UNIT I

BOOLEAN ALGEBRA AND LOGIC GATES

Number Systems - Arithmetic Operations - Binary Codes- Boolean Algebra and Logic Gates - Theorems and Properties of Boolean Algebra - Boolean Functions - Canonical and Standard Forms - Simplification of Boolean Functions using Karnaugh Map - Logic Gates – NAND and NOR Implementations.

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

Code Conversion:

- 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}

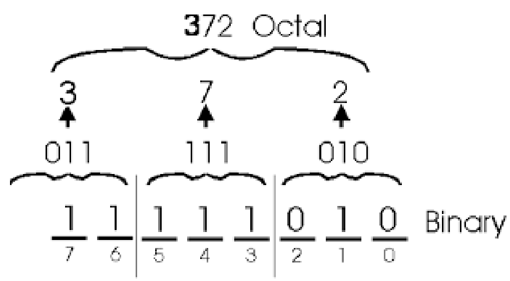
Decimal to binary Conversion:

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:

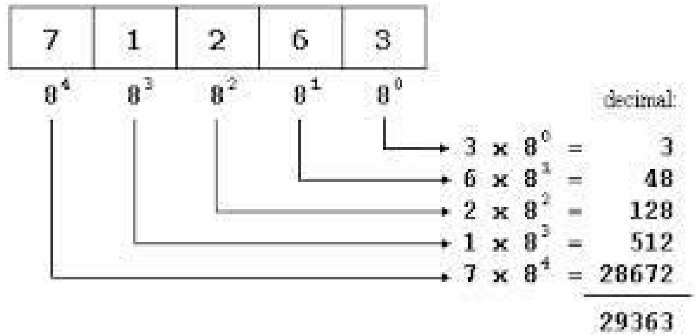


Decimal to octal:

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:



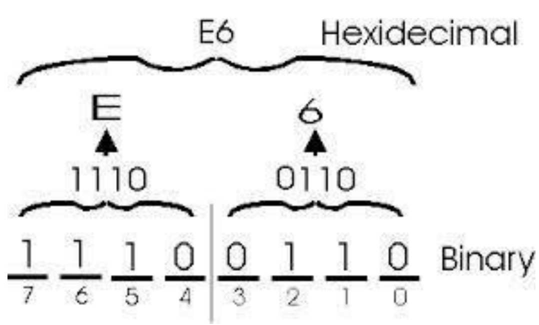
Decimal to Hexadecimal:

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 $(r^n - 1) - N$.

The9's complementof546700 is 999999−546700= 453299.

The9'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:
The10'scomplementof 012398 is 987602
The10'scomplementof246700 is 753300

Model 1: ***(Dec 2009)***

Using10'scomplement, subtract72532–3250.

$M =$ 72532

10's complement o fN = +<u>96750</u>

Sum = 169282

Discard endcarry10^5 = –<u>100000</u>

Answer = 69282

Model 2:

Using10'scomplement, subtract3250–72532.

$M =$ 03250

10's complementof$N$ = +<u>27468</u>

Sum = 30718

Model 3:

Given the two binary numbers X= 1010100and$Y$= 1000011, performthesubtraction (a)$X–Y$ and (b) $Y–X$byusing2'scomplements. [NOV – 2019]

(a) X= 1010100

2's complementof$Y$= + <u>0111101</u>

Sum = 10010001

Discard endcarry2^7 = –<u>10000000</u>

Answer:$X–Y$= 0010001

(b) Y= 1000011

2's complement of X = <u>0101100</u>

Sum = 1101111

There is no end carry.Therefore, the answer is$Y– X$=–(2's complementof1101111) = –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) $X - Y = 1010100 - 1000011$

$X = 1010100$

1's complement of $Y = +\underline{0111100}$

Sum = 10010000

End around carry = + $\underline{1}$

Answer: $X - Y = 0010001$

(b) $Y - X = 1000011 - 1010100$

$Y = 1000011$

1's complement of $X = +\underline{0101011}$

Sum = 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 1fromthe nextmoresignificantbit
- 1 -0 =1
- 1 -1 =0

Example:

Sub: 00100101-00010001= 00010100

	0	**0**	**1**	**0**	**0**	**1**	**0**	**1**
-	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1**
	0	**0**	**0**	**1**	**0**	**1**	**0**	**0**

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	Partial Products
			0	0	0	0		
		0	1	1	1			
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	Partial Products
		0	0	0	0		
	0	0	0	0			
1	0	1	1				
1	1 .	0	0	0	1	1	Final Product

Binary Division:

Binarydivisionisthe repeatedprocess ofsubtraction,justasindecimaldivision.

Example: Divide the following

(*a*) 11001 ÷ 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 ÷ 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 codedepends on the requirement.
- Binarycodesarecodeswhicharerepresentedinbinarysystemwithmodification 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 from5 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
$\underbrace{0\ 0\ 0\ 1}_{1}$	$\underbrace{0\ 0\ 1\ 1}_{3}$	→ Valid BCD number	13_{10}

(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
$\underbrace{0\ 0\ 1\ 1}_{3}$	$\underbrace{0\ 0\ 1\ 1}_{3}$	→ Valid BCD number	33_{10}

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 ofthe binary number is not assigned a fixed value.
- Excess-3 codes and Gray codes are such non-weighted codes.

Excess-3 code:

- Excess-3isanon- weightedcodeusedtoexpressdecimalnumbers.Thecodederivesitsnamefrom thefactthateachbinarycodeisthecorresponding8421codeplus0011(3).

Example:1000of8421 (BCD)=1011in 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:

- Thegraycodebelongstoaclassofcodescalledminimumchangecodes,inwhichonlyonebitin thecodechangeswhenmovingfrom onecodetothenext.
- TheGraycodeisnon-weightedcode,asthe positionofbitdoesnotcontainanyweight.In digitalGraycodehasgot a specialplace.

Decimal Number	**BinaryCode**	**GrayCode**
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

- Thegraycodeisareflective digitalcodewhichhas the special propertythat anytwosubsequentnumberscodes differ byonlyonebit. This is also calleda*unit-distance code*.
- Importantwhenananalogquantity mustbeconvertedtoadigitalrepresentation.Onlyonebitchanges between two successiveintegers whicharebeing coded.

Example:

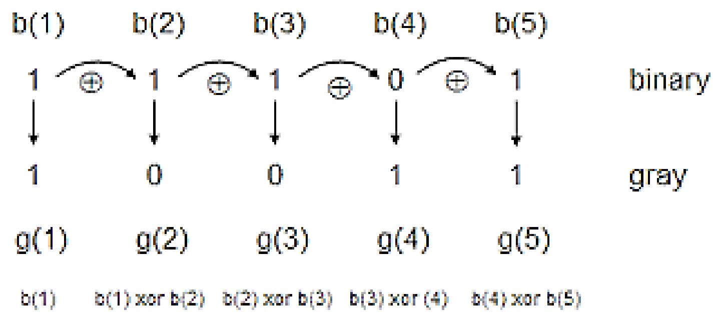
Binary toGray CodeConversion:

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.



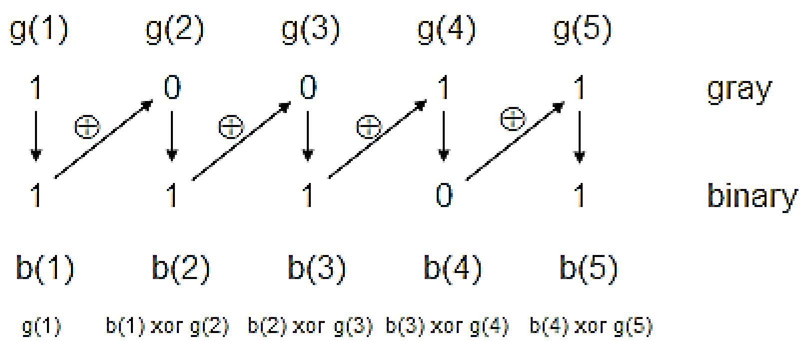
GrayCode to Binary Code Conversion:

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, or itisjuststored, likeinharddisksandmemories,therearechancesthatdata maygetcorrupted.
- 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.

****

Boolean Algebra and Theorems

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

$(\overline{A}) = \overline{A}$ and $(\overline{\overline{A}}) = A$

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) $\mathrm{A}.(\bar{A} + B) = A.B$

Postulates of Boolean algebra:

The postulates of a mathematical system form the basic assumptions from which itis 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 +:

$$a \,.\, (b + c) = (a \,.\, b) + (a \,.\, c)$$

Two-Valued Boolean Algebra:

A two-valued Boolean algebra is defined on a set of two elements, $B = \{0, 1\}$, with rulesfor 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)

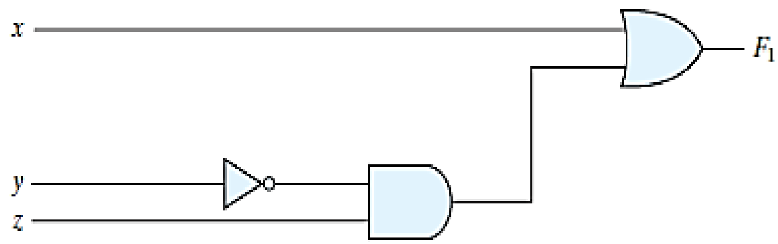
THEOREM 6(b): $x(x + y) = x$ by duality.

****

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 equalto 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 relationshipbetween binary variables and is evaluated by determining the binary value ofthe 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. $$\begin{aligned} xy + x'z + yz &= xy + x'z + yz(x + x') \\ &= xy + x'z + xyz + x'yz \\ &= xy(1 + z) + x'z(1 + y) \\ &= xy + x'z. \end{aligned}$$
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'sand 1's for 0's in the value of F.

Example:

1.

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

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 areobtained 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 beexpressed 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,

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

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 toremove one of those occurrences. Rearranging the minterms in ascending order, wefinally 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)*

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

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:

$$\begin{aligned} F = xy + x'z &= (xy + x')(xy + z) \\ &= (x + x')(y + x')(x + z)(y + z) \\ &= (x' + y)(x + z)(y + z) \end{aligned}$$

The function has three variables: x, y, and z. Each OR term is missing one variable; therefore,

$$\begin{aligned} x' + y &= x' + y + zz' = (x' + y + z)(x' + y + z') \\ x + z &= x + z + yy' = (x + y + z)(x + y' + z) \\ y + z &= y + z + xx' = (x + y + z)(x' + y + z) \end{aligned}$$

Combining all the terms and removing those which appear more than once, we finally obtain

$$\begin{aligned} F &= (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z') \\ &= M_0M_2M_4M_5 \end{aligned}$$

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.

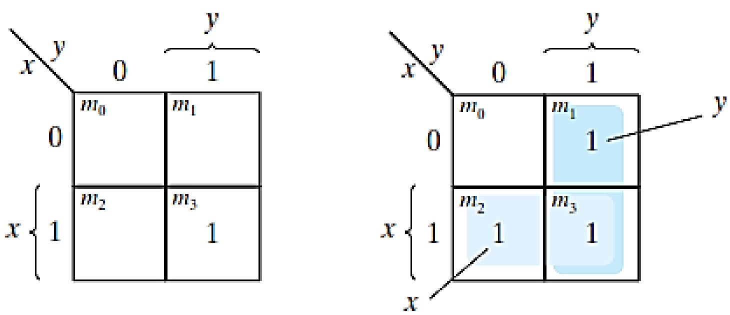
Obtaining Product Terms

- 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:

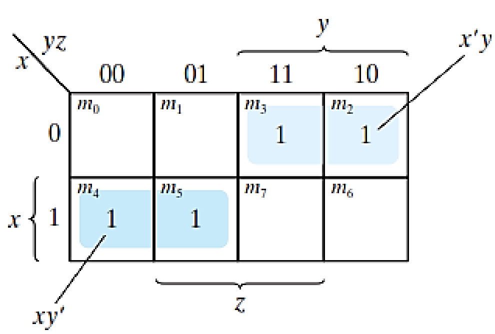
K-Maps of 2 Variables:



(a) xy (b) $x + y$

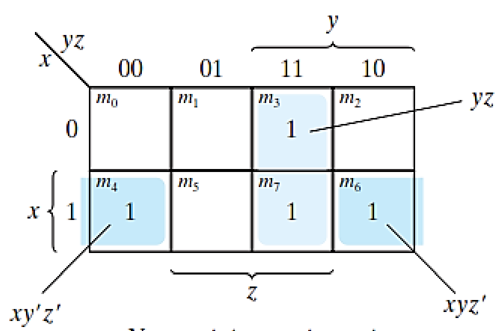
K-Maps of 3 Variables:

❖ Simplify the boolean function $F(x, y, z) = \Sigma(2, 3, 4, 5)$



$$F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$$

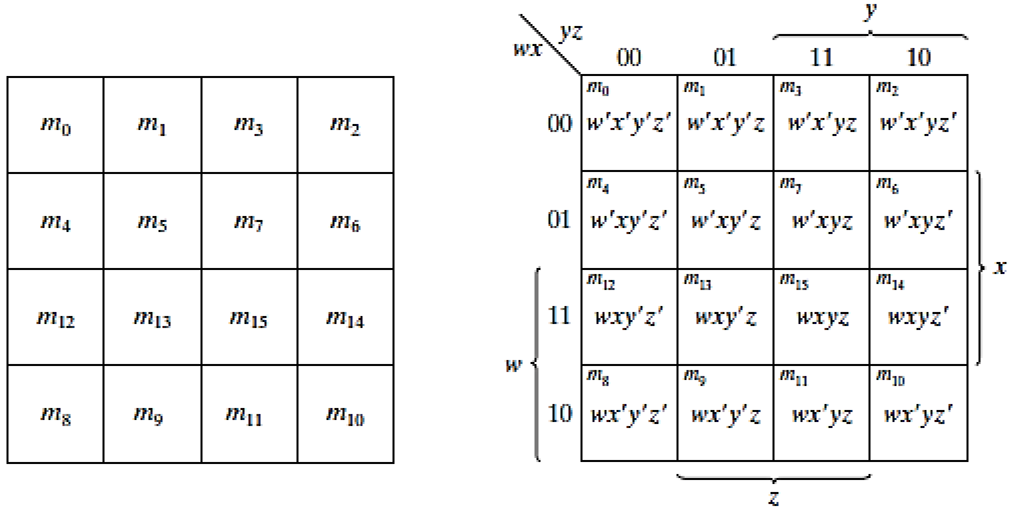
❖ Simplify the boolean function $F(x, y, z) = \Sigma\ 3, 4, 6, 7$



Note: $xy'z' + xyz' = xz'$

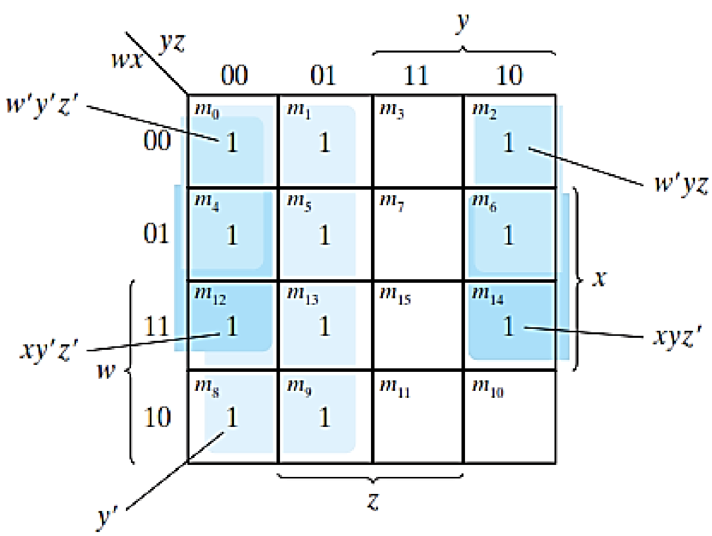
$$F(x, y, z) = \Sigma(3, 4, 6, 7) = yz + xz'$$

K-Maps of 4 Variables:



(a) (b)

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



Note: $w'y'z' + w'yz' = w'z'$

$xy'z' + xyz' = xz'$

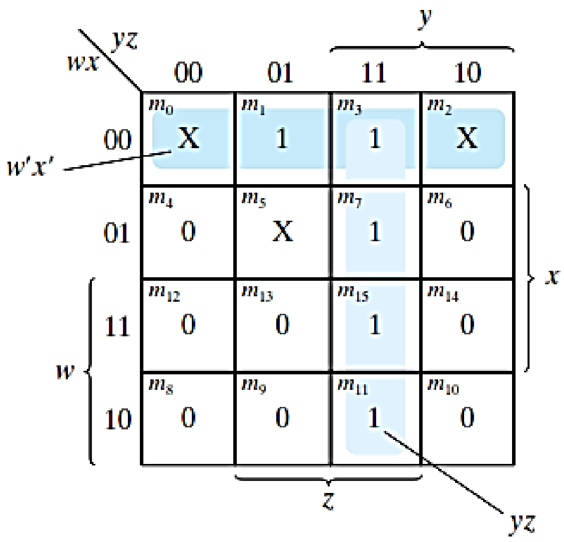
$$F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14) = y' + w'z' + xz'$$

❖ Simplify the Boolean function

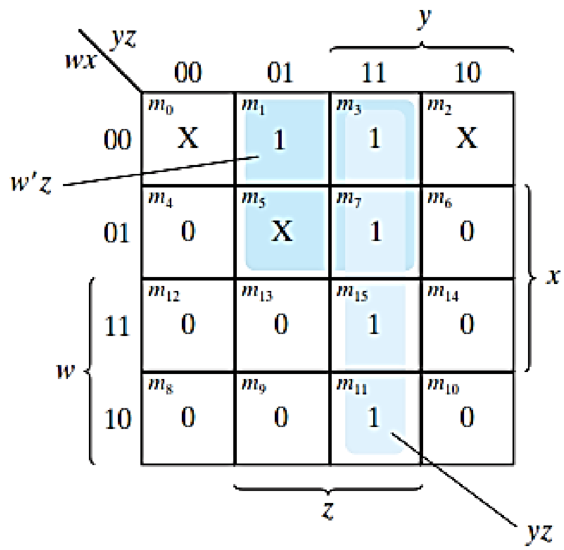
$$F(w, x, y, z) = \Sigma(1, 3, 7, 11, 15)$$

which has the don't-care conditions

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



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



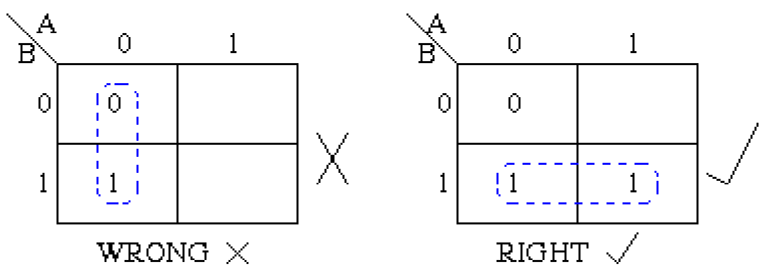
(b) $F = yz + w'z$

Note:

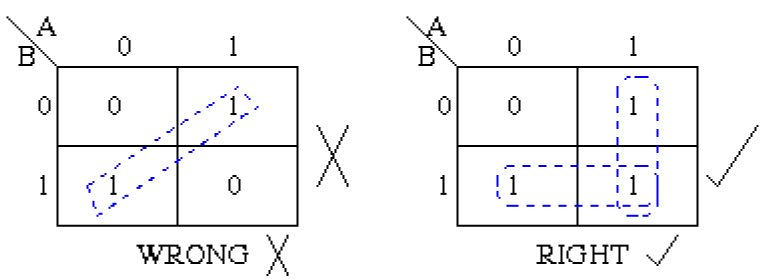
Karnaugh Maps - Rules of Simplification

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

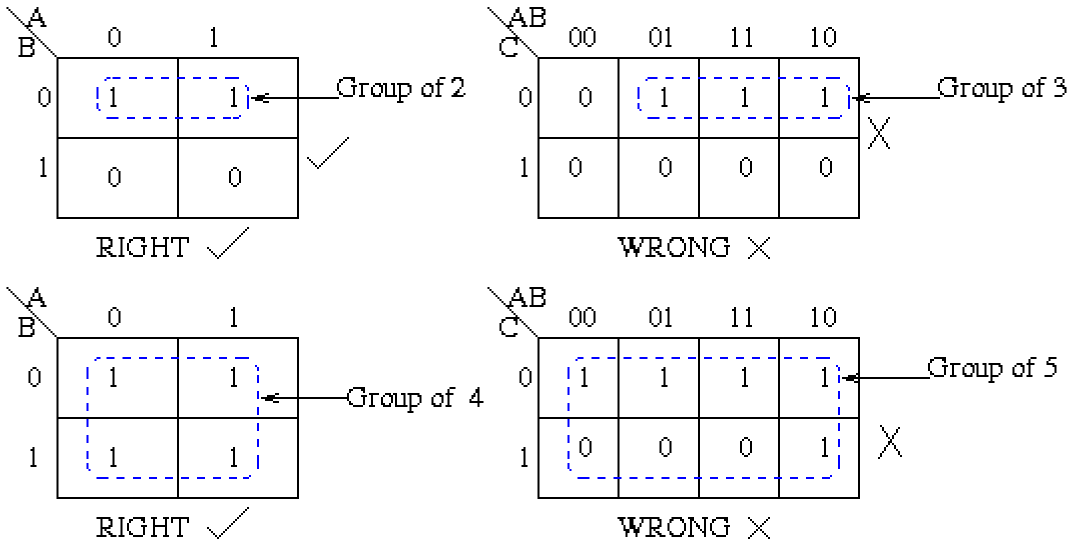
- **Groups may not include any cell containing a zero**



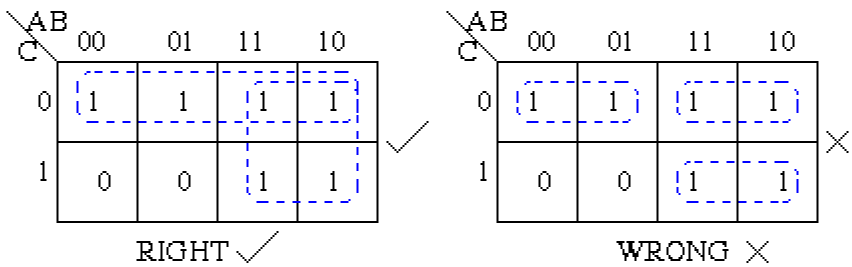
- **Groups may be horizontal or vertical, but not diagonal.**



- **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 $2^2 = 4$.**

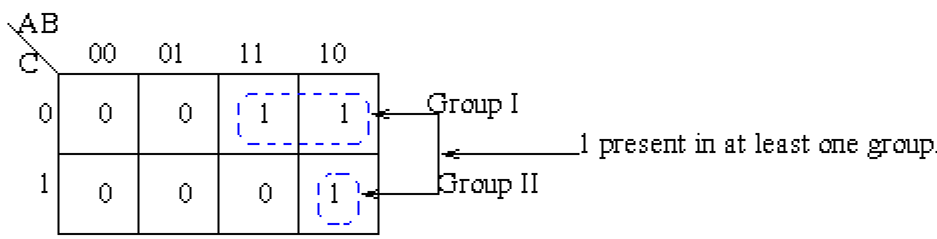


- **Each group should be as large as possible.**

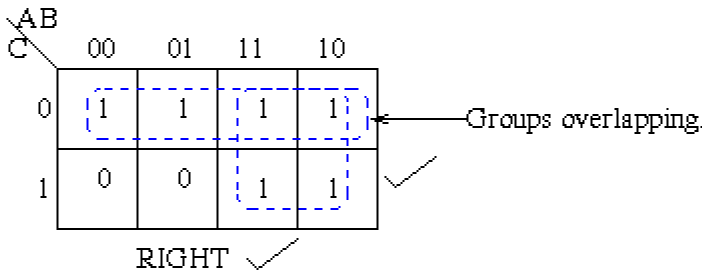


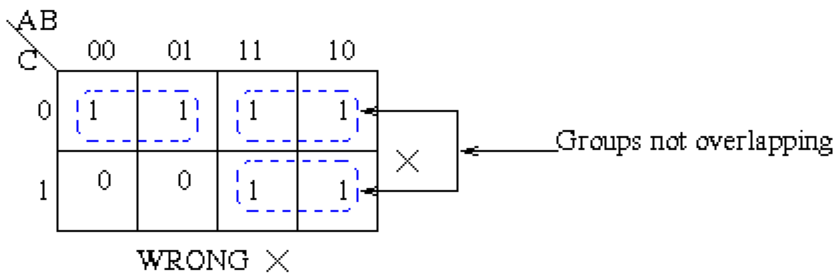
(Note that no Boolean laws broken,
but not sufficiently minimal)

- **Each cell containing a *one* must be in at least one group.**

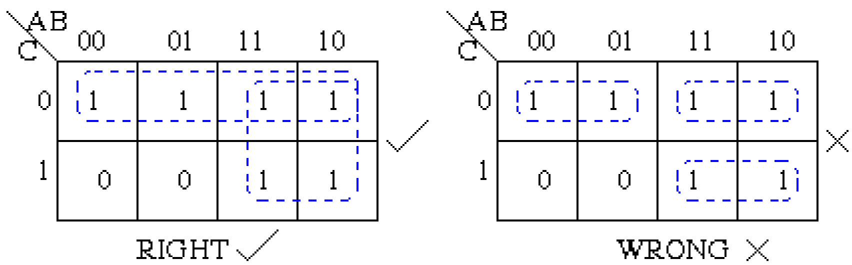


- **Groups may overlap.**





- **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
m_1	0 0 0 1
m_3	0 0 1 1
m_6	0 1 1 0
m_7	0 1 1 1
m_8	1 0 0 0
m_9	1 0 0 1
m_{10}	1 0 1 0
m_{12}	1 1 0 0
m_{14}	1 1 1 0
m_{15}	1 1 1 1
dm_{11}	1 0 1 1
dm_{13}	1 1 0 1

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

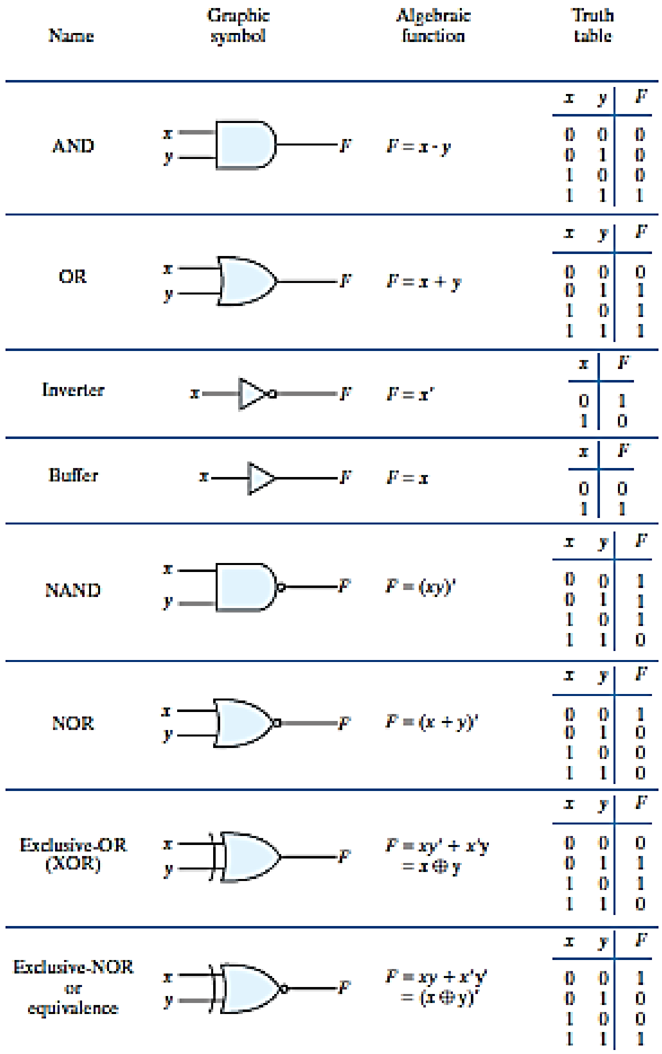
- A **logic gate** is an idealized or physical device implementing a Boolean function; that is, it performs a logical operation on one or more logical inputs, and produces a single logical output.

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 ofa logic system.



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

Universal Gates

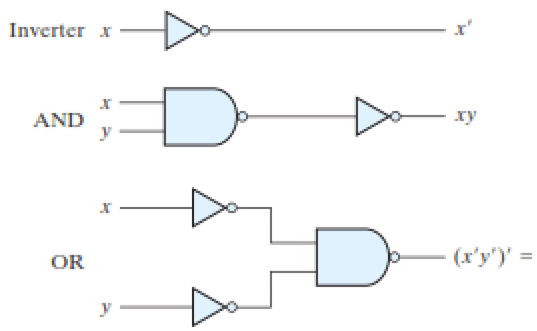
- The OR, AND and NOT gates are the three basic logic gates as they together can be used to construct the logic circuit for any given Boolean expression.
- The NOR and NAND gates have the property that they individually can be used to hardware-implement a logic circuit corresponding to any given Boolean 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.

NAND gatesandNOR gatesarecalled*universalgates* or *universalbuildingblocks*, as any type of gates or logic functions can be implemented by these gates. Figures Symbolshowshow variouslogic functionscan be realizedby NAND gatesandFigures Symbolshow therealizationofvariouslogic gatesby NOR gates.

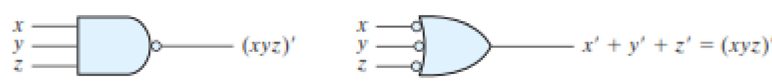
NOT function:F =A'AND function: F =AB

❖ *Implementation of basic gates using NAND gate:*

(convert AND gate to NAND gate)



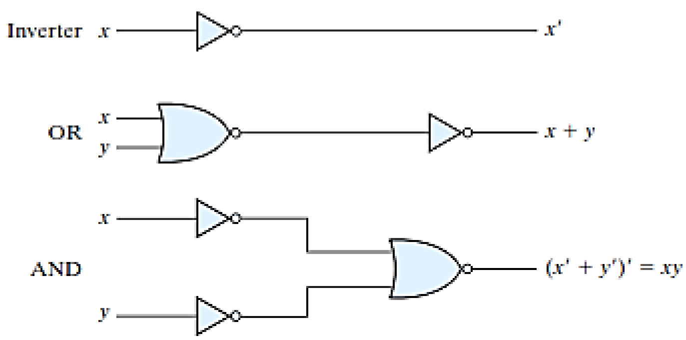
Logic operations with NAND gates



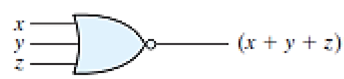
(a) AND-invert (b) Invert-OR

❖ *Implementation of basic gates using NOR gate:*

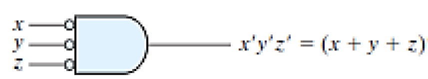
(convert OR gate to NOR gate)



Logic operations with NOR gates



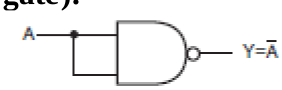
(a) OR-invert



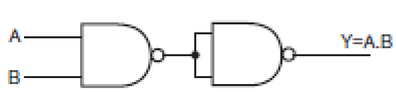
(b) Invert-AND

Implementation of basic gates using NAND gate:

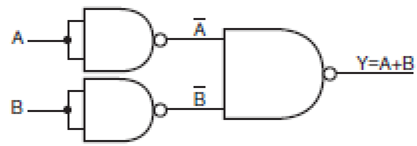
Inverter (NOT gate):



AND gate:

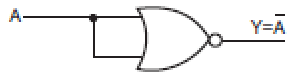


OR gate:

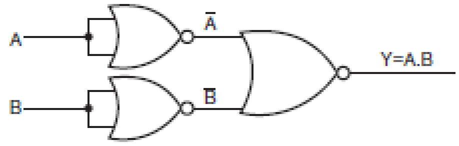


Implementation of basic gates using NOR gate:

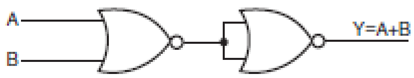
Inverter (NOT gate):



AND gate:



OR gate:

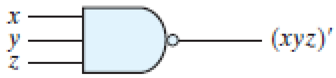


NAND–NOR implementations:

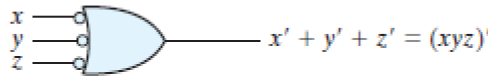
- Digital circuits are frequently constructed with NAND or NOR gates rather than with AND and OR gates.
- NAND and NOR gates are easier to fabricate with electronic components and are the basic gates used in all IC digital logic families.
- Because of the prominence of NAND and NOR gates in the design of digital circuits, rules and procedures have been developed for the conversion from Boolean functions given in terms of AND, OR, and NOT into equivalent NAND and NOR logic diagrams.

Only NAND/NOR gate circuit:

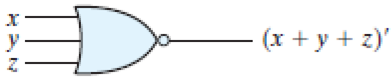
- A convenient way to implement a Boolean function with NAND/NOR gates is to obtain the simplified Boolean function in terms of Boolean operators and then convert the function to NAND/NOR logic.
- The conversion of an algebraic expression from AND, OR, and complement to NAND/NOR can be done by simple circuit manipulation techniques that change AND–OR diagrams to NAND/NOR diagrams.



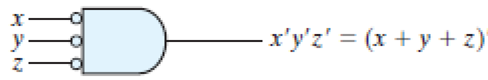
(a) AND-invert



(b) Invert-OR



(a) OR-invert



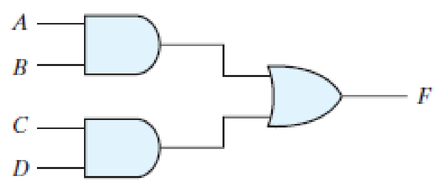
(b) Invert-AND

NAND Implementation Procedure:

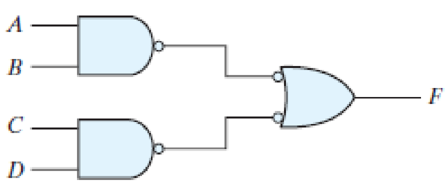
- ✓ Draw the AOI logic of given Boolean expression.
- ✓ Add bubble on input of OR gate & output of AND gate.
- ✓ Add an Inverter on each line that received bubbles.
- ✓ Eliminate double inversions
- ✓ Replace all by NAND gates

Example:

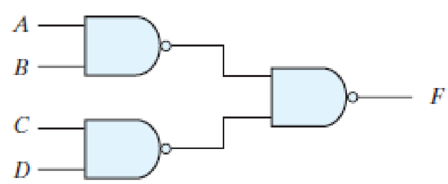
1. **Implement F = AB + CD using only NAND gate.**



(a)

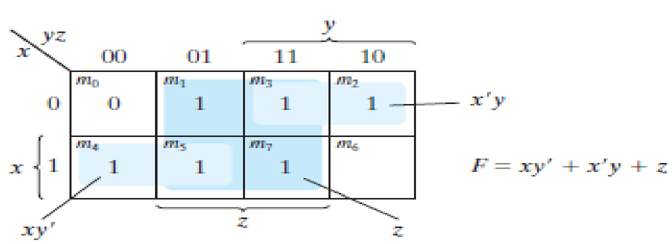


(b)

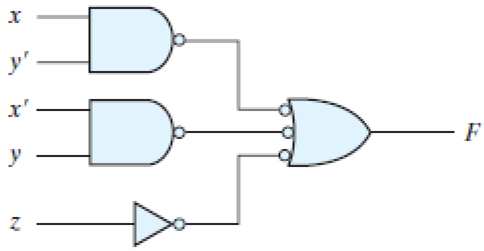


(c)

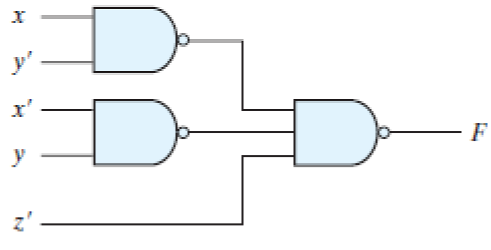
2. Implement the following Boolean function with NAND gates:*F* (*x*, *y*, *z*) = (1, 2, 3, 4, 5, 7) (Apr 2018)



(a)

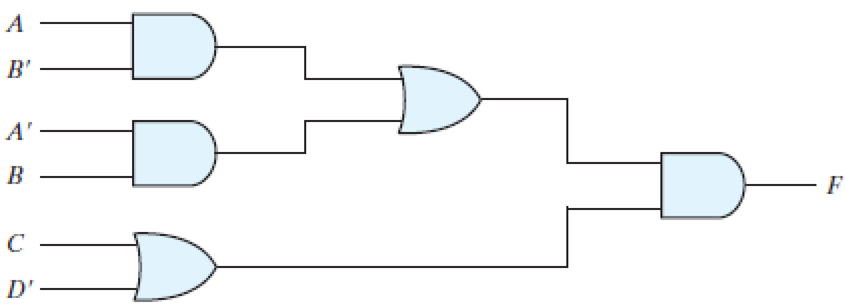


(b)

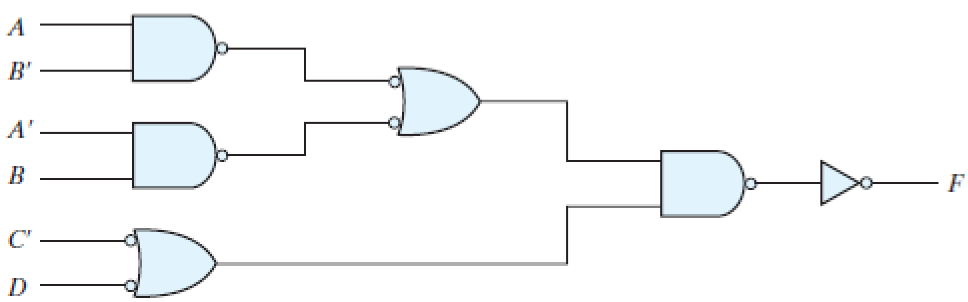


(c)

3. Implement the function $F = (AB' + A'B)(C + D')$ using only NAND gate.



(a) AND–OR gates



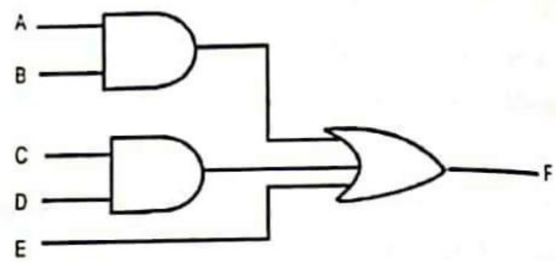
(b) NAND gates

2.7.1 NAND gate implementation

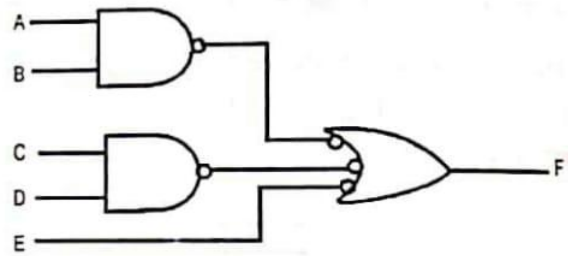
Example 2.10: Implement the Boolean function with NAND gates. $F = AB + CD + E$

Solution:

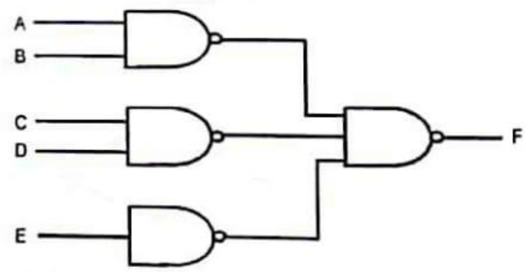
Step 1: Draw AND-OR circuit.



Step 2: Add bubbles on output of each AND gate and input of OR gate.



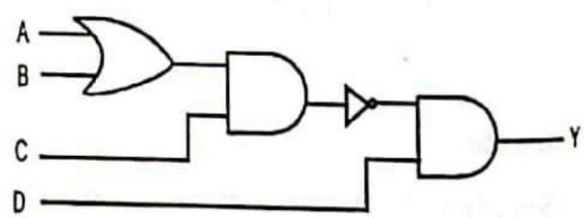
Step 3: Replace other gates by NAND gates.



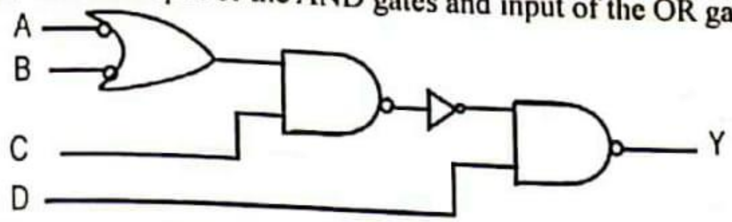
Example 2.11: Implement the Boolean expression with NAND gates $Y = \left(\overline{(A+B)C}\right)D$

Solution: Step 1: Draw original logic diagram for

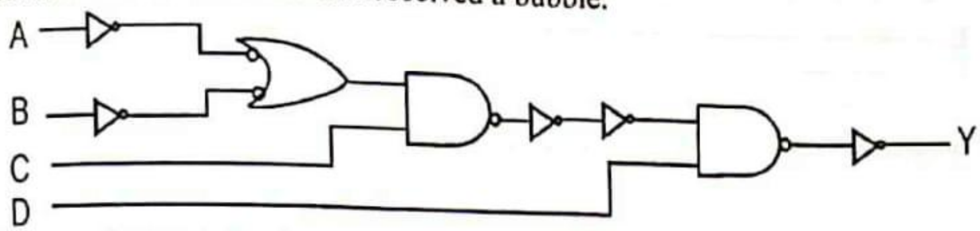
$$Y = \left(\overline{(A+B)C}\right)D$$



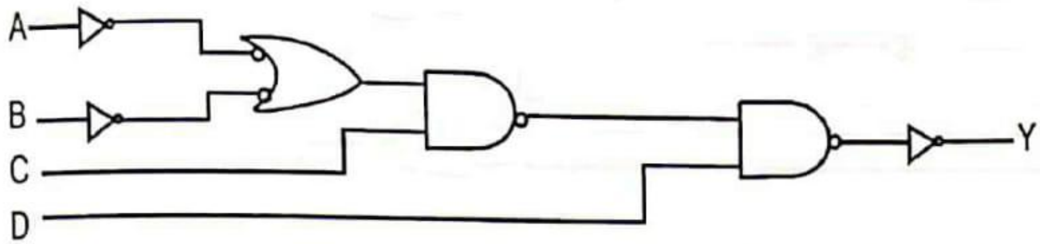
Step 2: Add bubbles on the output of the AND gates and input of the OR gate.



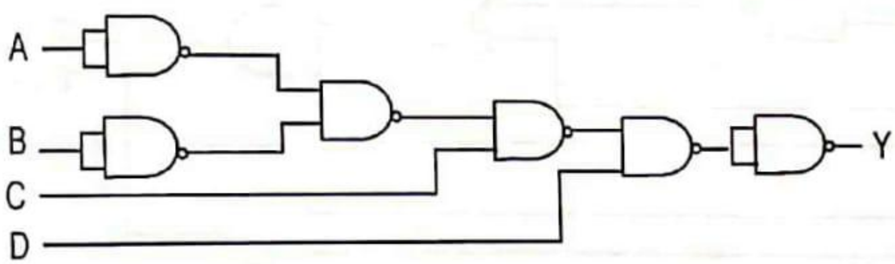
Step 3: Add inverters on each line that received a bubble.



Step 4: Eliminate double inversions.

