1	1	0

Truth table for 2-input NOR gate

- RTL gates the LOW level output voltage is 0.2 V. In RTL a HIGH level output voltage depends on the number of gates connected to the Output.
- As number of gates connected to the output increases, output voltage decreases.

Specifications:
- The RTL gates have poor noise margin, poor fan-out capabilities, low speed and high power dissipation.

Parameter	Value
Propagation Delay	12nsec
Power Dissipation	30-100 mW
Noise Margin	0.2 V
Fan-Out	4

4. **Write a note on DTL family. Explain the operation of two input DTL NAND gate.** **(Dec-17, May-18)**

Diode – Transistor Logic (DTL):
- The DTL is more complex than RTL but because of its greater fan-out and improved noise margins it has replaced RTL.

DTL Circuit:
- Figure 1.6 shows the discrete circuit for DTL NAND gate. It consists of input diodes and resistor R_D forming an AND gate and following them in transistor inverter.
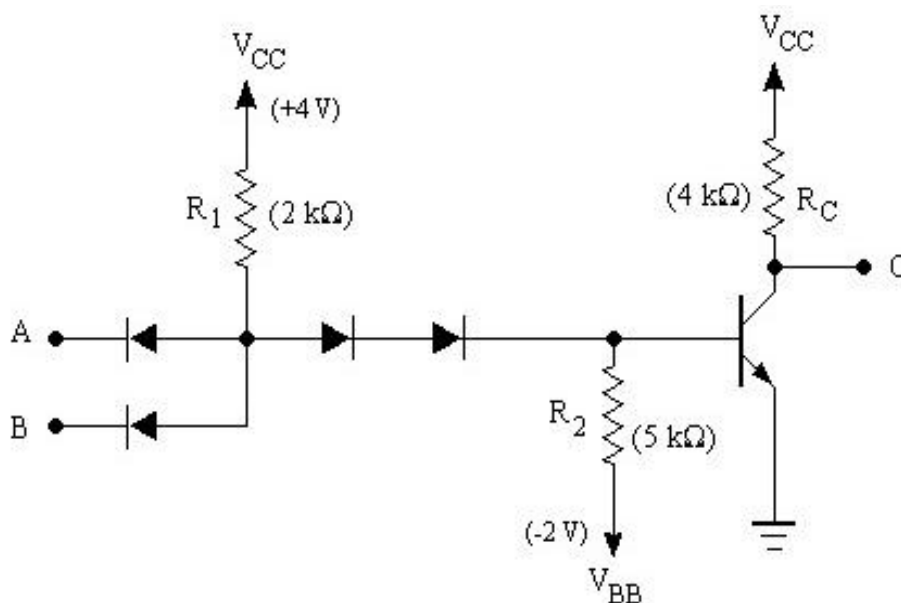
Figure 1.6: 2-input DTL NAND gate

48

Circuit operation:

- When both inputs are LOW, diode D_A and D_B conduct resulting 0.7 volts at point P. This 0.7 voltage at point is not sufficient to drive transistor Q_1.
- Therefore, Q_1 is cut off giving output voltage $V_0 = V_{CC}$ logic 1. A LOW level on any input cause corresponding diode to conduct resulting voltage at point P =0.7 V.
- This causes transistor to remain in cut-off and the output voltage is equal V_{CC} = logic 1.
- When both inputs are logic high, diodes D_A and D_B are reversed biased. This causes the base of the Q_1 to flow through R_D, D_1, D_2 and base of transistor Q_1.
- This drives the transistor Q_1 in saturation giving output voltage $V_{CE(sat)}$ = 0.2V = logic 0.
- When A and B inputs are HIGH, transistor Q_1 is driven to saturation and its base to emitter junction capacitance is charged.
- Now if any of the input goes low, voltage at the point B becomes 0.7V and transistor Q_1 will try to come out of saturation.
- To drive transistor from saturation to cut-off region it is necessary to discharge the stored charge on the internal capacitance.
- The resistance, R_B is connected to the -2V supply to increase the rate of discharge.

INPUT A	INPUT B	INPUT C
0	0	1
0	1	1
1	0	1
1	1	0

Truth table of 2-input DTL NAND gate

Specifications:

Parameter	Value
Propagation Delay	30nsec
Power Dissipation	60 W
Noise Margin	0.7 V
Pan-Out	8

- Thus we can say that the DTL has the advantage of greater fan-out and improved noise margins, but it suffers from somewhat lower speed.

Modified integrated DTL NAND gate:

- To increase the fan-out of DTL gate the base current of T_1 has to be increased. This can be done by replacing diode D_1 by the transistor, as shown in the figure 1.7.
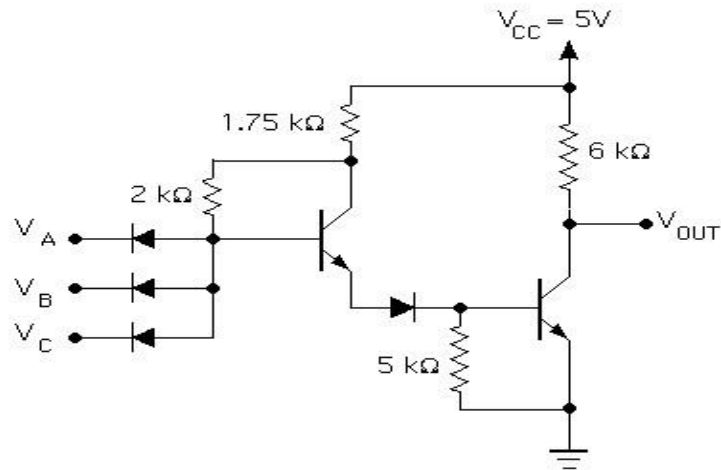
Figure 1.7: modified 3-input DTL NAND gate

5. Write short note on TTL family. Explain concept, operation and characteristics of TTL family. (Dec-07,10, 17)

Transistor TransistorLogic (TTL):

* TTL is named for its dependence on transistors alone to perform basic logic operations.

Draw the circuit diagram and explain the working of TTL inverter. (Dec-04)

TTL Inverter:

* We have seen that when input voltage is LOW, the output voltage is HIGH and vice-versa. Therefore we can make a logic inverter from an npn transistor in the common emitter configuration as shown in the figure 1.8.

(a) Circuit diagram

(b) Transfer characteristics
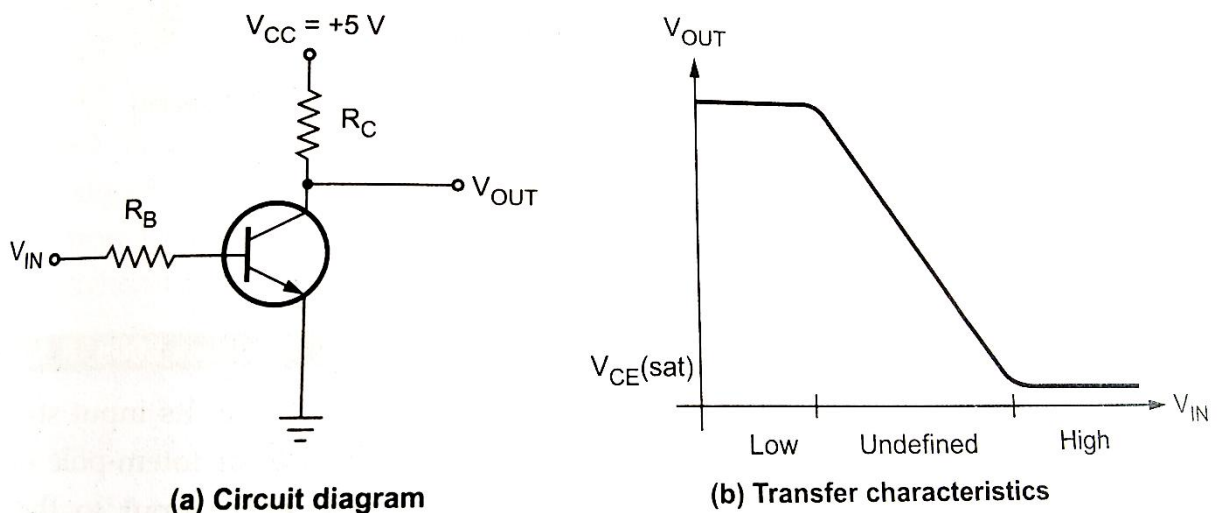
Figure 1.8: TTL inverter

The figures 1.9(a) and 1.9(b)Shows the operation of transistor inverter for both the inputs (HIGH and LOW) using switching analogy.

Figure: 1.9 (a)

Transistor cut-off

Switch OFF (open)
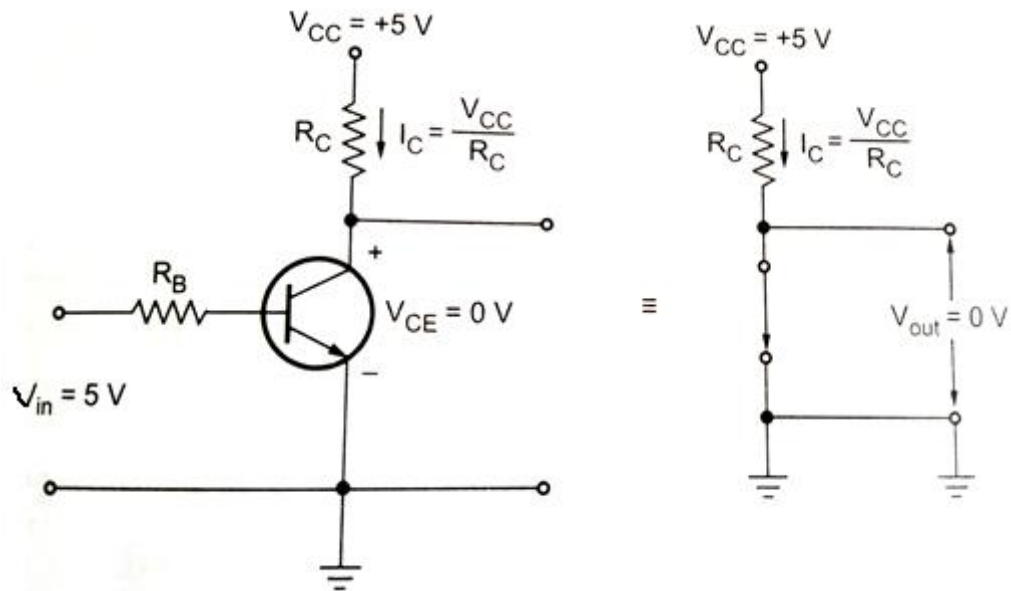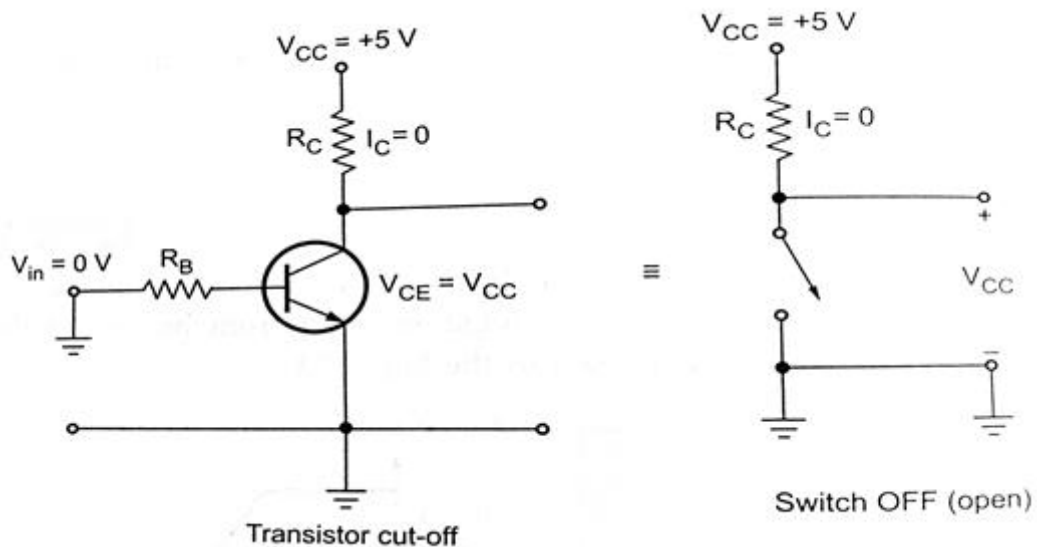
Figure: 1.9 (b)

6. Explain the working of 2-input TTL NAND gate. (Dec-05, May-08,17)

2-Input TTL NAND gates:
- The figure 1.10 Shows the circuit diagram of 2-input NAND gate.
- Its input structure consists of multiple-emitter transistor and output structure consists of totem-pole output.
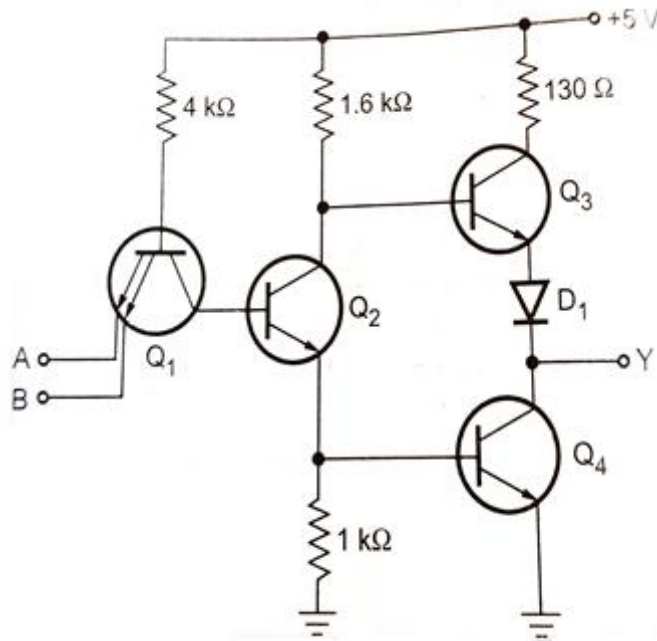- Here Q1 is the NPN transistor having two emitters, one for each input to the gate.

Figure 1.10:2-input TTL NAND gate

- Although this circuit looks complex, we can simplify its analysis using the diode equivalent of the multiple-emitter transistor Q1, as shown in figure 1.11.
- Diodes D2 and D3 represent the two E-B junction of Q1 and D4 is the collector – base (C-B) junction.
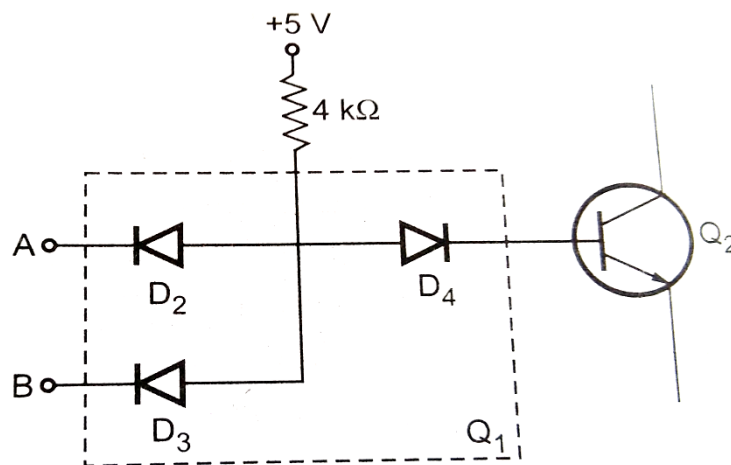
Figure 1.11: Diode equivalent for Q1.

- The input voltages A and B are either LOW (ideally grounded) or HIGH (ideally +5 volts).
- If either both A and B are low, the corresponding diode conducts and the base of Q1 is pulled down to approximately 0.7V.
- This reduces the base voltage of Q2 almost to zero. Therefore Q2 cuts off. With Q2 open, Q4 goes into cut-off and the Q3 base is pulled HIGH.
- Since Q3 acts as an emitter follower, the Y output is pulled up to a HIGH voltage.

- On the other hand, when both A and B are HIGH, the emitter diode of the Q1 are reversed biased making them off.
- This causes the collector diode D4 to go into forward condition. This forces Q2 base to go HIGH.
- In turn, Q4 goes into saturation, producing a low output. The following table summarizes all input and output conditions.

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Truth table for 2-input NAND gate

- Without diode D1 in the circuit, Q3 will conduct slightly when the output is low.
- To prevent this, the diode is inserted; its voltage drop keeps the base-emitter diode of Q3 reverse-biased. In this way, only Q4 conducts when the output is low.

7. Explain the working of 2-input TTL NAND gate. (Dec-05)

3-input TTL NAND gate:
- The figure 1.12 shows the three input TTL NAND gate. It is same as that of two input TTL NAND gate except that it's Q1 (NPN) transistor has three emitters instead of two. Rest of the circuit is same.
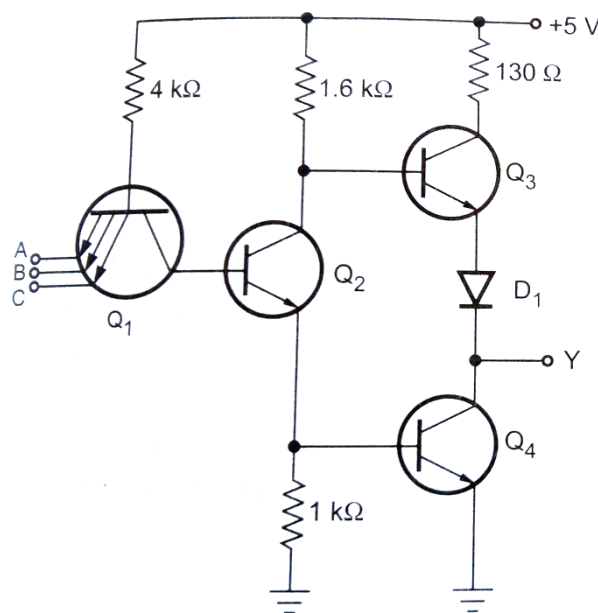
Figure 1.12: Three input TTL NAND gate

- For three input NAND gate if all the input are logic 1 then and then only output is logic 0; otherwise output is logic 1.
- The operation is similar to the 2-input NAND gate. The table below shows the truth table for 3-input NAND gate.

A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Truth table of 3-input NAND gate

8. Explain the totem-pole output stage in TTL with circuit diagram. (Dec-2018)

Totem-pole output:
- Transistor Q3 and Q4 form a totem-pole. Such a configuration is known as active pull-up or totem pole output.
- Totem-pole transistors are used because they produce LOW output impedance.
- Either way, the output impedance is low. This means that the output voltage can be change quickly from one state to the other because any stray output capacitance is rapidly charged or discharged through the low output impedance.
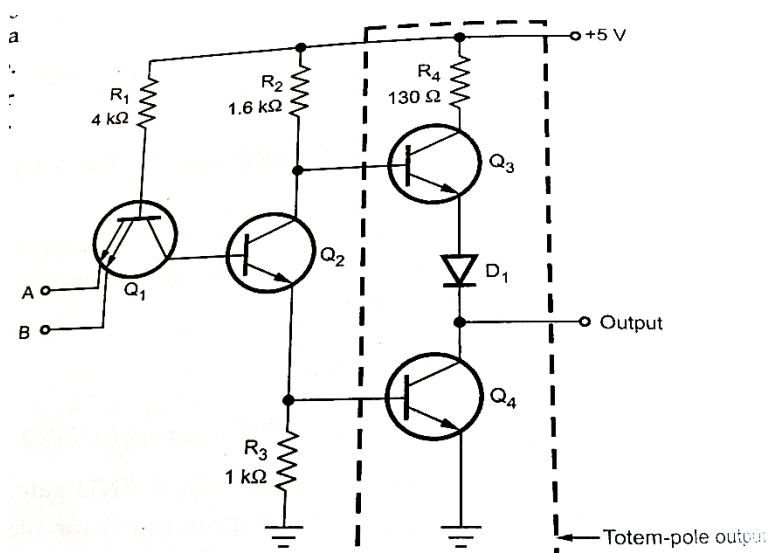- Thus the propagation delay is low in totem-pole TTL logic.

Figure 1.13: Two input NAND gate with totem-pole output.

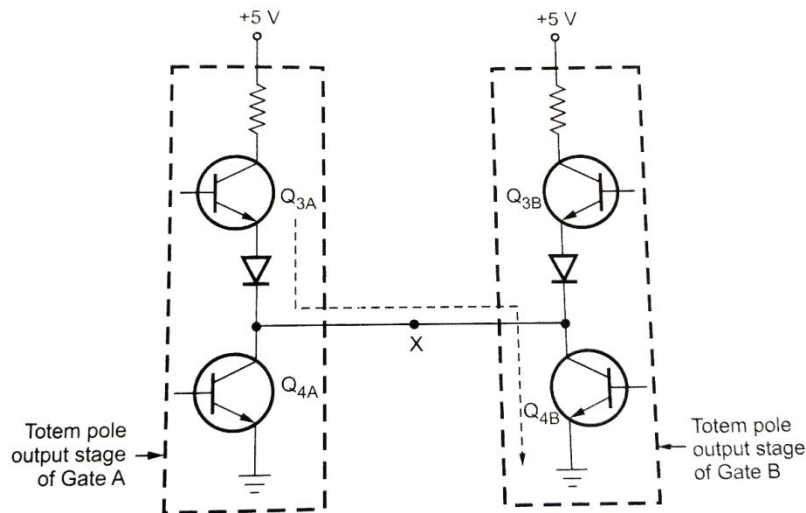- One problem with totem pole output is that two outputs cannot be tied together. The figure 1.14 Shows this problem.

Figure 1.14: Totem-pole outputs tied together can produce harmful current

Open-collector output:
- Some TTL devices provide another type of output called open collector output.
- The output of two different gates with open collector output can be tied together. This is known as wired logic.
- Figure 1.15 shows a 2-input NAND gate with an open-collector output eliminates the pull-up transistor Q3, D1 and R4.
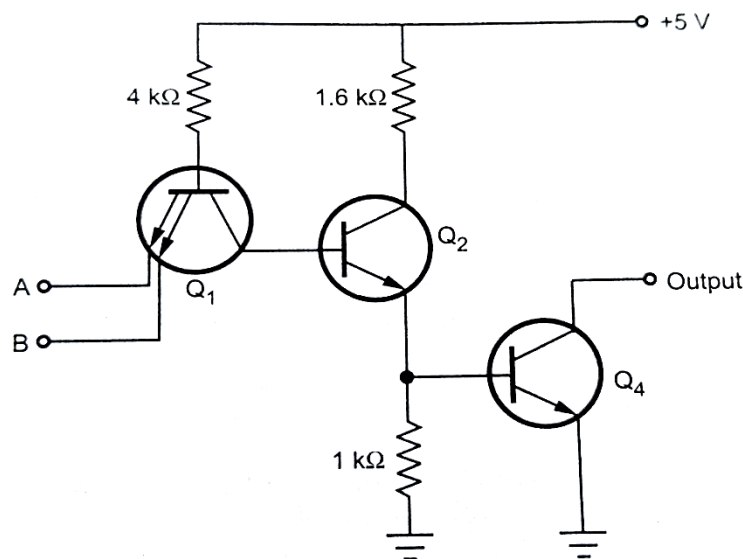- The output is taken from the open collector terminal of transistor Q4.

Figure 1.15: Open collector 2-input TTL NAND gate

- Because the collector of Q4 is open, a gate like this will not work properly until you connect an external pull-up resistor, as shown in figure 1.16.
- When Q4 is ON, output is low and when Q4 is OFF output is tied to V_{CC} through an external pull up resistor.
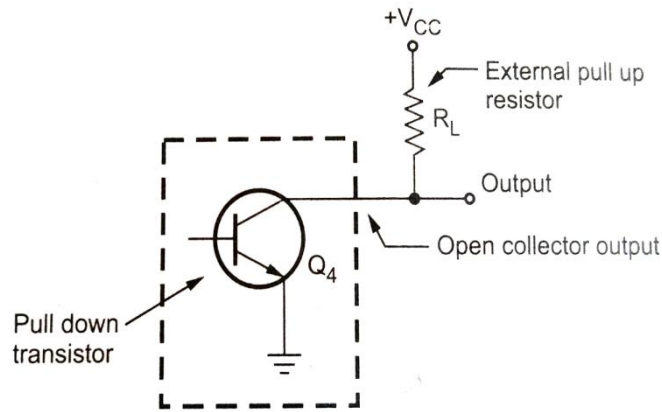
Figure 1.16: Open collector output with pull-up resistor

- As mentioned earlier, the open collector outputs of two or more gates can be connected together, as shown in the figure 1.17 (a).
- The connection is called a wired-AND and represented schematically by the special AND gate symbol as shown in figure 1.17 (b).
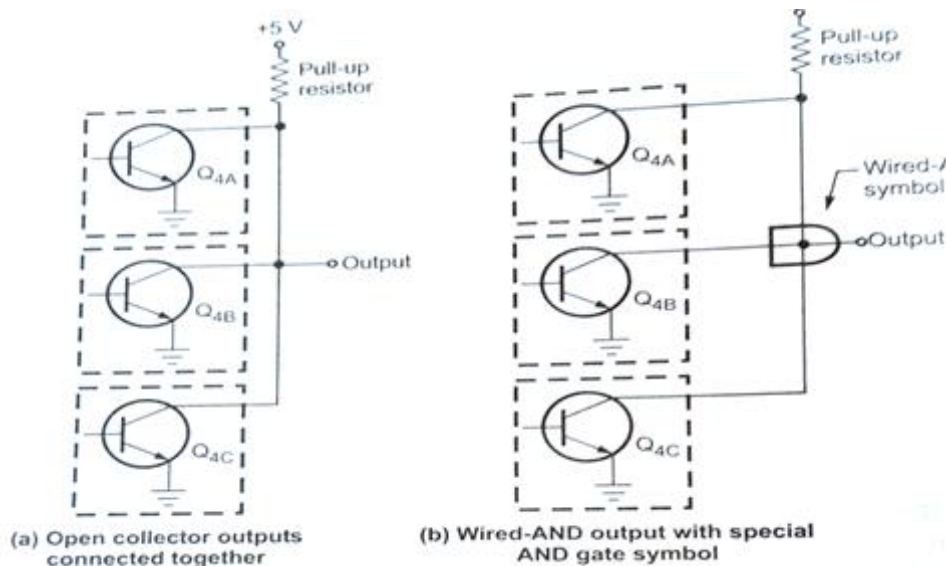

(a) Open collector outputs connected together

(b) Wired-AND output with special AND gate symbol

Figure 1.17

9. Give the difference between Totem-Pole and Open-Collector outputs. (May-15,17)

Comparison between Totem-Pole and Open-Collector outputs:

S.NO	Totem-pole	Open collector
1.	Output stage consists of pull-up Transistor(Q3),diode resistor and pull down transistor(Q4).	Output stage consists of only pull down transistor.
2.	External Pull-Up Resistor is not Required.	External Pull-Up Resistor is Required for proper operation of gate.
3.	Output of two gates cannot be tied together.	Output of two gate can be tied together using wired AND technique.
4.	Operating speed is high.	Operating speed is Low.

10. Explain the working of TTL inverter with tristate output. (May-09)

Tri-State TTL Inverter:

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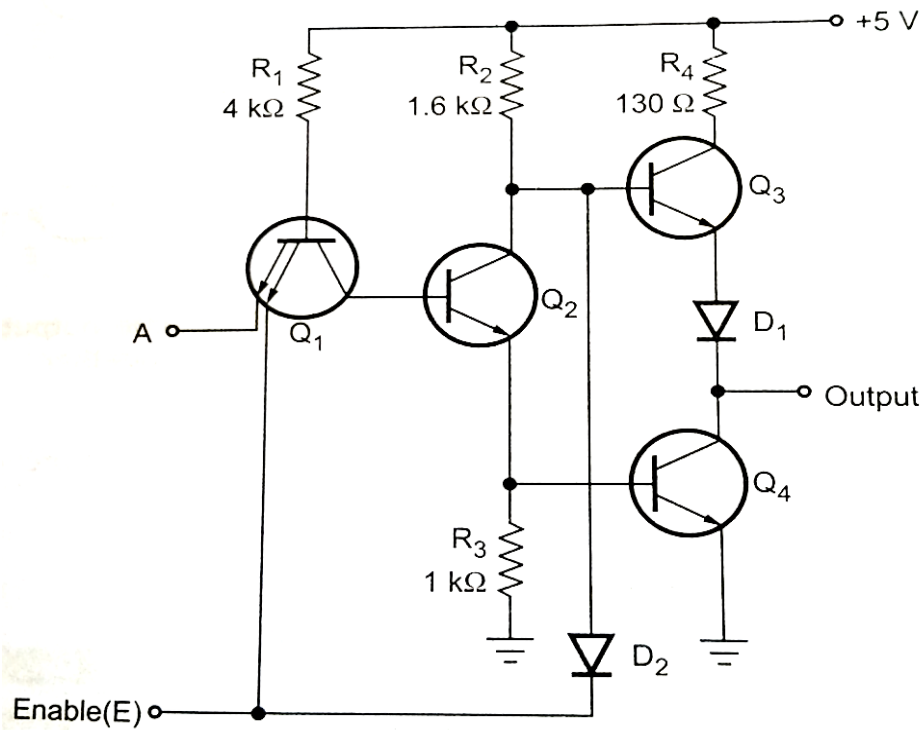- As a result, the output is open or floating, it is neither LOW nor HIGH.

Figure 1.18: Tristate TTL inverter

- A is the normal logic input whereas E is an ENABLE input. When ENABLE input is HIGH, the circuit works as a normal inverter.
- When ENABLE input is LOW, both transistors are OFF and output is at high impedance state.
- When ENABLE input is HIGH, ENABLE input is active high. In some circuits ENABLE input can be active LOW, i.e. circuit operates when ENABLE input LOW.

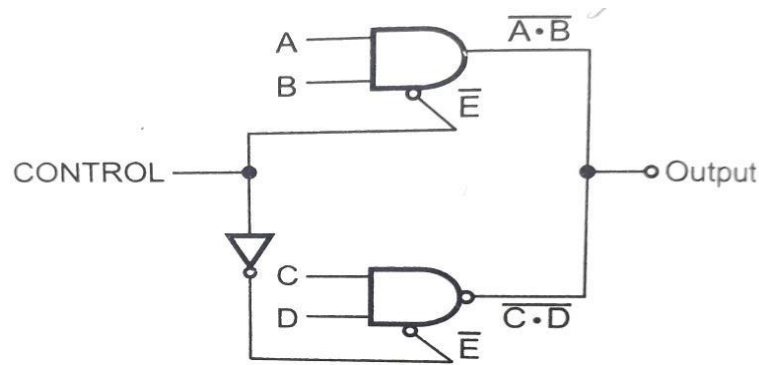Tristate outputs can be connected together as shown in the figure 1.19.

Figure 1.19: Tristate outputs connected together.

11. Name and explain the characteristics of TTL family.(Dec-04, 10) (May-06)

Characteristics of TTL Family:

- There are several series/ subfamilies in the TTL family of logic devices. Let us see the characteristics of standard TTL family.

Supply voltage and temperature range:

- Both the 74 series and 54 series operate on supply voltage of 5V.
- The 74 series works reliably over the range 4.75V to 5.25V, while the 54 series can tolerate a supply variation of 4.5 to 5.5 V.
- The 74 series devices are guaranteed to work reliably over a temperature range of 0to 70°C where as 54 series devices can handle temperature variations fromm-55 to +125°C.
- From the above values we can say that 54 series devices have greater tolerance of voltage and temperature variations.
- Hence, these devices are used where it is necessary to maintain reliable operation over an extreme range of conditions.
- For example, in military and space application. The only disadvantage of these devices is that they are expensive.

Voltage levels and noise margin:

- Table below shows the input and output logic voltage levels for the standard 74 series.
- The minimum and maximum values shown in the table are for worst case conditions of power supply, temperature and loading conditions.

Voltages	Minimum	Typical	Maximum
V_{OL}	-	0.2	0.4
V_{OH}	2.4	3.4	-
V_{IL}	-	-	0.8
V_{IH}	2.0	-	-

- Looking at the Table. We can say that, in the worst case, there is difference of 0.4V between the driver output voltages and the required load input voltages. For instance, the worst-case low values are

$$V_{OL(max)} = 0.4 \text{ V driver output}$$
$$V_{IL(max)} = 0.8 \text{ V load input}$$

- Similarly, the worst-case high values are

$$V_{OH(min)} = 2.4 \text{ V driver output}$$
$$V_{IH(max)} = 2 \text{ V load input}$$

- In either case, the difference is 0.4 V. This difference is called noise margin. For TTL, low state noise margin, V_{NL} and high state noise margin, V_{NH} both are equal and 0.4 V. This is illustrated in figure 1.20.
- It provides built-in protection against noise. It ensures reliable operation of the device for induced noise voltages less than 0.4 V.
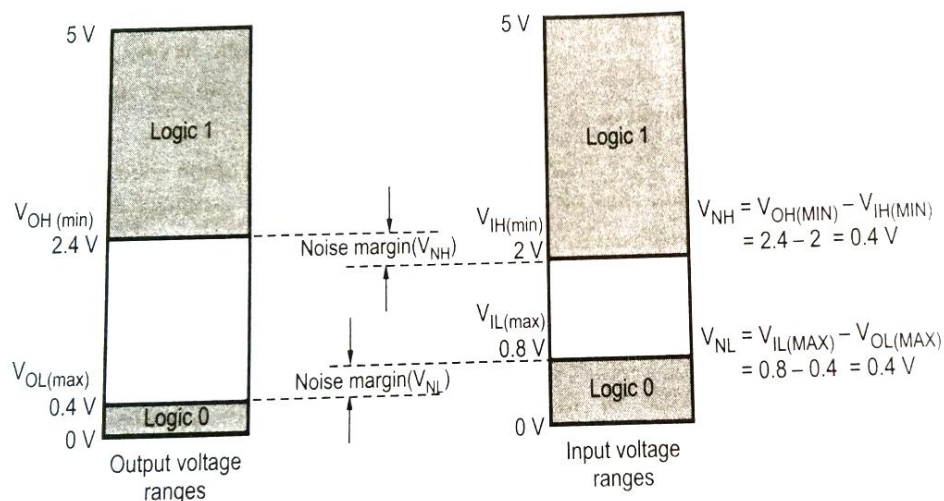

Figure 1.20: TTL logic levels and noise margin

Power dissipation and propagation delay:
- A standard TTL gate has an average power dissipation of about 10 mW. It may vary from this value because of signal levels, tolerances etc.
- We know that, the propagation delay time is the time it takes for the output of a gate to change after the inputs have changed.
- The propagation delay time of a TTL gate is approximately 10 nanoseconds.

Fan-out:
- A standard TTL output can typically drive 10 standard TTL inputs. Therefore, standard TTL has fan-out 10.

S.NO	Characteristics	Values
1.	Supply Voltage	For 74 series-(4.75 to 5.25) units For 54 series-(4.75 to 5.5)units
2.	Temperature	For 74 series-(0 to 70'C) units For 54 series-(-55 to 125'C)units
3.	Voltage Level	V_{OL}(max) – 0.4V V_{OH}(max) – 2.4V V_{IH}(max) – 0.8V V_{IH}(max) – 2.0V
4.	Noise Margin	0.4
5.	Power Dissipation	10mW per gate
6.	Propagation Delay	Typically 10 ns
7.	Fan-out	10

Standard TTL characteristics

12. Draw and explain the NOR gate using TTL logic.   (Dec-11)

TTL NOR GATE:

- The figure 1.21 Shows the circuit diagram for an LS-TTL two-input NOR gate (74LS02). The circuit is basically divided into three functional parts.
  - ➢ Input circuits.
  - ➢ Phase splitter.
  - ➢ Output stage.

- It is almost identical to those of an LS-TTL NAND gate.
- The difference is that an LS-TTL NAND gate uses diode to perform the AND function, while an LS-TTL NOR gate uses parallel transistors in the phase splitter to perform the OR function.
- If either input A and B is HIGH, the corresponding phase splitter transistor Q_{2A} or Q_{2B} is turned on, which turns off Q_3 and Q_4 while turning on Q_5 and Q_6 and the output is LOW.
- If both the inputs are LOW, then both phase-splitter transistors are OFF and the output is forced HIGH. This functional operation is summarized in functional table.
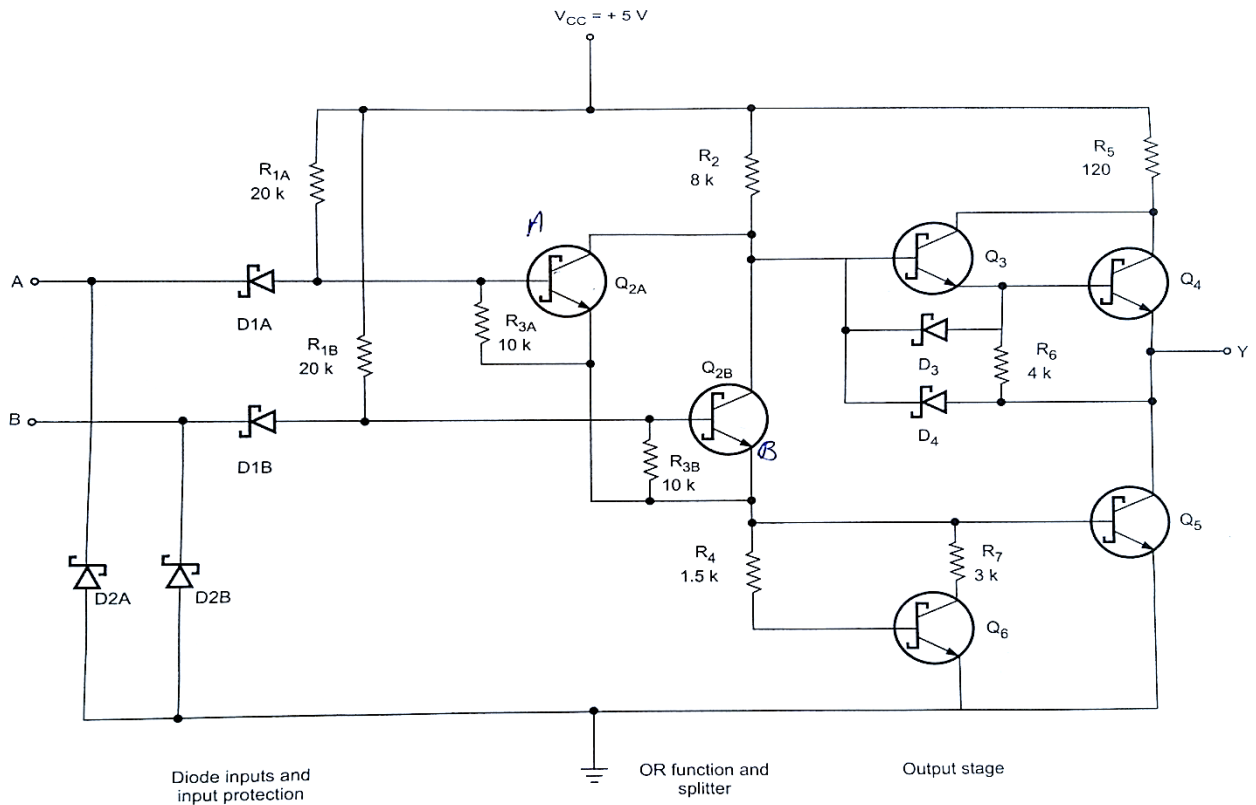
Figure 1.21: Circuit diagram of a two-input LS-TTL NOR gate

A	B	Q2A	Q2B	Q3	Q4	Q5	Q6	Y
0	0	OFF	OFF	ON	ON	OFF	OFF	1
0	1	OFF	ON	OFF	OFF	ON	ON	0
1	0	ON	OFF	OFF	OFF	ON	ON	0
1	1	ON	ON	OFF	OFF	ON	ON	0

(a) Functional table

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

(b) Truth table

Advantages of TTL:
- ➢ High speed operation. Fastest among the saturated logic families. The propagation delay time is about 10ns.
- ➢ Moderated power dissipation.
- ➢ Available in commercial and military versions.
- ➢ Available for wide range of functions..

61

- Low cost.
- Moderate packaging density.

Disadvantages of TTL:
- Higher power dissipation than CMOS.
- Lower noise immunity than CMOS.
- Less fan-out than CMOS.

MOS Families:(Dec 07,10)
Digital circuits with MOSFETs can be grouped into three categories:
- PMOS – uses only P-channel enhancement MOSFETs,
- NMOS – uses only N-channel enhancement MOSFETs and
- CMOS (complementary MOS) – uses both P and N-channel devices.

13.   Explain the CMOS inverter.

NMOS Inverter:
- Figure 1.22 shows the basic NMOS inverter circuit. It consist two N-channel MOSFETs. Q2 is a switching MOSFET and Q1 is a load MOSFET.
- Q1 acts as load resistance (Rd) for Q2. As gate of Q1 is permanently connected to the V_{DD}, it is always ON, and hence the load resistance is equal to the R_{ON} of the MOSFET.
- Particularly, Q1 is designed to have greater R_{ON} than the R_{ON} of Q2. To achieve this channel of Q1 is made much narrower than channel of Q2. Typically R_{ON} of Q1 is 100 K whereas R_{ON} of Q2 is 1 K.
- Know that MOS devices are voltage controlled devices. When positive voltage (HIGH input) is applied between gate and source, Q2 is switched ON and it makes the output low.
- On the other hand, when input is LOW Q2 is switched OFF and therefore, output is high.
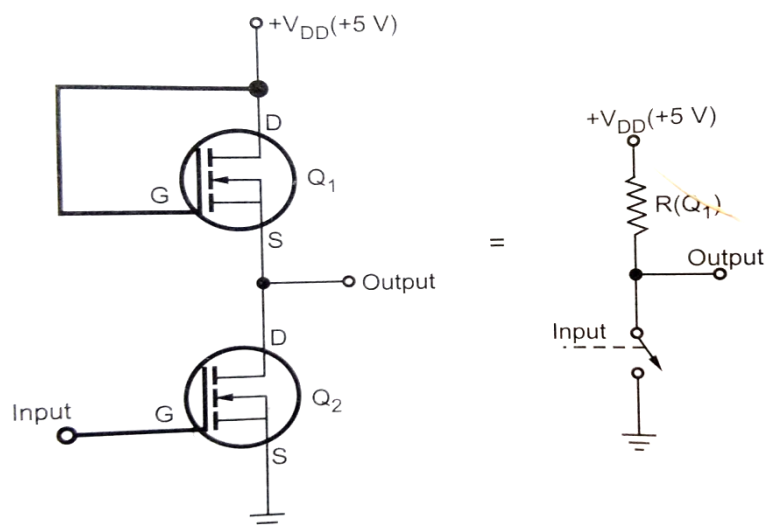

Figure 1.22: NMOS inverter circuit and its equivalent circuit

VIN (VGS)	Q2	$Vo = \overline{V_{IN}}$
0 V (logic 0)	OFF	+ 5 V (logic 1)
5 V (logic 1)	ON	0 V (logic 0)

Operation of NMOS inverter

14. Explain the operation of NMOS NAND Gate.

NMOS NAND Gate:
- Figure 1.23 shows 2-input NMOS NAND gate. Q1 acts as a load resistor and Q2 and Q3 are the switching MOSFETs controlled by the inputs A and B.
- The equivalent switching circuit consist of resistor and two switches in series. If either A and B or both inputs are low, the corresponding MOSFETs are OFF i.e, the corresponding switches are open and the output is high.
- If A and B both inputs are high, the corresponding MOSFETs are ON i.e, the corresponding switches Are closed and the output is low.
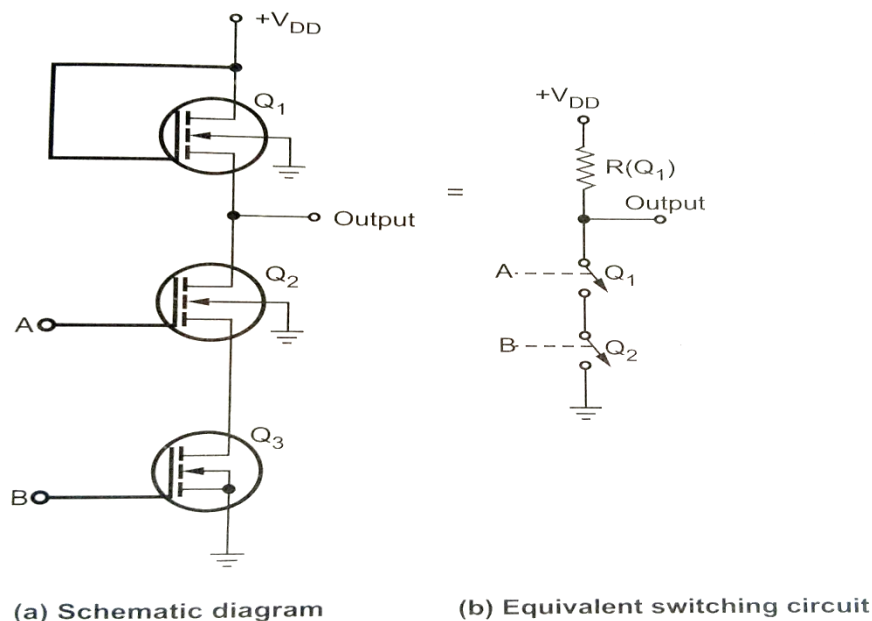

(a) Schematic diagram   (b) Equivalent switching circuit
Figure 1.23: 2-input NMOS NAND gate

A	B	Q2	Q3	$Vo = \overline{AB}$
0	0	OFF	OFF	1
0	1	OFF	ON	1
1	0	ON	OFF	1
1	1	ON	ON	0

Operation of NMOS NAND gate

15. Explain the operation of NMOS NOR Gate.

NMOS NOR Gate:
- Figure 1.24 shows 2-input NMOS NOR gate. Q1 acts as a load resistor and Q2 and Q3 are the switching MOSFETs controlled by the inputs A and B.
- The equivalent switching circuit consist of a resistor and two switches connected in parallel.
- When either or both input are high, the corresponding MOSFETs are ON i.e, corresponding switches are closed making the output low.
- If both inputs are low, both MOSFETs are OFF i.e, both switches are open and the output is high.

(a)

(b)
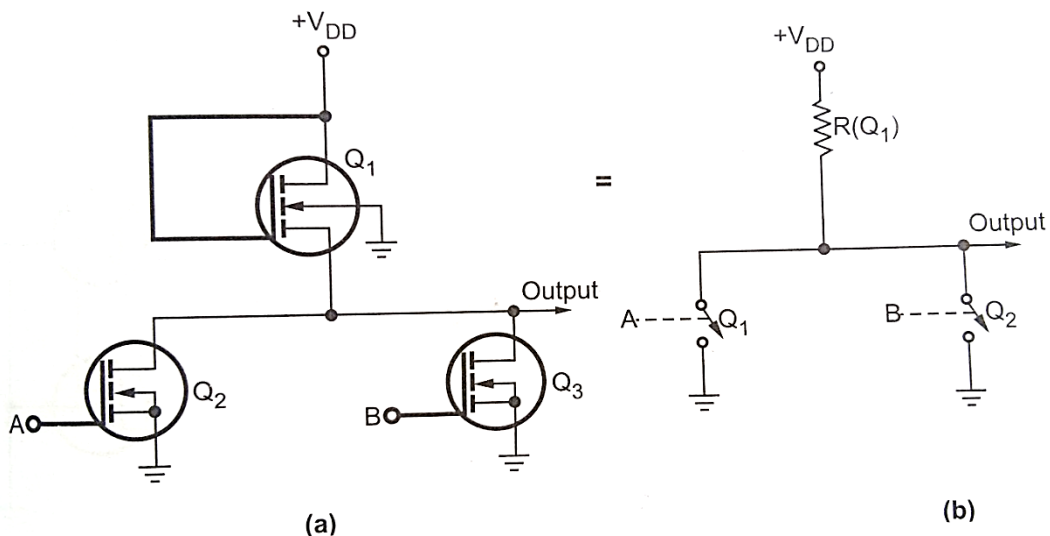
Figure 1.24:2-input NMOS NOR gate

A	B	Q2	Q3	Vo = $\overline{A+B}$
0	0	OFF	OFF	1
0	1	OFF	ON	0
1	0	ON	OFF	0
1	1	ON	ON	0

Operation of NMOS NOR gate

Characteristics of NMOS:

Operating speed:
- Low operating speed with propagation delay time around 50 ns.
- This is because it has high output resistance, very high input resistance and reasonably high input capacitance.

Noise margin: Typically 1.5 V.

Fan out: Typically 30.

Power drain: less, around 0.1 mW per gate.

16. Write short note on PMOS logic.

PMOS Logic:
- The figure 1.25 Shows PMOS inverter, NAND gate and NOR gate.
- For p-channel, enhancement-type MOSFET a negative voltage is needed at the gate terminal to form a channel.
- According to the positive logic, logic 0 is approximately $-V_{DD} < -V_T$, which is the low-voltage signal value, which logic 1 is approximately ground, which is the high voltage signal value.
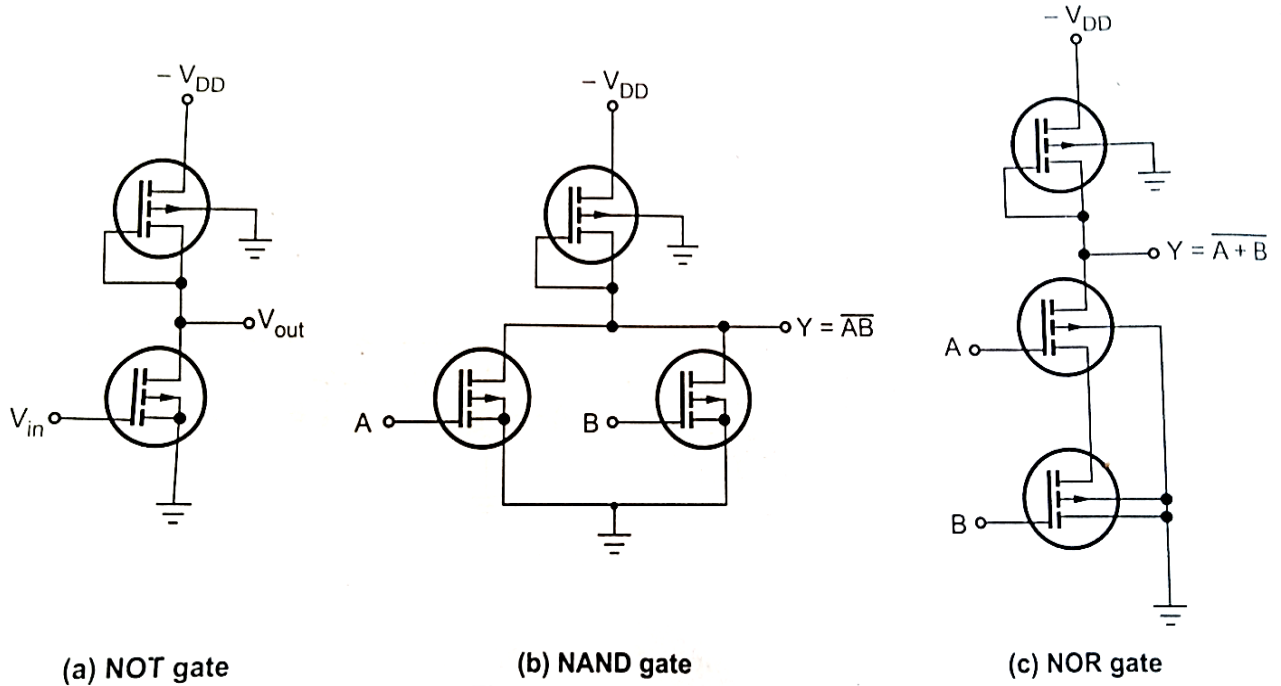

(a) NOT gate   (b) NAND gate   (c) NOR gate

Figure 1.25: PMOS logic gates

VIN (VGS)	Q2	Vo = $\overline{V_{IN}}$
0 V (logic 0)	OFF	+ 5 V (logic 1)
5 V (logic 1)	ON	0 V (logic 0)

Operation of PMOS inverter

A	B	Q2	Q3	Vo = $\overline{AB}$
0	0	OFF	OFF	1
0	1	OFF	ON	1
1	0	ON	OFF	1
1	1	ON	ON	0

Operation of PMOS NAND gate

A	B	Q2	Q3	Vo = $\overline{A + B}$
0	0	OFF	OFF	1
0	1	OFF	ON	0
1	0	ON	OFF	0
1	1	ON	ON	0

Operation of PMOS NOR gate

17. Explain the operation of CMOS Inverter. (May-04)

CMOS Inverter:

- Figure 1.26 shows the basic CMOS inverter circuit.
- It consist of two MOSFETs in series in such a way that the p-channel device has its source connected to $+V_{DD}$ (a positive voltage) and the n-channel device has its source connected to ground.
- The gates of the two devices are connected together as the common input and the drains are connected together as the common output.
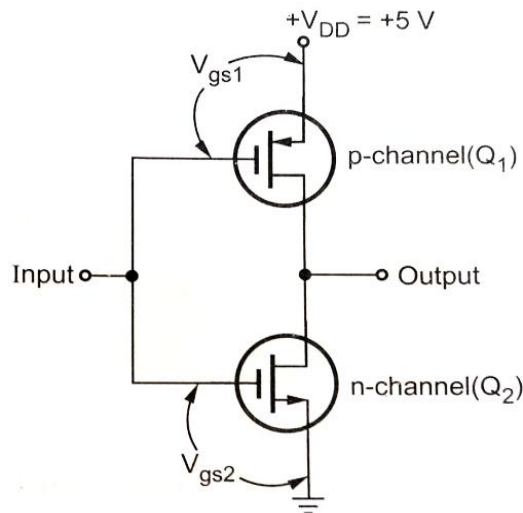
Figure 1.26: CMOS inverter circuit

- When input is HIGH, the gate of Q1 (p-channel) is at 0V relative to the source of Q1 i.e V_{gs1} = 0V. Thus, Q1 is OFF. On the other hand, the gate of Q2 (n-channel) is at $+V_{DD}$. Thus, Q2 is ON. This will produce V_{OUT} = 0V, a shown in figure 1.27 (a)

- When input is LOW, the gate of Q1(p-channel) is at a negative potential relative to its source while Q2 is ON and Q2 is OFF. This produces output voltage approximately +VDD, as shown in the figure1.27 (b).
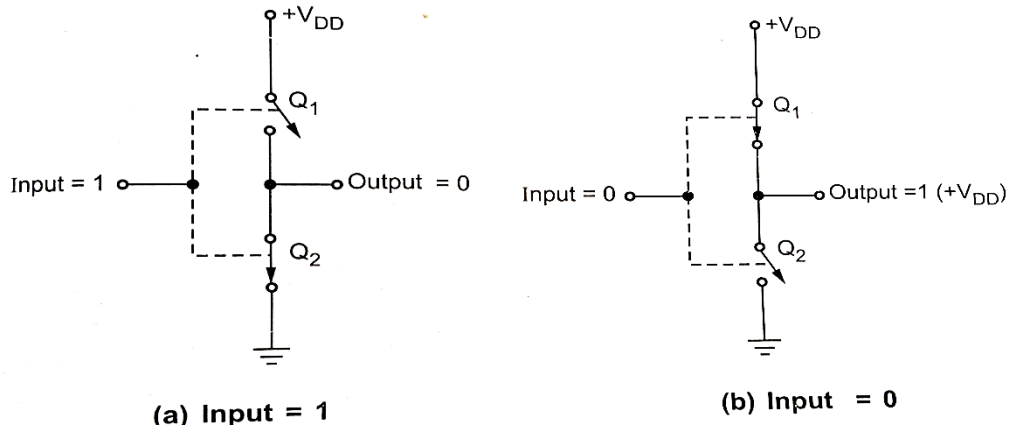
(a) Input = 1   (b) Input = 0

Figure 1.27: Operation of CMOS inverter for both input conditions

| A	Q1	Q2	Output
0	ON	OFF	1
1	OFF	ON	0

Operation of CMOS inverter circuit

18. Explain the operation of CMOS NAND Gate. (Dec-03, June-08)

CMOS NAND Gate:

- Figure 1.28 shows CMOS 2-input NAND gate.
- It consists of two p-channel MOSFETs, Q1 and Q2, connected in parallel and two n-channel MOSFETs, Q3 and Q4 connected in series.
- The equivalent switching circuit has both inputs are low. Here, the gates of both p-channel MOSFETs are negative with respect to their sources, since the sources are connected to $+V_{DD}$. Thus, Q1 and Q2 are both ON.