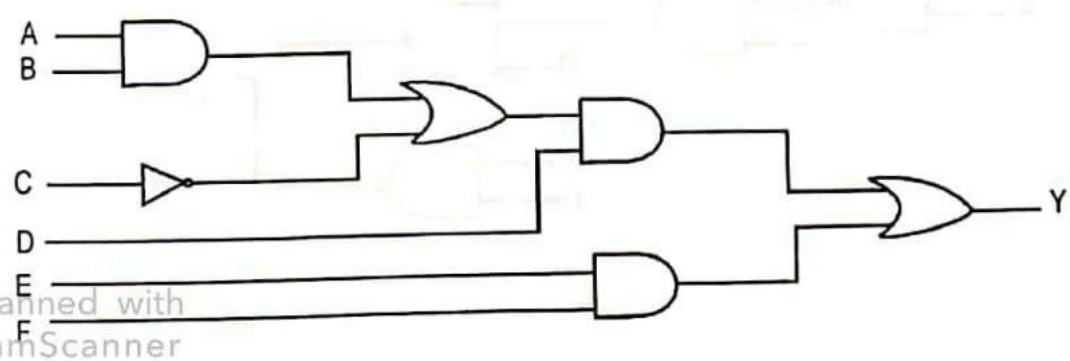


Step 5: Replace the other gates by only NAND gates.

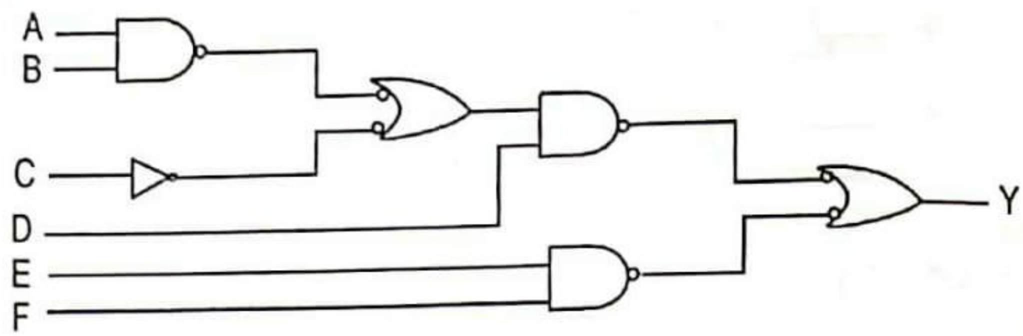


Example 2.12: Implement NAND gates for $Y=\left(AB+\overline{C}\right)D+EF$. ***(April 2004)***

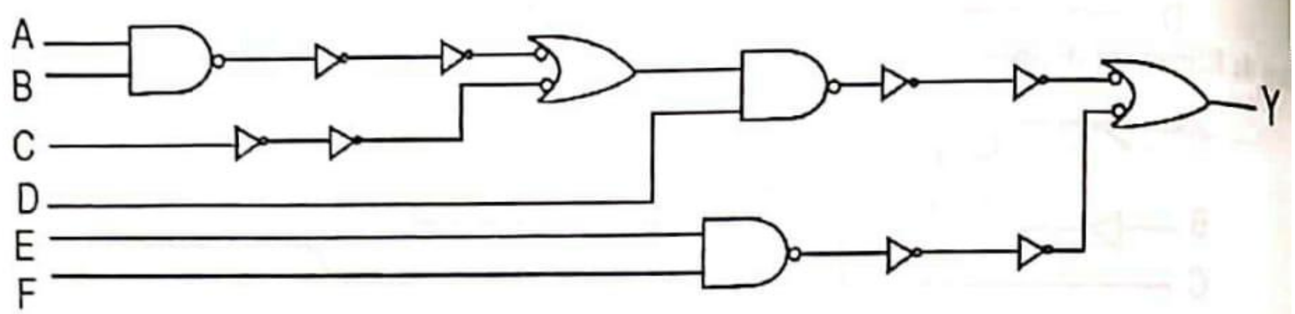
Solution: Step 1: Draw the original logic diagram for the expression $Y=\left(AB+\overline{C}\right)D+EF$



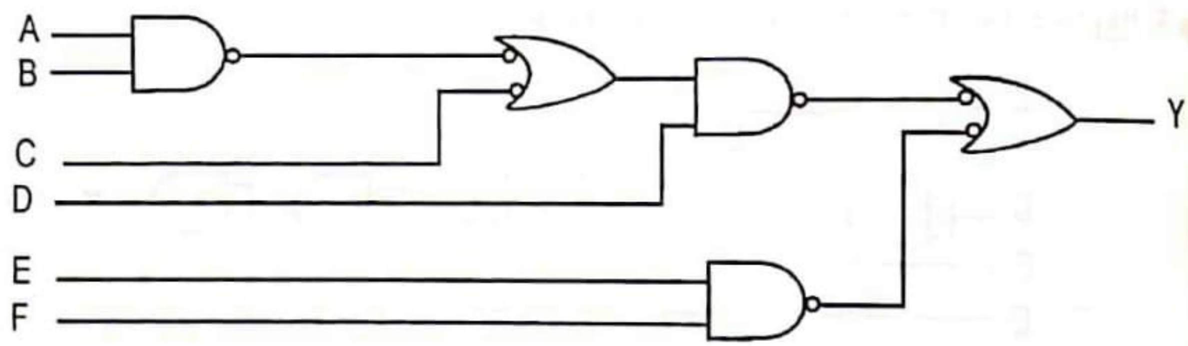
Step 2: Add bubbles on the output of the AND gates and input of the OR gates.



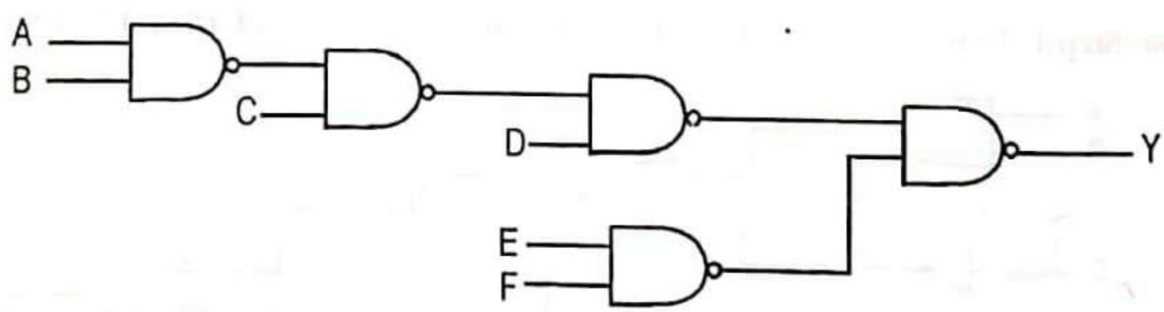
Step 3: Add inverters on each line that received a bubble.



Step 4: Eliminate double inversions.



Step 5: Draw the circuit with only NAND gates.

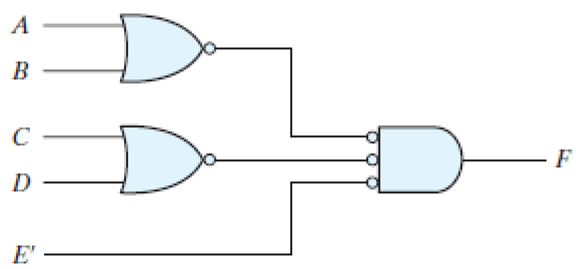


NOR Implementation Procedure:

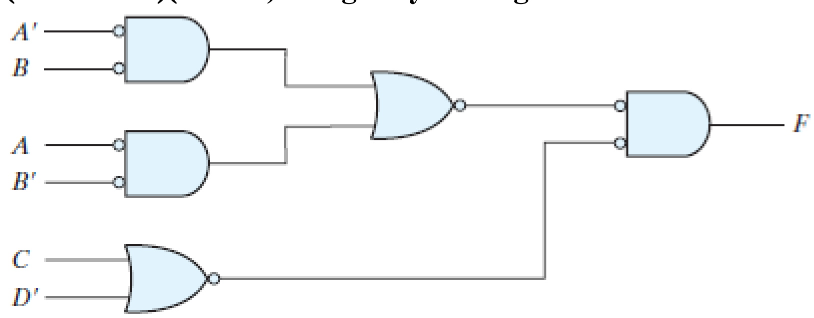
- ✓ Draw the AOI logic of given Boolean expression.
- ✓ Add bubble on input of AND gate & output of OR gate.
- ✓ Add an Inverter on each line that received bubbles.
- ✓ Eliminate double inversions
- ✓ Replace all by NOR gates

Example:

1. **Implement $F = (A + B)(C + D)E$ *using only NOR gate. (Apr 2018)***



2. **Implement $F = (AB' + A'B)(C + D')$ using only NOR gate.**



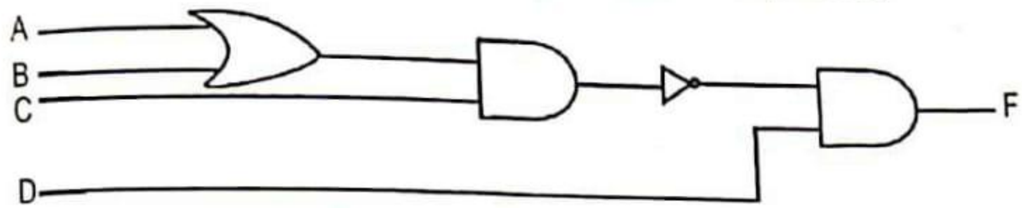
2.7.2 NOR gate implementation

Example 2.13: Implement the Boolean expression with NOR gates.

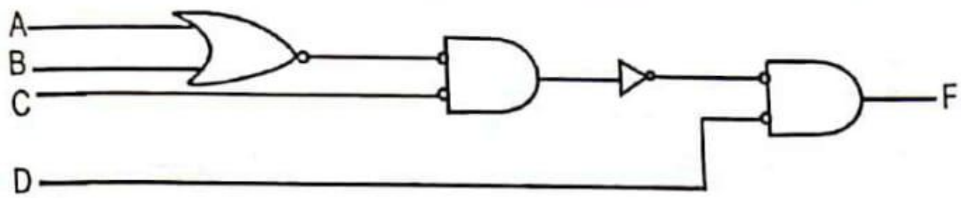
$$F = \overline{(A+B)C}.D$$

Solution:

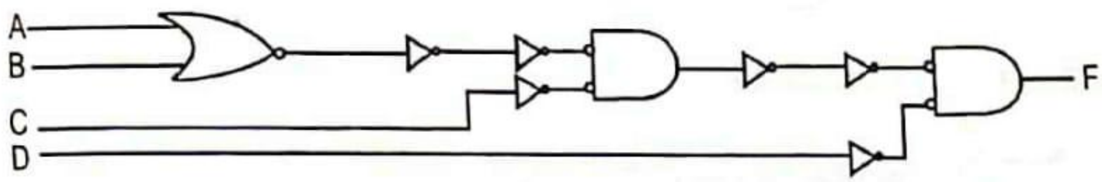
Step 1: Draw the original logic diagram for the given Boolean expression,



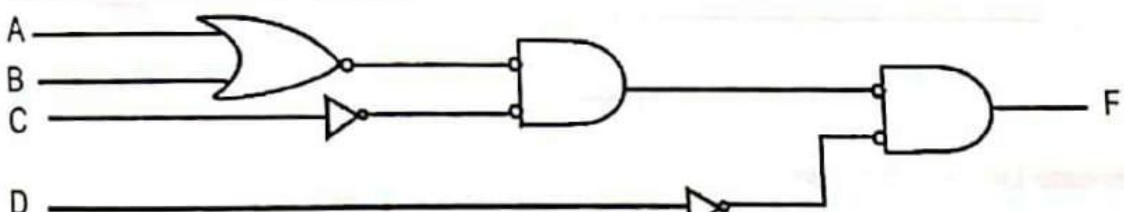
Step 2: Add bubbles on output of each OR gate and add bubbles on input of each AND gate.



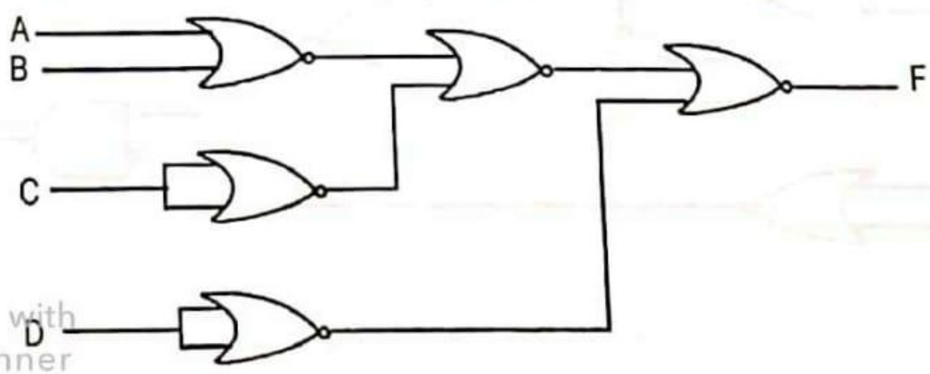
Step 3: Add inverters on each line that recieved bubbles.



Step 4: Eliminate double inversions.



Step 5: Draw the NOR diagram with one graphic symbol.

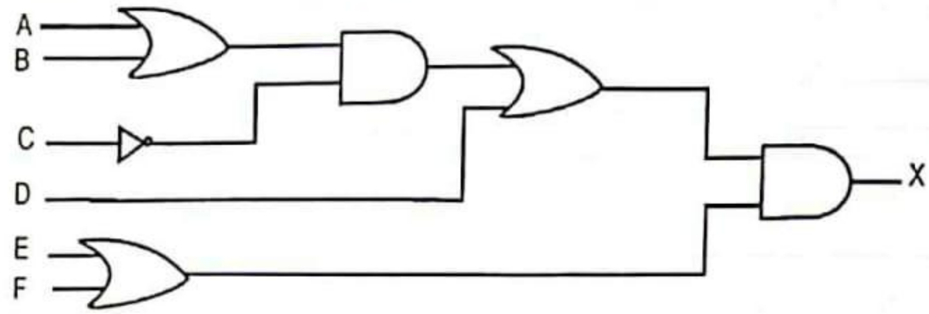


Example 2.14: Draw the multi level NOR circuit for the Boolean expression:

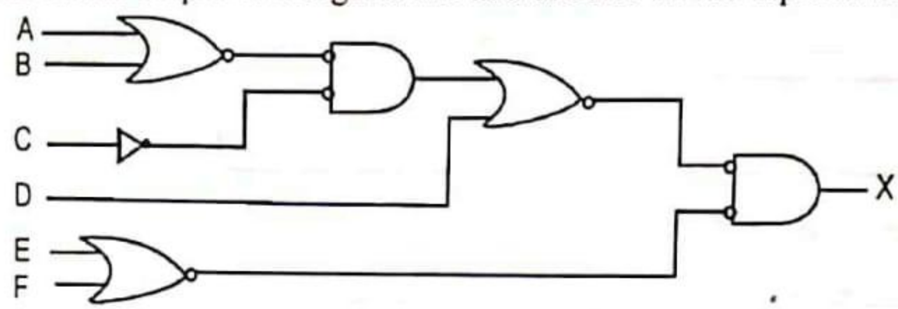
$$X = \left[(A+B)\overline{C} + D\right](E+F)$$

Solution:

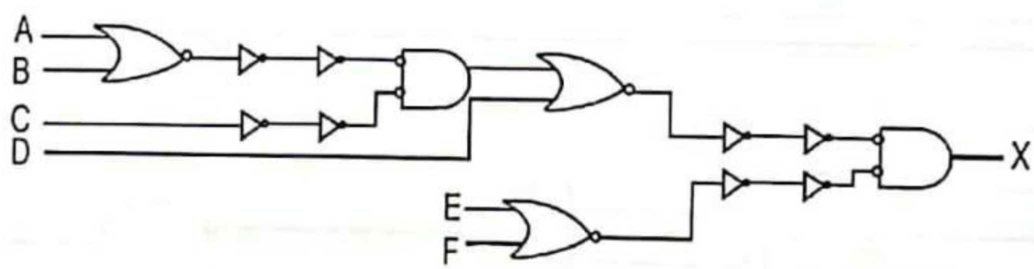
Step 1: Draw the original circuit diagram for $X = \left[(A+B)\overline{C} + D\right](E+F)$



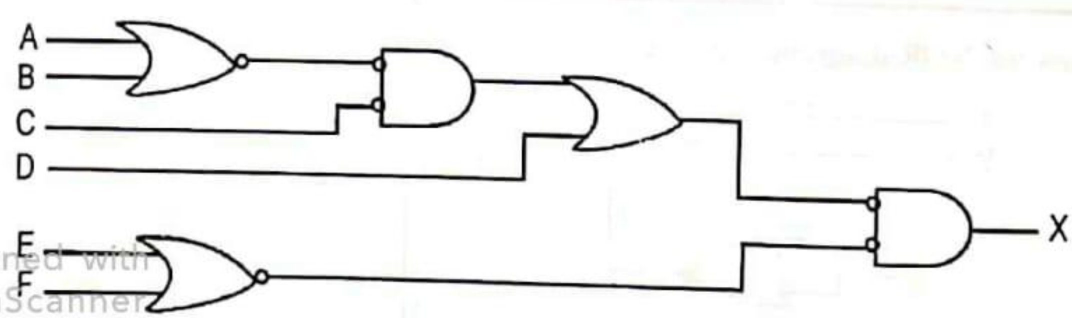
Step 2: Add bubbles on the output of OR gates and add bubbles on the input of AND gates.



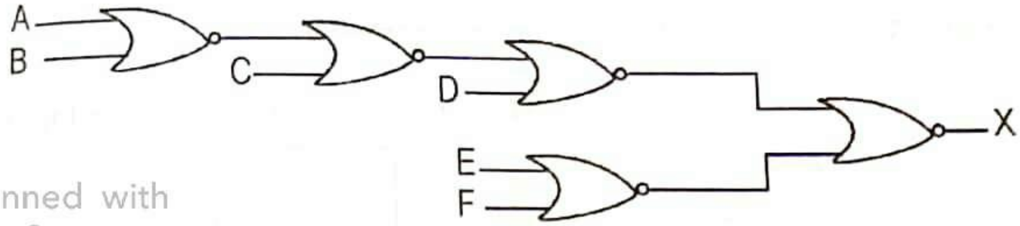
Step 3: Add inverters on each line that received bubbles.



Step 4: Eliminate Double Versions.



Step 5: Draw the NAND diagram using one graphic symbol.

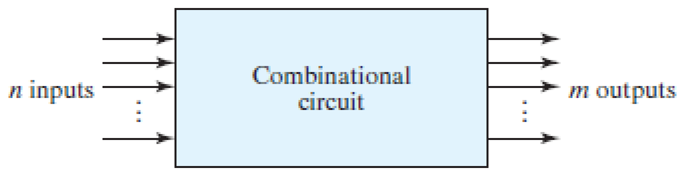


UNIT II
COMBINATIONAL LOGIC

Combinational Circuits – Analysis and Design Procedures - Binary Adder- Subtractor -Decimal Adder - Binary Multiplier - Magnitude Comparator - Decoders – Encoders – Multiplexers - Introduction to HDL – HDL Models of Combinational circuits.

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:

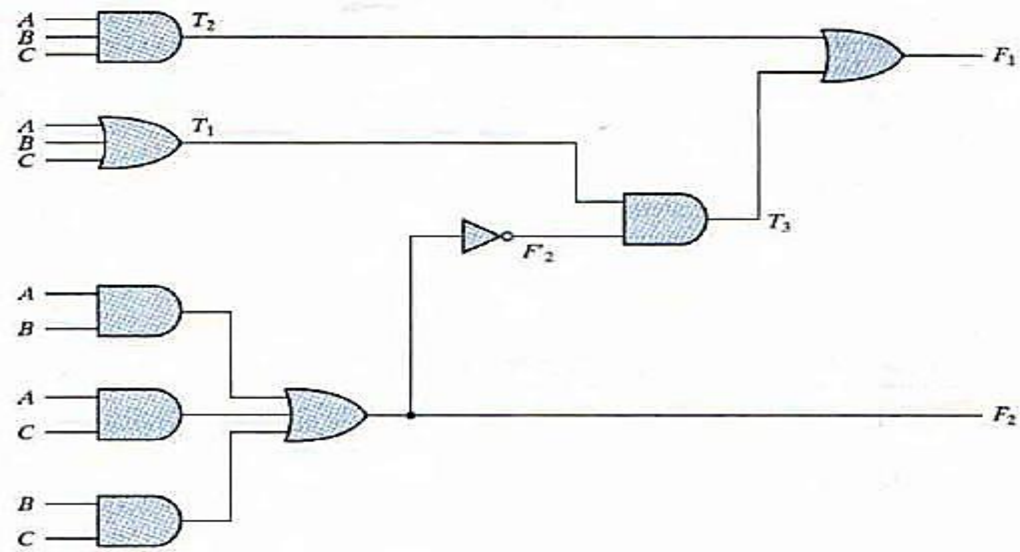
- *Sequential circuits employ storage elements in addition to logic gates. Their outputs are a function of the inputs and the state of the storage elements.*
- Because the state of the storage elements is a function of previous inputs, the outputs of a sequential circuit depend not only on present values 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

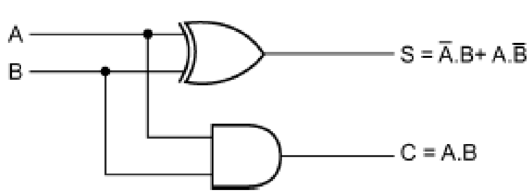
SUM $S = A.\overline{B} + \overline{A}.B$

CARRY $C = A.B$

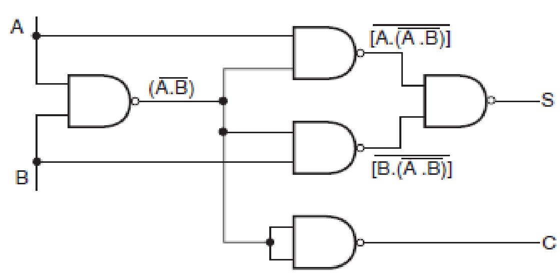
Truth Table:

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:



****

Full adder:

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:

	A′B′	A′B	AB	AB′
X′		1		1
X	1		1	

K-Map for Sum

	A′B′	A′B	AB	AB′
X′			1	
X		1	1	1

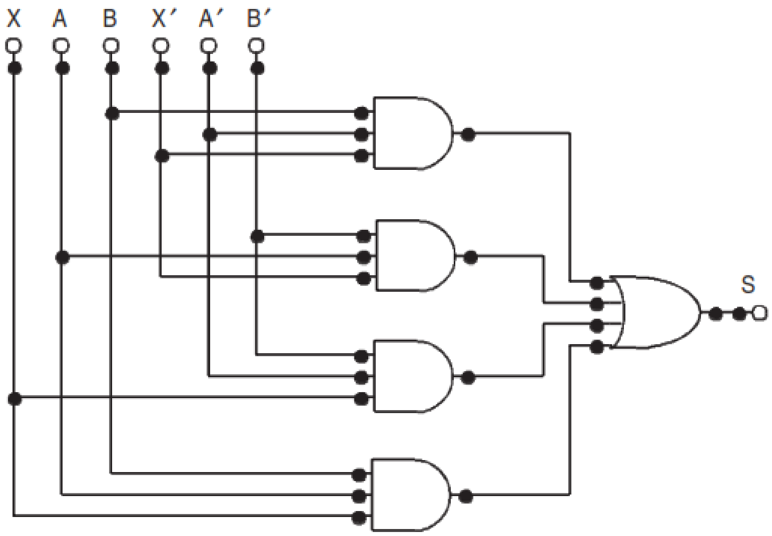
K-Map for Carry

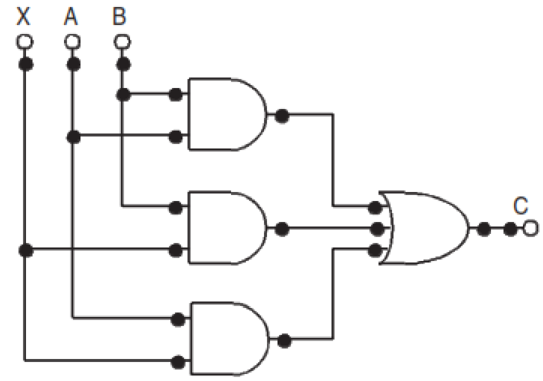
- The simplified Boolean expressions of the outputs are

S = X′A′B + X′AB′ + XA′B′ + XAB

C = AB + BX + AX

Logic diagram:



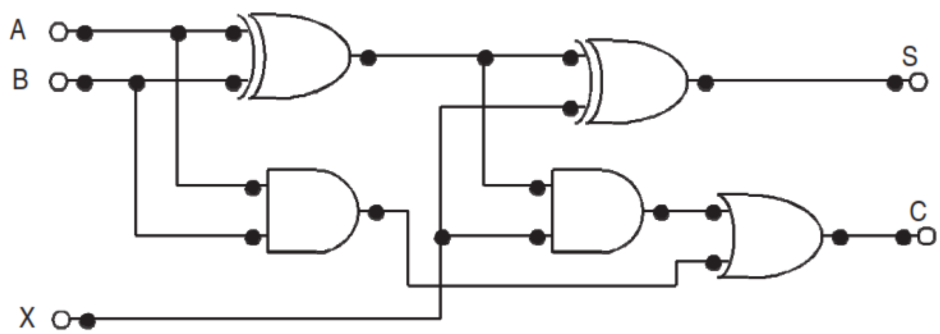


- The Boolean expressions of S and C are modified as follows

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

Full adder using Two half adder:

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



****

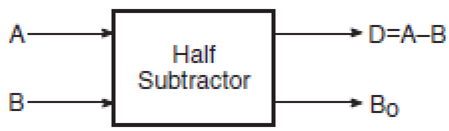
Half subtractor:

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

- A half-subtractor is a combinational circuit that can be used to subtract one binary digit from anotherto 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:

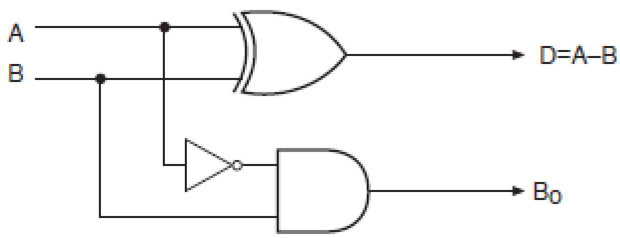
$$D = \overline{A}.B + A.\overline{B}$$

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



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)***

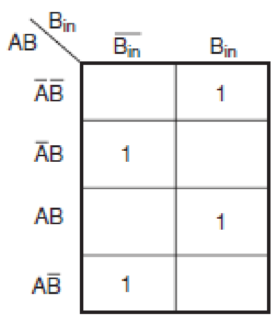
- A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend, and also takes into consideration whether a '1' has already been borrowed by the previous adjacent lower minuend bit or not.
- As a result, there are three bits to be handled at the input of a full subtractor, 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:

$$D = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + A.\overline{B}.\overline{B}_{in} + A.B.B_{in}$$

$$B_o = \overline{A}.B + \overline{A}.B_{in} + B.B_{in}$$

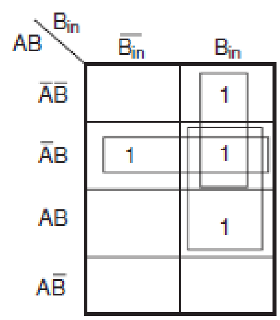
Minuend (A)	Subtrahend (B)	Borrow In (B_{in})	Difference (D)	Borrow Out (B_O)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-Map:



(a)

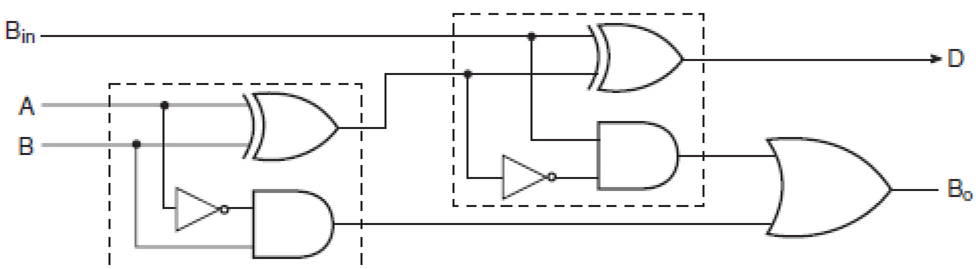
Difference

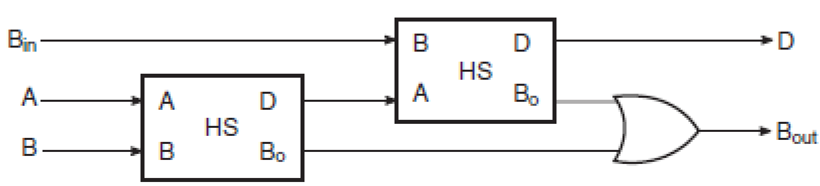


(b)

Barrow

Full subtractor using two half subtractor:





****

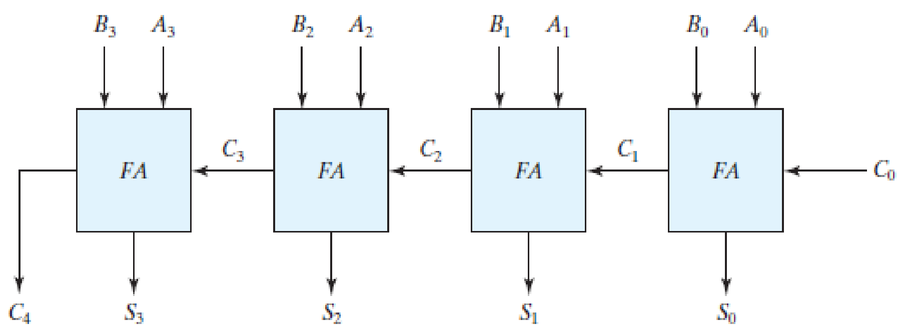
Parallel Binary Adder: (Ripple Carry Adder):

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 bychanging 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) $Y = 1000011$

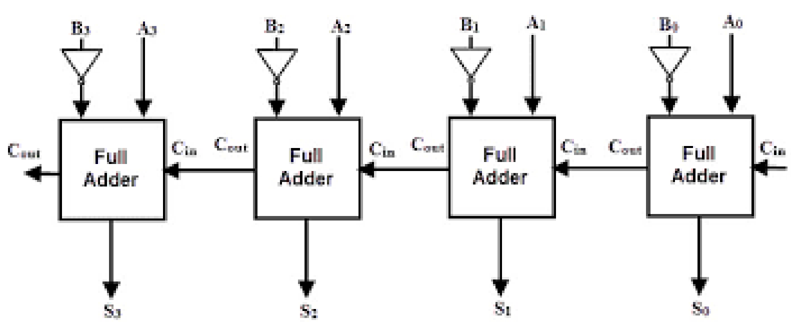
2's complement of $X = + 0101100$

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 meansof complements. The subtraction $A - B$ canbe done by taking the 2's complement of B and adding it to A . The 2's complement canbe 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.

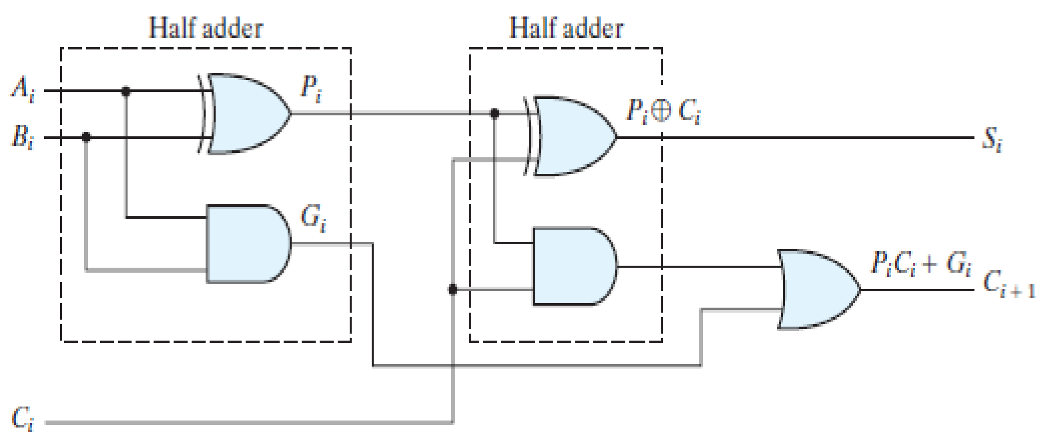
- ✓ The circuit for subtracting *A* - *B* consists of an adder with inverters placed betweeneach data input *B* and the corresponding input of the full adder. The input carry *C*in mustbe equal to 1 when subtraction is performed. The operation thus performed becomes *A*,plus the 1's complement of *B*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

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

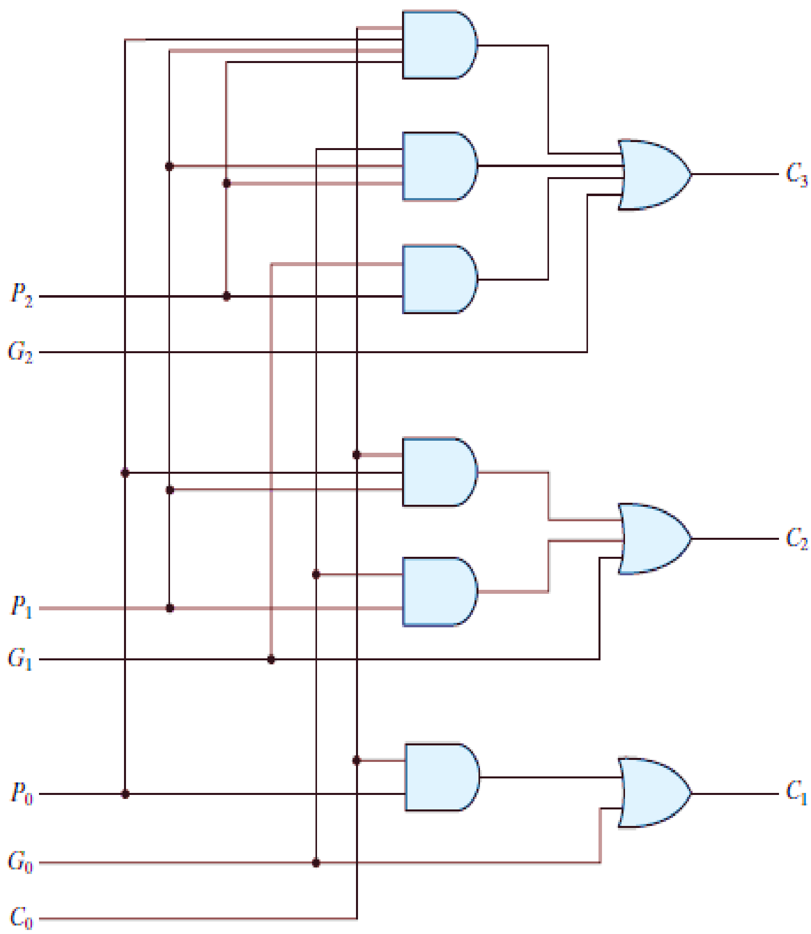
the output sum and carry can respectively be expressed as

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

Gi is called a carry generate, and it produces a carry of 1 when both Ai and Bi are 1,regardless of the input carry Ci. Pi 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 Ci will propagate to an assertion of Ci+1).

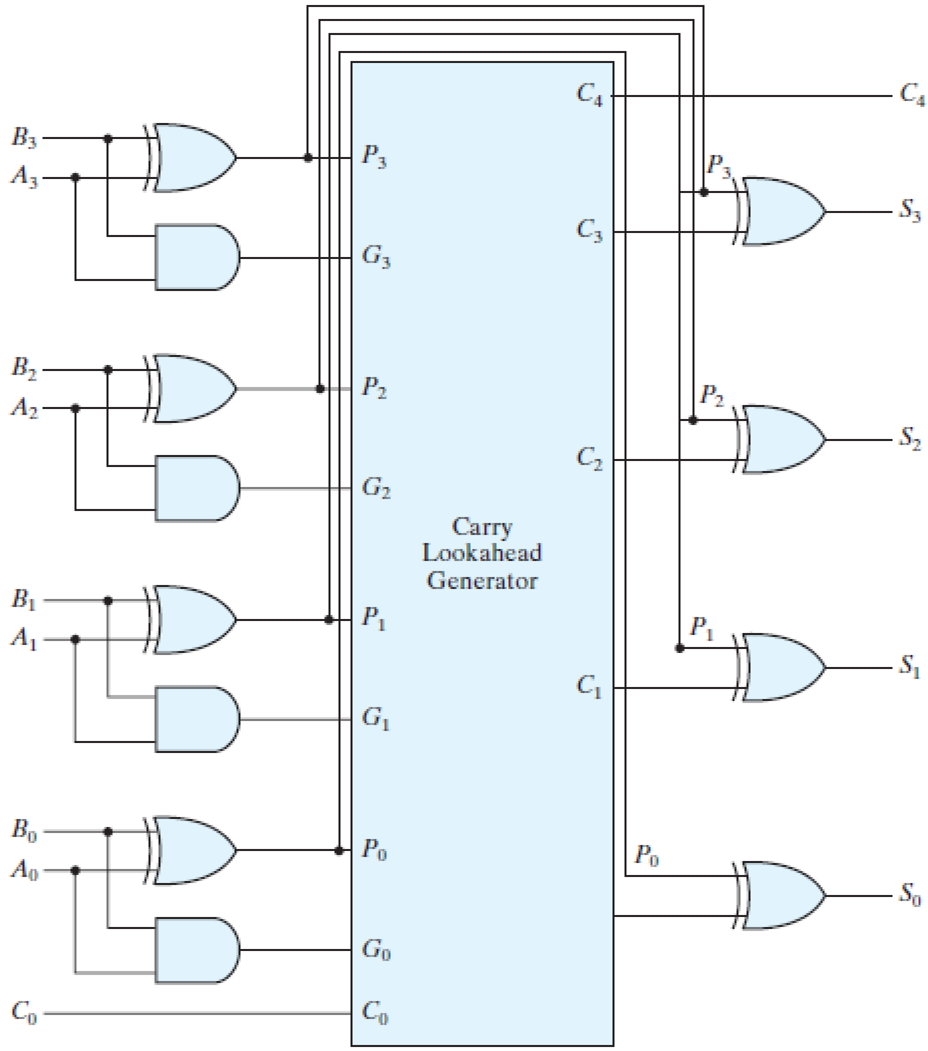
We now write the Boolean functions for the carry outputs of each stage and substitutethe value of each Ci from the previous equations:

$$C_0 = \text{input carry}$$
$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$$
$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 = P_2P_1P_0C_0$$



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.

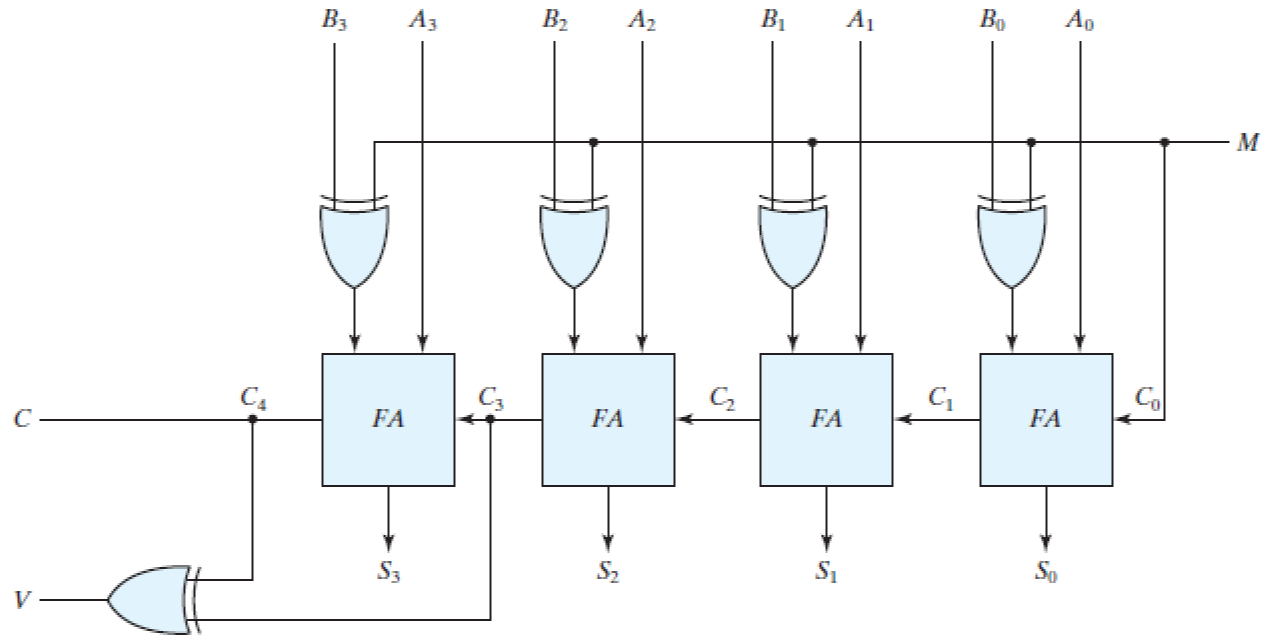


****

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.

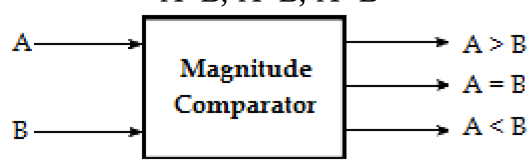
****

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:

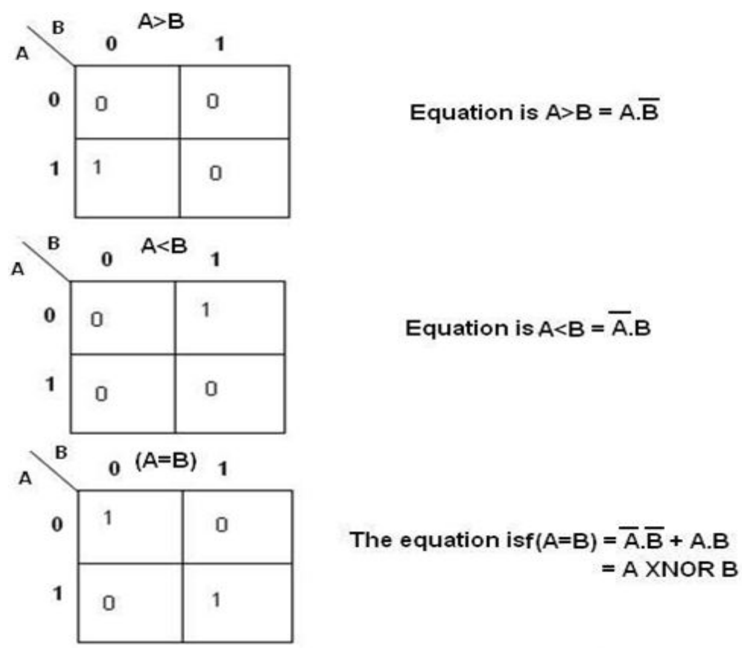
A<B, A=B, A>B



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



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

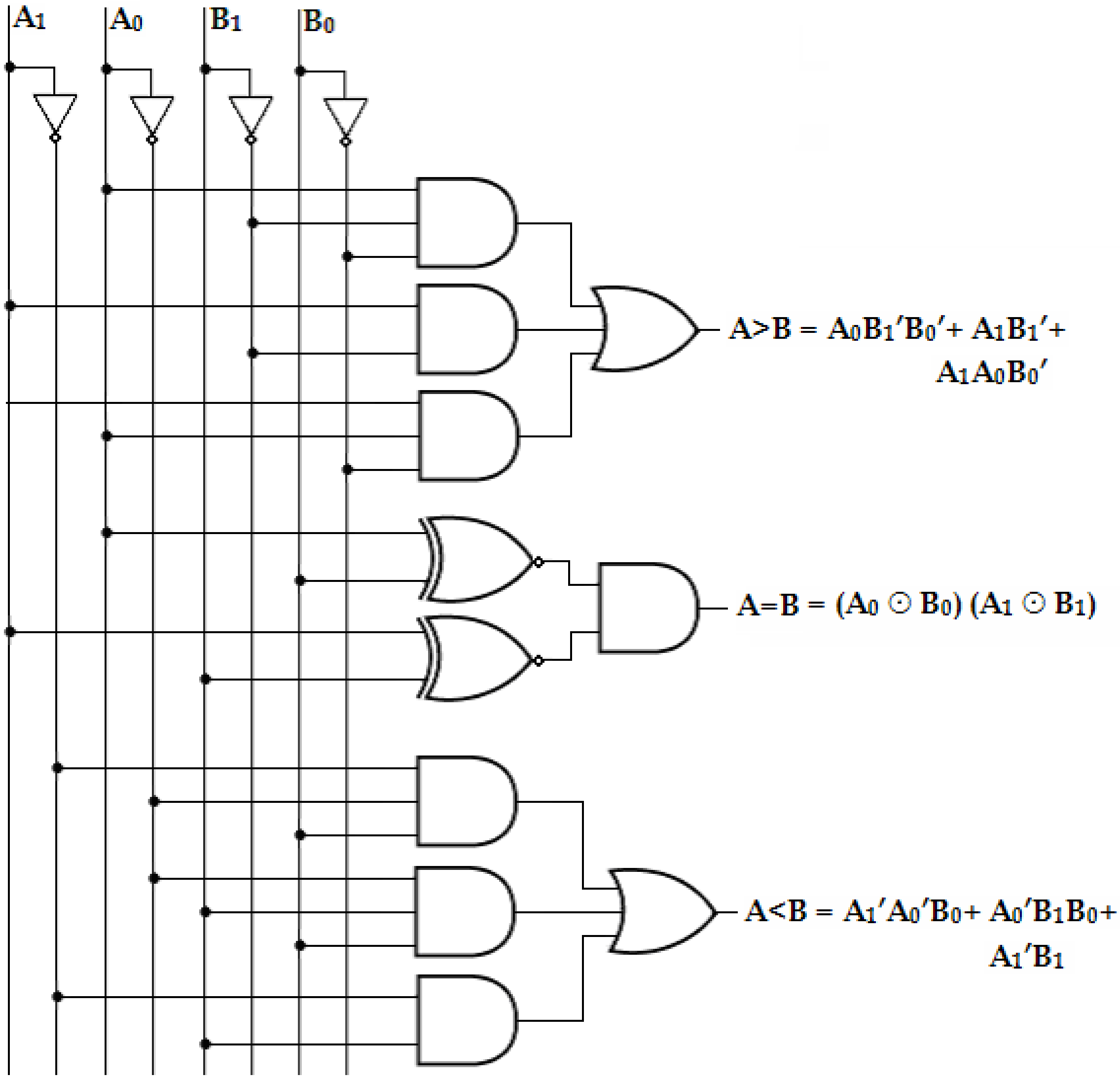
$$A=B = A_1'A_0'B_1'B_0' + A_1'A_0B_1'B_0 + A_1A_0B_1B_0 + A_1A_0'B_1B_0'$$
$$= A_1'B_1'(A_0'B_0' + A_0B_0) + A_1B_1(A_0B_0 + A_0'B_0')$$
$$= (A_0 \odot B_0)(A_1 \odot B_1)$$

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:



****

4 bit magnitude comparator:

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

Input

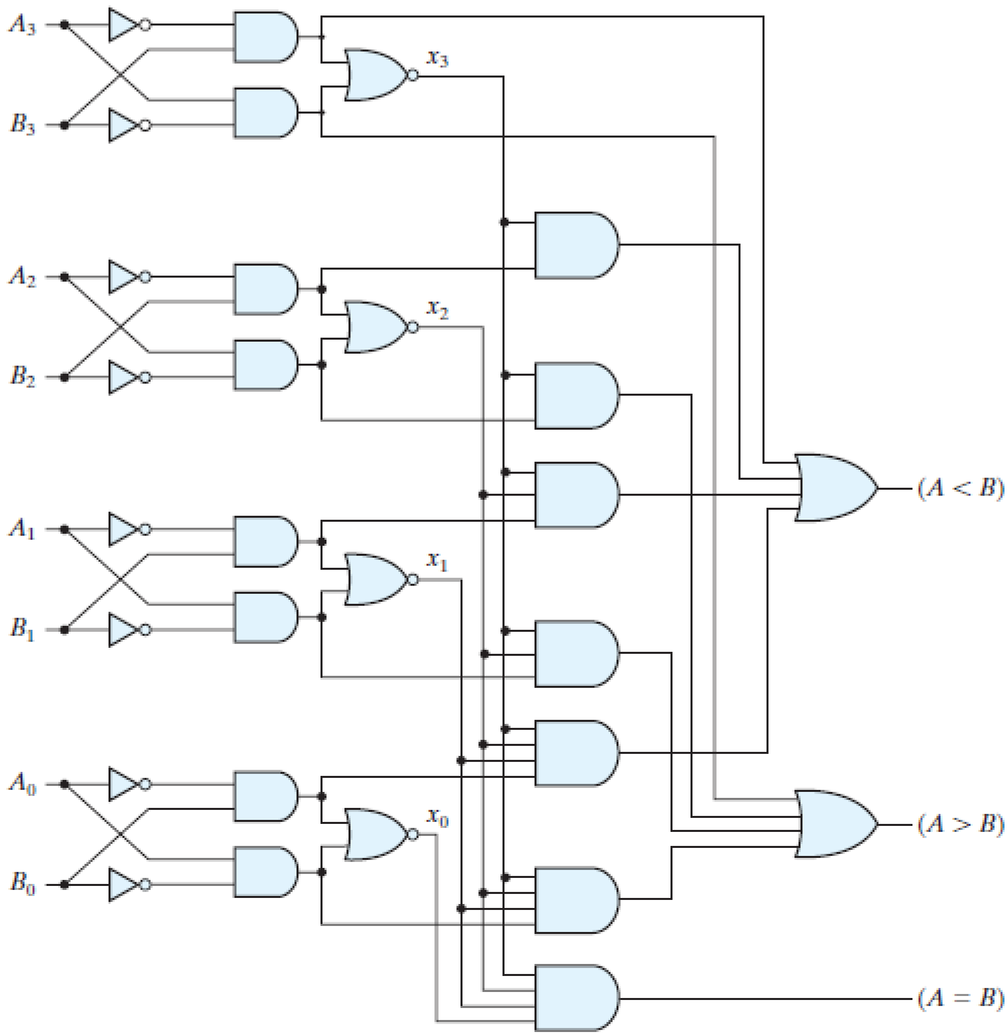
$$A = A_3\,A_2\,A_1\,A_0$$
$$B = B_3\,B_2\,B_1\,B_0$$

Function Equation

$$(A = B) = x_3x_2x_1x_0$$

$$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$

$$(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1' + x_3x_2x_1A'n_0B_0'$$



Four-bit magnitude comparator

****

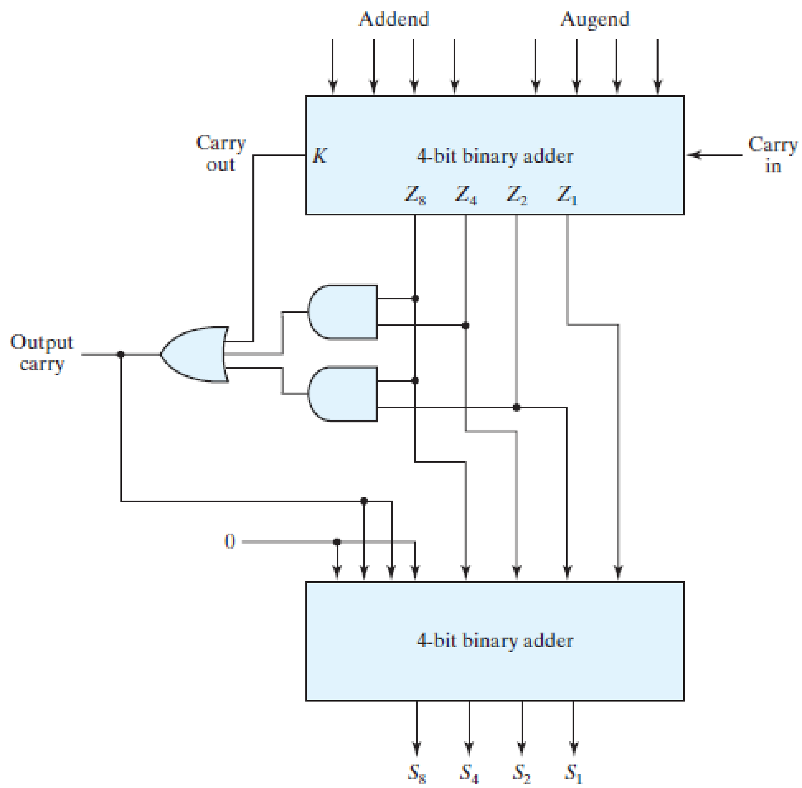
BCD Adder:

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

$$C = K + Z_8Z_4 + Z_8Z_2$$

- 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.

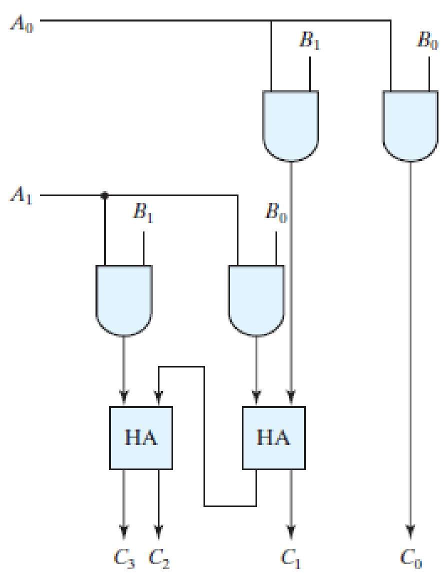
****

Binary Multiplier:

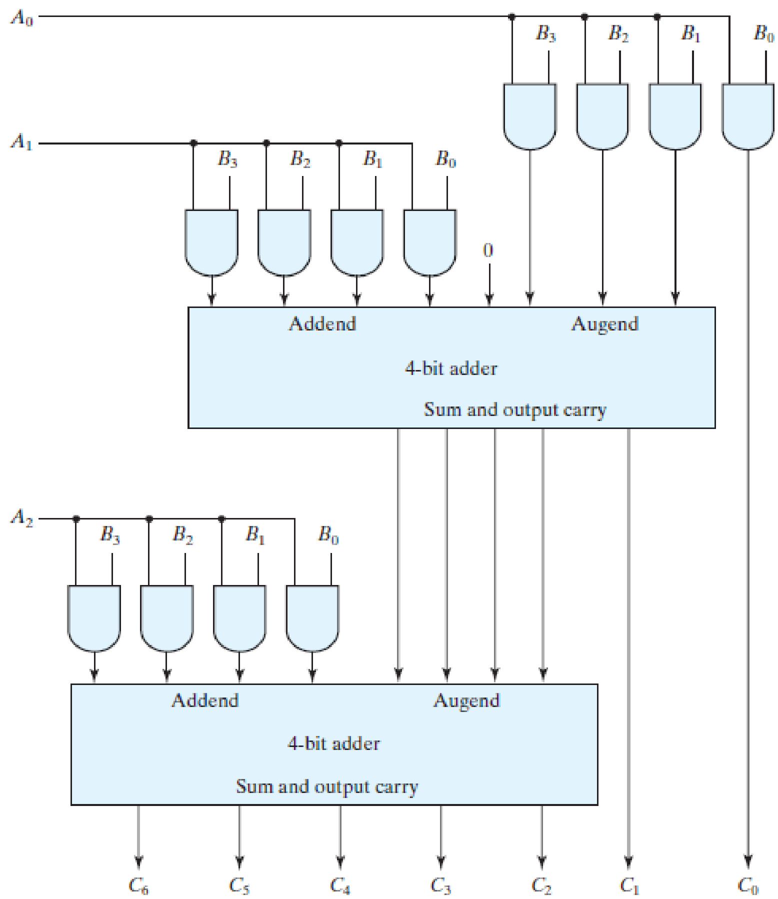
Explain about binary Multiplier.

- 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.

		B_1	B_0
		A_1	A_0
		A_0B_1	A_0B_0
	A_1B_1	A_1B_0	
C_3	C_2	C_1	C_0



- 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.



A_0
B_3
B_2
B_1
B_0
A_1
B_3
B_2
B_1
B_0
0
Addend
Augend
4-bit adder
Sum and output carry
A_2
B_3
B_2
B_1
B_0
Addend
Augend
4-bit adder
Sum and output carry
C_6
C_5
C_4
C_3
C_2
C_1
C_0

****

CODE CONVERSION

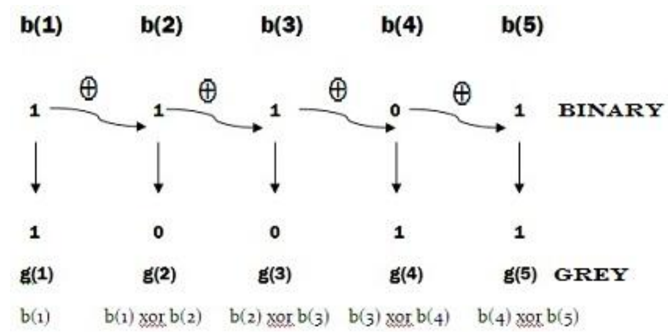
Design a binary to gray converter. ***(Nov-2009)(Nov 2017)***

Binary to Grayconverter

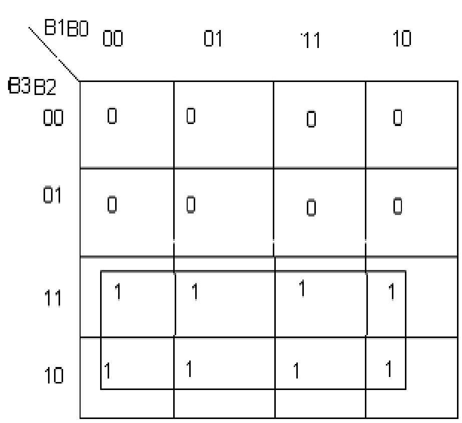
Gray code is unit distance code.
Input code: Binary [B_3 B_2 B_1 B_0]
output code: Gray [G_3 G_2 G_1 G_0]

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

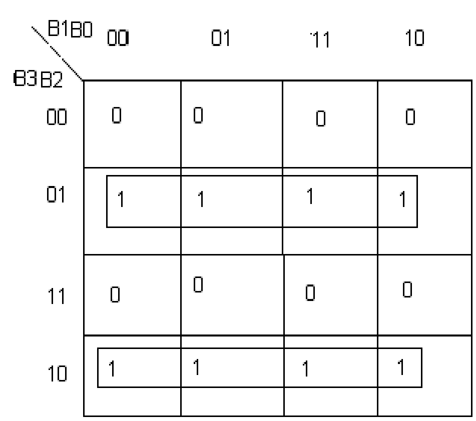


K-MAP FORG3:



G3=B3

K-MAP FORG2:



G2=B3’B2+B3B2’=B3 ⊕B2

K-MAP FORG1:

B3B2 \ B1B0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	1	1

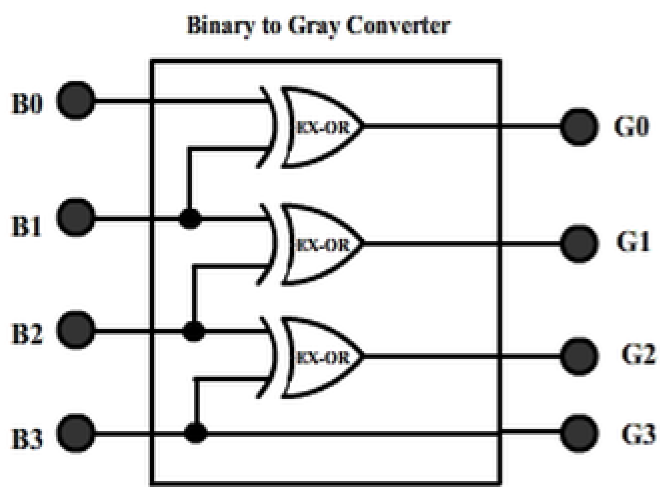
K-MAP FORG0:

B3B2 \ B1B0	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

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

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

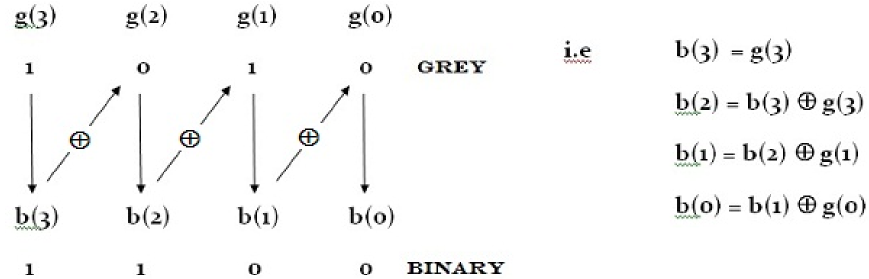
Logic diagram:



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]

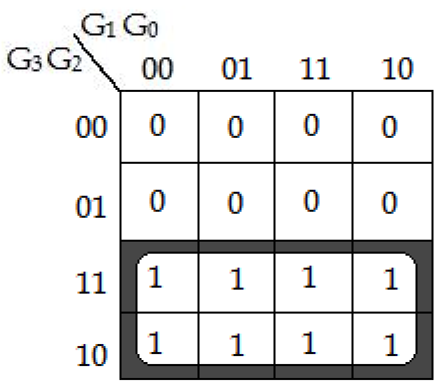


Truth Table:

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

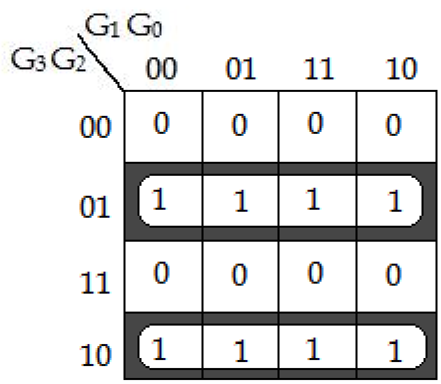
K-Map:

For B_3



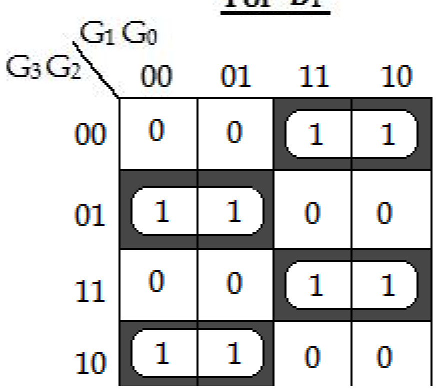
$B_3 = G_3$

For B_2

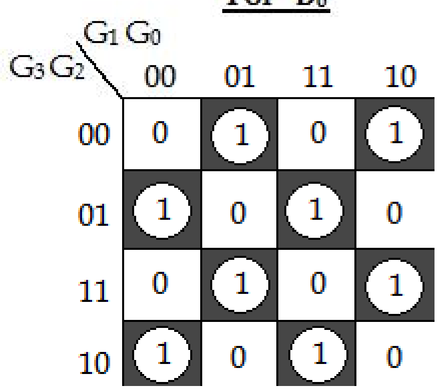


$B_2 = G_3'G_2 + G_3G_2'$
$= G_3 \oplus G_2$

For B_1



For B_0



From the above K-map,

$B_3 = G_3$

$B_2 = G_3'G_2 + G_3G_2'$

$B_2 = G_3 \oplus G_2$

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

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

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

$B_1 = G_3 \oplus G_2 \oplus G_1$

$B_0 = G_3'G_2' G_1'G_0 + G_3'G_2\_G_1G_0' + G_3G_2G_1\_G_0 + G_3G_2G_1 G_0' + G_3'G_2G_1'G_0' +$
$G_3G_2\_G_1'G_0' + G_3'G_2G_1G_0 + G_3G_2\_G_1 G_0.$

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

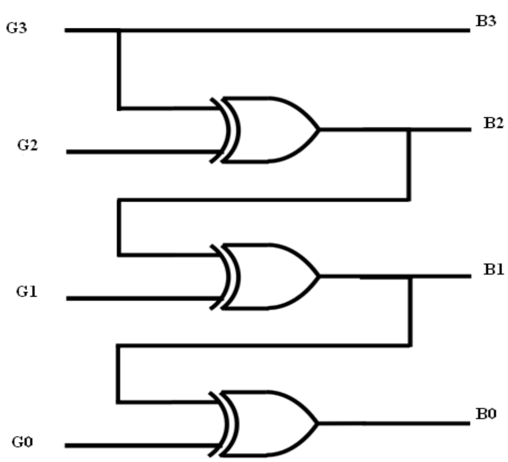
$= G_3'G_2' (G_0 \oplus G_1) + G_3G_2 (G_0 \oplus G_1) + G_1'G_0' (G_2 \oplus G_3) + G_1G_0 (G_2 \oplus G_3).$

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

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

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

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 3to 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

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	x	x	x	x
10	1	1	x	x

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

For E_2

B_3B_2 \ B_1B_0	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	x	x	x	x
10	0	1	x	x

$E_2 = B_2B_1'B_0' + B_2'\,(B_0 + B_1)$

For E_1

B_3B_2 \ B_1B_0	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	x	x	x	x
10	1	0	x	x

$E_1 = B_1'B_0' + B_1B_0$

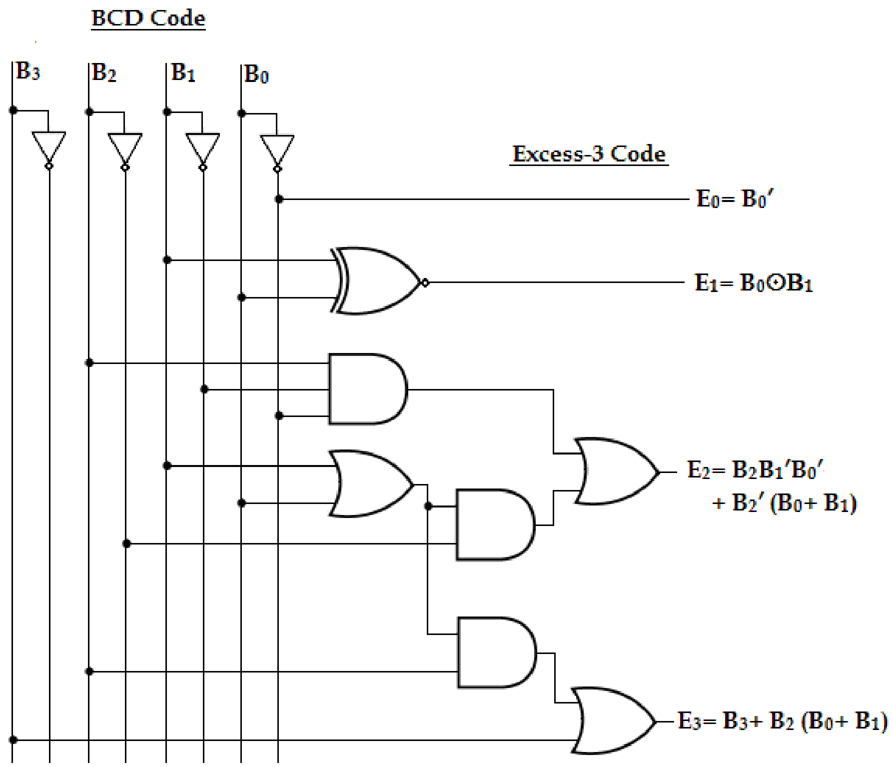
$= B_1 \odot B_0$

For E_0

B_3B_2 \ B_1B_0	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	x	x	x	x
10	1	0	x	x

$E_0 = B_0'$

Logic Diagram



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 B_0

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 B_1

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

$$B_1 = \overline{E}_1E_0 + E_1\overline{E}_0 = E_1 \oplus E_0$$

For B_2

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

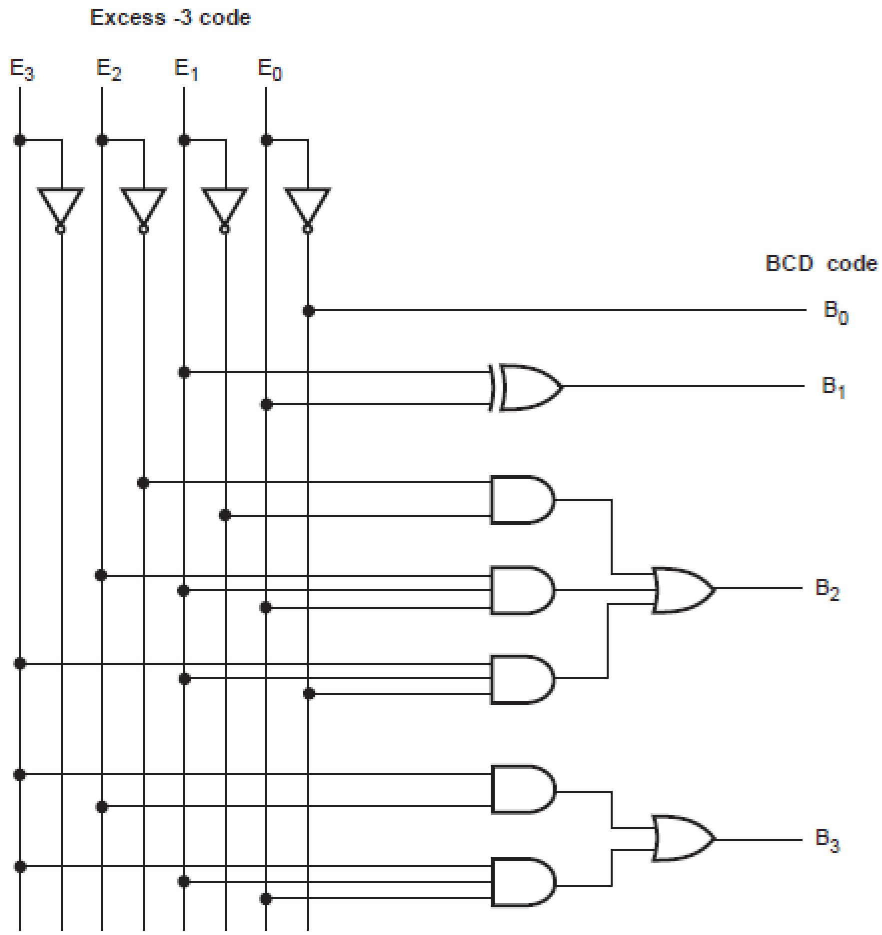
$B_2 = \overline{E}_2\overline{E}_1 + E_2E_1E_0 + E_3E_1\overline{E}_0$

For B_3

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



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

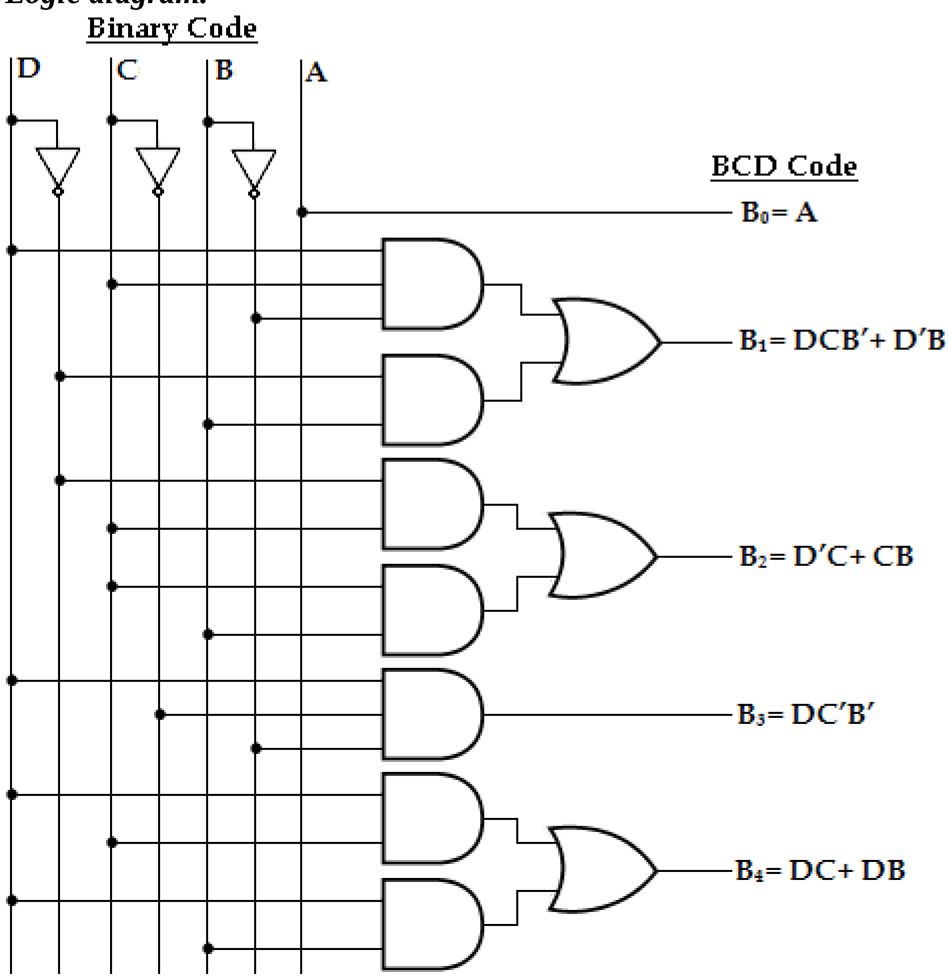
$B_3 = DC'B'$

For B_4

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:



****

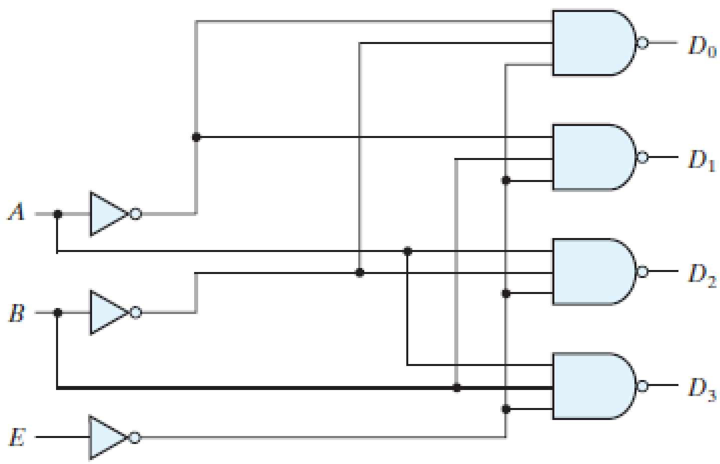
DECODERS AND ENCODERS

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:



(a) Logic diagram

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

(b) Truth table

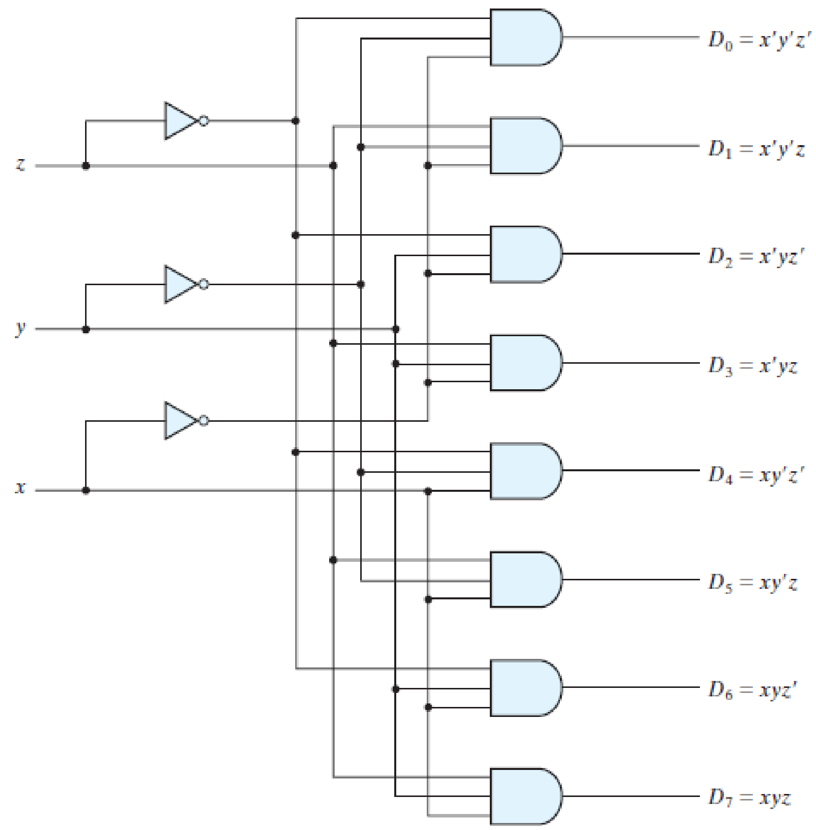
3 to 8 Decoder:

Design 3 to 8 line decoder with necessary diagram. ***May -10)***

Truth table:

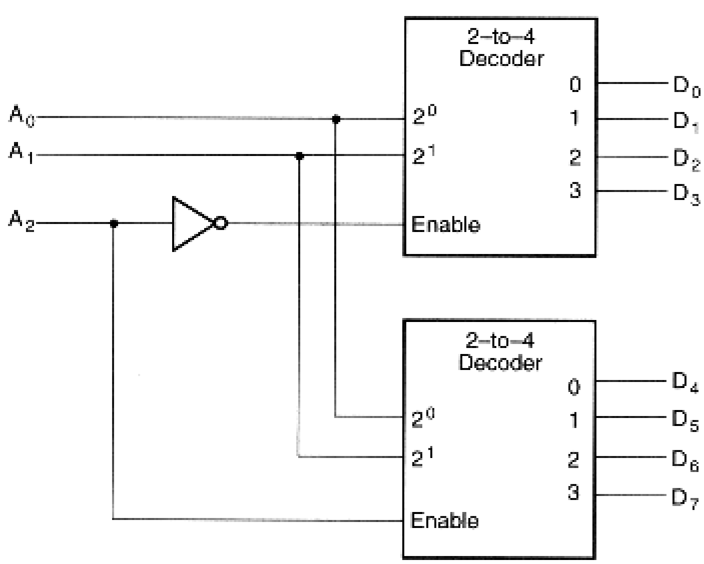
Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Logic diagram:



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.

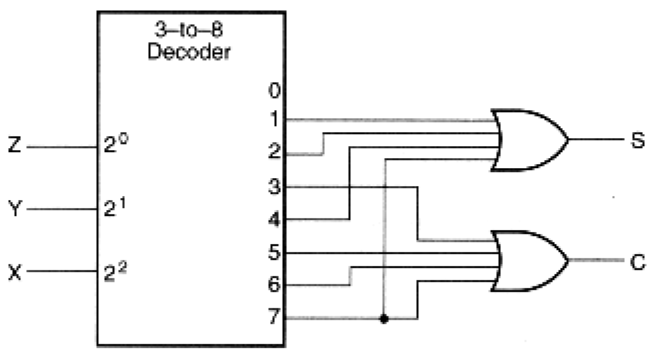


Implementation of Boolean function using decoder:

- 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.

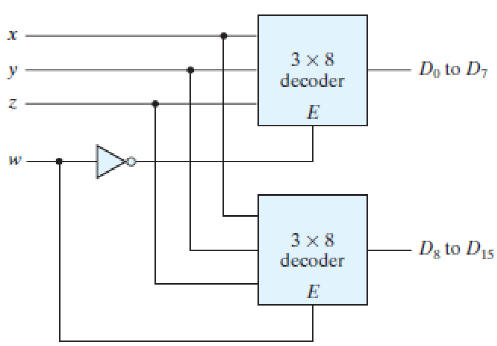


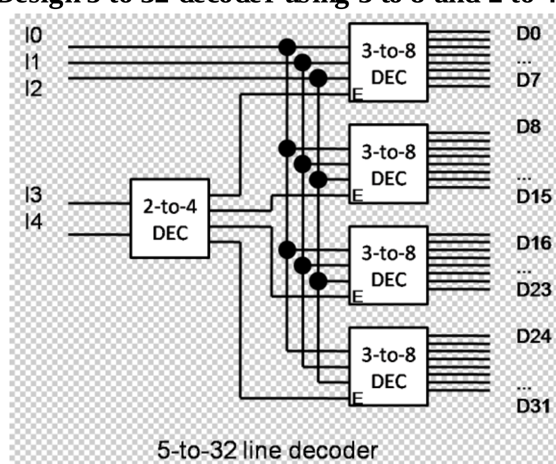
Solution :

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:**





****

BCD to seven segment decoder

Design a BCD to seven segment code converter. ***(May-06,10, Nov- 09)***

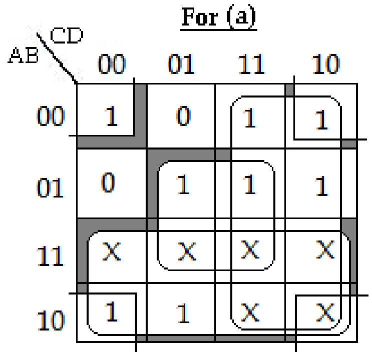
(a) Segment designation (b) Numeric designation for display

Truth table:

	BCD code				7-Segment code						
Digit	A	B	C	D	a	b	c	d	e	f	g
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

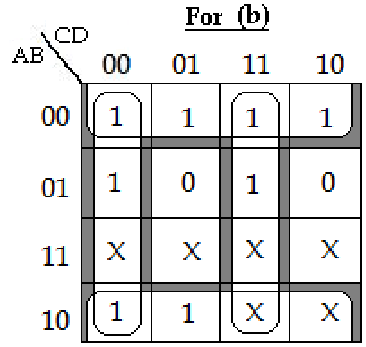
K-Map:

For (a)



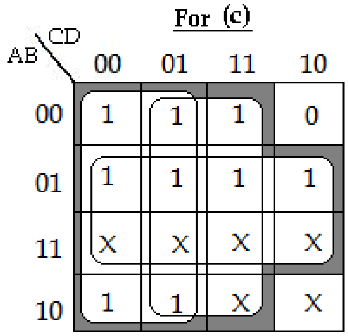
a= A+ C+ BD+ B'D'

For (b)



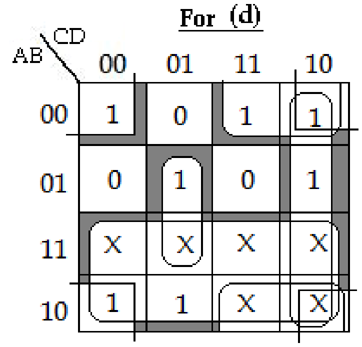
b= B'+ C'D'+ CD

For (c)



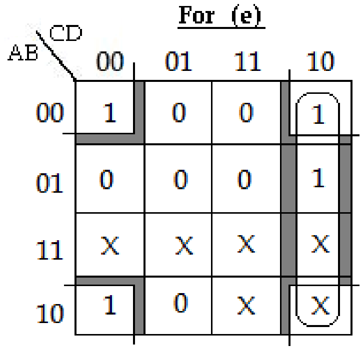
c= B+ C'+ D

For (d)



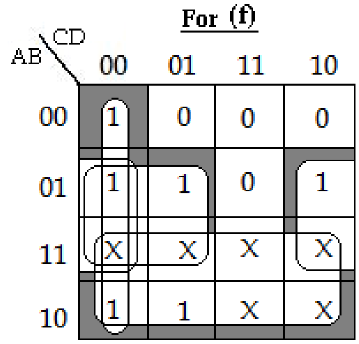
d= B'D'+ CD'+ BC'D+ B'C+ A

For (e)



e= B'D'+ CD'

For (f)



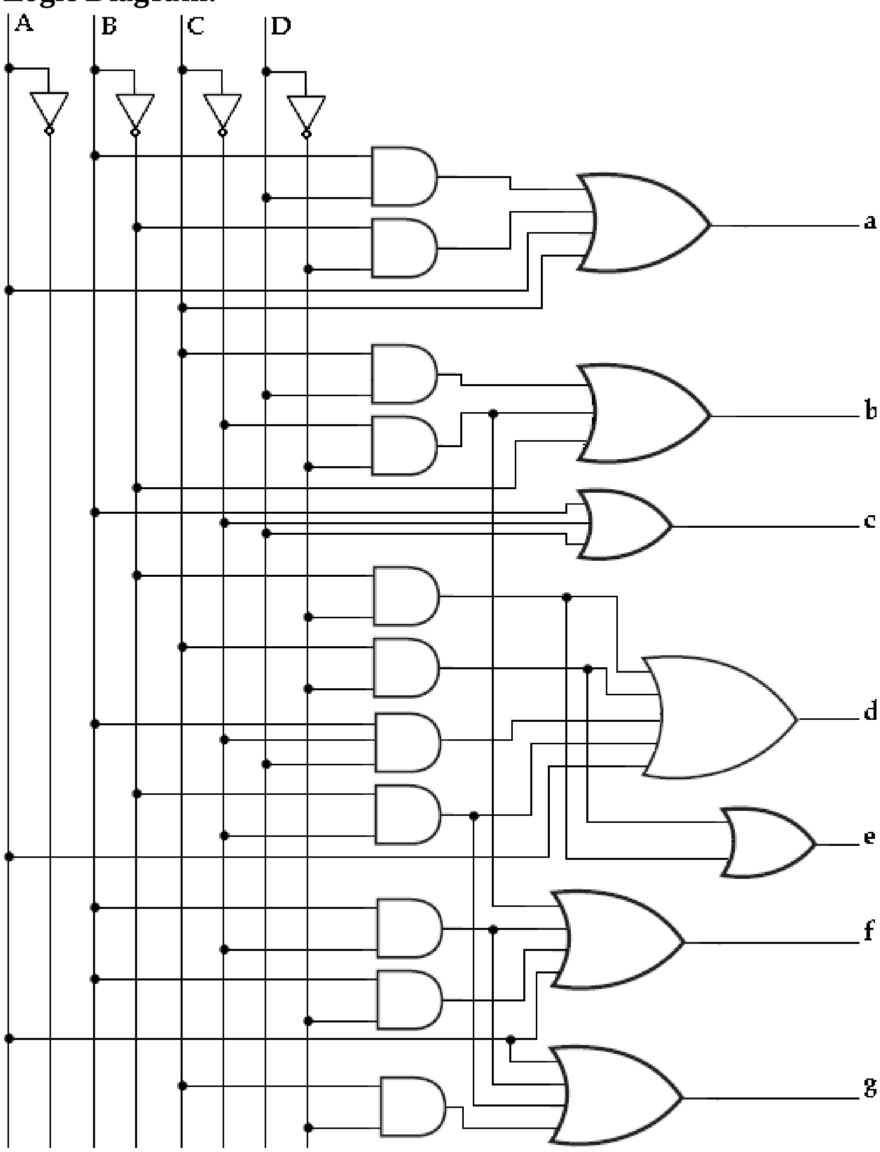
f= A+ C'D'+ BC'+ BD'

For (g)

AB \ CD	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	X	X	X	X
10	1	1	X	X

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

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.

****

Encoder:

Explain about encoders. (Nov 2018)

- 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:

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

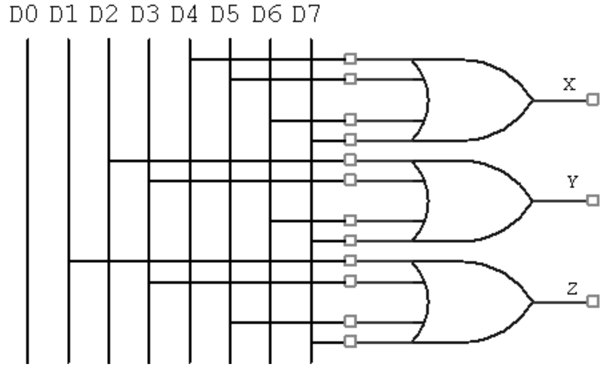
The encoder can be implemented with three OR gates.