(a) Schematic

$A = B = 0\ V$
$V_{GS1} = V_{GS2} = -V_{DD}$
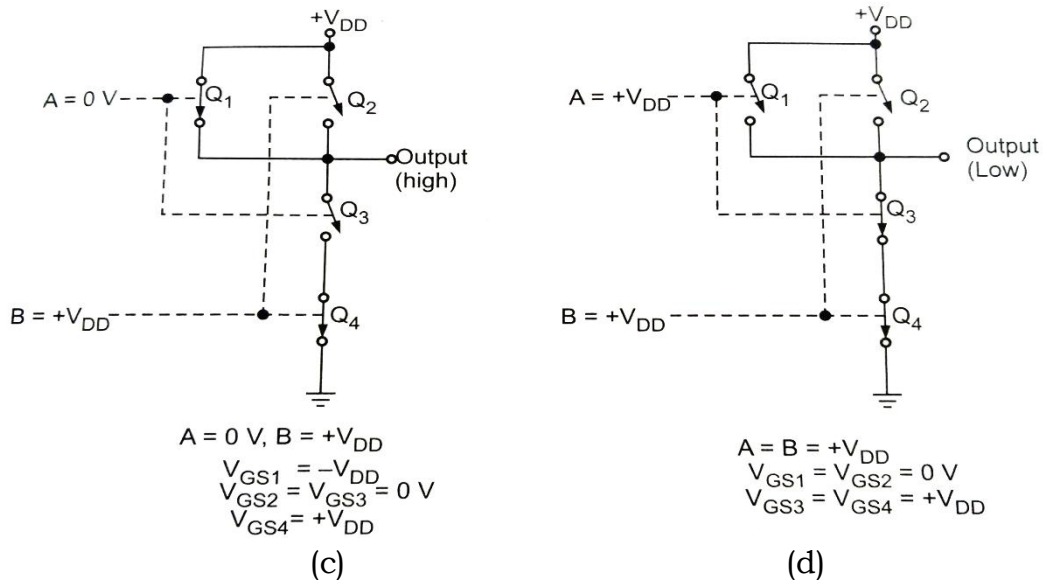$V_{GS3} = V_{GS4} = 0\ V$

(b)
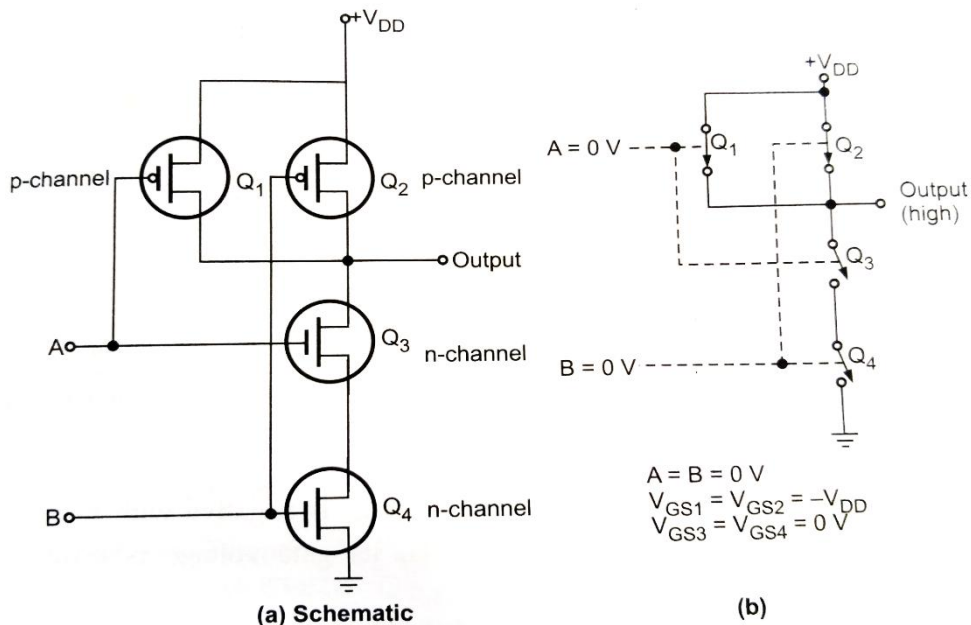
Figure 1.28: CMOS NAND gate

- Since the gate-to-source voltages of Q3 and Q4 (n-channel MOSFETs) are both 0V, those MOSFETs are Off. The output is therefore connected to $+V_{DD}$ (HIGH) through Q1 and Q2 and is disconnected from ground, as shown in the figure 1.28 (b).

- And figure 1.28(c) shows the equivalent switching circuit when A = 0 and B = $+V_{DD}$. In this case, Q1 is on because V_{GS1} = - V_{DD} and Q4 is ON because V_{GS4} = $+V_{DD}$.

- MOSFETs Q2 and Q3 are off because their gate-to-source voltages are 0V. Since Q1 is ON and Q3 is OFF, the output is connected to $+V_{DD}$ and it is disconnected from ground.

- When A = $+V_{DD}$ and B = 0V, the situation is similar (not shown); the output is connected to $+V_{DD}$ through Q2 and it is disconnected from ground because Q4 is OFF.

- Finally, when both the inputs are high(A= B = $+V_{DD}$), MOSFETs Q1 and Q2 are both OFF and Q3 and Q4 are both ON.

- Thus, the output is connected to the ground through Q3 and Q4 and it is disconnected from $+V_{DD}$. The table summarizes the operation of 2-input CMOs NAND gate.

A	B	Q1	Q2	Q3	Q4	Output
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

Truth table of NAND gate

68
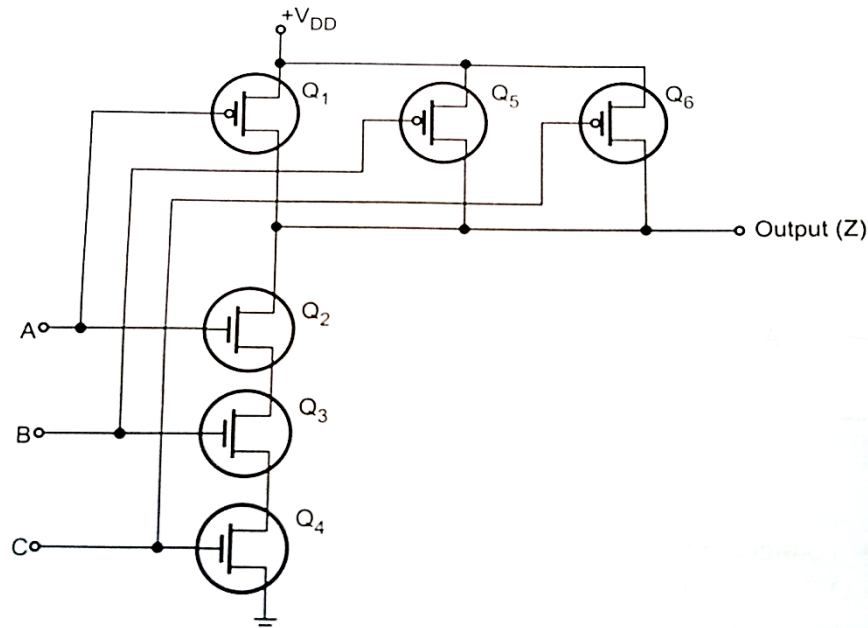
Figure 1.29: CMOS 3-input NAND gate

A	B	C	Q1	Q2	Q3	Q4	Q5	Q6	Z
0	0	0	ON	OFF	OFF	OFF	ON	ON	1
0	0	1	ON	OFF	OFF	ON	ON	OFF	1
0	1	0	ON	OFF	ON	OFF	OFF	ON	1
0	1	1	ON	OFF	ON	ON	OFF	OFF	1
1	0	0	OFF	ON	OFF	OFF	ON	ON	1
1	0	1	OFF	ON	OFF	ON	ON	OFF	1
1	1	0	OFF	ON	ON	OFF	OFF	ON	1
1	1	1	OFF	ON	ON	ON	OFF	OFF	0

Truth table of three input NAND gate

19. Explain the operation of CMOS NOR Gate.   (Dec-15)
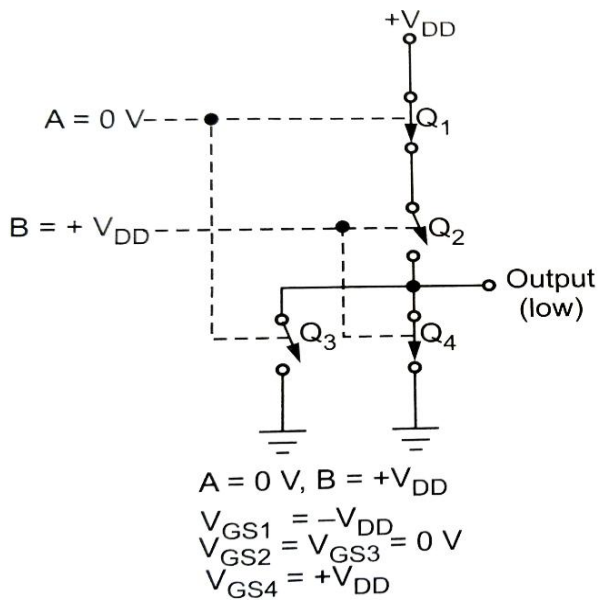
CMOS NOR Gate:

- Figure 1.30 shows 2-input CMOS NOR gate. Here, p-channel MOSFETs Q1 and Q2 are connected in series and n-channel MOSFETs Q3 and Q4 are connected in parallel.
- Like NAND circuit, this circuit can be analyzed by realizing that a LOW at any input turns ON its corresponding p-channel MOSFET and turns OFF its corresponding n-channel MOSFET, and vice-versa for high input.
- This is illustrated in figure 1.30. The table summarizes the operation of two input NOR gate.
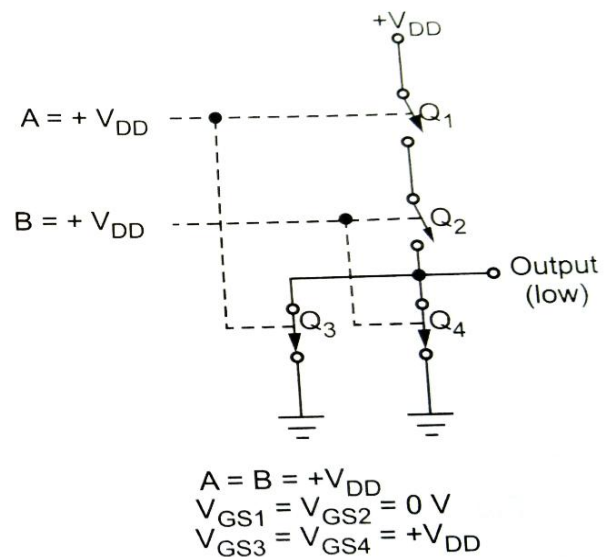
(a) Schematic

(b)

$A = B = 0\ V$
$V_{GS1} = V_{GS2} = -V_{DD}$
$V_{GS3} = V_{GS4} = 0\ V$

(c)

$A = 0\ V,\ B = +V_{DD}$
$V_{GS1} = -V_{DD}$
$V_{GS2} = V_{GS3} = 0\ V$
$V_{GS4} = +V_{DD}$

(d)
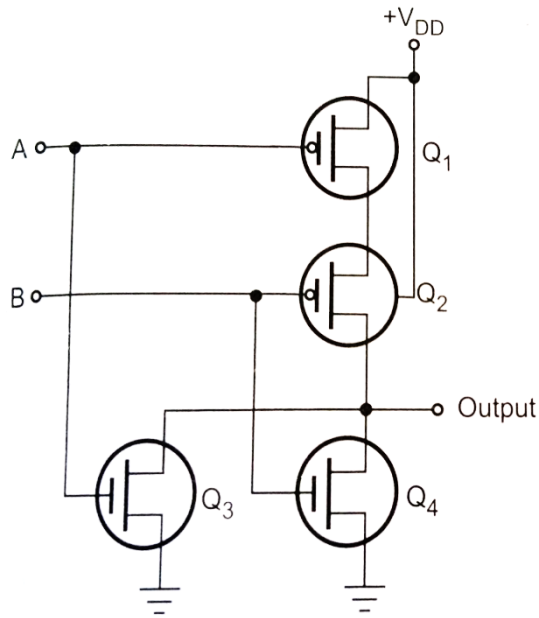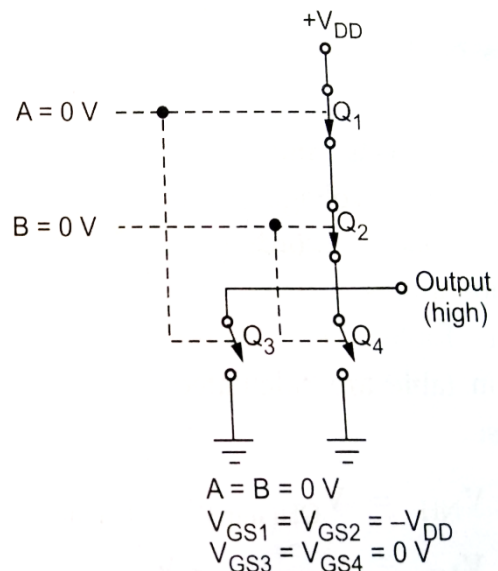
$A = B = +V_{DD}$
$V_{GS1} = V_{GS2} = 0\ V$
$V_{GS3} = V_{GS4} = +V_{DD}$

Figure 1.30: CMOS NOR gate

A	B	Q1	Q2	Q3	Q4	Output
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	0
1	0	OFF	ON	ON	OFF	0
1	1	OFF	OFF	ON	ON	0

Truth table for NOR gate

70

20. Explain the characteristics of CMOS family. (May-04, 07; Dec-06, 11)

Characteristics of CMOS Family:
Operating speed:
- Slower than TTL series. Approximately 25 to 100 ns depending on the subfamily of CMOS. It also depends on the power supply voltage.

Voltage levels and Noise Margins:
- The voltage level for CMOS varies according to their subfamilies. These are listed in table.

Parameter	CMOS series				
	4000 B	74 HC	74 HCT	74 AC	74 ACT
$V_{IH(min)}$	3.5	3.5	2.0	3.5	2.0
$V_{IL(max)}$	1.5	1.0	0.8	1.5	0.8
$V_{OH(min)}$	4.95	4.9	4.9	4.9	4.9
$V_{OL(max)}$	0..5	0.1	0.1	0.1	0.1
V_{NH}	1.45	1.4	2.9	1.4	2.9
V_{NL}	1.45	0.9	0.7	1.4	0.7

- Noise margins in table are calculated as follows.

$V_{NH} = V_{OH(min)} - V_{IH(min)}$
$V_{NL} = V_{IL(max)} - V_{OL(max)}$

Fan-out:
- The CMOS inputs have an extremely large resistance (10^{12}) that draws essentially no current from signal source, each CMOS input, however, typically presents a 5 pF load to ground as shown in figure 1.31
- This input capacitance limits the number of CMOS inputs that one CMOS output can drive.
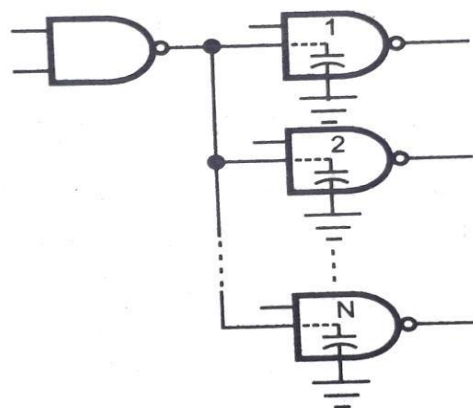

Figure 1.31: One CMOS output driving several CMOS inputs

- The fan-out for CMOS depends on the permissible maximum propagation delay.
- Typically, CMOS outputs are limited to a fan-out of 50 for low-frequency operation ($\leq$ 1MHz). Of course, for high-frequency operation the fan-out would have to be less.

Power Dissipation (P_D):
- The power dissipation of a CMOS IC is very low as long as it is in a d.c. condition. Unfortunately, power dissipation of CMOS IC increases in proportion to the frequency at which the circuits are switching states.
- When CMOS output switches from LOW to HIGH, a transient charging current has to be supplied to the load capacitance.
- Therefore, as the switching frequency increases, the average current drawn from VDD also increases, resulting increase in power dissipation.

Propagation Delay:
- The propagation delay in CMOS is the sum of delay due to internal capacitance and due to load capacitance.
- The delay due to internal capacitance is called the intrinsic propagation delay. The delay due to load capacitance can be approximated as follows,

$t_p(CL) = 0.5 \, R_0 \, C_L$ seconds

where $t_p(C_L)$ is either t_{pLH} or t_{pHL}.
R_0 is the output resistance of the gate and C_L is the total load capacitance.
The R_0 depends on the supply voltage and it can be approximated as

$$R_o \approx \frac{V_{cc}}{I_{os}}$$

Where I_{OS} is the short circuit output current.

Unused inputs:
- CMOS inputs should never be left disconnected. All CMOS inputs have to be tied either to a fixed voltage level (0 V or V_{DD}) or to another input.
- This rule applies even to the inputs of extra unused logic gates on a chip.
- An unused CMOS input is susceptible to noise and static charges that could easily bias both the P and N-channel MOSFETs in the conductive state, resulting in increased power dissipation and possible overheating.

Static-charge susceptibility (CMOS Hazards):
- The primary source of charge is "static" electricity, usually produced by handling and the motion of various kinds of plastics and textiles.
- The CMOS devices are protected against this static charge by on-chip diode-resistor network, as shown in figure 1.32
- These diodes are designed to turn ON and limit the size of the input voltage to well below any damaging value.
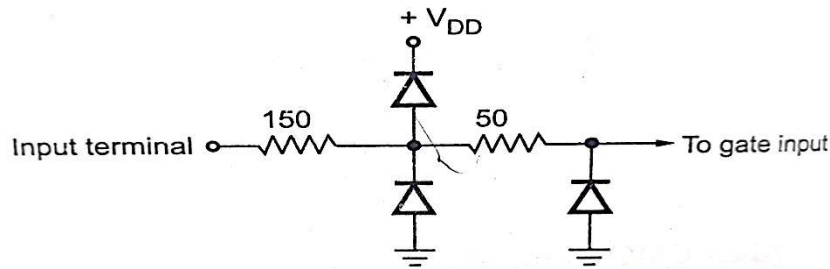
Figure 1.32: Typical network used to product CMOS from static charges.

Latch-up:
- CMOS integrated circuits contain parasitic PNP and NPN transistors.
- Their existence is not intentional but is unavoidable.
- Because of conducting paths between a pair of such transistors, a device can be triggered into a heavy conduction mode, known as latch-up.

Advantages of CMOS Family:
- ➢ Consumes less power.
- ➢ Can be operated at high voltages, resulting in improved noise immunity.
- ➢ Fan-out is more.
- ➢ Better noise margin.

Disadvantages of CMOS Family:
- ➢ Susceptible to static charge.
- ➢ Switching speed low.
- ➢ Greater propagation delay.

21. Write a short notes on ECL family. (May-08) (Dec-17)

ECL Family:
- Another logic family has been developed that prevents transistor saturation, thereby increasing overall switching speed by using a radically different circuit's structure, called Current Mode Logic (CML). This logic family is also called Emitter-Coupled Logic (ECL).
- Unlike TTL and CMOS families, ECL does not produce a large voltage swing between the LOW and HIGH levels.
- It has a small voltage swing, less than volt, and it internally switches current between two possible paths, depending on the output state.

Explain the characteristics of ECL family. (Dec-06)

Characteristics of ECL Family:

- It is the fastest of logic families. The popular 10K and 100K ECL families offer propagation delays as short as 1ns. The latest ECL family, ECL in PS (ECL in picoseconds), offers maximum delays under 0.5 ns (500 ps).

- Transistors are not allowed to go into complete saturation and thus eliminating storage delays.
- To prevent transistors from going into complete saturation, logic levels are kept close to each other. Due to this transistor is not driven into saturation when its input switches from low to high.
- As logic levels are kept close to each other, noise margin is reduced and it is difficult to achieve good noise immunity.
- Another disadvantage of this approach is that power consumption is more because transistors are not completely saturated.
- Switching transients are less because power supply current is more stable than in TTL and CMOS circuits.

Basic ECL circuit:
- The figure 1.33shows the basic inverter/buffer circuit in ECL family.
- It consists of two transistors connected in differential single ended input mode with a common emitter resistance.
- The circuit has two outputs: inverting output (OUT1) and non-inverting output (OUT2). For this circuit, the input LOW and HIGH voltage levels are defined as 3.6V and 4.4V, and it produces output LOW and HIGH levels as 4.2V and 5.0V.
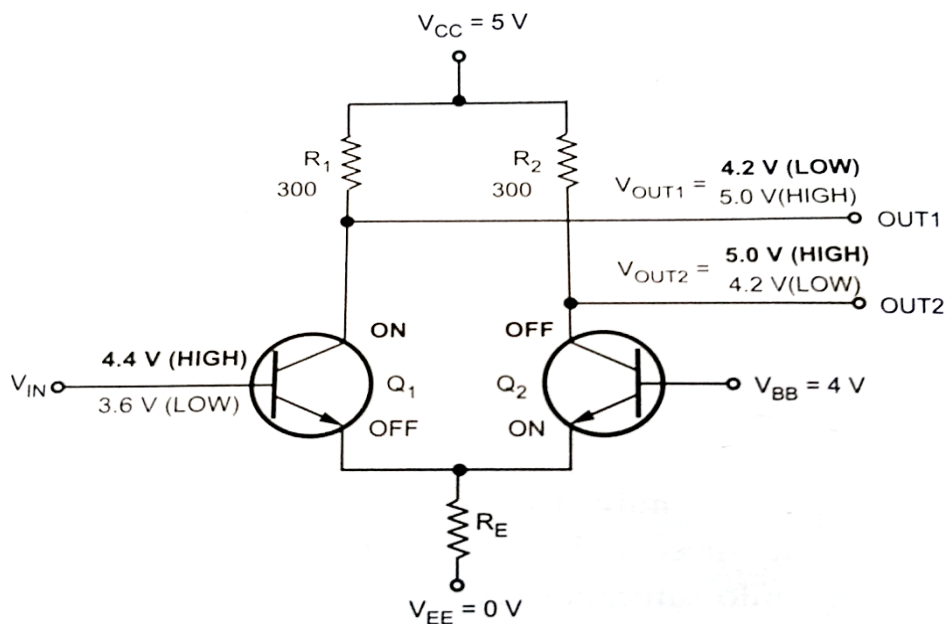

Figure 1.33: Basic ECL invertor/buffer circuit

- When V_{IN} is HIGH (4.4 V), transistor Q1 is ON, but not saturated and transistor Q2 is OFF. Thus, V_{OUT2} is pulled to 5.0 V (HIGH) through R2 and drop across R1 is 0.8 V so that V_{OUT1} Is 4.2 V(LOW).
- When V_{IN} is LOW (3.6V), transistor Q2 is ON, but not saturated and transistor Q1 is OFF. Thus, V_{OUT1} is pulled to 5.0 V (HIGH) through R1 and drop across R2 is 0.8 V so that V_{OUT2} is 4.2 V (LOW).

22. Explain ECL OR/NOR gate.

ECL OR / NOR Gate:

- Figure 1.34 shows, 2-input ECL OR/NOR gate and it logic symbol. There has an additional transistor in parallel with Q1 as compared to ECL inverter.
- If any input is HIGH corresponding transistor is active, and V_{OUT1} is LOW (NOR output). At the same time Q3 is OFF producing V_{OUT2} HIGH (OR output).
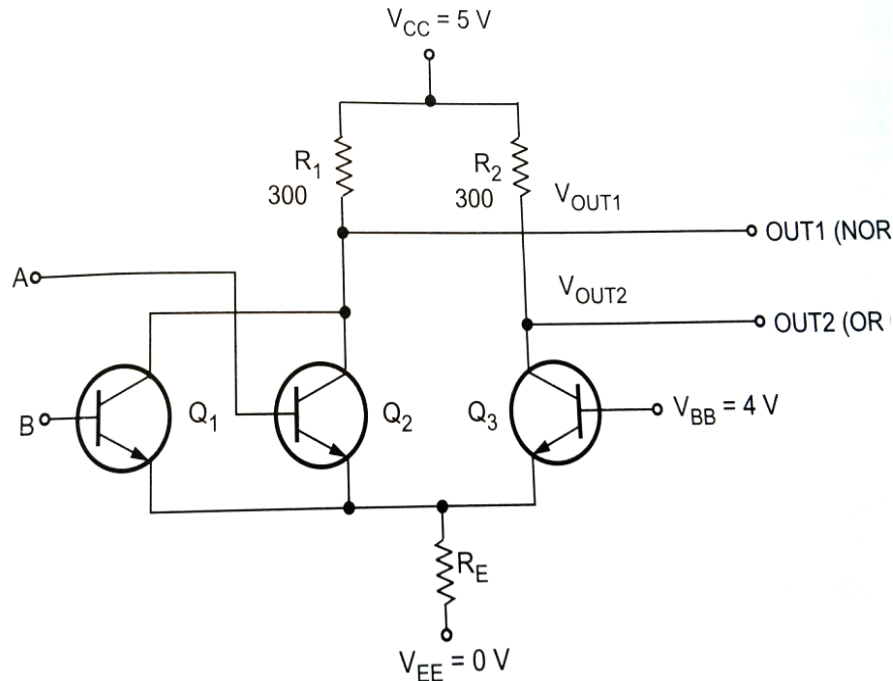

Figure 1.34: 2-input ECL OR/NOR gate

- We can observe that the input and output LOW and HIGH voltage levels for basic ECL family are not same, it has 0.6V difference.
- This is a problem. This problem cannot be solved by connecting diode in series with output to lower its voltage by 0.6 volt because if we do this, it results poor fan-out.
- Another problem occurs when output is HIGH and it drives another ECL input. This HIGH output has to drive base current of another ECL input, resulting drop across R1 or R2, reducing the output voltage.
- These problem of basic ECL are solved by 10 K ECL family.

Advantages of ECL Family:
- ➤ It is a fastest logic family. Offers propagation delay about 1 ns.
- ➤ Transistors are not allowed to go into complete saturation and thus eliminating storage delays.
- ➤ Less switching transients since power supply current is more stable.
- ➤ Large fan-out.

Disadvantages of ECL Family:
- ➤ Low noise immunity.
- ➤ High power dissipation.

23. Compare the characteristics of TTL, ECL and CMOS logic families. (Dec-12)

Comparison between TTL, CMOS and ECL Families:

S.NO	Parameter	CMOS	TTL	ECL
1.	Device used	n-channel and p-channel MOSFET	Bipolar junction transistor	Bipolar junction transistor
2.	$V_{IH(min)}$	3.5 V	2 V	-1.2 V
3.	$V_{IL(max)}$	1.5 V	0.8 V	-1.4 V
4.	$V_{OH(min)}$	4.95 V	2.7 V	-0.9 V
5.	$V_{OL(max)}$	0.005 V	0.4 V	-1.7 V
6.	High level noise margin	V_{NH} = 1.45 V	0.4 V	0.3 V
7.	Low level noise margin	V_{NL} = 1.45 V	0.4 V	0.3 V
8.	Noise immunity	Better than TTL	Less than CMOS	More vulnerable to noise
9.	Propagation delay	70 ns	10 ns	500 ps
10.	Switching speed	Less than TTL	Faster than CMOS	Fastest
11.	Power dissipation per gate	0.1 mW	10 mW	25 mW
12.	Speed power product	0.7 pJ	100 pJ	0.5 pJ
13.	Fan-out	50	10	25
14.	Power supply voltage	3-15 V	Fixed 5 V	-4.5 – 5.2 V
15.	Power dissipation	Increase with frequency	Increase with frequency	Constant with frequency
16.	Application	Portable instrument where battery supply is used.	Laboratory instruments.	High speed instruments.

24. Compare the various digital logic families. (Dec-08)

Comparison between various digital logic families:

Parameter	RTL	DTL	TTL	ECL	CMOS
Components used	Resistors and transistor	Resistor, diode and transistor	Resistor, diode and transistor	Resistor and transistor	N-channel and P-channel MOSFET

Circuits	Simple	Moderate	Complex	Complex	Moderate
Noise margin [Noise immunity]	Nominal	Good	Very good	Good	Very good
Fan-out	Low (4)	Medium (8)	More (10)	High (25)	50
Power dissipation in mW per gate	12	8 - 12	10	40 - 55	0.1
Basic gate	NOR	NAND	NAND	OR-NOR	NAND/NOR
Propagation delay in ns	12	30	10	2(ECL 10 K) 0.75(ECL 100 K)	70
Speed power product (PJ)	144	300	100	100 (ECL 10 K) 40 (ECL 100K)	0.7
Applications	Absolute	Absolute	Laboratory instruments	Due to low propagation delay the are used in high speed switching applications	Due to low power consumption they are used in portable instrument where battery supply is used.
Number of functions	High	Fairly high	Very high	High	Low
Clock rate MHz	8	12 - 30	15 - 60	60 - 400	5

UNIT II
COMBINATIONAL LOGIC

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

COMBINATIONAL CIRCUITS

❖ *A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs.*

❖ A combinational circuit performs an operation that can be specified logically by a set of Boolean functions.
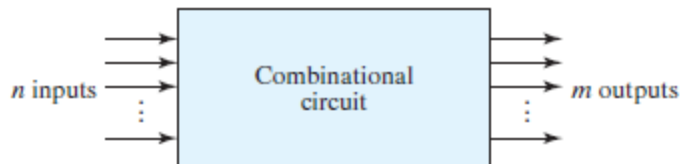
Sequential circuits:

❖ *Sequential circuits employ storage elements in addition to logic gates. Their outputs are a function of the inputs and the state of the storage elements.*

❖ Because the state of the storage elements is a function of previous inputs, the outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs, and the circuit behavior must be specified by a time sequence of inputs and internal states.

ANALYSIS PROCEDURE

Explain the analysis procedure. Analyze the combinational circuit the following logic diagram.
(May 2015)

❖ The analysis of a combinational circuit requires that we determine the function that the circuit implements.

❖ The analysis can be performed manually by finding the Boolean functions or truth table or by using a computer simulation program.

❖ The first step in the analysis is to make that the given circuit is combinational or sequential.

❖ Once the logic diagram is verified to be combinational, one can proceed to obtain the output Boolean functions or the truth table.

❖ To obtain the output Boolean functions from a logic diagram,
- ✓ Label all gate outputs that are a function of input variables with arbitrary symbols or names. Determine the Boolean functions for each gate output.
- ✓ Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols or names. Find the Boolean functions for these gates.
- ✓ Repeat the process in step 2 until the outputs of the circuit are obtained.
- ✓ By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

Logic diagram for analysis example
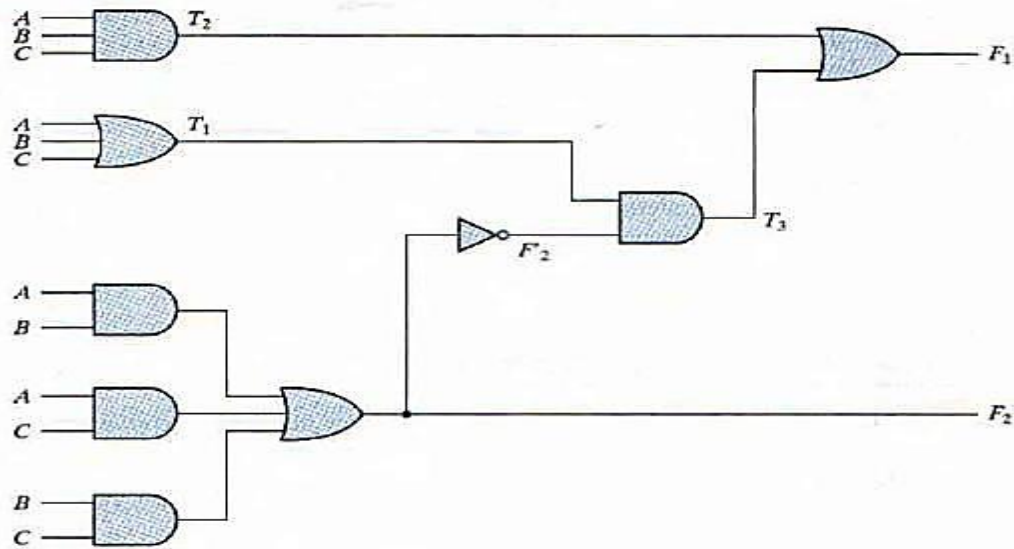
The Boolean functions for the above outputs are,

$$F_2 = AB + AC + BC$$
$$T_1 = A + B + C$$
$$T_2 = ABC$$

Next, we consider outputs of gates that are a function of already defined symbols:

$$T_3 = F'_2 T_1$$
$$F_1 = T_3 + T_2$$

To obtain F_1 as a function of A, B, and C, we form a series of substitutions as follows:

$$
\begin{aligned}
F_1 = T_3 + T_2 &= F'_2 T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC \\
&= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC \\
&= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\
&= A'BC' + A'B'C + AB'C' + ABC
\end{aligned}
$$

❖   Proceed to obtain the truth table for the outputs of those gates which are a function of previously defined values until the columns for all outputs are determined.

Truth Table for the Logic Diagram

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

DESIGNPROCEDURE

Explain the procedure involved in designing combinational circuits.

❖ The design of combinational circuits starts from the specification of the design objective and culminates in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be obtained.

❖ The procedure involved involves the following steps,

✓ From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.

✓ Derive the truth table that defines the required relationship between inputs and outputs.

✓ Obtain the simplified Boolean functions for each output as a function of the input variables.

✓ Draw the logic diagram and verify the correctness of the design.

****

CIRCUITS FOR ARITHMETIC OPERATIONS

Half adder:

Construct a half adder with necessary diagrams.   *(Nov-06, May- 07)*

❖ A half-adder is an arithmetic circuit block that can be used to add two bits and produce two outputs SUM and CARRY.

❖ The Boolean expressions for the SUM and CARRY outputs are given by the equations

$$\text{SUM } S = A.\overline{B} + \overline{A}.B$$

$$\text{CARRY } C = A.B$$

Truth Table:

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1
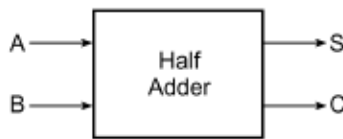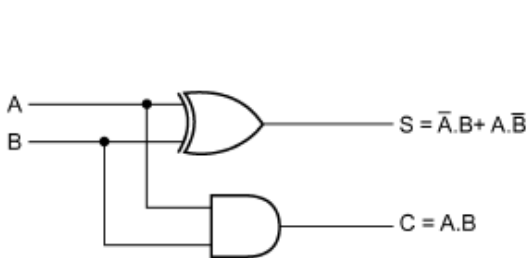
Logic Diagram: **Half adder using NAND gate:**

****
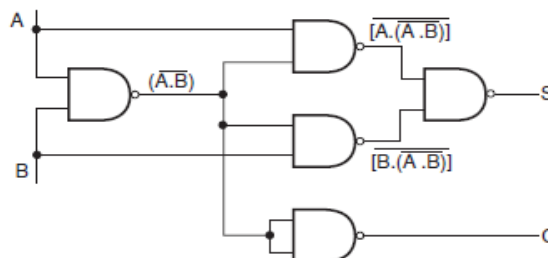
Design a full adder using NAND and NOR gates respectively.   *(Nov -10)*

❖ *A Full-adder is an arithmetic circuit block that can be used to add three bits and produce two outputs SUM and CARRY.*

❖ The Boolean expressions for the SUM and CARRY outputs are given by the equations

$$S = \overline{A}.\overline{B}.C_{in} + \overline{A}.B.\overline{C}_{in} + A.\overline{B}.\overline{C}_{in} + A.B.C_{in}$$
$$C_{out} = B.C_{in} + A.B + A.C_{in}$$

Truth table:

Input variables			Outputs	
X	A	B	S	C
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

Karnaugh map:

K-Map for Sum   K-Map for Carry

❖ The simplified Boolean expressions of the outputs are

$$S = X'A'B + X'AB' + XA'B' + XAB$$
$$C = AB + BX + AX$$

Logic diagram:
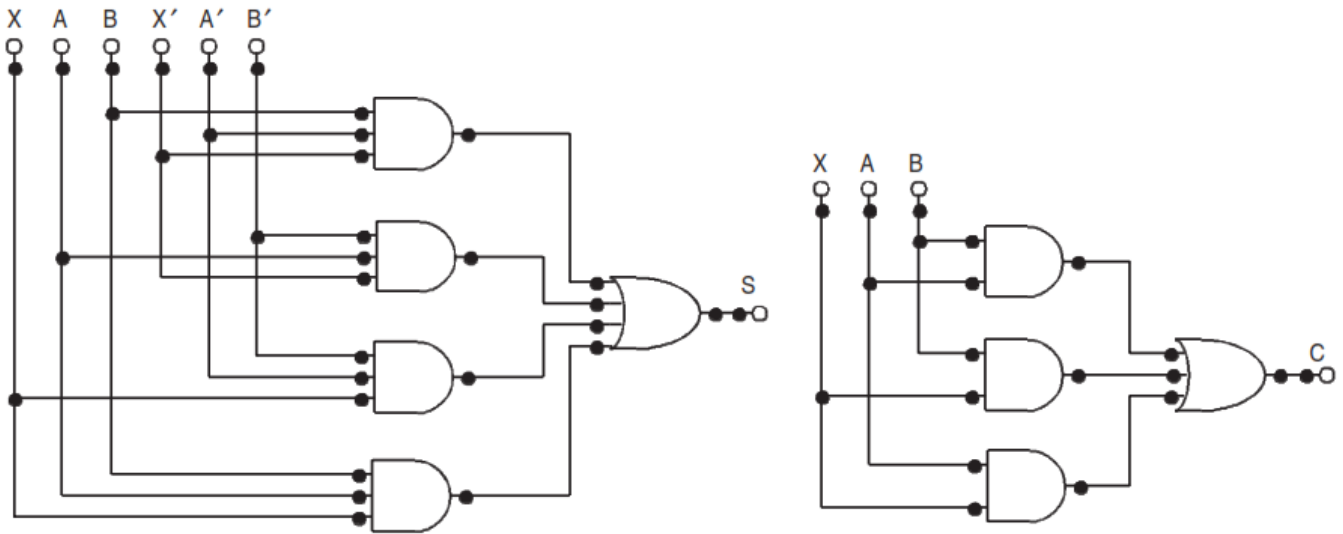
- ❖ The Boolean expressions of S and C are modified as follows

$$S = X'A'B + X'AB' + XA'B' + XAB$$
$$= X' (A'B + AB') + X (A'B' + AB)$$
$$= X' (A \oplus B) + X (A \oplus B)'$$
$$= X \oplus A \oplus B$$
$$C = AB + BX + AX = AB + X (A + B)$$
$$= AB + X (AB + AB' + AB + A'B)$$
$$= AB + X (AB + AB' + A'B)$$
$$= AB + XAB + X (AB' + A'B)$$
$$= AB + X (A \oplus B)$$

Full adder using Two half adder:

- ❖ Logic diagram according to the modified expression is shown Figure.

****
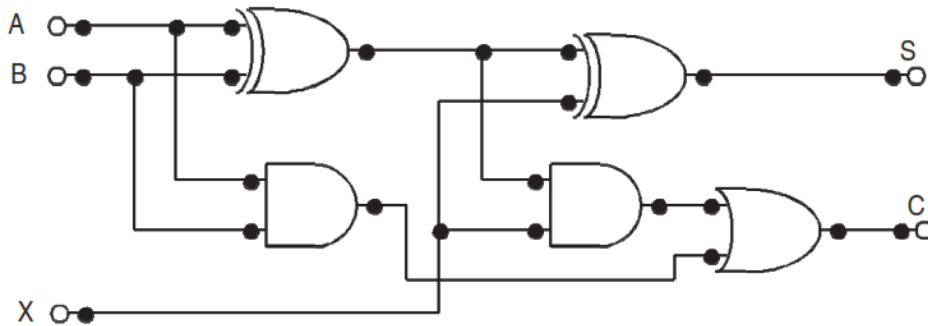
Design a half subtractor circuit. (Nov-2009)

- ❖ A half-subtractor is a combinational circuit that can be used to subtract one binary digit from another to produce a DIFFERENCE output and a BORROW output.
- ❖ The BORROW output here specifies whether a '1' has been borrowed to perform the subtraction. The Boolean expression for difference and borrow is:

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

$$B_0 = \overline{A}.B$$

A	B	D	B_O
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0
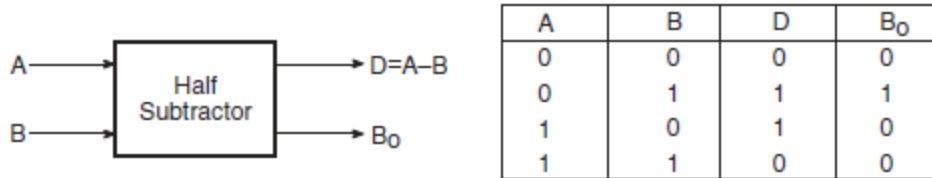
Logic diagram:

****
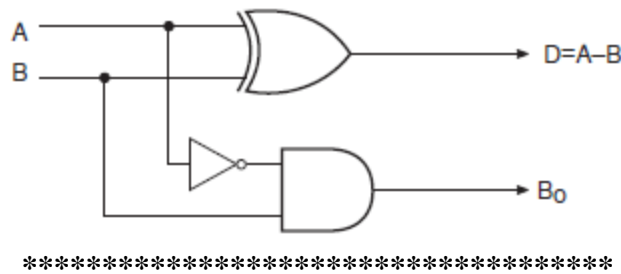
Full subtractor:

Design a full subtractor. (Nov-2009,07)

- ❖ A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend, and also takes into consideration whether a '1' has already been borrowed by the previous adjacent lower minuend bit or not.

- ❖ As a result, there are three bits to be handled at the input of a full subtractor, namely the two bits to be subtracted and a borrow bit designated as Bin .

- ❖ There are two outputs, namely the DIFFERENCE output D and the BORROW output Bo. The BORROW output bit tells whether the minuend bit needs to borrow a '1' from the next possible higher minuend bit. The Boolean expression for difference and barrow is:
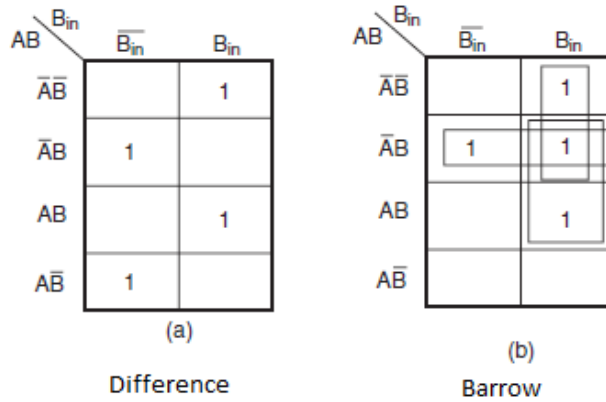
$$D = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + A.\overline{B}.\overline{B}_{in} + A.B.B_{in}$$

$$B_0 = \overline{A}.B + \overline{A}.B_{in} + B.B_{in}$$
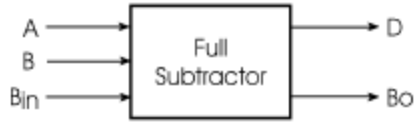
Minuend (A)	Subtrahend (B)	Borrow In (B_{in})	Difference (D)	Borrow Out (B_O)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-Map:

(a)

Difference

(b)

Barrow

Full subtractor using two half subtractor:

****
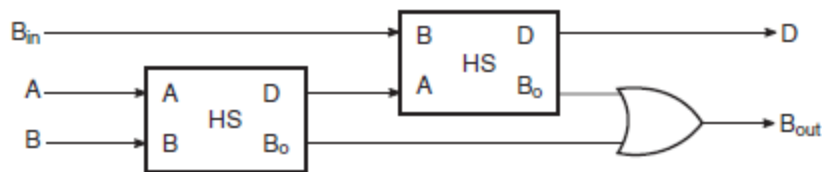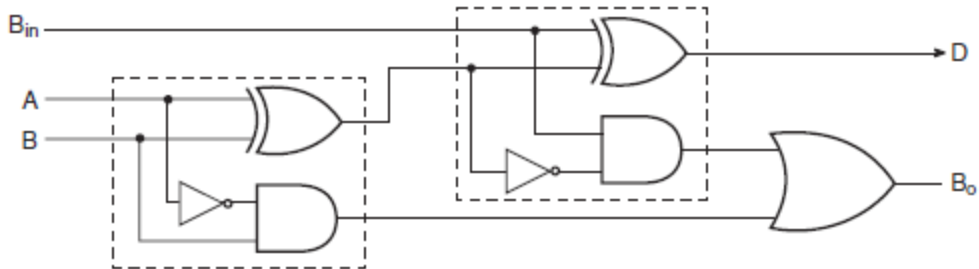
Parallel Binary Adder: (Ripple Carry Adder):

Explain about four bit adder. (or) Design of 4 bit binary adder – subtractor circuit. (Apr – 2019)

❖ *A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.*

❖ Addition of n-bit numbers requires a chain of n- full adders or a chain of one-half adder and n-1 full adders. In the former case, the input carry to the least significant position is fixed at 0.

❖ Figure shows the interconnection of four full-adder (FA) circuits to provide a four-bit binary ripple carry adder.

❖ The carries are connected in a chain through the full adders. The input carry to the adder is C0, and it ripples through the full adders to the output carry C4. The S outputs generate the required sum bits.

Example: Consider the two binary numbers A = 1011and B = 0011. Their sum S = 1110 is formed with the four-bit adder as follows:

Subscript i:	3	2	1	0	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}
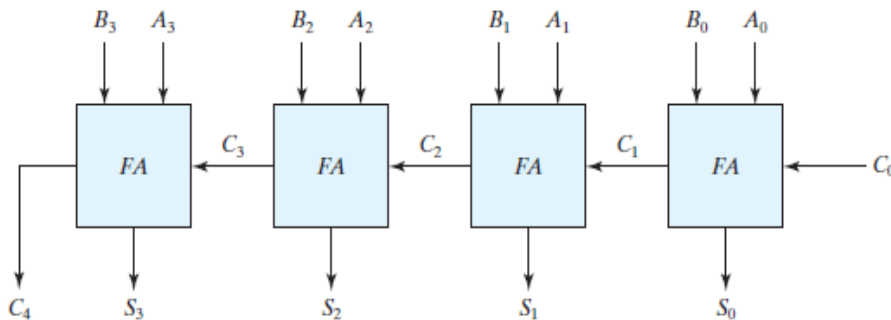
✓ The carry output of lower order stage is connected to the carry input of the next higher order stage. Hence this type of adder is called ripple carry adder.

✓ In a 4-bit binary adder, where each full adder has a propagation delay of tp ns, the output in the fourth stage will be generated only after 4tp ns.

✓ The magnitude of such delay is prohibitive for high speed computers.

✓ One method of speeding up this process is look-ahead carry addition which eliminates ripple carry delay.

****

Complement of a number:

1's complement:

The 1's complement of a binary number is formed by changing 1 to 0 and 0 to 1.

Example:

1. The 1's complement of 1011000 is 0100111.
2. The 1's complement of 0101101 is 1010010.

2's complement:

The 2's complement of a binary number is formed by adding 1 with 1's complement of a binary number.

Example:

1. The 2's complement of 1101100 is 0010100
2. The 2's complement of 0110111 is 1001001

Subtraction using 2's complement addition:

✓ The subtraction of unsigned binary number can be done by means of complements.
✓ Subtraction of A-B can be done by taking 2's complement of B and adding it to A.
✓ Check the resulting number. If carry present, the number is positive and remove the carry.
✓ If no carry present, the resulting number is negative, take the 2's complement of result and put negative sign.

Example:

Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction
(a) $X - Y$ and **(b)** $Y - X$ by using 2's complements.

Solution:

(a) $X = 1010100$
2's complement of $Y = + 0111101$
Sum= 10010001
Discard end carry. *Answer*: $X - Y = 0010001$
(b) $Y = 1000011$
2's complement of $X = + 0101100$
Sum= 1101111
There is no end carry. Therefore, the answer is $Y - X = -(2's$ complement of $1101111) = -0010001$.

****

Parallel Binary Subtractor:
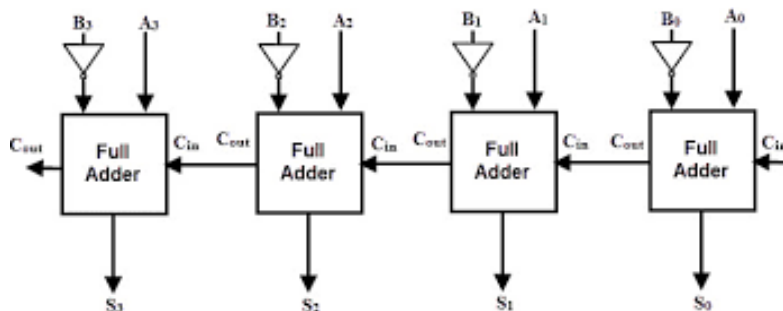
✓ The subtraction of unsigned binary numbers can be done most conveniently by means of complements. The subtraction $A - B$ can be done by taking the 2's complement of B and adding it to A. The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair

ofbits. The 1's complement can be implemented with inverters, and a 1 can be added tothe sum through the input carry.

✓ The circuit for subtracting $A - B$ consists of an adder with inverters placed betweeneach data input B and the corresponding input of the full adder. The input carry Cin mustbe equal to 1 when subtraction is performed. The operation thus performed becomes A,plus the 1's complement of B , plus 1. This is equal to Aplus the 2's complement of B.

✓ For unsigned numbers, that gives $A-B$ if $A>=B$ or the 2's complement of $B - A$if $A <B$. For signed numbers, the result is $A - B$, provided that there is no overflow.

****

Fast adder (or) Carry Look Ahead adder:

Design a carry look ahead adder circuit.   *(Nov-2010)*

❖ *The carry look ahead adder is based on the principle of looking at the lower order bits of the augend and addend to see if a higher order carry is to be generated.*

❖ It uses two functions carry generate and carry propagate.
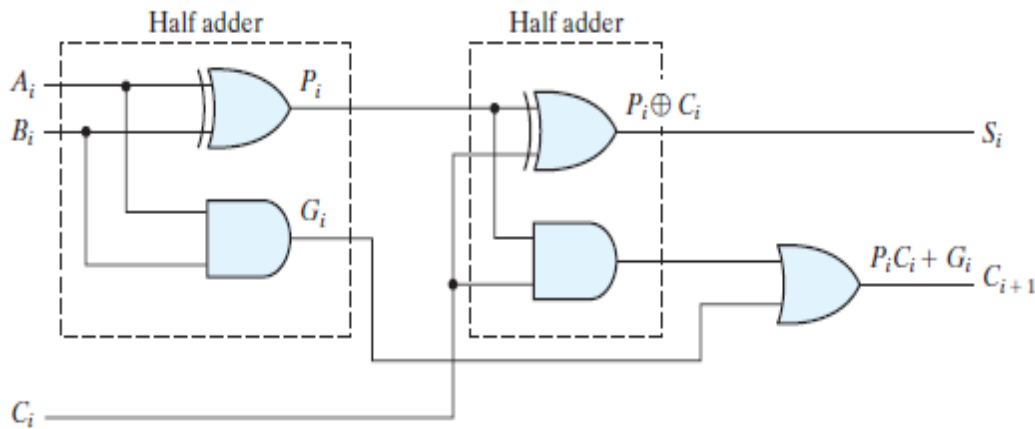
Consider the circuit of the full adder shown in Fig. If we define two new binaryvariables

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

the output sum and carry can respectively be expressed as

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

G_i is called a carry generate, and it produces a carry of 1 when both A_i and B_i are 1,regardless of the input carry C_i. P_i is called a carry propagate, because it determines whether a carry into stage i will propagate into stage i+ 1 (i.e., whether an assertion of C_i will propagate to an assertion of C_{i+1}).
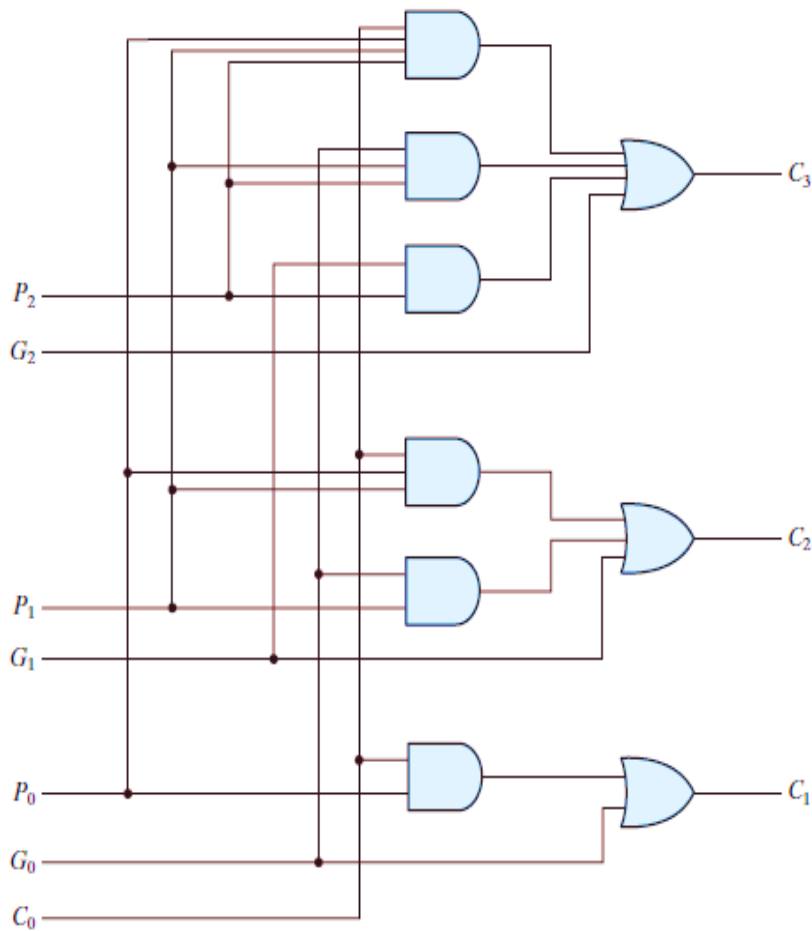
We now write the Boolean functions for the carry outputs of each stage and substitutethe value of each C_i from the previous equations:

$C_0 =$ input carry

$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$

$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 = P_2P_1P_0C_0$

**Logic diagram of carry lookahead generator**

- ❖ The construction of a four-bit adder with a carry lookahead scheme is shown in Fig.
- ❖ Each sum output requires two exclusive-OR gates.
- ❖ The output of the first exclusive-OR gate generates the Pi variable, and the AND gate generates the Gi variable.
- ❖ The carries are propagated through the carry look ahead generator and applied as inputs to the second exclusive-OR gate.
- ❖ All output carries are generated after a delay through two levels of gates.
- ❖ Thus, outputs S1 through S3 have equal propagation delay times. The two-level circuit for the output carry C4 is not shown. This circuit can easily be derived by the equation-substitution method.

****
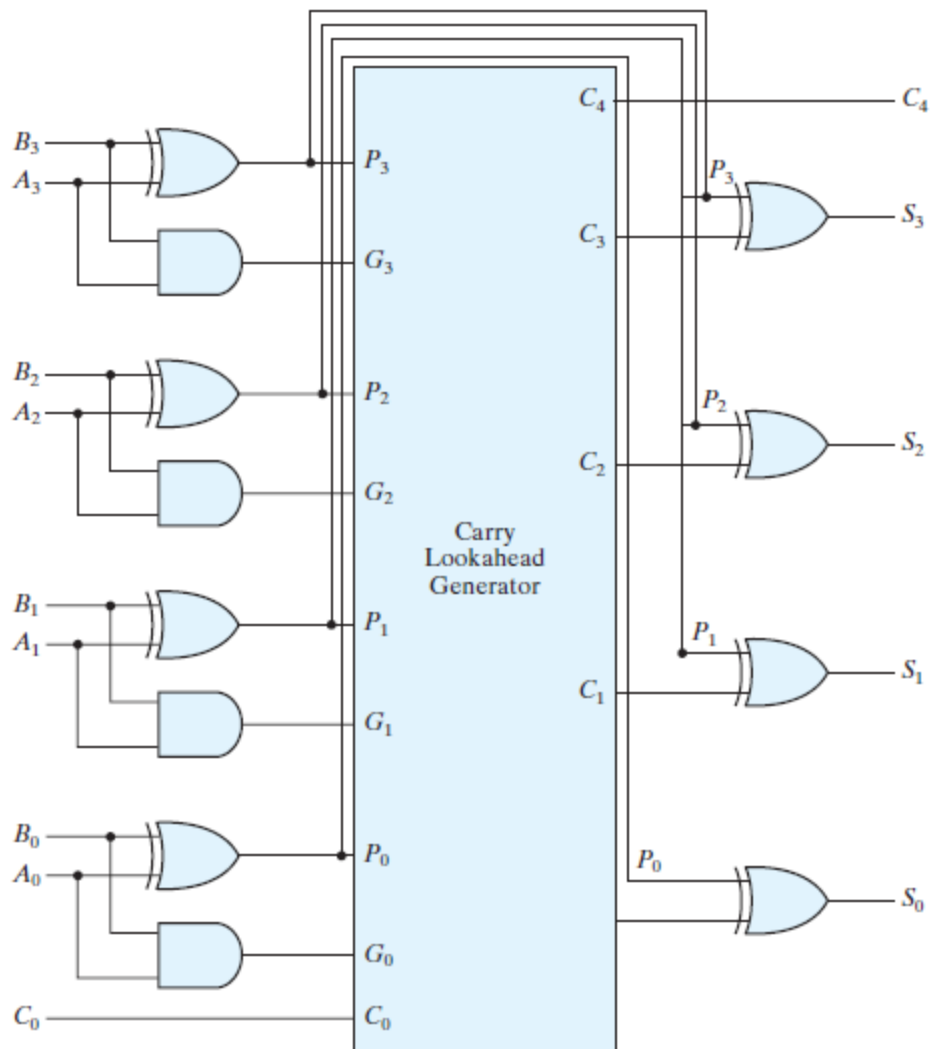
4 bit-Parallel adder/subtractor:

Explain about binary parallel / adder subtractor. [NOV – 2019]

❖ The addition and subtraction operations can be combined into one circuit with one common binary adder by including an exclusive-OR gate with each full adder. A four-bit adder–subtractor circuit is shown in Fig.