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:



****

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 operationof 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:

<table>
<tr><th colspan="4">Inputs</th><th colspan="3">Outputs</th></tr>
<tr><th>D_0</th><th>D_1</th><th>D_2</th><th>D_3</th><th>x</th><th>y</th><th>V</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>×</td><td>×</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">1</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td rowspan="4">1</td><td rowspan="4">0</td><td rowspan="4">1</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td rowspan="7">1</td><td rowspan="7">1</td><td rowspan="7">1</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td></td><td></td><td></td></tr>
</table>

K-Map:

For x

D_0D_1 \ D_2D_3	00	01	11	10
00	x	1	1	1
01	0	1	1	1
11	0	1	1	1
10	0	1	1	1

$x = D_2 + D_3$

For y

D_0D_1 \ D_2D_3	00	01	11	10
00	x	1	1	0
01	1	1	1	0
11	1	1	1	0
10	0	1	1	0

$y = D_3 + D_1D_2$

For V

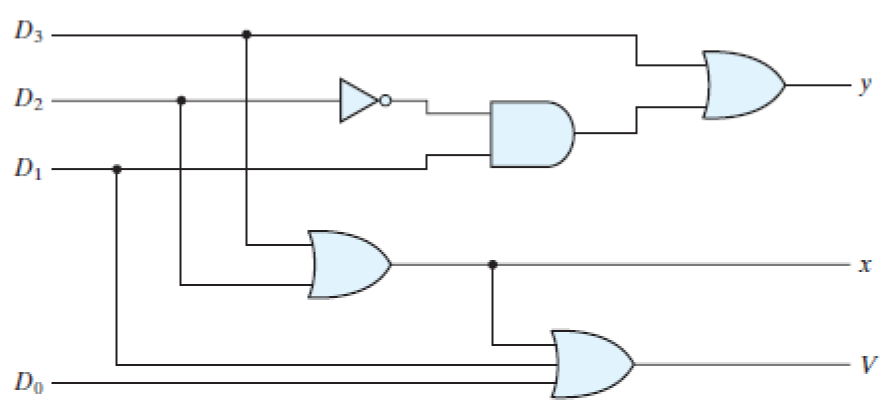
D_0D_1 \ D_2D_3	00	01	11	10
00	0	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$V = D_0 + D_1 + D_2 + D_3$

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

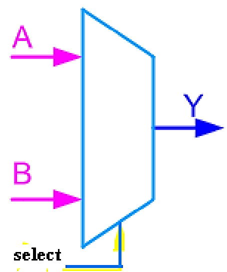
Multiplexer: (MUX)

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.

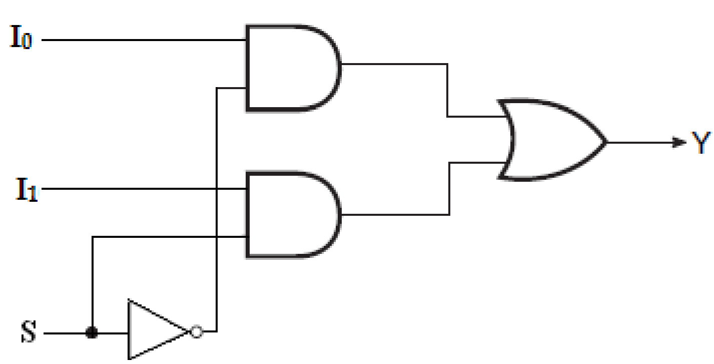


- To derive the gate level implementation of 2:1 mux we need to have truth table as shown in figure. And once we have the truth table, we can draw the K-map as shown in figure for all the cases when Y is equal to '1'.

Truth table:

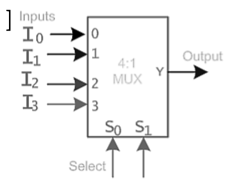
S	Y
0	I_0
1	I_1

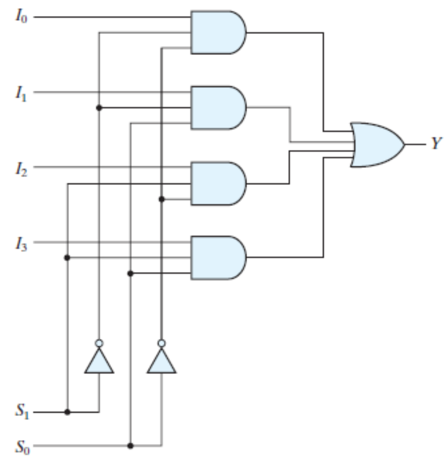
Logic Diagram:



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

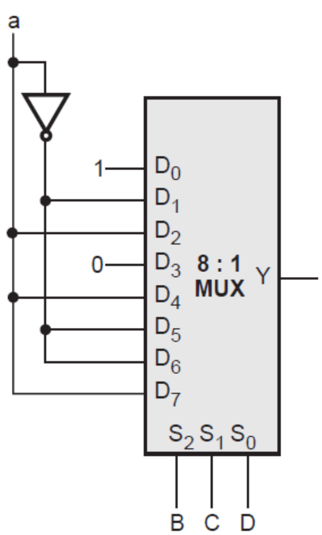
<u>Problems :</u>

Example: Implement the Boolean expression using MUX

F(A,B,C,D) = ∑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]

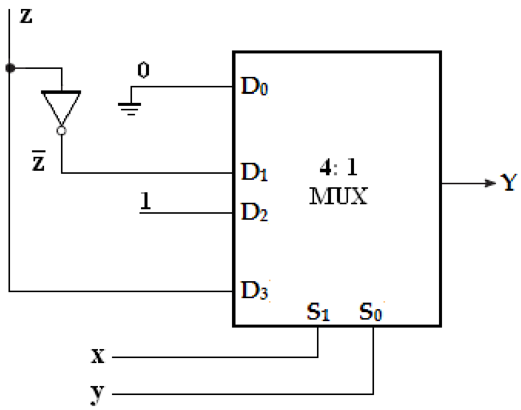
F (x, y, z) = Σm (1, 2, 6, 7)

Solution:

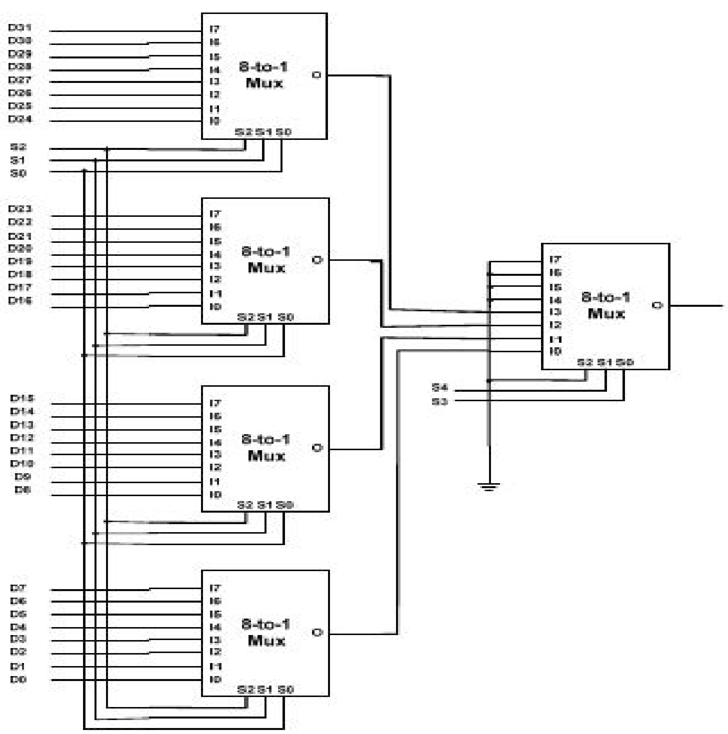
Implementation table:

	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:



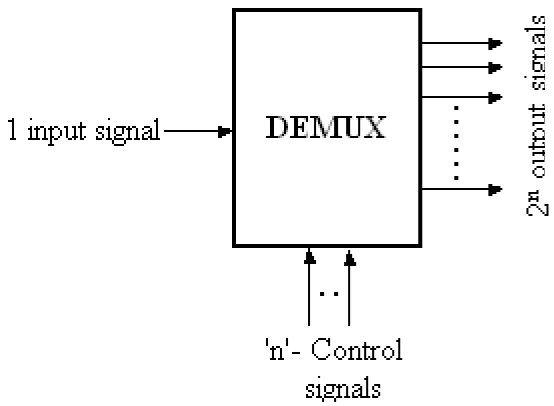
Example: 32:1 Multiplexer using 8:1 Mux (Nov 2018) (Apr – 2019)



DEMULTIPLEXERS:

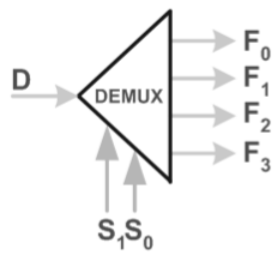
Explain about demultiplexers.

- 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

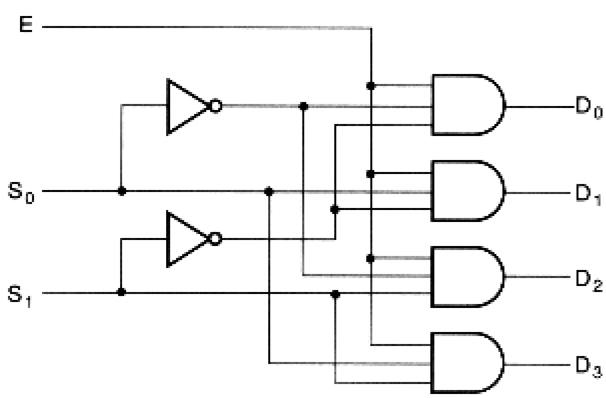


Truth table

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

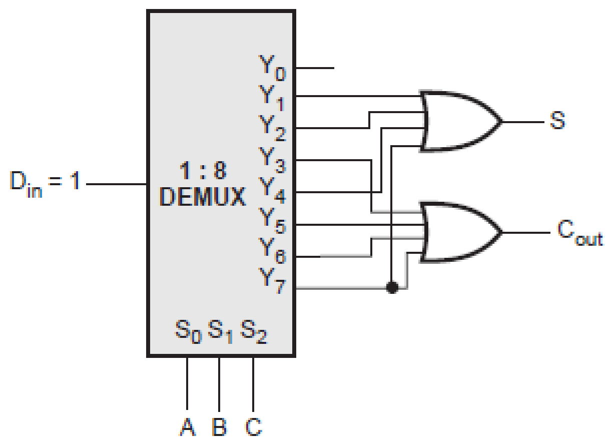
Inputs			Outputs	
A	B	C_{in}	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Step 2 : For full adder

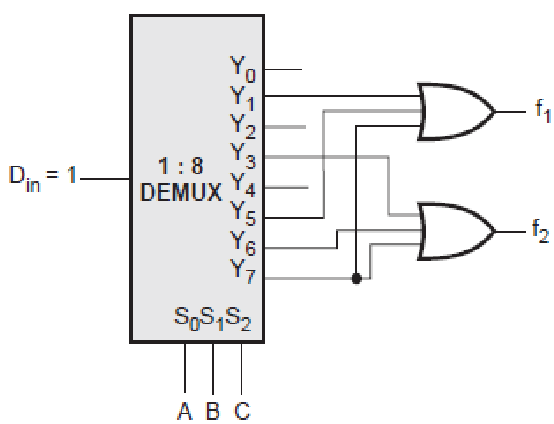
Carry $= C_{out} = \sum m\ (3, 5, 6, 7)$

and 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:

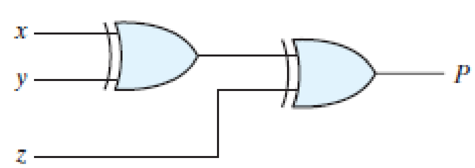
- A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even. The message, including the parity bit, is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond with the one transmitted.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



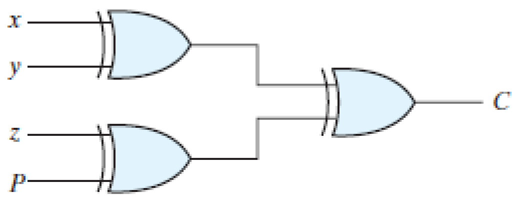
4-bit Even parity checker:

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:



INTRODUCTION TO HDL

- In electronics, a **hardware description language or HDL** is any language from a class of computer languages and/or programming languages for formal description of digital logic and electronic circuits.
- HDLs are used to write executable specifications of some piece of hardware.
- A simulation program, designed to implement the underlying semantics of the language statements, coupled with simulating the progress of time, provides the hardware designer with the ability to model a piece of hardware before it is created physically.
- *Logic synthesis* is the process of deriving a list of components and their interconnection (called net list) from the model of a digital system.
- *Logic Simulation* is the representation of the structure and behavior of a digital logic synthesis through the use of a computer.
- The standard HDLs that supported by IEEE.
  - ✓ VHDL (very High Speed Integrated Circuit HDL)
  - ✓ Verilog HDL

HDL MODELS OF COMBINATIONAL CIRCUITS

The Verilog HDL model of a combinational circuit can be described in any one of the following modeling styles,

✓Gate level modeling-using instantiations of predefined and user defined primitive gates.
✓Dataflow modeling using continuous assignment with the keyword **assign**.
✓Behavioral modeling using procedural assignment statements with the keyword **always**.

Gate level modeling

In this type, a circuit is specified by its logic gates and their interconnections. Gate level modeling provides a textual description of a schematic diagram. The verilog HDL includes 12basic gates as predefined primitives. They are and, nand, or, nor, xor, xnor, not &buf.

HDL

```
// Gate-level description of two-to-four-line decoder
// Refer to Fig. 4.19 with symbol E replaced by enable, for clarity.

module decoder_2x4_gates (D, A, B, enable);
  output      [0: 3]     D;
  input                  A, B;
  input                  enable;
  wire                   A_not, B_not, enable_not;

  not
    G1  (A_not, A),
    G2  (B_not, B),
    G3  (enable_not, enable);
  nand
    G4  (D[0], A_not, B_not, enable_not),
    G5  (D[1], A_not, B, enable_not),
    G6  (D[2], A, B_not, enable_not),
    G7  (D[3], A, B, enable_not);
endmodule
```

Data flow modeling

Data flow modeling of combinational logic uses a number of operators that act on operands to produce desired results. Verilog HDL provides about 30 different operators. Data flow modeling uses continuous assignments and the keyword **assign**. A continuous assignment is a statement that assigns a value to a net. The data type family **net** is used to represent a physical connection between circuit elements.

HDL for2-to-4 line decoder

Some Verilog HDL Operators

Symbol	Operation
+	binary addition
−	binary subtraction
&	bitwise AND
|	bitwise OR
^	bitwise XOR
~	bitwise NOT
==	equality
>	greater than
<	less than
{ }	concatenation
? :	conditional

```
module decoder_2x4_df (
 output     [0: 3]     D,
 input                 A, B,
                       enable
);

 assign     D[0] = ~(~A & ~B & ~enable),
            D[1] = ~(~A & B & ~enable),
            D[2] = ~(A & ~B & ~enable),
            D[3] = ~(A & B & ~enable);
endmodule
```

Behavioral modeling

- Behavioral modeling represents digital circuits at a functional and algorithmic level. It is used mostly to describe sequential circuits, but can also be used to describe combinational circuits.
- Behavioral descriptions use the keyword **always,** followed by an optional event control expression and a list of procedural assignment statements.

```
// Behavioral description of two-to-one-line multiplexer

module mux_2x1_beh (m_out, A, B, select);
 output     m_out;
 input      A, B, select;
 reg        m_out;

 always     @(A or B or select)
  if (select == 1) m_out = A;
  else m_out = B;
 endmodule
```

UNIT III

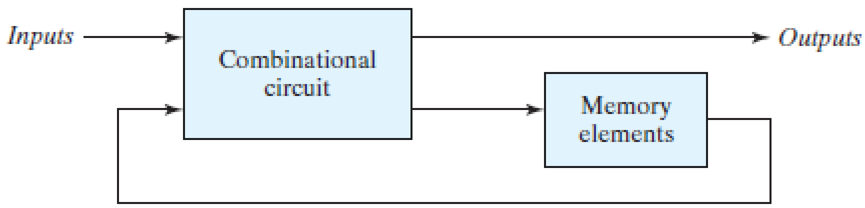
SYNCHRONOUS SEQUENTIAL LOGIC

Sequential Circuits - Storage Elements: Latches , Flip-Flops - Analysis of Clocked Sequential Circuits - State Reduction and Assignment - Design Procedure - Registers and Counters - HDL Models of Sequential Circuits

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, and the circuit behavior must be specified by a time sequence of inputs and internal states.



Types of sequential circuits:

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

1. **Synchronous sequential circuit:**
   It is a system whose behaviorcan be defined from the knowledge of its signals at discrete instants of time.
2. **Asynchronous sequential circuits:**
   The behaviorof an asynchronous sequential circuit depends upon the input signals at any instant of timeand the order in which the inputs change. The storage elements commonly used in asynchronoussequential circuits are time-delay devices.

LATCHES AND FLIP FLOPS

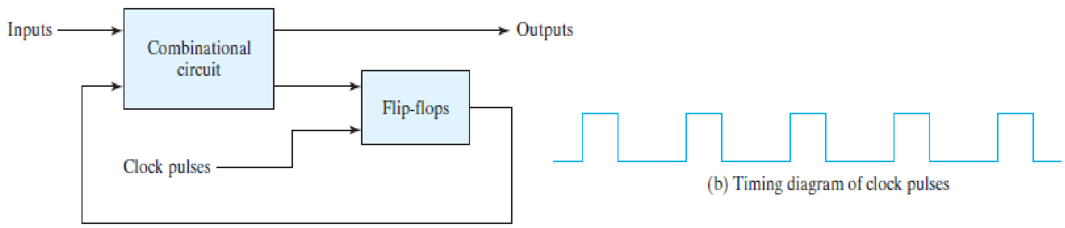
Flip-Flop:

- *The storage elements (memory) used in clocked sequential circuits are called flipflops.* 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.

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

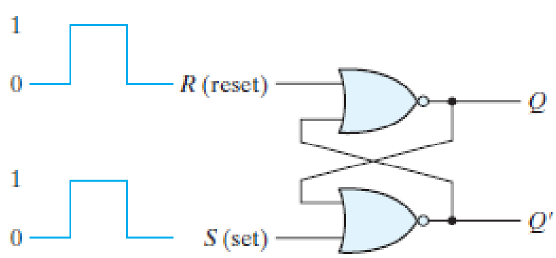
(b) Timing diagram of clock pulses

Synchronous clocked sequential circuit

SR Latch: Using NOR gate

Realize SR Latch using NOR and NAND gates and explain its operation.

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

SR latch with NOR gates

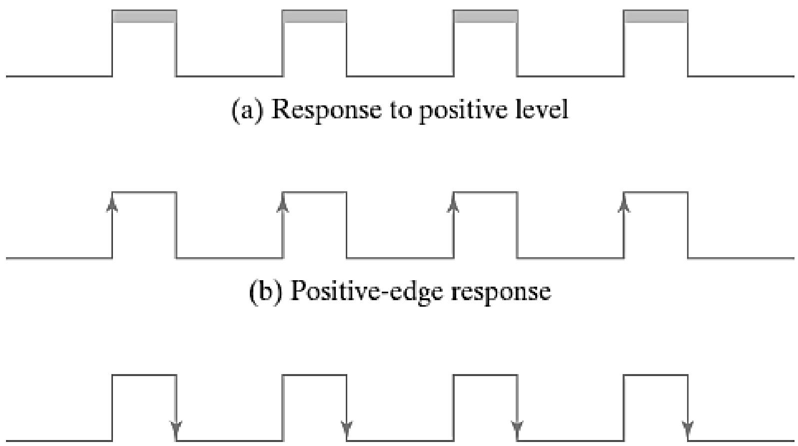
- The latch has two useful states. When output Q = 1 and Q'= 0, the latch is said to be in the *set state* . When Q = 0 and Q' = 1, it is in the *reset state* . Outputs Q and Q' are normally the complement of each other.
- However, when both inputs are equal to 1 at the same time, a condition in which 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.

FLIP FLOPS

Triggering of Flip Flops:

Explain about triggering of flip flops in detail.

- 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:

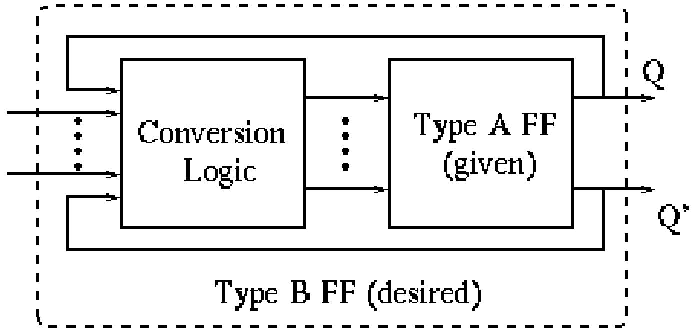
- A clock pulse goes through two transitions: from 0 to 1 and the return from 1 to 0.
- As shown in above Fig (b) and (c)., the positive transition is defined as the positive edge and the negative transition as the negative edge.

****

Explain the operation of flipflops.(Nov 2017)

FLIP FLOP CONVERSIONS

The purpose is to convert a given type A FF to a desired type B FF using some conversion logic.



The key here is to use the excitation table, which shows the necessary triggering signal (S,R, J,K, D and T) for a desired flipflop state transition $Q_t \rightarrow Q_{t+1}$:

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

1. SR Flip Flop to JK Flip Flop

The truth tables for the flip flop conversion are given below. The present state is represented by Qp and Qp+1 is the next state to be obtained when the J and K inputs are applied.

For two inputs J and K, there will be eight possible combinations. For each combination of J, K and Qp, the corresponding Qp+1 states are found. Qp+1 simply suggests the future values to be obtained by the JK flip flop after the value of Qp.

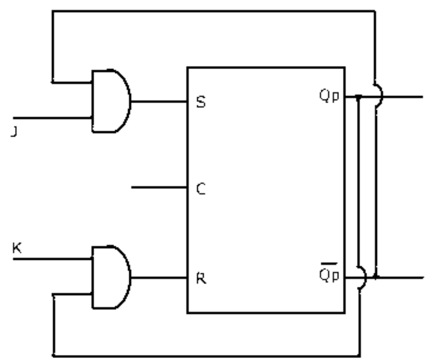
The table is then completed by writing the values of S and R required to get each Qp+1 from the corresponding Qp. That is, the values of S and R that are required to change the state of the flip flop from Qp to Qp+1 are written.

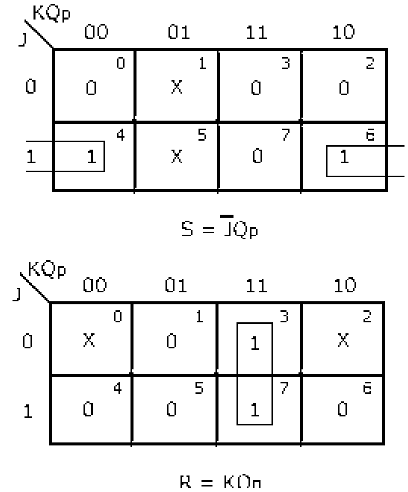
S-R Flip Flop to J-K Flip Flop

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





2.JK Flip Flop to SR Flip Flop

This will be the reverse process of the above explained conversion. S and R will be the external inputs to J and K. As shown in the logic diagram below, J and K will be the outputs of the combinational circuit. Thus, the values of J and K have to be obtained in terms of S, R and Qp. The logic diagram is shown below.

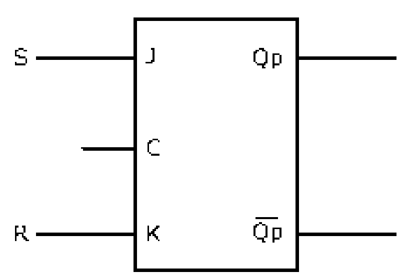
A conversion table is to be written using S, R, Qp, Qp+1, J and K. For two inputs, S and R, eight combinations are made. For each combination, the corresponding Qp+1 outputs are found. The outputs for the combinations of S=1 and R=1 are not permitted for an SR flip flop. Thus the outputs are considered invalid and the J and K values are taken as “don’t cares”.

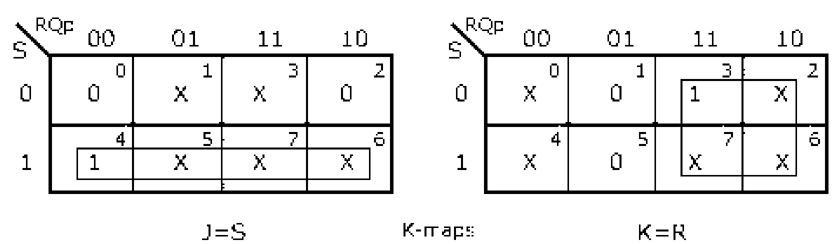
J-K Flip Flop to S-R Flip Flop

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





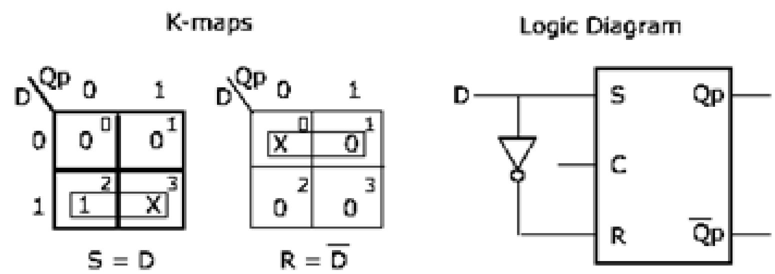
3.SR Flip Flop to D Flip Flop

As shown in the figure, S and R are the actual inputs of the flip flop and D is the external input of the flip flop. The four combinations, the logic diagram, conversion table, and the K-map for S and R in terms of D and Qp are shown below.

S-R Flip Flop to D Flip Flop

Conversion Table

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



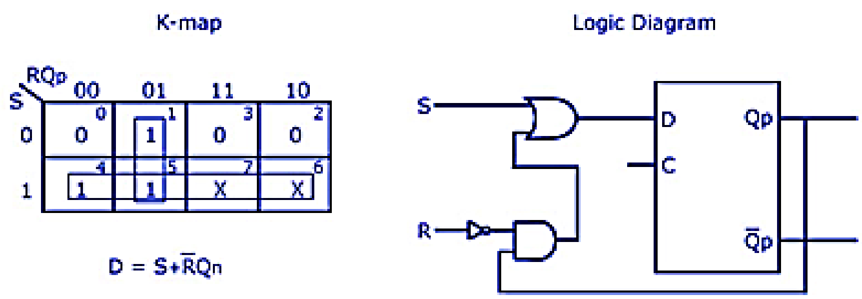
4.D Flip Flop to SR Flip Flop

D is the actual input of the flip flop and S and R are the external inputs. Eight possible combinations are achieved from the external inputs S, R and Qp. But, since the combination of S=1 and R=1 are invalid, the values of Qp+1 and D are considered as “don’t cares”. The logic diagram showing the conversion from D to SR, and the K-map for D in terms of S, R and Qp are shown below.

D Flip Flop to S-R Flip Flop

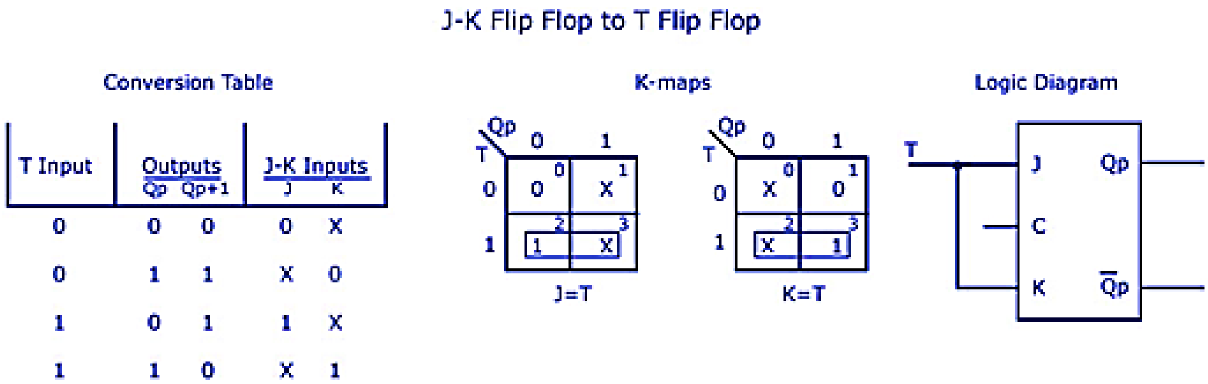
Conversion Table

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



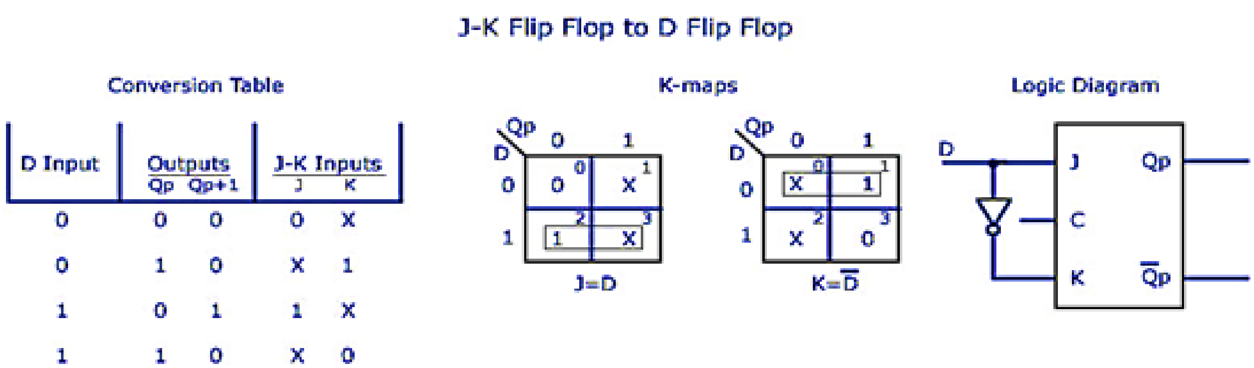
5.JK Flip Flop to T Flip Flop

J and K are the actual inputs of the flip flop and T is taken as the external input for conversion. Four combinations are produced with T and Qp. J and K are expressed in terms of T and Qp. The conversion table, K-maps, and the logic diagram are given below.



6.JK Flip Flop to D Flip Flop

D is the external input and J and K are the actual inputs of the flip flop. D and Qp make four combinations. J and K are expressed in terms of D and Qp. The four combination conversion table, the K-maps for J and K in terms of D and Qp, and the logic diagram showing the conversion from JK to D are given below.



7.D Flip Flop to JK Flip Flop

AUQ: How will you convert a D flip-flop into JK flip-flop? (AUQ: Dec 2009,11,Apr 2017)

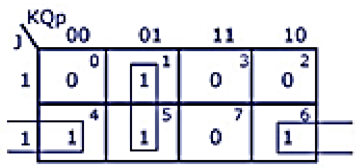
In this conversion, D is the actual input to the flip flop and J and K are the external inputs. J, K and Qp make eight possible combinations, as shown in the conversion table below. D is expressed in terms of J, K and Qp.The conversion table, the K-map for D in terms of J, K and Qp and the logic diagram showing the conversion from D to JK are given in the figure below.

D Flip Flop to J-K Flip Flop

Conversion Table

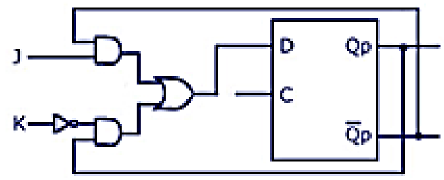
J-K Input		Outputs		D Input
J	K	Qp	Qp+1	
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

K-map



$D = J\overline{Q}p + \overline{K}Qp$

Logic Diagram



****

MEALY AND MOORE MODELS

Write short notes on Mealy and Moore models in sequential circuits.

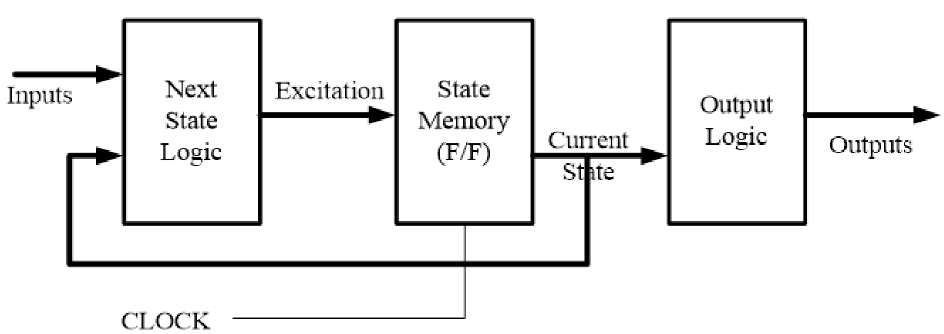
- In synchronous sequential circuit the outputs depend upon the order in which its input variables change and can be affected at discrete instances of time.

General Models:

- There are two models in sequential circuits. They are:
    1. Mealy model
    2. Moore model

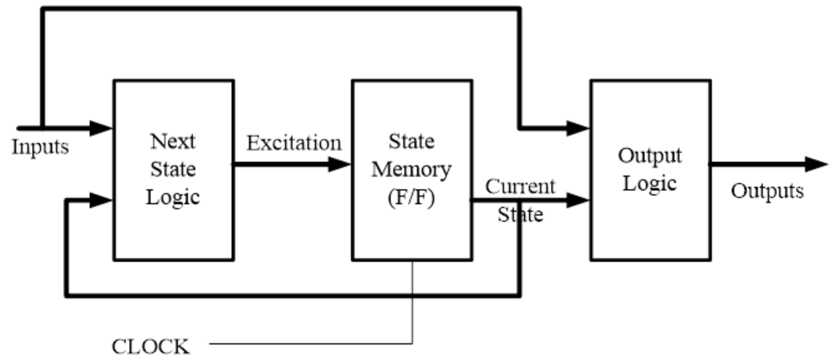
Moore machine:

- 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.



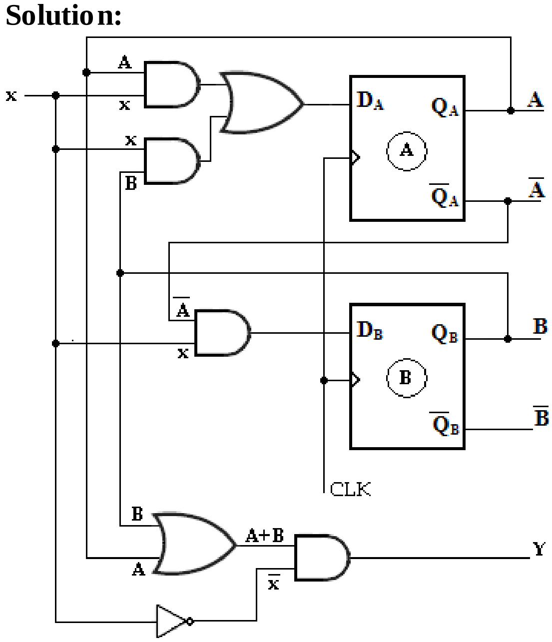
Difference between Moore model and Mealy model.

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'xand
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:



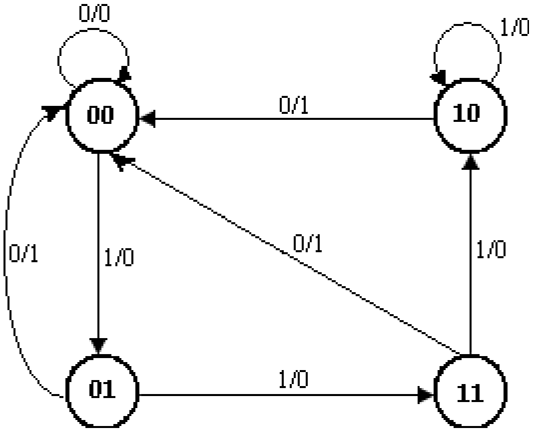
State table:

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:



****

****

COUNTERS

Counter:

- A counter is a register (group of Flip-Flop) capable of counting the number of clock pulse arriving at its clock input.
- A counter that follows the binary number sequence is called a binary counter.
- Counter are classified into two types,
  1. Asynchronous (Ripple) counters.
  2. Synchronous counters.

- In ripple counter, a flip- flop output transition serves as clock to next flip-flop.
    - With an asynchronous circuit, all the bits in the count do not all change at the same time.
- In a synchronous counter, all flip-flops receive common clock.
    - With a synchronous circuit, all the bits in the count change synchronously with the assertion of the clock
- A counter may count up or count down or count up and down depending on the input control.

Uses of Counters:

The most typical uses of counters are

- ✓ To count the number of times that a certain event takes place; the occurrence of event to be counted is represented by the input signal to the counter
- ✓ To control a fixed sequence of actions in a digital system
- ✓ To generate timing signals
- ✓ To generate clocks of different frequencies

Modulo 16 ripple /Asynchronous Up Counter

Explain the operation of a 4-bit binary ripple counter.

- The output of up-counter is incremented by one for each clock transition.
- A 4-bit asynchronous up-counter consists of 4JK Flip-Flops.
- The external clock signal is connected to the clock input of the first FlipFlop.
- The clock inputs of the remaining Flip-Flops are triggered by the Q output of the previous stage.
- We know that in JK Flip-Flop, if J=1 , K=1 and clock is triggered the past output will be complemented.
- Initially, the register is cleared, $Q_DQ_CQ_BQ_A$ =0000.
- During the first clock pulse, Flip-Flop A triggers, therefore Q_A=1, Q_B=Q_C=Q_D=0.

$$Q_DQ_CQ_BQ_A=0001$$

- At the second clock pulse FLipFlop A triggers, therefore Q_A changes from 1 to 0, which triggers FlipFlop B, therefore Q_B=1,Q_A=Q_C=Q_D=0

$$Q_DQ_CQ_BQ_A=0010$$

- At the third clock pulse FlipFlop A triggers, therefore Q_A changes from 0 to 1, This never triggers FlipFlop B because 0 to 1 transition gives a positive edge triggering,but here the FlipFlops are triggered only at negative edge(1 to 0 transition) therefore Q_A=Q_B=1, Q_C=Q_D=0.

$$Q_DQ_CQ_BQ_A=0011$$

- At the fourth clock pulse Flip-Flop A triggers, therefore Q_A changes from 1 to 0, This triggers FlipFlop B therefore Q_B changes from 1 to 0. The change in Q_B from 1 to 0 triggers C Flip-Flop,
- Therefore Q_C changes from 0 to 1. Therefore $Q_A=Q_B=Q_D=0$, $Q_C=1$.

$$Q_DQ_CQ_BQ_A=0100$$

Truth table:

CLK	Outputs			
	Q_D	Q_C	Q_B	Q_A
-	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Truth table for 4-bit asynchronous up-counter

Timing diagram:

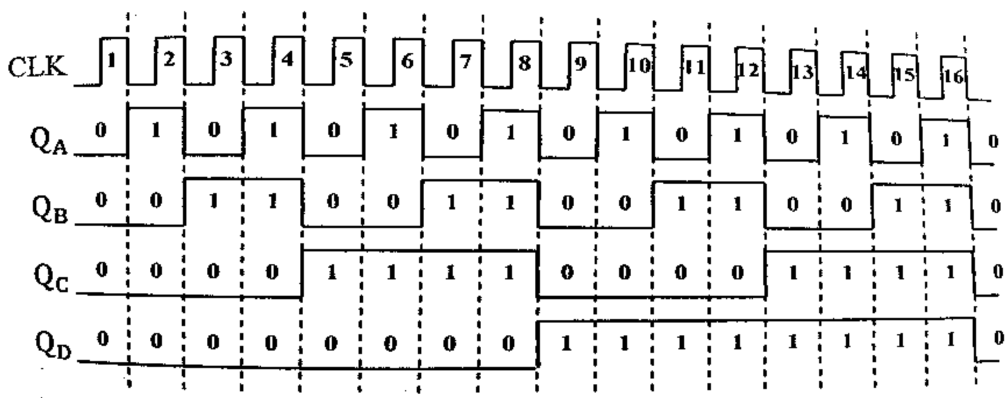


Figure 4.37 Timing diagram of 4-bit asynchronous up-counter.

****

Modulo 16 /4 bit Ripple Down counter/ Asynchronous Down counter

Explain about Modulo 16 /4 bit Ripple Down counter.

- The output of down-counter is decremented by one for each clock transition.
- A 4-bit asynchronous down-counter consists of 4JK Flip-Flops.
- The external clock signal is connected to the clock input of the first Flip-Flop.
- The clock inputs of the remaining Flip-Flops are triggered by the $\overline{Q}$ output of the previous stage.

- We know that in JK Flip-Flop, if J=1 , K=1 and clock is triggered the past output will be complemented.

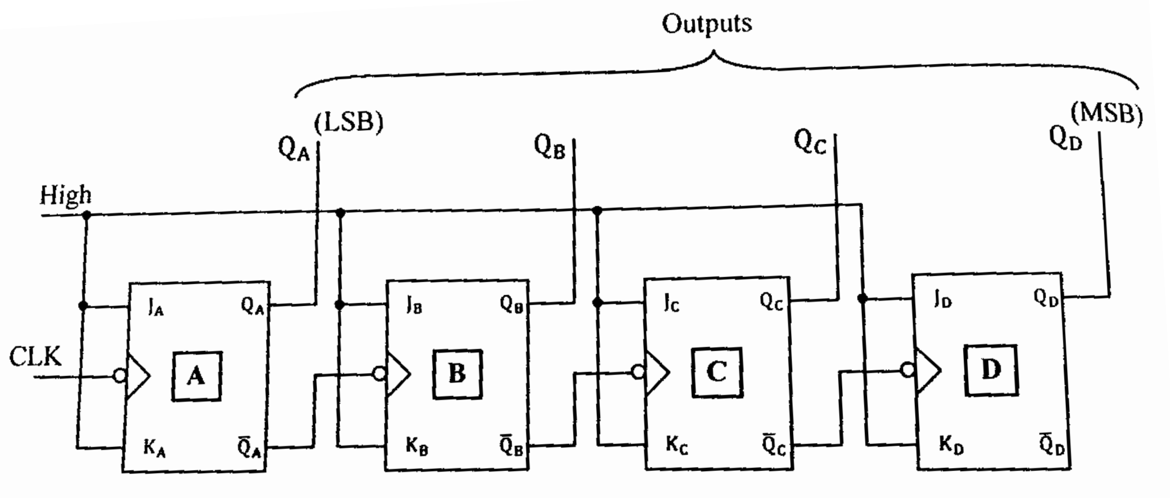


Figure Logic diagram of 4-bit asynchronous down-counter

- Initially, the register is cleared, $Q_DQ_CQ_BQ_A$ =0000.
- During the first clock pulse, Flip-Flop A triggers, therefore Q_A changes from 0 to 1 also $\overline{Q_A}$ changes from 1 to 0.This triggers Flip-Flop B, therefore Q_B changes from 0 to 1, also $\overline{Q_B}$ changes from 1 to 0which triggers Flip-FlopC. Hence Q_C changes from 0 to 1 and $\overline{Q_C}$ changes from 1 to 0, which further triggers, Flip-Flop D.

$$Q_DQ_CQ_BQ_A=1111$$

$$\overline{Q_D}\,\overline{Q_C}\,\overline{Q_B}\,\overline{Q_A}=0000$$

- During the second clock pulse Flip-Flop A triggers, therefore Q_A changes from 1 to 0 also $\overline{Q_A}$ changes from 0 to 1 which never triggers B Flip-Flop. Therefore C and D Flip-Flop are not triggered.

$$Q_DQ_CQ_BQ_A = 1110$$

- The same procedure repeats until the counter decrements upto 0000.

CLK	Outputs			
	Q_D	Q_C	Q_B	Q_A
-	0	0	0	0
1	1	1	1	1
2	1	1	1	0
3	1	1	0	1
4	1	1	0	0
5	1	0	1	1
6	1	0	1	0
7	1	0	0	1
8	1	0	0	0
9	0	1	1	1
10	0	1	1	0
11	0	1	0	1
12	0	1	0	0
13	0	0	1	1
14	0	0	1	0
15	0	0	0	1
16	0	0	0	0

Table .... Truth table for 4-bit asynchronous down-counter

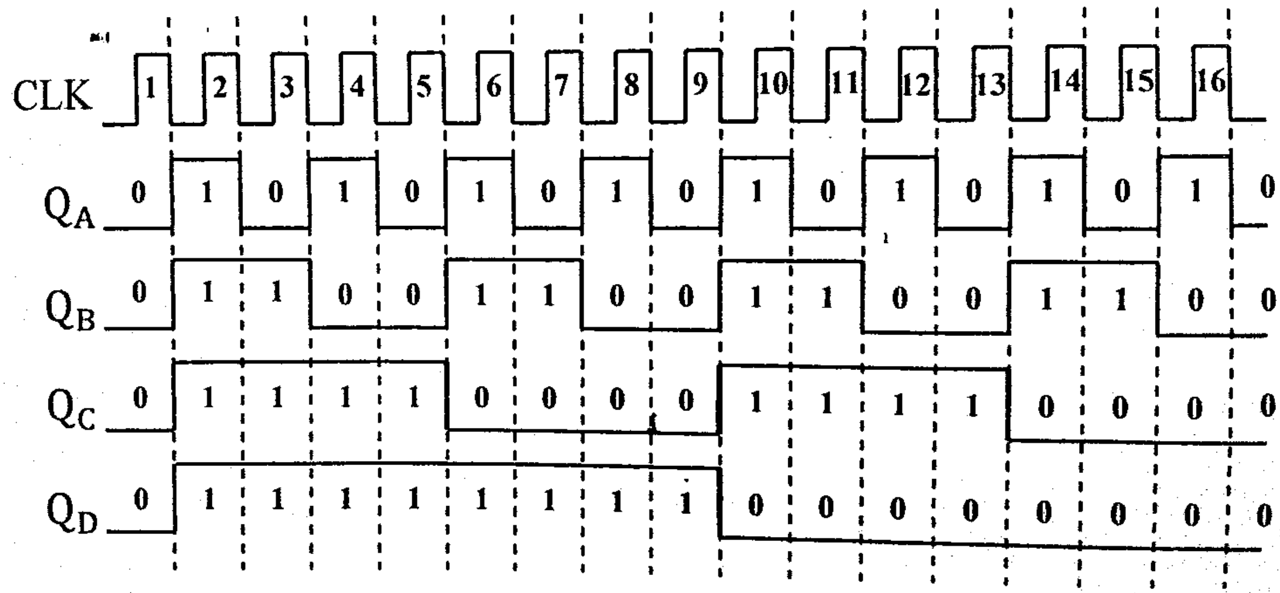


Figure 4) Timing diagram of 4-bit asynchronous down-counter.

****

Asynchronous Up/Down Counter:

Explain about Asynchronous Up/Down counter.

- The up-down counter has the capability of counting upwards as well as downwards. It is also called multimode counter.
- In asynchronous up-counter, each flip-flop is triggered by the normal output Q of the preceding flip-flop.
- In asynchronous down counter, each flip-flop is triggered by the complement output $\overline{Q}$ of the preceding flip-flop.
- In both the counters, the first flip-flop is triggered by the clock output.

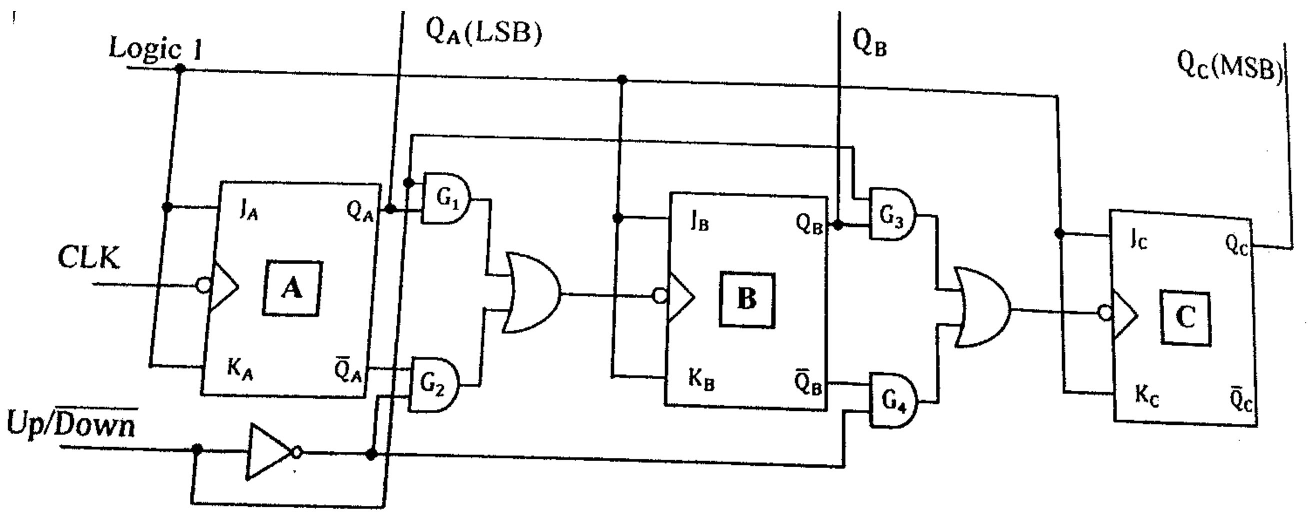


Figure 3-bit asynchronous up/down-counter

- *If Up/$\overline{\text{Down}}$ =1,* the 3-bit asynchronous up/down counter will perform up-counting. It will count from 000 to 111. If Up/$\overline{\text{Down}}$ =1 gates G_2 and G_4 are disabled and gates G_1 and G_3 are enabled. So that the circuit behaves as an up-counter circuit.
- *If Up/$\overline{\text{Down}}$ =0,* the 3-bit asynchronous up/down counter will perform down-counting. It will count from 111 to 000. If Up/$\overline{\text{Down}}$ =0 gates G_2 and G_4 are enabled and gates G_1 and G_3 are disabled. So that the circuit behaves as an down-counter circuit.

Up/$\overline{\text{Down}}$ = 1 (↓)   Up/$\overline{\text{Down}}$ = 0 (↑)

Q_C	Q_B	Q_A
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Table : Truth table for 3-Bit asynchronous Up/Down-counter

****

4- bitSynchronous up-counter:

Explain about 4-bit Synchronous up-counter.

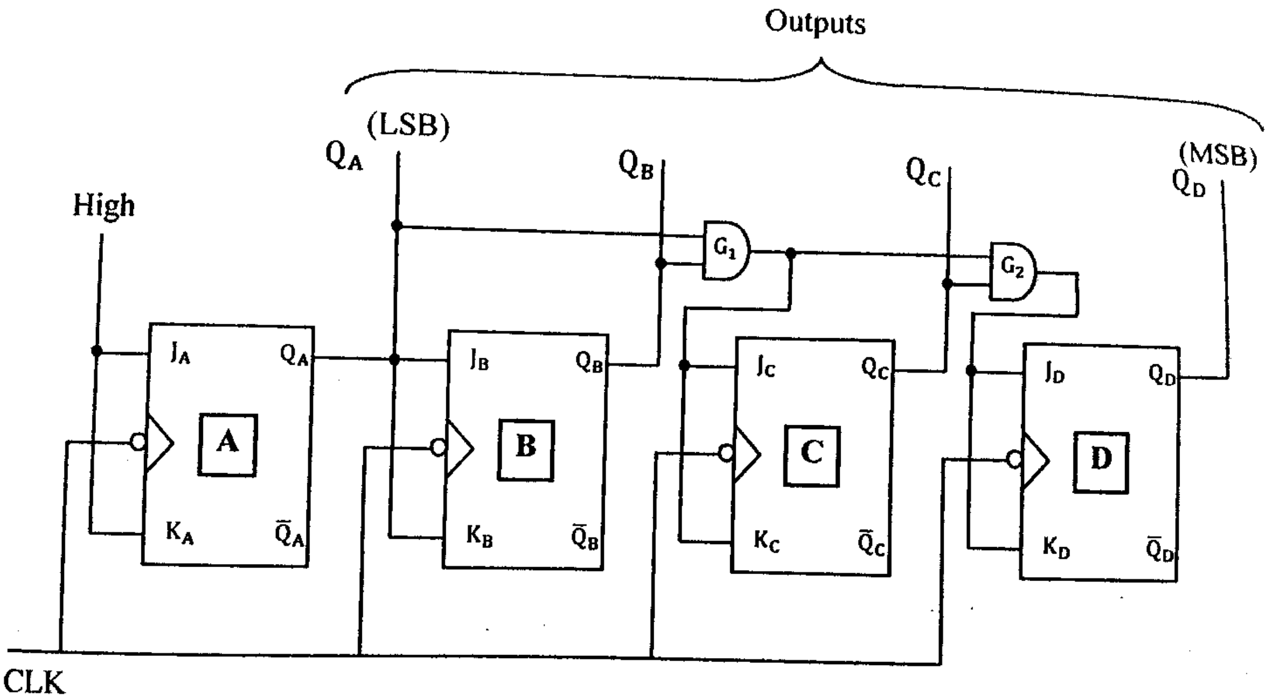


Figure Logic diagram of 4-bit Synchronous up-counter

- In JK Flip-Flop, If J=0, K=0 and clock is triggered, the output never changes. If J=1 and K=1 and the clock is triggered, the past outpit will be complemented.

Initially the register is cleared $Q_DQ_CQ_BQ_A$= 0000.

During the first clock pulse, J_A= K_A = 1, Q_A becomes 1, Q_B, Q_C, Q_D remains 0.

$Q_DQ_CQ_BQ_A$= 0001.

During second clock pulse, J_A= K_A = 1, Q_A=0.

J_B= K_B = 1, Q_B =1, Q_C, Q_D remains 0.

$Q_DQ_CQ_BQ_A$= 0010.

During third clock pulse, J_A= K_A = 1, Q_A=1.

J_B= K_B = 0, Q_B =1, Q_C, Q_D remains 0.

$Q_DQ_CQ_BQ_A$= 0011.

During fourth clock pulse, J_A= K_A = 1, Q_A=0.

J_B= K_B = 1, Q_B =0

J_C= K_C = 1, Q_C=1

Q_D remains 0

$Q_DQ_CQ_BQ_A$= 0100.

The same procedure repeats until the counter counts up to 1111.

CLK	Outputs			
	Q_D	Q_C	Q_B	Q_A
-	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Table Truth table for 4-bit synchronous up-counter

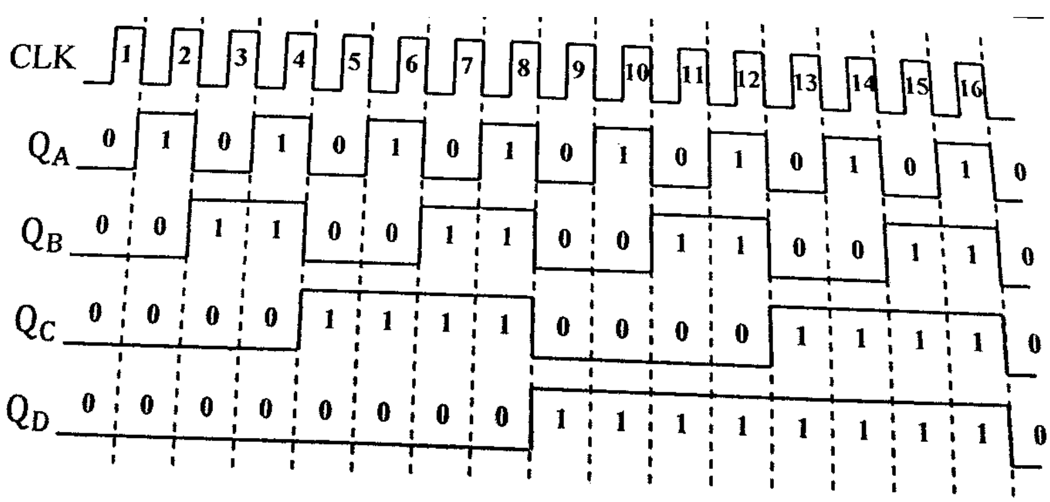


Figure Timing diagram of 4-bit synchronous up-counter

****

4- bit Synchronous down-counter:

Explain about 4-Bit Synchronous down counter.

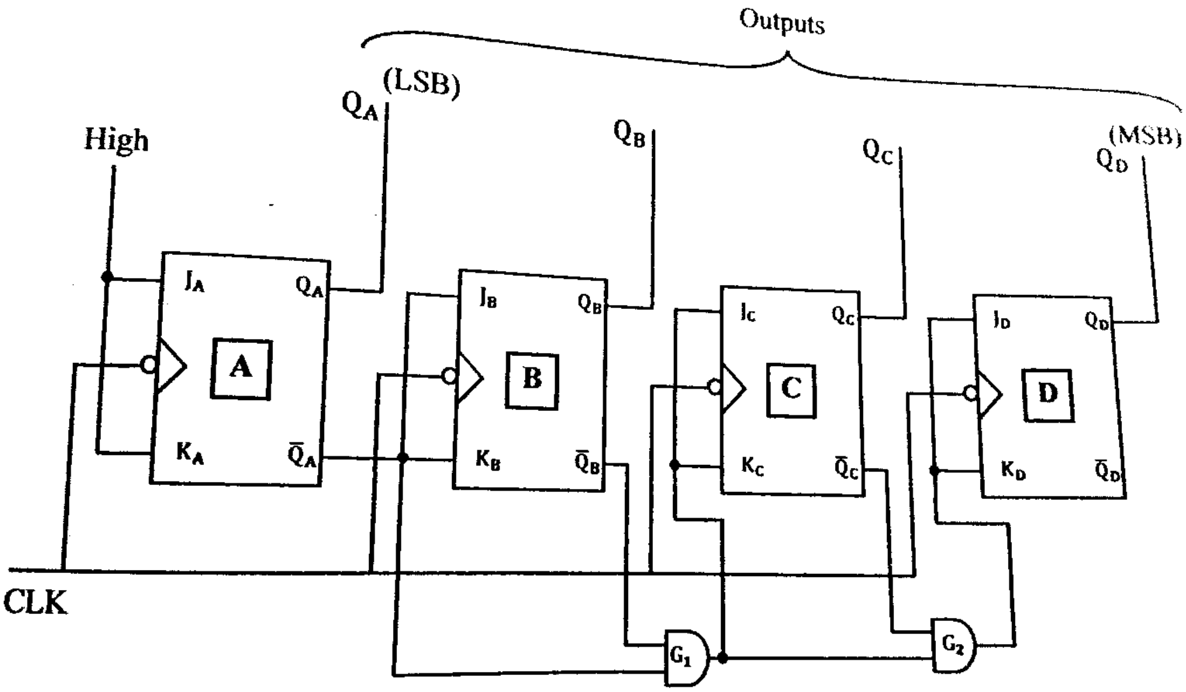


Figure 3 Logic diagram of 4-bit synchronous down-counter

In JK Flip-Flop, If J=0, K=0 and clock is triggered, the output never changes. If J=1 and K=1 and the clock is triggered, the past outpit will be complemented.

Initially the register is cleared $Q_DQ_CQ_BQ_A= 0000$

$\overline{Q_D}\,\overline{Q_C}\,\overline{Q_B}\,\overline{Q_A}= 1111$

During the first clock pulse, $J_A= K_A = 1, Q_A=1$

$J_B= K_B = 1, Q_B =1$

$J_C= K_C = 1, Q_C =1$

$J_D= K_D = 1, Q_D =1$

$Q_DQ_CQ_BQ_A= 1111$

$\overline{Q_D}\,\overline{Q_C}\,\overline{Q_B}\,\overline{Q_A}= 0000$

During the second clock pulse, $J_A= K_A = 1, Q_A =0$

$J_B= K_B = 0, Q_B =1$

$J_C= K_C = 0, Q_C =1$

$J_D= K_D = 0, Q_D =1$

$Q_DQ_CQ_BQ_A= 1110$

$Q_DQ_CQ_B\overline{Q_A}= 0001$

During the second clock pulse, $J_A = K_A = 1$, $Q_A = 1$

$J_B = K_B = 1$, $Q_B = 0$

$J_C = K_C = 0$, $Q_C = 1$

$J_D = K_D = 0$, $Q_D = 1$

$Q_DQ_CQ_BQ_A = 1101$

The process repeats until the counter down-counts up to 0000.

CLK	Outputs			
	Q_D	Q_C	Q_B	Q_A
-	0	0	0	0
1	1	1	1	1
2	1	1	1	0
3	1	1	0	1
4	1	1	0	0
5	1	0	1	1
6	1	0	1	0
7	1	0	0	1
8	1	0	0	0
9	0	1	1	1
10	0	1	1	0
11	0	1	0	1
12	0	1	0	0
13	0	0	1	1
14	0	0	1	0
15	0	0	0	1
16	0	0	0	0

Table Truth table of 4-bit synchronous down-counter

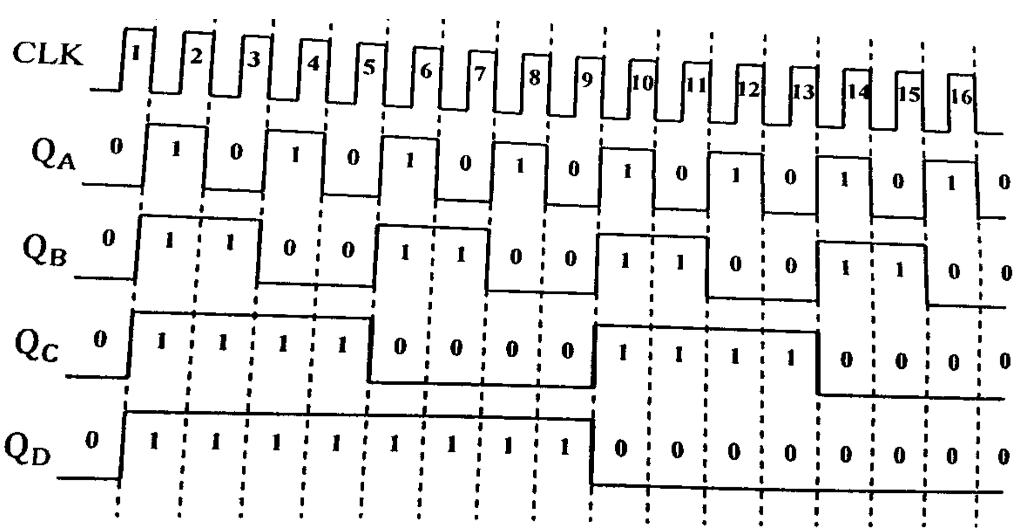


Figure ... Timing diagram of 4-bit synchronous down-counter

****

Modulo 8 Synchronous Up/Down Counter:

Explain about Modulo 8 Synchronous Up/Down Counter.

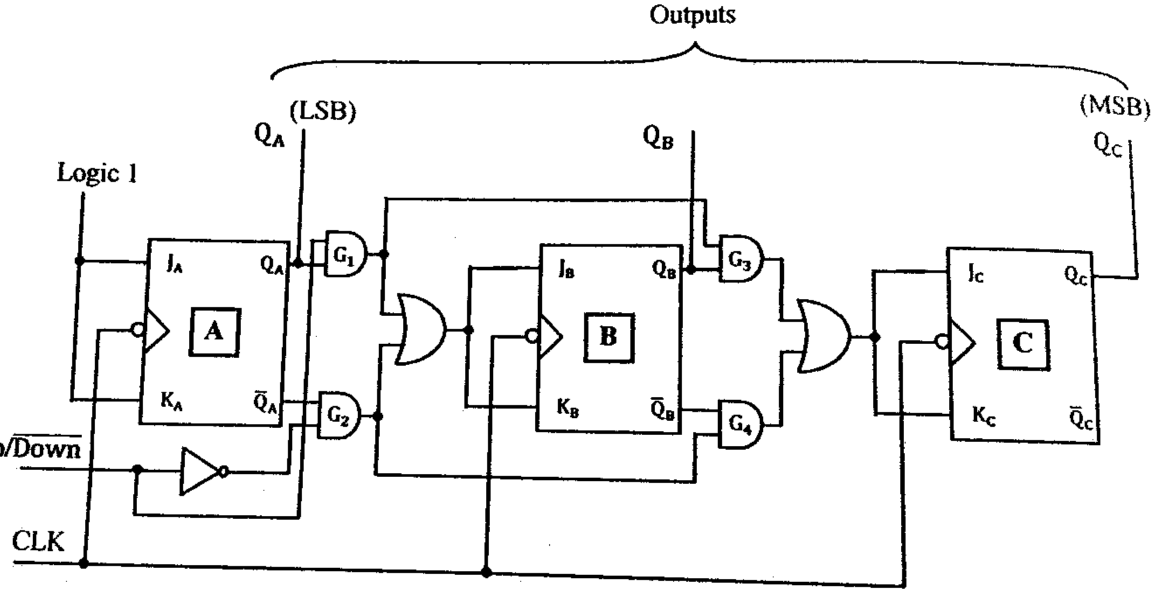


Figure : 3-bit synchronous up/down-counter

In synchronous up-counter the Q_A output is given to J_B, $K_B$and Q_A. Q_B is given to J_C, K_C. But in synchronous down –counter $\overline{Q_A}$output is given toJ_B, K_B and $\overline{Q_A}$. $\overline{Q_B}$ is given to J_C, K_C.

A control input Up/$\overline{\text{Down}}$ is used to select the mode of operation.

If *Up/$\overline{\text{Down}}$ =1,* the 3-bit asynchronous up/down counter will perform up-counting. It will count from 000 to 111. If Up/$\overline{\text{Down}}$ =1 gates G_2 and G_4 are disabled and gates G_1 and G_3 are enabled. So that the circuit behaves as an up-counter circuit.

If *Up/$\overline{\text{Down}}$ =0,* the 3-bit asynchronous up/down counter will perform down-counting. It will count from 111 to 000. If Up/$\overline{\text{Down}}$ =0 gates G_2 and G_4 are enabled and gates G_1 and G_3 are disabled. So that the circuit behaves as an down-counter circuit.

Up/$\overline{\text{Down}}$ = 1

Q_C	Q_B	Q_A
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Up/$\overline{\text{Down}}$ = 0

Table : Truth table for 3-Bit asynchronous Up/Down-counter

****

****

DESIGN OF RIPPLE COUNTERS

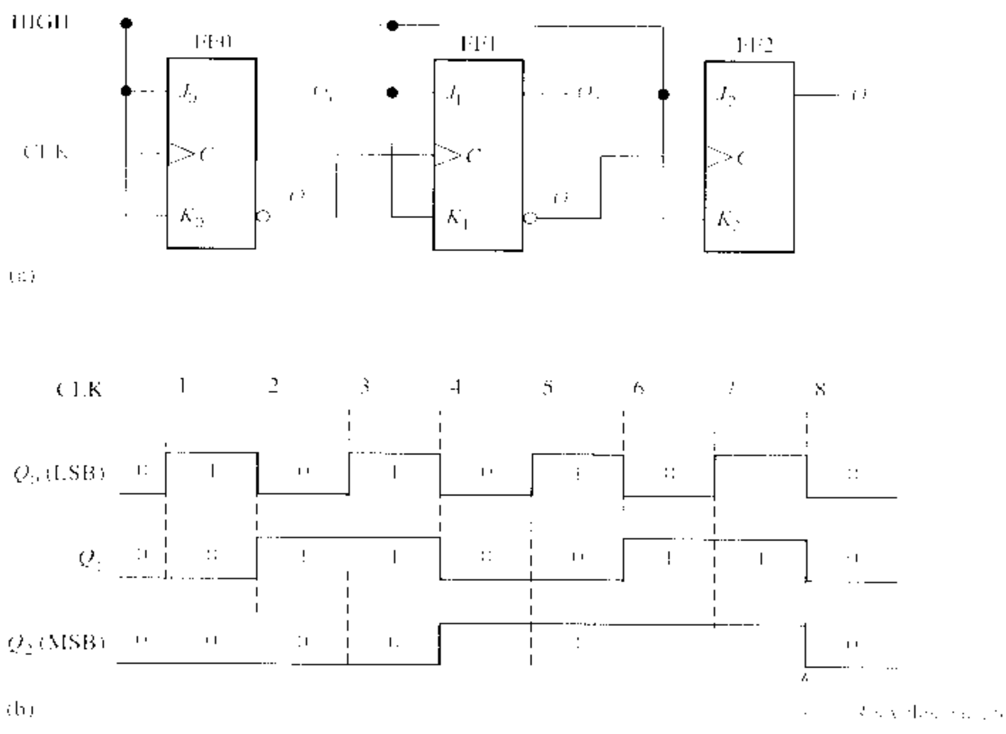
3-Bit Asynchronous Binary Counter/ modulo -7 ripple counter:

Design a 3-bit binary counter using T-flip flops. [NOV – 2019]

Explain about 3-Bit Asynchronous binary counter. ***(Nov -2009)***

The following is a three-bit asynchronous binary counter and its timingdiagram for one cycle. It works exactly the same way as a two-bitasynchronous binary counter mentioned above, except it has eight statesdue to the third flip-flop.

Clock Pulse	Q_2	Q_1	Q_0
Initially	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
8 (recycles)	0	0	0



Asynchronous counters are commonly referred to as ripple counters forthe following reason: The effect of the input clock pulse is first “felt” byFFO. This effect cannot get to FF1 immediately because of thepropagation delay through FF0. Then there is the propagation delaythrough FF1 before FF2 can be triggered. Thus, the effect of an inputclock pulse "ripples" through the counter, taking some time, due topropagation delays, to reach the last flip-flop.

****

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.

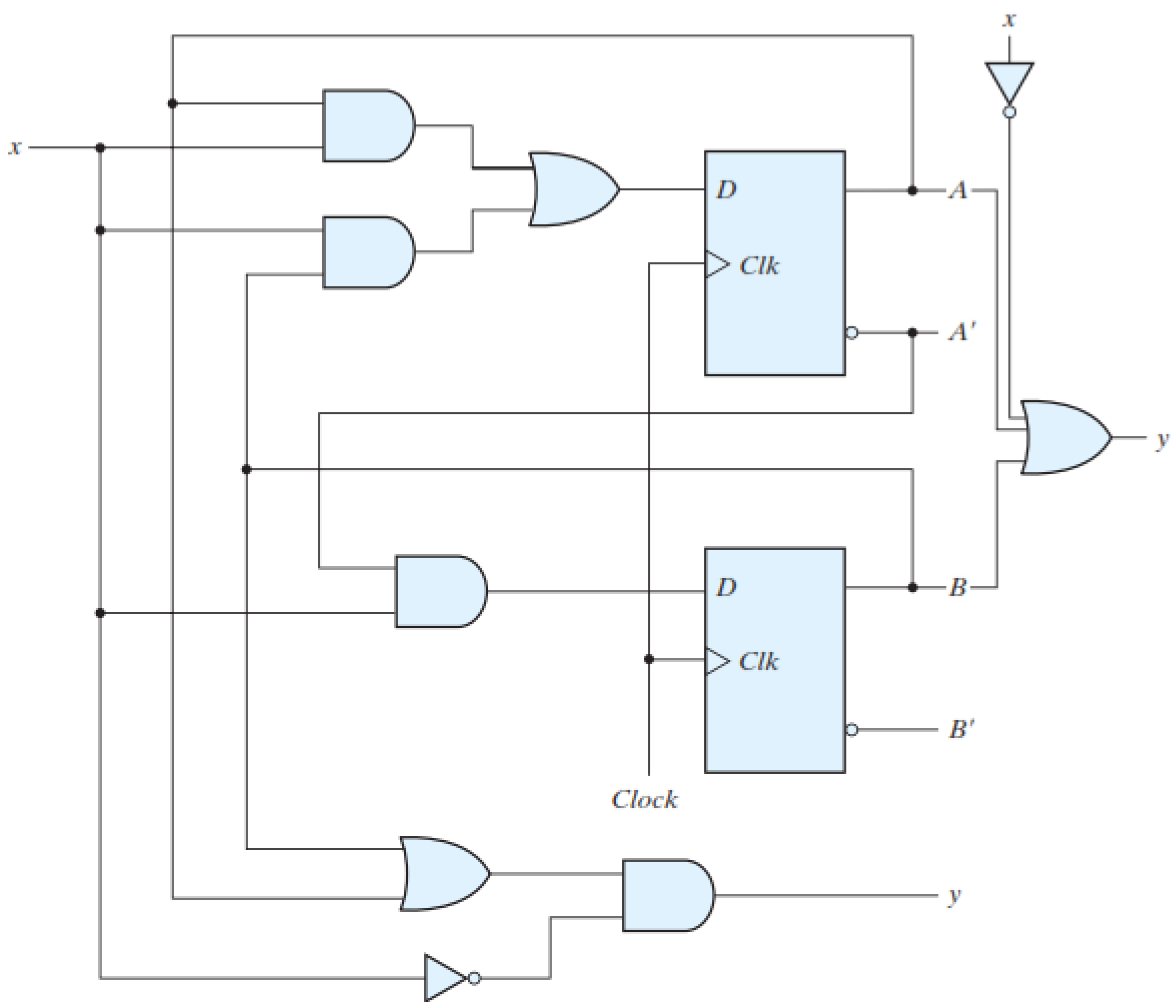


Fig: Example of sequential circuit

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.

$$A(n+1)= A(n)x(n)+B(n)x(n)$$

$$B(n+1)= \overline{A(n)}x(n)$$

They can be written in simplified form as,

$$A(n+1) = \underline{Ax} + Bx$$

$$B(n+1) = Ax$$

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.

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.

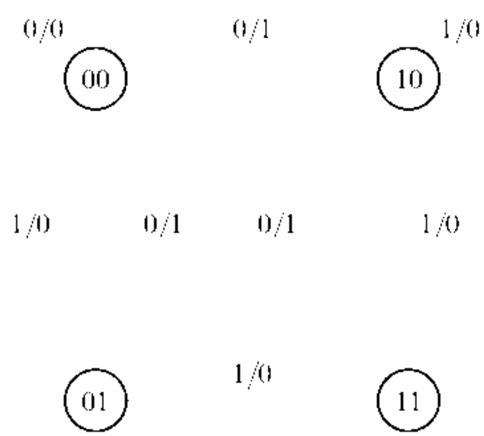
The state table for the circuit is shown. This is derived using state equations.

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.

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,

$$D_A = Ax + Bx$$
$$D_B = Ax$$

****

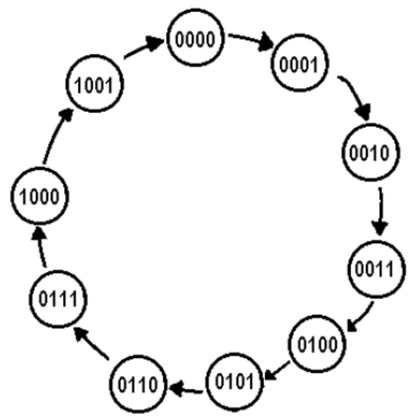
****

***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 thesequence.

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:

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	1	X	X	1
01	1	X	X	1
11	X	X	X	X
10	1	X	X	X

$J_0 = 1$

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	X	1	1	X
01	X	1	1	X
11	X	X	X	X
10	X	1	X	X

$K_0 = 1$

Q_3Q_2 \ Q_1Q_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_3Q_2 \ Q_1Q_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_3Q_2 \ Q_1Q_0	00	01	11	10
00			1	
01	X	X	X	X
11	X	X	X	X
10			X	X

$J_2 = Q_1Q_0$

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	X	X	X	X
01			1	
11	X	X	X	X
10			X	X

$K_2 = Q_1Q_0$

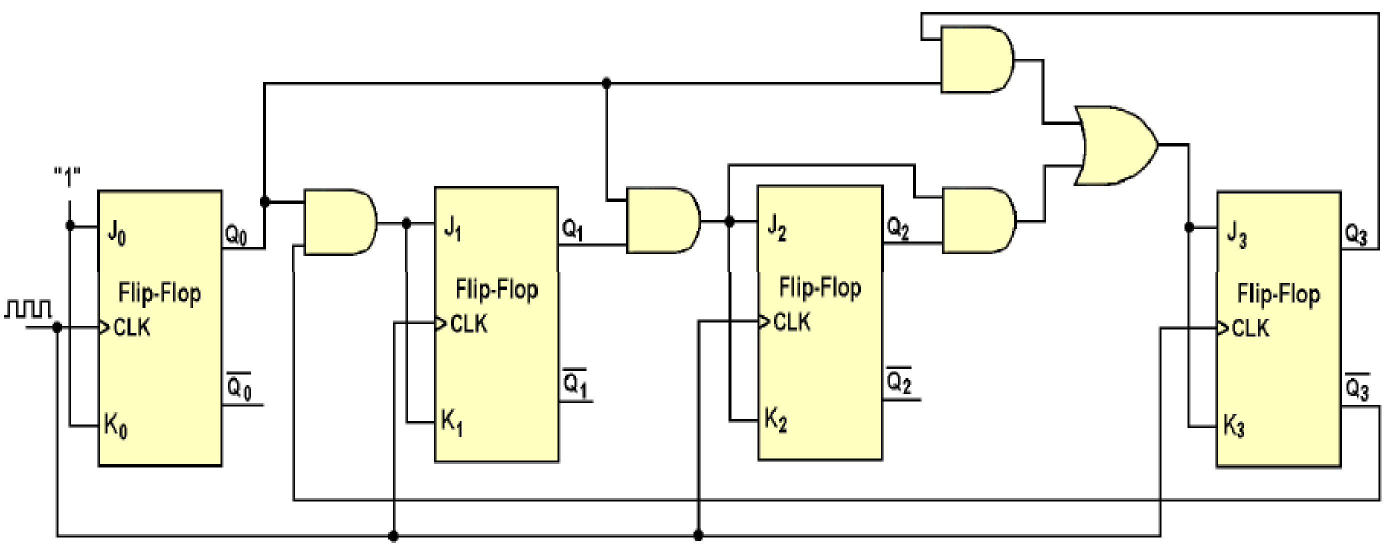
Q_3Q_2 \ Q_1Q_0	00	01	11	10
00				
01			1	
11	X	X	X	X
10	X	X	X	X

$J_3 = Q_3Q_0 + Q_2Q_1Q_0$

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	X	X	X	X
10		1	X	X

$K_3 = Q_3Q_0 + Q_2Q_1Q_0$

Logic Diagram:

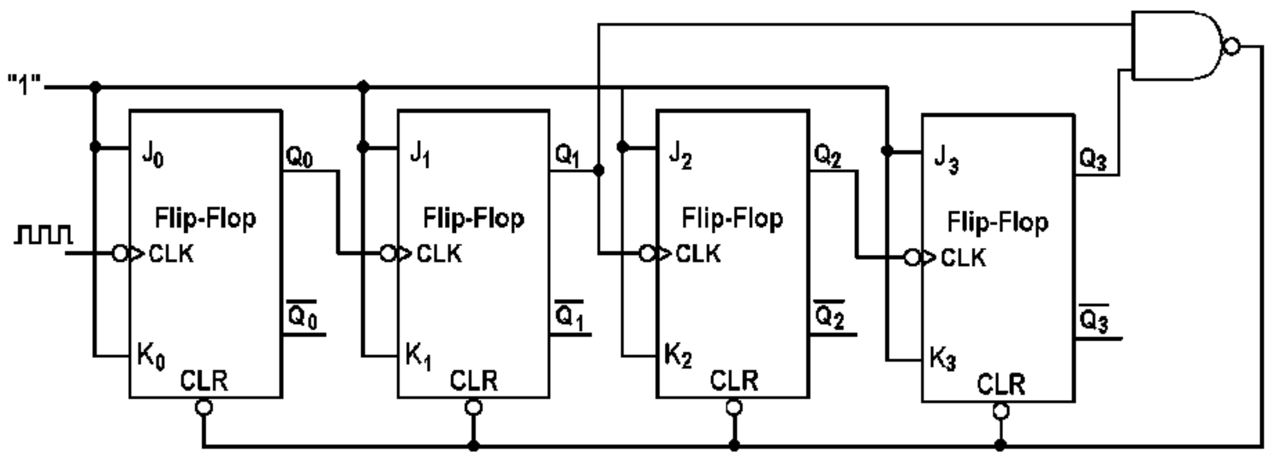


****

Design of an Asynchronous Decade Counter Using JK Flip- Flop.

An asynchronous decade counter will count from zero to nine and repeat thesequence. 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 gateis used to reset the counter at the ninth count. At the ninth count the outputs offlip-flop Q3 and Q1 will be high simultaneously. This will cause the output ofNAND to go to logic "0" that would reset the flip-flip. The logic design of thecounter is shown in Fig.

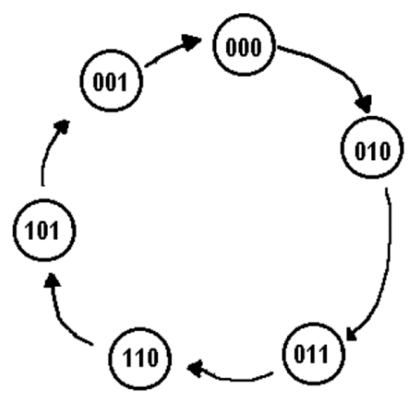


****

***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:

$Q_2 \backslash Q_1Q_0$	00	01	11	10
0	0	0	0	1
1	X	X	X	1

$R_0 = Q_1 \cdot \overline{Q}_0$

$Q_2 \backslash Q_1Q_0$	00	01	11	10
0	X	1	1	0
1	X	0	X	0

$S_0 = \overline{Q}_2 \cdot Q_0$

$Q_2 \backslash Q_1Q_0$	00	01	11	10
0	1	0	X	X
1	X	0	X	0

$R_1 = \overline{Q}_1 \cdot \overline{Q}_0$

$Q_2 \backslash Q_1Q_0$	00	01	11	10
0	0	X	0	0
1	X	X	X	1

$S_1 = Q_2$

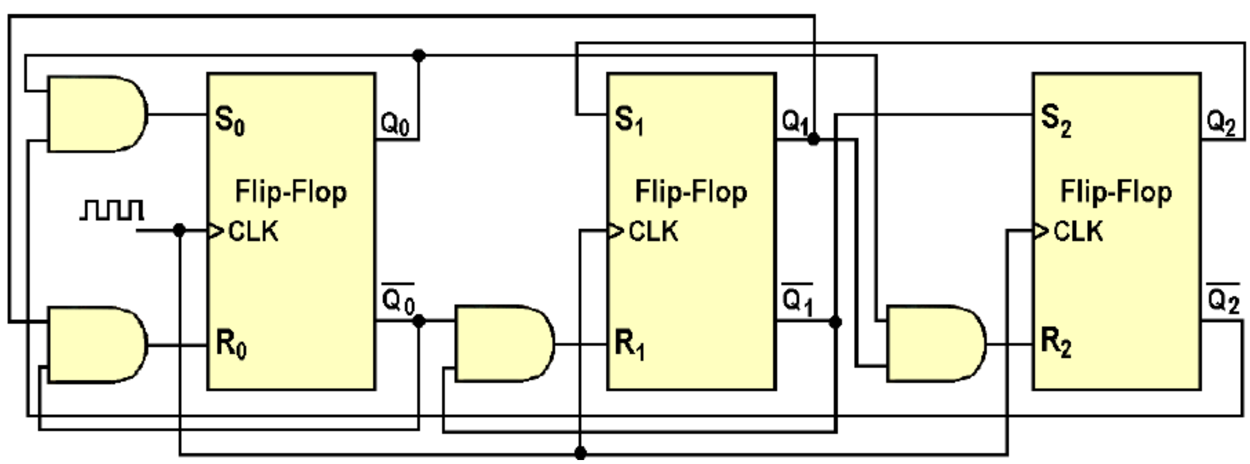
$Q_2 \backslash Q_1Q_0$	00	01	11	10
0	0	0	1	0
1	X	0	X	X

$R_2 = Q_1 \cdot Q_0$

$Q_2 \backslash Q_1Q_0$	00	01	11	10
0	X	X	0	X
1	X	1	X	0

$S_2 = \overline{Q}_1$

Logic Diagram:



SHIFT REGISTERS

Explain various types of shift registers. (or) Explain the operation of a 4-bit bidirectional shift register. (Or) What are registers? Construct a 4 bit register using D-flip flops and explain the operations on the register.(or) With diagram explain how two binary numbers are added serially using shift registers. (Apr – 2019)[NOV – 2019]

- A register is simply a group of Flip-Flops that can be used to store a binary number.
- There must be one Flip-Flop for each bit in the binary number.
- For instance, a register used to store an 8-bit binary number must have 8 Flip-Flops.

- The Flip-Flops must be connected such that the binary number can be entered (shifted) into the register and possibly shifted out.
- A group of Flip-Flops connected to provide either or both of these functions is called a ***shift register***.
- A register capable of shifting the binary information held in each cell to its neighboring cell in a selected direction is called a shift register.

- There are four types of shift registers namely:
  1. Serial In Serial Out Shift Register,
  2. Serial In Parallel Out Shift Register
  3. Parallel In Serial Out Shift Register
  4. Parallel In Parallel Out Shift Register

1.SerialIn Serial Out Shift Register

- The block diagram of a serial out shift register is as below.

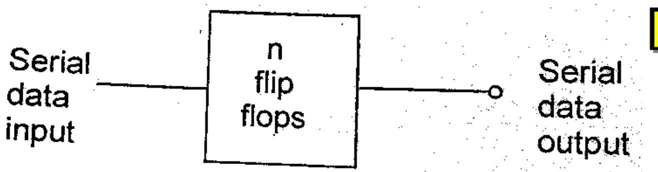
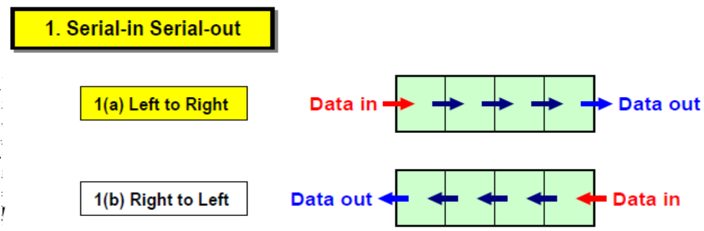


Fig. Block Diagram of SISO



- As seen, it accepts data serially .i.e., one bit at a time on a single input line. It produces the stored information on its single output also in serial form.
- Data may be shifted left using shift left register or shifted right using shift right register.