❖ The mode input M controls the operation. When $M = 0$, the circuit is an adder, and when $M = 1$, the circuit becomes a subtractor.
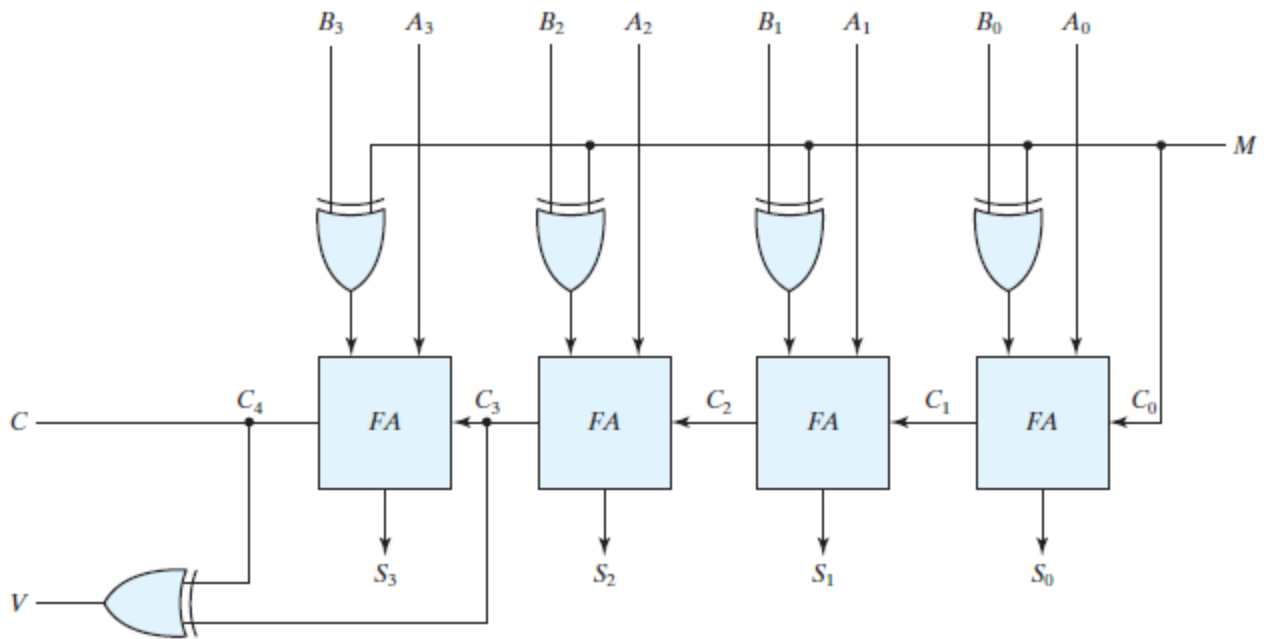
- ❖ It performs the operations of both addition and subtraction.
- ❖ It has two 4bit inputs $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$.
- ❖ The mode input M controls the operation when M=0 the circuit is an adder and when M=1 the circuits become subtractor.
- ❖ Each exclusive-OR gate receives input M and one of the inputs of B.
- ❖ When $M = 0$, we have B xor $0 = B$. The full adders receive the value of B, the input carry is 0, and the circuit performs A plus B. This results in sum $S_3S_2S_1S_0$ and carry C_4.
- ❖ When $M = 1$, we have B xor $1 = B'$ and $C_0 = 1$. The B inputs are all complemented and a 1 is added through the input carry thus producing 2's complement of B.
- ❖ Now the data $A_3A_2A_1A_0$ will be added with 2's complement of $B_3B_2B_1B_0$ to produce the sum i.e., A-B if A≥B or the 2's complement of B-A if A<B.

****

Comparators
Design a 2 bit magnitude comparator. *(May 2006)*

It is a combinational circuit that compares two numbers and determines their relative magnitude. The output of comparator is usually 3 binary variables indicating:

A<B, A=B, A>B
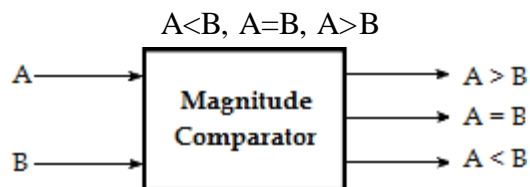
1-bitcomparator: Let's begin with 1bit comparator and from the name we can easily make out that this circuit would be used to compare 1bit binary numbers.

A	B	A>B	A=B	A<B
0	0	0	1	0
1	0	1	0	0
0	1	0	0	1
1	1	0	1	0

For a 2-bit comparator we have four inputs A1 A0 and B1 B0 and three output E (is 1 if two numbers are equal) G (is 1 when A>B) and L (is 1 when A<B) If we use truth table and K-map the result is

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

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

The equation isf(A=B) = $\overline{A}.\overline{B}$ + A.B
                        = A XNOR B

Design of 2 – bit Magnitude Comparator.

The truth table of 2-bit comparator is given in table below

Truth table:

Inputs				Outputs		
A_3	A_2	A_1	A_0	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

K-Map:

For A>B

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

$A>B = A_0B_1'B_0' + A_1B_1' + A_1A_0B_0'$

For A=B

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

$A=B = A_1'A_0'B_1'B_0' + A_1'A_0B_1'B_0 +$
$\qquad\quad A_1A_0B_1B_0 + A_1A_0'B_1B_0'$
$\quad\; = A_1'B_1'(A_0'B_0' + A_0B_0) + A_1B_1(A_0B_0 + A_0'B_0')$
$\quad\; = (A_0 \odot B_0)(A_1 \odot B_1)$

For A<B

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

$A<B = A_1'A_0'B_0 + A_0'B_1B_0 + A_1'B_1$

Logic Diagram:

$$A>B = A_0B_1'B_0' + A_1B_1' + A_1A_0B_0'$$

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

$$A<B = A_1'A_0'B_0 + A_0'B_1B_0 + A_1'B_1$$

****
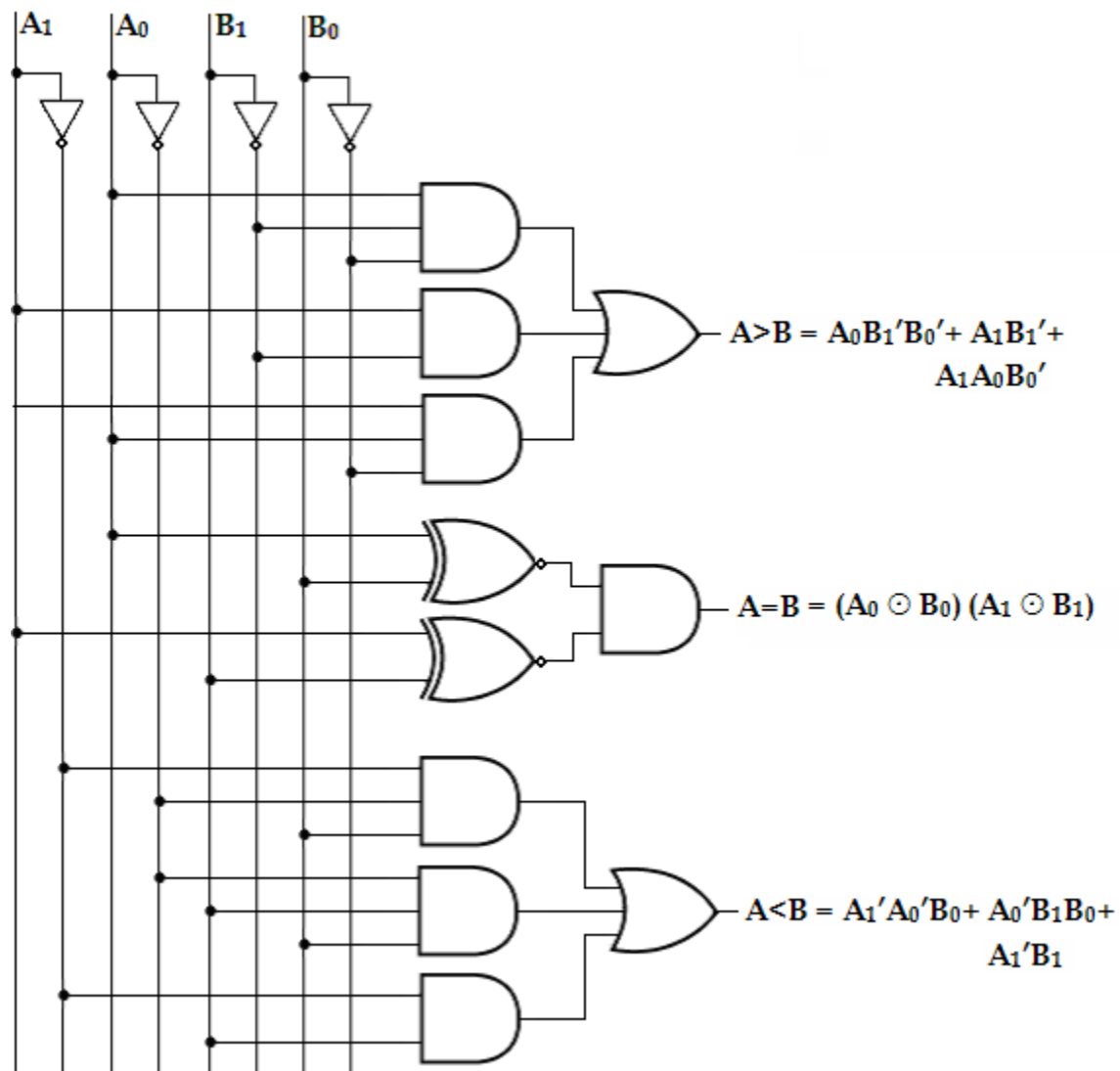
4 bit magnitude comparator:

Design a 4 bit magnitude comparators. (Apr – 2019)

Input

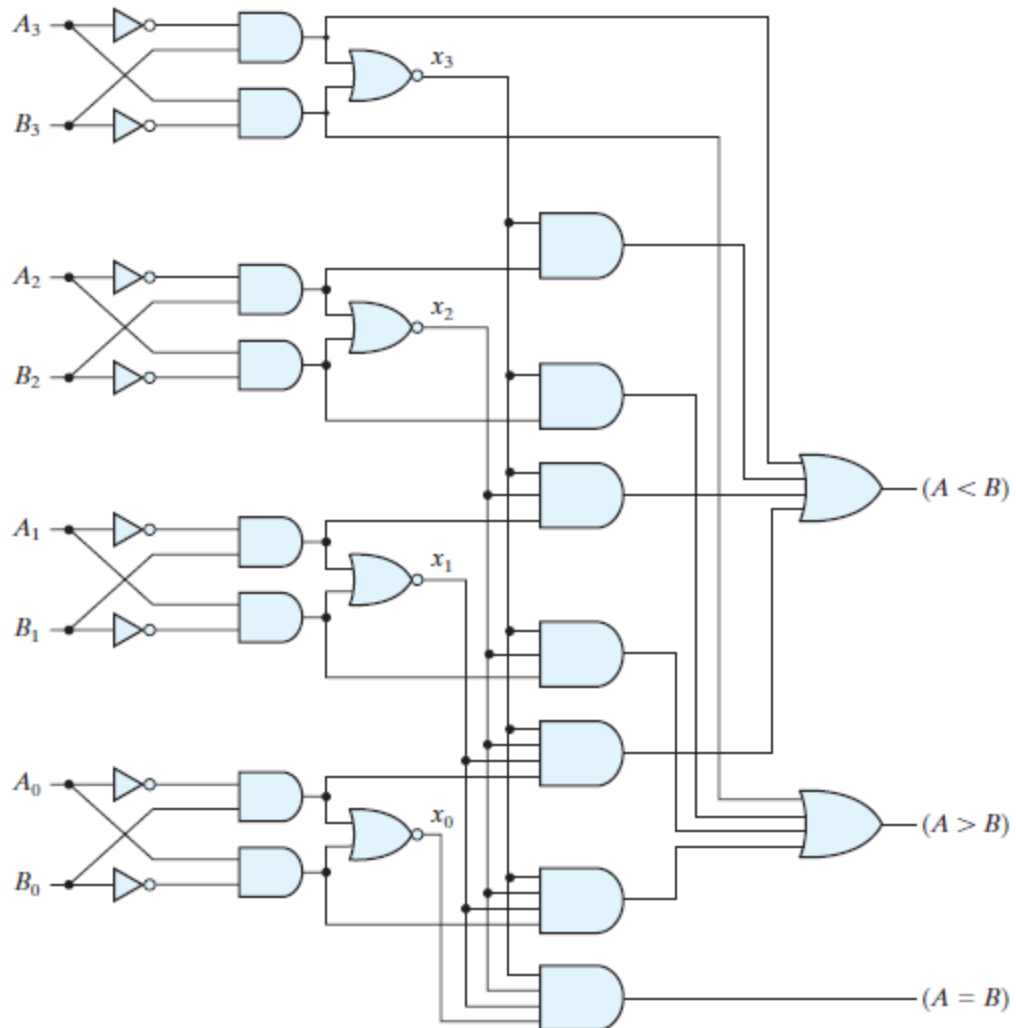$$A = A_3 \, A_2 \, A_1 \, A_0$$
$$B = B_3 \, B_2 \, B_1 \, B_0$$

Function Equation

$$(A = B) = x_3 x_2 x_1 x_0$$

$$(A > B) = A_3 B_3' + x_3 A_2 B_2' + x_3 x_2 A_1 B_1' + x_3 x_2 x_1 A_0 B_0'$$

$$(A < B) = A_3' B_3 + x_3 A_2' B_2 + x_3 x_2 A_1' B_1' + x_3 x_2 x_1 A' n_0 B_0'$$

Four-bit magnitude comparator

****

BCD Adder:

Design to perform BCD addition.(or) What is BCD adder? Design an adder to perform arithmetic addition of two decimal bits in BCD. *(May -08)(Apr 2017,2018)[Nov – 2019]*

❖ Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than $9 + 9 + 1 = 19$, the 1 in the sum being an input carry.

❖ Suppose we apply two BCD digits to a four-bit binary adder. The adder will form the sum in binary and produce a result that ranges from 0 through 19. These binary numbers are listed in Table and are labeled by symbols K, Z8, Z4, Z2, and Z1. K is the carry, and the subscripts under the letter Z represent the weights 8, 4, 2, and 1 that can be assigned to the four bits in the BCD code.

Derivation of BCD Adder

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19
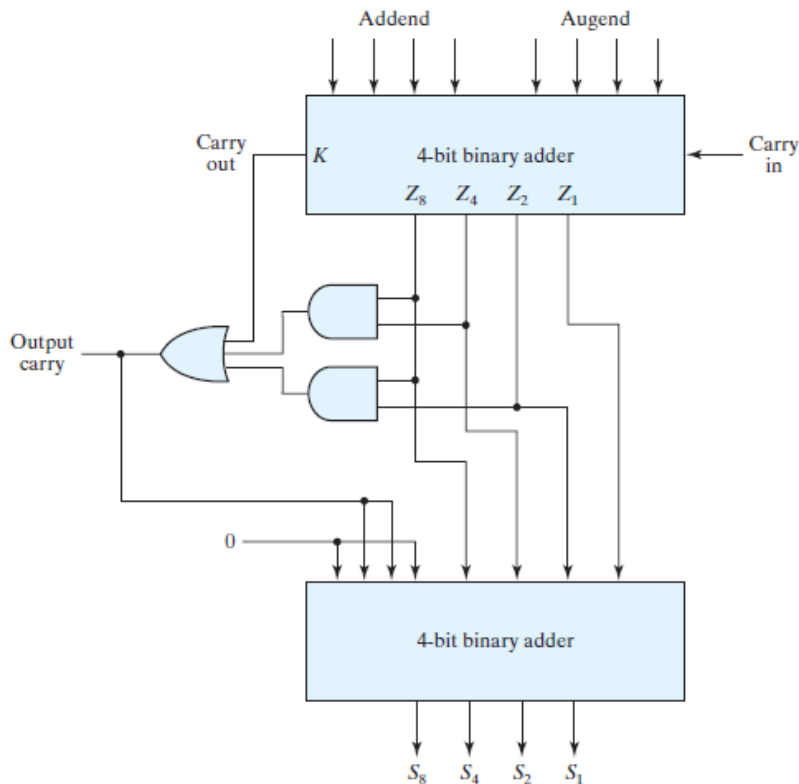
❖ A BCD adder that adds two BCD digits and produces a sum digit in BCD is shown in Fig. The two decimal digits, together with the input carry, are first added in the top four-bit adder to produce the binary sum.

❖ When the output carry is equal to 0, nothing is added to the binary sum. When it is equal to 1, binary 0110 is added to the binary sum through the bottom four-bit adder.

❖ The condition for a correction and an output carry can be expressed by the Boolean function

$$C = K + Z_8 Z_4 + Z_8 Z_2$$

❖ The output carry generated from the bottom adder can be ignored, since it supplies information already available at the output carry terminal.

❖ A decimal parallel adder that adds n decimal digits needs n BCD adder stages. The output carry from one stage must be connected to the input carry of the next higher order stage.

****

Binary Multiplier:
Explain about binary Multiplier.

❖ Multiplication of binary numbers is performed in the same way as multiplication of decimal numbers. The multiplicand is multiplied by each bit of the multiplier, starting from the least significant bit. Each such multiplication forms a partial product.

❖ Successive partial products are shifted one position to the left. The final product is obtained from the sum of the partial products.
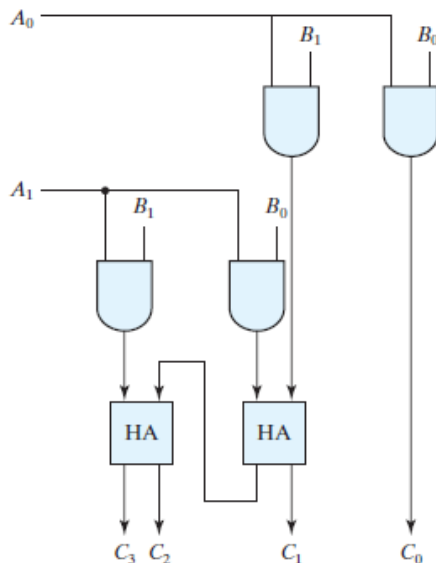
❖ A combinational circuit binary multiplier with more bits can be constructed in a similar fashion.

❖ A bit of the multiplier is ANDed with each bit of the multiplicand in as many levels as there are bits in the multiplier.

❖ The binary output in each level of AND gates is added with the partial product of the previous level to form a new partial product. The last level produces the product.

****
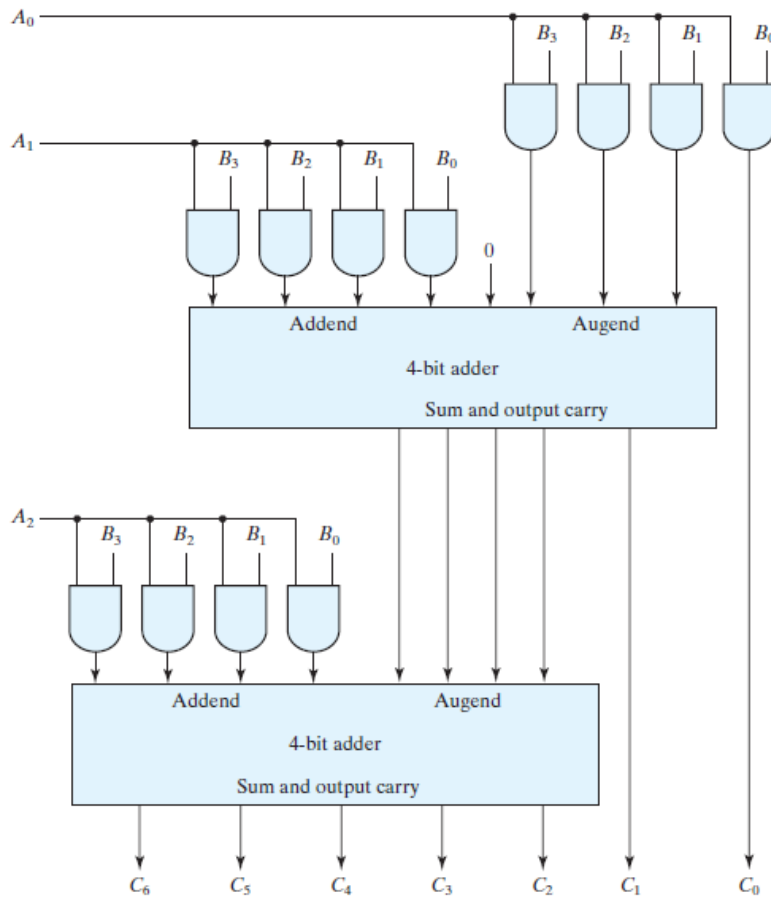
Design a binary to gray converter.   *(Nov-2009)(Nov 2017)*
Binary to Grayconverter

Gray code is unit distance code.
Input code: Binary [B_3 B_2 B_1 B_0]
output code: Gray [G_3 G_2 G_1 G_0]

Truth Table

B3	B2	B1	B0	G3	G2	G1	G0
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
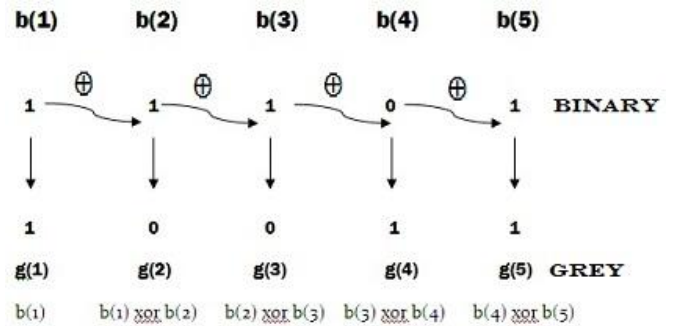
K-MAP FORG3:   **K-MAP FORG2:**

G3=B3   G2=B3'B2+B3B2'=B3 $\oplus$ B2

K-MAP FORG1 :

$G1 = B1'B2 + B1B2' = B1 \oplus B2$

K-MAP FORG0 :

$G0 = B1' B0 + B1B0' = B1 \oplus B0$

Logic diagram:

Binary to Gray Converter
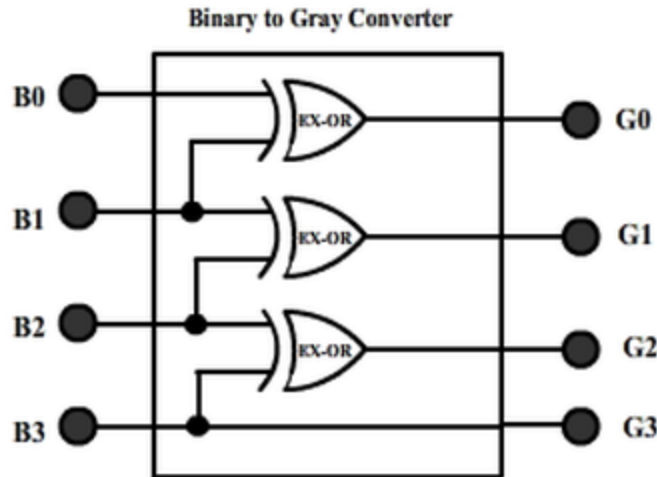
Gray to Binary converter:

Design a gray to binary converter.(OR) Design a combinational circuit that converts a four bit gray code to a four bit binary number using exclusive – OR gates.   (Nov-2009) [NOV – 2019]

Gray code is unit distance code.

Input code: Gray [G_3 G_2 G_1 G_0]

output code: Binary [B_3 B_2 B_1 B_0]

i.e
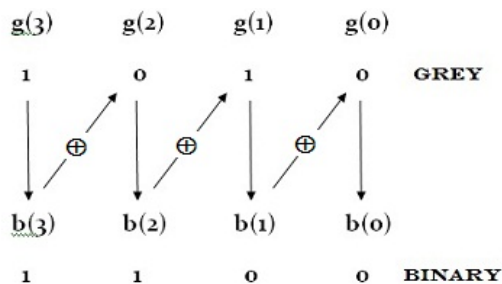
$b(3) = g(3)$

$b(2) = b(3) \oplus g(3)$

$b(1) = b(2) \oplus g(1)$

$b(0) = b(1) \oplus g(0)$

Truth Table:

Gray code				Natural-binary code			
G3	G2	G1	G0	B3	B2	B1	B0
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	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0

K-Map:

For B_3

$B_3 = G_3$

For B_2

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

For B_1

For B_0

From the above K-map,

$B_3 = G_3$

$B_2 = G_3'G_2 + G_3G_2'$

$B_2 = G_3 \oplus G_2$

$B_1 = G_3'G_2'G_1 + G_3'G_2G_1' + G_3G_2G_1 + G_3G_2'G_1'$

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

    $= G_3' (G_2 \oplus G_1) + G_3 (G_2 \oplus G_1)'$ $[x \oplus y = x'y + xy'], [(x \oplus y)' = xy + x'y']$

$B_1 = G_3 \oplus G_2 \oplus G_1$

$B_0 = G_3'G_2' G_1'G_0 + G_3'G_2\_G_1G_0' + G_3G_2G_1\_G_0 + G_3G_2G_1 G_0' + G_3'G_2G_1'G_0' +$

    $G_3G_2\_G_1'G_0' + G_3'G_2G_1G_0 + G_3G_2\_G_1 G_0.$

    $= G_3'G_2' (G_1'G_0 + G_1G_0') + G_3G_2 (G_1'G_0 + G_1G_0') + G_1'G_0' (G_3'G_2 + G_3G_2') +$

    $G_1G_0 (G_3'G_2 + G_3G_2').$

    $= G_3'G_2' (G_0 \oplus G_1) + G_3G_2 (G_0 \oplus G_1) + G_1'G_0' (G_2 \oplus G_3) + G_1G_0 (G_2 \oplus G_3).$

    $= G_0 \oplus G_1 (G_3'G_2' + G_3G_2) + G_2 \oplus G_3 (G_1'G_0' + G_1G_0)$

    $= (G_0 \oplus G_1) (G_2 \oplus G_3)' + (G_2 \oplus G_3) (G_0 \oplus G_1)$ $[x \oplus y = x'y + xy']$

$B_0 = (G_0 \oplus G_1) \oplus (G_2 \oplus G_3).$

Logic Diagram:
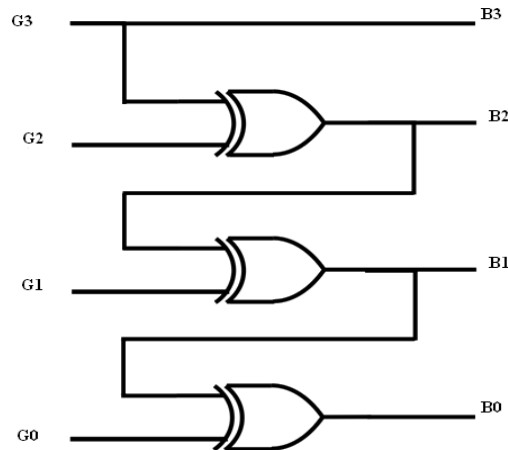
BCD to Excess -3 converter:

Design a combinational circuits to convert binary coded decimal number into an excess-3 code.

❖ Excess-3 code is modified form of BCD code.   *(Nov-06,09,10, May-08,10)*

❖ Excess -3 code is derived from BCD code by adding 3 to each coded number.

Truth table:

Decimal	BCD code				Excess-3 code			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

K-Map:

For E_3

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

For E_2

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

For E_1

$E_1 = B_1' B_0' + B_1 B_0$

$\quad = B_1 \odot B_0$

For E_0

$E_0 = B_0'$

Logic Diagram

BCD Code

B3 B2 B1 B0

Excess-3 Code

$E_0 = B_0'$

$E_1 = B_0 \odot B_1$

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

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

Excess -3 to BCD converter:

Design a combinational circuit to convert Excess-3 to BCD code. *(May 2007)*

Truth table:

Decimal	Excess-3 code				BCD code			
	E_3	E_2	E_1	E_0	B_3	B_2	B_1	B_0
3	0	0	1	1	0	0	0	0
4	0	1	0	0	0	0	0	1
5	0	1	0	1	0	0	1	0
6	0	1	1	0	0	0	1	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	0	1	0	1
9	1	0	0	1	0	1	1	0
10	1	0	1	0	0	1	1	1
11	1	0	1	1	1	0	0	0
12	1	1	0	0	1	0	0	1

K-map simplification

For B₀

E_3E_2 \ E_1E_0	00	01	11	10
00	X	X	0	X
01	1	0	0	1
11	1	X	X	X
10	1	0	0	1

$$B_0 = \overline{E_0}$$

For B₁

E_3E_2 \ E_1E_0	00	01	11	10
00	X	X	0	X
01	0	1	0	1
11	0	X	X	X
10	0	1	0	1

$$B_1 = \overline{E_1}E_0 + E_1\overline{E_0}$$
$$= E_1 \oplus E_0$$

For B₂

E_3E_2 \ E_1E_0	00	01	11	10
00	X	X	0	X
01	0	0	1	0
11	0	X	X	X
10	1	1	0	1

$$B_2 = \overline{E_2}\overline{E_1} + E_2E_1E_0 + E_3E_1\overline{E_0}$$

For B₃

E_3E_2 \ E_1E_0	00	01	11	10
00	X	X	0	X
01	0	0	0	0
11	1	X	X	X
10	0	0	1	0

$$B_3 = E_3E_2 + E_3E_1E_0$$

Logic diagram

Excess -3 code

E_3 E_2 E_1 E_0

BCD code

B_0

B_1

B_2

B_3
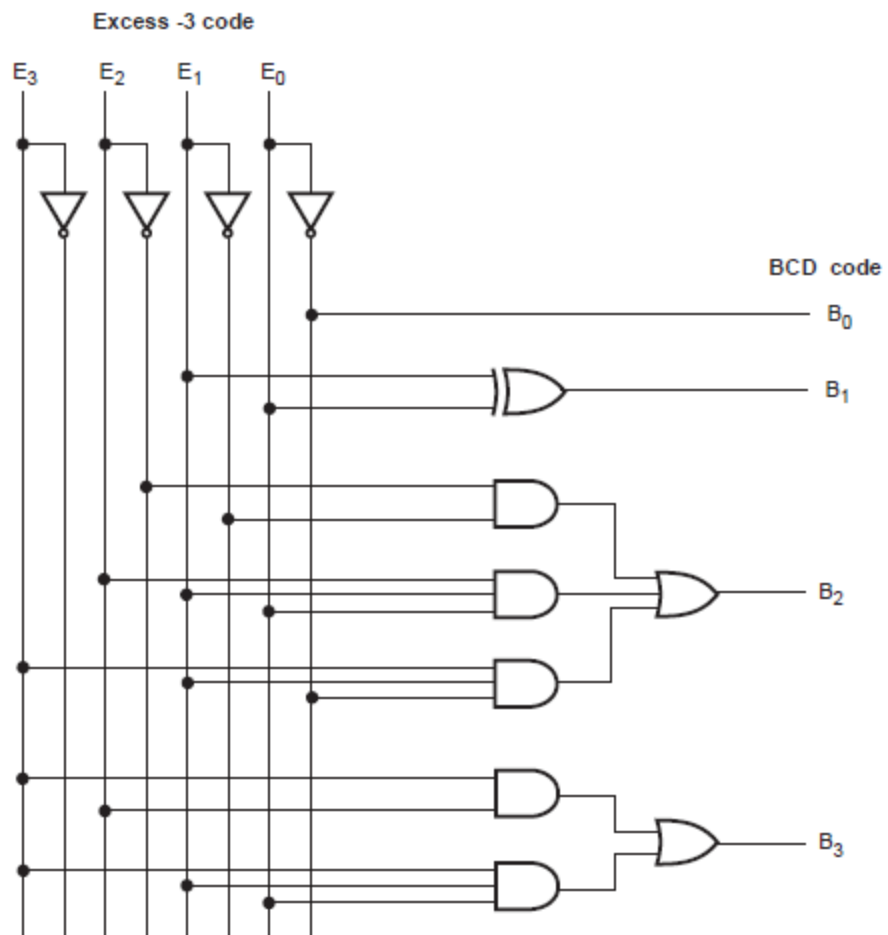
Design Binary to BCD converter.
Truth table:

Decimal	Binary Code				BCD Code				
	D	C	B	A	B_4	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	0	1	1
4	0	1	0	0	0	0	1	0	0
5	0	1	0	1	0	0	1	0	1
6	0	1	1	0	0	0	1	1	0
7	0	1	1	1	0	0	1	1	1
8	1	0	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	0	1
10	1	0	1	0	1	0	0	0	0
11	1	0	1	1	1	0	0	0	1
12	1	1	0	0	1	0	0	1	0
13	1	1	0	1	1	0	0	1	1
14	1	1	1	0	1	0	1	0	0
15	1	1	1	1	1	0	1	0	1

K-map:

For B_0

DC\BA	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

$B_0 = A$

For B_1

DC\BA	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	1	0	0
10	0	0	0	0

$B_1 = DCB' + D'B$

For B_2

DC\BA	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	1	1
10	0	0	0	0

$B_2 = D'C + CB$

For B_3

DC\BA	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	1	1	0	0

$B_3 = DC'B'$

For B₄

DC\BA	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

$B_4 = DC + DB$

Logic diagram:

Binary Code

$B_0 = A$

$B_1 = DCB' + D'B$

$B_2 = D'C + CB$

$B_3 = DC'B'$

$B_4 = DC + DB$

****
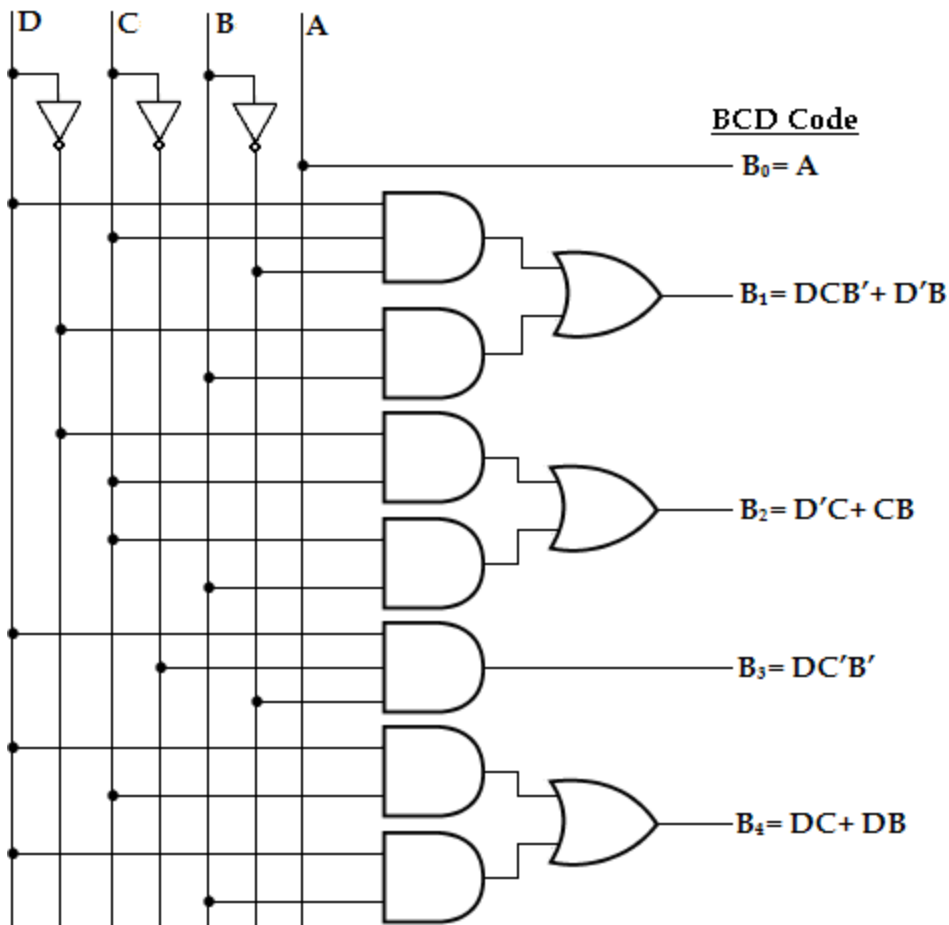
Decoder:

Explain about decoders with necessary diagrams.   *(Apr 2018)(Nov 2018)*

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

❖ The purpose of a decoder is to generate the 2^n (or fewer) minterms of n input variables, shown below for two input variables.

2 to 4 decoder:
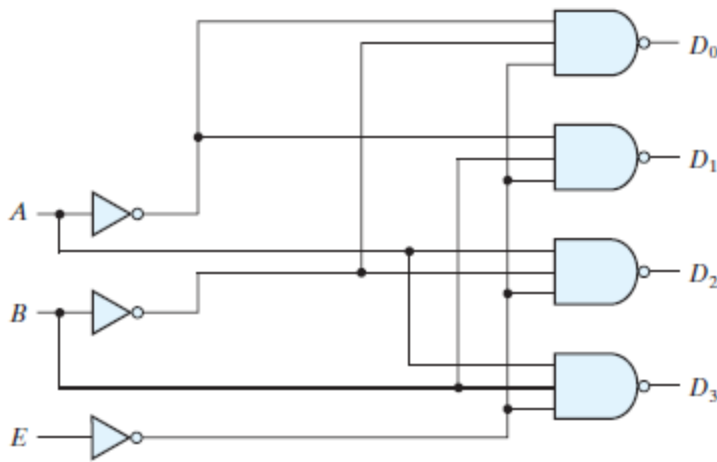
E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	1

(a) Logic diagram   (b) Truth table

3 to 8 Decoder:
Design 3 to 8 line decoder with necessary diagram.   *May -10)*
Truth table:

Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Logic diagram:

$D_0 = x'y'z'$

$D_1 = x'y'z$

$D_2 = x'yz'$

$D_3 = x'yz$

$D_4 = xy'z'$

$D_5 = xy'z$

$D_6 = xyz'$

$D_7 = xyz$
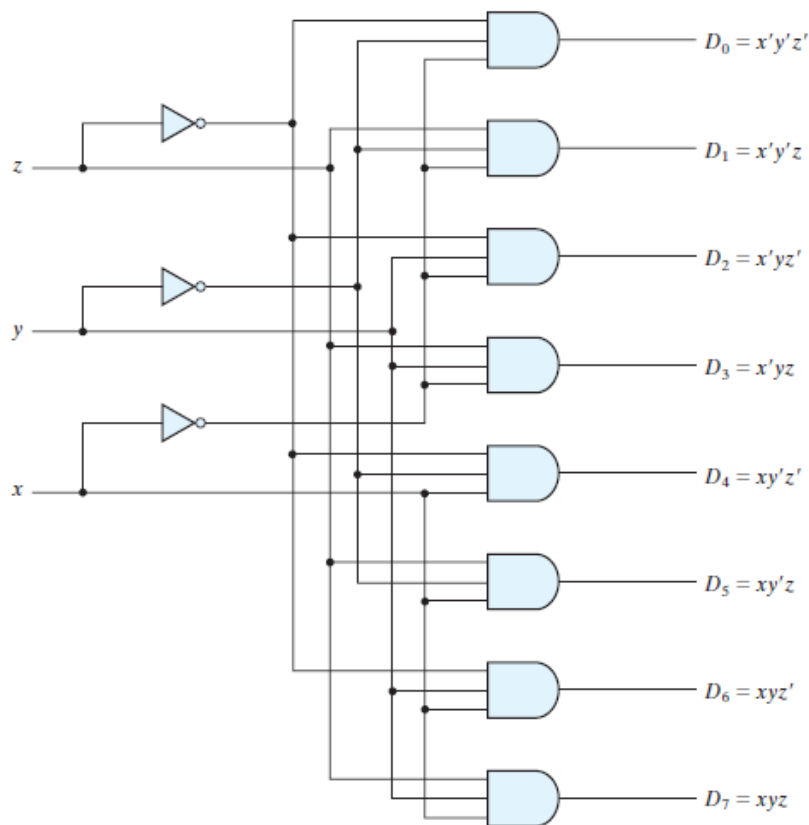
Design for 3 to 8 decoder with 2 to 4 decoder:

❖ Not that the two to four decoder design shown earlier, with its *enable* inputs can be used to build a three to eight decoder as follows.
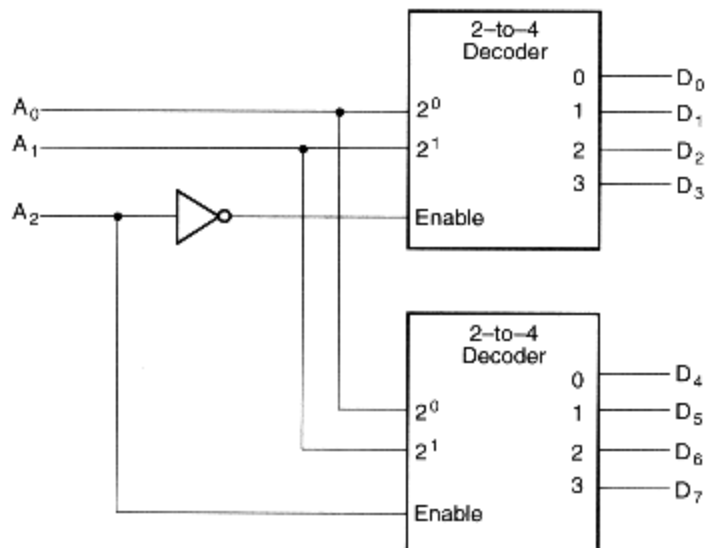
Implementation of Boolean function using decoder:

❖ Since the three to eight decoder provides all the minterms of three variables, the realisation of a function in terms of the sum of products can be achieved using a decoder and OR gates as follows.

Example: *Implement full adder using decoder.*

Sum is given by $\sum m(1, 2, 4, 7)$ while *Carry* is given by $\sum m(3, 5, 6, 7)$ as given by the minterms each of the OR gates are connected to.

Solution :

Step 1 : Truth table
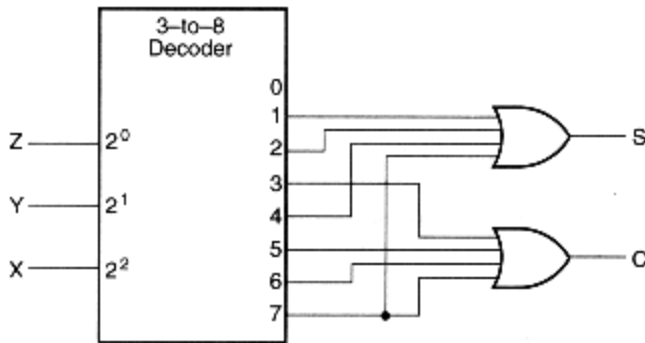
	Inputs			Outputs	
A	B	C_{in}		Carry	Sum
0	0	0		0	0
0	0	1		0	1
0	1	0		0	1
0	1	1		1	0
1	0	0		0	1
1	0	1		1	0
1	1	0		1	0
1	1	1		1	1

Design for 4 to 16 decoder using 3 to 8 decoder: **Design 5 to 32 decoder using 3 to 8 and 2 to 4 decoder:**

5-to-32 line decoder

❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖❖
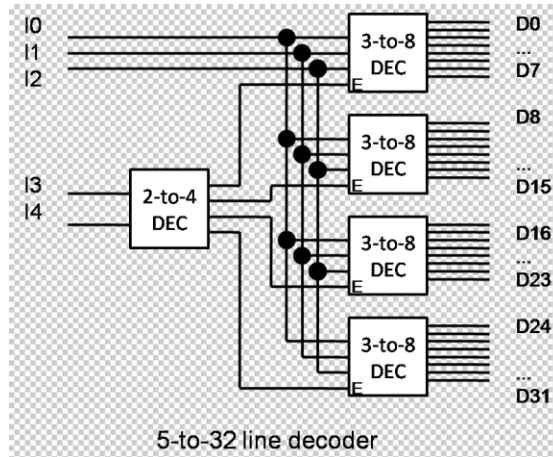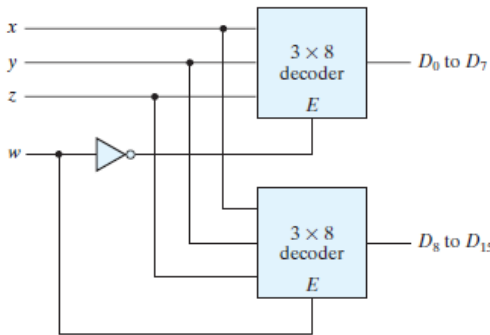
BCD to seven segment decoder
Design a BCD to seven segment code converter. (May-06,10, Nov- 09)
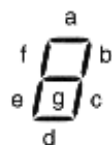
(a) Segment designation   (b) Numeric designation for display
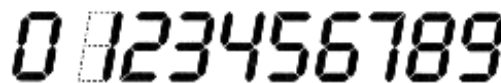
Truth table:

Digit	A	B	C	D	a	b	c	d	e	f	g
	BCD code				**7-Segment code**						
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

K-Map:
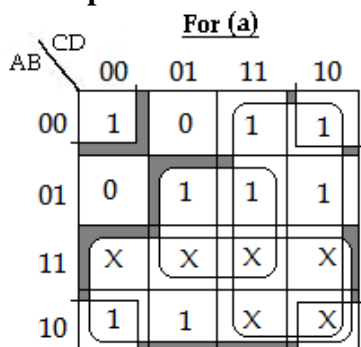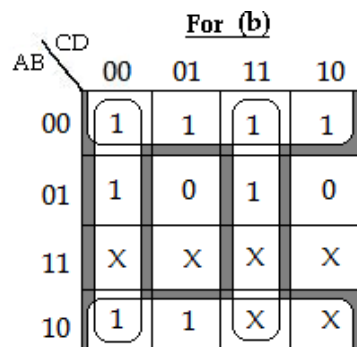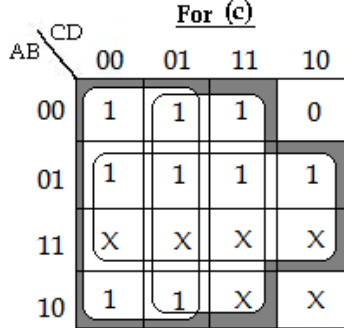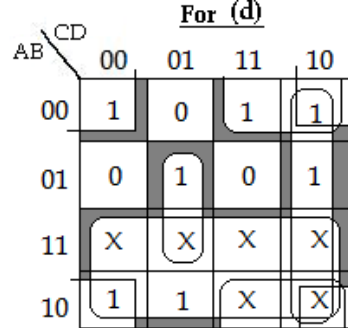
For (a)

$a= A+ C+ BD+ B'D'$

For (b)

$b= B'+ C'D'+ CD$

For (c)

$c= B+ C'+ D$

For (d)

$d= B'D'+ CD'+ BC'D+ B'C+ A$

For (e)

$e= B'D'+ CD'$

For (f)
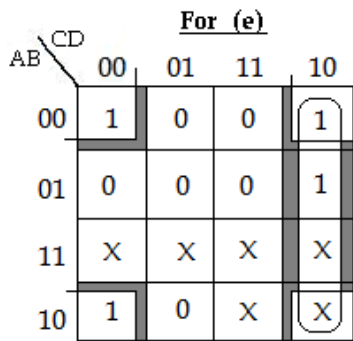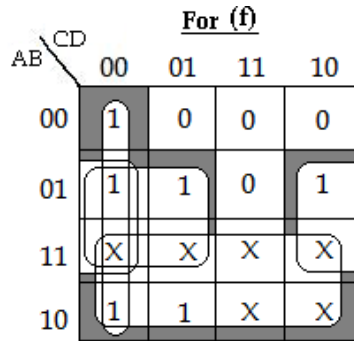
$f= A+ C'D'+ BC'+ BD'$

For (g)

$$g = A + BC' + B'C + CD'$$

Logic Diagram:
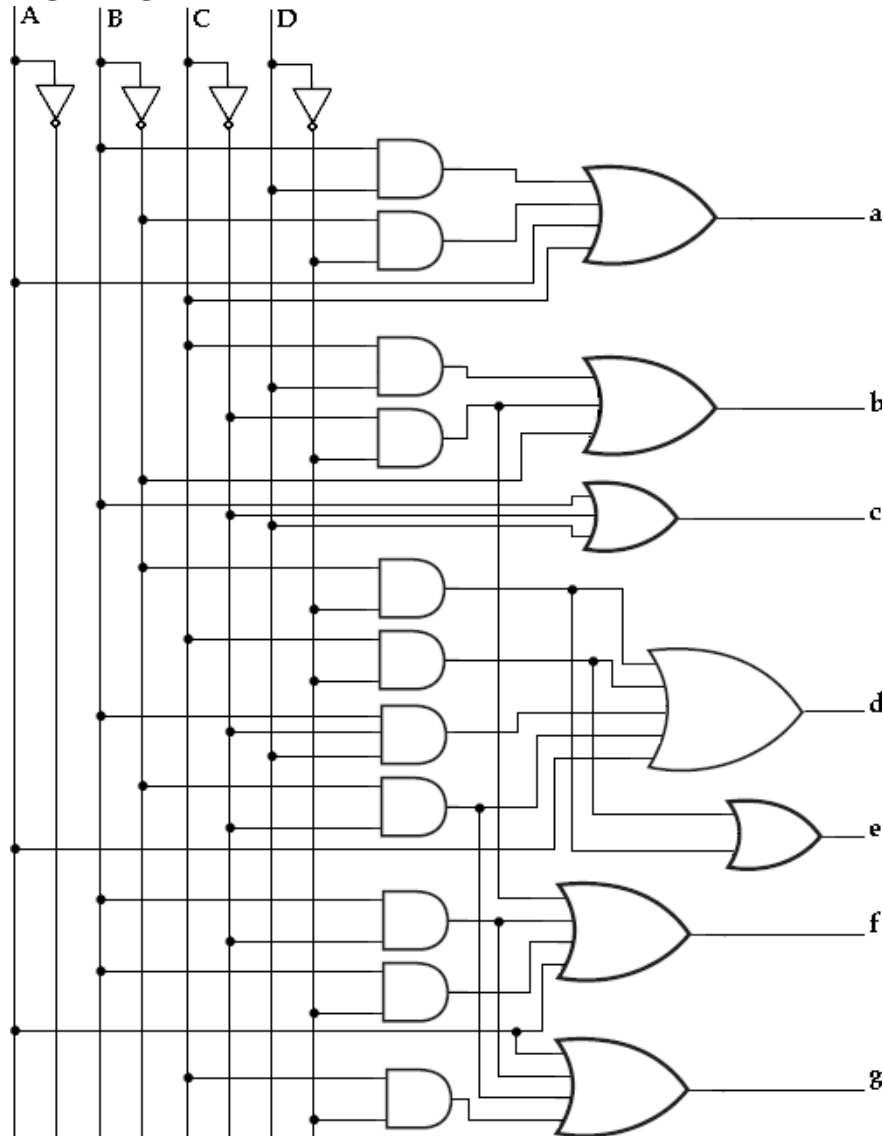
❖ The specification above requires that the output be zeroes (none of the segments are lighted up) when the input is not a BCD digit.

❖ In practical implementations, this may defer to allow representation of hexadecimal digits using the seven segments.

****

❖ An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2^n (or fewer) input lines and n output lines. The output lines, as an aggregate, generate the binary code corresponding to the input value.

Octal to Binary Encoder:

❖ The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output z is equal to 1 when the input octal digit is 1, 3, 5, or 7.

❖ Output y is 1 for octal digits 2, 3, 6, or 7, and output x is 1 for digits 4, 5, 6, or 7. These conditions can be expressed by the following Boolean output functions:

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

The encoder can be implemented with three OR gates.

Truth table:

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

❖ Another ambiguity in the octal-to-binary encoder is that an output with all 0's is generated when all the inputs are 0; but this output is the same as when D0 is equal to 1. The discrepancy can be resolved by providing one more output to indicate whether at least one input is equal to 1.

Logic Diagram:

DO D1 D2 D3 D4 D5 D6 D7

X

Y

Z

****
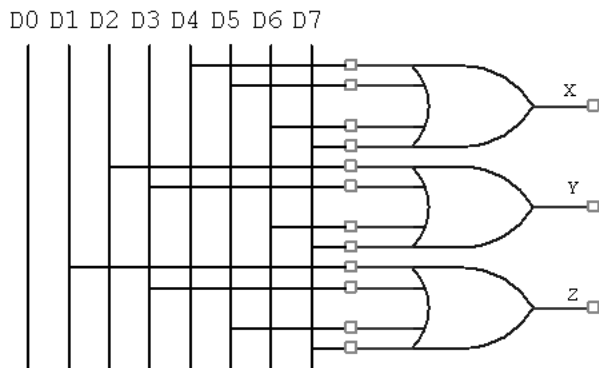
Priority Encoder:

Design a priority encoder with logic diagram.(or) Explain the logic diagram of a 4 – input priority encoder. (Apr – 2019)

A priority encoder is an encoder circuit that includes the priority function. The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

Truth table:

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Modified Truth table:

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	x	x	0
1	0	0	0	0	0	1
0	1	0	0	0	1	1
1	1	0	0			
0	0	1	0			
0	1	1	0	1	0	1
1	0	1	0			
1	1	1	0			
0	0	0	1			
0	0	1	1			
0	1	0	1			
0	1	1	1	1	1	1
1	0	0	1			
1	0	1	1			
1	1	0	1			
1	1	1	1			

K-Map:

D_0D_1 \ D_2D_3 <u>For X</u>

	00	01	11	10
00	X	1	1	1
01	0	1	1	1
11	0	1	1	1
10	0	1	1	1

x= D₂+ D₃

D_0D_1 \ D_2D_3 <u>For y</u>

	00	01	11	10
00	X	1	1	0
01	1	1	1	0
11	1	1	1	0
10	0	1	1	0

y= D₃+ D₁D₂

D_0D_1 \ D_2D_3 <u>For V</u>

	00	01	11	10
00	0	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

V= D₀+ D₁+ D₂+ D₃

Logic Equations:

$$x = D_2 + D_3$$
$$y = D_3 + D_1 \, D_2'$$
$$V = D_0 + D_1 + D_2 + D_3$$

Logic diagram:

****
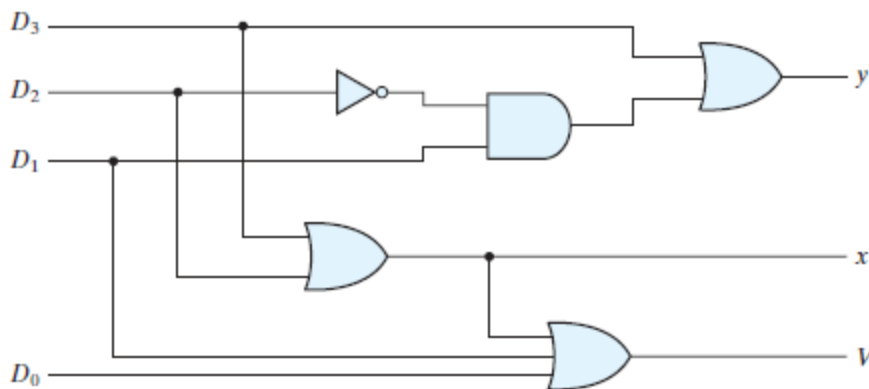
MULTIPLEXERS AND DEMULTIPLEXERS

Multiplexer: (MUX)

Design a 2:1 and 4:1 multiplexer.

❖ A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines.

❖ Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

2 to 1 MUX:

A 2 to 1 line multiplexer is shown in figure below, each 2 input lines A to B is applied to one input of an AND gate. Selection lines S are decoded to select a particular AND gate. The truth table for the 2:1 mux is given in the table below.
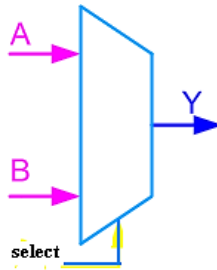
❖ To derive the gate level implementation of 2:1 mux we need to have truth table as shown in figure. And once we have the truth table, we can draw the K-map as shown in figure for all the cases when Y is equal to '1'.

Truth table:

S	Y
0	I_0
1	I_1

Logic Diagram:
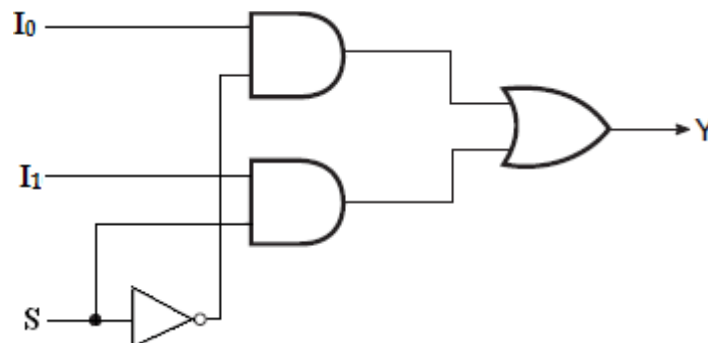
4 to 1 MUX:

❖ A 4 to 1 line multiplexer is shown in figure below, each of 4 input lines I0 to I3 is applied to one input of an AND gate.

❖ Selection lines S0 and S1 are decoded to select a particular AND gate.