Shift Right Register

The circuit diagram using D flip-fops is shown in figure

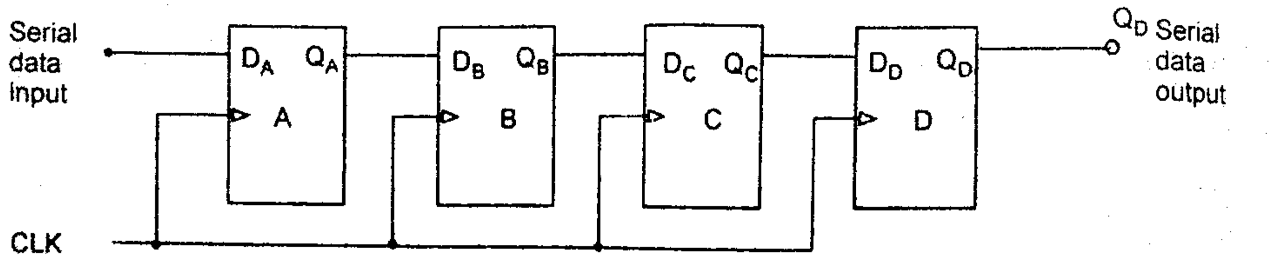


Fig. Serial in serial out right shift register

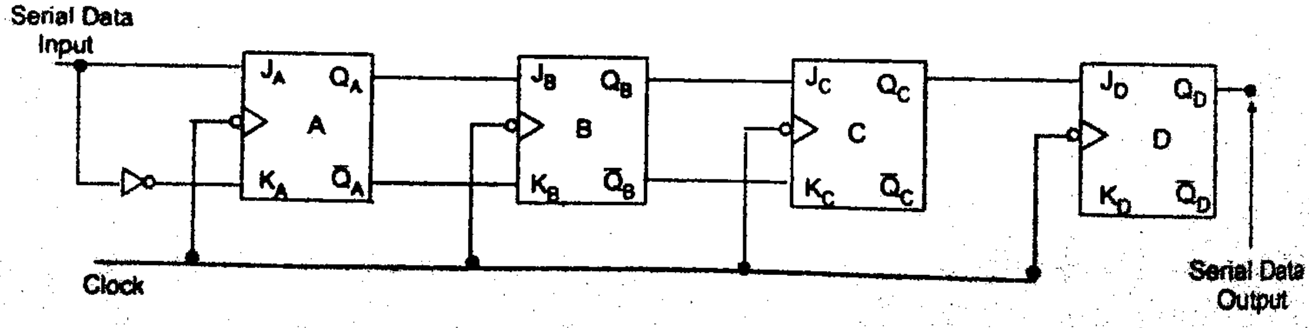


Fig. : SISO Shift Register using JK Flip-flop

- As shown in above figure,the clock pulse is applied to all the flip-flops simultaneously.
- The output of each flip-flop is connected to D input of the flip-flop at its right.
- Each clock pulse shifts the contents of the register one bit position to the right.
- New data is entered into stage A whereas the data presented in stage D are shifted out.
- For example, consider that all stages are reset and a steady logical 1 is applied to the serial input line.
- When the *first clock pulse* is applied, flip-flop A is set and all other flip-flops are reset.
- When the *second clock pulse* is applied,the '1' on the data input is shifted into flip-flop A and '1' that was in flip flop A is shifted to flip-flop B.
- This continues till all flip-flop sets.
- The data in each stage after each clock pulse is shown in table below

Shift Pulse	Serial Data Input	Q_A	Q_B	Q_C	Serial Output Q_D
0	1	0	0	0	0
1	1	1	0	0	0
2	1	1	1	0	0
3	1	1	1	1	0
4	1	1	1	1	1

Shift Left Register

The figure below shows the shift left register using D flip-flops.

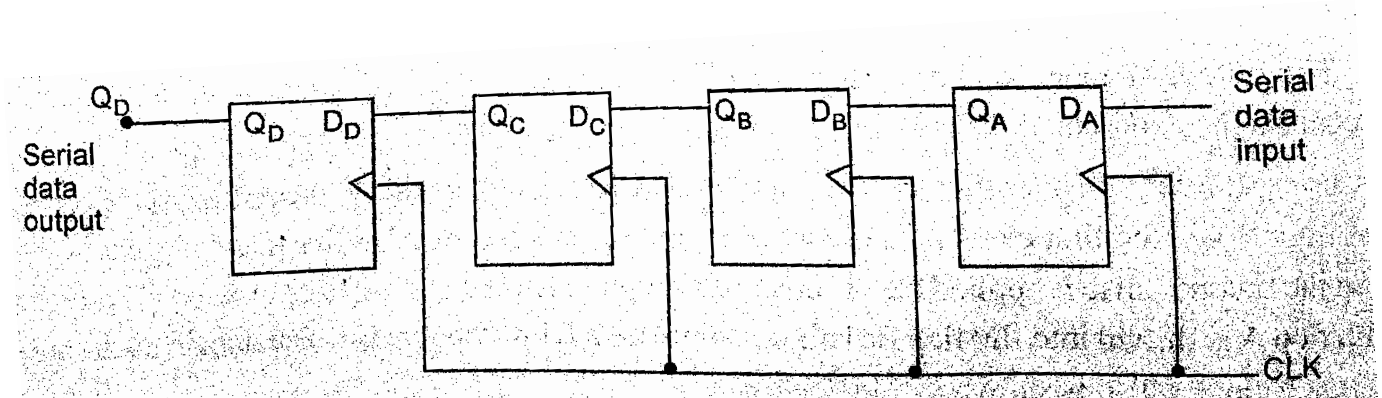
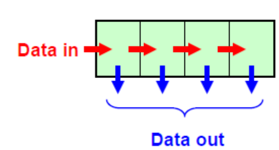


Fig. : Serial in serial out shift left register

- The clock is applied to all the flip-flops simultaneously. The output of each flip-flop is connected to D input of the flip-flop at its left.
- Each clock pulse shifts the contents of the register one bit position to the left.
- Let us illustrate the entry of the 4-bit binary number 1111 into the register beginning with the right most bit.
- When the *first clock pulse* is applied, flip flop A is set and all other flip-flops are reset.
- When *second clock pulse* is applied, '1' on the data input is shifted into flip-flop A and '1' that was in flip flop A is shiftedtoflip-flop B. This continues fill all flip-flop are set.
- The data in each stage after each clock pulse is shown in table below.

Q_D	Q_C	Q_B	Q_A	Serial Input Data	Clock Pulse
0	0	0	0	1	0
0	0	0	1	1	1
0	0	1	1	1	2
0	1	1	1	1	3
1	1	1	1	1	4

2. Serial in Parallel out shift register:



A 4 bit serial in parallel out shift register is shown in figure.

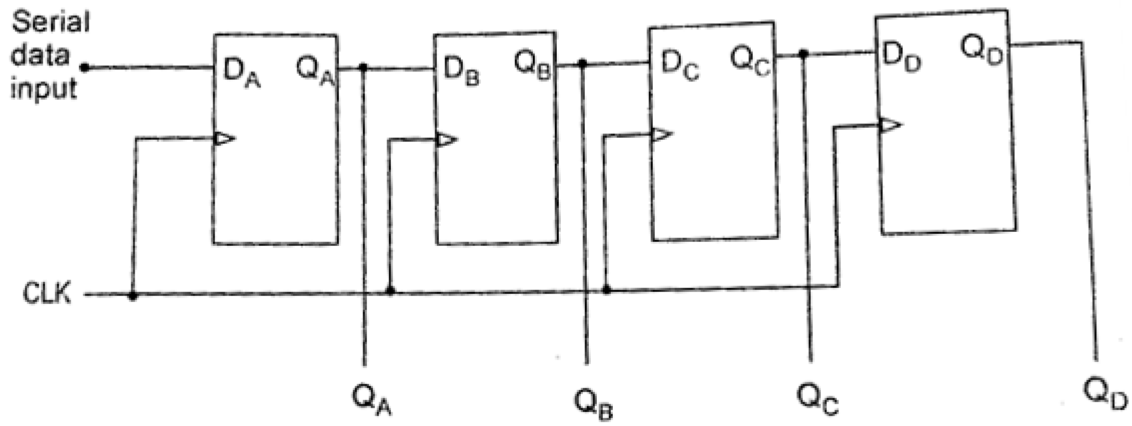


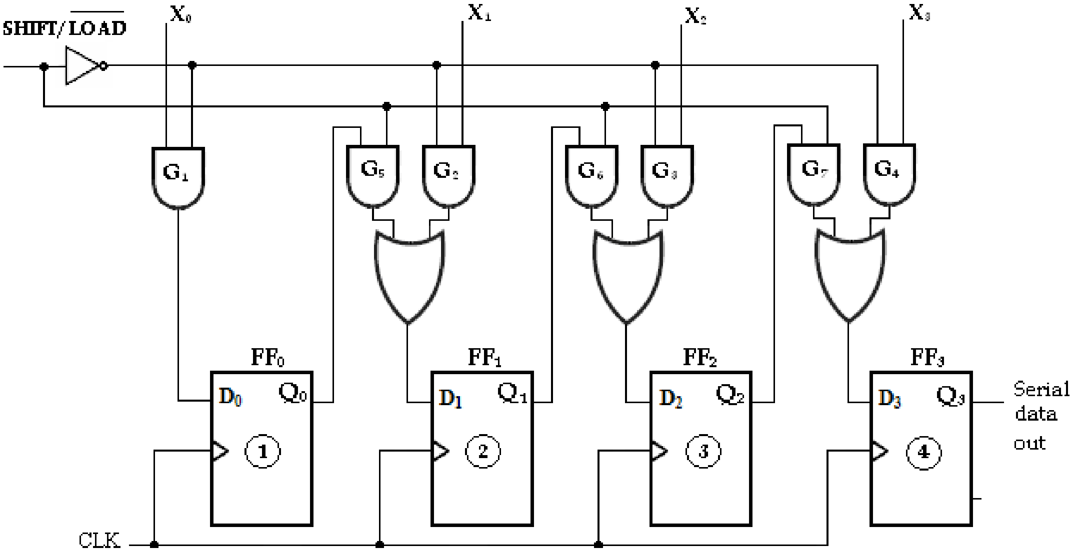
Fig. 3.42: Serial in parallel out shift register

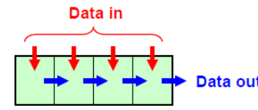
- It consists of one serial input and outputs are taken from all the flip-flops simultaneously.
- The output of each flip-flop is connected to D input of the flip-flop at its right. Each clock pulse shifts the contents of the register one bit position to the right.
- For example, consider that all stages are reset and a steady logical '1' is applied to the serial input line.
- When the *first clock pulse* is applied flip flop A is set and all other flip-flops are reset.
- When the *second pulse* is applied the '1' on the data input is shifted into flip flop A and '1' that was in flip flop A is shifted into flip-flop B. This continues till all flip-flops are set. The data in each stage after each clock pulse is shown in table below.

Shift Pulse	Serial Data Input	Parallel Outputs			
		Q_A	Q_B	Q_C	Q_D
0	1	0	0	0	0
1	1	1	0	0	0
2	1	1	1	0	0
3	1	1	1	1	0
4	1	1	1	1	1

3. Parallel In Serial Out Shift register:

- For register with parallel data inputs, register the bits are entered simultaneously into their respective stages on parallel lines.
- A four bit parallel in serial out shift register is shown in figure. Let A,B,C and D be the four parallel data input lines and SHIFT/$\overline{\text{LOAD}}$ is a control input that allows the four bits of data to be entered in parallel or shift the serially.



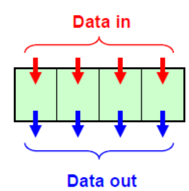


- When SHIFTS/$\overline{\text{LOAD}}$ is low, gates G1 through G3 are enabled, allowing the data at parallel inputs to the D input of its respective flip-flop. When the clock pulse is applied the flip-flops with D=1 will set and those with D=0 will reset, thereby storing all four bits simultaneously.

- When SHIFT/$\overline{\text{LOAD}}$is high. AND gates G1 through G3 are disabled and gates G4 through G6are enabled, allowing the data bits to shifts right from one stage to next. The OR gates allow either the normal shifting operation or the parallel data entry operation, depending on which AND gates are enabled by the level on the SHIFT/$\overline{\text{LOAD}}$ input.

Parallel In Parallel OutShift Register:

- In parallel in parallel out shift register, data inputs can be shifted either in or out of the register in parallel.
- A four bit parallel in parallel out shift register is shown in figure.Let A, B, C, D be the four parallel data input lines and Q_A,Q_B,Q_C and Q_D be four parallel data output lines. The SHIFT/$\overline{\text{LOAD}}$ is the control input that allows the four bits data to enter in parallel or shift the serially.



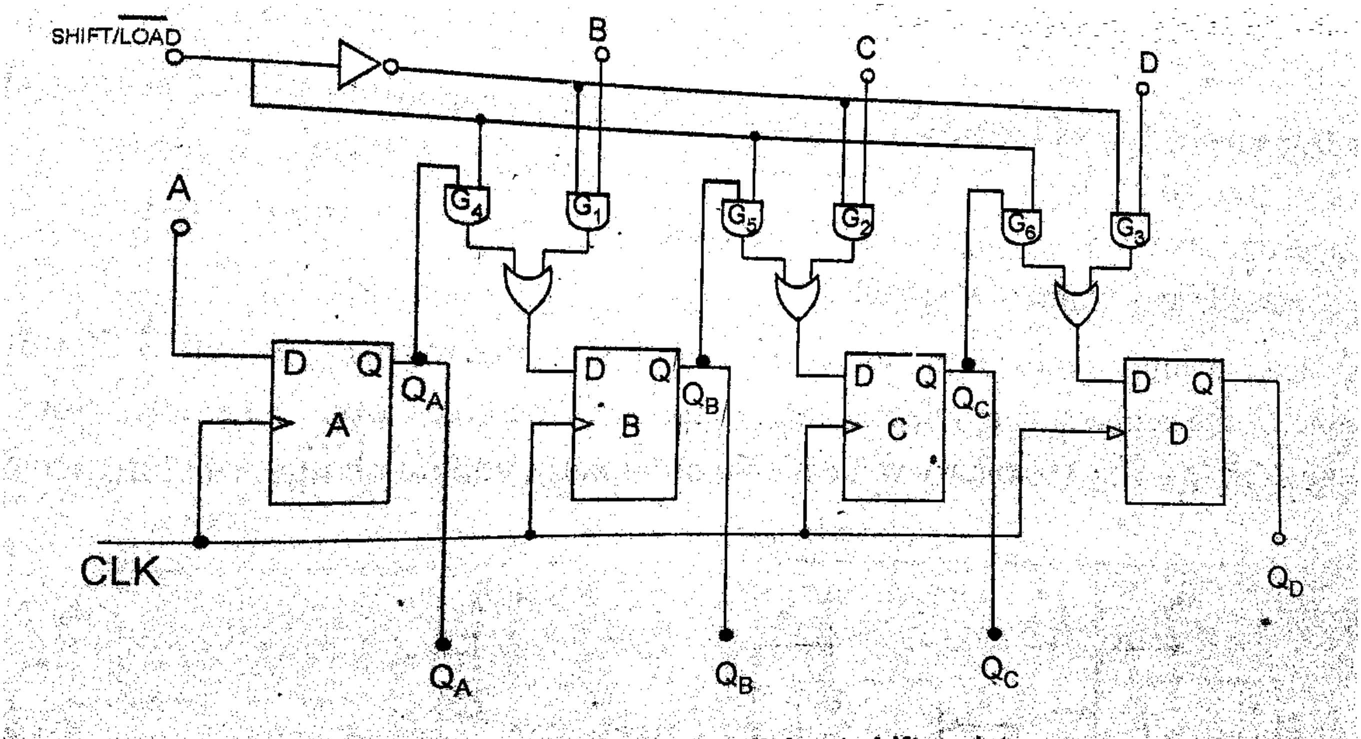


Fig. : Parallel in parallel out shift register

- When *SHIFT/$\overline{LOAD}$ is low*, gates G1 through G3 are enabled, allowing the data at parallel inputs to the D input of its respective flip-flop. When the clock pulse is applied, the flip-flops with D =1 will*set* those with D=0 will *reset* thereby storing all four bits simultaneously. These are immediately available at the outputs Q_A,Q_B,Q_C and Q_D.

- When *SHIFT/$\overline{LOAD}$ is high*, gates G1, through G3 are disabled and gates G4 through G6 are enabled allowing the data bits to shift right from one stage to another. The OR gates allow either the normal shifting operation or the parallel data entry operation, depending on which AND gates are enabled by the level on the SHIFT/$\overline{LOAD}$ input.

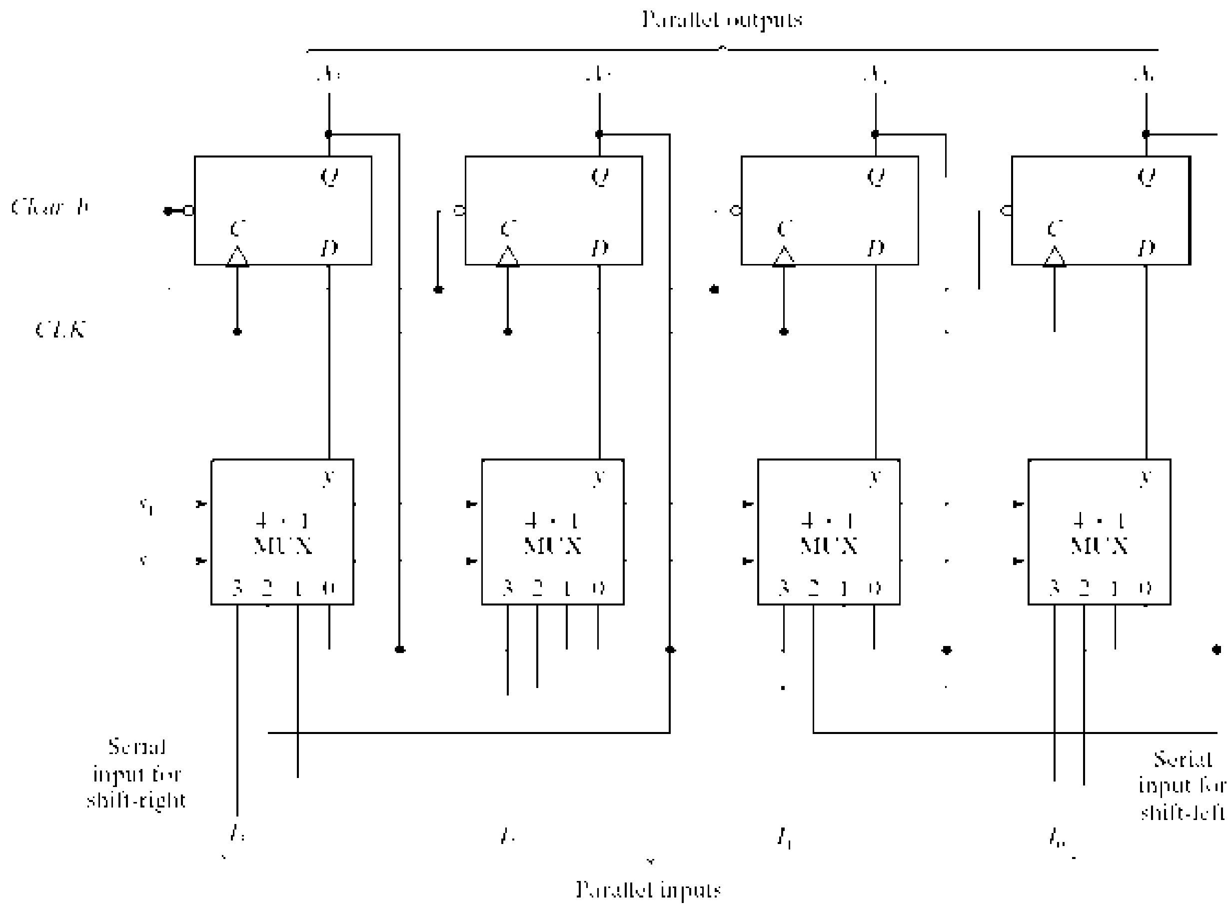
****

Universal Shift Register:

Explain about universal shift register.(Apr -2018)

- A register that can shift data to right and left and also has parallel load capabilities is called universal shift register.
- It has the following capabilities.
    1. A clear control to clear the register to 0.
    2. A clock input to synchronize the operations.

3. A shift right control to enable the shift right operation and the associated serial input and output lines.
4. A shift left control to enable the shift left operation and the associated serial input and output lines.
5. A parallel load control to enable a parallel transfer and the *n* input lines.
6. *n* parallel output lines.
7. A control state that leaves the information in the register unchanged in the presence of the clock.



- The diagram of 4-bit universal shift register that has all that capabilities listed above is shown in figure. It consists of four D flip-flop and four multiplexers.Allthe multiplexers have two common selection inputs S_1 and S_0. Input 0 is selected when S_1S_0=00, input 1 is selected when S_1S_0=01 and similarly for other two inputs.

- The selection inputs control the mode of operation of the register. When S_1S_0=00, the present value of the register is applied to the D inputs of the flip-flop. The next clock pulse transfers into each flip-flop the binary value it held previously, and no change of state occurs.
- When S_1S_0=01,terminal 1 of the multiplexer inputs has a path to be the D inputs of the flip-flops. This causes a shift right operation, with the serial input transferred into flip-flop A_3.
- When S_1S_0=10, a shift left operation results with the other serial input going into flip-flop A_0. Finally, when S_1 S_0 = 11, the binary information on the parallel input lines is transferred into the register simultaneously during the next clock edge. The function table is shown below.

Mode Control		
s_1	s_0	**Register Operation**
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

****

SHIFT REGISTER COUNTERS:

Explain about Johnson and Ring counter. (Nov 2018)

Most common shift register counters are Johnson counter and ring counter.

Johnson counter:

- A 4 bit Johnson counter using D flip-flop is shown in figure. It is also called shift counter or twisted counter.

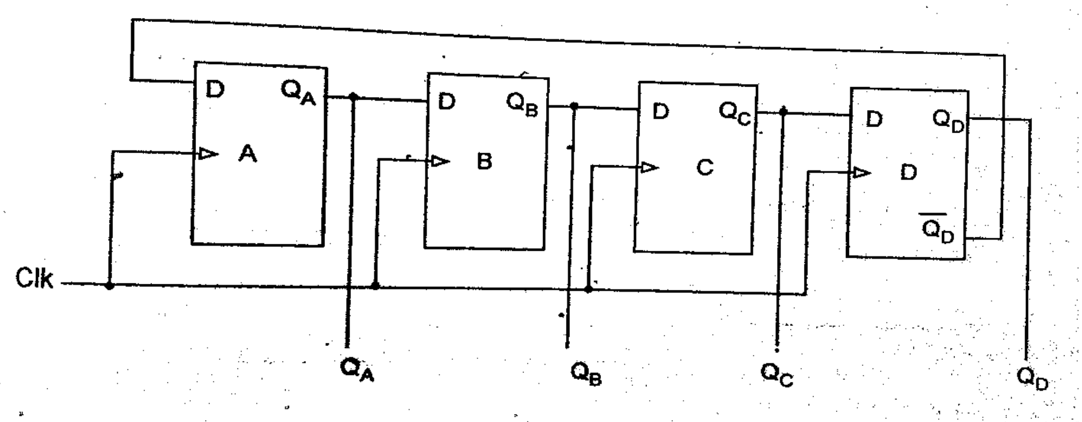


Fig. : Johnson Counter

- The output of each flip-flop is connected to D input of the next stage. The inverted output of last flip-flop $\overline{Q_D}$ is connected to the D input of the first flip-flop A.
- Initially, assume that the counter is reset to 0. i.e., Q_A Q_B Q_C Q_D =0000. The value at D_B = D_C=D_D=0, whereas D_A =1 since $\overline{Q_D}$.

- When the *first clock pulse is* applied, the first flip-flop A is set and the other flip-flops are reset. i.e., $Q_A\ Q_B\ Q_C\ Q_D$ =1000.
- When the *second clock pulse* is applies, the counter is $Q_A\ Q_B\ Q_C\ Q_D$ = 1100. This continues and the counter will fill up with 1's from left to right and then it will fill up with 0's again.
- The sequence of states is shown in the table. As observed from the table, a 4-bit shift counter has 8 states. In general, an *n*-flip-flop Johnson counter will result in *2n* states.

Clock Pulse	Q_A	Q_B	Q_C	Q_D	$\overline{Q_D}$
0	0	0	0	0	1
1	1	0	0	0	1
2	1	1	0	0	1
3	1	1	1	0	1
4	1	1	1	1	0
5	0	1	1	1	0
6	0	0	1	1	0
7	0	0	0	1	0
0	0	0	0	0	1

The timing diagram of Johnson counter is as follows:

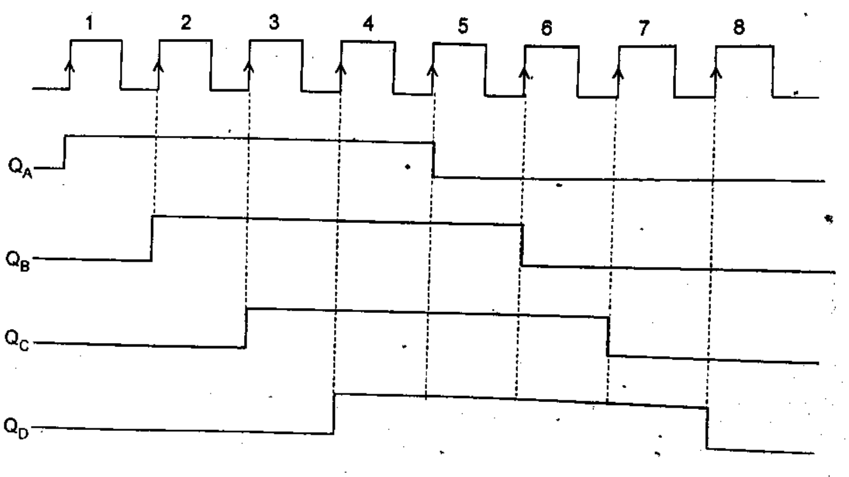


Fig. : Timing Diagram of Johnson Counter

Ring Counter:

A 4- bit ring counter using D Flip-Flop is shown in figure.

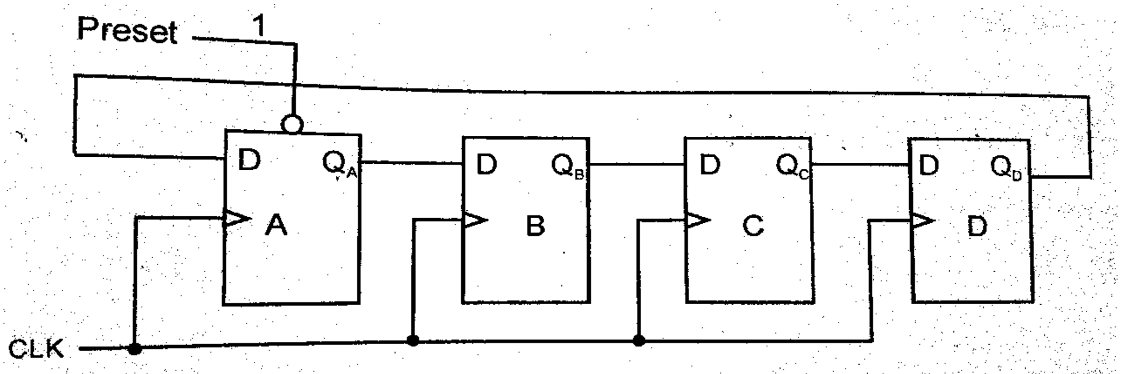


Fig. : Ring Counter

- As shown in figure, the true output of flip-flop D. i.e., Q_D is connected back to serial input of flip-flop A.
- Initially, 1 preset into the first flip-flop and the rest of the flip-flops are cleared i.e., $Q_AQ_BQ_CQ_D$=1000.
- When the *first clock pulse is applied,* the second flip-flop is set to 1while the other three flip flops are reset to 0.
- When the second clock pulse is applied, the '1' in the second flip-flop is shifted to the third flip-flop and so on.
- The truth table which describes the operation of the ring counter is shown below.

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
0	1	0	0	0

- As seen a 4-bit ring counter has 4 states. In general, an *n*-bit ring counter has *n* states. Since a single '1' in the register is made to circulate around the register, it is *called a ring counter.* The timing diagram of the ring counter is shown in figure.

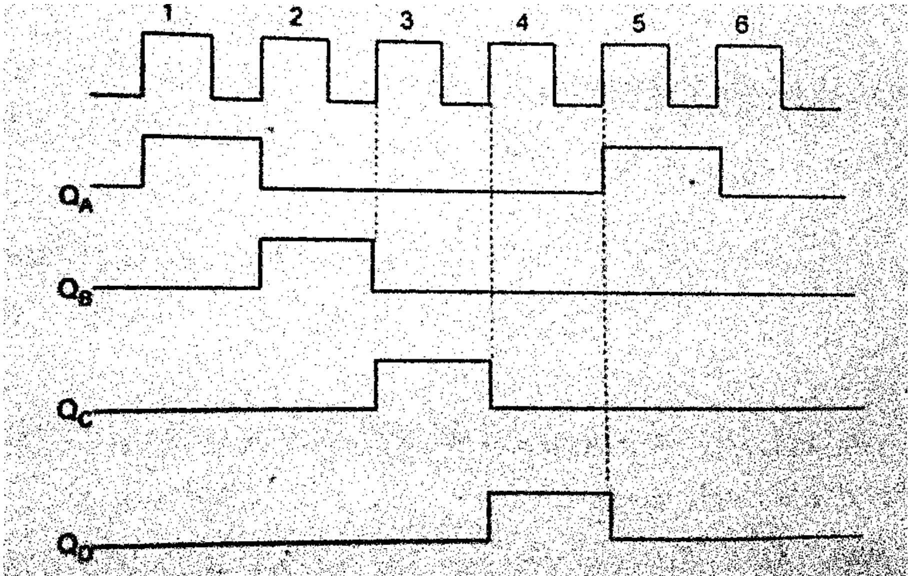


Fig. Timing Diagram of Ring Counter

****

HDL FOR SEQUENTIAL CIRCUITS

Write coding in HDL for various flip-flops.

D Flip Flops

```
module DFF (q, d, clock, clock, reset);
input clock, reset, d;
output q;
reg q;
always @ (posedge clock, negedge reset)
begin
   if (~ reset)
            q ⇐ 1'b0;
   else
            q ⇐ d;
   end
end module
```

T Flip-Flop

```
module TFF (q, t, clock, reset);
input clock, reset, t;
output q;
reg q;
always @ (posedge clock, negedge reset)
begin
   if (~ reset)    (// same as if (reset==0))
   q ⇐ 1'b0;
   else if (t)
   q ⇐ ~q;
   end
end module
```

J-K Flip Flop

```
module JKFF (q, j, k, clock, reset);
input clock, reset, j, k;
output q;
```

```
reg q;
always @ (posedge clock, negedge reset)
begin
   if (~ reset)
   q <= 1'b0;
   else
         begin
               case ({j, k})
               2'b00 : q <= q;
               2'b01 : q <= 0;
               2'b10 : q <= 1;
               2'b11 : q <= ~q;
               end case
         end
   end
end module.
```

T flip flop from D flip flop and gates

```
module T_FF (Q, T, CLK, RST);
   output Q;
   input T, CLK, RST'
   wire DT
   assign DT = Q ^ T              // T flip flop characteristic equation is Q(t + 1) = Q ⊕ T.
   DFF TF1 (Q, DT, CLK, RST);    //Instantiate D flip flop.
endmodule.
```

JK flip flop from D flip flop and gates

```
module JK_FF (Q, J, K, CLK, RST);
   wire JK;
   assign JK = (J&~Q) | (~K &Q); // JK flip flop characteristic equation is Q(t+1) = JQ̄+K̄Q
   //Instantiate D flip flop
   DFF JK1 (Q, JK, CLK, RST);
endmodule
```

Ripple Counter using T flip flop

```
module ripple counter (q, t, CLK, reset);
input t, CLK, reset ;
output [3:0] q;
// Instantiate t flip flop
   TFF t1 (q[0], t, CLK, reset);
   TFF t2 (q[1], t, q[0], reset);
   TFF t3 (q[2], t, q[1], reset);
   TFF t4 (q[3], t, q[2], reset);
end module
```

Synchronous Counter

```
module synchronous counter (CLK, reset, q);
input CLK, reset;
output [3:0]q;
reg [3:0]q;
always @ (posedge reset, posedge CLK)
   begin
            if (reset)
            q <= 4'b0000;
            else
            q <= q + 1'b1;
   end
end module
```

Ripple Counter using D-flip flop

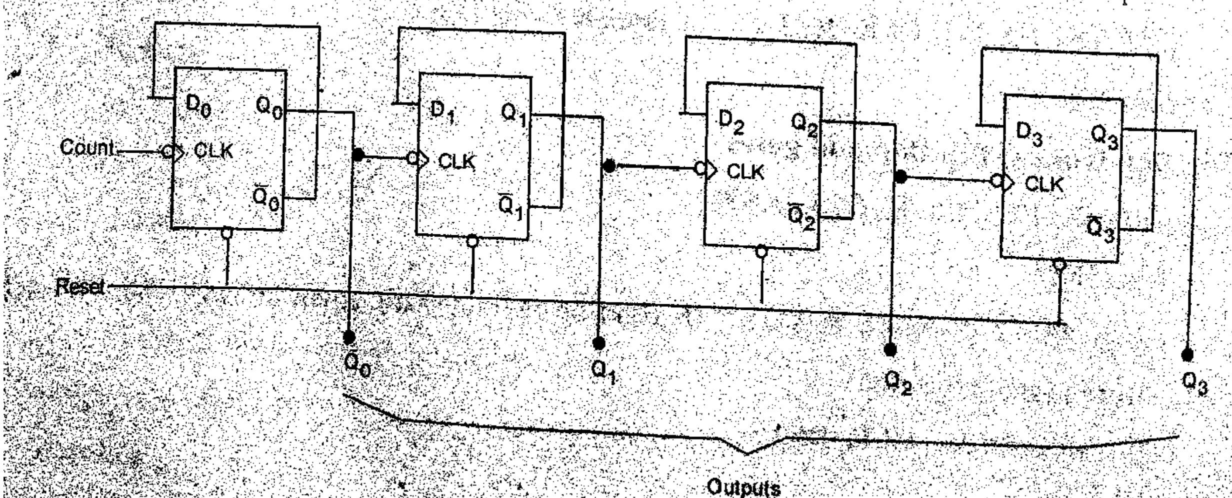


Fig. 3.50 Ripple counter using D-flipflop

```
module ripple_counter (Q3, Q2, Q1, Q0, count, reset);
  output Q3, Q2, Q1, Q0 ;
  input count, reset;
//instantiate complementing flip flop.
  comp_DFF F0 (Q0, count, reset);
  comp_DFF F1 (Q1, Q0, reset);
  comp_DFF F2 (Q2, Q1, reset);
  comp_DFF F3 (Q3, Q2, reset);
endmodule.
//complementing D-flip flop
module comp_DFF (Q, CLK, reset);
  output Q;
  input CLK, RESET;
  reg Q;
  always @(negedge CLK, posedge Reset)
  if (~Reset), Q <=1'b0;
   else Q <= #2 ~Q                //intra-assignment delay
endmodule
```

Universal Shift Register

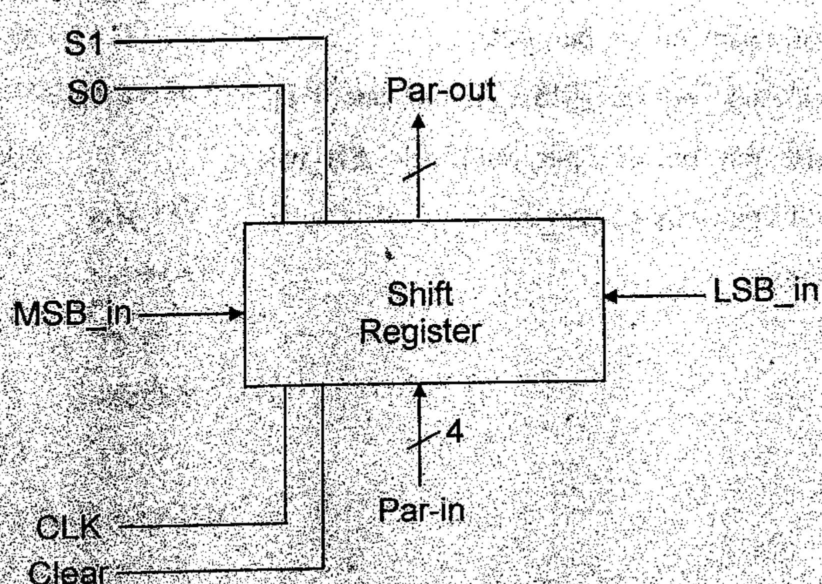


Fig. 3.51 Four Bit Universal Shift Register

Function Table		
Mode Control		
S_1	**S_0**	**Operation**
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

```
module shift register (S1, S0, LSB_in, MSB_in, Par_in, CLK, Clear, par_out);
input S1, S0, LSB_in, MSB_in, Clk, Clear;
input [3:0] par_in;
output [3:0] par_out;
reg [3:0] par_out;
always @ (posedge CLK, negedge Clear)
begin
    if (~ clear)
            par_out <= 4'b000;
    else
            case ({S1, S0})
            2'b00 : par_out <= par_out;
            2'b01 : par_out <= {MSB_in, par_out[3:1]};
            2'b10 : par_out <= {par_out [2:0], LSB_in};
            2'b11 : par_out <= par_in;
            endcase
    end
endmodule
```

Test Bench:

D flip flop

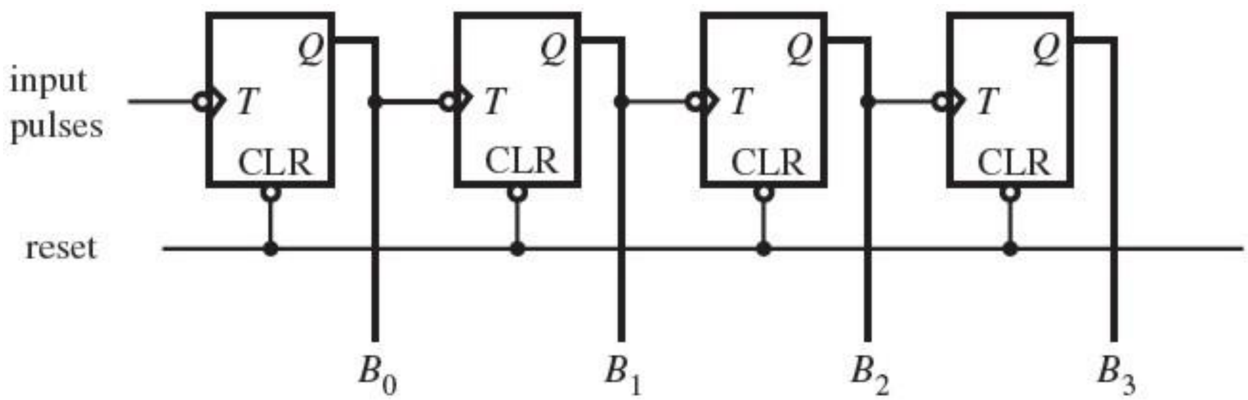
```
module DFF_test bench;
  wire tq;
  reg tclock, treset, td;
  DFF d1 (tclock, treset, td, tq);          //Instantiate D-flip flop module
  initial    begin
                  td = 0;
                  tclock = 0;
                  treset = 0;
             #3   treset = 1;
          end
  always #3 tclock =  ~tclock;
  always #5 d = ~d
  initial #100 $ stop;
endmodule.
```

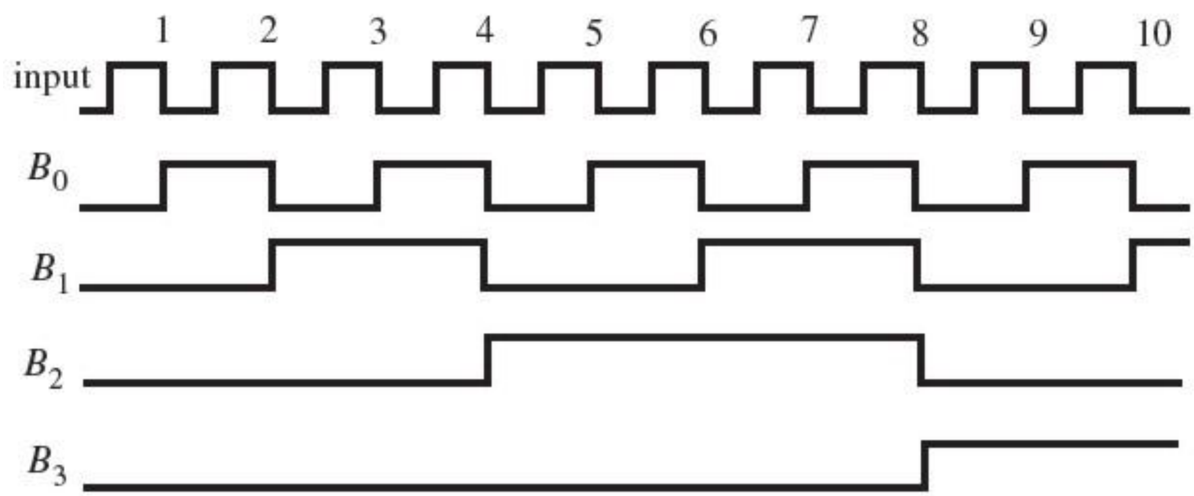
Synchronous Counter

```
module synchronouscounter_test;
  wire [3;0]tq;
  reg tCLK, trest;
  synchronous counter SC1 (tclk, treset, tq);          //Instantiate synchronous counter module
  initial    begin
                  tclk = 0;
                  treset = 0;
          #5      treset = 1;
          #5      treset = 0;
          end
  always #5 tCLK = ~tCLK
  initial #200 $stop
endmodule
```

Write the VHDL Code for 4-Bit Binary Up Counter and explain. (Apr 2019)
VHDL Code for 4-Bit Binary Up Counter

The clock inputs of all the flip-flops are connected together and are triggered by the input pulses. Thus, all the flip-flops change state simultaneously (in parallel).






```
libraryieee;
use ieee.std_logic_1164.all;
useieee.std_logic_unsigned.all;
entityvhdl_binary_counter is
port(C, CLR : in std_logic;
Q : out std_logic_vector(3 downto 0));
endvhdl_binary_counter;
architecturebhv of vhdl_binary_counter is
signaltmp: std_logic_vector(3 downto 0);
begin
process (C, CLR)
begin
if (CLR=’1′) then
tmp<= "0000";
elsif (C’event and C=’1′) then
tmp<= tmp + 1;
```

```
end if;
end process;
Q <= tmp;
```

UNIT IV
ASYNCHRONOUS SEQUENTIAL LOGIC

Analysis and Design of Asynchronous Sequential Circuits – Reduction of State and FlowTables – Race-free State Assignment – Hazards.

Draw the block diagram of a typical asynchronous sequential circuit and explain. Also write the procedure for obtaining transition table from circuit diagram of an asynchronous sequential circuit. [Nov – 2019]

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 instant of time.
- ✓ Inputs and outputs are represented by levels

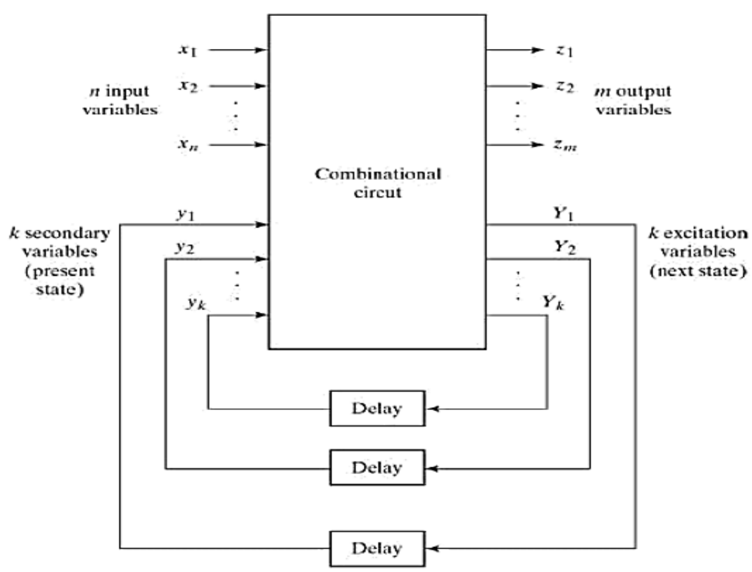
Pulse Mode Circuits:

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.