❖ The truth table for the 4:1 mux is given in the table below.
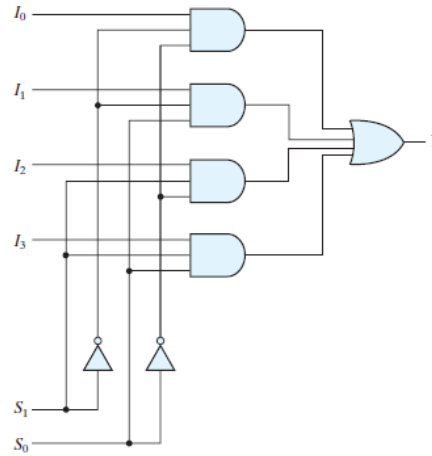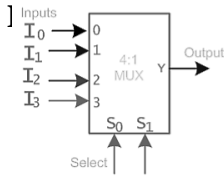
Truth Table:

SELECT INPUT		OUTPUT
S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

Problems :

Example: Implement the Boolean expression using MUX

$$F(A,B,C,D) = \sum m(0,1,5,6,8,10,12,15)$$ **(Apr 2017, Nov 2017)**

Solution : Implementation table :

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{a}$	⓪	①	2	3	4	⑤	⑥	7
a	⑧	9	⑩	11	⑫	13	14	⑮
	1	$\bar{a}$	a	0	a	$\bar{a}$	$\bar{a}$	a
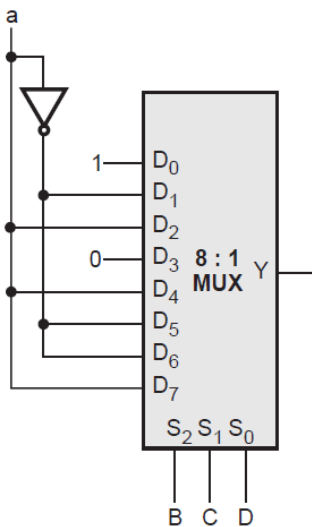
Example: Implement the boolean function using Multiplexer. [NOV – 2019]

F (x, y, z) = Σm (1, 2, 6, 7)

Solution:
Implementation table:

		D_0	D_1	D_2	D_3
	$\bar{z}$	0	1	2	3
	z	4	5	6	7
		0	$\bar{z}$	1	z

Multiplexer Implementation:
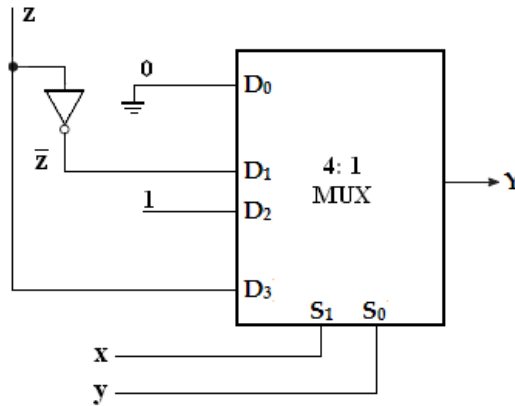
Example: 32:1 Multiplexer using 8:1 Mux (Nov 2018) (Apr – 2019)
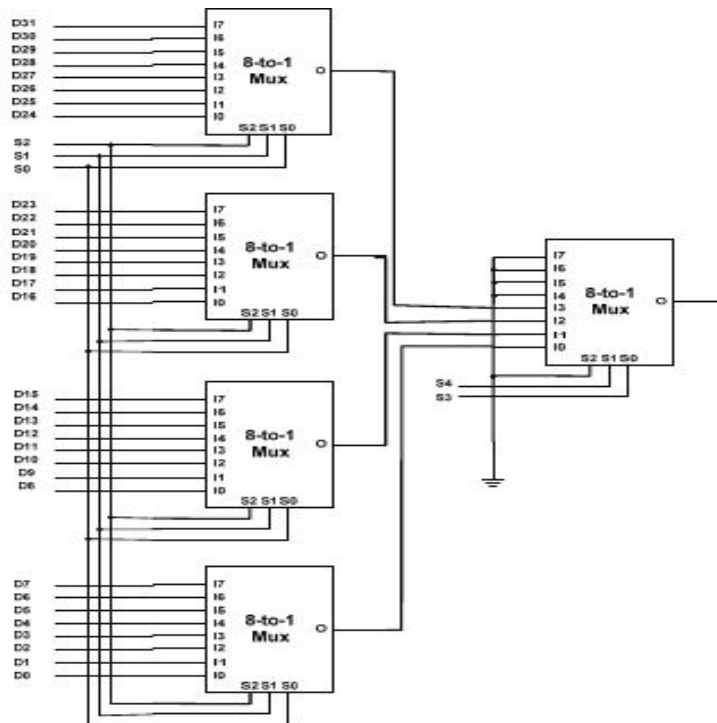
DEMULTIPLEXERS:

Explain about demultiplexers.

- ❖ The de-multiplexer performs the inverse function of a multiplexer, that is it receives information on one line and transmits its onto one of 2n possible output lines.
- ❖ The selection is by n input select lines. Example: 1-to-4 De-multiplexer

Truth table

S1	S0	F0	F1	F2	F3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D
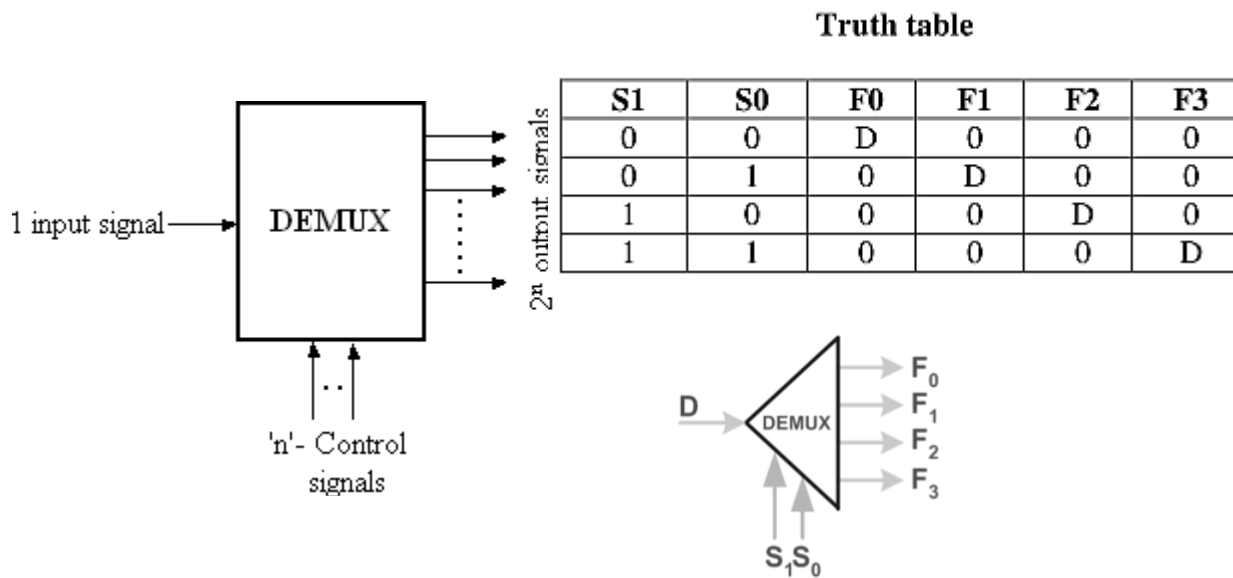
Logic Diagram:
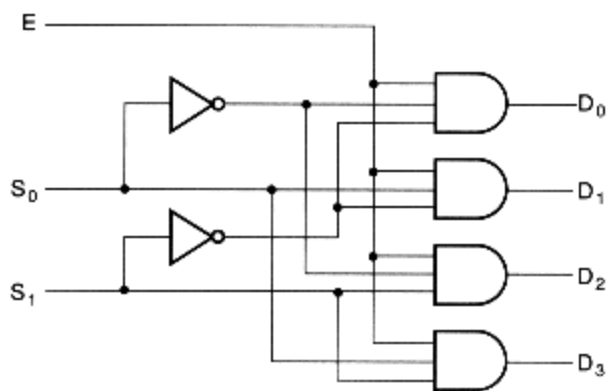
Truth Table:

INPUT				OUTPUT			
E	D	S0	S1	Y0	Y1	Y2	Y3
1	1	0	0	1	0	0	0
1	1	0	1	0	1	0	0
1	1	1	0	0	0	1	0
1	1	1	1	0	0	0	1

Example:

1. Implement full adder using De-multiplexer.

Solution :

Step 1 : Truth table

Inputs			Outputs	
A	B	C_{in}	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Step 2 : For full adder

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

and

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

Step 3 : When $D_{in} = 1$, the demultiplexer gives minterms at the output.
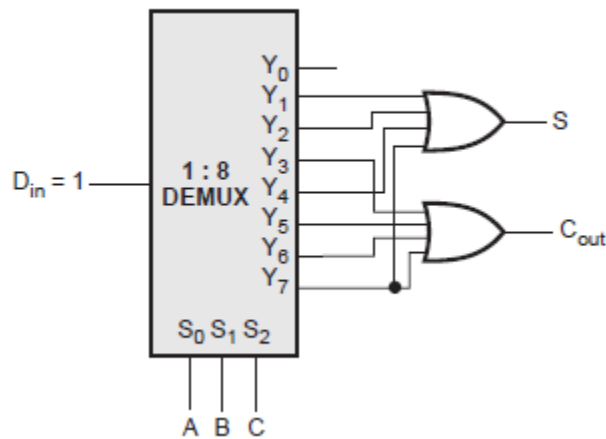
2. Implement the following functions using de-multiplexer.

f1 (A,B,C) = $\sum$m(1,5,7), f2 (A,B,C) = $\sum$m(3,6,7)

Solution:

****
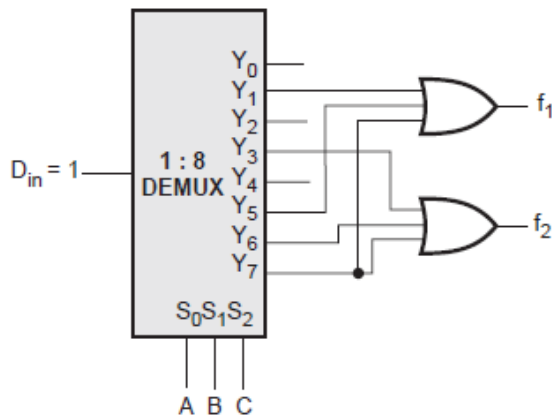
Parity Checker / Generator:

➢ A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even. The message, including the parity bit, is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond with the one transmitted.

➢ The circuit that generates the parity bit in the transmitter is called a *parity generator*. The circuit that checks the parity in the receiver is called a *parity checker*.

➢ In even parity system, the parity bit is '0' if there are even number of 1s in the data and the parity bit is '1' if there are odd number of 1s in the data.

➢ In odd parity system, the parity bit is '1' if there are even number of 1s in the data and the parity bit is '0' if there are odd number of 1s in the data.

3-bit Even Parity generator:
Truth Table:

Three-Bit Message			Parity Bit
x	y	z	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$P = x \oplus y \oplus z$$

Logic Diagram:
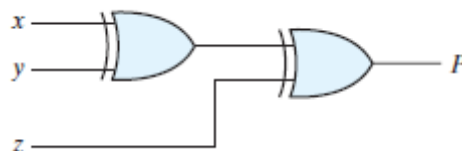
4-bit Even parity checker:
Truth Table:

Four Bits Received				Parity Error Check
x	y	z	P	C
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

$$C = x \oplus y \oplus z \oplus P$$

Logic Diagram:
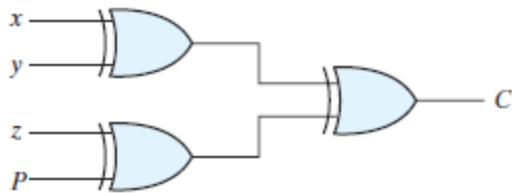
1. Define sequential circuits.

Sequential circuits:

*   Sequential circuits employ storage elements in addition to logic gates. Their outputs are a function of the inputs and the state of the storage elements.
*   Because the state of the storage elements is a function of previous inputs.
*   The outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs.
*   The circuit behaviour must be specified by a time sequence of inputs and internal states.
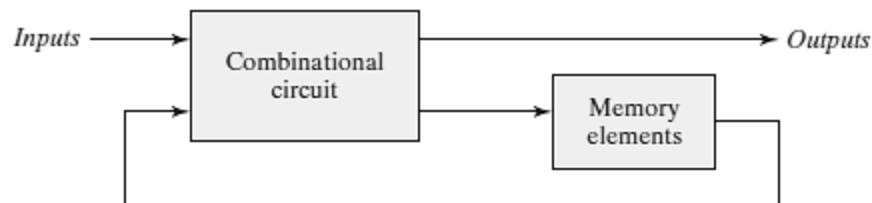
Figure 3.1: Block diagram of sequential circuit/FSM

Types of sequential circuits:

*   There are two main types of sequential circuits, and their classification is a function of the timing of their signals.

    1. **Synchronous sequential circuit:**
        It is a system whose behaviour can be defined from the knowledge of its signals at discrete instants of time.

    2. **Asynchronous sequential circuits:**
        The behaviour of an asynchronous sequential circuit depends upon the input signals at any instant of time and the order in which the inputs change.
        The storage elements commonly used in asynchronous sequential circuits are time-delay devices.

2. Define Flip-Flop.

Flip-Flop:

*   The storage elements used in clocked sequential circuits are called flip-flops.
*   A flip-flop is a binary storage device capable of storing one bit of information. In a stable state, the output of a flip-flop is either 0 or 1.
*   A sequential circuit may use many flip-flops to store as many bits as necessary. The block diagram of a synchronous clocked sequential circuit is shown in Fig.
*   A storage element in a digital circuit can maintain a binary state indefinitely (as long as power is delivered to the circuit), until directed by an input signal to switch states.
*   The major differences among various types of storage elements are in the number of inputs they possess and in the manner in which the inputs affect the binary state.

3.   Define Latch.

Latch:

- The storage elements that operate with signal levels (rather than signal transitions) are referred to as latches those controlled by a clock transition are flip-flops.
- Latches are said to be level sensitive devices.
- Flip-flops are edge-sensitive devices.

(a) Block diagram

(b) Timing diagram of clock pulses
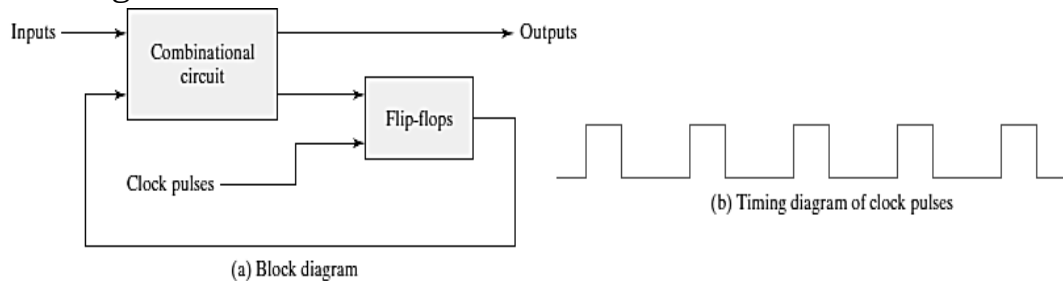
Synchronous clocked sequential circuit

4. Explain about triggering of flip flops in detail.   (Dec-14)

Triggering of Flip Flops:

- The state of a latch or flip-flop is switched by a change in the control input.
- This momentary change is called a trigger, and the transition it causes is said to trigger the flip-flop.

(a) Response to positive level

(b) Positive-edge response

(c) Negative-edge response
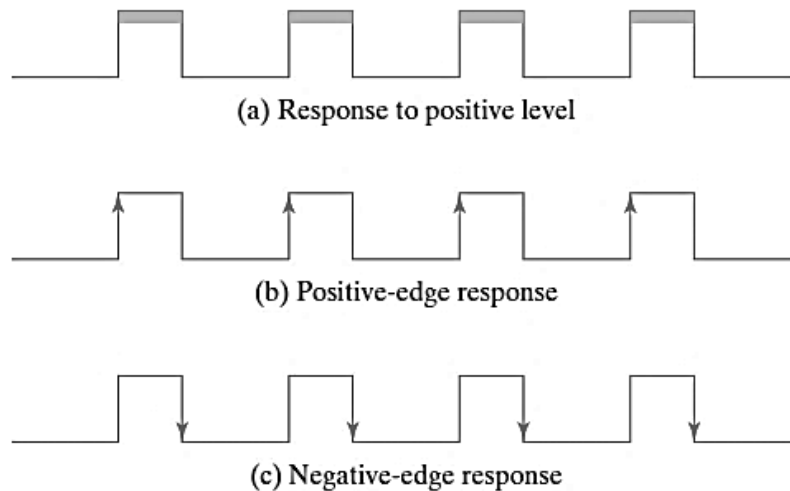
Level Triggering:

- SR, D, JK and T latches are having enable input.
- Latches are controlled by enable signal, and they are level triggered, either positive level triggered or negative level triggered as shown in figure (a).
- The output is free to change according to the input values, when active level is maintained at the enable input.

Edge Triggering:

- A clock pulse goes through two transitions: from 0 to 1 and the return from 1 to 0. As shown in above Fig (b) and (c)., the positive transition is defined as the positive edge and the negative transition as the negative edge.

2

Excitation table for all flip flops:

Q_t	Q_{t-1}	S	R	**D**	J	K	T
0	0	0	X	0	0	X	0
0	1	1	0	1	1	X	1
1	0	0	1	0	X	1	1
1	1	X	0	1	X	0	0

SR latch: It is also called as RS latch.
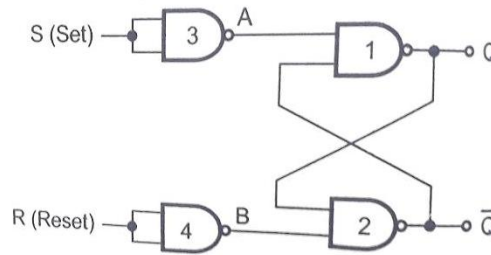
5. **Realize SR Latch using NOR and NAND gates and explain its operation. (Dec-2018)**

- The SR latch is a circuit with two cross-coupled NOR gates or two cross-coupled NAND gates, and two inputs labelled S for set and R for reset.
- The SR latch constructed with two cross-coupled NOR gate is shown in Fig.

(a) Logic diagram
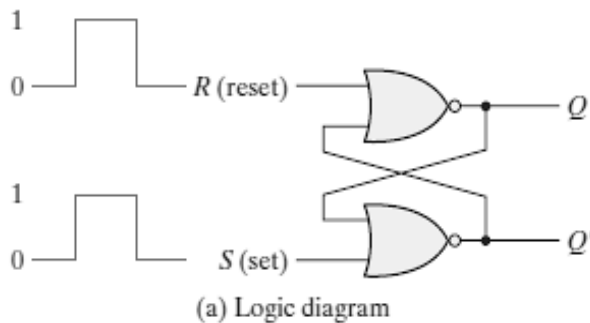
S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(after $S = 1, R = 0$)
0	1	0	1	
0	0	0	1	(after $S = 0, R = 1$)
1	1	0	0	(forbidden)

(b) Function table

- The latch has two useful states. When output Q = 1 and Q'= 0, the latch is said to be in the set state. When Q = 0 and Q' = 1, it is in the reset state.
- Outputs Q and Q' are normally the complement of each other.
- However, when both inputs are equal to 1 at the same time, a condition in which both outputs are equal to 0 (rather than be mutually complementary) occurs.
- If both inputs are then switched to 0 simultaneously, the device will enter an unpredictable or undefined state or a metastable state. Consequently, in practical applications, setting both inputs to 1 is forbidden.

SR Flip-Flop: (Dec-14,15, May-12, 18)

Logic symbol

3

(a) SR flip-flop using NAND gates

(b) SR flip-flop using NOR gates
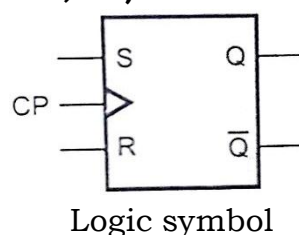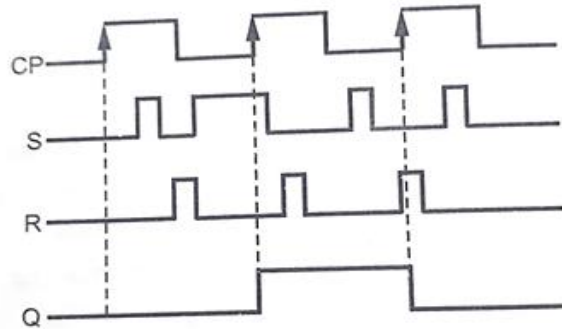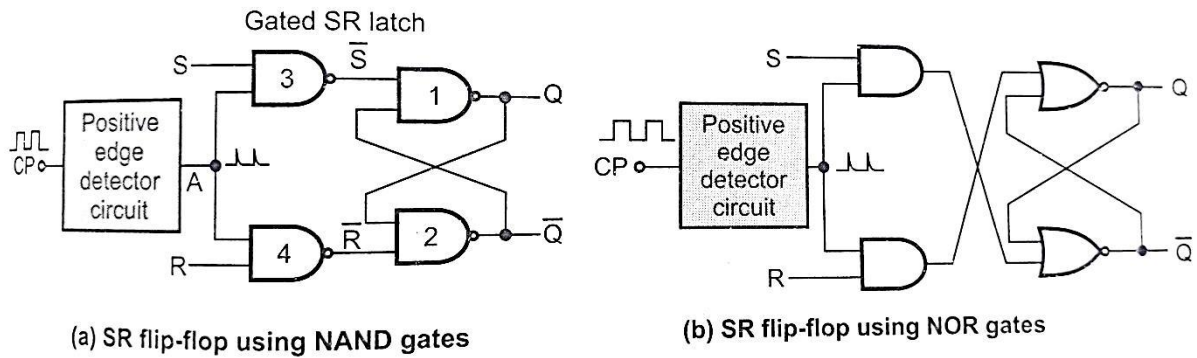
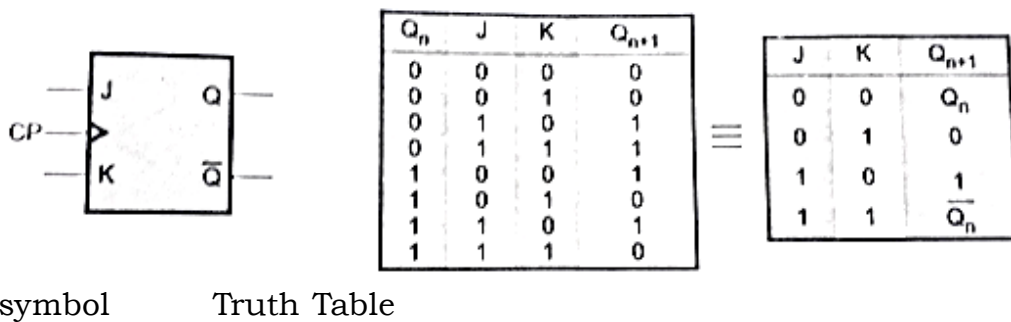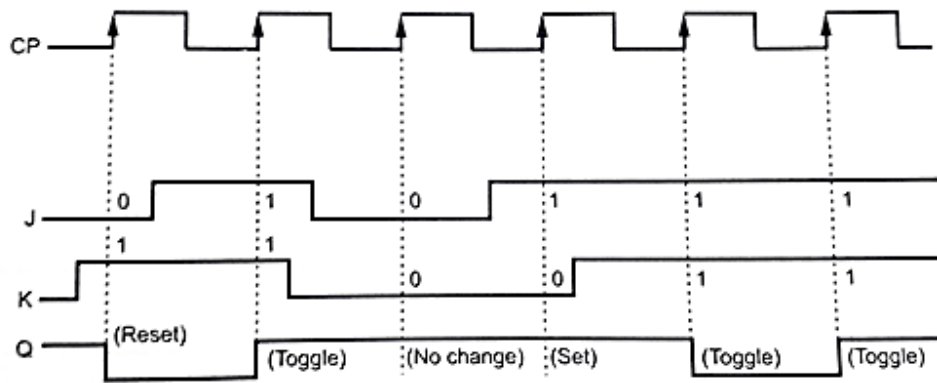Figure 3.2: Input and Output waveform of SR Flip-Flop

D Flip-Flop: (Dec-12)

Logic symbol

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

Truth table of D flip-flop
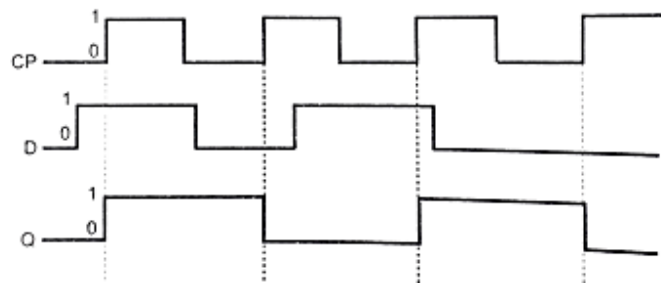
Figure 3.3: Input and output waveform of D Flip-Flop

JK Flip-Flop: (May-08, 18, Dec-12)

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

$\equiv$

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

Logic symbol   Truth Table

4

Input and output waveforms for positive edge triggered JK flip-flop

$S = J\bar{Q}$
$R = KQ$

$\bar{S} = \overline{J\bar{Q}}$
$\bar{R} = \overline{KQ}$

$\bar{S} = \overline{J\bar{Q}}$ $\bar{R} = \overline{KQ}$
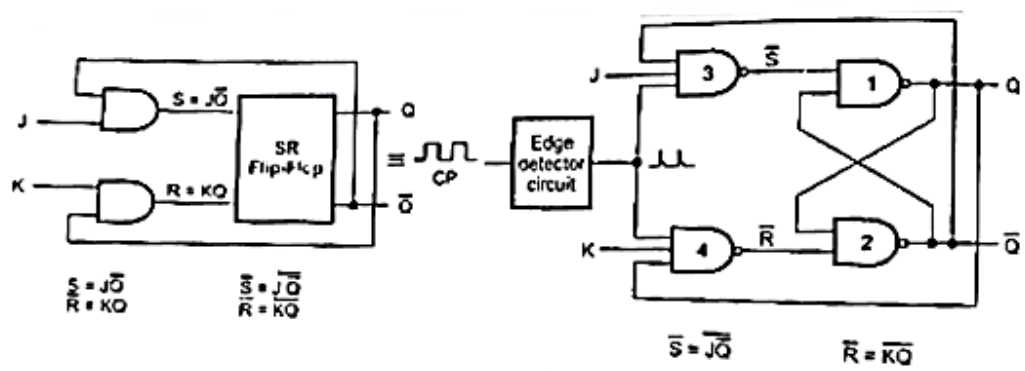
Figure 3.4: JK flip-flop using NAND gates

6. What is race around condition and how is it overcome?   (May-06)

Race-around condition:
- In JK flip-flop, when J = K = 1, the output toggles (output changes either from 0 to 1 or from 1 to 0).
- Consider that initially Q=0 and J = K = 1. After the time interval Δt equal to the propagation delay through two NAND gates in series, the output will change to Q=1 and after another time interval of Δt the output will change back to Q=0.
- This toggling will continue until the flip-flop is enabled and J = K = 1.
- At the end of clock pulse the flip-flop is disabled and the value of Q is uncertain. This situation is referred to as the race-around condition.
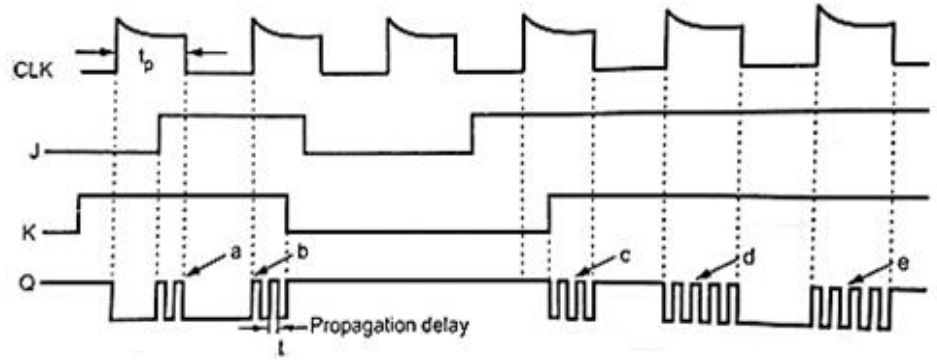
Figure 3.5: Input and output waveform for clocked JK flip-flop

7. **Explain the working operation of master-slave JK Flip-flop.**
   (Dec-2018)(Dec-08, 10, May-15)

Master-Slave JK Flip-flop:

- It consists of clocked JK flip-flop as a master and clocked JK flip-flop as a slave.
- The output of the master flip-flop is fed as an input to the slave flip-flop.
- As shown in figure, the clock signal is connected directly to the master flip-flop, but it is connected through inverter to the slave flip-flop.
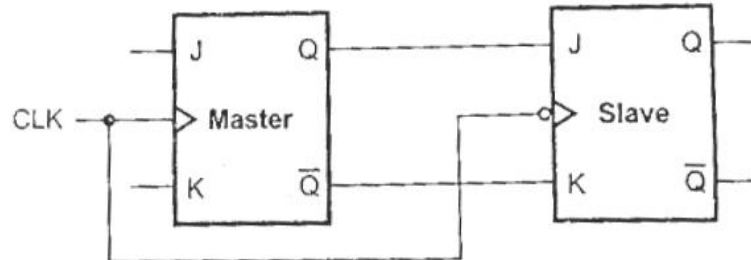
Figure 3.6: Master-Slave JK Flip-flop

- Therefore, the information present at the J and K inputs is transmitted to the output of master flip-flop on the positive clock pulse and it is held there until the negative clock pulse occurs, after which it is allowed to pass through to the output of slave flip-flop.

CLK	J	K	Q_{n+1}
⎍	0	0	Q_n (No change)
⎍	0	1	0 (Reset)
⎍	1	0	1 (Set)
⎍	1	1	$\overline{Q}_n$ (Toggle)

(a) Truth table
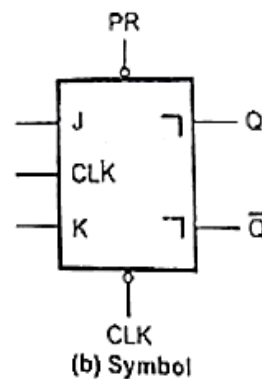
(b) Symbol

T Flip-Flop: (May-18) (Dec-2018)

T Flip-flop is also known as Toggle flip-flop.

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

T	Q_{n+1}
0	Q_n
1	$\overline{Q}_n$
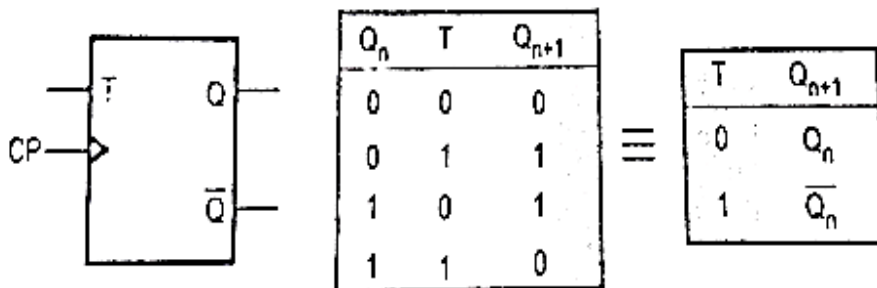
(a) Logic symbol   (b) Truth table

8. Explain various Realization of one Flip-Flop using other Flip-Flop.

1. Design of JK Flip-flop using SR Flip-Flop.

S-R Flip Flop to J-K Flip Flop

Conversion Table

J-K Inputs		Outputs		S-R Inputs	
J	K	Qp	Qp+1	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

Logic Diagram

$S = \overline{J}Q_p$

$R = KQ_n$

2. Design of SR Flip-flop using JK Flip-Flop.

J-K Flip Flop to S-R Flip Flop

Conversion Table

S-R Inputs		Outputs		J-K Inputs	
S	R	Qp	Qp+1	J	K
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	X	1
1	0	0	1	1	X
1	0	1	1	X	0
1	1	Invalid		Dont care	
1	1	Invalid		Dont care	

Logic Diagram

K-maps

J=S   K=R

3. Design of D Flip-flop using SR Flip-Flop.

S-R Flip Flop to D Flip Flop

Conversion Table

D Input	Outputs		S-R Inputs	
	Qp	Qp+1	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

K-maps

$S = D$ $R = \overline{D}$

Logic Diagram

7
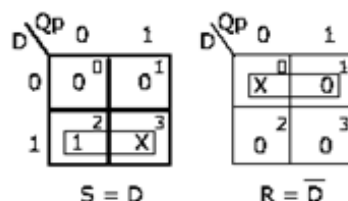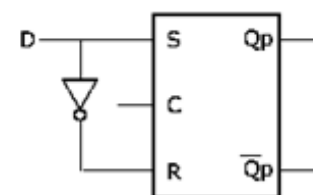
4. Design of SR Flip-flop using D Flip-Flop.

D Flip Flop to S-R Flip Flop

Conversion Table

S-R Inputs		Outputs		D Input
S	R	Qp	Qp+1	
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	Invalid		Dont care
1	1	Invalid		Dont care

K-map

RQp / S	00	01	11	10
0	0	1	0	0
1	1	1	X	X
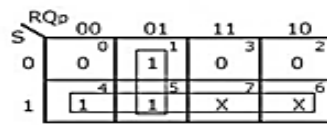
$$D = S + \overline{R}Qn$$
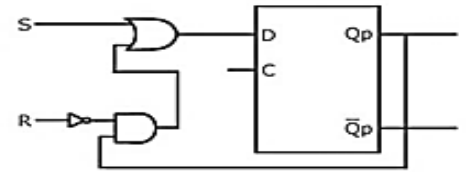
Logic Diagram

5. Design of T Flip-flop using JK Flip-Flop.   (Dec-16)

J-K Flip Flop to T Flip Flop

Conversion Table

T Input	Outputs		J-K Inputs	
	Qp	Qp+1	J	K
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1

K-maps

Qp / T	0	1
0	0	X
1	1	X

$$J = T$$

Qp / T	0	1
0	X	0
1	X	1
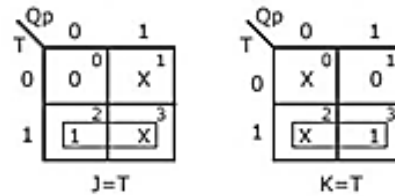
$$K = T$$
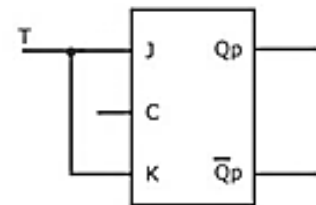
Logic Diagram

6. Design of D Flip-flop using JK Flip-Flop.

J-K Flip Flop to D Flip Flop

Conversion Table

D Input	Outputs		J-K Inputs	
	Qp	Qp+1	J	K
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	0	X	0

K-maps

Qp / D	0	1
0	0	X
1	1	X

$$J = D$$

Qp / D	0	1
0	X	1
1	X	0
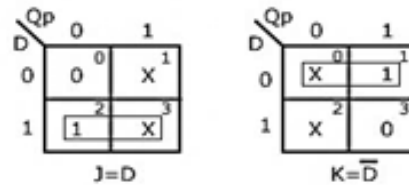
$$K = \overline{D}$$

Logic Diagram
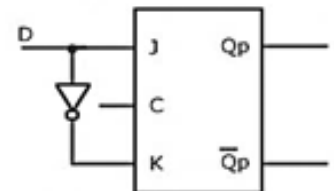
7. Design of JK Flip-flop using D Flip-Flop.   Dec 2009,11

D Flip Flop to J-K Flip Flop
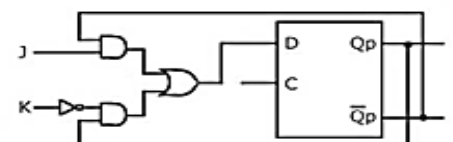
Conversion Table

J-K Input		Outputs		D Input
J	K	Qp	Qp+1	
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

K-map

KQp / J	00	01	11	10
0	0	1	0	0
1	1	1	0	1
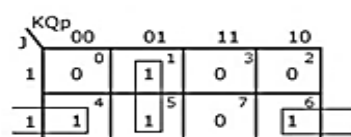
$$D = J\overline{Q}p + \overline{K}Qp$$

Logic Diagram

9. Explain clocked sequential circuits.

8

Clocked sequential circuits:

The states of the output of the flip-flop in the sequential circuit give the state of the sequential circuit.

Present state: The status of all state variables, at some time t, before the next clock edge, represents condition called present state.

Next state: The status of all state variables, at some time, t+1, represents a condition called next state.

The synchronous or clocked sequential circuits are represented by two models.

Moore model: The output depends only on the present state of the flip-flops.
Mealy model: The output depends on both the present state of the flip-flop and on the input.

10. Draw and explain the operation of Moore and Mealy model.

Moore Model:
When the output of the sequential circuit depends only on the present state of the flip-flop, the sequential circuit is referred to as Moore model.
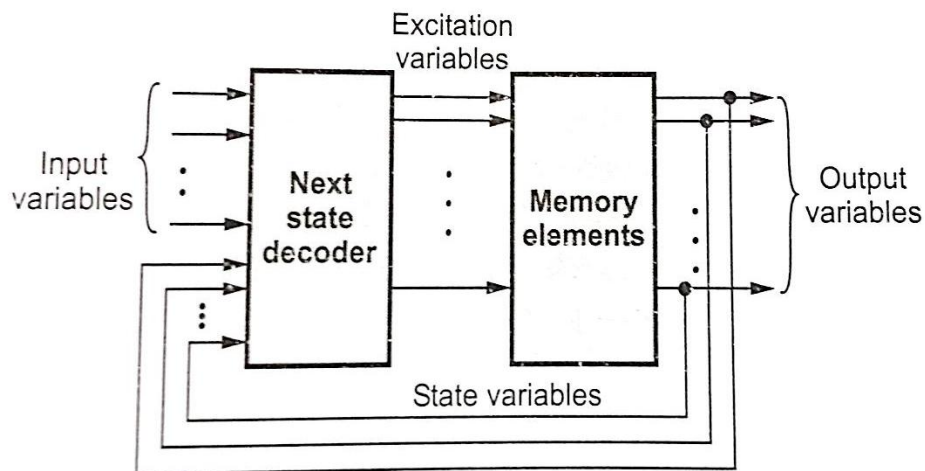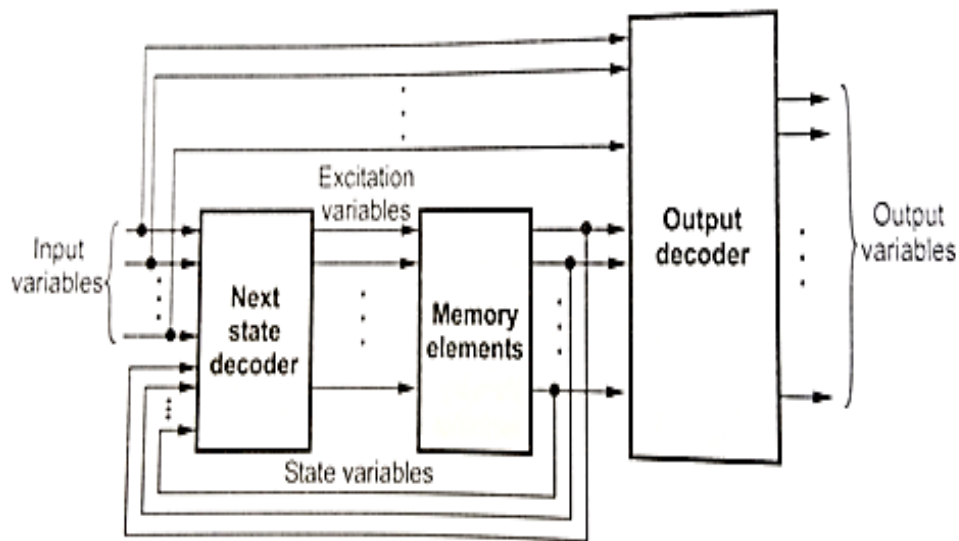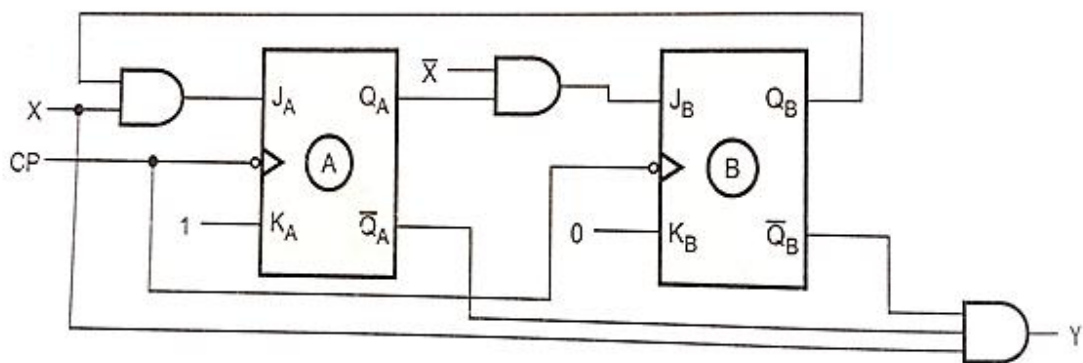

Moore model
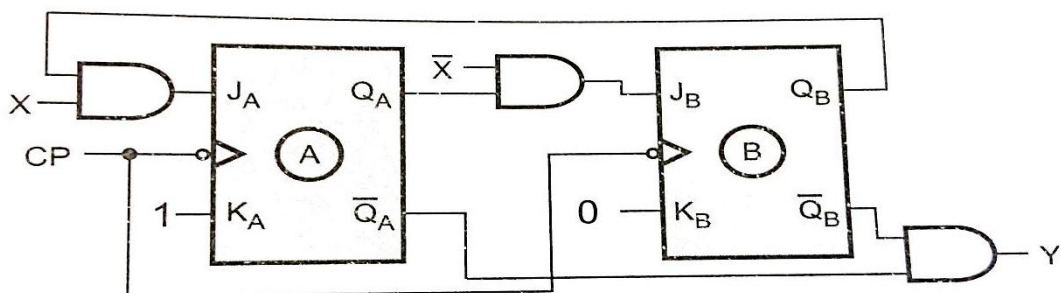
Example of Moore model

Mealy Model:

9

When the output of the sequential circuits depends on both the present state of flip-flop and on the inputs, the sequential circuit is referred to Mealy model.

Mealy circuit model

Example of Mealy model

11. Compare Moore and Mealy circuit Models.   (May-05)

S.NO	Moore model	Mealy model
1.	Its output is a function of present state only.	Its output is a function of present state as well as present input.
2.	Input changes does not affect the output.	Input changes may affect the output of the circuit.
3.	Moore model requires more number of states for implementing same function.	It requires less number of states for implementing same function.

12. What is counter? State the types of counter.   (Dec-08)

Counters:

- A counter is a register capable of counting the number of clock pulse arriving at its clock input.
- Count represents the number of clock pluses arrived.
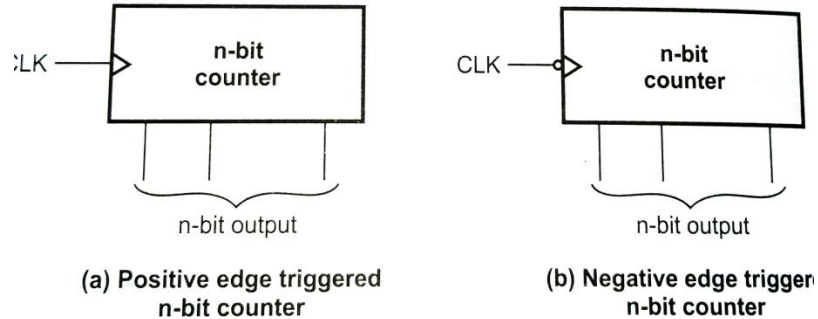- On arrival of each clock pulse, the counter is incremented by one.

Figure 3.2: Logic symbol of counter

Types of counters

- ➢ Synchronous Counter
- ➢ Asynchronous Counter / Ripple Counter

Synchronous Counter:

- When counter is clocked such that each flip-flop in the counter is triggered at the same time, the counter is called Synchronous Counter.

Asynchronous Counter / Ripple Counter:

- A binary asynchronous/ ripple counter consists of a series connection of complementing flip-flop, with the output of each flip-flop connected to a clock inputs of the next higher order flip-flop.
- The flip-flop holding the least significant bit receives the incoming clock pulses.

What is MOD counter?

Module of counter

- The total number of counts or stable states a counter can indicate is called 'Modulus'.
- The modulus of a four-stage counter would be 16_{10}, since it is capable of indicating 0000_2 to1111_2.
- The term 'modulo' is used to describe the count capability of counters.

13. Explain in detail about ripple counter. (Dec-09, 14)

Ripple/Asynchronous counters:

- A binary ripple/asynchronous counter consist of a series connection of complementing flip-flops, with the output of each flip-flop connected to the clock input of the next higher-order flip-flop.
- The flip-flop holding the least significant bit receives the incoming clock pulses.
- A complementing flip-flop can be obtained from a JK flip-flop with the J and K inputs tied together as shown in figure 3.3

11

- A third alternate is to use a D flip-flop with the complement output connected to the D input.
- The D input is always the complement of the present state and the next clock pulse will cause the flip-flop to complement.
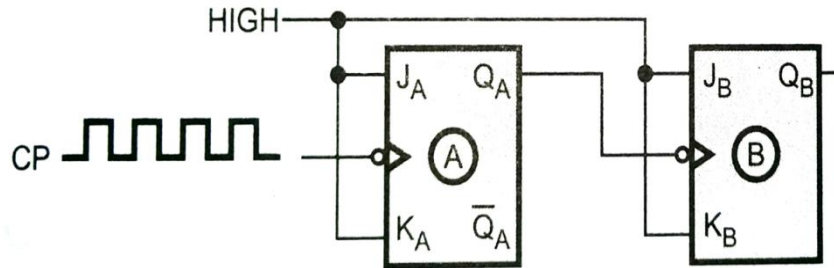

Figure 3.3: A two-bit asynchronous binary counter

- The clock signal is connected to the clock input of only first stage flip-flop.
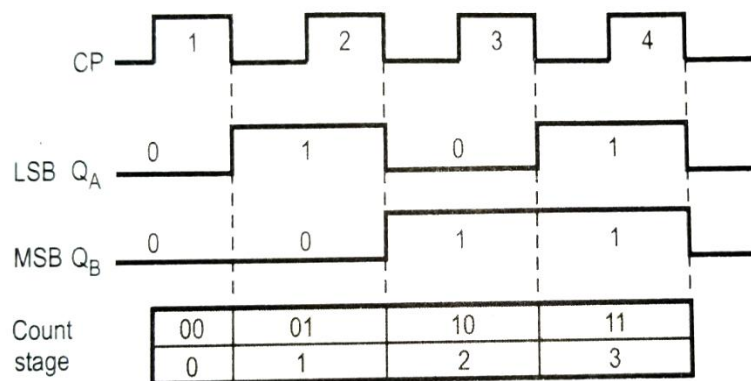

Figure 3.4: Timing diagram for the counter of two-bit asynchronous counter

- It illustrates the changes in the state of the flip-flop outputs in response to the clock.
- J and K input of JK flip-flops are tied to logic HIGH hence output will toggle for each negative edge of the clock input.

14. **Extend the 2-bit asynchronous binary counter for 3-stages, and draw output waveforms.**

3-bit asynchronous counter:
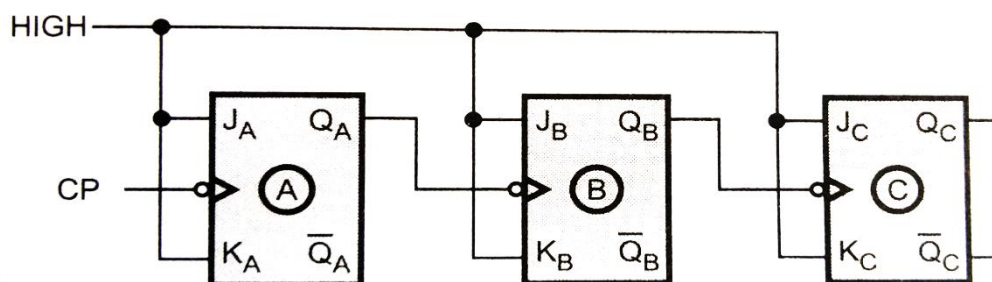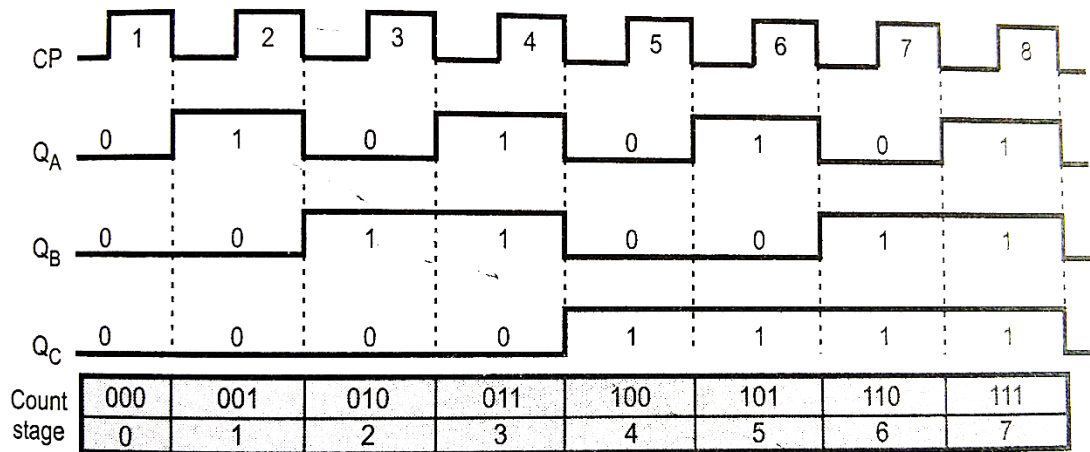**Solution:**

Figure 3.5: Logic diagram

Figure 3.6: Output waveform for 3-bit asynchronous counter

- In timing diagram for 3-bit asynchronous counter, we have not considered the propagation delays of flip-flops, for simplification.
- If we consider the propagation delay, we see that the propagation delay of the first stage is added in the propagation delay of second stage to decide the transition time of the third stage.
- This cumulative delay of the asynchronous counter is the major disadvantage in many applications because it limits the rate at which the counter can be clocked and creates decoding problems.

15. **Draw the logic diagram for 4-stage asynchronous counter with positive edge triggered flip-flops. Also draw necessary timing diagram. Is there any frequency division concept in it.** (Dec-09)

4-stage asynchronous counter with positive edge triggered:

- When flip-flops are positive edge triggered, the $\overline{Q}$ output of previous stage is connected to the clock input of the next stage.
- Figure 3.7 shows the 4-stage asynchronous counter with positive edge triggered flip-flops.
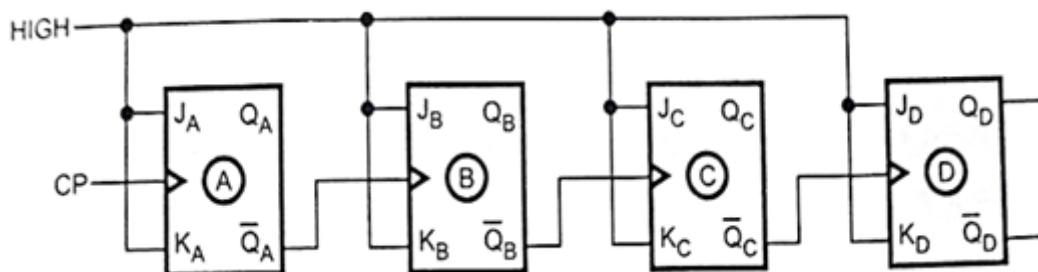

Figure 3.7: Logic diagram of 4-stage positive edge triggered ripple counter.

- The figure 3.8Shows the timing diagram for 4-bit ripple up counter using positive edge triggered JK-FFs.
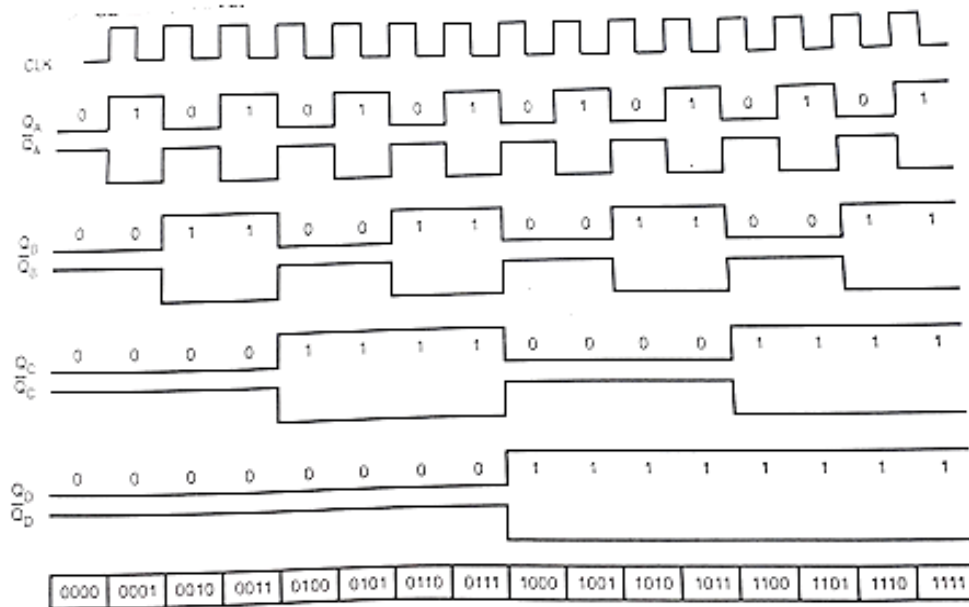
Figure 3.8: Timing diagram of the 4-bit counter

From the timing diagram we can observe that

- Frequency at output $Q_A = \frac{F_{CLK}}{2}$

- Frequency at output $Q_B = \frac{Q_A}{2} = \frac{F_{CLK}}{4}$

- Frequency at output $Q_C = \frac{Q_B}{2} = \frac{Q_A}{4} = \frac{F_{CLK}}{8}$

- Frequency at output $Q_D = \frac{Q_C}{2} = \frac{Q_B}{4} = \frac{Q_A}{8} = \frac{F_{CLK}}{16}$

- In general we say that the frequency at the MSB output of counter is $\frac{F_{CLK}}{2^N}$

- where N represent number of stages/bits of the counter.

16. **Design an asynchronous down counter using JK flip-flop. (Dec-14)**
Asynchronous/Ripple down counter:

- The down counter will count downward from a maximum count to zero.
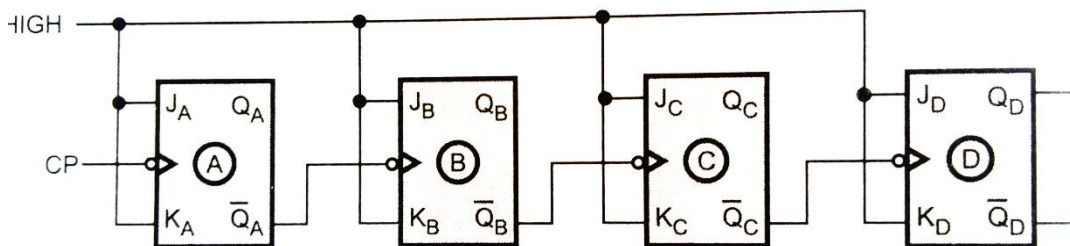

Figure 3.9: 4-bit asynchronous down counter using JK flip-flop

- The clock signal is connected to the clock input of only first flip-flop.
- The clock input of the remaining flip-flop is triggered by the $\overline{Q_A}$ output of the previous stage instead of Q_A output of the previous stage.

14

- The figure 3.10 shows the timing diagram for 4-bit asynchronous down counter. It illustrates the changes in the state of the flip-flop outputs in response to the clock.
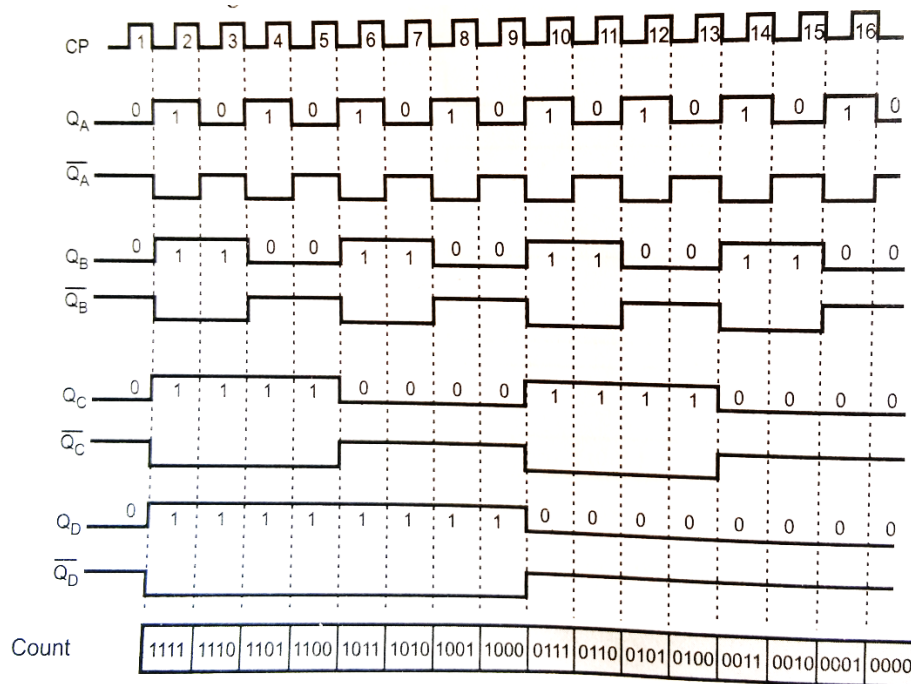

Figure 3.10: Timing diagram of the 4-bit asynchronous down counter.

- The J and K inputs of JK flip-flops are tied to logic HIGH hence output will toggle for each negative edge of the clock input.

17. **Design and explain the working of an up-down ripple counter. (May-03)**

Asynchronous Up/Down counters:
- To form an asynchronous up/down counter one control input say M is necessary to control the operation of the up/down counter.
- When M=0, the counter will count up and when M=1, the counter will count down.
- To achieve this M input should be used to control whether the normal flip-flop output (Q) or the inverted flip-flop output $\overline{(Q)}$ is fed to drive the clock signal of the successive stage flip-flop, as shown in the figure 3.11

(a) The block diagram of combinational circuit

(b) Truth table

a) K-map simplification

$Y = \overline{M}\,\overline{Q} + MQ$
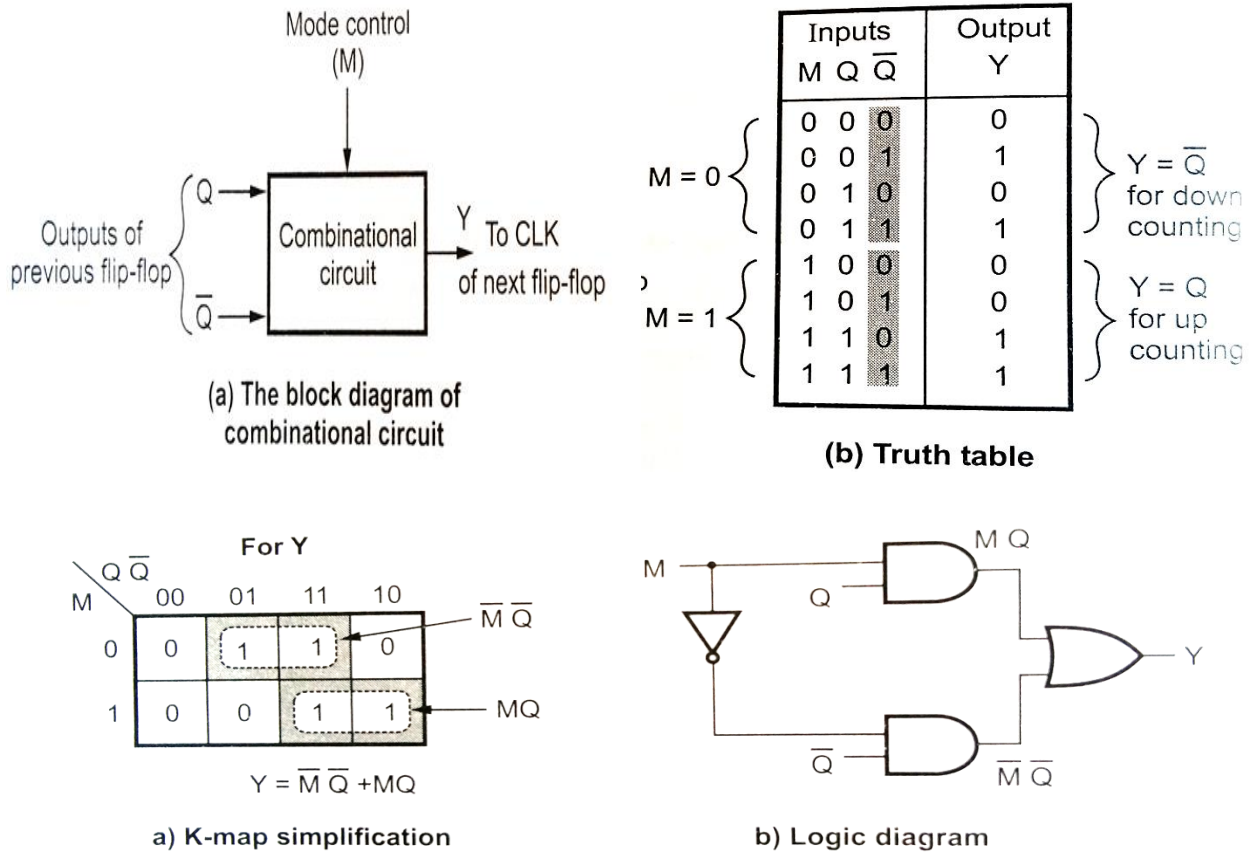
b) Logic diagram

Figure 3.11: Asynchronous up/down counter.

- Figure 3.12 shows the 3-bit up/down counter that will count from 000 up to 111. When mode control input M is 1 and from 111 down to 000 when mode control input M is 0.
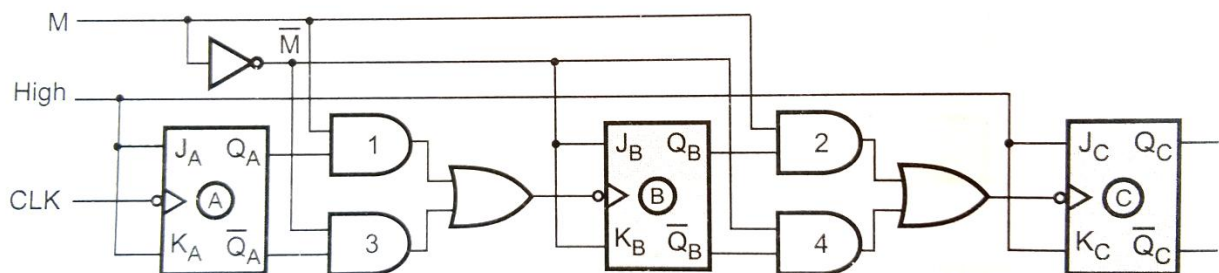
Figure 3.12: 3-bit asynchronous up/down counter

- Logic 1 on M enables AND gates 1 and 2 and disables AND gates 3 and 4. This allows the Q_A and Q_B outputs to drive the clock inputs of their respective next stages. So the counter will count up.
- When M is logic 0, AND gates 1 and 2 are disabled and AND gate 3 and 4 are enabled. This allows the $\overline{Q_A}$ and $\overline{Q_B}$ outputs to drive the clock inputs of their respective next stages so that counter will count down.

$UP/\overline{DOWN} = 1$

16
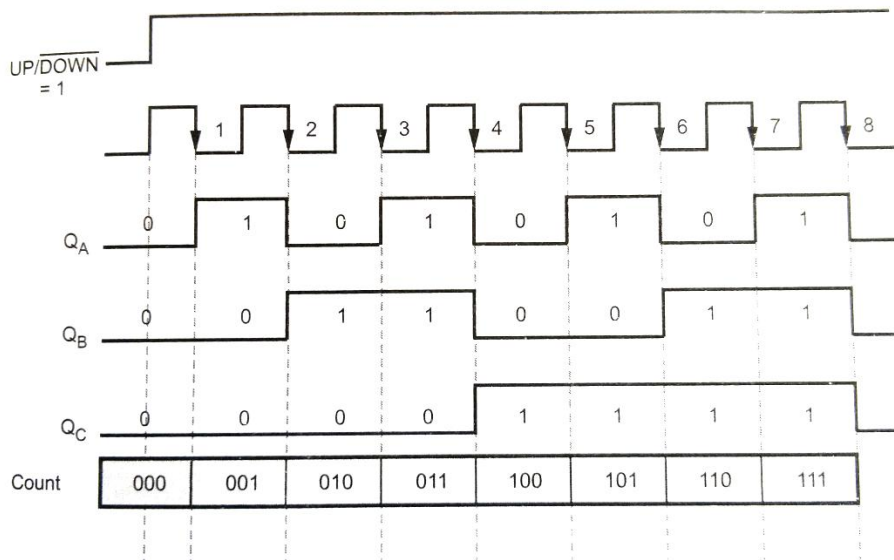
Figure 3.13: Timing diagram for 3-bit up/down ripple counter.

18. Design a 4-bit up/down ripple counter with a control for up/down counting.

Solution:

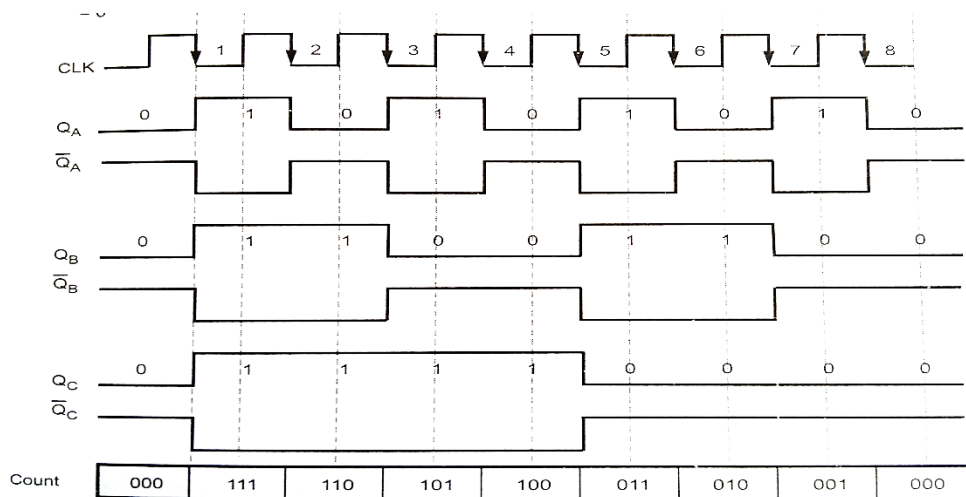- The 4-bit counter needs four flip-flops. The circuit for 4-bit up/down ripple counter is similar to 3-bit up/down ripple counter except that 4-bit counter has one more flip-flop and its clock driving circuiting.
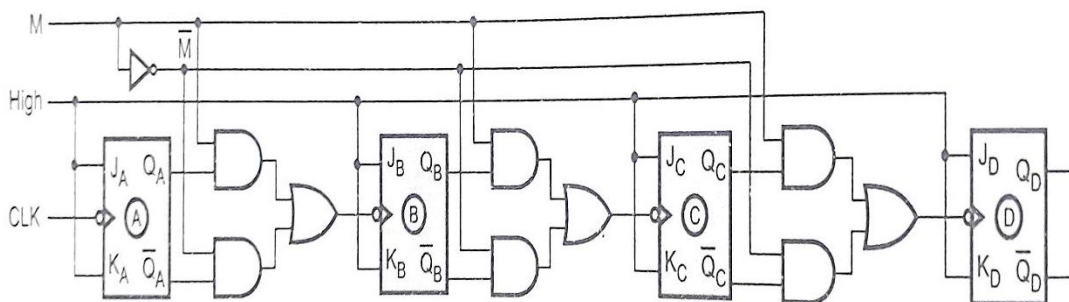

Figure 3.14: 4-bit asynchronous up/down counter.

Design of Ripple (asynchronous) counters:
Steps involved in the design of asynchronous counter
1. Determine the number of flip-flops needed.
2. Choose the type of flip-flops to be used: T or JK. If T flip-flops are used connect T input of all flip-flops to logic 1. If JK flip-flops are used connect both J and K inputs of all flip flops to logic 1.
   Such connection toggles the flip-flop output on each clock transition.
3. Write the truth table for the counter.
4. Derive the reset logic by K-map simplification.
5. Draw the logic diagram.

19. Design BCD ripple counter using JK flip-flop. (May-10,11)
Solution:
Step 1: Determine the number of flip-flops needed. The BCD counter goes through states 0-9, i.e. total 10 states. Thus, N=10 and for $2^n \geq N$, we need n=4, i.e. 4 flip-flops required.

Step 2: Type of flip-flops to be used : JK

Step 3: Write the truth table for the counter.

CLK	A	B	C	D	Output of reset logic Y	
0	0	0	0	0	1	
1	0	0	0	1	1	
2	0	0	1	0	1	
3	0	0	1	1	1	
4	0	1	0	0	1	Valid states
5	0	1	0	1	1	
6	0	1	1	0	1	
7	0	1	1	1	1	
8	1	0	0	0	1	
9	1	0	0	1	1	
–	1	0	1	0	0	
–	1	0	1	1	0	
–	1	1	0	0	0	Invalid states
–	1	1	0	1	0	
–	1	1	1	0	0	
–	1	1	1	1	0	

Truth table for BCD counter

Step 4: Derive reset logic

18

Karnaugh map for $Y = \overline{A} + \overline{B}\,\overline{C}$

$$Y = \overline{A} + \overline{B}\,\overline{C}$$

Step 5: Draw logic diagram

Reset logic
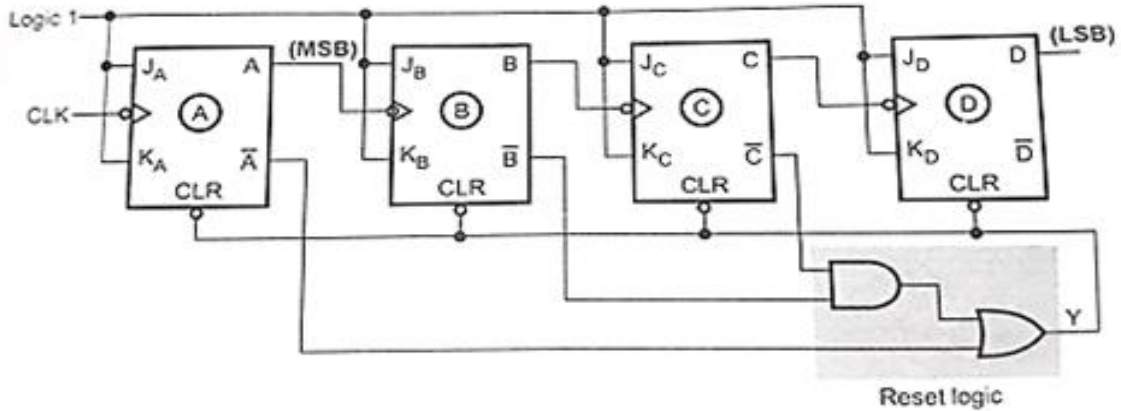
20. Define synchronous counter and various types of synchronous counter.

Synchronous counters:
- When counter is clocked such that each flip-flop in the counter is triggered at the same time, the counter is called as synchronous counter.

2-bit synchronous Binary Up counter:
- Clock signal is connected in parallel to clock inputs of both the flip-flops.

Figure 3.15: A two-bit synchronous binary counter

- The Q_A output of the first stage is used to drive the J and K inputs of the second stage.
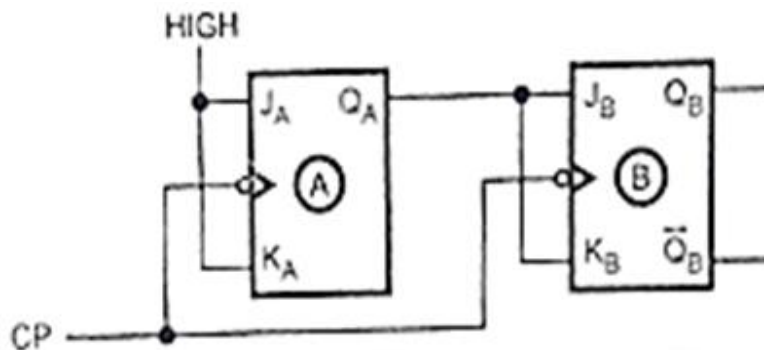- Initially, we assume that the $Q_A = Q_B = 0$.

19

- When positive edge of the first clock pulse is applied, flip-flop A will toggle because $J_A = K_A = 1$, whereas flip-flop B output will remain zero, because JB = KB = 0.
- After first clock pulse $Q_A = 1$ and $Q_B = 0$.
- At negative going edge of the second clock pulse both flip-flops will toggle because they both have a toggle condition on their J and K inputs ($J_A = K_A = J_B = K_B = 1$). Thus after second clock pulse, $Q_A = 0$ and $Q_B = 1$.
- At negative going edge of the third clock pulse flip-flop A toggles making $Q_A = 1$, but flip-flop B remains set i.e. $Q_B = 1$.
- Finally, at the leading edge of the fourth clock pulse both flip-flops toggle as their JK inputs are at logic 1. This results $Q_A = Q_B = 0$ and counter recycled back to its original state.

CP	Q_B	Q_A
0	0	0
1	0	1
2	1	0
3	1	1
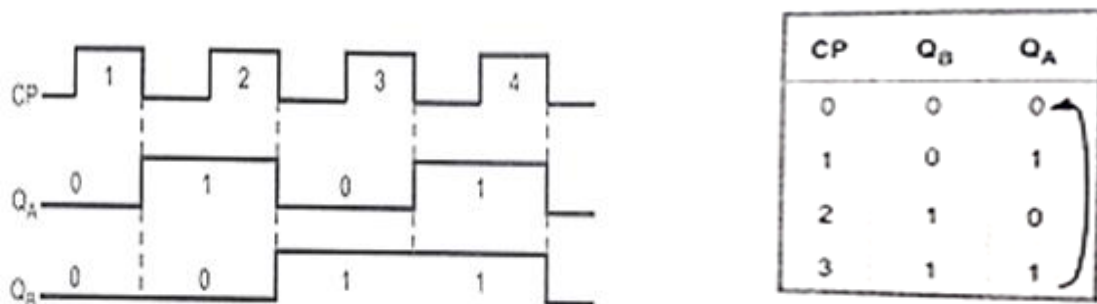
Figure 3.16: Timing diagram and state sequence for the 2-bit synchronous counter.

21. Explain the working of 3-bit synchronous binary up counter.   (May-08)
3-bit synchronous Binary Up counter:

Figure 3.17: A three-bit synchronous binary counter

- We see that Q_A changes on each clock pulse as we progress from its original state to its original state to its final state and then back to its original state.
- Flip-flop A is held in the toggle mode by connecting J and K inputs to HIGH.
- Flip-flop B toggles, when Q_A is 1.
- When Q_A is 0, flip-flop B is in the no-change mode and remains in its present state.

Figure 3.18: Timing diagram

- Looking at the table we can notice that flip-flop C has to change its state only when Q_B and Q_A both are at logic 1.
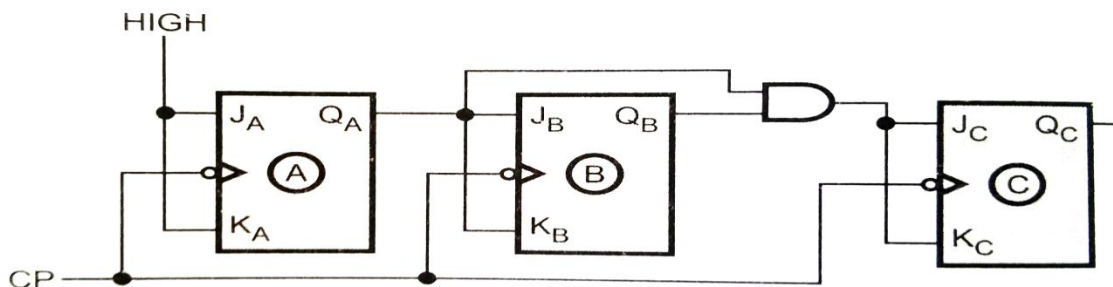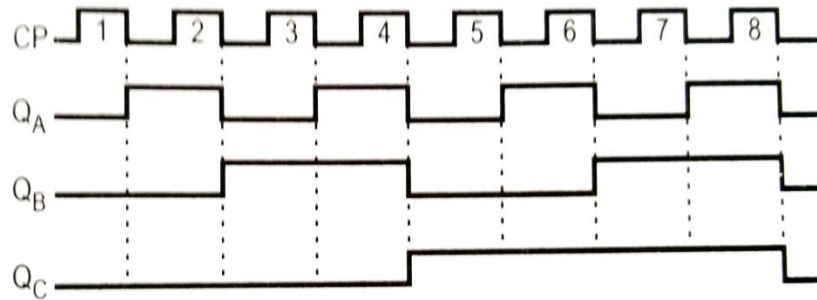- This condition is detected by AND gate and applied to the J and K inputs of flip-flop C. whenever both Q_A and Q_B are HIGH, the output of the AND gate makes the J and K inputs of flip-flop C HIGH and flip-flop C toggles on the following clock pulse.
- At all other times, the J and K inputs of flip-flop C are held LOW by the AND gate output and flip-flop does not change state.

CP	Q_C	Q_B	Q_A
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

State sequence

4-bit synchronous Binary up counter:

Figure 3.19: Logic diagram for 4-bit synchronous binary counter.
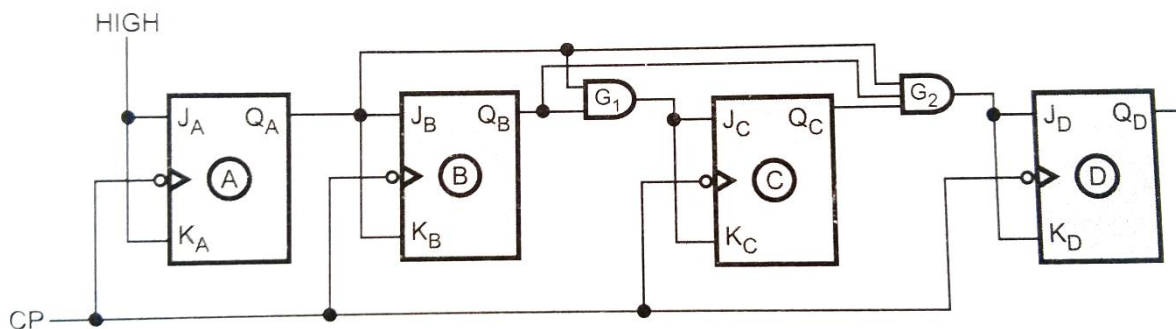
- As counter is implemented with negative edge triggered flip-flops, the transitions occur at the negative edge of the clock pulse.
- In this circuit, first three flip-flops work same as 3-bit counter discussed previously.
- For the fourth stage, flip-flop has to change the state when $Q_A = Q_B = Q_C = 1$. This condition is decoded by 3-input AND gate G_2.

21

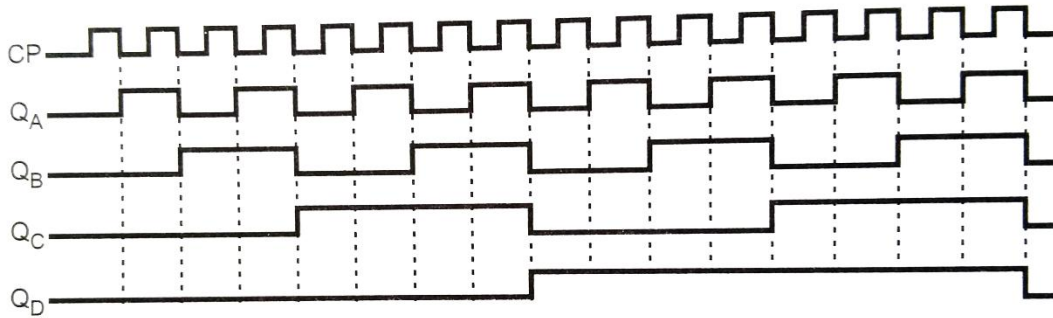- Therefore, when $Q_A = Q_B = Q_C = 1$, flip-flop D toggles and for all other times it is in no change condition.

Figure 3.20: A four bit synchronous binary counter and timing diagram

22. Explain synchronous up/down counters.

Synchronous Down and up/down counters:

- A parallel/synchronous down counter can be constructed by using the inverted FF outputs to drive the following JK inputs.
- For example, the parallel up counter can be converted to a down counter by connecting the $\overline{Q_A}$, $\overline{Q_B}$, $\overline{Q_C}$ and $\overline{Q_D}$ outputs in place of Q_A, Q_B, Q_C and Q_D respectively.
- The counter will then proceed through the following sequence as input pulses are applied:
- To form a parallel up/down counter the control input (UP/$\overline{DOWN}$) is used to control whether the normal flip-flop outputs or the inverted flip-flop outputs are fed to the J and K inputs of the following flip-flops.
- A logic 1 on the UP/$\overline{DOWN}$ enables AND gates 1 and 2 and disables AND gates 3 and 4. This allows the QA and QB outputs through to the J and K inputs of the next flip-flops so that the counter will count up as pulses are applied.
- When UP/$\overline{DOWN}$ line is logic 0, AND gates 1 and 2 are disables and AND gates 3 and 4 are enabled. This allows the $\overline{Q_A}$ and $\overline{Q_B}$ outputs through to the J and K inputs of the next flip-flops so that the counter will count down as pulses are applied.
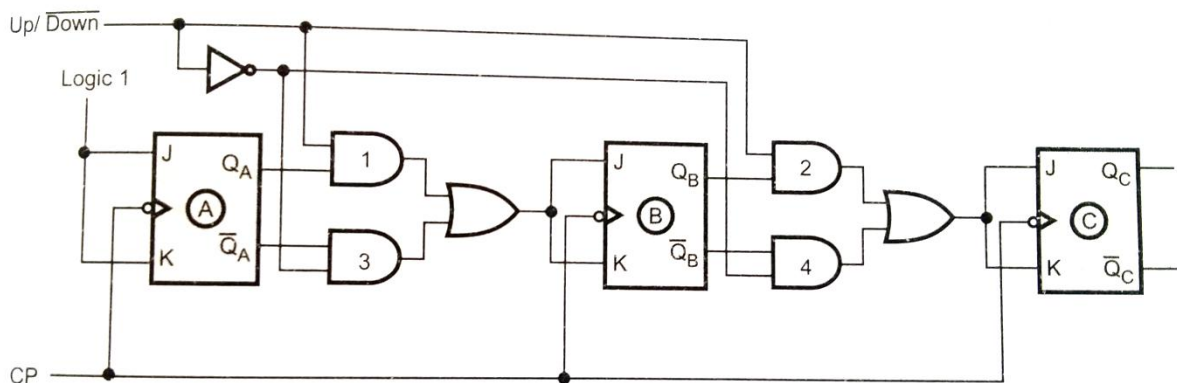

Figure 3.21: 3-bit synchronous/parallel up/down counter
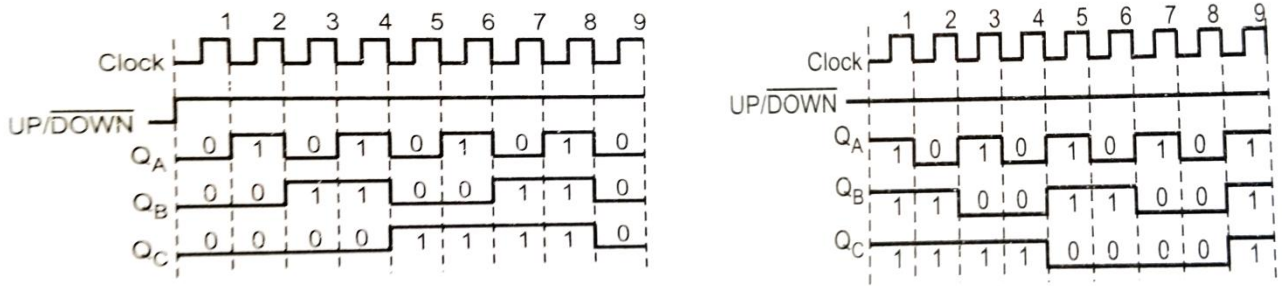
Figure 3.22: Timing diagram for 3-bit up-down counter

Design of synchronous counters:

1. Determine the number of flip-flops needed. If n represents number of flip-flops $2^n \geq$ number of states in the counter.
2. Choose the type of flip-flops to be used.
3. Using excitation table for selected flip-flop determine the excitation table for the counter.
4. Use K-map or any other simplification method to derive the flip-flop input functions.
5. Draw the logic diagram.

Ex.1: Design a MOD-5 synchronous counter using JK flip-flops and implement it. Also draw the timing diagram.   (May-15)

Solution:

Step 1: Determine the number of flip-flop needed

Flip-flops required are $2^n \geq N$.

Hence, N=5 ∴ n=3 i.e. three flip-flops are required.

Step 2: Type of flip-flop to be used : JK

Step 3: Determine the excitation table for the counter.

Present state			Next state			Flip-flop inputs					
A	**B**	**C**	**A⁺**	**B⁺**	**C⁺**	**J_A**	**K_A**	**J_B**	**K_B**	**J_C**	**K_C**
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	1	0	X	1	X	X	0
0	1	0	X	X	X	X	X	X	X	X	X
0	1	1	1	1	1	1	X	X	0	X	0
1	0	0	0	0	0	X	1	0	X	0	X
1	0	1	X	X	X	X	X	X	X	X	X
1	1	0	1	0	0	X	0	X	X	0	X

23

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

Step 4: K-Map simplification

$$J_C = Q_B Q_A$$

$$K_C = 1$$

$$J_B = Q_A$$

$$K_B = Q_A$$

$$J_A = \bar{Q}_C$$

$$K_A = 1$$

Step 5: Draw the logic diagram

Synchronous counter

Count	000	001	010	011	100	000

Timing diagram

Ex:2: Design a counter with the sequence 0,1,3,7,6,4,0   (Dec-10,11)

Solution:

24

Step 1: Determine the number of flip-flops needed. Here, counter should count maximum count = 7 = $(111)_2$ which is 3-bit. Thus, we need 3-flip-flops.

Step 2: Flip-flops to be used : JK.

Step 3: Determine the excitation table for counter. Here, the next state for each present state is written according to given sequence. For example, the next state for the present state 3 (001) is 7 (111). The counts which are not in sequence are treated as don't cares.

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

Excitation table of JK flip-flop

Present state			Next state			Flip-flop inputs					
A	B	C	A^+	B^+	C^+	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	1	0	X	1	X	X	0
0	1	0	X	X	X	X	X	X	X	X	X
0	1	1	1	1	1	1	X	X	0	X	0
1	0	0	0	0	0	X	1	0	X	0	X
1	0	1	X	X	X	X	X	X	X	X	X
1	1	0	1	0	0	X	0	X	X	0	X
1	1	1	1	1	0	X	0	X	X	X	1

Step 4: K-Map simplification

For J_A

A \ BC	$\overline{B}\overline{C}$ 00	$\overline{B}C$ 01	BC 11	$B\overline{C}$ 10
$\overline{A}$ 0	0	0	1	X
A 1	X	X	X	X

$J_A = B$

For K_A

A \ BC	$\overline{B}\overline{C}$ 00	$\overline{B}C$ 01	BC 11	$B\overline{C}$ 10
$\overline{A}$ 0	X	X	X	X
A 1	1	X	0	0

$K_A = \overline{B}$

For J_B

A \ BC	$\overline{B}\overline{C}$ 00	$\overline{B}C$ 01	BC 11	$B\overline{C}$ 10
$\overline{A}$ 0	0	1	X	X
A 1	0	X	X	X

$J_B = C$

For K_B

A \ BC	$\overline{B}\overline{C}$ 00	$\overline{B}C$ 01	BC 11	$B\overline{C}$ 10
$\overline{A}$ 0	X	X	0	X
A 1	X	X	0	1

$K_B = \overline{C}$

For J_C

A \ BC	$\overline{B}\overline{C}$ 00	$\overline{B}C$ 01	BC 11	$B\overline{C}$ 10
$\overline{A}$ 0	1	X	X	X
A 1	0	X	X	0

$J_C = \overline{A}$

For K_C

A \ BC	$\overline{B}\overline{C}$ 00	$\overline{B}C$ 01	BC 11	$B\overline{C}$ 10
$\overline{A}$ 0	X	0	0	X
A 1	X	X	1	X

$K_C = A$
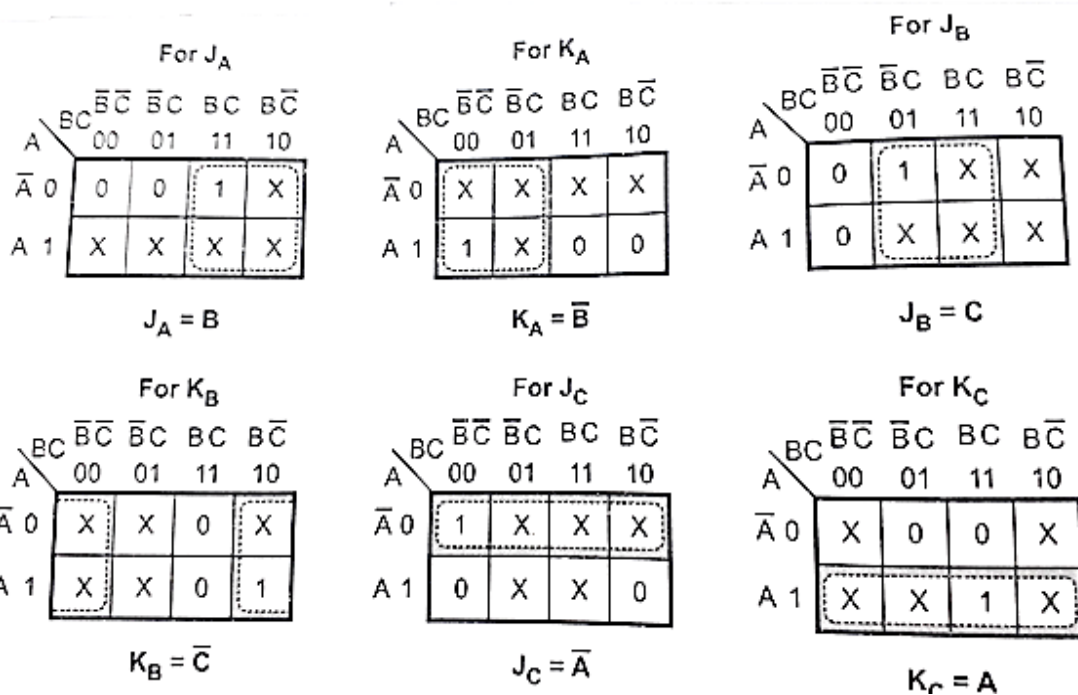
Step 5: Draw logic diagram.
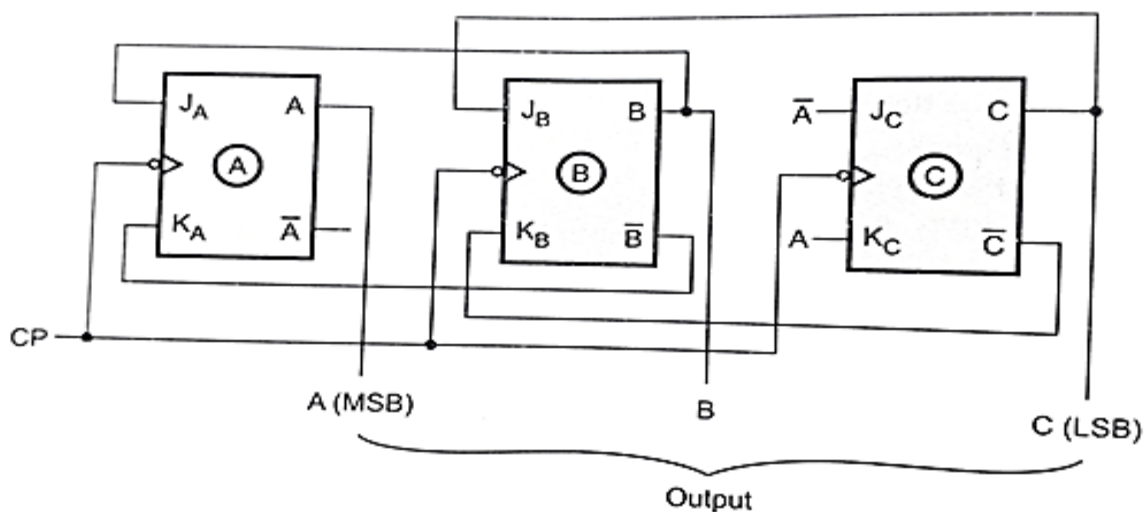
Ex:3: Design and implement a synchronous decade counter using T flip-flop. Draw the timing diagram. (May-08,10,11, Dec-15)

Solution:
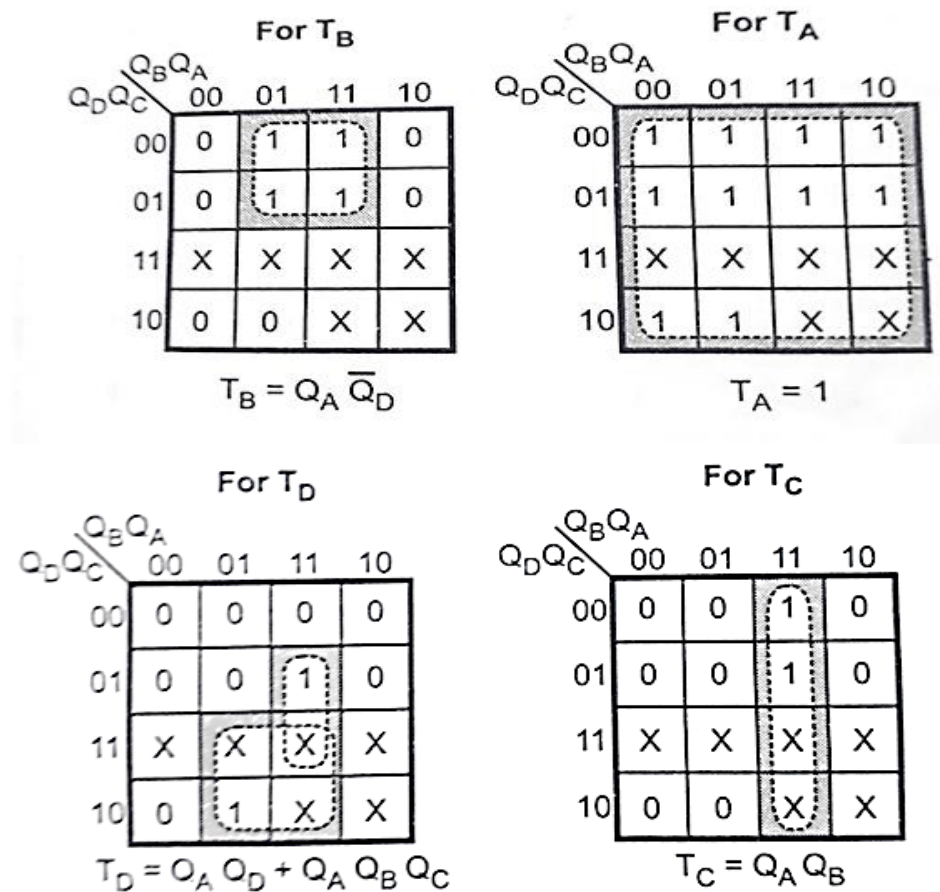**Step 1:** Since N=10, n=4 i.e. flip-flops needed = 4.

Step 2: Flip-flops to be used : T
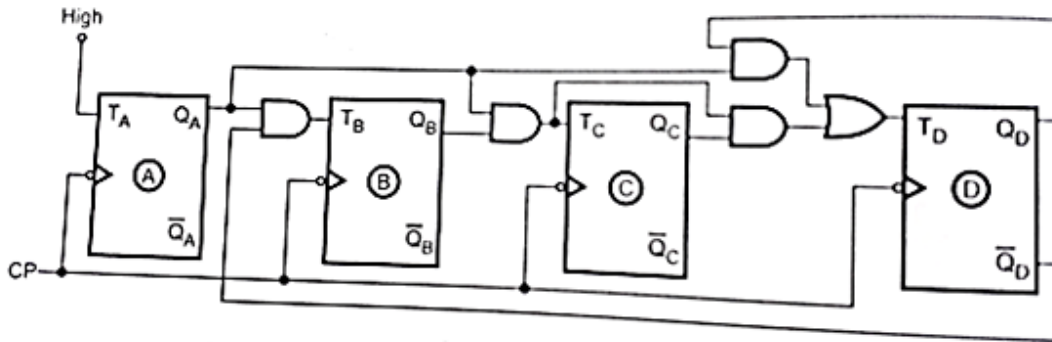
Step 3: determine excitation table for counter

Present state	Next state	Flip-flop inputs

26

Q_D	Q_C	Q_B	Q_A	Q_{D+1}	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_D	T_C	T_B	T_A
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	1	0	0	1	1
0	0	1	1	0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	1	0	1	1	1
0	1	1	1	1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	1	1	0	0	1
1	0	0	1	X	0	0	0	X	0	0	0
1	0	1	0	X	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X

Step 4: K-Map simplification

For T_B

$$T_B = Q_A \overline{Q}_D$$

For T_A

$$T_A = 1$$

For T_D

$$T_D = Q_A Q_D + Q_A Q_B Q_C$$

For T_C

$$T_C = Q_A Q_B$$

Step 5: Logic diagram

27

Step 6: Timing diagram
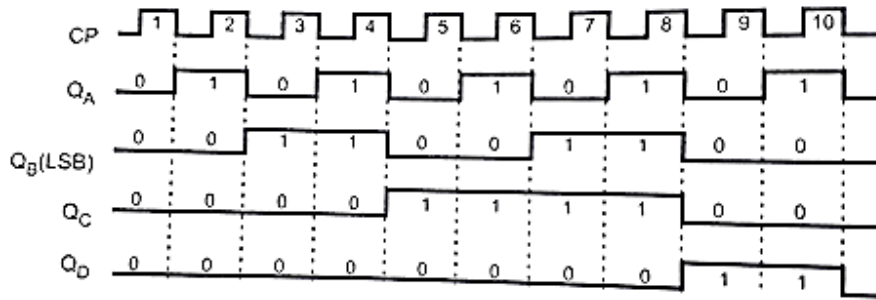
23. What is register?
REGISTER:
- A group of flip-flop can be used to store a word, which is called register.
- A flip-flop can store 1-bit information. So an n-bit register has a group of n flip-flops and is capable of storing any binary information/number containing n-bits.

24. What is buffer register?
Buffer Register:
- Figure 3.23 shows the simplest register constructed with four D flip-flops. This register is also called buffer register.
- Each D flip-flop is triggered with a common negative edge clock pulse.
- The input bits set up the flip-flops for loading.
- When the first negative clock edge arrives, the stored binary information becomes $Q_A Q_B Q_C Q_D$ = ABCD.
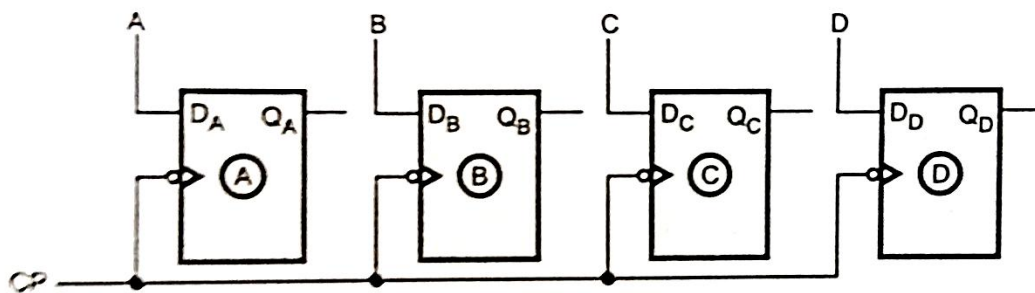

Figure 3.23: Buffer Register

- In this register, four D flip-flops are used. So it can store 4-bit binary information.
- The number of flip-flop stages in a register determines its total storage capacity.

28

25. Controlled Buffer Register:

- We can control input and output of the register by connecting tri-state devices at the input and output sides of register, so this register is called controlled buffer register.
- Hence the tri-state switches are used to control the operation.
- When you want to store data in the register, you have to make $\overline{LOAD}$ or $\overline{WR}$ signal low to activate the tri-state buffers.
- When you want the data at the output, you have to make $\overline{RD}$ signal low to active the buffers.
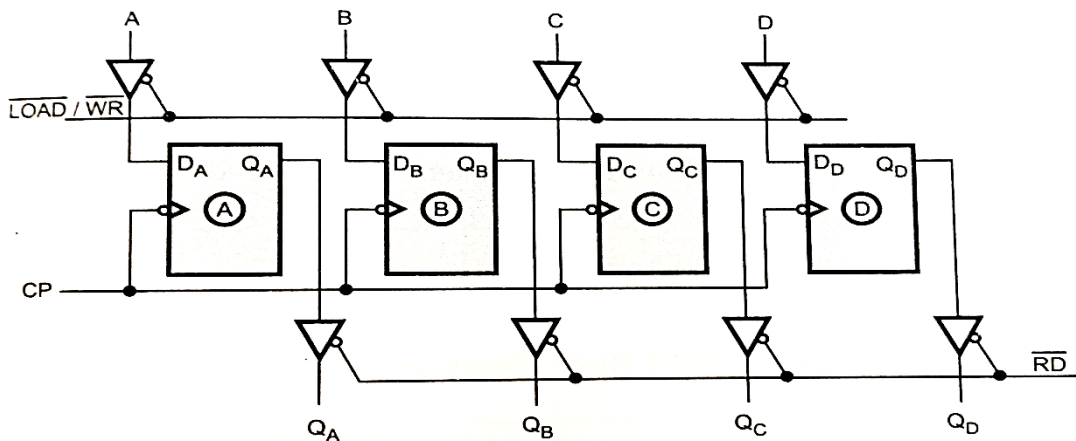- Controlled buffer registers are commonly used for temporary storage of data within a digital system.

Figure 3.24: Controlled buffer register

26. What is shift register? Explain its types.   (May-06, 15, Dec-06, 17)

Shift registers:

- The binary information in the register can be moved from stage to stage within the register or into or out of the register upon application of clock pulses.
- This type of bit movement or shifting is essential for certain arithmetic and logic operations used in microprocessors. This gives rise to a group of registers called shift registers.

(a) Serial shift right, then out

(b) Serial shift left, then out

(c) Parallel shift in

(d) Parallel shift out

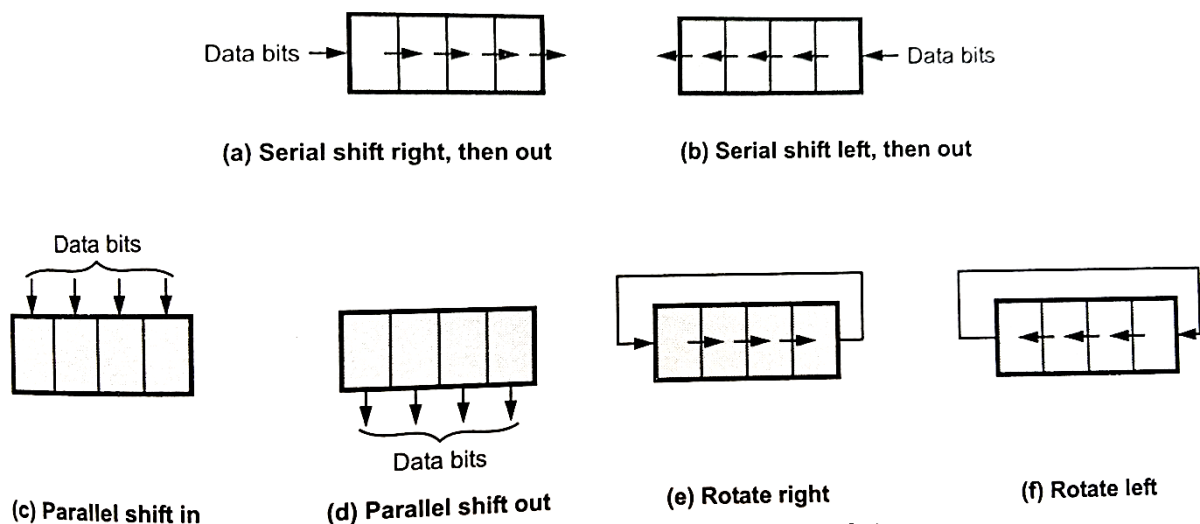(e) Rotate right

(f) Rotate left

29

Figure 3.25: Basic data movement in registers.

- They are very important in applications involving the storage and transfer of data in digital systems.
- The figure 3.25 shows the symbolical representation of the different types of data movement in shift register operation.

Types of shift registers:

- According to data movement there are different types of shift registers, they are
  - ➢ Serial In Serial Out (SISO)
  - ➢ Serial In Parallel Out(SIPO)
  - ➢ Parallel In Parallel Out(PIPO)
  - ➢ Parallel In Serial Out(PISO)

Series In series Out (SISO) shift register:

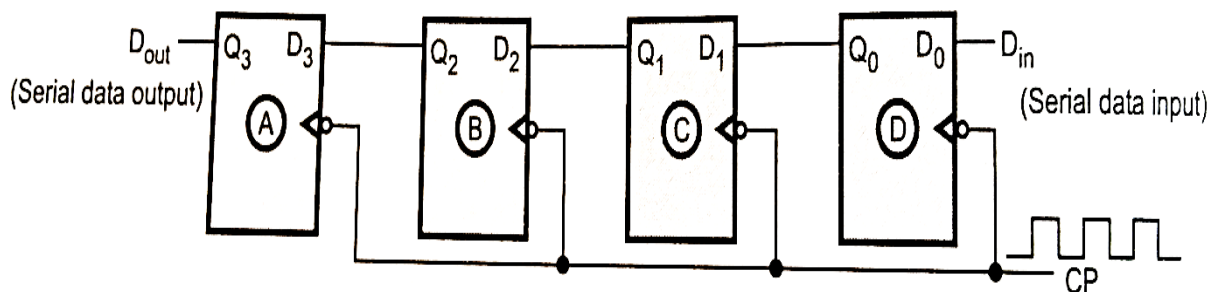- Figure 3.26 shows the SISO shift left registers.

Figure 3.26: shift-left register

- We will illustrate the entry of the four bit binary number 1111 into the register, beginning with the right-most bit. Initially, register is cleared.
- So $Q_3Q_2Q_1Q_0$ = 0000
- The table summarizes the shift left operation.

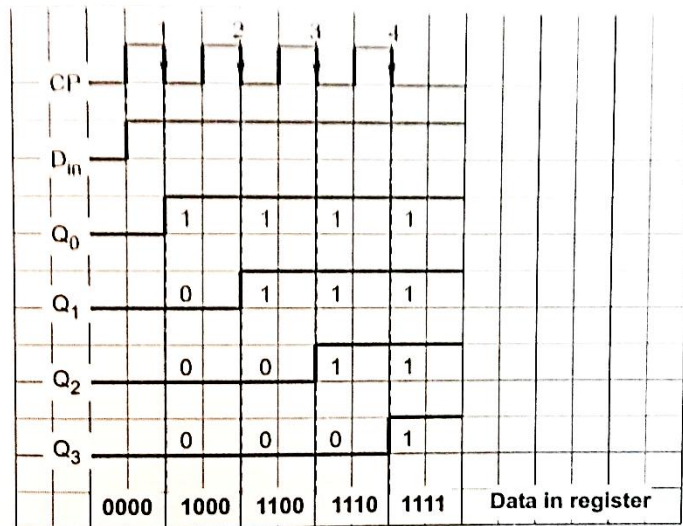| CP	Q_3	Q_2	Q_1	Q_0	D_{in}
Initially	0	0	0	0	1
1st	0	0	0	1	1
2nd	0	0	1	1	1
3rd	0	1	1	1	1
4th	1	1	1	1	1

shift left operation

30

Figure 3.27: waveform for the shift left register

Shift Right Mode:

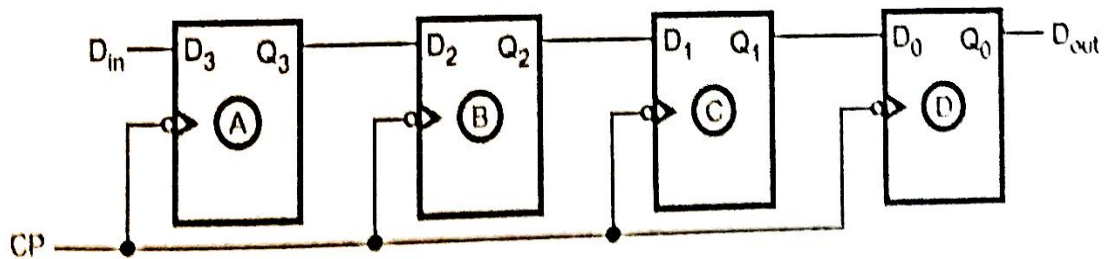- Figure 3.28 shows SISO shift right register.

Figure 3.28: shift right register.

- We will illustrate the entry of the four bit binary number 1111 into the register, beginning with the left-most bit. Initially, register is cleared. So $Q_3Q_2Q_1Q_0$ = 0000.

- Table summarizes the shift right operation

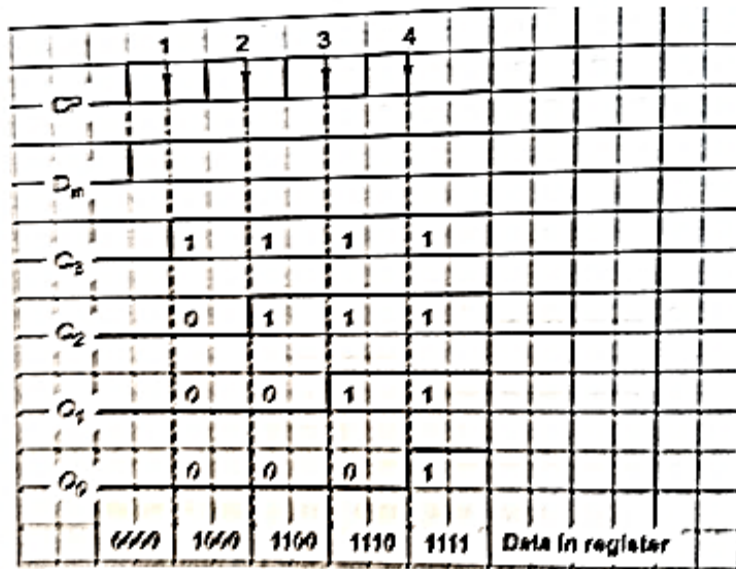| CP	D_{in}	Q_3	Q_2	Q_1	Q_0
Initially	1	0	0	0	0
↓ 1st	1	1	0	0	0
↓ 2nd	1	1	1	0	0
↓ 3rd	1	1	1	1	0
↓ 4th	1	1	1	1	1

Shift right operation

31

Figure 3.29: waveform for shift right register

Series In Parallel Out (SIPO) shift register: (Dec-16)

- The data bit are entered serially into the register but the output is taken in parallel.
- Once the data are stored, each bit appears on its respective output line and all bits are available simultaneously as shown in figure 3.30.
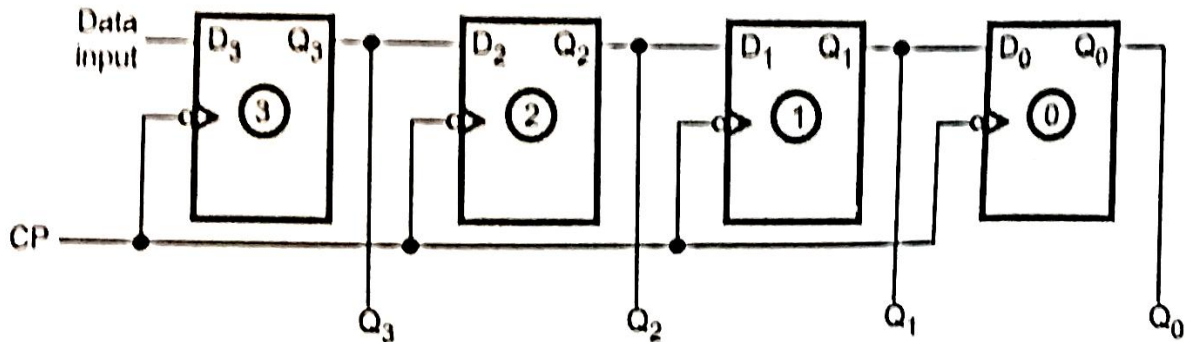

Figure 3.30: A Serial In Parallel Out (SIPO) shift register

CP	Q_3	Q_2	Q_1	Q_0
-	NC	NC	NC	NC
↓	D_3	D_2	D_1	D_0

Truth Table

Parallel In Serial Out (PISO) Shift Register:

- In this type, the bits are entered in parallel i.e simultaneously into their respective stages on parallel lines.
- Figure 3.31 illustrates a four-bit parallel in series out register.
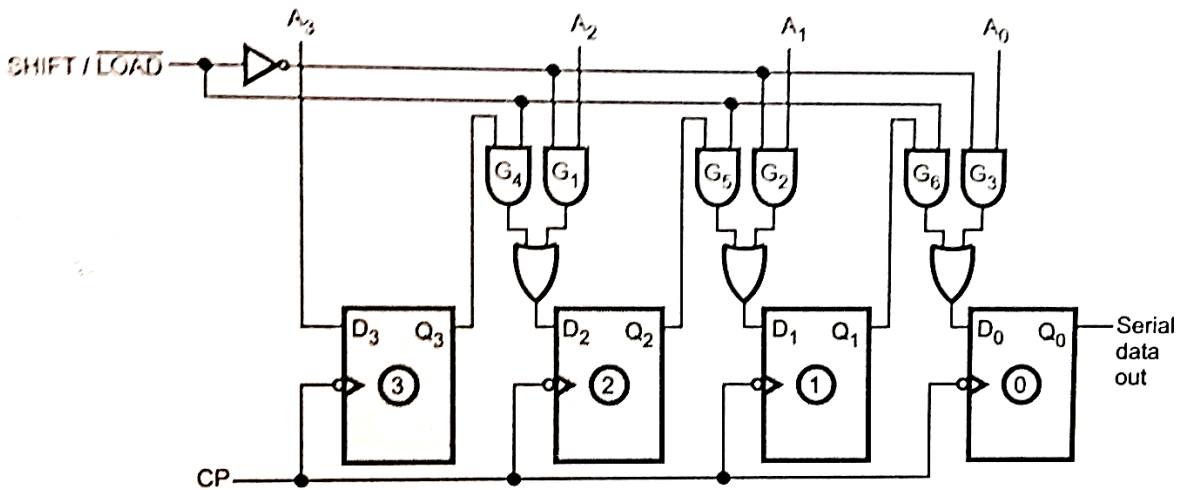
32

Figure 3.31: Parallel in Serial Out (PISO) Shift Register

- There are four input lines A_3, A_2, A_1, A_0 for entering data in parallel into the register.
- SHIFT/$\overline{\text{LOAD}}$ is the control input which allows shift or loading data operation of the register.
- When SHIFT/$\overline{\text{LOAD}}$ is low, gates G_1, G_2, G_3 are enabled, allowing each input data bit to be applied to D input of its respective flip-flop.
- When a clock pulse is applied, the flip-flops with D=1 will SET and those with D=0 will RESET.
- All four bits are stored simultaneously.
- When SHIFT/$\overline{\text{LOAD}}$ is high, gates G_1, G_2, G_3 are disabled and gates G_4, G_5, G_6 are enabled. This allows the data bits to shift right from one stage to the next.
- The OR gates at the D-inputs of the flip-flops allow either the parallel data entry operation or shift operation, depending on which AND gates are enabled by the level on the SHIFT/$\overline{\text{LOAD}}$ input.