Explain about Asynchronous Sequential circuits. (Apr 2017, Nov 2017)

Block diagram of Asynchronous Sequential circuits



The communication oftwo units, with each unithavingits own independent 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:

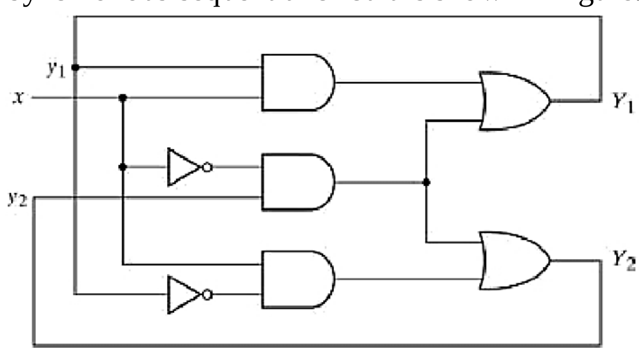
In a circuit,if **present state $y_i \neq$ next state** *Y_i for i=1,2,3…K then the circuit is said to be unstable state. The circuit will be in continuous transition till it reached a stable state.*

****

ANALYSISPROCEDURE OF FUNDAMENTAL MODE SEQUENTIAL CIRCUITS

Explain in detail about analysis procedure of fundamental mode sequential circuits. (or) Outline the procedure for analyzing asynchronous sequential circuits. (Apr 2019) (Dec2011)

- 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 asynchronous 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,

$$Y_1 = xy_1 + x'y_2$$

$$Y_2 = x\,y_1' + x'y_2$$

Step 2:

- Thenext step is to plottheY$_1$and Y_2 functions in amap

$y_1 y_2$ \ x	0	1
00	0	0
01	1	0
11	1	1
10	0	1

Map for
$Y_1 = xy_1 + x'y_2$

$y_1 y_2$ \ x	0	1
00	0	1
01	1	1
11	1	0
10	0	0

Map for
$Y_2 = xy'_1 + x'y_2$

- Combiningthe binaryvalues in correspondingsquares, the followingtransition table is obtained.
- Thetransitiontableshowsthevalueof$Y=Y_1Y_2$insideeachsquare.Thoseentrieswhere$Y=y$arecircled to indicatea*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 thesame information as the transition table.

Step 3:
Transition table

- The transition table is obtained by combining the maps for $Y_1$and Y_2.

y_1y_2 \ x	0	1
00	00	01
01	11	01
11	11	10
10	00	10

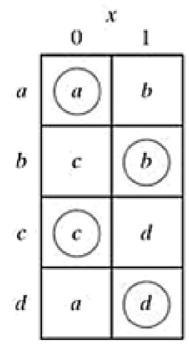
- The transition table is a table which gives the relation between present state, input and next state. If the secondary variables y_1 y_2 is same as excitation variables Y_1 $Y_{2,}$ the state is said to be stable.

- The stable states are indicated by circles. An uncircled entry represents an unstable state.
- 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:

Primitive 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.

****

Explain the problems in asynchronous circuits with examples. ***(Dec 2010,Dec 2012, Dec 2013)***

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 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:

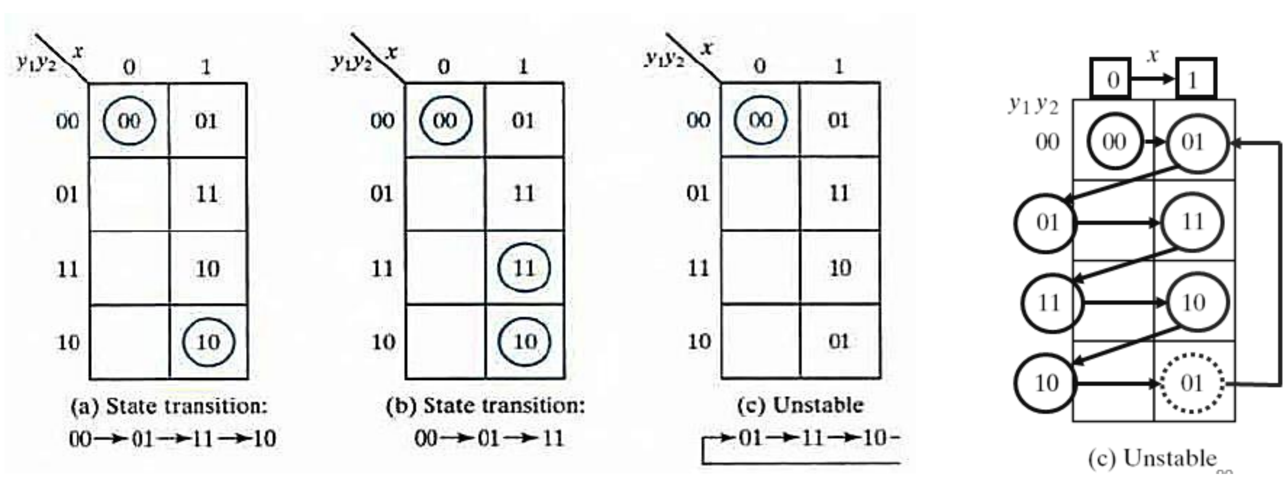
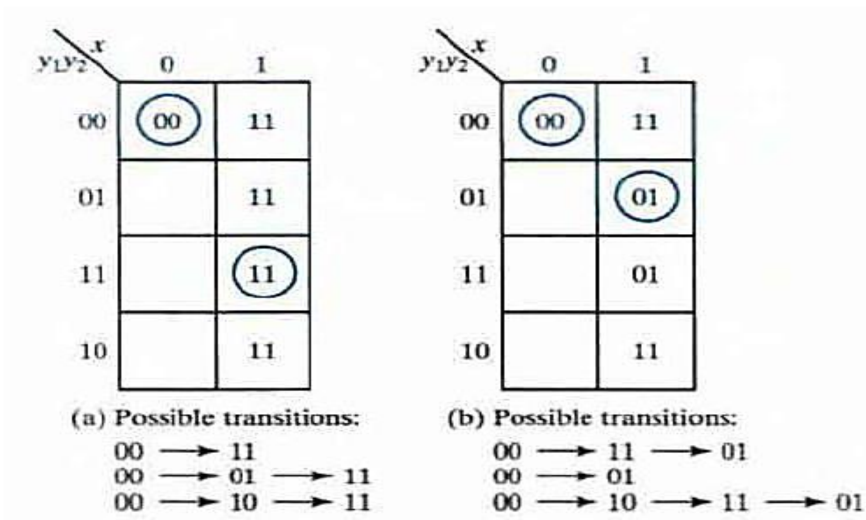


Fig: Examples ofcycles

RaceConditions

- *Araceconditionexistsinanasynchronouscircuitwhentwoormorebinary state variableschangevalue inresponsetoachangeinaninputvariable.*
- Whenunequaldelaysare encountered,araceconditionmay cause thestatevariableto changein an unpredictable manner.
- Ifthe final stable statethat the circuitreaches**does not depend on the order**in which thestatevariables change, theraceis calledanoncritical race.
- Ifthe final stable statethat the circuitreaches**depends on the order**in which thestatevariables change, theraceis calleda critical race.
- **Examples of noncritical races**areillustratedin the transition tables below:



- 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 → 11
00 → 01(y_2 faster) → 11
00 → 10(y_1 faster) → 11

- 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:

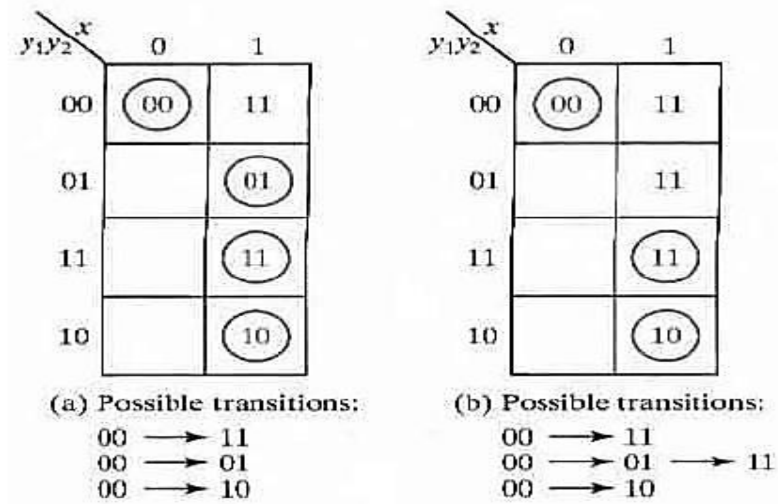


Fig: Examples ofcritical races

- 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.

****

CIRCUITSWITH SRLATCHES

- TheSRlatchisusedasatime-delayelementinasynchronoussequentialcircuits.TheNORgateSRlatch and its truth table are:

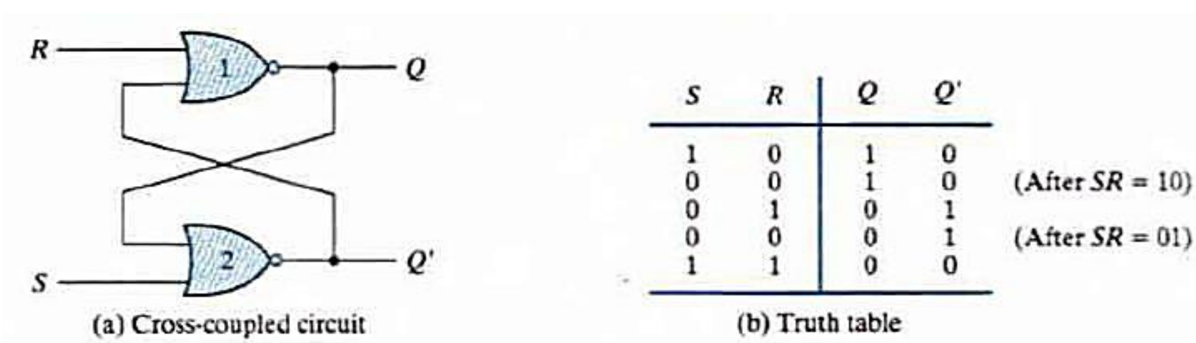
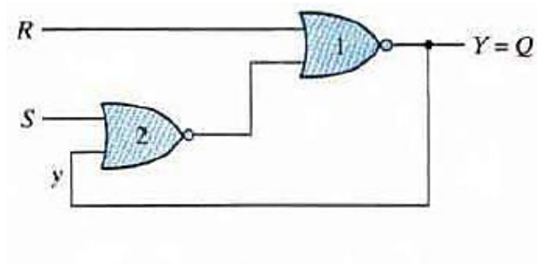


Fig: SR latchwith NORgates

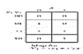
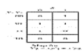
- The feedback is morevisible when the circuitisredrawn as:



(c) Circuit showing feedback

- TheBoolean function ofthe output is:

$$Y = S\overline{R} + \overline{R}y$$

The reduced excitation function,  

and the transition table for the circuitis

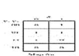
y \ SR	00	01	11	10
0	0	0	0	1
1	1	0	0	1

$Y = SR' + R'y$
$Y = S + R'y$ when $SR = 0$

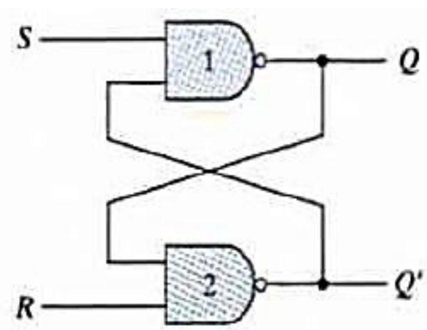
(d) Transition table

- The behavior of the SR latch can be investigated from the transition table. The condition to be avoided is that both S and R inputs must not be 1 simultaneously.
- This condition is avoided when SR = 0 (i.e., ANDing of S and R must always result in 0). When SR = 0 holds at all times, the excitation function derived previously:

$$Y = S\overline{R} + \overline{R}y$$

can   beexpressed as:

- The NAND gate SR latch and its truth table are:



(a) Cross-coupled circuit

S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(After SR = 10)
0	1	1	0	
1	1	1	0	(After SR = 01)
0	0	1	1	

(b) Truth table

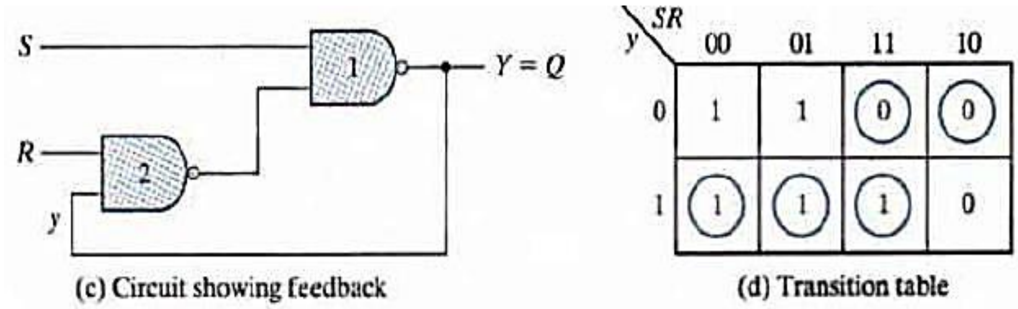


Fig: SR latchwith NAND gates

- TheconditiontobeavoidedhereisthatbothSandRnotbe0simultaneouslywhichissatisfiedwhenS′R′=0.
- The excitation functionforthe circuitis:

$$Y = [S\overline{(Ry)}]' = \overline{S} + Ry$$

Difference between Synchronous and Asynchronous Sequential Circuit (Apr 2019)

Synchronous Sequential Circuit	Asynchronous Sequential Circuit
• It is easy to design.	• It is difficult to design.
• A clocked flip flop acts as memory element.	• An unclocked flip flop or time delay is used as memory element.
• They are slower as clock is involved.	• They are comparatively faster as no clock is used here.
• The states of memory element is affected only at active edge of clock, if input is changed.	• The states of memory element will change any time as soon as input is changed.

****

ANALYSISEXAMPLE

Analyze the Asynchronous sequential circuit with suitable example.

Consider the following circuit:

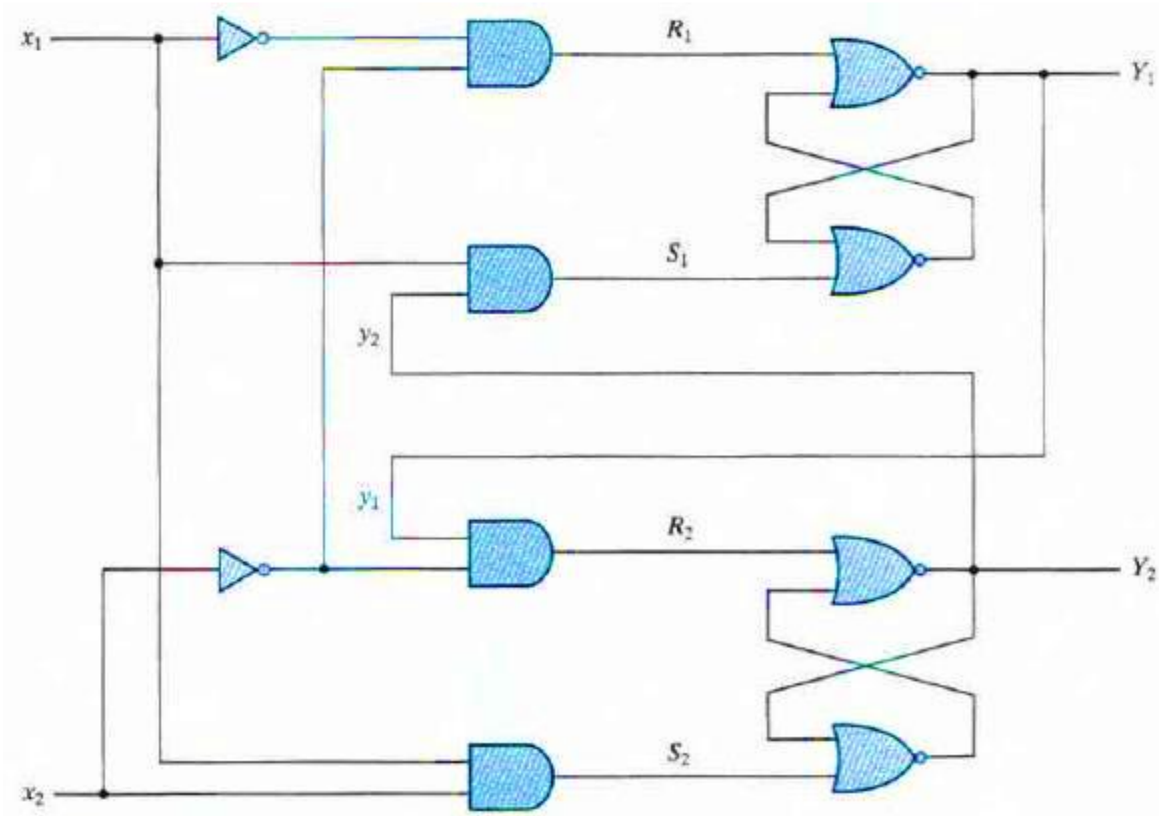


Fig: Example ofa circuit with SR latches

The first step is to obtain the Booleanfunctions forthe S and R inputs in each latch:

$$S_1 = x_1 y_2 \qquad S_2 = x_1 x_2$$
$$R_1 = \overline{x_1\ x_2} \qquad R_2 = \overline{x_2} y_1$$

The next step is to check if SR = 0 is satisfied:

$$S_1 R_1 = x_1 y_2 \overline{x_1}\ \overline{x_2} = 0$$
$$S_2 R_2 = x_1 x_2 \overline{x_2}\ y_1 = 0$$

The result is 0 because

$$x_1 \overline{x_1} = x_2 \overline{x_2} = 0$$

The next step is to derive the transition table of the circuit. The excitation functions are derived from the relation Y = S + R′y as:

$$\mathrm{Y_1 = S_1 + \overline{R_1} y_1 = x_1 y_2 + (x_1 + x_2) y_1 = x_1 y_2 + x_1 y_1 + x_2 y_1}$$

$$\mathrm{Y_2 = S_2 + \overline{R_2} y_2 = x_1 x_2 + (x_2 + \overline{y_1}) y_2 = x_1 x_2 + x_2 y_2 + \overline{y_1} y_2}$$

Next a composite map for $Y = Y_1 Y_2$ is developed

$y_1y_2 \backslash x_1x_2$	00	01	11	10
00	00	00	01	00
01	01	01	11	11
11	00	11	11	10
10	00	10	11	10

- Investigation of the transition table reveals that the circuit is stable.
- There is a critical race condition when the circuit is initially in total state $y_1y_2x_1x_2$ = 1101 and x_2 changes from 1 to 0.
- If Y_1 changes to 0 before Y_2, the circuit goes to total state 0100 instead of 0000.

****

ImplementationExample of Asynchronous sequential circuits. (Nov 2018)

Consider the followingtransition table:

$y \backslash x_1x_2$	00	01	11	10
0	0	0	0	1
1	0	0	1	1

Transition table

$Y = x_1x'_2 + x_1y$

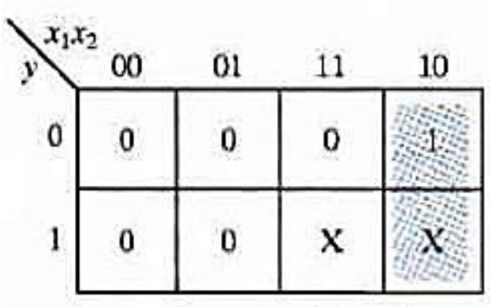
SR LatchExcitationTable:

y	Y	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	1

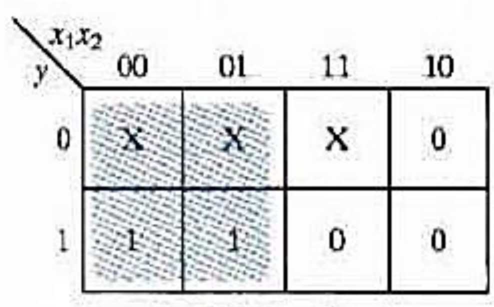
Latch excitation table

UsefulforobtainingtheBooleanfunctionsforSandRandthecircuit'slogicdiagramfromagiven transition table.

FromtheinformationgiveninthetransitiontableandtheSRlatchexcitationtable,wecanobtainmaps forthe S and R inputs of thelatch:



Map for $S = x_1x'_2$

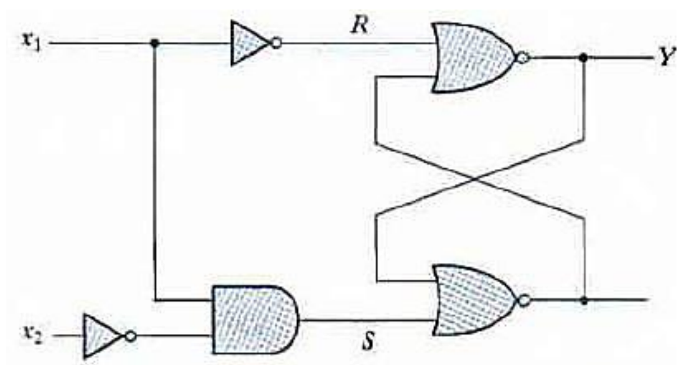


Map for $R = x'_1$

- X represents a don'tcarecondition.
- Themaps arethen used to derivethe simplifiedBoolean functions:

$$S = x_1\overline{x_2} \text{ and } R = \overline{x_1}$$

- ThelogicdiagramconsistsofanSRlatchandgatesrequiredtoimplementtheSandRBoolean functions.
- The circuitwhen aNOR SR latch is usedis as shown below:



Circuit with NOR latch

With a NANDSR latch the complemented valuesforSand R mustbeused.

****

DESIGNPROCEDURE

Explain in detail about design procedure. ***May 2011***

Thereareanumberofstepsthatmustbecarriedoutinordertominimizethecircuitcomplexityandto produceastable circuitwithoutcritical races.Briefly, 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:

Designagatedlatchcircuitwithtwoinputs,G(gate)andD(data),andoneoutputQ.Thegatedlatchis 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 output does not change for any change in D. Use SR latch for implementation of the circuit.

PrimitiveFlowTable

- Aprimitiveflowtableisaflowtablewithonlyonestabletotalstateineachrow.Thetotalstateconsists ofthe internal statecombined with the input.
- To derivethe primitive flow table, first a tablewithallpossible total states in thesystem is needed:

State	Inputs D	Inputs 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:

States	Inputs *DG* 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,wefillinonesquareineachrowbelonging tothestablestateinthatrow.Nextrecallingthat both inputsarenotallowedtochangeatthesametime,weenterdashmarksineachrowthatdiffersintwo 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.
- Two states are compatible, if the states are identical with non-conflicting outputs.
- The cross (x) mark implies non-compatible.

b	✓				
c	✓	d, e X			
d	✓	d, e X	✓		
e	c, f X	✓	d, e X c, f X	X	
f	c, f X	✓	X	c, f X d, e X	✓
	a	b	c	d	e

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.

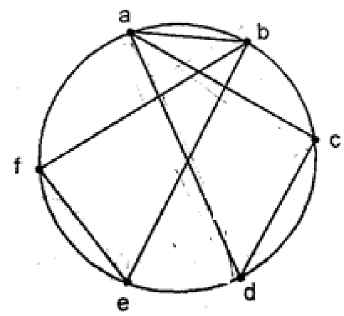


Fig. : Merger Diagram

- 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.
- 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	c, 0	a, 0	b, –	d, 0
b, e, f	f, 1	a, –	b, 1	e, 1

Fig. : Reduced flow table

- Consider *a,c,d =a and b,e,f =b*

States \ DG	00	01	11	10
a	a, 0	a, 0	b, –	a, 0
b	b, 1	a, –	b, 1	b, 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:

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

Q \ DG	00	01	11	10
0	0	0	1	0
1	X	0	X	X

$S = DG$

Q \ DG	00	01	11	10
0	X	X	0	X
1	0	1	0	0

$R = \overline{D} G$

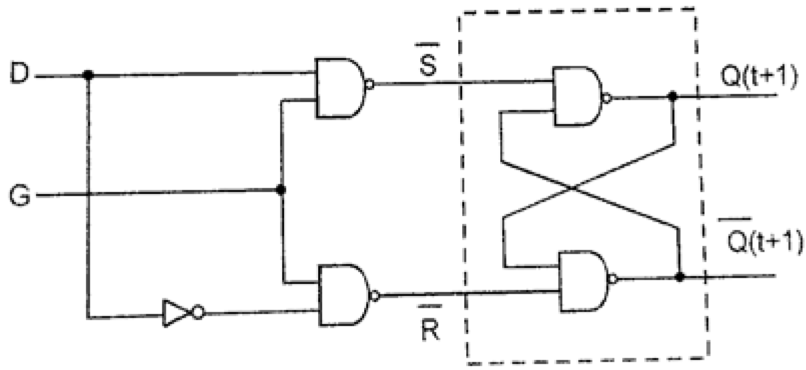


Fig. : Logic diagram using SR latch

****

Design an asynchronous sequential circuit that has two inputs X_2 and X_1 and one output Z. the output is to remain a 0 as long as X_1 is 0. The first change in X_2 that occurs while X_1 is a 1 will cause output Z to be 1. The output Z will remain 1 until Xreturns to 0. (Apr 2018)

Step 1:

States	Inputs X_2	Inputs X_1	Outputs Z	Comments
a	0	0	0	after *b* or *c* or *f*
b	1	0	0	after *a* or *d* or *e*
c	0	1	0	after *a*
d	1	1	0	after *b*
e	1	1	1	after *c* or *f*
f	0	1	1	after *d* or *e*

Step 2: Primitive Flow Table

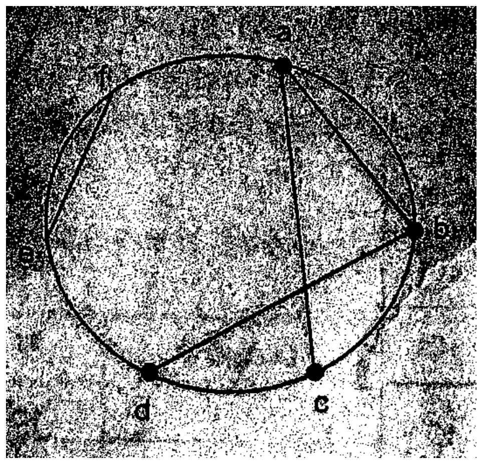
Present State	Input X_2X_1 00	01	11	10
a	a , 0	c, 0	-, -	b, 0
b	a , 0	-, -	d, 0	b, 0
c	a , 0	c, 0	e, -	-, -
d	- , -	f, -	d, 0	b, 0
e	- , -	f, 1	e, 1	b, -
f	a , -	f, 1	e, 1	-, -

Step 3: A reduced flow table is obtained using implication table and merger diagram.

b	✓				
c	✓	d, e ×			
d	c, f ×	✓	c, f × d, e ×		
e	×	×	×	×	
f	×	×	×	×	✓
	a	b	c	d	e

The compatible pairs are (a,b) (a,c) (b,d) (e,f).

The merger diagram is used to find more compatible pairs.



We obtain 4 separate lines.

Therefore, the compatible pairs are again (a,b) (a,c) (b,d) (e,f).

If we remove (a,b), then the remaining pairs (a,c) (b,d) (e,f) covers all the 6 states.

Therefore the reduced flow table is as follows:

States \ x_2x_1	00	01	11	10
a, c	a, 0	c, 0	e, -	b, 0
b, d	a, 0	f, -	d, 0	b, 0
e, f	a, -	f, 1	e, 1	b, -

(or)

States \ x_2x_1	00	01	11	10
S_0	S_0, 0	S_0, 0	S_2, -	S_1, 0
S_1	S_0, 0	S_2, -	S_1, 0	S_1, 0
S_2	S_0, -	S_2, 1	S_2, 1	S_1, -

Step 4: In order to avoid critical race, one more stable state is added and values are assigned for states.

Present state	y_2	y_1	Next State and output for X_2X_1 inputs 00	01	11	10
S_0	0	0	S_0, 0	S_0, 0	S_3, –	S_1, 0
S_1	0	1	S_0, 0	S_2, –	S_1, 0	S_1, 0
S_2	1	1	S_3, –	S_2, 1	S_2, 1	S_1, –
S_3	1	0	S_0, –	–, –	S_2, –	–, –

Step 5: The transition table and output maps are as follows:

Transition Table

$y_2y_1 \backslash x_2x_1$	00	01	11	10
00	00	00	10	01
01	00	11	01	01
11	10	11	11	01
10	00	-	11	-

Output Table Z

$y_2y_1 \backslash x_2x_1$	00	01	11	10
00	0	0	X	0
01	0	X	0	0
11	X	1	1	X
10	X	X	X	X

$Z = y_2$

Map for Y_2

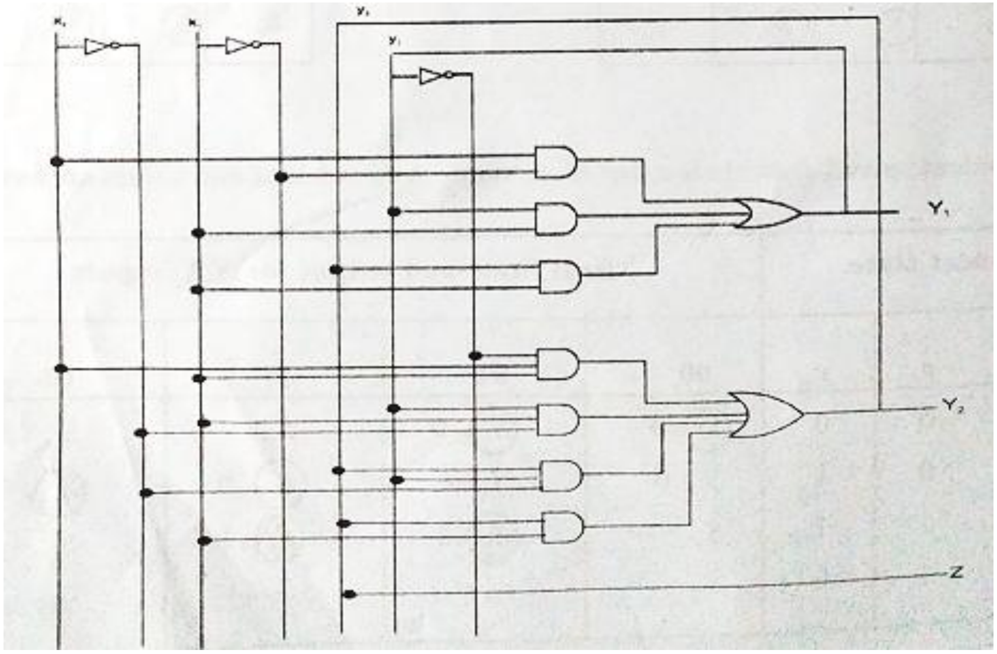
$y_2y_1 \backslash x_2x_1$	00	01	11	10
00	0	0	1	0
01	0	1	0	0
11	1	1	1	0
10	0	X	1	X

$$Y_2 = \bar{y}_1 x_2 x_1 + y_1 \bar{x}_2 x_1 + y_2 y_1 \bar{x}_2 + y_2 x_1$$

Map for Y_1

$y_2y_1 \backslash x_2x_1$	00	01	11	10
00	0	0	0	1
01	0	1	1	1
11	0	1	1	1
10	0	X	1	X

$$Y_1 = x_2 \bar{x}_1 + y_1 x_1 + y_2 x_1$$



****

Design an asynchronous sequential circuit with inputs X_1 and X_2 and one output Z. Initially and at any time if both the inputs are 0, output is equal to 0. When X_1 and X_2 becomes 1, Z becomes 1. When second input also becomes 1, Z = 0; The output stays at 0 until circuit goes back to initial state.

Step 1:

States	Inputs		Outputs	Comments
	X_2	X_1	Z	
a	0	0	0	after b or c
b	0	1	1	after a
c	1	0	1	after a
d	1	1	0	after b or c
e	1	0	0	after d
f	0	1	0	after d

Step 2: Primitive Flow Table

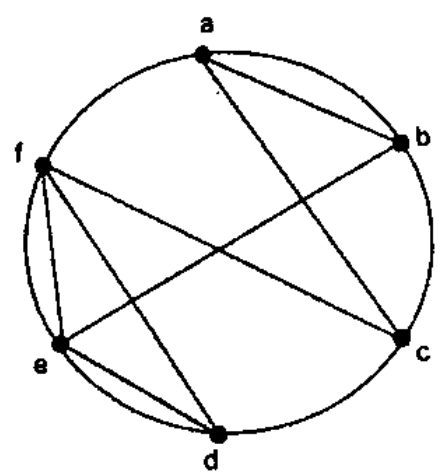
	00	01	11	10
a	a,0	b,--	--,--	c,--
b	a,--	b,1	d,--	--,--
c	a,--	--,--	d,--	c,1
d	--,--	f,--	d,0	e,--
e	a,--	--,--	d,--	e,0
f	a,--	f,0	d,--	--,--

Step 3: A reduced flow table is obtained using Implication table and merger diagram.

b	✓				
c	✓	✓			
d	b, fX c, eX	b, f X	c, e X		
e	c, e X	✓	X	✓	
f	b, f X	X	✓	✓	✓
	a	b	c	d	e

The compatible pairs are (a,b) (a,c) (b,c) (b,e) (c,f) (d,e) (d,f) (e,f).

Merger Diagram:



The Maximal Compatibles are (a,b,c) (d,e,f) (c,f) (b,e).

(c,f) and (b,e) can be removed. Since the remaining terms themselves cover all six states.

States \ X_2X_1	00	01	11	10
a, b, c	a 0	b 1	d,--	c 1
d, e, f	a, –	f 0	d 0	e 0

Step 4 : State Assignment

$$a = b = c = 0 \quad d = e = f = 1.$$

y \ X_2X_1	00	01	11	10
0	0	0	1	0
1	0	1	1	1

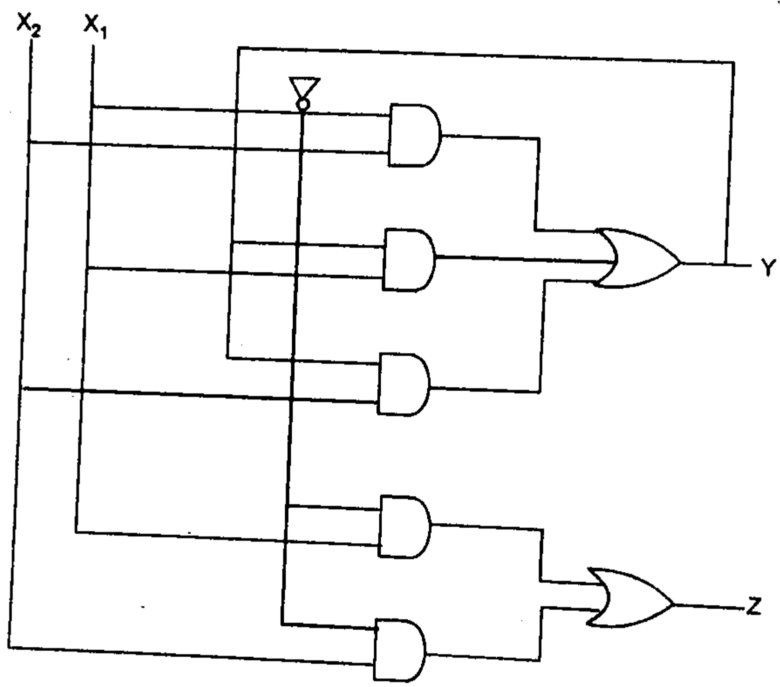
Fig. : Transition Table

y \ X_2X_1	00	01	11	10
0	0	1	X	1
1	X	0	0	0

Fig. : Output Map

$$Y = X_2X_1 + yX_1 + yX_2 \quad Z = \bar{y}X_1 + \bar{y}X_2.$$

Step 5: Logic Diagram



****

Practice Problems:

Design a sequential circuit with two D flip flops A and B and one input X. When X = 0, the state of the circuit remains the same. When X = 1, the circuit goes through the state transitions from 00 to 10 to 11 to 01, back to 00 and then repeats. (Apr 2019)

****

REDUCTION OFSTATE AND FLOW TABLES

Explain in detail about reduction of state and flow tables. ***Dec.2012***

The procedure for reducing the number of internal states in an asynchronous sequential circuit resembles the procedurethat is used forsynchronous circuits.

Implication Table andImplied State

- The state-reduction procedure for completely specified state tables is based on an algorithm that combinestwostatesinastatetableintooneaslongastheycanbeshowntobeequivalent.
- Twostates areequivalentif,foreachpossibleinput,theygiveexactlythesameoutputandgotothesamenext states or to equivalent next states.

State Table to Demonstrate Equivalent States

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	c	b	0	1
b	d	a	0	1
c	a	d	1	0
d	b	d	1	0

- *Considerfor examplethestatetableshowninabovetable.*
- *Thepresentstatesaandbhavethesame outputforthesameinput.*
- *Theirnextstatesarec anddforx=0andband aforx=1.*
- *Ifwe canshow thatthe pair of states(c,d) are equivalent,thenthe pair ofstates(a ,b) willalsobeequivalent,because they willhavethesameorequivalentnextstates.*
- *Whenthisrelationshipexists,wesaythat(a.b)imply (c,d)inthesensethatifaandbare equivalentthenranddhavetobeequivalent.*
- *Similarly,fromthe lasttworowsofabovetable,wefindthatthepairofstales(c,d)implies thepairofstates(a,b).*
- *The characteristicofequivalentstatesisthatif(a,b)imply (c,d)and(c,d)imply (a,b),thenbothpairsof statesare equivalentthatis,aandbare equivalent,andsoare candd.*
- *Asaconsequence,thefour rows oftable can bereduced to tworows bycombining a and b into onestate*

and c and d into asecond state.

State Table to Be Reduced

Present State	Next State		Output	
	x = 0	x = 1	x = 0	x = 1
a	*d*	*b*	0	0
b	*e*	*a*	0	0
c	*g*	*f*	0	1
d	*a*	*d*	1	0
e	*a*	*d*	1	0
f	*c*	*b*	0	0
g	*a*	*e*	1	0

- The implicationtableisshowninFig.Ontheleftsidealongtheverticalarelistedallthestates definedinthestatetableexceptthefirstandacrossthebottomhorizontallyarelistedallthestates exceptthelast.
- Theresultisadisplayofallpossiblecombinationsoftwostareswithasquareplacedin theintersectionofarowandacolumnwhere thetwostatescanbetestedfor equivalence.Twostates havingdifferent outputs forthe same input arenot equivalent.

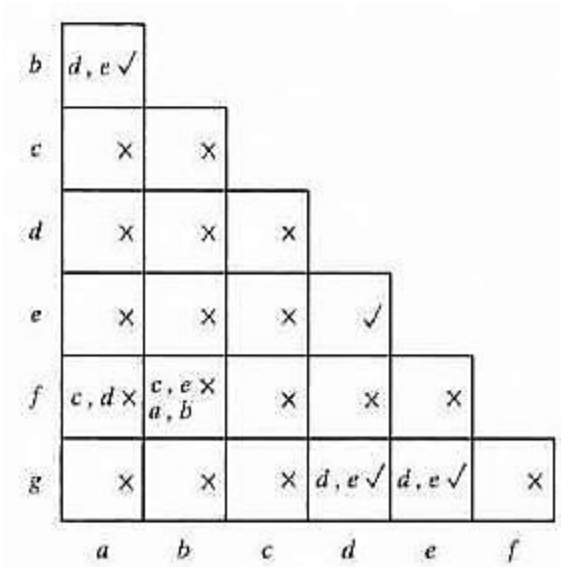


Fig: Implication table

- Twostatesthatarenotequivalentaremarkedwithacross[X]inthecorrespondingsquarewhereas theirequivalenceisrecordedwithacheckmark(✓').Someofthesquareshaveentriesofimplied statesthatmustbeinvestigatedfurthertodeterminewhetherthey areequivalent.
- Thustablecanbe reducedfromsevenstatestofour onefor eachmember ofthe precedingpartition.Thereducedstate tableis obtained byreplacingstateb bya and states e andgbydand it is shown below,

Reduced State Table

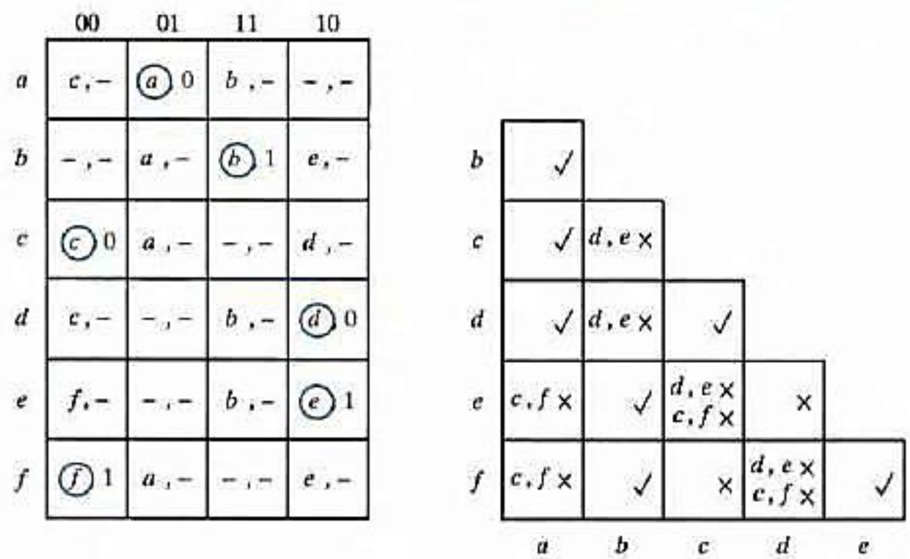
Present State	Next State		Output	
	$x=0$	$x=1$	$x=0$	$x=1$
a	d	a	0	0
c	d	f	0	1
d	a	d	1	0
f	c	a	0	0

Merging oftheFlowTable

- Incompletelyspecifiedstatescanbecombinedtoreducethenumberofstateintheflowtable.Such stares cannotbe called equivalentbecause theformaldefinitionof equivalencerequiresthatalloutputs and nextstatesbe specified forallinputs.
- Instead, twoincompletely specified states that can be combinedaresaidtobe*Compatible*.Theprocessthatmustbeappliedinordertofindasuitablegroup of compatibles forthe purpose ofmerging a flow table can bedivided into threesteps:
    1. *Determine allcompatible pairs byusingthe implication table.*
    2. *Find themaximal compatibles with the useof amergerdiagram.*
    3. *Find a minimal collection of compatibles that covers allthe states andis closed.*

CompatiblePairs-

- Theentriesineachsquareofprimitiveflowtablerepresentthenextstate andoutputThedashesrepresenttheunspecifiedstatesoroutputs.
- Theimplicationtable isusedtofmdcompatiblestatesjustasitisusedtofindequivalentstalesinthecompletely specifiedcase.Theonly differenceisthat,whencomparingrows,weareatlibertytoadjustthedashestofitany desired condition.



	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, –

(a) Primitive flow table

	a	b	c	d	e
b	✓				
c	✓	d, e ×			
d	✓	d, e ×	✓		
e	c, f ×	✓	d, e × c, f ×	×	
f	c, f ×	✓	×	d, e × c, f ×	✓

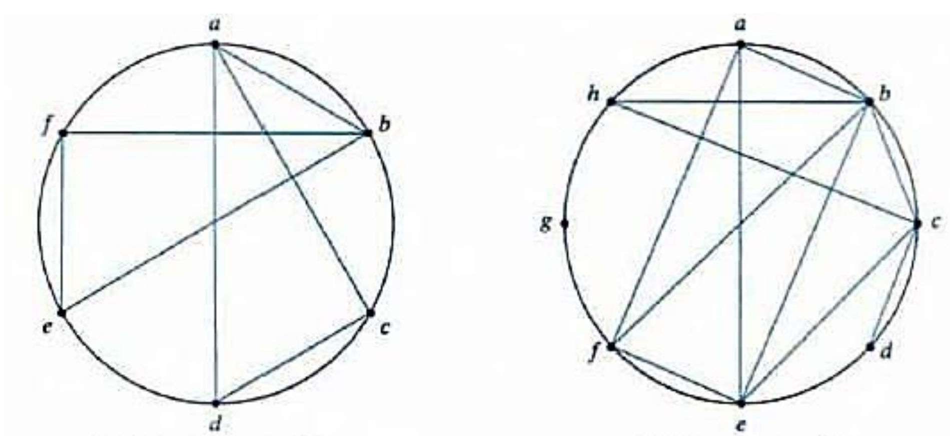
(b) Implication table

The compatiblepairsare,

$$(a,b)(a,c)(a,d)(b,e)(b,f)(c,d)(e,f)$$

MaximalCompatibles

- The*maximal compatible*isagroupofcompatiblesthatcontainsallthe possiblecombinations of compatible states. The maximal compatiblecanbeobtained from a merger diagram.
- The merger diagramisagraphinwhicheachstateisrepresented by adotplacedalongthecircumferenceofa circle. Linesaredrawnbetweenany twocorrespondingdotsthatformacompatiblepair.
- Allpossible compatiblescanbeobtainedfromthemergerdiagramby observingthegeometricalpatternsinwhich statesareconnectedtoeachother.Anisolateddotrepresentsastatethatisnotcompatiblewithany otherstate. A line representsa compatiblepair.Atriangleconstitutes a compatiblewith threestates.