Parallel In Parallel Out (PIPO) Shift Register:
- In parallel in parallel out register, there is simultaneous entry of all data bits and the bits appear on parallel outputs simultaneously.
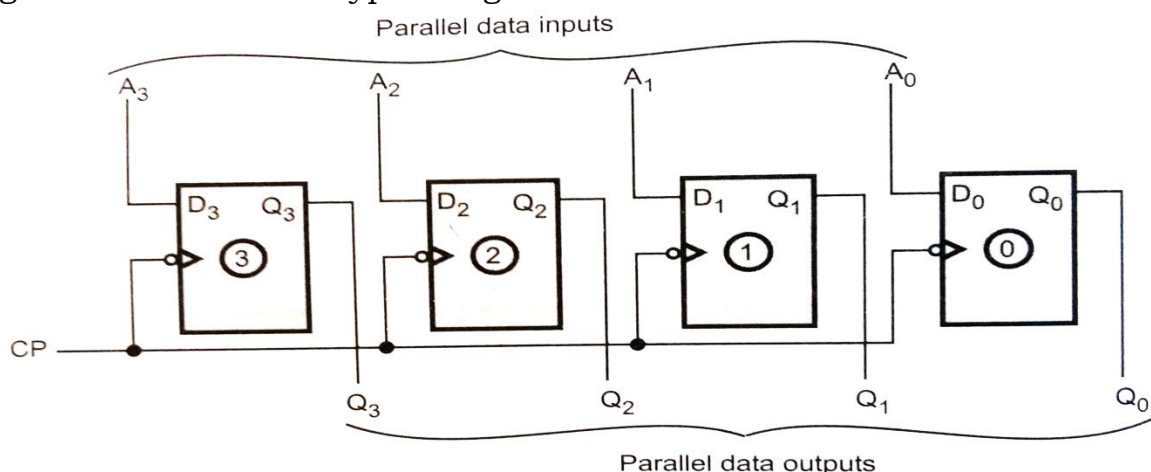- Figure 3.32 shows this type of register.

33

Figure 3.32: Parallel In Parallel Out (PIPO) Shift Register

27. Explain bidirectional shift register.   (May -15)(Dec-2018)

Bidirectional Shift Register:

- This type of register allows shifting of data either to the left or to the right side. It can be implemented by using logic gate circuitry that enables the transfer of data from one stage to the next stage to the right or to the left, depending on the level of a control line.

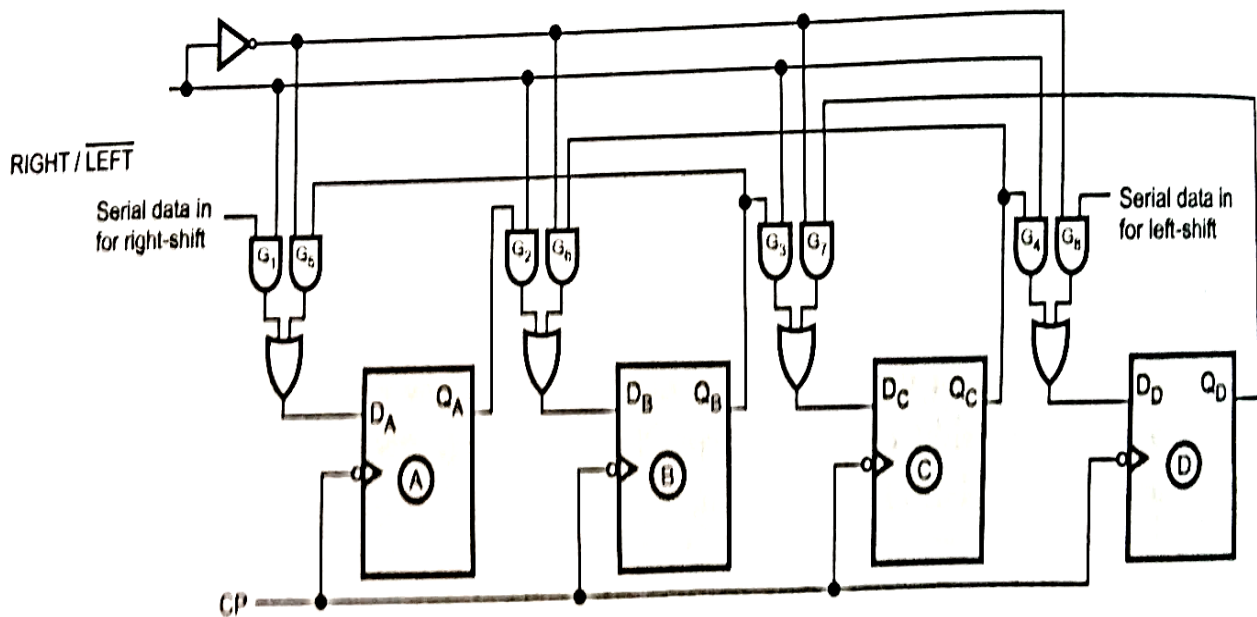- Figure 3.33 illustrate a four-bit bidirectional register.

Figure 3.33: 4-bit bidirectional shift register

- The RIGHT/$\overline{\text{LEFT}}$ is the control input signal which allows data shifting either towards right or towards left.

- A high on this line enables the shifting of data towards right and a low enables it towards left.

- When RIGHT/$\overline{\text{LEFT}}$ signal is high, gates G_1, G_2, G_3, G_4 are enabled.

- The state of the Q output of each flip-flop is passed through the D input of the following flip-flop.

- When a clock pulse arrives, the data are shifted one place to the right.

- When the RIGHT/$\overline{\text{LEFT}}$ signal is low, gates G5, G6, G7, G8 are enabled.

- The Q output of each flip-flop is passed through the D input of the preceding flip-flop.

- When clock pulse arrives, the data are shifted one place to the left.

Bidirectional Shift Register with Parallel Load:

- When parallel load capability is added to the shift register, the data entered in parallel can be taken out in serial fashion by shifting the data stored in the register.

- Such a register is called bidirectional shift register with parallel load.

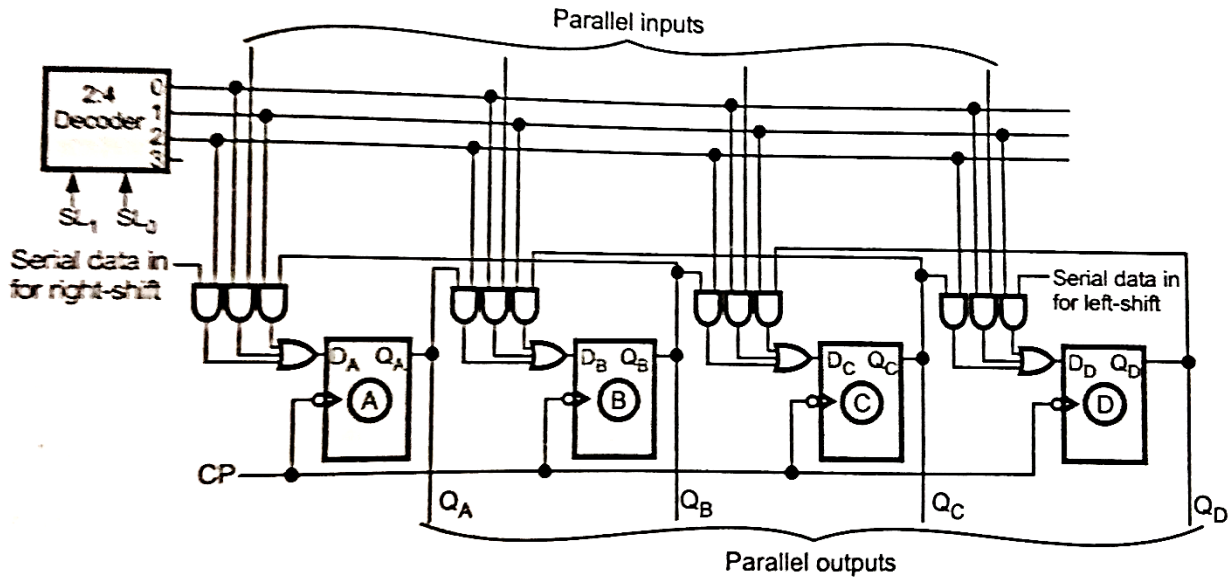- Figure 3.34 showsbidirectional shift register with parallel load.

Figure 3.34: 4-bit bidirectional shift register with parallel load.

- The decoder select lines select the one source out of three as shown in the table.

SL_1	SL_0	Selected source
0	0	Parallel input
0	1	Output of right adjacent FF
1	0	Output of left adjacent FF

Shift Register using JK Flip-flops:
- By applying complement inputs to J and K we can construct serial-in serial-out shift right register using JK flip-flops.
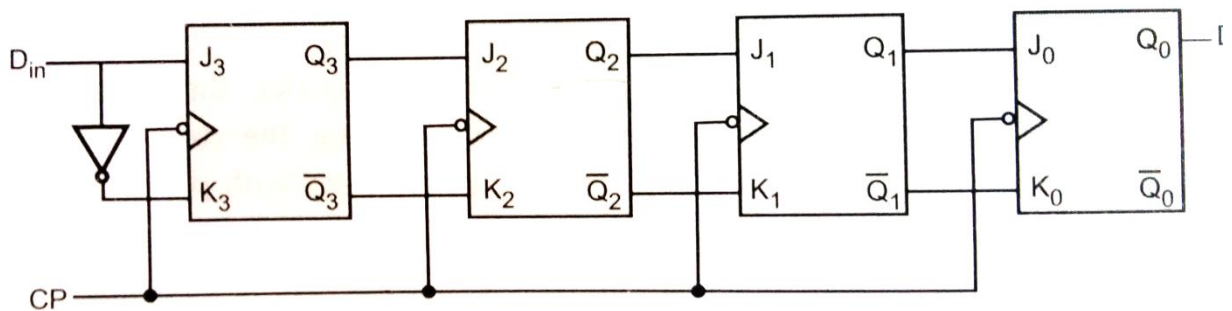

Figure 3.35: Shift right register using JK flip-flops

28. Explain the operation of universal shift register.   (Dec-08, 10, May-09, 10)
Universal Shift Register:
- A register capable of shifting in one direction only is a unidirectional shift register.
- A register capable of shifting in both directions is a bidirectional shift register.
- If the register has both shifts (right shift and left shift) and parallel load capabilities, it is referred to as **universal shift register.**
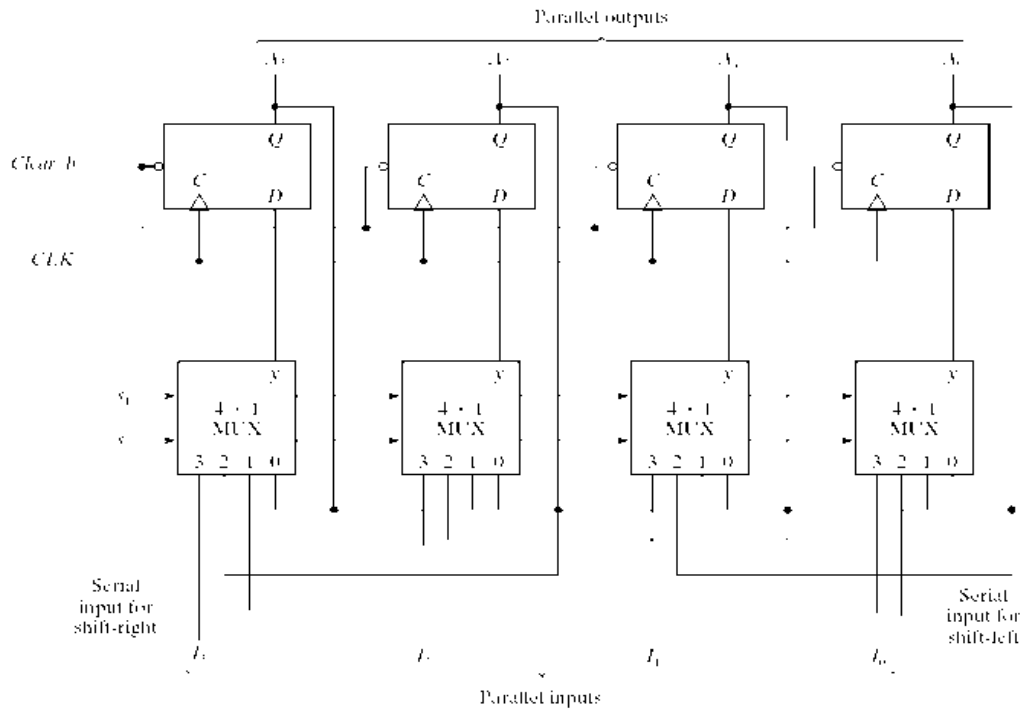- Figure 3.36 shows the 4-bit universal shift register.

35

Figure 3.36:4-bit universal shift register

- It consists of four flip-flops and four multiplexers.
- The four multiplexers have two common selection inputs S1 and S0, and they select appropriate input of D flip-flop.
- The table shows the register operation depending on the selection inputs of multiplexers.

Mode control		Register operation
S_1	S_2	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

- When $S_1 S_0$ = 00, input 0 is selected and the present value of the register is applied to the D inputs of the flip-flops. This results no change in the register value.
- When $S_1 S_0$ = 01, input 1 is selected and the circuit connections are such that it operates as a right shift register.
- When $S_1 S_0$ = 10, input 2 is selected and the circuit connections are such that it operates as a left shift register.
- Finally, when $S_1 S_0$ = 11, the binary information on the parallel input lines is transferred into the register simultaneously and it is a parallel load operation.

29. What are the applications of shift register?

Applications of Shift Registers:
- Primary use of shift register is temporary data storage and bit manipulations. Some of the common applications of shift registers are

36

Delay line:
- A Serial-In-Serial-Out (SISO) shift register can be used to introduce time delay Δt in digital signals. The time delay can be given as

$$\Delta t = N \times \frac{1}{f_c}$$

- Where N is the number of stages (i.e. flip-flops) and fc is the clock frequency.
- An input pulse train appears at the output delayed by Δt.
- The amount of delay can be controlled by the clock frequency or by the number of flip-flop in the shift register.

Serial-to-parallel converter:
- A Serial-In-Parallel-Out (SIPO) shift register can be used to convert data in the serial form to the parallel form.

Parallel-to-serial converter:
- A Parallel-In-Serial-Out (PISO) shift register can be used to convert data in the parallel form to the serial form.

Shift register counters:
- A shift register with the serial output connected back to the serial input is called shift register counter.
- The most common shift register counters are the ring counter and the Johnson counter.

Pseudo-Random Binary sequence (PRBS) generator:
- A shift register can be used as a pseudo-random binary sequence generator.
- A suitable feedback is used to generate pseudo-random sequence.
- The term random here means that the outputs do not cycle through a normal binary count sequence.
- The term pseudo here refers to the fact that the sequence is not truly random because it does cycle through all possible combinations once every 2n – 1 clock cycles, where n represents the number of shift register stages (number of flip-flops).

Sequence Generator:
- The shift register can be used to generate a particular bit pattern respectively.
- Left most flip-flop input accepts the serial input and the right most flip-flop gives serial data output.
- The serial data output signal is connected as a serial data in.
- On every clock pulse the data shift operation takes place.
- The loaded bit pattern at the serial output is in a sequence.

- Same bit pattern is again loaded in the register since serial output is connected serial in of the register. Thus, the circuit generates a particular bit pattern respectively.

Sequence Detector:
- The shift register can be used to detect the desired sequence.
- The detection process requires two registers: one register stores the bit pattern to be detected i.e. R_1 and other register accepts the input data stream i.e. R_2.
- Input data stream enters a shift register as serial data in and leaves as serial out.
- In every clock cycle, bit-wise comparisons of these two registers are done using EX-NOR gates as shown in figure 3.37.
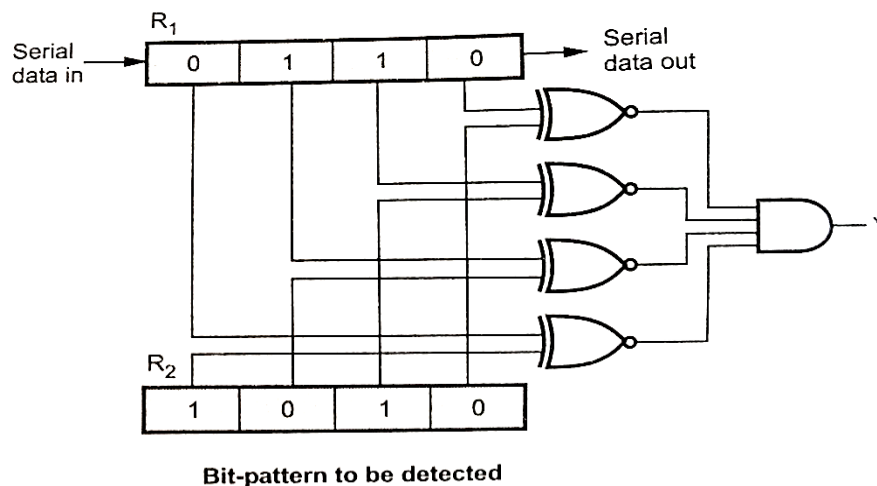

Figure 3.37: 4-bit sequence detector

- The two-input EX-NOR gate gives logic high output when both inputs are either low or high, i.e. when both the inputs are equal.
- When outputs of all the EX-NORs gates are logic high we can say that all bits are matched and hence the desired bit pattern is detected.
- The final output which indicates that the pattern is detected is taken from four-input AND gate.
- The 4-bit sequence detector can be made programmable by loading the desired 4-bit data in the register R_2.

30. Explain the operation of 4-bit ring counter.   **(Dec-04, 05)**

Ring counters:
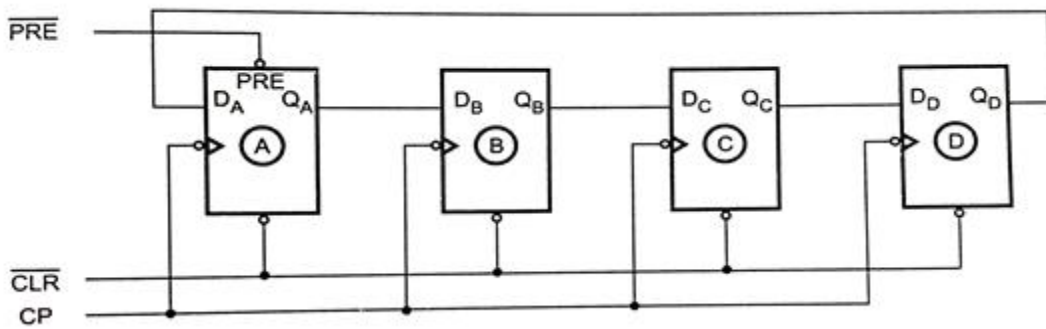- The figure 3.38 shows the logic diagram for four-bit ring counter.

Figure 3.38: Four-bit ring counter

- The Q output of each stage is connected to the D input of the next stage and the output of the last stage is fed back to the input of first stage.
- The $\overline{CLR}$ followed by $\overline{PRE}$ makes the output of first stage to '1' and remaining outputs are zero, i.e. Q_A is one and Q_B, Q_C, Q_D are zero.
- The first clock pulse produces $Q_B = 1$ and remaining outputs are zero.
- According to the clock pulses applied at the clock input CP, a sequence of four states is produced. These states are listed in table.

Clock pulse	Q_A	Q_B	Q_C	Q_D
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0

Ring counter sequence 4-bits

- 1 is always retained in the counter and simply shifted around the ring advancing one stage for each clock pulse. In this case four stages of flip-flops are used. So a sequence of four states is produced and repeated.
- Figure 3.39 gives the timing sequence for a four-bit ring counter.
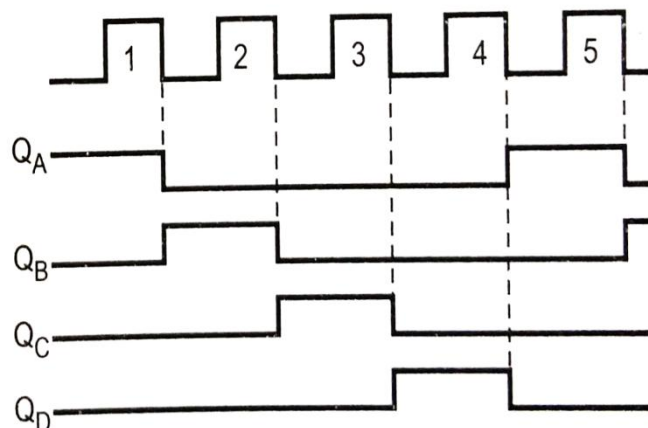

Figure 3.39: Timing sequence for a four-bit ring counter.

39

- The ring counter can be used for counting the number of pulses.
- The number of pulses counted is read by noting which flip-flop is in state 1.
- No decoding circuitry is required.
- Since there is one pulse at the output for each of the N clock pulses, this circuit is also referred to as a divide-by-N-counter or an N:1 scalar.
- Ring counters can be instructed for any desired MOD number, that is MOD-N ring counter requires N flip-flops.

31. Explain the operation of Johnson counter. (Dec-05, 06, 11)

Johnson or Twisting Ring or Switch Tail Counter:
- In a Johnson counter, the Q output of each stage of flip-flop is connected to the D input of the next stage.
- The single exception is that the complement output of the last flip-flop is connected back to the D-input of the first flip-flop .
- Johnson counter can be implemented with SR or JK flip-flops as well.
- As shown in figure 3.40 there is a feedback from the rightmost flip-flop complement output to the leftmost flip-flop input. This arrangement produces a unique sequence of states.
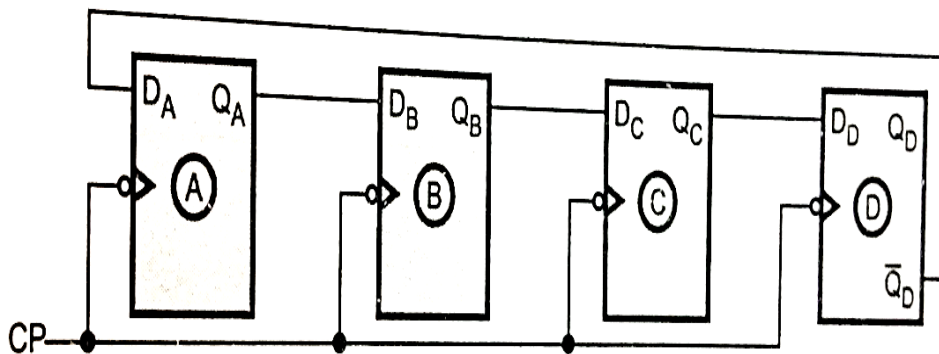
Figure 3.40: Four-bit Johnson counter.

- Initially, the register (all flip-flops) is cleared. So all the outputs, Q_A, Q_B, Q_C, Q_D are zero.
- The output of last stage, Q_D is zero. Therefore complement output of last stage, $\overline{Q_D}$ is one. This is connected back to the D input of first stage. So D_A is one.
- The first falling clock edge produces $Q_A = 1$ and $Q_B = 0$, $Q_C = 0$, $Q_D = 0$ since D_B, D_C, D_D are zero.
- The next clock pulse produces $Q_A = 1$, $Q_B = 1$, $Q_C = 0$, $Q_D = 0$.
- The sequence of states is summarized in table.

Clock pulse	Q_A	Q_B	Q_C	Q_D
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

Four-bit Johnson sequence

- After 8 states the same sequence is repeated.
- In this case, four-bit register is used. So the four-bit sequence has a total of eight states.
- Figure 3.41 gives the timing sequence for a four-bit Johnson counter.
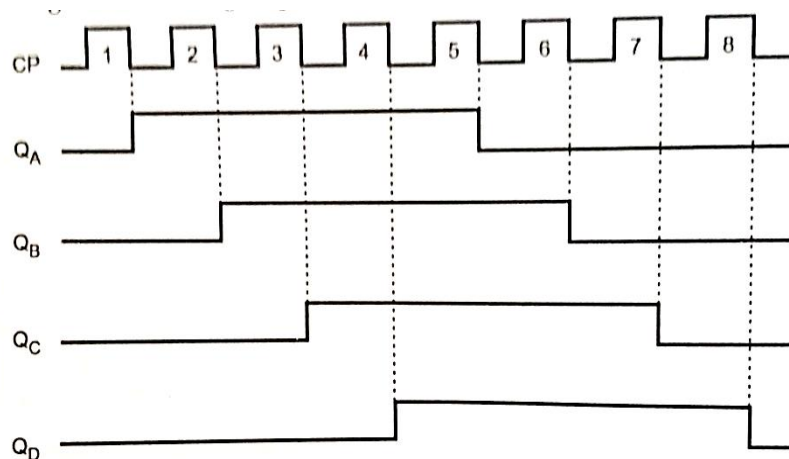

Figure 3.41: timing sequence for a four-bit Johnson counter

- If we design a counter of five-bit sequence, it has a total of ten states.
- An n-stage Johnson counter will produce a modulus of 2Xn, where n is the number of stages (i.e. flip-flops) in the counter.
- Johnson counter requires only half the number of flip-flops compared to the standard ring counter. However, it requires more flip-flop than binary counter.

Ex.1:Draw a 5 flip-flop shift (Johnson) counter, its truth table and waveforms. Explain its operation as a decade counter. **(Dec-05)**

Solution: The figure shows the 5-bit shift (Johnson) counter. Since this counter goes through 10 states, the frequency at the output of last flip-flop is $1/10^{th}$ of the clock frequency and hence it is a decade counter.
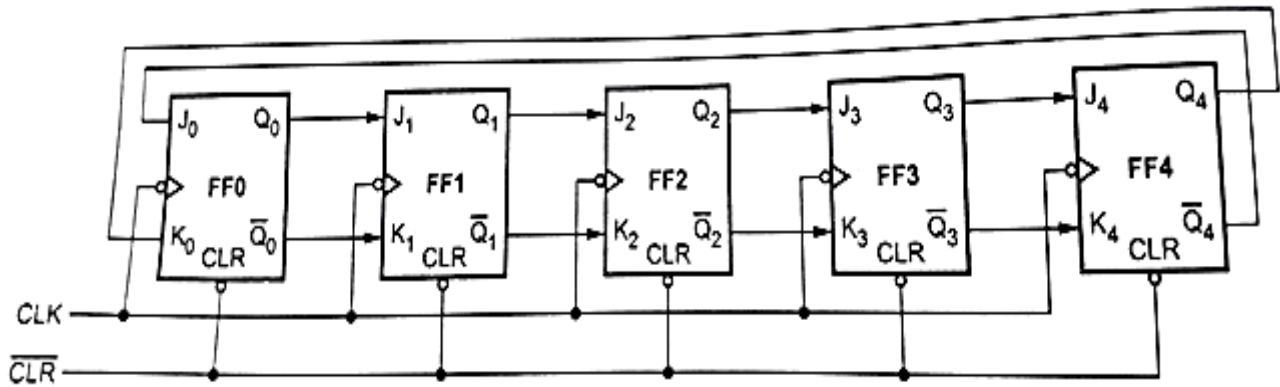
Figure 3.42: Flip-flop shift (Johnson counter)

The table shows the truth table for the 5 flip-flop shift counter and illustrate its operation.

$\overline{CLR}$	CLK	Q_4	Q_4	Q_4	Q_4	Q_4	state	Decimal equivalent
	-	0	0	0	0	0	1	0
1	↓	0	0	0	0	1	2	1
1	↓	0	0	0	1	1	3	3
1	↓	0	0	1	1	1	4	7
1	↓	0	1	1	1	1	5	15
1	↓	1	1	1	1	1	6	31
1	↓	1	1	1	1	0	7	30
1	↓	1	1	1	0	0	8	28
1	↓	1	1	0	0	0	9	24
1	↓	1	0	0	0	0	10	16
1	↓	0	0	0	0	0	1	0

Truth table
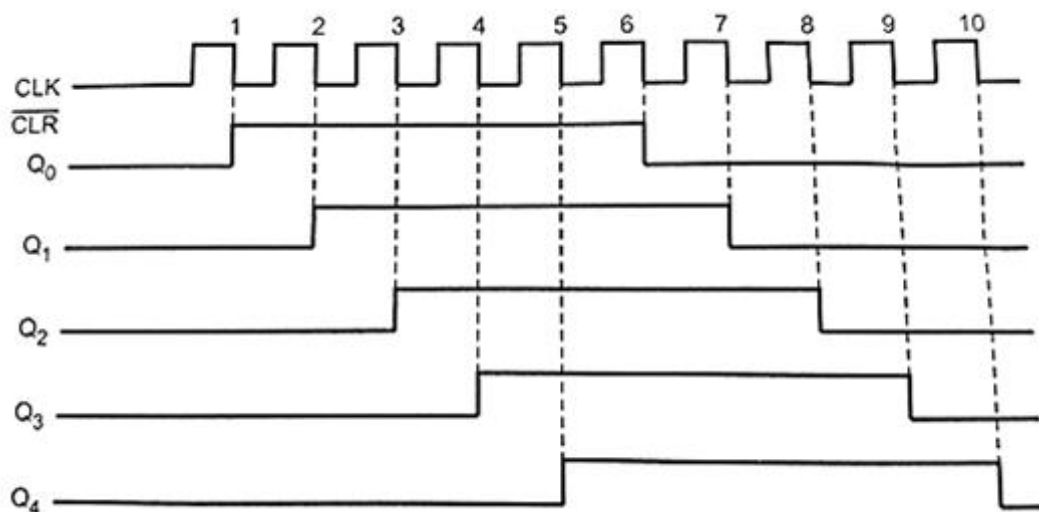
Figure 3.43: Waveform for 5 flip-flop shift counter

ANALYSIS OF CLOCKED SEQUENTIAL CIRCUIT

Design and analyze of clocked sequential circuit with an example.

The analysis of a sequential circuit consists of obtaining a table or a diagram for the time sequence of inputs, outputs and internal states.

Fig: Example of sequential circuit

Consider the sequential circuit is shown in figure. It consists of two D flip-flops A and B, an input x and an output y.

A state equation specifies the next state as function of the present state and inputs.

$$A(n+1) = A(n)x(n) + B(n)x(n)$$
$$B(n+1) = A\overline{(n)}x(n)$$

They can be written in simplified form as,

$$A(n+1) = Ax + Bx$$
$$B(n+1) = Ax$$

The present state value of the output can be expressed algebraically as,

$$y(n) = (A+B)\ \overline{x}$$

DESIGN OF SYNCHRONOUS COUNTERS

Design and analyze of clocked sequential circuit with an example.

43

The procedure for designing synchronous sequential circuit is given below,

1. *From the given specification, Draw the state diagram.*
2. *Plot the state table.*
3. *Reduce the number of states if possible.*
4. *Assign binary values to the states and plot the transition table by choosing the type of Flip-Flop.*
5. *Derive the Flip flop input equations and output equations by using K-map.*
6. *Draw the logic diagram.*

State Diagram:

➢ State diagram is the *graphical representation of the information available in a state table.*

➢ In state diagram, a state is represented by a circle and the transitions between states are indicated by directed lines connecting the circles.

State Table:

➢ A state table gives the time sequence of inputs, outputs ad flip flops states. The table consists of four sections labeled present state, next state, input and output.

➢ The present state section shows the states of flip flops A and B at any given time 'n'. The input section gives a value of x for each possible present state.

➢ The next state section shows the states of flip flops one clock cycle later, at time n+1.

The state table for the circuit is shown. This is derived using state equations.

Present State		Input	Next State		Output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

The above state table can also be expressed in different forms as follows.

44

Present State		Next State				Output	
		x = 0		x = 1		x = 0	x = 1
A	B	A	B	A	B	y	y
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

The state diagram for the logic circuit in below figure.
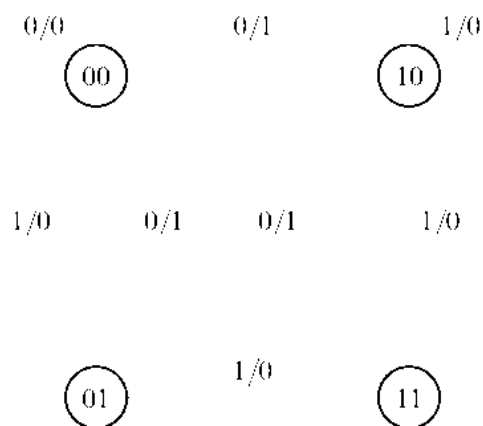
Flip-Flop Input Equations:

The part of the circuit that generates the inputs to flip flops is described algebraically by a set of Boolean functions called flip flop input equations.

The flip flop input equations for the circuit is given by,
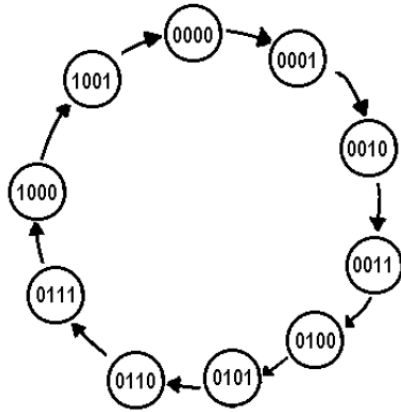
$$D_A = Ax + Bx$$
$$D_B = A x$$

****

****

Design of a Synchronous Decade Counter Using JK Flip- Flop (Apr 2018, Nov 2018)

A synchronous decade counter will count from zero to nine and repeat the sequence.

State diagram:

The state diagram of this counter is shown in Fig.

Excitation table:

| Present State				Next State				Output							
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	J_3	K_3	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	0	0	1	0	X	0	X	0	X	1	X
0	0	0	1	0	0	1	0	0	X	0	X	1	X	X	1
0	0	1	0	0	0	1	1	0	X	0	X	X	0	1	X
0	0	1	1	0	1	0	0	0	X	1	X	X	1	X	1
0	1	0	0	0	1	0	1	0	X	X	0	0	X	1	X
0	1	0	1	0	1	1	0	0	X	X	0	1	X	X	1
0	1	1	0	0	1	1	1	0	X	X	0	X	0	1	X
0	1	1	1	1	0	0	0	1	X	X	1	X	1	X	1
1	0	0	0	1	0	0	1	X	0	0	X	0	X	1	X
1	0	0	1	0	0	0	0	X	1	0	X	0	X	X	1

K-Map:

$J_0 = 1$

$K_0 = 1$

46

Karnaugh map ($Q_1 Q_0$ columns: 00, 01, 11, 10 / $Q_3 Q_2$ rows: 00, 01, 11, 10)

$Q_3 Q_2$ \ $Q_1 Q_0$	00	01	11	10
00		1	X	X
01		1	X	X
11	X	X	X	X
10			X	X

$$J_1 = \overline{Q_3}\, Q_0$$

$Q_3 Q_2$ \ $Q_1 Q_0$	00	01	11	10
00	X	X	1	
01	X	X	1	
11	X	X	X	X
10	X	X	X	X

$$K_1 = \overline{Q_3}\, Q_0$$

$Q_3 Q_2$ \ $Q_1 Q_0$	00	01	11	10
00			1	
01	X	X	X	X
11	X	X	X	X
10			X	X

$$J_2 = Q_1 Q_0$$

$Q_3 Q_2$ \ $Q_1 Q_0$	00	01	11	10
00	X	X	X	X
01			1	
11	X	X	X	X
10			X	X

$$K_2 = Q_1 Q_0$$

$Q_3 Q_2$ \ $Q_1 Q_0$	00	01	11	10
00				
01			1	
11	X	X	X	X
10	X	X	X	X

$$J_3 = \overline{Q_3}\, Q_0 + Q_2 Q_1 Q_0$$

$Q_3 Q_2$ \ $Q_1 Q_0$	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	X	X	X	X
10		1	X	X

$$K_3 = \overline{Q_3}\, Q_0 + Q_2 Q_1 Q_0$$

47

Logic Diagram:

****
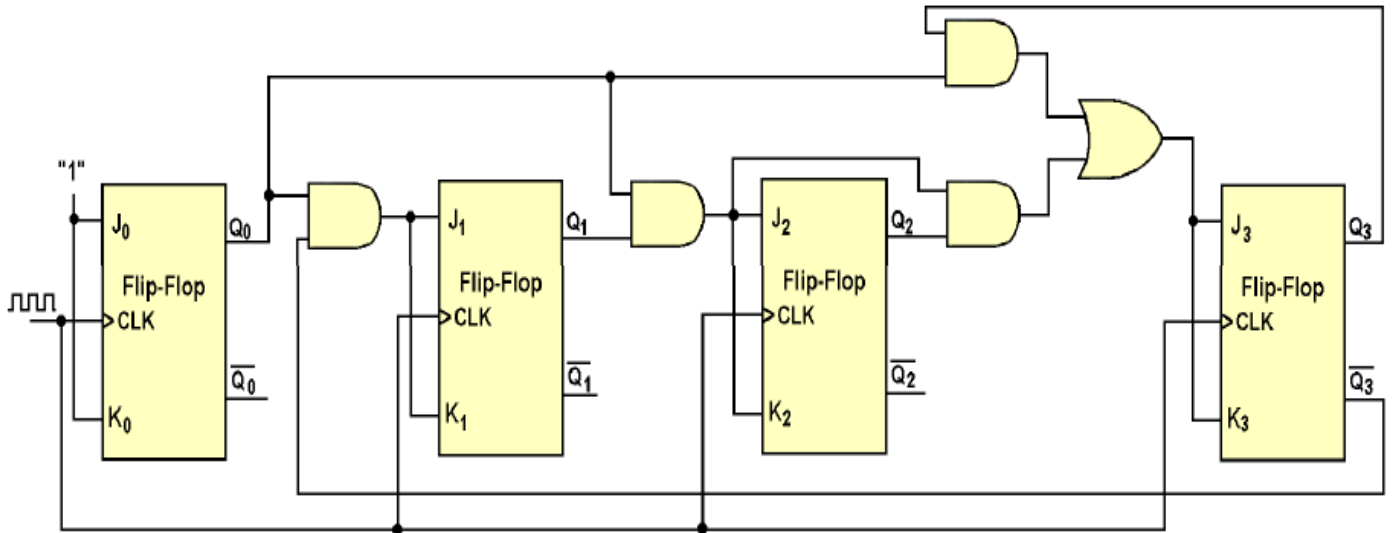
Design of an Asynchronous Decade Counter Using JK Flip- Flop.

An asynchronous decade counter will count from zero to nine and repeat the sequence. Since the JK inputs are fed from the output of previous flip-flop, therefore, the design will not be as complicated as the synchronous version.

At the ninth count, the counter is reset to begin counting at zero. The NAND gate is used to reset the counter at the ninth count. At the ninth count the outputs of flip-flop Q3 and Q1 will be high simultaneously. This will cause the output of NAND to go to logic "0" that would reset the flip-flip. The logic design of the counter is shown in Fig.

****

48
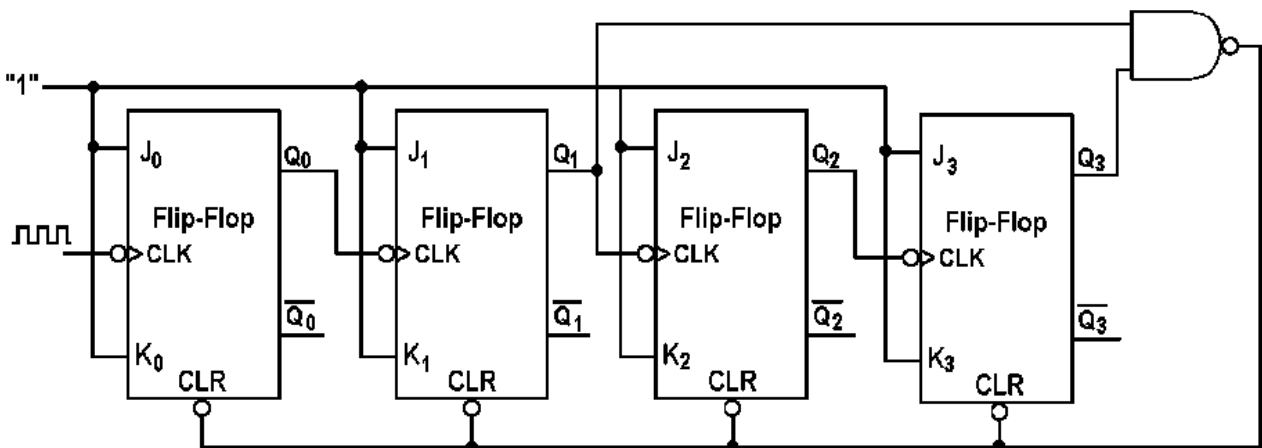
Design of a Synchronous Modulus-Six Counter Using SR Flip-Flop(Nov 2017)

The modulus six counters will count 0, 2, 3, 6, 5, and 1 and repeat the sequence. This modulus six counter requires three SR flip-flops for the design.

State diagram:
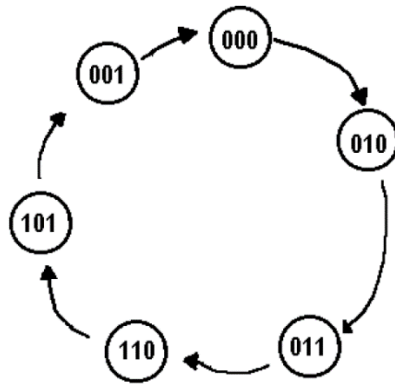
Truth table:

[Present State			Next State			Output					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	R_2	S_2	R_1	S_1	R_0	S_0
0	0	0	0	1	0	0	X	1	0	0	X
0	1	0	0	1	1	0	X	X	0	1	0
0	1	1	1	1	0	1	0	X	0	0	1
1	1	0	1	0	1	X	0	0	1	1	0
1	0	1	0	0	1	0	1	0	X	X	0
0	0	1	0	0	0	0	X	0	X	0	1

K-Map:

$$R_0 = Q_1 \cdot \overline{Q_0}$$

$$S_0 = \overline{Q_2} \cdot Q_0$$

$$R_1 = \overline{Q_1} \cdot \overline{Q_0}$$

$$S_1 = Q_2$$

49

$$R_2 = Q_1 \cdot Q_0$$

$$S_2 = \overline{Q_1}$$

Logic Diagram:

MEALY AND MOORE MODELS

Write short notes on Mealy and Moore models in sequential circuits.

> ➢ In synchronous sequential circuit the outputs depend upon the order in which its input variables change and can be affected at discrete instances of time.

General Models:

> ➢ There are two models in sequential circuits. They are:
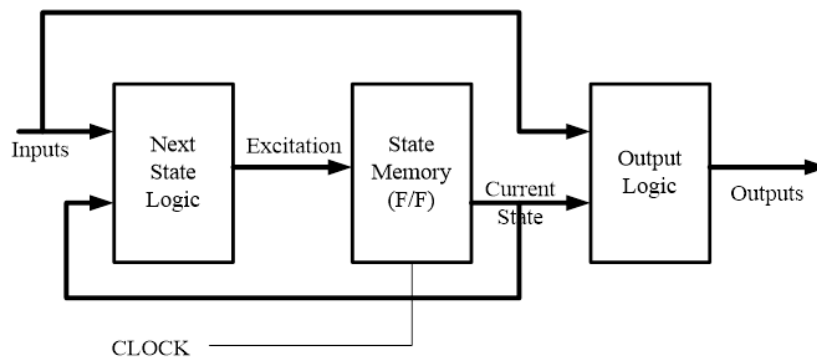>> 1. Mealy model
>> 2. Moore model

Moore machine:

> ➢ In the Moore model, the outputs are a function of present state only.
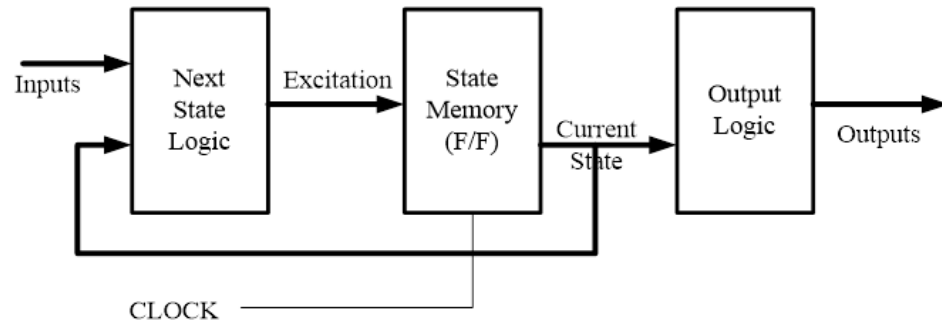
Mealy machine:

➢ In the Mealy model, the outputs are a function of present state and external inputs.

Difference between Moore model and Mealy model.

Sl.No	Moore model	Mealy model
1	Its output is a function of present state only.	Its output is a function of present state as well as present input.
2	Input changes does not affect the output.	Input changes may affect the output of the circuit.
3	It requires more number of states for implementing same function.	It requires less number of states for implementing same function.

Example:

A sequential circuit with two 'D' Flip-Flops A and B, one input (x) and one output (y).

The Flip-Flop input functions are:

$D_A = Ax + Bx$

$D_B = A'x$ and

the circuit output function is, $Y = (A + B) x'$.

(a) Draw the logic diagram of the circuit, (b) Tabulate the state table, (c) Draw the state diagram.
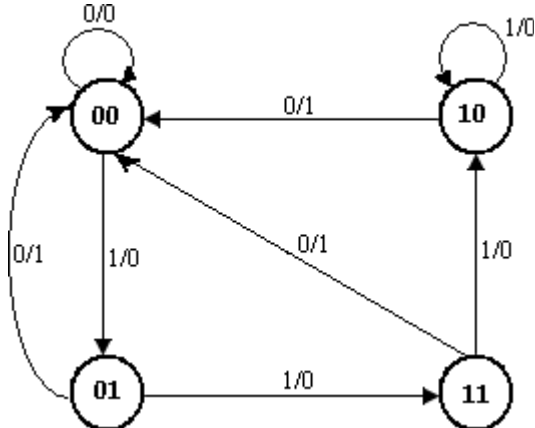
Solution:
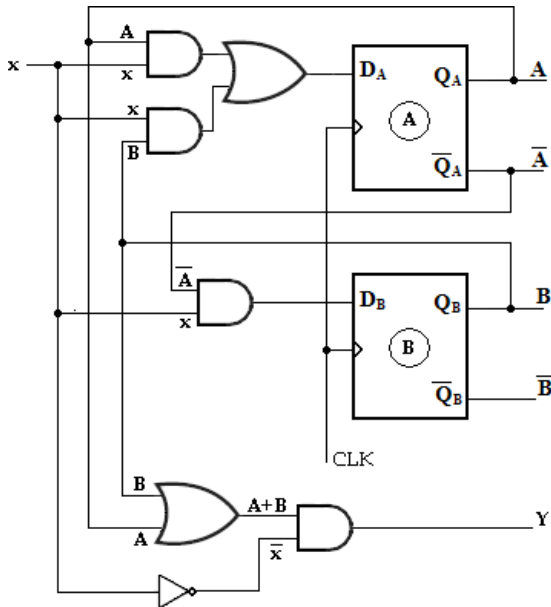
State table:

Present state		Input	Flip-Flop Inputs		Next state		Output
A	B	x	$D_A = Ax+Bx$	$D_B = A'x$	A(t+1)	B(t+1)	$Y = (A+B)x'$
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	0	0	0	1
0	1	1	1	1	1	1	0
1	0	0	0	0	0	0	1
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	1
1	1	1	1	0	1	0	0

Present state		Next state				Output	
		x= 0		x= 1		x= 0	x= 1
A	B	A	B	A	B	Y	Y
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

Second form of state table

State diagram:

52

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

1. Write short notes on types of Asynchronous sequential circuits.

(Dec-12, 15)

ASYNCHRONOUS SEQUENTIAL CIRCUITS

- Sequential circuits without clock pulses are called Asynchronous Sequential Circuits. They are classified into 2 types:
    1. Fundamental mode circuits
    2. Pulse mode circuits

Fundamental Mode Circuits:

It assumes that:
  - ➢ The input variables should change only when the circuit is stable.
  - ➢ Only one input variable can change at a given time.
  - ➢ Inputs and outputs are represented by levels.

Pulse Mode Circuits:

It assumes that:
  - ➢ Inputs and outputs are represented by pulses.
  - ➢ The width of the pulse is long enough for the circuit to respond to the input.
  - ➢ The pulse width must not be so long that it is still present after the new state is reached.

Block diagram of Asynchronous Sequential circuits
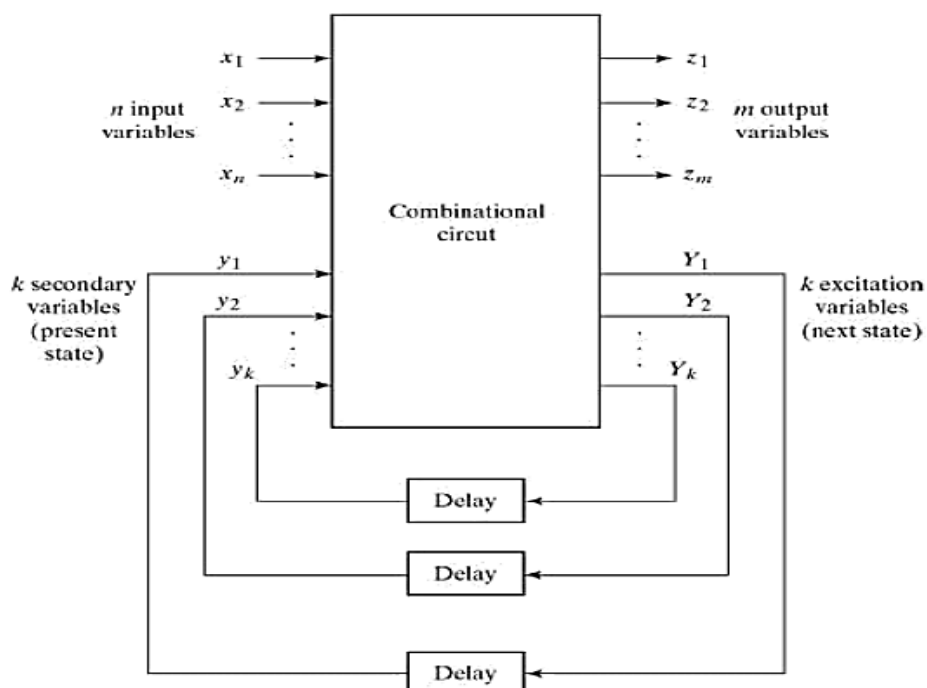
The communication oftwo units, with each unithavingits own independent

1

clock, must be donewith asynchronous circuits.

Stable state:
- If the circuit reaches a steady state condition with **present state y_i = next state** Y_i for i=1,2,3...K then the circuit is said to be stable state.
- A transition from one stable to another occurs only in response to a change in an input variable.

Unstable state:
- In a circuit, if **present state $y_i \neq$ next state** Y_i for i=1,2,3...K then the circuit is said to be unstable state.
- The circuit will be in continuous transition till it reached a stable state.

2. **Explain in detail about analysis procedure of fundamental mode sequential circuits.**

ANALYSIS PROCEDURE OF FUNDAMENTAL MODE SEQUENTIAL CIRCUITS:
- The analysis of asynchronous sequential circuits consists of obtaining a table or a diagram that described the sequence of internal states and outputs as a function of changes in the input variables.
- Let us consider the synchronous sequential circuit is shown in figure.
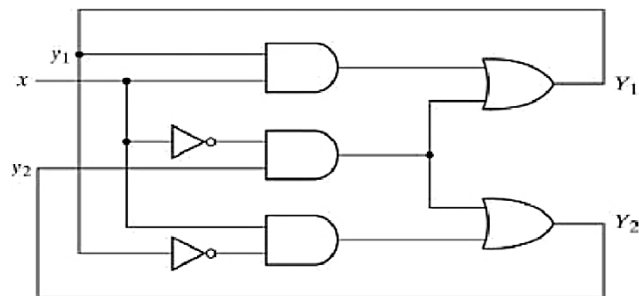
- The analysis of the circuit starts by considering the excitation variables (Y_1 and Y_2) as outputs and the secondary variables (y_1 and y_2) as inputs.

Step1:
- The Boolean expressions are,

$$Y_1 = xy_1 + x'y_2$$
$$Y_2 = xy_1' + x'y_2$$

Step 2:
- Thenext step is to plottheY₁and Y₂functions in amap

Map for
$Y_1 = xy_1 + x'y_2$

Map for
$Y_2 = xy'_1 + x'y_2$

- Combining the binary values in corresponding squares the following transition table is obtained
- The transition table shows the value of $Y=Y_1Y_2$ inside each square. Those entries where $Y=y$ are circled to indicate a stable condition.
- The circuit has four stable total states, $y_1y_2x=000, 011, 110,$ and 101 and four unstable total states-$001, 010, 111$ and 100.
- The state table of the circuit is shown below:

Present State		Next State			
		$x = 0$		$x = 1$	
0	0	0	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	1	1	1	1	0

- This table provides the same information as the transition table.

Step 3:
Transition table
- The transition table is obtained by combining the maps for Y_1 and Y_2.

- The transition table is a table which gives the relation between present state, input and next state. If the secondary variables $y_1 y_2$ is same as excitation variables $Y_1 Y_2$, the state is said to be stable.
- The stable states are indicated by circles. An uncircled entry represents an unstable state.

3

- In a transition table, usually there will be at least one stable state in each row. Otherwise, all the states in that row will be unstable.

Step 4:
Flowtable

- Ina flow table thestatesarenamed bylettersymbols. Examples of flow tables areas follows:

(a) Four states with one input

- In order to obtain thecircuitdescribed bya flow table, it is necessaryto assign to each stateadistinct value.

3. **Explain in detail about design procedure of asynchronous sequential circuits.** **(Dec-11, 12, May-12, 13, 15)**

DESIGN PROCEDURE OF ASYNCHRONOUS SEQUENTIAL CIRCUITS

- Thereareanumberofstepsthatmustbecarriedoutinordertominimizethecircuitcomplexityandto produceastable circuitwithoutcritical races.
- Thedesign steps areas follows:
  - ➢ Obtain a primitive flow table from thegivenspecification.
  - ➢ Reducethe flow table bymerging rows inthe primitive flow table.
  - ➢ Assign binarystates variables to eachrowof the reduced flowtable to obtain thetransition table.
  - ➢ Assign outputvalues to thedashes associated with the unstable states to obtainthe output maps.
  - ➢ Simplifythe Boolean functions of the excitation and output variables and draw the logic diagram.
  - ➢ Thedesign process will be demonstrated bygoingthrough aspecificexample

Example:
Designagatedlatchcircuitwithtwoinputs,G(gate)andD(data),andoneoutputQ.The gatedlatchis amemory elementthatacceptsthevalueofDwhenG=1andretainsthisvalueafterGgoesto0.Once G =0, a change in D does notchangethe valueofthe output Q.

(Or)

Design an asynchronous sequential circuit with two inputs D and G with one output Z. Whenever G is 1, input D is transferred to Z. When G is 0, the

4

output does not change for any change in D. Use SR latch for implementation of the circuit.   (Dec-2018)

PrimitiveFlowTable

- Aprimitiveflowtableisaflowtablewithonlyonestabletotalstateineachrow.Thetotalsta teconsists ofthe internal statecombined with the input.
- To derivethe primitive flow table, first a tablewithallpossible total states in thesystem is needed:

State	Inputs D	G	Output Q	Comments
a	0	1	0	$D = Q$ because $G = 1$
b	1	1	1	$D = Q$ because $G = 1$
c	0	0	0	After state a or d
d	1	0	0	After state c
e	1	0	1	After state b or f
f	0	0	1	After state e

- Eachrowintheabovetablespecifiesatotalstate;theresulting primitivetableforthegatedlatchis shown below:

Inputs DG

States	00	01	11	10
a	$c,-$	$(a),0$	$b,-$	$-,-$
b	$-,-$	$a,-$	$(b),1$	$e,-$
c	$(c),0$	$a,-$	$-,-$	$d,-$
d	$c,-$	$-,-$	$b,-$	$(d),0$
e	$f,-$	$-,-$	$b,-$	$(e),1$
f	$(f),1$	$a,-$	$-,-$	$e,-$

- First,fillinonesquareineachrowbelonging tothestablestateinthatrow.
- Nextrecallingthat both inputsarenotallowedtochangeatthesametime.
- Thenenterdashmarksineachrowthatdiffersintwo ormorevariablesfromtheinputvariablesassociatedwiththestablestate.

Reduction of primitive flow table:

- Two or more rows in the primitive flow table can be merged into one row if there are non- conflicting states and outputs on each of the columns.
- This can be done by implication table and merger diagram.
- The implication table has all states except the first vertically and all states except the last across bottom horizontally.
- The tick (√)mark denotes that the pair (rows) is compatible.

5

- Two states are compatible, if the states are identical with non-conflicting outputs.
- The cross (X) mark implies non-compatible.

Fig. : Implication table

- The compatible pairs are

    (a,b), (a,c), (a,d), (b,e), (b,f), (c,d), (e,f)

Merger Diagram:
- The maximum compatible sets can be obtained from merger diagram as shown in figure.
- The merger diagram is a graph in which each state is represented by a dot placed along the circumference of a circle.
- Lines are drawn between any two corresponding dot that form a compatible pair.
- Based on the geometrical patterns formed by the lines, all the possible compatibilities can be obtained.
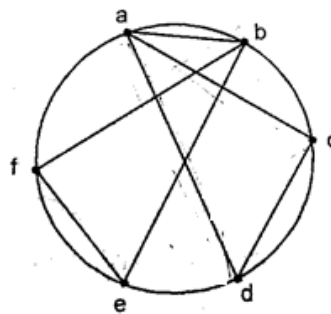
Fig. : Merger Diagram

- An isolated dot represents a state that is not compatible with any other state.
- A line represents a compatible pair.
- A triangle constitutes a compatible with three states.
- An n-state compatible is represented in the merger diagram by an n-sided polygon with all its diagonal connected.
- So, the maximal compatibilities are

    (a,b) , (a,c,d) , (b,e,f)

Closed covering condition:
- In the above, if only (a,c,d) and (b,e,f) are selected, all the six states are incuded.

6

- This set satisfies the covering condition.
- Thus, the rows a,c,d can be merged as one row and b,e,f states can be merged as another row.

States \ DG	00	01	11	10
a, c, d	©, 0	ⓐ, 0	b, –	ⓓ, 0
b, e, f	ⓕ, 1	a, –	ⓑ, 1	ⓔ, 1

Fig.   : Reduced flow table

- Consider a,c,d =a and b,e,f =b

States \ DG	00	01	11	10
a	ⓐ, 0	ⓐ, 0	b, –	ⓐ, 0
b	b, 1	a, –	ⓑ, 1	ⓑ, 1

Fig.   : Reduced flow table with common symbol

- A race free binary assignment is made and transition table and output map is obtained.a -> 0, b-> 1

Q \ DG	00	01	11	10
0	0	0	1	0
1	1	0	1	1

Fig. : Transition table

Q \ DG	00	01	11	10
0	0	0	1	0
1	1	0	1	1

Fig. : Output map

Logic Diagram using SR Latch:

7

Excitation table of SR flip-flop is used to find expressions for S and R.

Excitation Table of SR Flip-Flop

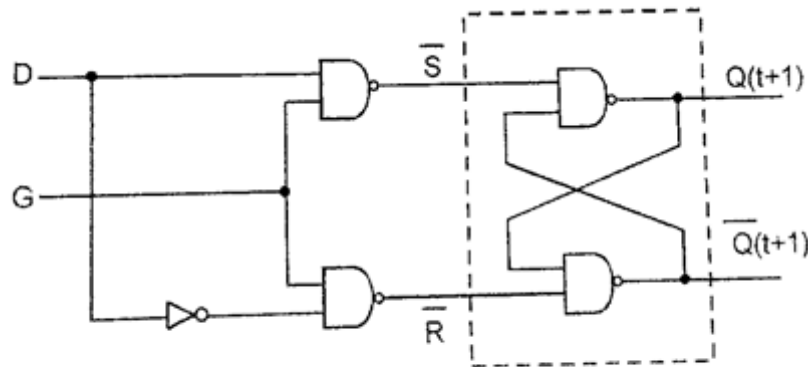| Q_n	$Q_n + 1$	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

$S = DG$

$R = \overline{D}\,G$

Fig. : Logic diagram using SR latch

4. Explain in detail about race -Free State assignment. (Dec-10, 14)
Explain the problems in asynchronous circuits with examples.

RACE -FREE STATE ASSIGNMENT
- Once areducedflow tablehasbeenderivedfor anasynchronoussequentialcircuit,thenextstepinthe designistoassignbinary variablestoeachstablestate.
- Theprimaryobjectiveinchoosing aproperbinary state assignmentisthepreventionofcriticalraces.
- Criticalracescanbeavoidedbymakingabinarystate assignmentinsuchawaythatonly onevariablechangesatanygiventimewhenastatetransitionoccurs in the flow table.

Cycles
- A cycle occurs when an asynchronous circuit makes a transition through a series of unstable state.
- When a state assignment is made so that it introduces cycles, care must be

8

taken that it terminates with a stable state.
- Otherwise, the circuit will go from one unstable state to another, until the inputs are changed.
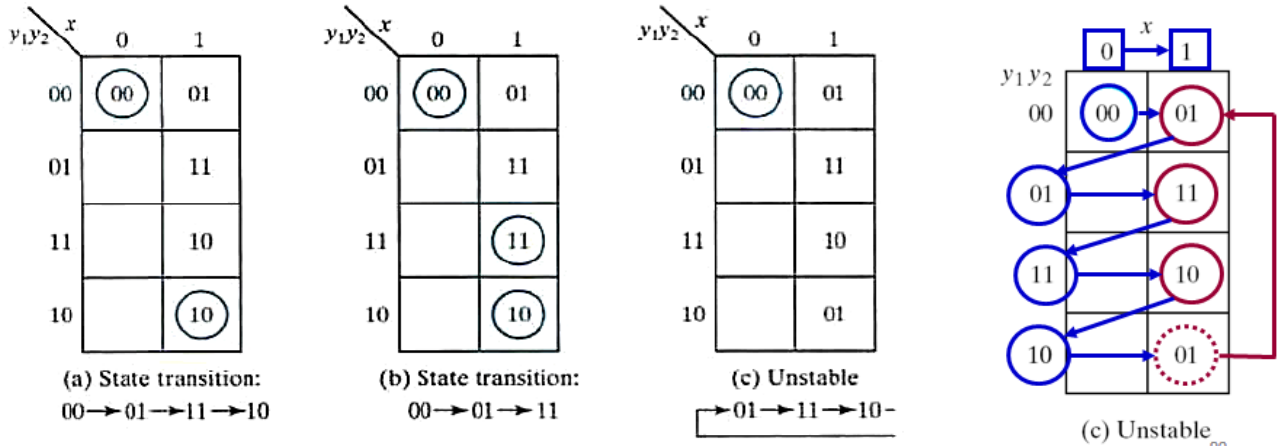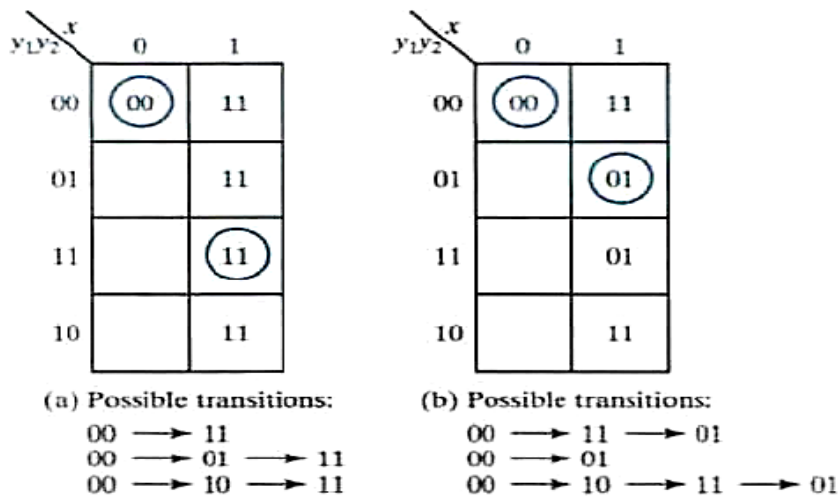- Examples of cycles are:

(a) State transition:
$00 \rightarrow 01 \rightarrow 11 \rightarrow 10$

(b) State transition:
$00 \rightarrow 01 \rightarrow 11$

(c) Unstable
$\rightarrow 01 \rightarrow 11 \rightarrow 10 -$

(c) Unstable

Fig: Examples of cycles

Race Conditions

- A race condition exists in an asynchronous circuit when two or more binary state variables change value in response to a change in an input variable.
- When unequal delays are encountered, a race condition may cause the state variable to change in an unpredictable manner.
- If the final stable state that the circuit reaches **does not depend on the order** in which the state variables change, the race is called a noncritical race.
- If the final stable state that the circuit reaches **depends on the order** in which the state variables change, the race is called a critical race.
- **Examples of noncritical races** are illustrated in the transition tables below:

(a) Possible transitions:
$00 \longrightarrow 11$
$00 \longrightarrow 01 \longrightarrow 11$
$00 \longrightarrow 10 \longrightarrow 11$

(b) Possible transitions:
$00 \longrightarrow 11 \longrightarrow 01$
$00 \longrightarrow 01$
$00 \longrightarrow 10 \longrightarrow 11 \longrightarrow 01$

- Initial stable state is $y_1y_2x = 000$ and then input changes from 0 to 1.
- The state variables y_1y_2 must change from 00 to 11, (race condition).

Possible transitions are

$00 \rightarrow 11$

$00 \rightarrow 01$(y2 faster) $\rightarrow 11$

$00 \rightarrow 10$(y1 faster) $\rightarrow 11$

9

- In all cases final stable state is same, which results in a non-critical race condition.
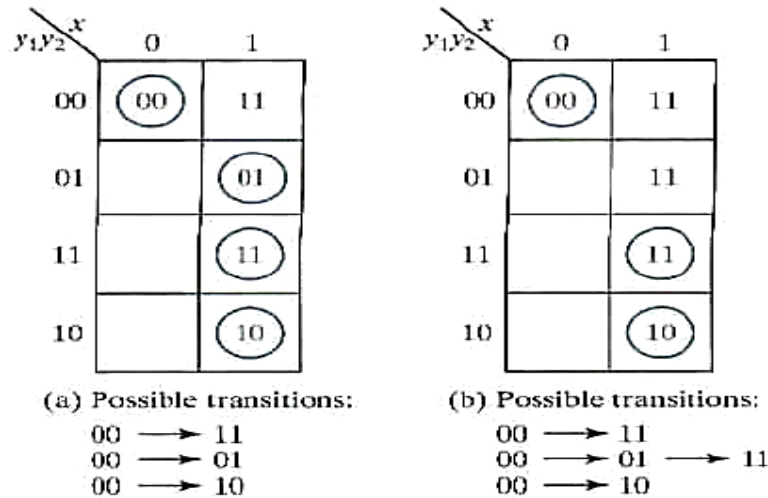- **Examples of critical races** areillustratedin the transition tables below:

(a) Possible transitions:
$$00 \longrightarrow 11$$
$$00 \longrightarrow 01$$
$$00 \longrightarrow 10$$

(b) Possible transitions:
$$00 \longrightarrow 11$$
$$00 \longrightarrow 01 \longrightarrow 11$$
$$00 \longrightarrow 10$$

Fig: Examples ofcritical races

- The initial stable state is $y_1y_2 \, x=000$ and let us consider that the input changes from 0 to 1. Then, the state variables must change from 00 to 11.
- If they change simultaneously, the final total state is 111.
- Due to unequal propagation delay, if y_2 changes to 1 before y_1 does, then the circuit goes to total stable state $y_1y_2 \, x=011$ and remains there.
- If y_1 changes first, then the circuit will be in total stable state is $y_1y_2 \, x=101$.
- Hence the race is critical because the circuit goes to different stable states depending on the order in which the state variables change.

Three-RowFlow-Table Example

(a) Flow table

(b) Transition diagram
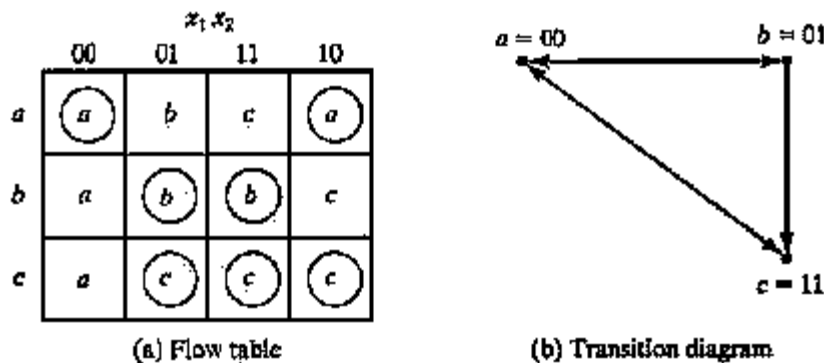
Fig: Three rowflowtable example

- Toavoidcriticalraces,wemustfindabinarystateassignmentsuchthatonly onebinaryvariable changesduring eachstatetransition.
- Anattempttofindsuchanassignmentisshownninthetransition diagram.
  - ➢ Statea isassignedbinary 00,
  - ➢ Statecisassignedbinary 11.
- Thisassignmentwillcausea criticalraceduringthetransitionfromatoc .

10

- Because there are two changes in the binary state variables and the transition from a to c may occur directly or pass through b.
- Note that the transition from c to a also causes a race condition, but it is noncritical because the transition does not pass through other states
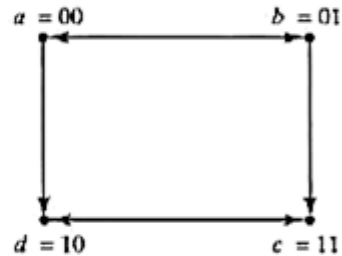
(a) Flow table   (b) Transition diagram

Fig:Flow table with an extra row

- A race-free assignment can be obtained if we add an extra row to the flow table.
- The use of a fourth row does not increase the number of binary state variables, but it allows the formation of cycles between two stable states.
- The transition table corresponding to the flow table with the indicated binary state assignment is shown in Fig.
- The two dashes in row d represent unspecified states that can be considered d don't-care conditions.
- However, care must be taken not to assign 10 to these squares, in order to avoid the possibility of an unwanted stable state being established in the fourth row.
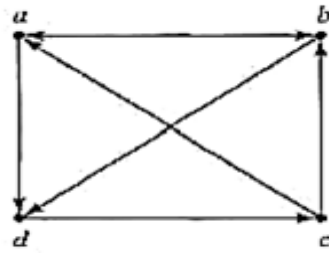

Fig: Transition table

Four-Row Flow-Table

- A flow table with four rows requires a minimum of two state variables.
- Although a race-free assignment is sometimes possible with only two binary state variables, in many cases the requirement of extra rows to avoid critical races will permit to use of three binary state variables
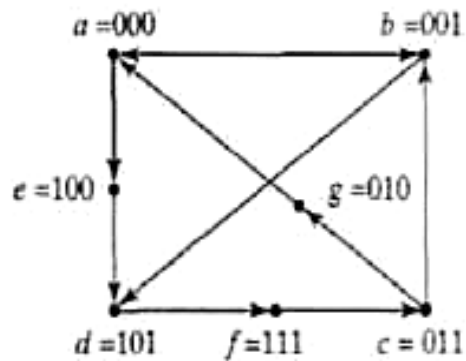
(a) Flow table   (b) Transition diagram

Fig:Four-rowflow-table example

- Thefollowing figureshowsastateassignmentmapthatissuitableforany four-rowflowtable.Statesa, b,canddaretheoriginalstatesande,fandgareextrastates.
- Thetransitionfromatodmustbe directedthroughtheextrastateetoproduceacyclesothatonly onebinaryvariablechangesatatime.
- Similarly,thetransitionfromctoaisdirectedthroughgandthetransitionfromdtocgoesthroughf.
- Byusingtheassignmentgivenby themap,thefour-rowtablecanbeexpandedtoaseven-rowtablethat is freeofcritical races.

(a) Binary assignment   (b) Transition diagram

Fig: Choosing extrarowsfor theflowtable

- Notethatalthoughtheflowtablehas sevenrowsthereareonlyfourstablestates.
- Theuncircledstatesin the three extra rowsaretheremerelyto providearace-freetransition between thestablestates.

12

Fig: State assignment to modified flow table

Multiple-Row Method

- The method for making race-free state assignments by adding extra rows in the flow table is referred to as the shared-row method.
- A second method called the multiple-row method is not as efficient, but is easier to apply.
- In multiple-row assignment each state in the original row table is replaced by two or more combinations of state variables.

(a) Binary assignment

(b) Flow table

Fig: Multiple row assignment

- There are two binary state variables for each stable state, each variable being the logical complement of the other.

****

13

5. **Explain various types of hazards in sequential circuits design and the methods to eliminate them.** (Dec-12, 14, 17) (Dec-2018)

HAZARDS

- **Hazards** areunwantedswitchingtransientsthatmay appearattheoutputofacircuitbecausedifferent pathsexhibitdifferentpropagationdelays.
- Hazardsoccurincombinationalcircuits,wheretheymay causeatemporary falseoutputvalue.
- But inasynchronous sequentialcircuits hazardsmay resultin a transitionto awrongstable state.

Types of Hazards
  - ➢ **Static Hazard**
  - ➢ **Dynamic Hazard**
  - ➢ **Essential Hazard**

Static Hazard

- Static Hazard is a condition which results in a single momentary incorrect output due to change in a single input variable when the output is expected to remain in the same state.
- The static hazard may be either static-0 or Static -1.

Hazards inCombinational Circuits

- Ahazardisaconditioninwhichachangeinasinglevariableproducesamomentary changeinoutput when no changein output should occur.
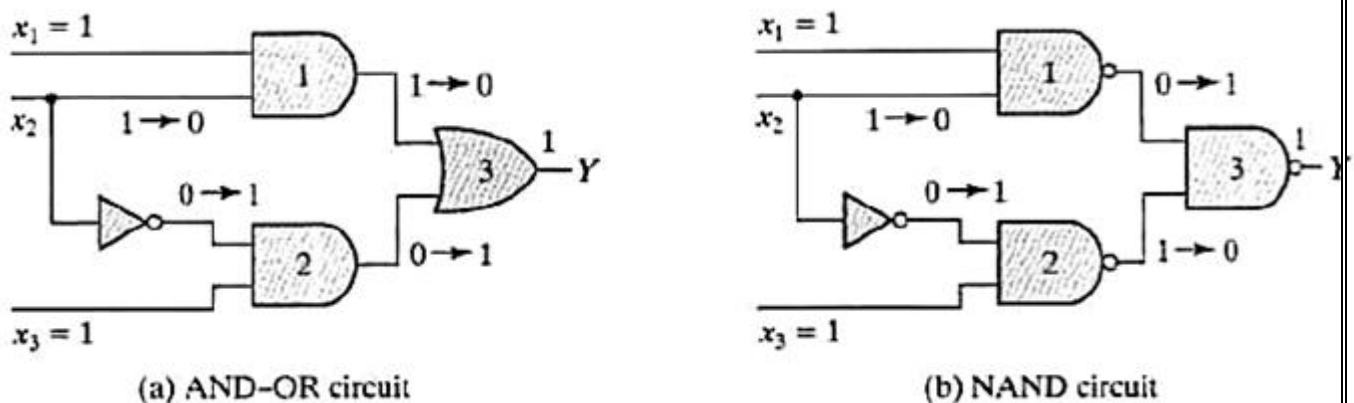

(a) AND–OR circuit   (b) NAND circuit

Fig: Circuits with Hazards

- Assumethatallthreeinputsareinitially equalto1.Thiscausestheoutputofgate110be1,thatofgate 2tobe0andthatofthecircuittobe1.Nowconsiderachangeinx2from1to0.
- Thentheoutputof gate1changesto0and thatofgate2changesto1,leaving theoutputat 1.
- However,theoutputmay

14

momentarilygoto0ifthepropagationdelaythroughtheinverteristakenintoconsideration.

- Thedelay in theinvertermaycausetheoutput ofgate1 to changeto 0 beforethe output of gate2 changes to1.
- Thetwo circuits shown in Figimplement theBoolean function in sum-of-products form:

$$Y = x_1x_2 + \overline{x_2}x_3$$

- This type of implementation may cause the output to go to 0 when it should remain a 1. If however, the Circuit is implemented instead in product-of-sums form namely,

$$Y = (x_1 + \overline{x_2})(x_2 + x_3)$$

then the output may momentarily go to 1 when it should remain 0.

- The first case is referred to a static 1-hazard and the second case as static 0-hazard.
- Athirdtypeofhazard,knownas**dynamichazard**,causestheoutputtochangethreeormoretimes when itshould changefrom1 to 0 or from 0 to 1.

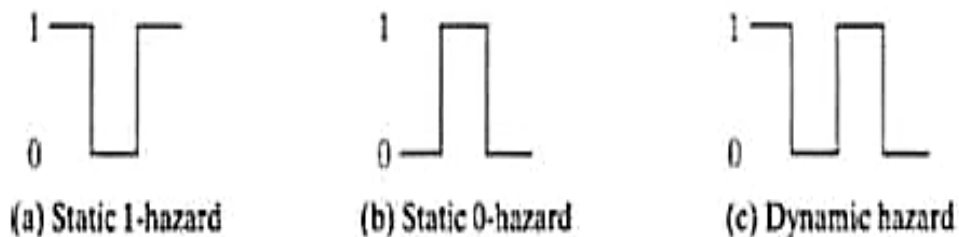

(a) Static 1-hazard   (b) Static 0-hazard   (c) Dynamic hazard

Fig: Types ofhazards

- The change in $x_2$from1to0movesthecircuitfromminterm111tominterm101.The hazard exists because the changein input results in a different product term coveringthe twominterm.
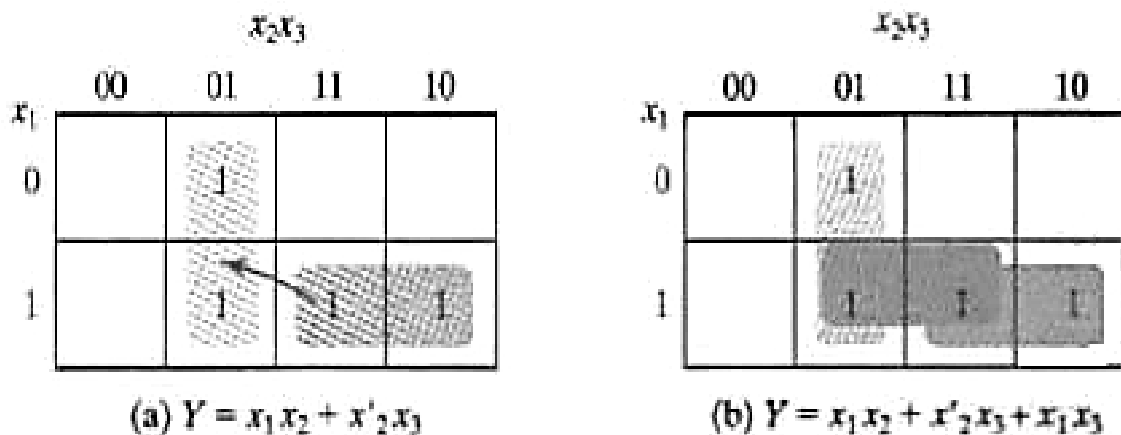

(a) $Y = x_1x_2 + x'_2x_3$ (b) $Y = x_1x_2 + x'_2x_3 + x_1x_3$

Fig: Illustrates hazardandits removal

- Minterm111iscoveredby theproducttermimplementedingate1andminterm101iscoveredbythe

15

producttermimplementedingate2.

- Theremedy foreliminating ahazardistoenclosethetwominterms withanotherproducttermthatoverlapsbothgroupings.
- Thehazard-freecircuitobtainedbysucha configurationisshowninfigure below.
- The extra gate inthecircuitgeneratesthe producttermx1x3.
- In general,hazardsincombinationalcircuitscanberemoved by coveringany twomintermsthatmay producea hazardwithaproducttermcommontoboth.
- Theremovalofhazardsrequirestheadditionof redundantgates to thecircuit.
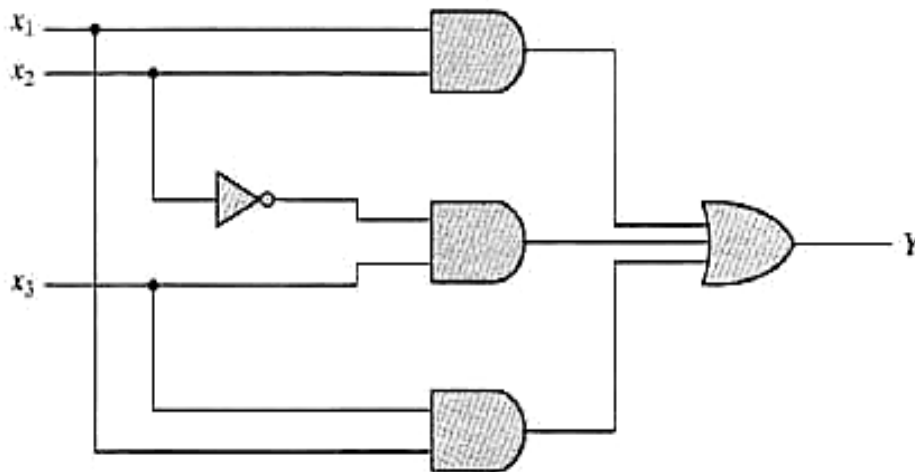
Hazards in SequentialCircuits:

(a) Logic diagram

$$Y = x_1x_2 + x'_2y$$
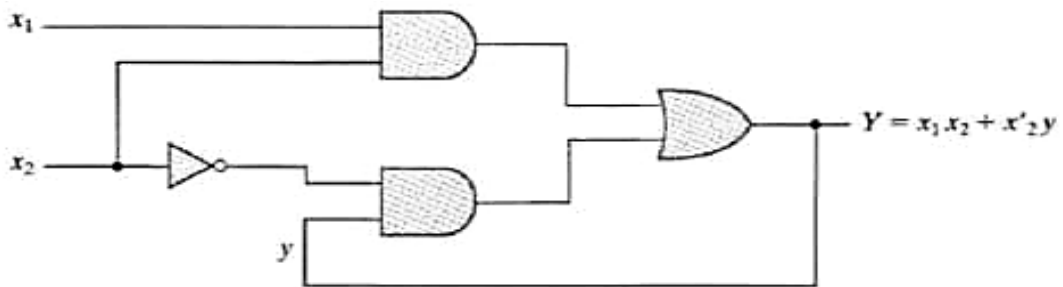
(b) Transition table

(c) Map for Y

Fig: HazardinanAsynchronous sequential circuit

- Innormalcombinational-circuitdesignassociatedwithsynchronoussequentialcircuits,hazardsareof

16

- noconcern,sincemomentaryerroneoussignalsare notgenerallytroublesome.
- However,ifamomentary incorrectsignalisfed backinanasynchronoussequentialcircuit,itmay causethecircuittogotothe wrongstable state.
- Ifthe circuitis in total stable state $yx_1x_2=111$ and input x_2 changesfromIto0, the next total stablestate shouldbe110.However,because ofthehazard,outputYmay goto0momentarily.
- Ifthisfalsesignal feedsbackintogate2beforetheoutputoftheinvertergoesto1,theoutputofgate2willremainat0 andthecircuitwillsswitchtotheincorrecttotalstablestate 010.
- Thismalfunctioncanbe eliminatedby adding an extragate.

Essential Hazards
- **Essential**hazardiscausedby unequaldelaysalongtwoormorepathsthatoriginatefromthesame input.
- Anexcessivedelaythroughaninvertercircuitincomparisontothedelayassociatedwiththe feedback path maycausesuchahazard.
- Essentialhazards cannotbecorrectedby addingredundantgatesasinstatichazards.The problemthat they imposecanbecorrectedbyadjustingtheamountofdelay intheaffectedpath.
- Toavoidessential hazards,eachfeedbackloopmustbehandledwithindividualcaretoensurethatthedelayinthe feedbackpathislongenoughcomparedwithdelaysofothersignalsthatoriginate fromtheinput terminals.

****

CLASSIFICATION OF MEMORIES

6. **Discuss the classification of ROM and RAM memories.**
- A**memoryunit**isadevicetowhichbinary informationistransferredforstorageandfromwhich informationisretrievedwhenneededfor processing.
- When data processing takes place, information from memory is transferred to selected registers in the processing unit.
- Amemoryunitisacollectionof cells capableof storingalargequantityof binaryinformation.

TWO TYPESOFMEMORIES:
- Therearetwotypes ofmemoriesthatare usedindigitalsystems:
  - Random-accessmemory(RAM)and Read-onlymemory(ROM)
  (i) Random-access memory (RAM)
  - RAM stores new information for later use.

- The process of storing new information into memory is referred to as a memory "write" operation.
- The process of transferring the stored information out of memory is referred to as a memory "read" operation.
- RAM can perform both write and read operations.

(ii) Read-only memory (ROM)
- ROM can perform only the read operation.
- This means that suitable binary information is already stored inside memory and can be retrieved or read at any time.
- However, that information cannot be altered by writing.

- ROM is a programmable logic device (PLD).
- The binary information that is stored within such a device in some fashion and then embedded within the hardware in a process is referred to as programming the device.

7. Explain in detail about read-only memory.
READ-ONLY MEMORY(ROM)

- ROM is a non-volatile memory. It can hold data even if power is turned off.
  - A ROM is essentiallyamemorydevicein whichpermanent binaryinformation is stored.
  - It is embedded in the unit and cannot be altered.
  - It consists of 'k' inputs and 'n' outputs.
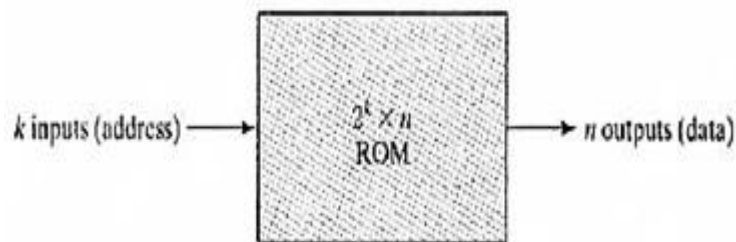

Fig: ROM block diagram

ROM Organization:
- The inputs provide the address for memory, and the outputs give the data bits of the stored word that is selected by the address.
- Number of words is get from number of address inputs, here it is 'k', hence 2^k words of n bits each is present in the memory.
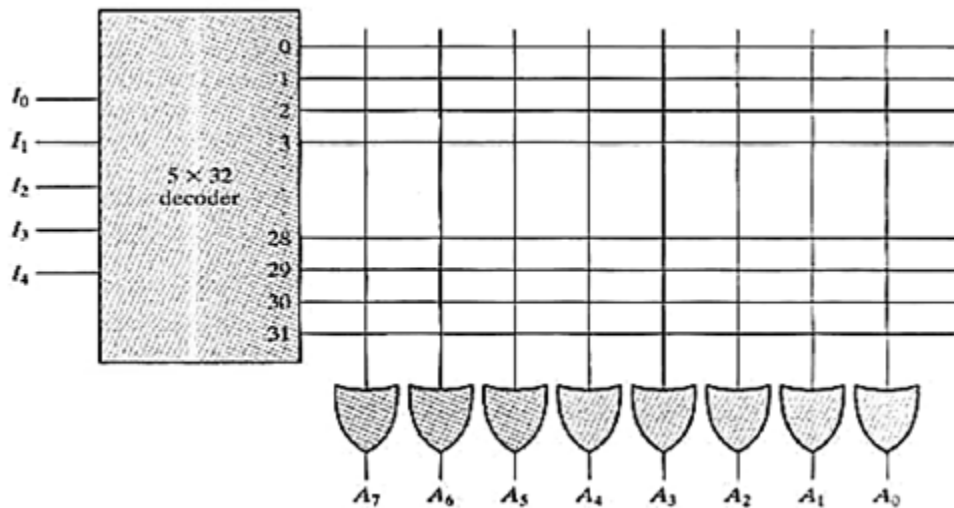
Fig: Internal logic ofa32×8 ROM

Example:

- The five inputs are decoded into 32 distinct outputs by means of a 5×32 decoder. Each output of the decoder represents a memory address.
- The 32 outputs of the decoder are connected to each of the eight OR gates. Each OR gate must be considered as having 32 inputs.
- Each output of the decoder is connected to one of the inputs of each OR gate.
- Since each OR gate has 32 input connections and there are 8 OR gates, the ROM contains 32 x 8 = 256 internal connections.
- A programmable connection between **two lines** is logically equivalenttoaswitchthatcanbealteredto beeither**closed**(twolinesareconnected)or**open**(twolinesare disconnected).
- Theprogrammableintersection between two lines issometimes called a**cross point**.

ROM Truth Table (Partial)

Inputs					Outputs							
I_4	I_3	I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
⋮								⋮				
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

- For anexample,programmingtheROMaccordingtothetruthtablegivenbytable.
- Every0listedinthe truthtablespecifiestheabsence ofaconnectionandevery1listedspecifiesapaththatisobtainedby a connection.
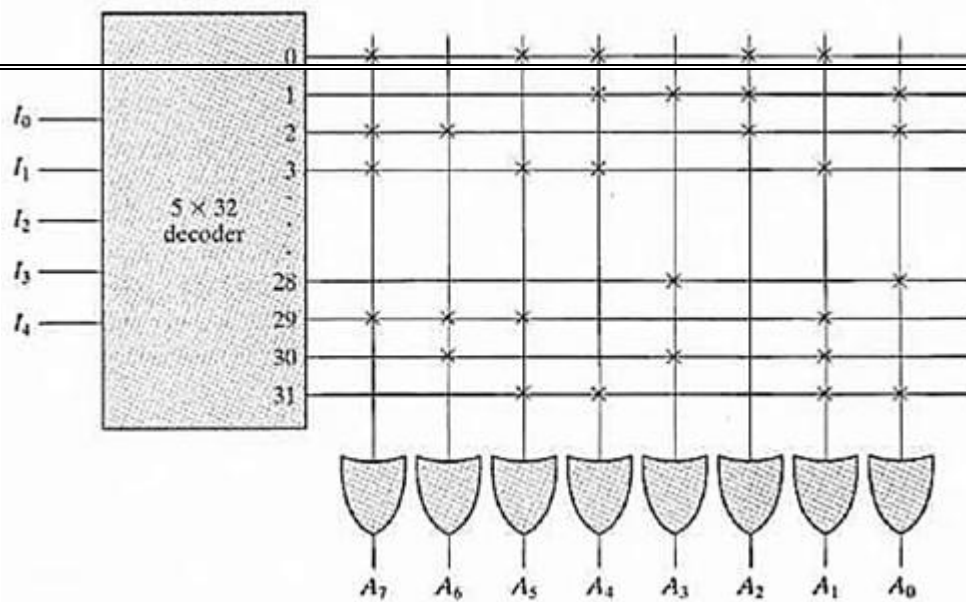
19

Fig:ProgrammingtheROM according to ROM truth table

TYPES OF ROM

8. **Briefly explain EPROM and EEPROM technology.**

The required paths in aROM maybeprogrammedin fourdifferent ways.

- **Maskprogramming**
  - Donebythesemiconductorcompanyduringthe lastfabricationprocessoftheunit.
  - Thisprocedureiscostly becausethevendorchargesthecustomera special fee forcustommaskingthe particular ROM.

- **Programmable read-onlymemory-PROM**.
  - Economical for small quantity.
  - ThefusesinthePROMareblownbytheapplicationofahigh-voltage pulsetothedevicethroughaspecialpin.
  - Ablownfusedefinesabinary0stateandanintactfusegives abinary1state.
  - Theprocedureisirreversibleandonce programmed; the fixed pattern is permanent and cannot be altered.

- **ErasablePROMorEPROM**
  - This   canberestructuredtothe initialstateeventhoughithasbeenprogrammedpreviously.
  - It is erased by placing undera specialultravioletlightfor agivenlengthoftime.

- **ElectricallyerasablePROM(EEPROM)or Electrically Alterable PROM (EAPROM)**
  - Electrical signals are used to erase the previouslyprogrammedconnectionsinsteadof ultravioletlight.
 - Theadvantage isthatthe device canbeerasedwithout removingitfrom its socket.

9. **Explain in detail about Random Access Memory.**

RANDOM ACCESS MEMORY (RAM):

- RAM stores new information for later use.
- The process of storing new information into memory is referred to as a memory write operation.
- The process of transferring the stored information out of memory is referred to as a memory read operation.
- RAM can perform both write and read operations.

RAM Organization:

- A block diagram of a memory unit is shown in Fig.
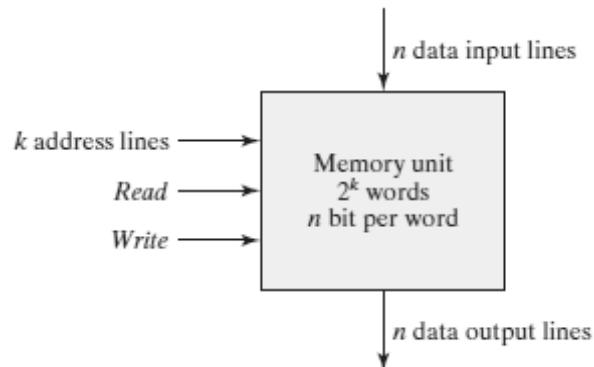
- The n data input lines provide the information to be stored in memory, and the n data output lines supply the information coming out of memory.
- The k address lines specify the particular word chosen among the many available.
- The two control inputs specify the direction of transfer desired: Write input and Read input.
- A memory unit stores binary information in groups of bits called words.
- Each word in memory

  is assigned an identification number called an address starting from 0 up to 2^k-1. where k is the number of address lines.
- Consider for example, a memory unit with a capacity of 1K words of 16 bits each. Since
- Here 1024 x 16 RAM consists of 10 x 1024 decoder (1K=1024bytes=2^{10}), where the decoder inputs are the 10 address lines .
- The decoder accepts the address lines and provides the path needed to select the word specified.

Fig: Contents of a 1024 × 16 memory

Read andwriteoperations:
- Thetwo operations that RAM can perform arethewriteand read operations.

Stepsto Write operation as follows:
> Applythe binaryaddress of thedesired word to the address lines.
> Applythe data bits that must be stored in memoryto thedatainputlines.
> Activate thewrite input.

Stepsto Read operation as follows:
> Applythe binaryaddressof thedesiredword to the address lines.
> Activate thereadinput.

- Thememoryenableorchipselectisusedtoenabletheparticularmemorychipina multichipimplementationofalargememory.

Control Inputs to Memory Chip		
Memory Enable	**Read/Write**	**Memory Operation**
0	X	None
1	0	Write to selected word
1	1	Read from selected word

- Whenthememory enableisinactive,thememory chipis notselectedandnooperationisperformed.Whenthememoryenableinputisactive,the read/write operation to be performed.

TYPES OF RAM

10. Explain the types of RAM with neat diagram.
RAM is classified into two types.
1. Static RAM
2. Dynamic RAM

STATIC RAM (SRAM):

22

- Memories that consist of circuits capable of retaining their state as long as power is applied are known as static memories.
- Two basic SRAM cell technologies are
    - Bipolar and MOS.
- All those types use cross-coupled transistors to make up the basic flip-flop storage cell.

Bipolar Static RAM cell: (May-18)

- It is implemented using TTL (Transistor-Transistor Logic) multiple emitter technology.
- It can store either 0 or 1 as long as power is applied.

Operation:

- Row and column select lines select a cell.
- The Q1 and Q2 are cross coupled inverters.
- A "1" is stored in the cell if Q1 is ON and Q2 is OFF.
- A "0" is stored in the cell if Q2 is ON and Q1 is OFF.
- When pulsing HIGH on Q1 emitter (SET), State is changed to '0'.
- When pulsing HIGH on Q2 emitter (RESET), State is changed to '1'
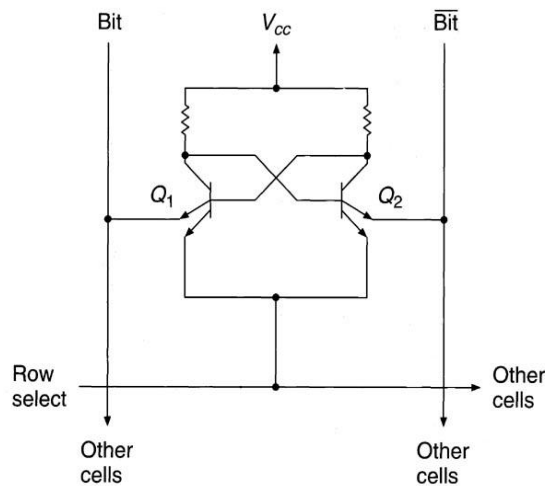
a. Bipolar SRAM cell

## MOSFET Static RAM Cell:
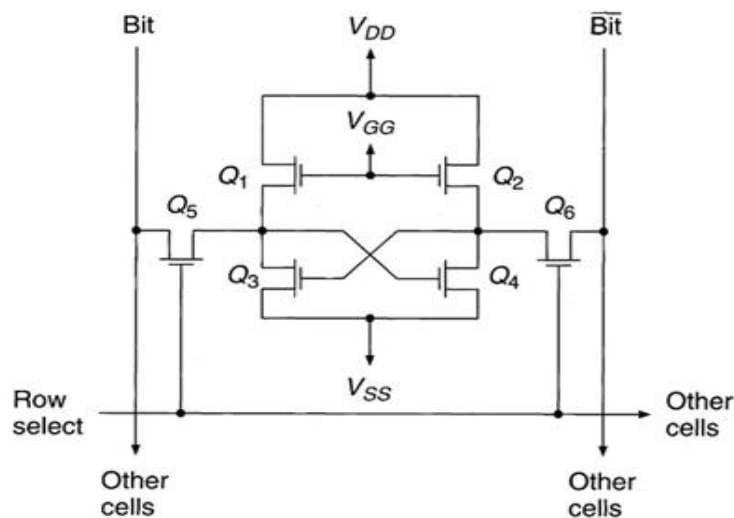
b.   MOS SRAM cell

Operation:

23

- In the basic NMOS cell, Q_1 and Q_2 are always biased to act as a Load Resistor for Q_3 and Q_4.
- The Data in a cell can be read by setting ROW_SELECT = 1 to turn on Pass Transistors Q_5, Q_6.
- The Data from cell is then "passed" to the BIT Line and (BIT)′ Line.
- To store a '0', place a 0 on the bit line and set ROW_SELECT = 1. This turns on the Pass Transistors (Q_5, Q_6) to place a 0 to Q_4 (it is off). Q_3 is then ON to store the 0.
- A '1' can be stored in a similar fashion.

11. Write note on dynamic RAM cell.

Dynamic RAM cell:
Dynamic RAM (DRAM) stores data as a charge on capacitors.
- The stored charge on the capacitors tends to discharge with time, and the capacitors must be periodically recharged by refreshing the dynamic memory.
- Refreshing is done by cycling through the words every few milliseconds to restore the decaying charge.
- DRAM offers reduced power consumption and larger storage capacity in a single memory chip.
- Memory units that lose stored information when power is turned off are said to be volatile.
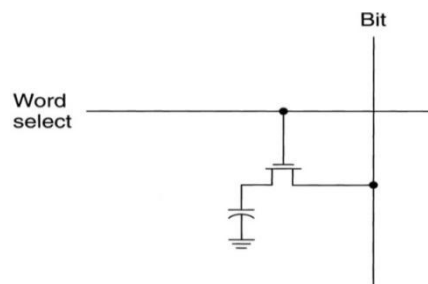
Comparison of Static and Dynamic RAMS.

S.No	Static RAM	Dynamic RAM
1	Static RAM contains less memory cells per unit area	Dynamic RAM contains more memory cells as compared to static RAM per unit area.
2	It has less access time hence faster memories.	Its access time is greater than static RAMs.
3	Static RAM consists of number of flip flops. Each flip flop stores one bit.	Dynamic RAMs store the data as a charge on the capacitor. It consists of MOSFET and the capacitor.
4.	Refreshing circuitry is not required.	Refreshing circuitry is required to maintain the charge on the capacitors after every few milliseconds
5	Cost is more	Cost is less

PROGRAMMABLE LOGIC DEVICES (PLDs)

12. Write brief notes on combinational programmable logic device PLD.

Combinational PLDs

- The PROMisa combinationalprogrammable logicdevice (PLD)- anintegratedcircuitwith programmablegatesdividedintoanANDarrayandanORarray toprovideanAND-ORsumof- product implementation.

- Thereare threemajortypesofcombinational PLDs,differing intheplacementoftheprogrammable connections in the AND-OR array.
    1. PROM- FixedANDarray andaprogrammableORarray.
    2. PAL - ProgrammableAND arrayand a fixed OR array.
    3. PLA - ProgrammableAND arrayand a programmable OR array.

(a) Programmable read-only memory (PROM)

(b) Programmable array logic (PAL)

(c) Programmable logic array (PLA)

13. Write short notes on PLA.

PROGRAMMABLE LOGIC ARRAY (PLA)   **(Dec-08, 12, 17, May-10, 11)**

- Programmablelogicarrays(PLAs) is a type of fixed architecture logic devices with programmable AND gates followed by programmable OR array.
- PLA is used to implement a complex combinational circuit.
- The AND and OR gates inside the PLA are initially fabricated with fuses among them.
- The specific Boolean functions are implemented in sum of products (SOP) form by blowing appropriate fuses and leaving the desired connections.
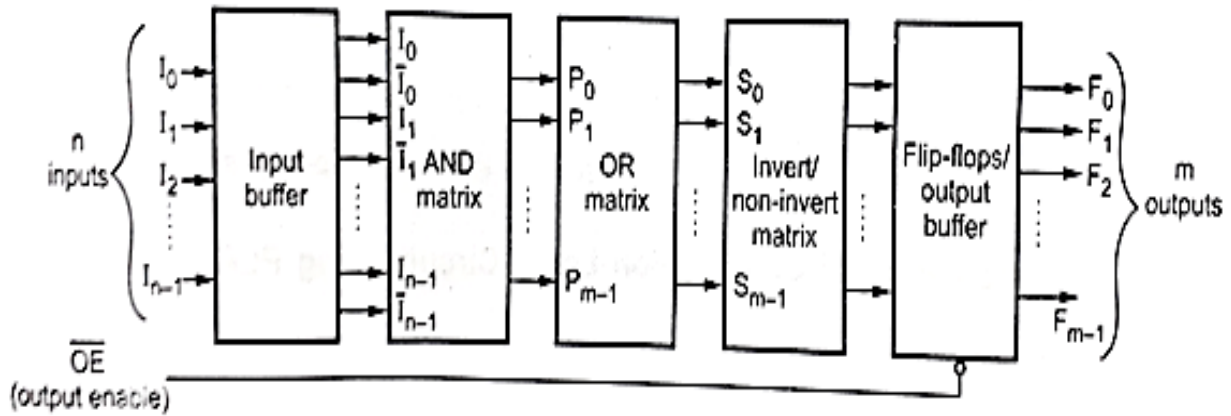
Fig: Block diagram of a PLA

Input Buffer:

- Input buffers are provided in the PLA to limit loading of the sources that drive the inputs. They also provide inverted and non-inverted form of inputs at its output.

(a)

(b)
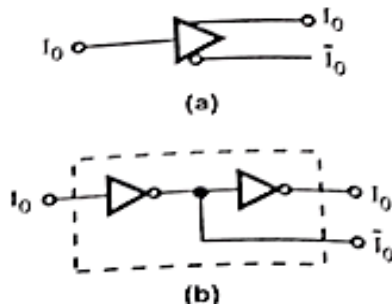
Output Buffer:

- The driving capacity of PLA is increased by providing buffers at the output. They are usually TTL compatible. The output buffer may provide totem-pole, open collector or tri-state output.
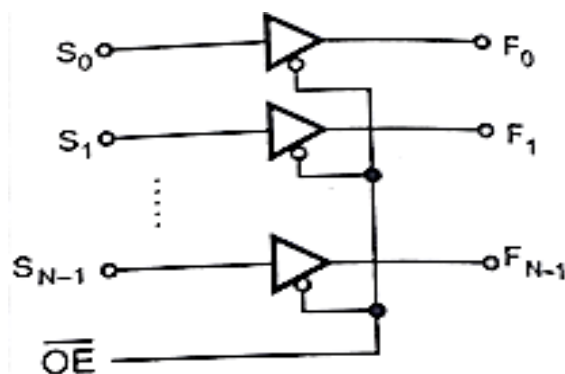
Output through Flip-flops:

- For the implementation of sequential circuits we need memory elements, flip-flops and combinational circuitry for deriving the flip-flop inputs. To satisfy both the needs some PLAs are provided with flip-flop at each output.

26
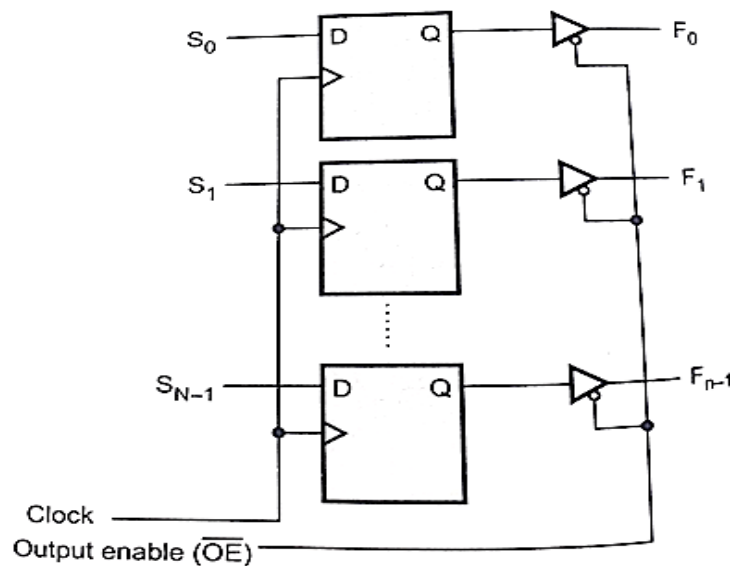
For an example, the Boolean expressions are,

$$F_1 = A\bar{B} + AC + \bar{A}B\bar{C}$$

$$F_2 = \overline{(AC + BC)}$$
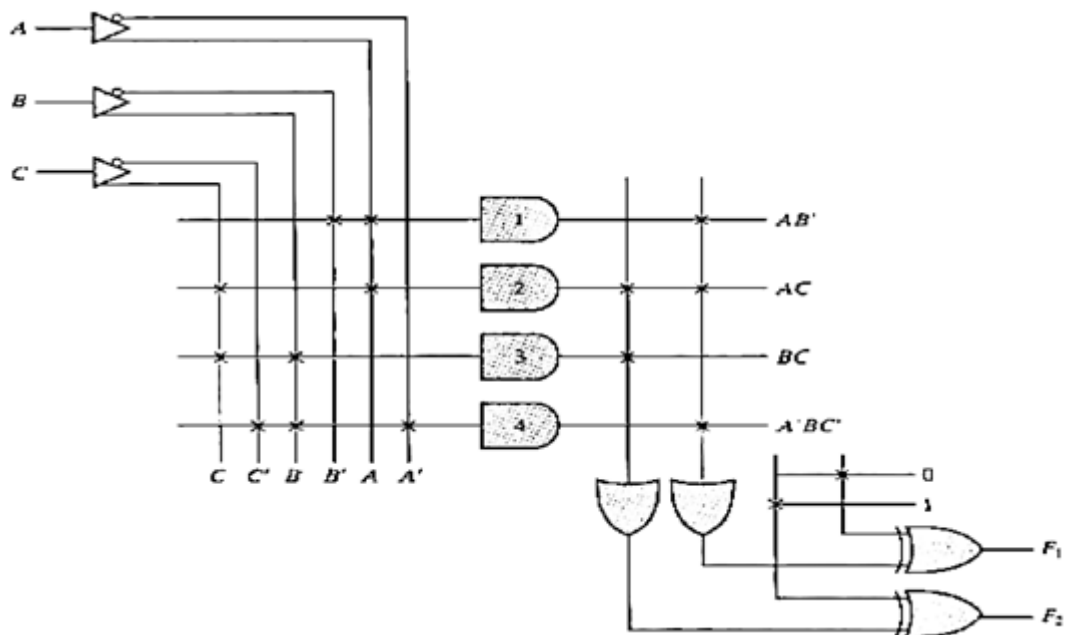
Fig:PLA with threeinputs, fourproduct termsandtwo outputs

- ThefusemapofaPLAcanbespecifiedinatabularform. Thefirstsectionliststheproductterms numerically.
- The secondsectionspecifies the requiredpathsbetweeninputsandANDgates.
- The   third sectionspecifiesthepathsbetweentheANDandORgates.Foreachoutputvariable,we may havea T'(fortrue) orC (for complement) forprogrammingthe XOR gate.
- For eachproductterm,the inputsare markedwith1,0,or- (dash).If avariable inthe productterm   appearsintheforminwhichitistrue,thecorresponding inputvariableismarkedwitha1.
- Ifitappears   complemented,thecorresponding

27

input variable is marked with a 0. If the variable is absent from the product term, it is marked with a dash.

PLA Programming Table

Product Term		Inputs			Outputs (T) (C)	
		A	B	C	F_1	F_2
AB'	1	1	0	—	1	—
AC	2	1	—	1	1	1
BC	3	—	1	1	—	1
A'BC'	4	0	1	0	1	—

14. Implement the following two Boolean functions with a PLA: (Dec-13)

$$F_1(A, B, C) = \Sigma(0, 1, 2, 4)$$

$$F_2(A, B, C) = \Sigma(0, 5, 6, 7)$$

Solution:

Kmap

PLA programming table

Product term		Inputs A B C			Outputs (C) F_1	(T) F_2
AB	1	1	1	–	1	1
AC	2	1	–	1	1	1
BC	3	–	1	1	1	–
A'B'C'	4	0	0	0	–	1
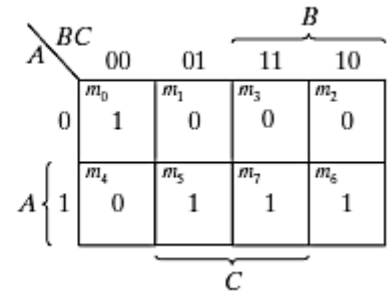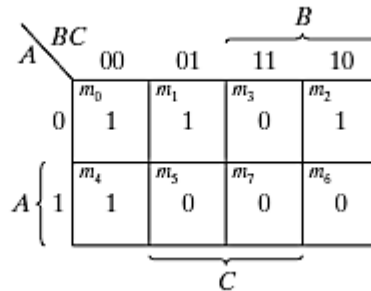
Boolean Expressions:

$$F_1 = (AB + AC + BC)' \qquad F_2 = AB + AC + A'B'C'$$

PLA Diagram:
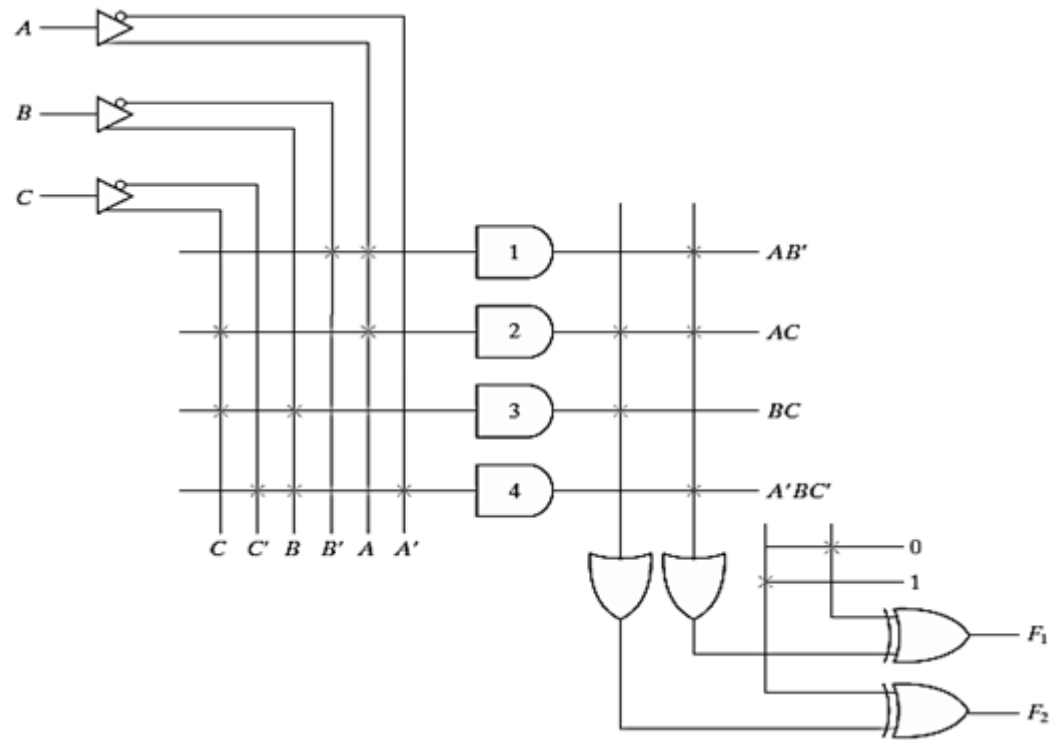
Problems using PROM :

Design a large circuit for the following Boolean expressions using PROM.

$$F_1(x, y, z) = \sum m(1, 2, 4, 7).$$

$$F_2(x, y, z) = \sum m(3, 5, 6, 7).$$

✍ *Solution :*

The Truth Table for the functions F_1 and F_2 are given below.

Inputs			Outputs	
x	y	z	F_1	F_2
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

29

The PROM implementation is given below.
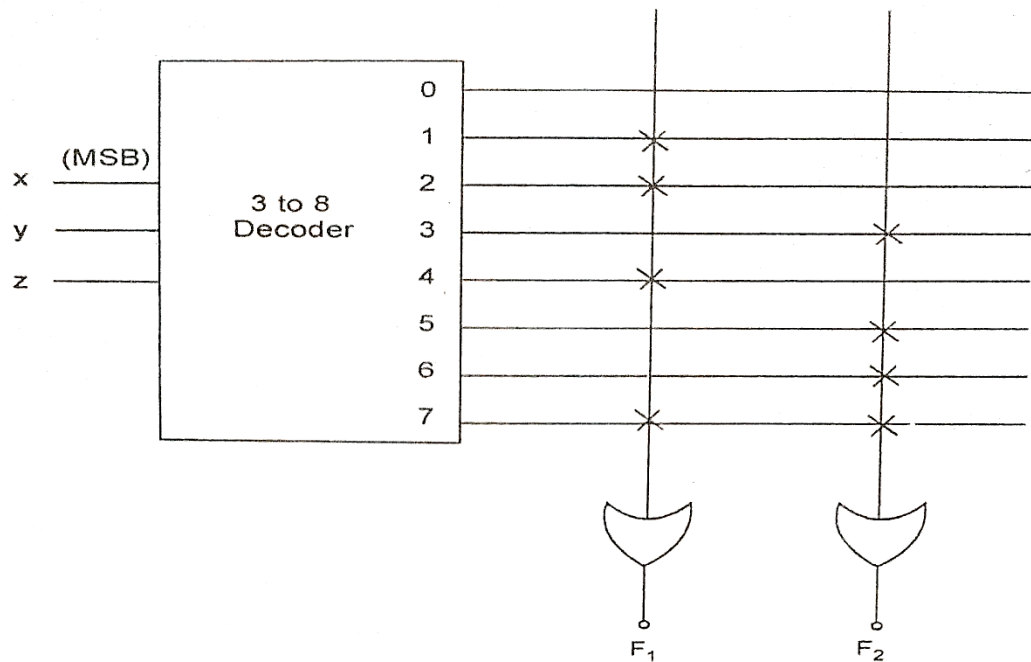
15. **Write short notes on PAL.**

PROGRAMMABLE ARRAY LOGIC (PAL): (Dec-05, 17, May-12)

- ThePAL isaprogrammablelogicdevicewithafixedORarrayandaprogrammableANDarray.
- BecauseonlytheANDgatesareprogrammable,thePALiseasiertoprogramthanbutisn otasflexible as the PLA.
- The PAL is a programmable logic device with a fixed OR array and a programmable AND array.
- Below figure shows the logic configuration of a typical PAL with four inputs and four outputs.
- Each input has a buffer–inverter gate, and each output is generated by a fixed OR gate.
- There are four sections in the unit, each composed of an AND–OR array that is three wide.
- Each AND gate has 10 programmable input connections, shown in the diagram by 10 vertical lines intersecting each horizontal line.
- The horizontal line symbolizes the multiple-input configuration of the AND gate.
- One of the outputs is connected to a buffer–inverter gate and then fed back into two inputs of the AND gates.

Example:
Implement the followingBoolean functions, using PAL.
    (May 2013)

$$w(A, B, C, D) = \Sigma(2, 12, 13)$$
$$x(A, B, C, D) = \Sigma(7, 8, 9, 10, 11, 12, 13, 14, 15)$$
$$y(A, B, C, D) = \Sigma(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$$
$$z(A, B, C, D) = \Sigma(1, 2, 8, 12, 13)$$

Sol:

Simplify the functions using K Map:

W

CD AB	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}B$	0	0	0	1
$\overline{A}\overline{B}$	0	0	0	0
AB	1	1	0	0
$A\overline{B}$	0	0	0	0

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

X

CD AB	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}B$	0	0	0	0
$\overline{A}\overline{B}$	0	0	1	0
AB	1	1	1	1
$A\overline{B}$	1	1	1	1

$$X = A + BCD$$

Y

CD AB	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$	1	0	1	1
$\overline{A}B$	1	1	1	1
AB	0	0	1	0
$A\overline{B}$	1	0	1	1

$$Y = \overline{A}B + CD + \overline{B}\,\overline{D}$$

Z

CD AB	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}B$	0	1	0	1
$\overline{A}\overline{B}$	0	0	0	0
AB	1	1	0	0
$A\overline{B}$	1	0	0	0

$$Z = AB\overline{C} + \overline{A}\,\overline{B}C\overline{D}$$
$$+ AC\overline{D} + \overline{A}\,\overline{B}\overline{C}D$$
$$= W + AC\overline{D} + \overline{A}\,\overline{B}\overline{C}D$$

PAL Programming Table:

31

PAL Programming Table

Product Term	AND Inputs					Outputs
	A	B	C	D	W	
1	1	1	0	–	–	$w = ABC'$
2	0	0	1	0	–	$+ A'B'CD'$
3	–	–	–	–	–	
4	1	–	–	–	–	$x = A$
5	–	1	1	1	–	$+ BCD$
6	–	–	–	–	–	
7	0	1	–	–	–	$y = A'B$
8	–	–	1	1	–	$+ CD$
9	–	0	–	0	–	$+ B'D'$
10	–	–	–	–	1	$z = w$
11	1	–	0	0	–	$+ AC'D'$
12	0	0	0	1	–	$+ A'B'C'D$

PAL Logic Diagram:

32

Problem 1: (Dec-11)

Implement the following function using PLA

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

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

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

Ans:

yz / x	00	01	11	10
0	0	(1)	0	1
1	1	0	0	1

$$A = \bar{x}\,\bar{y}\,z + x\,\bar{z} + y\,\bar{z}$$

yz / x	00	01	11	10
0	1	1	0	0
1	0	0	1	1

$$B = \bar{x}\,\bar{y} + x\,y$$

yz / x	00	01	11	10
0	0	0	0	1
1	0	0	0	1

$$C = y\,\bar{z}$$

PLA Programming Table:

Product	Inputs			Outputs		
Term	x	y	z	A	B	C
$\bar{x}\,\bar{y}\,z$	0	0	1	1	1	0
$x\,\bar{z}$	1	–	0	1	–	–
$y\,\bar{z}$	–	1	0	1	–	1
$\bar{x}\,\bar{y}$	0	0	–	–	1	–
$x\,y$	1	1	–	–	1	–

PLA Logic Diagram:
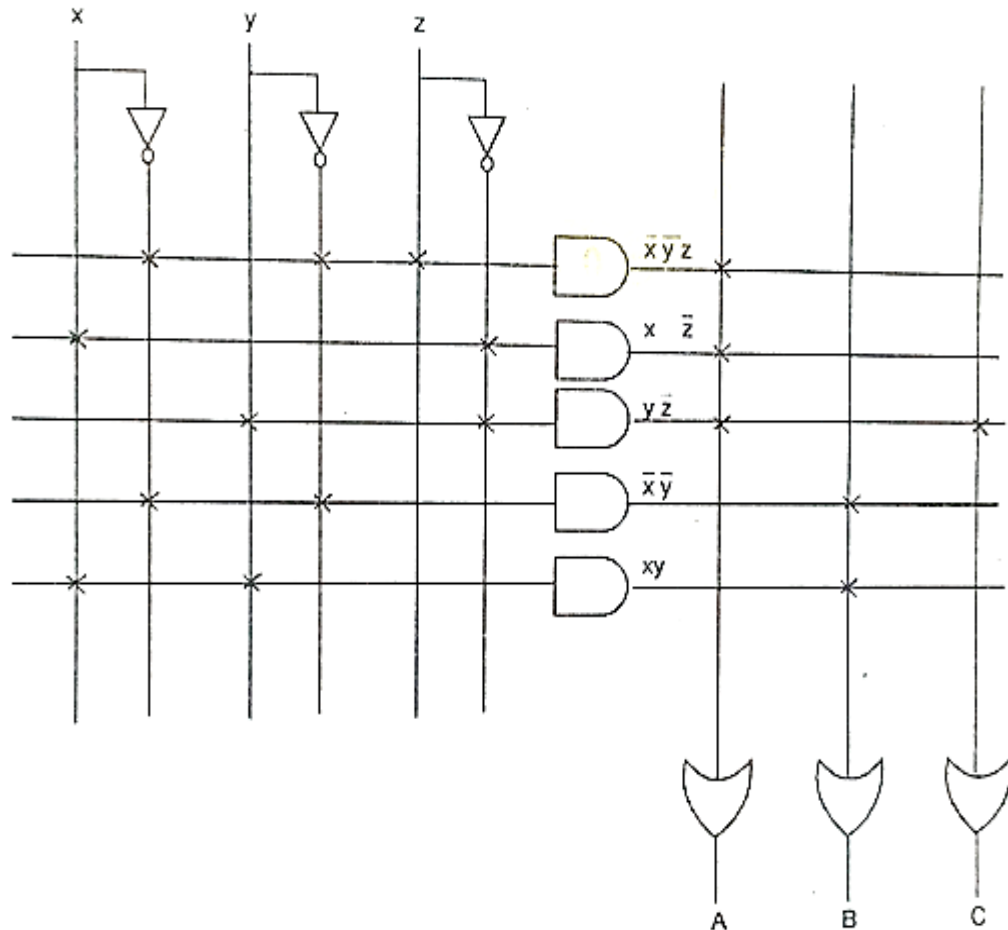
16. Write a short note on PROM.

Programmable Read Only Memory (PROM):   (May-10, 11, 18)

- It consists of n-input lines and m-output lines. Each bit combination of the input variables is called an address. Each bit combination that comes out of the output lines is called word.
- The word available on the output lines at any given time depends on the address value applied to the input lines.

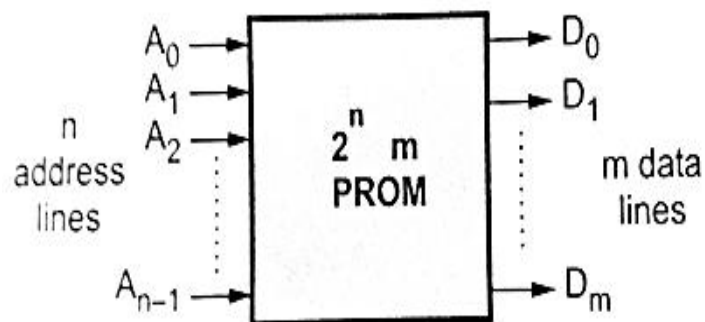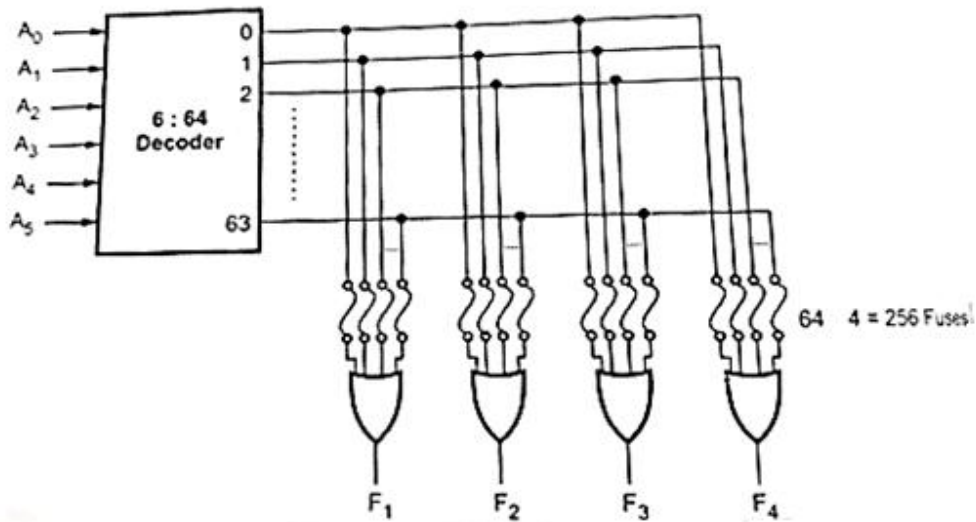**Fig: Block diagram of PROM**
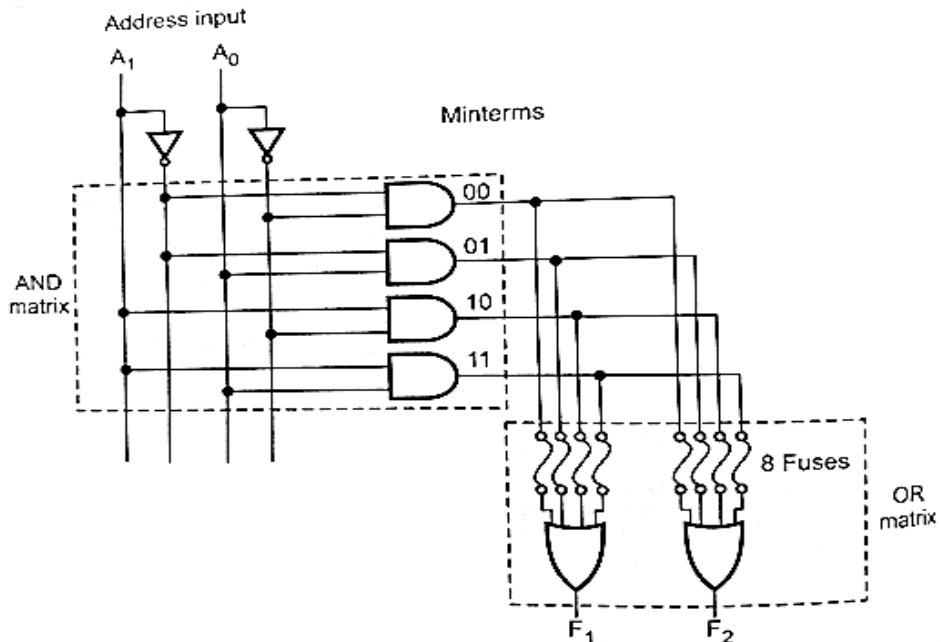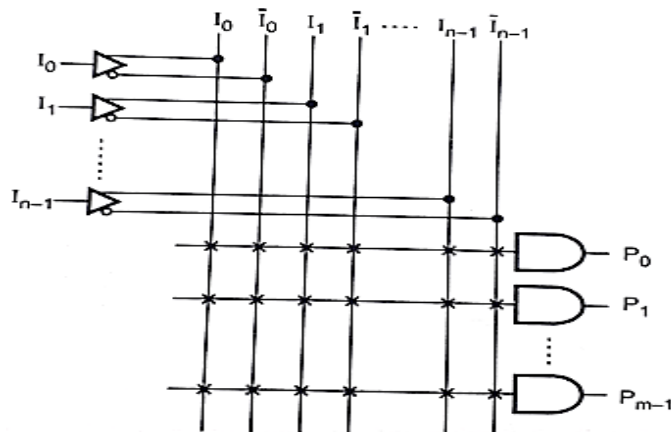
Fig: Logic construction of 64 X 4 PROM

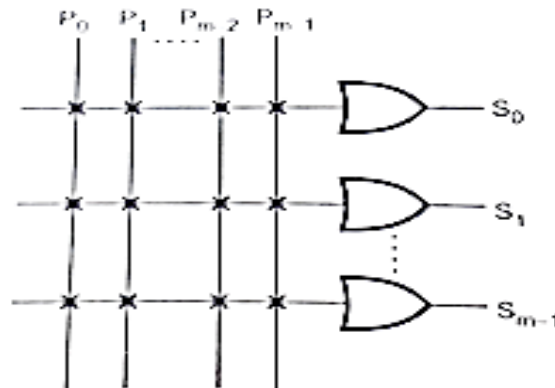**Fig: 4 X 2 PROM with AND-OR gates**

AND Matrix:

- It is used to form product terms. It has m AND gates with 2n-inputs and m-outputs, one for each AND gate.

36

OR Matrix:

- The OR matrix is provided to produce the logical sum of the product term outputs of the AND matrix.

Invert/Non-invert Matrix:

- Invert/Non-invert matrix provides output in the complement or uncomplemented form. The user can program the output in either complement or uncomplement form as per design requirements.
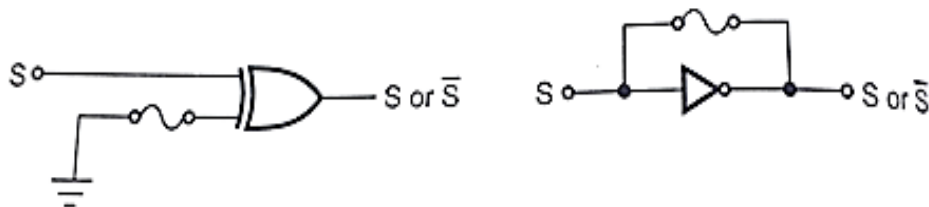
17. Write short notes on FPGA.

Field-Programmable Gate Array (FPGA)

- Afield programmablegatearray(FPGA)isaVLSIcircuitthatcanbeprogrammed attheuser'slocation.
- AtypicalFPGAconsistsofanarray ofmillionsoflogicblocks, surroundedbyprogrammableinputandoutputblocksandconnectedtogethervi aprogrammableinterconnections.
- There isawidevarietyofinternalconfigurationswithin thisgroup ofdevices.
- Theperformanceofeachtypeofdevicedependsonthecircuit containedinitslogicblocksandtheefficiencyofitsprogrammed interconnections.

Fig: Largeprogrammable-logic-devicescaling approaches: (a)CPLD;(b) FPGA.

37

- A typical FPGA logic block consists of look up tables, multiplexers, gates, and flip-flops.
- A look up table is a truth table stored in an SRAM and provides the combinational circuit functions for the logic block.
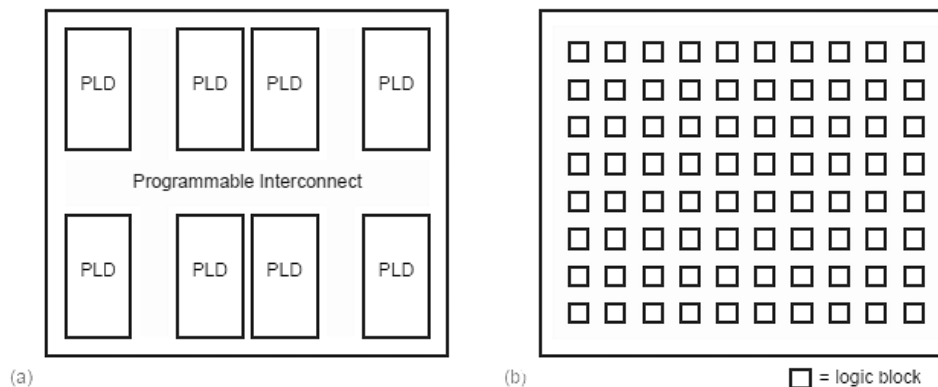- The combinational logic section, along with a number of programmable multiplexers, is used to configure the input equations for the flip-flop and the output of the logic block.
- The advantage of using RAM instead of ROM to store the truth table is that the table can be programmed by writing into memory.
- The disadvantage is that the memory is volatile and presents the need for the look up table's content to be reloaded in the event that power is disrupted.
- The program can be downloaded either from a host computer or from an onboard PROM.
- The program remains in SRAM until the FPGA is reprogrammed or the power is turned off. The device must be reprogrammed every time power is turned on.

Basic Xilinx Architecture:
- The basic architecture of Spartan and earlier device families consists of an array of configurable logic blocks (CLBs), a variety of local and global routing resources, and input–output (I/O) blocks (IOBs), programmable I/O buffers, and an SRAM based configuration memory, as shown in Fig.

38
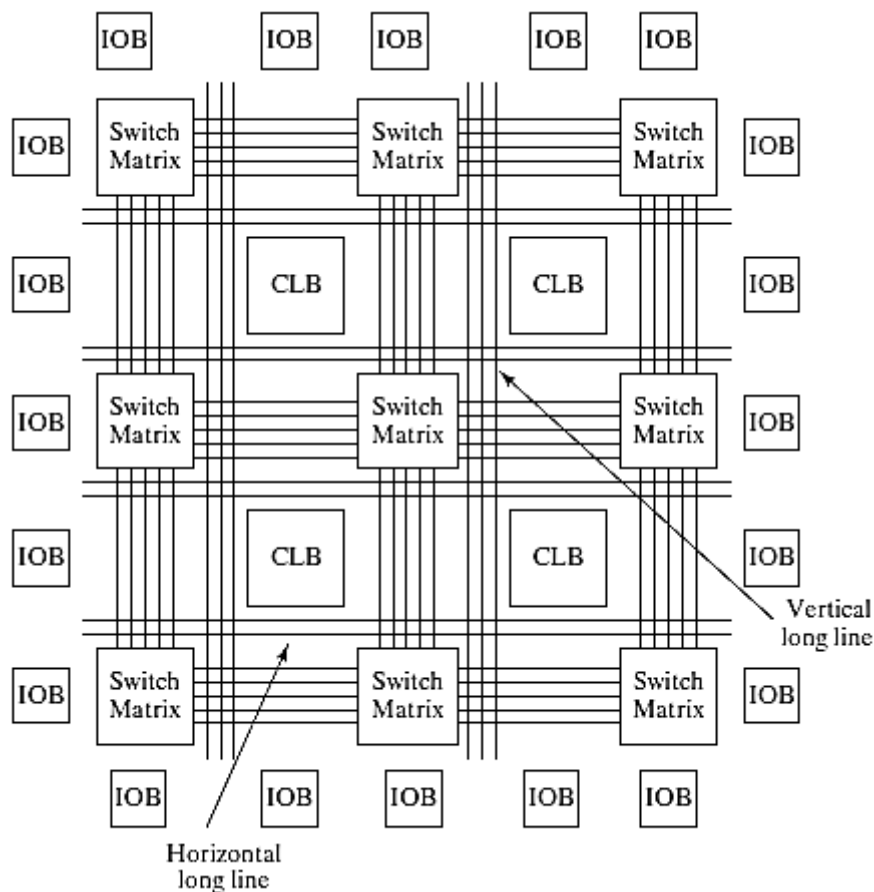
18. Write a short note on CPLD?

Complex Programming Logic Devices (CPLD):
- PLA and PAL are useful for implementing a wide variety of small digital circuits like MUX, encoders, counters etc.
- These devices have a very low number of inputs and outputs at the range of 32 as a maximum value.
- For digital circuits with more number of inputs, (Example: ALU, Microprocessor), a complex programmable logic device (CPLD), can be used.
- A CPLD contains multiple circuit blocks on a single chip with internal wiring resources connected to the circuit blocks. Each circuit block is similar to PLA or PAL in their function.
- All the PAL like blocks are connected using inter connection wires.
- Each block has a subcircuit called as I/O block which is connected to the chips input and output pins.

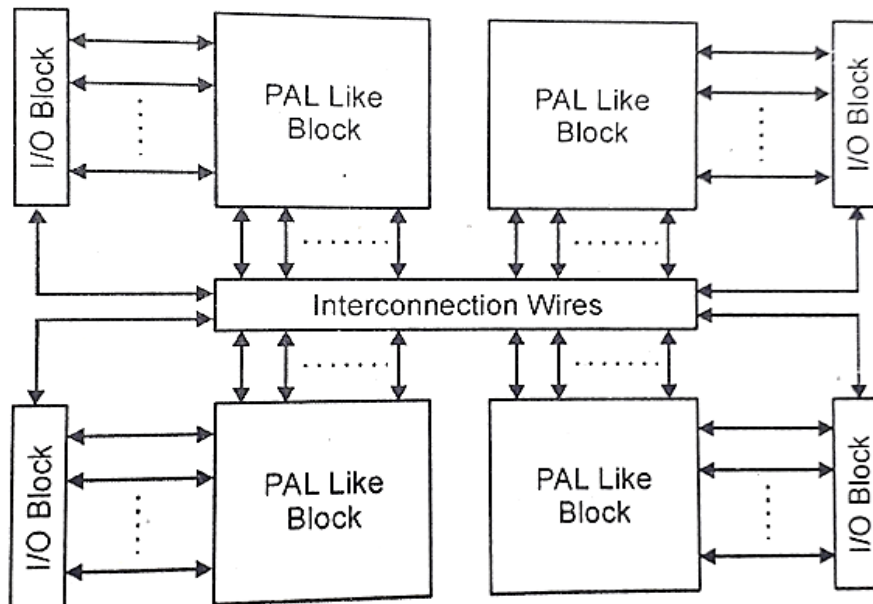**Fig: Structure of a CPLD**

- There are several PAL like block in the CPLD. Each PAL like block consists of some macro cells (about 16), each macro cell consisting of a four input OR gate from programmable AND gates.
- The inputs of programmable AND gates are connected to I/O blocks through interconnection wires.
- The macro cell consists of some more gates, flip-flops and multiplexers.

39
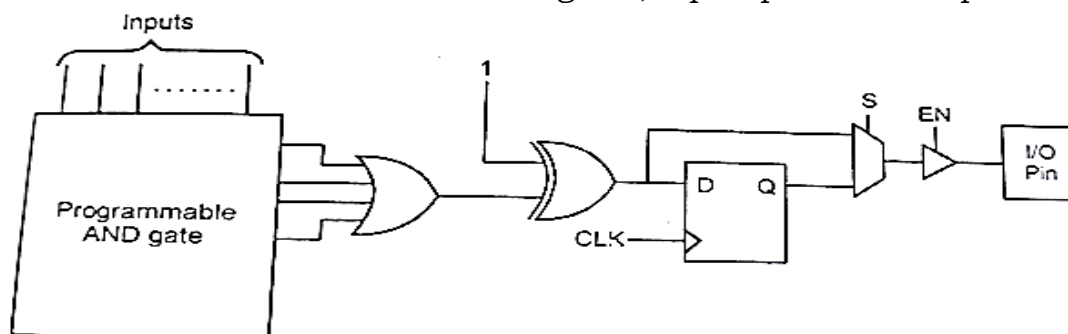
EE8351 DIGITAL LOGIC CIRCUITS

UNIT V – VHDL

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

TWO MARKS

1. **What is a HDL?**

   Computer Aided Design (CAD) tools are used in the design of digital systems. One such tool is a Hardware Description Language (HDL).

2. **What are the main components of a VHDL description?**

   The main components of a VHDL description are:
   - Package (optional)
   - Entity
   - Architecture
   - Configuration (optional)

3. **What is entity?**

   Entity gives the specification of input/output signals to external circuitry. It gives interfacing between device and the other peripherals. An entity usually has one or more ports, which are analogous to the pins on a schematic symbol. All information must flow into and out of the entity through the ports. Each port must contain name, data flow direction and type.

4. **Give the syntax for VHDL entity declaration.(April/May 2017)**

   The syntax of a VHDL entity declaration is as shown below:

   Entity entity_ name is

       Port (signal_ names: mode signal_ type;

                         Signal_ name: mode signal_ type;

                         :

                         :

                         Signal_ names: mode signal_ type);

       end entity_ name;

5. **What is architecture?**

   Architecture specifies behavior, functionality, interconnections or relationship between inputs and outputs. It is the actual description of the design. Architecture consists of two portions: architecture declaration and architecture body.

6. **List the internal details of an entity specified by architecture body.**

   An architecture body specifies the following internal details of an entity:
   - As a set of concurrent assignment statements (to represent dataflow)

> ➢ As a set of interconnected components (to represent structure)
> ➢ As a set of sequential assignment statement (to represent behavior)
> ➢ As any combination of above three.

7. Give the syntax for VHDL architecture declaration. (April/May 2017)

The syntax for architecture is given below:

Architecture architecture _ name of entity_ name is

Declarations

Begin

      Concurrent statement;

      Sequential statement;

  end architecture_ name;

8. What is the use of configuration declaration?

Configuration declarations may be used to associate particular design entities to component instances (unique reference to lower-level components) in a hierarchical design, or to associate a particular architecture to an entity.

9. What is the need of package declaration? (April/May 2015)

There are some declarations which are common across many design units. A package is a convenient mechanism to store and share such declarations. A set of declarations contained in a package declaration may be shared by many design units. It defines items that can be made visible to other design units.

10. How is package represented?

A package is represented by:

> ➢ Package declaration
> ➢ Package body (optional)

11. What are the various modeling techniques in HDL? (May-10, Dec.-12)

There are three modelling techniques in HDL for describing a module:

> ➢ Gate-level modeling/structural modeling
> ➢ Dataflow modeling
> ➢ Behavioral modeling

12. What is behavioral modeling?

The modeling style which directly describes the behavior or the functionality of a circuit is called behavioral modeling. It is very similar in syntax and semantics to that of a high-level programming language (for example: C, Pascal). A behavioral description models the system as to how the outputs behave with the inputs.

13. Write HDL behavioral model of D flip flop.

In the previous code, we have used IF-THEN statement. The similar code can be writing using WAIT-UNTIL statement. This statement has the same effect as IF-THEN statement. In this case, the sensitively list is omitted. The VHDL code for a

positive-edge triggered D flip-flop using a WAIT-UNTIL statement is given below. The WAIT-UNTIL construct implies that the sensitivity list includes only the clock signal.

14. What is data flow modeling?

Data flow describes how the circuit signal flow from the input to the outputs. There are some concurrent statement which allows describing the circuit in terms of operations on signals and flow of signals in the circuit. When such concurrent statement is used in a program, the style is called a dataflow modeling.

15. What is structural modeling?

The modeling style which uses components or gates to model the system is called structural modeling.

16. List the data objects supported by VHDL.

The data objects supported by VHDL are:
- ➢ Signals
- ➢ Variables
- ➢ Constants
- ➢ File

17. Give the comparison between concurrent and sequential statement.

S.NO	CONCURRENT STATEMENT	SEQUENTIAL STATEMENT
1.	Concurrent statement in VHDL can refer only to signals. Therefore, the order of the execution of concurrent statements is not important.	Sequential statement is executed in the order in which they appear within the process. Thus, the order of the execution of sequential statements.
2.	A process block is considered to be a single concurrent statement.	Sequential statements can only appear inside of a process block.
3.	It can be anywhere in the architecture body.	It should be in the process.
4.	Examples for concurrent statements are: process, component signal assignment.	Examples of sequential statement are: if, for, case, sequential signal assignment statement.

18. States the use of generate statement in VHDL.

A generate statement in VHDL is used to create repetitive structure for repetitive sub circuits. This concept is similar to use a FOR loop. When generate statement is used, it is not necessary to write out all of the component instantiations individually.

19. What is subprogram?

A subprogram defines a sequential algorithm that performs particular task. Two types of subprograms are used in VHDL: Procedures and functions. Procedures and functions in VHDL, are directly analogous to functions and procedures in a high-level programming language such as C or Pascal.

20. How procedure differs from function?

A procedure differs from a function in that there is no return value, and the arguments of the procedure have modes (in, out, or inout).

21. What do you mean by subprogram overloading?

It is possible to define two or more different subprograms having the same name but differing in number or type of parameters. The function is said to be overloaded in this case. The simulator or synthesizer automatically selects the correct subprogram by looking at the parameters in the call statement. Overloading is a very convenient mechanism for defining a set of function that perform the same operation on different data types.

22. Write HDL for half adder. (May-10, 12)

Module half_ adder (A, B, Sum, C out);
    Input A;
    Input B;
    Output Sum;
    Output C out;
    reg Sum, C out;
    always @(A, B)
    begin
    #10 Sum = a b;
    #10 Cout = a & b;
    End

23. When can RTL be used to represent digital systems?

When digital systems are composed of registers and combinational function locks, the RTL can be used to represent digital systems.

24. What is test bench?

Before processing a design by synthesis tool, the designer usually wants to verify that the design performs according to the specification. This is almost always done by running a simulation. Simulating a design requires generation of test data and observation of simulation results. This process is done by used of a VHDL module that is referred to as test bench.

25. List the different types of test benches?

The different types of test benches are:
- Stimulus only
- Full test bench

➢ Simulator specific
➢ Hybrid test-bench
➢ Fast test bench

26. What is the meaning of the following RTL statement?

T1: ACC<- ACC and MDR.

The contents of register ACC are bit wised ANDed with the contents of MDR register and the result is stored in the ACC register when control signal T1 is activated.

27. What is Verilog?

Verilog is a general purpose hardware descriptor language. It is similar in syntax to the C programming language. It can be used to model a digital system at many levels of abstraction ranging from the algorithmic level to the switch level.

28. What are the various modeling used in Verilog?

1. Gate-level modeling
2. Data-flow modeling
3. Switch-level modeling
4. Behavioral modeling

29. What is the structural gate-level modeling?

Structural modeling describes a digital logic networks in terms of the components that make up the system. Gate-level modeling is based on using primitive logic gates and specifying how they are wired together.

30. What is Switch-level modeling? (May-18)

Verilog allows switch-level modeling that is based on the behavior of MOSFETs. Digital circuits at the MOS-transistor level are described using the MOSFET switches.

31. What are identifiers?

Identifiers are names of modules, variables and other objects that we can reference in the design. Identifiers consists of upper and lower case letters, digits 0 through 9, the underscore character(_) and the dollar sign($). It must be a single group of characters.

Examples: A014, a, b, in_o, s_out

32. What are the value sets in Verilog?

Verilog supports four levels for the values needed to describe hardware referred to as value sets.

Value levels	Condition in hardware circuits
0	Logic zero, false condition
1	Logic one, true condition
X	Unknown logic value
Z	High impedance, floating state

33. What are the types of gate arrays in ASIC?

        1) Channeled gate arrays

        2) Channel less gate arrays

        3) Structured gate arrays

34. Give the classifications of timing control

Methods of timing control:

1. Delay-based timing control

2. Event-based timing control

3. Level-sensitive timing control

Types of delay-based timing control:

1. Regular delay control

2. Intra-assignment delay control

3. Zero delay control

Types of event-based timing control:

1. Regular event control

2. Named event control

3. Event OR control

4. Level-sensitive timing control.

35. What are the advantages of Hardware Description Languages?

➢ Boolean expressions, logic diagrams and digital circuits can be represented.

➢ Simulation, modeling, testing, design and documentation of digital circuit can be easily done by using Hardware Description Languages.

36. Give the different arithmetic operators?

Operator symbol	Operation performed	Number of operands
*	Multiply	Two
/	Divide	Two
+	Add	Two
-	Subtract	Two
%	Modulus	Two
**	Power (exponent)	Two

37. Give the different bitwise operators.

Operator symbol	Operation performed	Number of operands
~	Bitwise negation	One
&	Bitwise and	Two
|	Bitwise or	Two
^	Bitwise xor	Two
^~ or ~^	Bitwise xnor	Two
~&	Bitwise nand	Two
~|	Bitwise nor	Two

38. What is the meaning of the following RTL statement? (Nov/Dec 2011)

T1 : ACC ←---- ACC and MDR

> ➢ T1 represents control function,
> ➢ Colon(:) separates the control function from accumulator (ACC),
> ➢ Reverse arrow (←--) indicates transfer of information.
> ➢ MDR (Memory Data Register) --→ Holds contents of memory word.
> ➢ The information present in Accumulator and Data Register are added and then stored in the accumulator.

39. Write HDL for Half adder. (May/June 2012)

module half_adder (a, b, sum, cout);
input a;
input b;
output sum;
output cout;
reg sum,cout;
always @ (a,b)
begin
#10 sum = a ^ b;
10 cout = a & b ;
end
endmodule

40. Write HDL behavioural model of D flip-flop. (May/June 2013) (Nov/Dec 2015,2016) (May/June 2015)

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
ENTITY DFF IS
        PORT (D, Clock   : IN STD_LOGIC;
                 Q   : OUT STD_LOGIC);
END DFF;
ARCHITECTURE Behaviour OF DFF IS
BEGIN
        IF Clock' EVENT AND Clock = '1' THEN
                Q< = D;
        END IF;
    END PROCESS;
  END Behaviour;

41. What are gate primitives?

Verilog supports basic logic gates as predefined primitives. Primitive logic function keyword provides the basics for structural modeling at gate level. These primitives are instantiated like modules except that they are predefined in verilog

and do not need a module definition. The important operations are and, nand, or, xor, xnor, and buf (non-inverting drive buffer).

42. **Give the two blocks in behavioral modeling.**
    1. An initial block executes once in the simulation and is used to set up initial conditions
       and step-by-step data flow.
    2. An always block executes in a loop and repeats during the simulation.

43. **What are the types of conditional statements?**
    1. No else statement
    Syntax: if ([expression]) true – statement;
    2. One else statement
    Syntax: if ([expression]) true – statement;
    else false-statement;
    3. Nested if-else-if
    Syntax : if ([expression1]) true statement 1;
        else if ([expression2]) true-statement 2;
        else if ([expression3]) true-statement 3;
         else default-statement;
    The [expression] is evaluated. If it is true (1 or a non-zero value) true-statement is executed. If it is false (zero) or ambiguous (x), the false-statement is executed.

44. **Name the types of ports in Verilog**

Types of port	Keyword
Input port	Input
Output port	Output
Bidirectional port	inout

45. **What is the need for VHDL? (May/June 2013)**
    HDL describes the hardware of digital systems. This description is in textual form. The Boolean expressions, logic diagrams and digital circuits can be represented using HDL. The most prominent modern HDLs in industry are Verilog and VHDL. It is one of the two major Hardware Description Languages (HDLs) used by hardware designers in industry and academia.

46. **What are the operators present in VHDL? (Nov/Dec 2011) (Nov/Dec 2015)**
    VHDL includes the following kinds of operators:
    ➢ Logical
    ➢ Relational
    ➢ Arithmetic
    ➢ Shift and Rotate.

47. **What are the types of procedural assignments?**
    ➢ Blocking assignment
    ➢ Non-blocking assignment

EE8351 DIGITAL LOGIC CIRCUITS

PART-B -16marks
RTL Design

**Explain in detail about principle of operation of RTL Design.**

(Dec-15, May -18)

Register Transfer Level, or RTL design lies between a purely behavioral description of the desired circuit and a purely structural one. An RTL description de- scribes a circuit's registers and the sequence of transfers between these registers but does not describe the hardware used to carry out these operations.

As a simple example, consider a device that needs to add four numbers. In VHDL, given signals of the correct type, we can simply write:

   $s \Leftarrow a+b+c+d;$

This particular description is simple enough that it can be synthesized. However, the resulting circuit will be a fairly large combinational circuit comprising three adder circuits as follows:
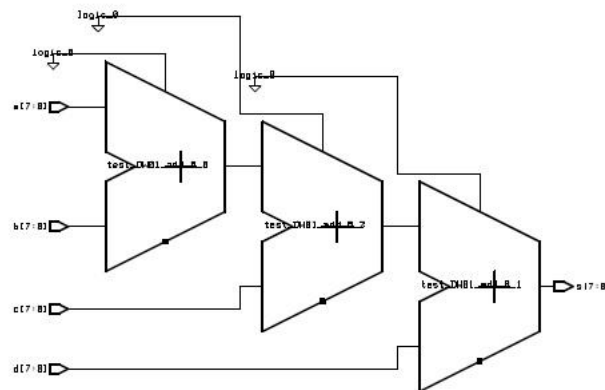
A behavioral description, not being concerned with implementation details would be complete at this point.

However, if we were concerned about the cost of the implementation we might decide to break down the computation into a sequence of steps, each one involving only a single addition:

   $s=0$
   $s=s+a$
   $s=s+b$
   $s=s+c$
   $s=s+d$

Where, each operation is executed sequentially. The logic required is now one adder, a register to hold the value of s in-between operations, a multiplexer to select the input to be added on, and a circuit to clear s at the start of the computation.

Although this approach only needs one adder, the process requires more steps and

will take longer. Circuits that divide up a computation into a sequence of arithmetic and logic operations are quite common and this type of design is called Register Transfer Level (RTL) or "dataflow" design.

An RTL design is composed of (1) registers and combinational function blocks (e.g. adders and multiplexers) called the data path and (2) a finite state machine, called the controller that controls the transfer of data through the function blocks and between the registers.

In VHDL RTL design the gate-level design and optimization of the data path (registers, multiplexers, and combinational functions) is done by the synthesizer. However, the designer must design the state machine and decide which register transfers are per- formed in which state.

The RTL designer can trade off data path complexity (e.g. using more adders and thus using more chip area) against speed (e.g. having more adders means fewer steps are required to obtain the result).

RTL design is well suited for the design of micro- processors and special-purpose processors such as disk drive controllers, video display cards, network adapter cards, etc. It gives the designer great flexibility in choosing between processing speed and circuit complexity.

The diagram below shows a generic component in the data path. Each RTL design will be composed of from controller clock one of the following building blocks for each register. The structure allows the contents of each register to be updated at the end of each clock period with a value selected by the controller.

The widths of the registers, the types of combinational functions and their inputs will be determined by the application. A typical design will include many of these components.
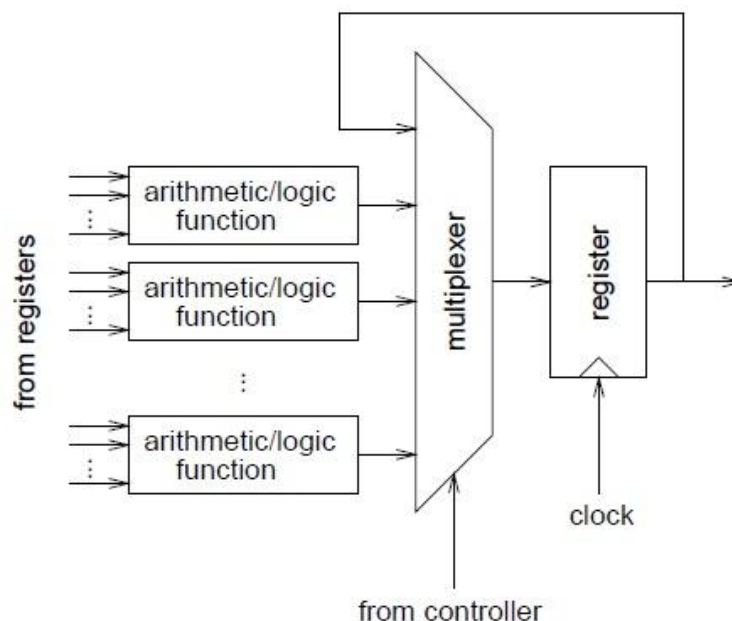
EE8351 DIGITAL LOGIC CIRCUITS

RTL Design Example

To show how an RTL design is described in VHDL and to clarify the concepts involved, we will design a four-input adder. This design will also demonstrate how to create packages of components that can be re-used.

The data path shown below can load the register at the start of each clock cycle with zero, the current value of the register, or the sum of the register and one of the four inputs.

It includes one 8-bit register, an 8-bit adder and a multiplexer that selects one of the four inputs as the value to be added to the current value of the register.
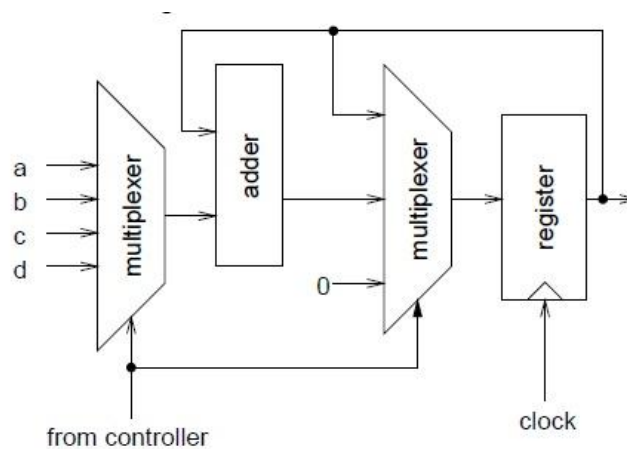
Exercise: Other data paths could compute the same result. Draw the block diagram of a data path capable of computing the sum of the four numbers in three clock cycles.

The first design unit is a package that defines a new type, num, for eight-bit unsigned numbers and an enumerated type, states, with six possible values. Nums are defined as a subtype of the unsigned type.

```
-- subtype used in design

library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;

package averager_types is
subtype num is unsigned (7 downto 0) ;
type states is (clr, add_a, add_b, add_c, add_d, hold) ;
end averager_types ;
```

The first entity defines the data path. In this case the four numbers to be added are available as inputs to the entity and there is one output for the current sum.
Data path

```
library ieee ;
```

```
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use work.averager_types.all ;

entity datapath is port (
a, b, c, d : in num ; sum : out num ;
sel : in std_logic_vector (1 downto 0) ;
load, clear, clk : in std_logic) ;
end datapath ;

architecture rtl of datapath is
signal mux_out, sum_reg, next_sum_reg : num ;
constant sum_zero : num :=
conv_unsigned(0,next_sum_reg'length) ;
begin

-- mux to select input to add with sel select mux_out <=
a when "00", b when "01", c when "10",
d when others ;

-- mux to select register input next_sum_reg <=
sum_reg + mux_out when load = '1' else
sum_zero when clear = '1' else
sum_reg ;

-- register sum
process(clk) begin
if clk'event and clk = '1' then
sum_reg <= next_sum_reg ;
end if ;
end process ;

-- entity output is register output
sum <= sum_reg ;

end rtl ;
```

Program for AND gate(*)

```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity\and\is
Port(
a:in STD_LOGIC;
b:inSTD_LOGIC;
```

```
c:outSTD_LOGIC;
);
End\and\;
Architecture\and\of\and\is
Being
  C<=a and b;
End\and\;
```

Program for OR gate

```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity\or\is
Port(
a:in STD_LOGIC;
b:inSTD_LOGIC;
c:outSTD_LOGIC;
);
End\or\;
Architecture\or\of\or\is
Being
  C<=a or b;
End\or\;
```

Program for NOT

```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity\NOT\is
Port(
a:in STD_LOGIC;
b:OUTSTD_LOGIC;
);
End\NOT\;
Architecture\NOT\of\NOT\is
Being
  b<=not a;
End\not\;
```

Program for NAND gate

```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity\nand\is
Port(
a:in STD_LOGIC;
b:inSTD_LOGIC;
```

```
c:outSTD_LOGIC;
);
End\nand\;
Architecture\nand\of\nand\is
Being
 C<=a nand b;
End\nand\;
```

Program for NOR gate

```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity\nor\is
Port(
a:in STD_LOGIC;
b:inSTD_LOGIC;
c:outSTD_LOGIC;
);
End\nor\;
Architecture\nor\of\nor\is
Being
 C<=a nor b;
End\nor\;
```

Program for XOR gate

```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity\xor\is
Port(
a:in STD_LOGIC;
b:inSTD_LOGIC;
c:outSTD_LOGIC;
);
End\xor\;
Architecture\xor\of\xor\is
Being
 C<=a xor b;
End\xor\;
```

Program for XNOR gate

```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity\xnor\is
Port(
a:in STD_LOGIC;
```

```
b:inSTD_LOGIC;
c:out STD_LOGIC;
);
End\xnor\;
Architecture\xnor\of\xnor\is
Being
 C<=a xnor b;
End\xnor\;
```

Program for HALF ADDER (**)

```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity\half_adder\is
Port(
a:in STD_LOGIC;
b :in STD_LOGIC;
sum: out STD_LOGIC;
carry: out STD_LOGIC;
);
End\HALF ADDER\;
Architecture\halfadder\of\half adder\is
Being
 Sum<=a xor b;
Carry<=a and b;
End\half adder\;
```

Program for FULL ADDER   (May-15, Dec-15, 16)

```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity fulladder is
Port(
a:in STD_LOGIC;
b :in STD_LOGIC;
sum: out STD_LOGIC;
carry: out STD_LOGIC;
);
End full ADDER ;
Architecture fulladder of fulladder is
Being
 Sum<=a xor b xor c;
Carry<=(a and b)or(a and c)or(b and c);
End fulladder;
```

Program for HALF SUBTRACTER

```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity half subtractors is
Port(
a:in STD_LOGIC;
b :in STD_LOGIC;
borrow: out STD_LOGIC;
difference: out STD_LOGIC;
);
End half subtractors ;
Architecture half subtractors of half subtractors is
Being
 difference<=a xor b ;
borrow<=(not a) and b;
End half subtractors;
```

Program for full SUBTRACTORS

```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity full subtractors is
Port(
a:in STD_LOGIC;
b :in STD_LOGIC;
difference: out STD_LOGIC;
borrow: out STD_LOGIC;
);
End full subtractors ;
Architecture full subtractors of full subtractors is
Being
 difference<=a xor b xor c;
borrow<=(((not a)and b)or((not a)and c)or(b and c));
End fullsubtractors;
```

Program for MULTIPLEXER (***)   (May-17, Dec-16)

```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity multi is

Port(
D:in STD_LOGIC_vector(o to3);
S:in STD_LOGICvector(0 to1);
z:out STD_LOGIC;
);
End multi;
```

```
Architecture multi of multi is
Being
Process(d,s)
Begin
Case s is
        When ''00''=>
         Z<=d(0);
         When''01''=>
        Z<=d(1);
         When''10''=>
         Z<=d(2);
         When''11''=>
        Z<=d(3);
         When other=>
          Z<='0';
End case;
End process;
End multi;
```

Program for DEMULTIPLEXER (***)   (Dec-2015)

```
Library ieee;
use ieee.std_logic_1164.all;
entity Demux_4_to_1 is
port (
E:in std_logic;
S0,S1 : in std_logic;
D0,D1,D2,D3 : out std_logic);
End Demux_4_to_1;
--
Architecture Func of Demux_4_to_1 is component and Gate is import AND gate entity
Port(A,B,C : in std_logic;
      F : out std_logic);
End component;

Component not Gate is          --import NOT Gate entity
 port( inPort : in std_logic;
      outPort : out std_logic);
end component;

signal invOut0, invOut1: std_logic;

begin
  --Just like the real circuit, there are
  --four components : G1 to G4
```

GI1 : notGate port map(S0, invOut0);
GI2 : notGate port map(S1, invOut1);

GA1 : andGate port map(E, invOut1, invOut0, D0); -- D0
GA2 : andGate port map(E, S0, invOut1, D1);   -- D1
GA3 : andGate port map(E, invOut0, S1, D2);   -- D2
GA4 : andGate port map(E, S0, S1, D3);   -- D3

End Func;
---------------END

Program for T flip Flop

```
Library IEEE;
Use IEEE.std_logic_1164.all;
Entity tflipflop is
Port(
Qn: in STD_LOGIC;
Clk:IN std_logic;
T:in STD_LOGIC;
Qn1 : out STD_LOGIC
);
End tflipflop;
Architecture flipflop1 of tflipflop is
Begin
Process
Begin
Wait until clk='1';
Qn1<=qn xor t;
End process;
End flipflop1;
```


VHDL description of a D Flip Flop

```
library ieee;
use ieee.std_logic_1164.all;
entity D_FF is
Port(D,CP:in std.logic;
     Q,Qbar:buffer std_logic);
end D_FF;
architecture FF of D_FF is
--here Q and Qbar singal are declared as buffer ; however these signals are  mapped with in and out
signals. Some simulators may not allow such mapping. In this case, change ll in and out buffer.
Component nand2
port ( i1,i2:in std_logic;
o1:out std_logic);
```

```
end component;
for all:NAND2 use entity work.two_input(nand2_7);
signal S1,R,R1:std_logic;
begin
NA1:NAND2 port map (D, CP, S1);
NA2:NAND2 port map (R, CP, R1);
NA3:NAND2 port map (D, D, R); --NAND gate used as inverter
NA4:NAND2 port map (s1, Qbar,Q);
NA5:NAND2 port map(Q,R1,Qbar);
```

VHDL description of a JK Flip Flop

```
library ieee;
use ieee.std_logic_1164.all;
entity JK_FF is
Port( J,K,CP:in std.logic;
      Q,Qbar:buffer std_logic);
end JK_FF;
architecture FF of JK_FF is
--here Q and Qbar singal are declared as buffer ; however these signals are  mapped with in and out
signals. Some simulators may not allow such mapping. In this case, change ll in and out buffer.
Component NOR2
port ( i1,i2:in std_logic;
o1:out std_logic);
end component;
component and3
port ( i1,i2,i3:in std_logic;
o1:out std_logic);
end component;
for all:NOR2 use entity work.two_input(nor2_7);
for all:NOR3 use entity work.three_input(nor3_7);
signal S,R
begin
NA1:NOR2 port map (S,Q,Qbar);
NA2:NOR2 port map (R,Qbar,Q);
A1:and3 port map (Q,K,CP,R);
A2:and3 port map (Qbar,J,CP,S);
end FF;
```

VHDL description of a SR Flip Flop

```
library ieee;
use ieee.std_logic_1164.all;
entity SR_FF is
Port( S,R,CP:in std.logic;
      Q,Qbar:buffer std_logic);
```

```
end SR_FF;
architecture FF of SR_FF is
--here Q and Qbar singal are declared as buffer ; however these signals are  mapped with in and out
signals. Some simulators may not allow such mapping. In this case, change ll in and out buffer.
Component nand2
port ( i1,i2:in std_logic;
o1:out std_logic);
end component;
for all:nand2 use entity work.two_input(nand2_7);
signal S1,R1:std_logic;
begin
NA1:nand2 port map (S1,Qbar,Q);
NA2:nand2 port map (Q,R1,Qbar);
NA3:nand2 port map (S,CP,S11);
NA4:nand2 port map (R,CP,R1);
end latch;
```

VHDL code for a 4-bit down Counter (*)
```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
ENTITY downctr IS
    GENERIC (MOD: INTEGER: =8;
    PORT (clock,load,EN :IN STD_LOGIC;
          Q: OUT INTEGER RANGE 0 TO MOD-1);
END downctr;
ARCHITECTURE behavior OF downctr IS
    SIGNAL count: INTEGER RANGE 0 TO MOD-1;
BEGIN
  PROCESS
   BEGIN
        WAIT UNTIL (clock'EVENT AND clock='1');
        IF EN ='1' THEN
         IF LOAD ='1' THEN
         Count<=MOD-1;
ELSE
      Count<=count-1
    END IF;
    END IF;
END PROCESS;
Q<=count;
END behavior;
```

VHDL code for a Four-Bit Up Counter
```
LIBRARY IEEE;
```

EE8351 DIGITAL LOGIC CIRCUITS

```vhdl
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_unsigned.all;
ENTITY upctr IS
    PORT ( clock,resetn,EN : IN STD_LOGIC;
          Q :OUT STD_LOGIC_VECTOR (3 DOWNTO));
END upctr;
ARCHITECTURE behavior OF upctr IS
  SIGNAL count : STD_LOGIC_VECTOR(3 DOWNTO);
BEGIN
   PROCESS (clock, Resetn)
   BEGIN
        IF Resetn='0' THEN
          Count<=''0000'';
ELSIF(clock'EVENT AND clock ='1')THEN
    IF EN ='1' THEN
     Count<=count+1;
ELSE
     Count<=count;
END IF;
END IF;
      END PROCESS;
Q<=count;
END Behavior;
```

VHDL code for synchronous mod-6 counter(******)   (May-15, Dec-16)

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_unsigned.ALL;
ENTITY counter3 IS
PORT(
     CLK : IN STD_LOGIC;
    Resetn : IN STD_LOGIC;
    Setn : IN STD_LOGIC;
    Q : IN0UT STD_LOGIC_VECTOR (2 DOWNTO 0)
      );
END counter3;
ARCHITECTURE Synch_Cntr of Counter3 IS
BEGIN
PROCESS(CLK,Resetn,Sten)
VARIABLE Qtemp: STD_LOGIC_VECTOR (2 DOWNTO 0);
BEGIN
IF Resetn ='0' THEN
Qtemp:=''000'';
ELSIF Setn='0' THEN
```

```
Qtemp:="111";
ELSIF CLK='1' AND CLK'EVENT THEN
IF Qtemp<5 THEN
Qtemp:Qtemp +1;
ELSE
Qtemp:=''000'';
END IF;
END IF;
Q<=Qtemp;
END PROCESS;
END Synch _Cntr;
```

VHDL description of n-bit magnitude comparator using generate statement

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity comp_gen is
Generic (N : integer :=3);
Port(A,B :in std_logic_vector(N downto 0);
      AgtB,AltB,AeqB : buffer std_lodic);
end comp_gen;
Architecture compare of comp_gen is
Component full_adder
Port(I1,I2,I3:in std_logic;O1,O2:out std_logic);
end component;
Component inv
Port(I1:in std_logic;O1:out std_logic);
end component;
component nor2
port(I1,I2:in std_logic;O1:out std_logic);
end component;
component and2
port(I1,I2:in std_logic;O1:out std_logic);
end component;
signal Sum,Bbar:std_logic_vector(N downto 0);
signsal C,eq:std_logic_vector(N+1 downto);
for all:full_adder use entity work.bind32(full_add);
for all:inv use entity work.bind1(inv_0);
for all:nor2 use entity work.bind(nor2_7);
for all:and2 use entity work.bind(and2_7);
begin
   C(0)<='0';
   Eq(0)<='1';
G1:for I in 0 to N generate
   V1:inv port map (B(i),Bbar(i));
```

22

```
    FA:full_adder port map (A(i),Bbar(i),C(i),Sum(i),C(I+1));
    A1:and2 port map (eq(i),Sum(i),eq(i+1));
end generate G1;
    AgtB<=C(N+1);
    AeqB<=eq(N+1);
    n1:nor2 port map (AeqB,AgtB,AltB);
end compare;
```

Construct a VHDL module listing for a 16:1 MUX that is based on the assignment statement. Use a 4-bit select word S3,S2,S1,S0 to map the selected input Pi(i = 0,15) to the output.(Multiplexer)(******)

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
entity Mux8_1 is
        Port(SEL:in STD_LOGIC_VECTOR(3 downto 0):
        P0,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14,P15:in STD_LOGIC;
        MUX_OUT:out STD_LOGIC);
end Mux8_1;
architecture BEHAVIORAL of Mux16_1 is
begin
        process(SEL,P0,P1,P2,P3,P4,P5,P6,
                P7,P8,P9,P10,P11,P12,P13,P14,P15)
        begin
case SEL is
        when"0000"=>MUX_OUT<=P0;
        when"0001"=>MUX_OUT<=P1;
        when"0010"=>MUX_OUT<=P2;
        when"0011"=>MUX_OUT<=P3;
        when"0100"=>MUX_OUT<=P4;
        when"0101"=>MUX_OUT<=P5;
        when"0110"=>MUX_OUT<=P6;
        when"0111"=>MUX_OUT<=P7;
        when"1000"=>MUX_OUT<=P8;
        when"1001"=>MUX_OUT<=P9;
        when"1010"=>MUX_OUT<=P10;
        when"1011"=>MUX_OUT<=P11;
        when"1100"=>MUX_OUT<=P12;
        when"1101"=>MUX_OUT<=P13;
        when"1110"=>MUX_OUT<=P14;
        when"1111"=>MUX_OUT<=P15;
        when others =>null;
end case;
end process;
end BEHAVIORAL
```