(a) Maximal compatible:
(a, b) (a, c, d) (b, e, f)

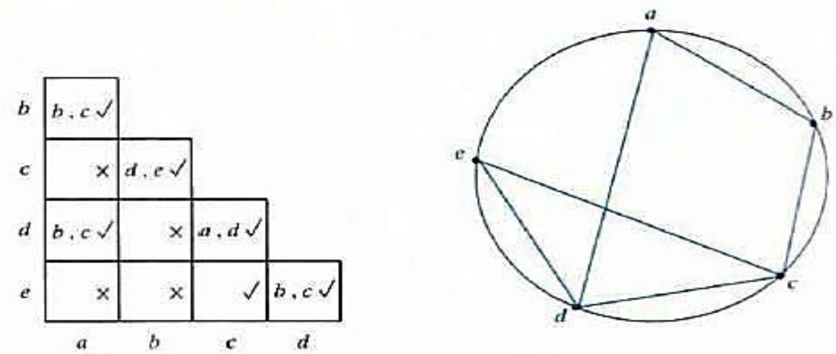
(b) Maximal compatible:
(a, b, e, f) (b, c, h) (c, d) (g)

Themaximal compatiblesof fig (a)are $(a, b)(a, c, d)(b, e, f)$

Themaximal compatiblesof fig (b)are $(a, b, e, f)(b, c, h)(c, d)(g)$

Closed-Covering Condition

- Theconditionthatmustbesatisfiedformergingrowsisthatthesetofchosencompatiblesmust*cover* allthestatesandmustbe*closed*.
- Thesetwillcoverallthestatesifitincludesallthestatesofthe originalstatetable.The closureconditionis satisfiedifthereare noimpliedstatesorif theimplied states are includedwithintheset.Aclosedsetofcompatiblesthatcoversallthestatesiscalleda*closed covering*.



(a) Implication table

(b) Merger diagram

Compatibles	(a, b)	(a, d)	(b, c)	(c, d, e)
Implied states	(b, c)	(b, c)	(d, e)	(a, d) (b, c)

(c) Closure table

****

RACE -FREE STATE ASSIGNMENT

Explain in detail about race -free state assignment. ***May 2012,Dec. 2014***

- Once areducedflow tablehasbeenderivedfor anasynchronoussequentialcircuit,thenextstepinthe designistoassignbinary variablestoeachstablestate.
- Thisassignmentresultsinthetransformationof theflowtableintoitsequivalenttransitiontable.Theprimaryobjectiveinchoosing aproperbinary state assignmentisthepreventionofcriticalraces.
- Criticalracescanbeavoidedbymakingabinarystate assignmentinsuchawaythatonly onevariablechangesatanygiventimewhenastatetransitionoccurs in the flow table.

Three-RowFlow-Table Example

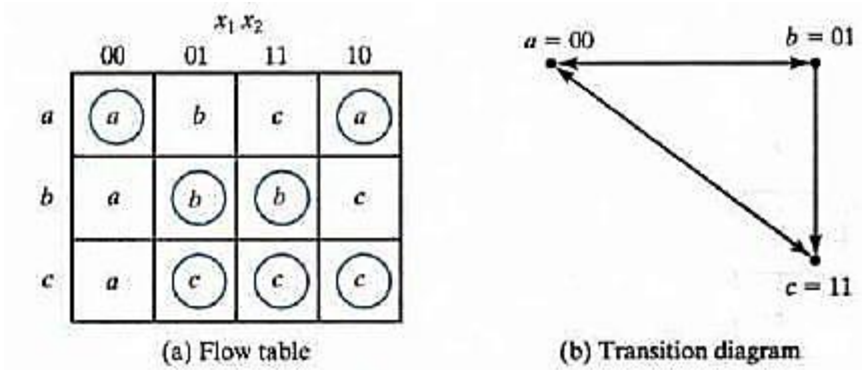


Fig: Three rowflowtable example

- Toavoidcriticalraces,wemustfindabinarystateassignmentsuchthatonly onebinaryvariable changesduring eachstatetransition.
- Anattempttofindsuchanassignmentisshowninthetransition diagram.State*a* isassignedbinary 00,andstate*c*isassignedbinary 11.
- Thisassignmentwillcausea criticalraceduringthetransitionfrom*a*to*c*becausetherearetwochangesinthebinary statevariables andthetransitionfrom*a*to*c*may occurdirectlyorpassthrough*b*.
- Notethatthetransitionfrom*c*to*a*alsocausesaracecondition,butitisnoncriticalbecausethetransitiondoesn otpassthroughother states

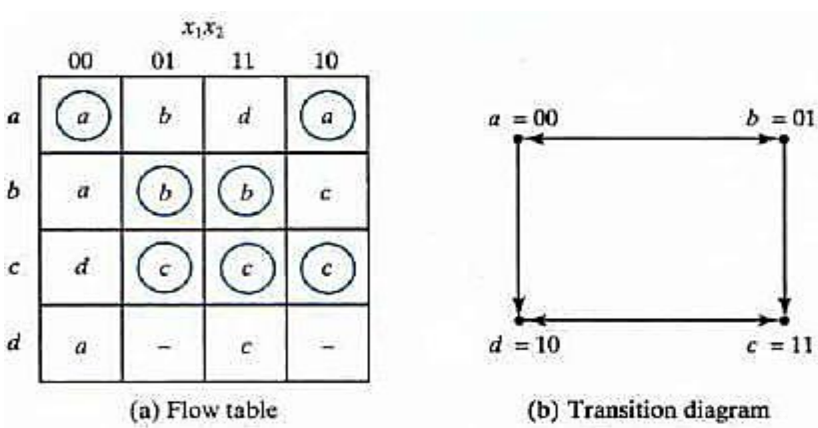


Fig:Flowtable with an extrarow

- Arace- freeassignmentcanbeobtainedifweaddanextra rowtotheflow table.Theuseofa fourthrow doesnotincreasethenumberofbinarystatevariables,butitallowstheformationof cycles betweentwo stablestates.
- Thetransitiontablecorrespondingtotheflowtablewiththeindicatedbinarystateassignmentisshown inFig.

- Thetwodashesinrowdrepresentunspecifiedstatesthatcanbeconsidereddon't-care conditions.
- However,caremustbetakennottoassign10tothesesquares,inordertoavoidthepossibility ofan unwanted stablestatebeingestablishedin thefourth row.

	x_1x_2 00	01	11	10
a = 00	00	01	10	00
b = 01	00	01	01	11
c = 11	10	11	11	11
d = 10	00	–	11	–

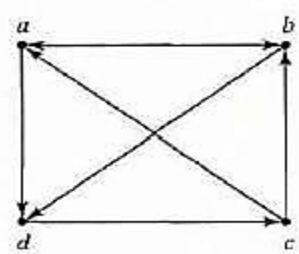
Fig: Transitiontable

Four-RowFlow-Table Example

- A flow tablewith four rows requiresaminimum of two statevariables.
- Although arace-freeassignment issometimespossiblewithonly twobinary statevariables,inmany casestherequirementofextrarows to avoid critical races willdictate theuse of threebinarystate variables

	00	01	11	10
a	b	a	d	a
b	b	d	b	a
c	c	a	b	c
d	c	d	d	c

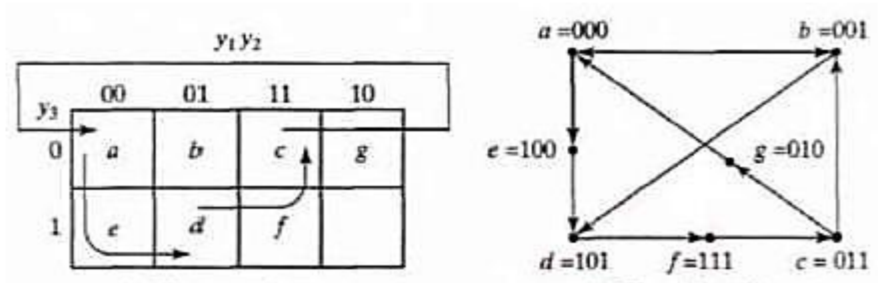
(a) Flow table



(b) Transition diagram

Fig:Four-rowflow-table example

- Thefollowing figureshowsastateassignmentmapthatissuitableforany four-rowflowtable.States*a*, *b*,*c*and*d*aretheoriginalstatesand*e*,*f*and*g*areextrastates.
- Thetransitionfrom*a*to*d*mustbe directedthroughtheextrastate*e*toproduceacyclesothatonly onebinaryvariablechangesatatime. Similarly,thetransitionfrom*c*to*a*isdirectedthrough*g*andthetransitionfrom*d*to*c*goesthrough*f*.
- 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.

	00	01	11	10
000 = a	b	a	e	a
001 = b	b	d	b	a
011 = c	c	g	b	c
010 = g	–	a	–	–
110 –	–	–	–	–
111 = f	c	–	–	c
101 = d	f	d	d	f
100 = e	–	–	d	–

Fig: Stateassignment to modified flowtable

Multiple-RowMethod

- Themethodformakingrace-freestaleassignmentsbyaddingextrarowsintheflowtableisreferredto asthe *shared-row*method.
- A second methodcalled the*multiple-row*methodisnotasefficient, butis easiertoapply.Inmultiple-rowassignmenteachstateintheoriginalrowtableisreplacedby twoor more combinationsofstate variables.

(a) Binary assignment

y_1 \ y_2y_3	00	01	11	10
0	a_1	b_1	c_1	d_1
1	c_2	d_2	a_2	b_2

(b) Flow table

	00	01	11	10
$000 = a_1$	b_1	a_1	d_1	a_1
$111 = a_2$	b_2	a_2	d_2	a_2
$001 = b_1$	b_1	d_2	b_1	a_1
$110 = b_2$	b_2	d_1	b_2	a_2
$011 = c_1$	c_1	a_2	b_1	c_1
$100 = c_2$	c_2	a_1	b_2	c_2
$010 = d_1$	c_1	d_1	d_1	c_1
$101 = d_2$	c_2	d_2	d_2	c_2

Fig:Multiple rowassignment

- Therearetwobinary statevariablesforeachstablestate,eachvariablebeingthelogicalcomplementof the other.
- Forexample,the originalstatea is replacedwithtwoequivalentstates a_1=000anda_2=111. Theoutputvalues,notshownheremustbethesameina$_1$anda$_2$.
- Notethata$_1$ isadjacenttob$_1$,c$_2$and d$_1$,anda$_2$ isadjacenttoc$_1$, b$_2$andd$_2$,andsimilarlyeachstateisadjacenttothreestateswithdifferent letterdesignations.
- Theexpandedtableisformedbyreplacingeachrowoftheoriginaltablewithtworows.Inthemultiple-rowassignment,thechange fromonestablestate10another willalwayscauseachangeofonly one binarystate variable.
- Each stable stalehastwo binaryassignmentswith exactlythe sameoutput.

****

Practice Problems:

A sequential Circuit with two D flip flops A and B, two inputs X and Y, and one output Z is specified by the following input equations:

$$A(t+1) = x'y + xA$$
$$B(t+1) = x'B + xA$$
$$Z = B$$

Draw the logic diagram of the circuit. Derive the state table and state diagram and state whether it is a Mealy or a Moore machine. (Apr 2019)

****

HAZARDS

Discuss about the possible hazards and methods to avoid them in combinational circuits. (or) Explain in detail about hazards. May 2011, Dec. 2013, Apr 2017, Nov 2017, Apr 2018, Nov 2018, Apr 2019

➢ ***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.

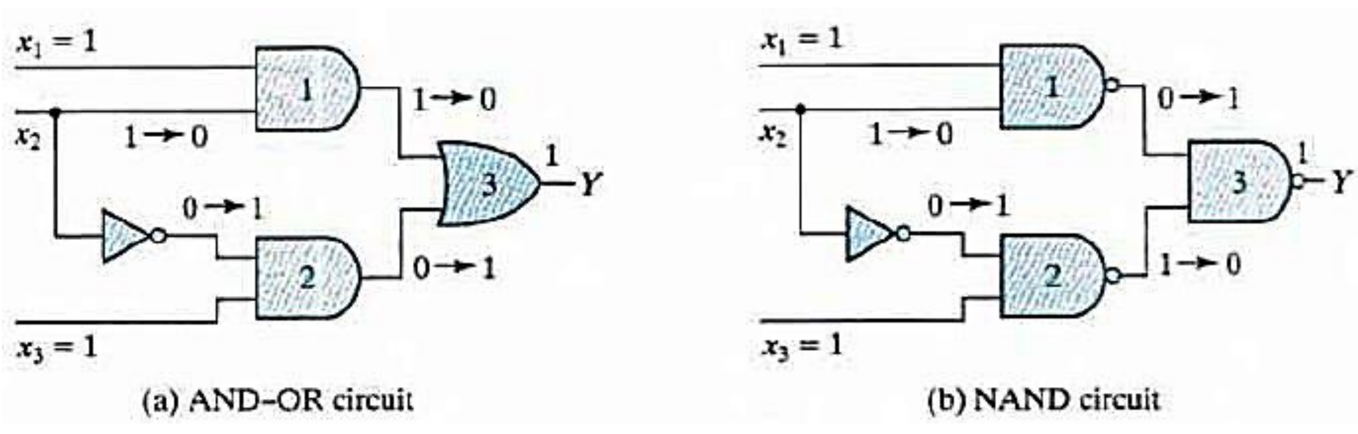


Fig: Circuits with Hazards

➢ Assumethatallthreeinputsareinitially equalto1.Thiscausestheoutputofgate110be1,thatofgate 2tobe0andthatofthecircuittobe1.Nowconsiderachangeinx2from1to0.

➢ Thentheoutputof gate1changesto0and thatofgate2changesto1,leaving theoutputat 1.However,theoutputmay 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, theCircuit 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 static1-hazard and the second case as static 0-hazard.

- Athirdtypeofhazard,knownas**dynamichazard**,causestheoutputtochangethreeormoretimes when itshould changefrom1 to 0 or from 0 to 1.

Fig: Types ofhazards

- The change in $x_2$from1to0movesthecircuitfromminterm111tominterm101.The hazard exists because the changein input results in a different product term coveringthe twominterm.

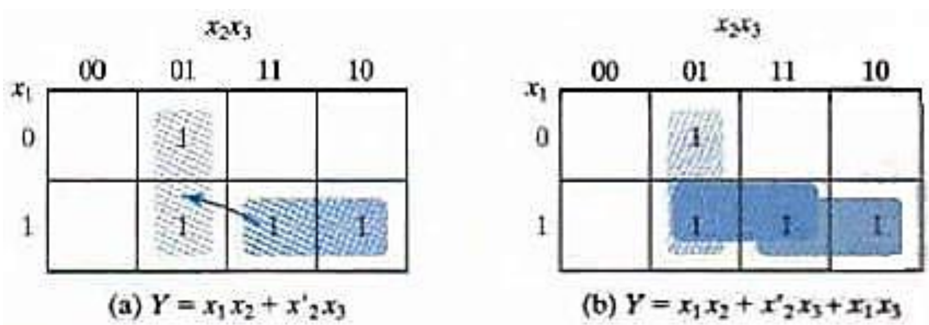
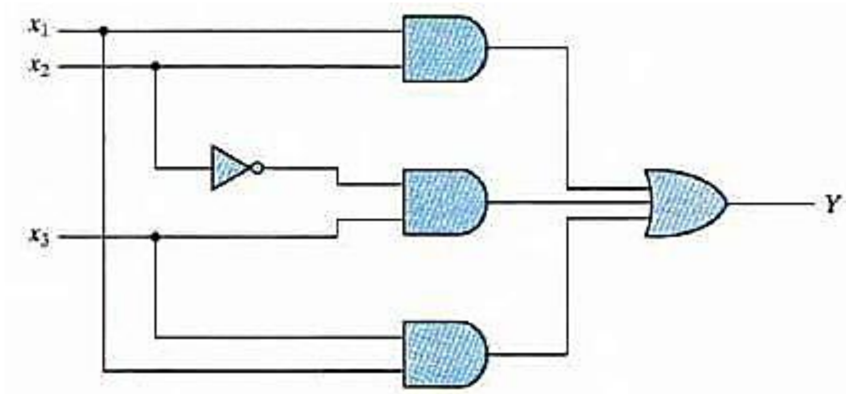


Fig: Illustrates hazardandits removal

- Minterm111iscoveredby theproducttermimplementedingate1andminterm101iscoveredbythe producttermimplementedingate2.
- The remedy for eliminating a hazard is to enclose the two min terms withanotherproducttermthatoverlapsbothgroupings. Thehazard-freecircuitobtainedbysucha configurationisshowninfigure below.
- The extra gate inthecircuitgeneratesthe producttermx_1x_3.In

general,hazardsincombinationalcircuitscanberemoved by coveringany twomintermsthatmay producea hazardwithaproducttermcommontoboth.

- Theremovalofhazardsrequirestheadditionof redundantgates to thecircuit.



Hazards in SequentialCircuits

- Innormalcombinational-circuitdesignassociatedwithsynchronoussequentialcircuits,hazardsareof noconcern,sincemomentaryerroneoussignalsare notgenerallytroublesome.
- However,ifamomentary incorrectsignalisfed backinanasynchronoussequentialcircuit,itmay causethecircuittogotothe wrongstable state.

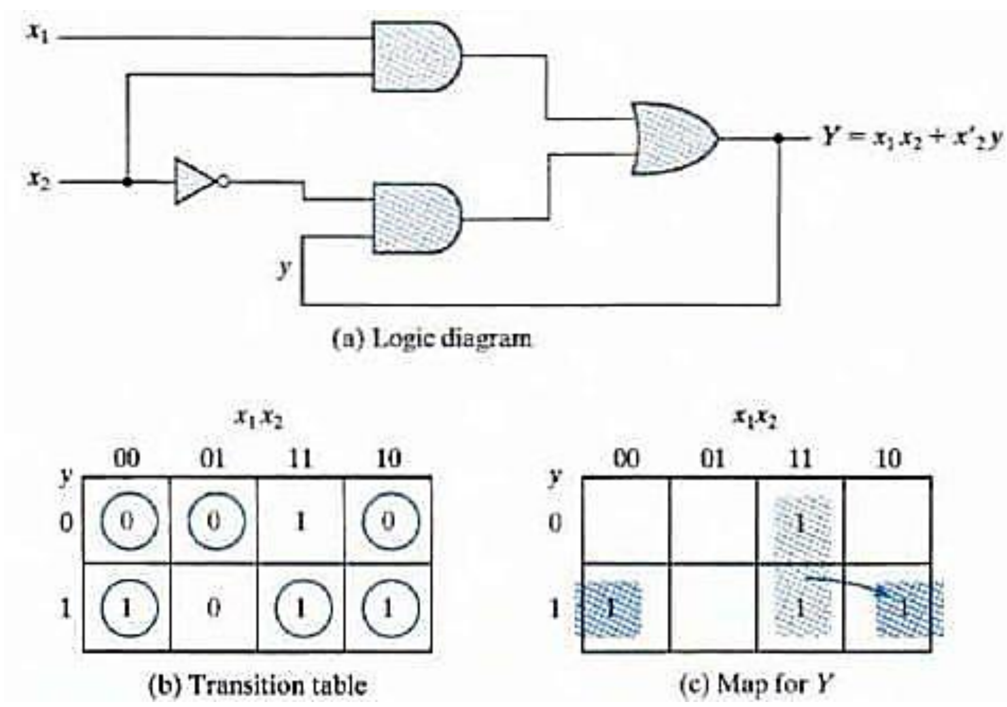


Fig: HazardinanAsynchronous sequential circuit

- Ifthe circuitis in total stable stateyx$_1$x$_2$=111 and inputx2 changesfromIto0, the next total stablestate shouldbe110.However,because ofthehazard,outputYmay goto0momentarily.
- Ifthisfalsesignal feedsbackintogate2beforetheoutputoftheinvertergoesto1,theoutputofgate2willremainat0

andthecircuitwillswitchtotheincorrecttotalstablestate 010.

- Thismalfunctioncanbe eliminatedby adding an extragate.

Essential Hazards

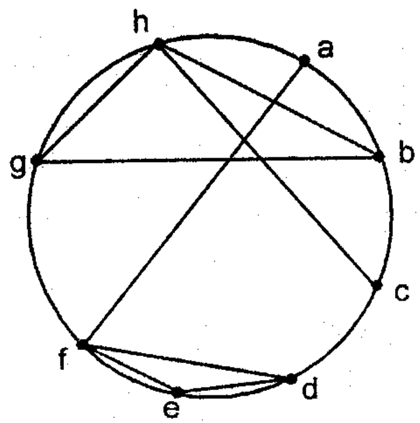
- **Essential**hazardiscausedby unequaldelaysalongtwoormorepathsthatoriginatefromthesame input.
- Anexcessivedelaythroughaninvertercircuitincomparisontothedelayassociatedwiththe feedback path maycausesuchahazard.
- Essentialhazards cannotbecorrectedby addingredundantgatesasinstatichazards.The problemthat they imposecanbecorrectedbyadjustingtheamountofdelay intheaffectedpath.
- Toavoidessential hazards,eachfeedbackloopmustbehandledwithindividualcaretoensurethatthedelayinthe feedbackpathislongenoughcomparedwithdelaysofothersignalsthatoriginate fromtheinput terminals.

****

	a	b	c	d	e	f	g
b	a,c X						
c	X	b,d X					
d	b,d X	X	a,c X				
e	b,d X	e,g X b,d X	f,h X	✓			
f	✓	e,g X a,c X	f,h X a,c X	✓	✓		
g	f,h X	✓	b,d X	e,g X b,d X	X	e,g X f,h X	
h	f,h X a,c X	✓	✓	e,g X a,c X	e,g X f,h X	X	✓

The compatible pairs are (*a, f*) (*b, g*) (*b, h*) (*c, h*) (*e, f*) (*g, h*) (*d, e*) (*d, f*).

Merger Diagram



The maximal compatibles are (*a, f*) (*c, h*) (*b, g, h*) (*d, e, f*).

The reduced flow table is given below.

Present State \ TC	00	01	11	10
a,f	e, -	f , 0	a, 0	b -
c,h	g, -	h , 1	c , 1	d -
b,g,h	g , 1	h ,1	c, -	b 1
d,e,f	e ,0	f , 0	a, -	d , 0

TC	00	01	11	10
a	d, -	a , 0	a, 0	c -
b	c, -	b , 1	b , 1	d -
c	c, 1	c,1	b, -	c 1
d	d ,0	d , 0	a, -	d , 0

Step 3: A race free binary state assignment is made. Transition table and output map is obtained.

Transition Table

y_1y_2 \ TC	00	01	11	10
00	10	00	00	01
01	01	01	11	01
11	01	11	11	10
10	10	10	00	10

Output Map

y_1y_2 \ TC	00	01	11	10
00	X	0	0	X
01	1	1	X	1
11	X	1	1	X
10	0	0	X	0

$$Q = y_2$$

Step 4: Logic diagram.

Map for Y_1

y_1 \ TC	00	01	11	10
0	1	0	0	0
0	0	0	1	0
1	0	1	1	1
1	1	1	0	1

Map for Y_2

y_2 \ TC	00	01	11	10
0	0	0	0	1
1	1	1	1	1
1	1	1	1	0
0	0	0	0	0

Use SR latches.

The excitation table for SR latch is

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

y_1y_2 \ TC	00	01	11	10
00	1	0	0	0
01	0	0	1	0
11	0	X	X	X
10	X	X	0	X

$$S_1 = \bar{y}_2\bar{T}\bar{C} + y_2TC$$

y_1y_2 \ TC	00	01	11	10
00	0	X	X	X
01	X	X	0	X
11	1	0	0	0
10	0	0	1	0

$$R_1 = y_2\bar{T}\bar{C} + \bar{y}_2TC$$

y_1y_2 \ TC	00	01	11	10
00	0	0	0	1
01	X	X	X	X
11	X	X	X	0
10	0	0	0	0

$$S_2 = \bar{y}_1T\bar{C}$$

y_1y_2 \ TC	00	01	11	10
00	X	X	X	0
01	0	0	0	0
11	0	0	0	1
10	X	X	X	X

$$R_2 = y_1T\bar{C}$$

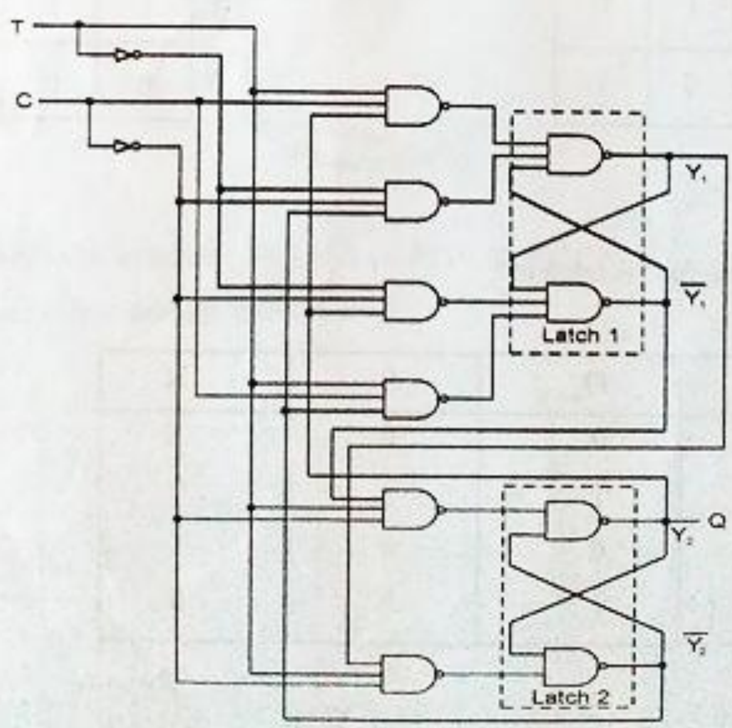


Fig. 4.17: Logic diagram of Example 4.3

****

Problems on hazards: (Nov 2018)

Example

Give hazard free realization for the following Boolean functions.

$$f(A, B, C, D) = \sum m(1, 3, 6, 7, 13, 15).$$

Solution :

The simplified expression for the given function can be obtained using k-map.

AB \ CD	00	01	11	10
00	0	1	1	0
01	0	0	1	1
11	0	1	1	0
10	0	0	0	0

The simplified expression is,

$$f(A, B, C, D) = \bar{A}\bar{B}D + \bar{A}BC + ABD$$

But to remove hazards, we have to include another two terms, which is shown in dotted lines in map.

$$f(A, B, C, D) = \bar{A}\bar{B}D + \bar{A}BC + ABD + \bar{A}CD + BCD$$

The logic diagram is as shown in Figure 4.38.

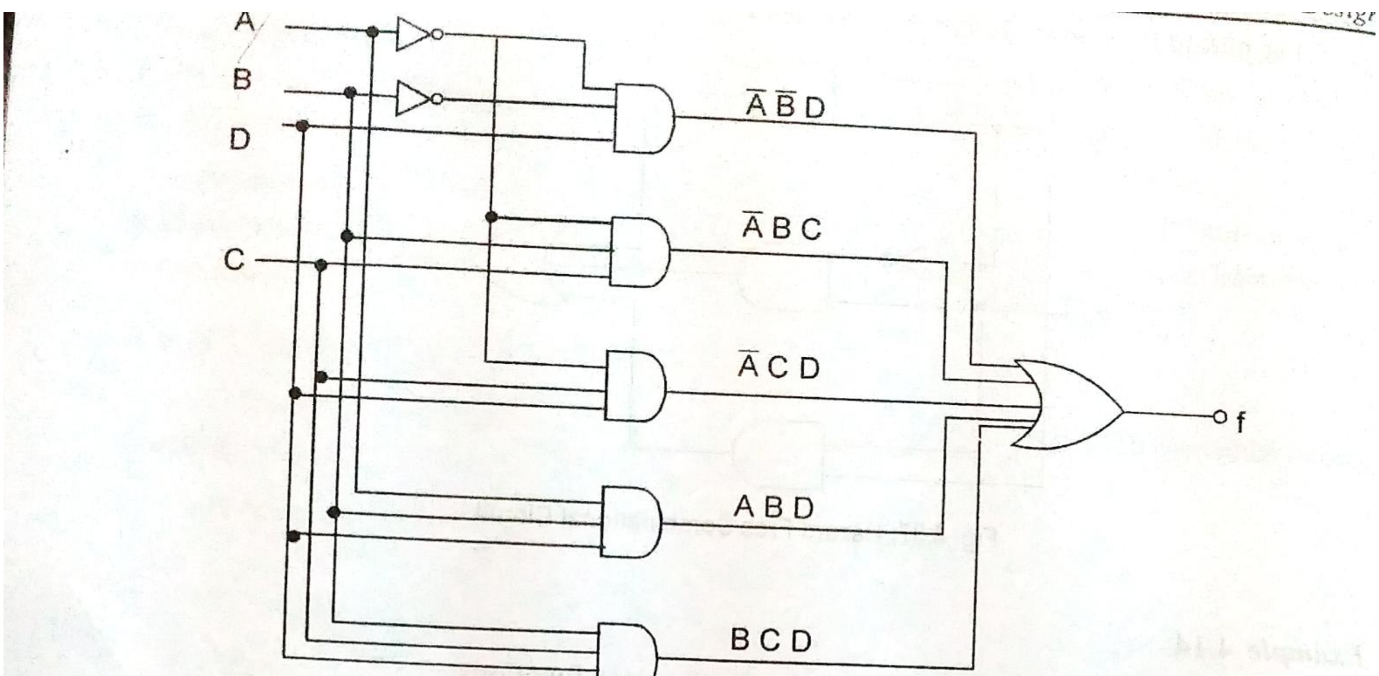


Fig. 4.38 : Logic diagram of Example 4.14

Example

Give hazard free realization for the following Boolean functions.

$$F(A, B, C, D) = \sum m(0, 1, 5, 6, 7, 9, 11).$$

✍ ***Solution :***

The simplified expression for the given function can be obtained using K-map.

AB \ CD	$\overline{C}\,\overline{D}$ 00	$\overline{C}D$ 01	CD 11	$C\overline{D}$ 10
$\overline{A}\,\overline{B}$ 00	1	1	0	0
$\overline{A}B$ 01	0	1	1	1
AB 11	0	0	0	0
$A\overline{B}$ 10	0	1	1	0

The simplified expression is,

$$F = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}BD + \overline{A}BC + A\overline{B}D$$

But to remove hazards, we have to include two more terms, that are shown as dotted lines in K-map.

$$F = \bar{A}\,\bar{B}\,\bar{C} + \bar{A}BD + \bar{A}BC + A\bar{B}D + \bar{A}\,\bar{C}D + \bar{B}\,\bar{C}D$$

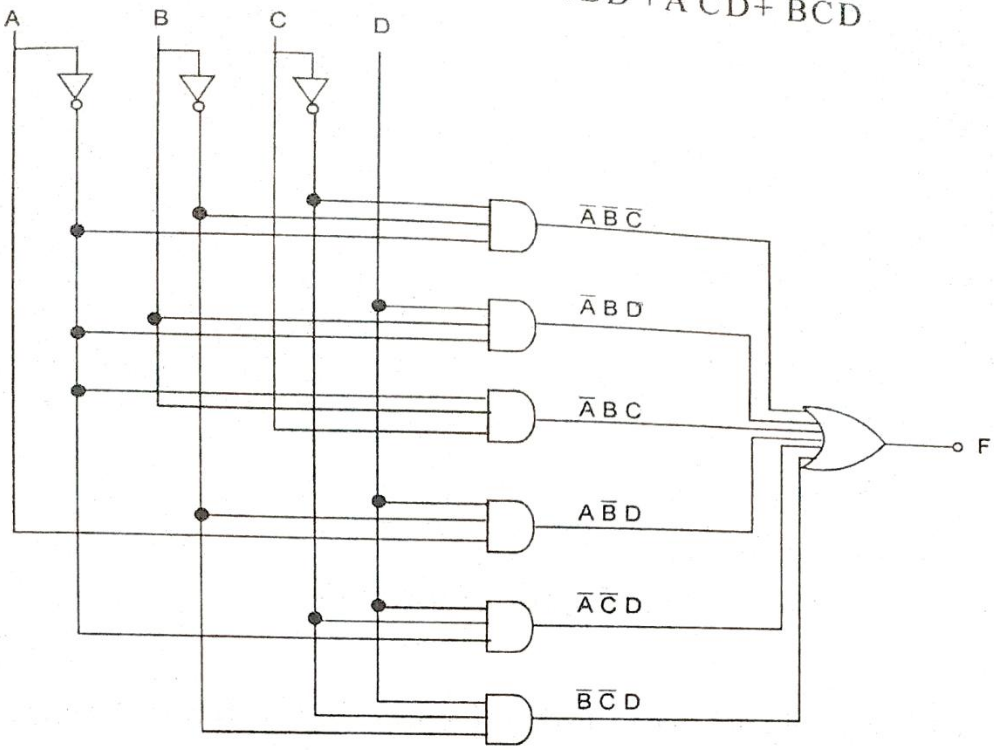


Fig. 4.39 : Logic diagram of Example 4.15

Example

Give hazard free realization for the following Boolean function

$$F(I, J, K, L) = \sum m(1, 3, 4, 5, 6, 7, 9, 11, 15).$$

Solution :

The simplified expression for the given function can be obtained using K-map.

IJ \ KL	$\bar{K}\bar{L}$ 00	$\bar{K}L$ 01	KL 11	$K\bar{L}$ 10
$\bar{I}\bar{J}$ 00	0	1	1	0
$\bar{I}J$ 01	1	1	1	1
IJ 11	0	0	1	0
$I\bar{J}$ 10	0	1	1	0

The simplified expression is,

$$F = \bar{I}J + KL + \bar{J}L$$

But to remove hazards, one more redundant quad have to be included, which is shown as dotted lines in K-map.

$$F = \bar{I}J + KL + \bar{J}L + \bar{I}L$$

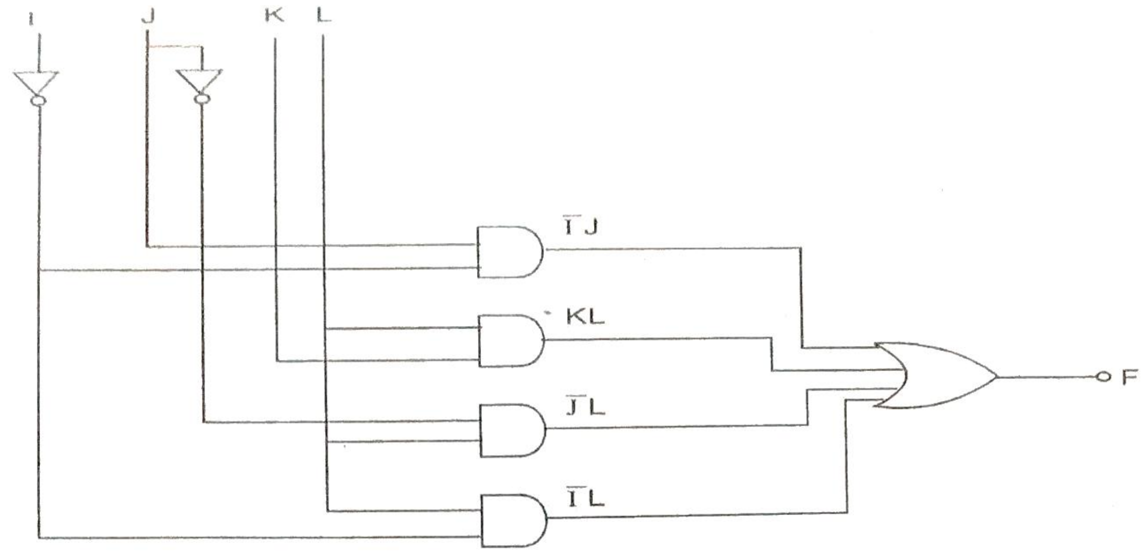


Fig. 4.40 : Logic Diagram of Example 4.16

***Example* 4.17**

Find a static and dynamic hazard free realization for the following function using

(i) NAND gates. **(ii)** NOR gates.

$$f(a, b, c, d) = \sum m(1, 5, 7, 14, 15).$$

Solution :

The simplified expression for the given function can be obtained using K-map.

ab \ cd	00	01	11	10
00	0	1	0	0
01	0	1	1	0
11	0	0	1	1
10	0	0	0	0

The simplified expression is,

$$f = \bar{a}\bar{c}d + \bar{a}bd + abc$$

But to remove static and dynamic hazards one more term has to be included, which is shown as dotted lines in K-map.

$$f = \bar{a}\bar{c}d + \bar{a}bd + abc + bcd$$

This can be implemented using NAND gates as follows.

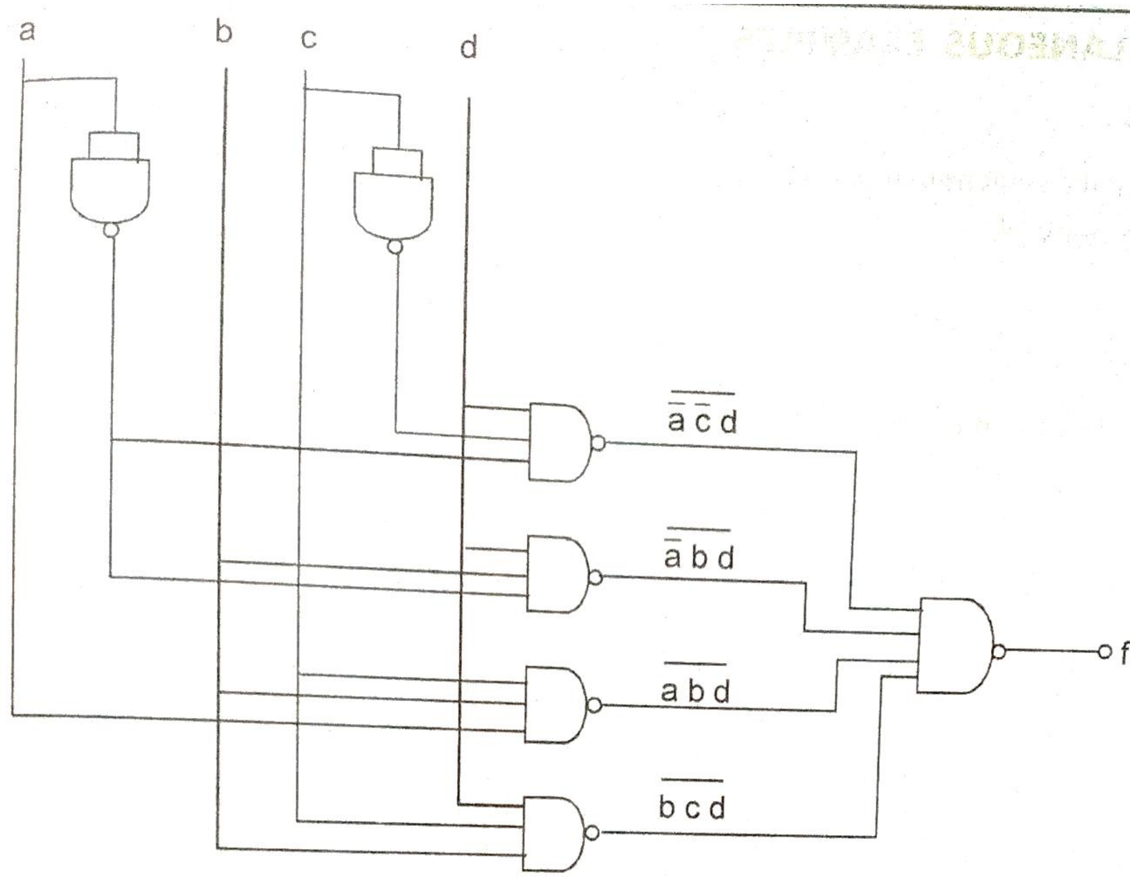


Fig. 4.41: Hazard free circuit using NAND gates

The above expression can also be implemented using NOR gates, as shown in Figure 4.40.

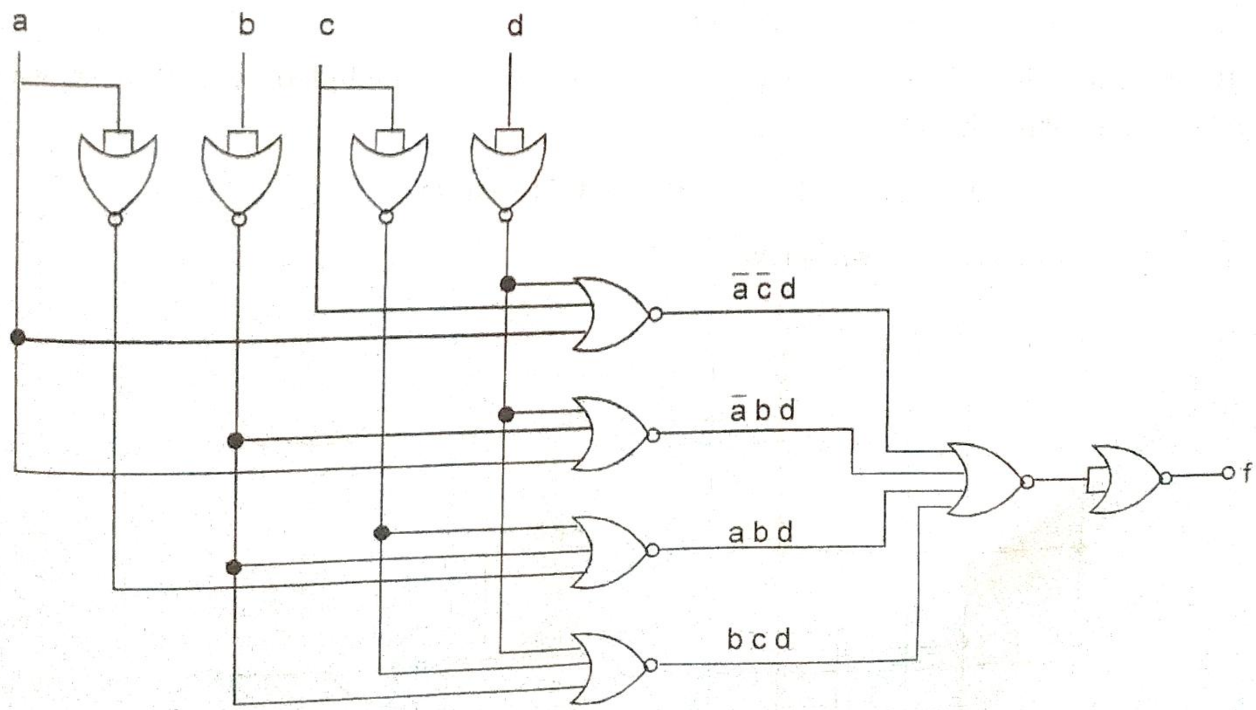


Fig. 4.42: Hazard free circuit using NOR gates

RAM – Memory Decoding – Error Detection and Correction - ROM -Programmable Logic Array – Programmable Array Logic – Sequential Programmable Devices.

51 INTRODUCTION

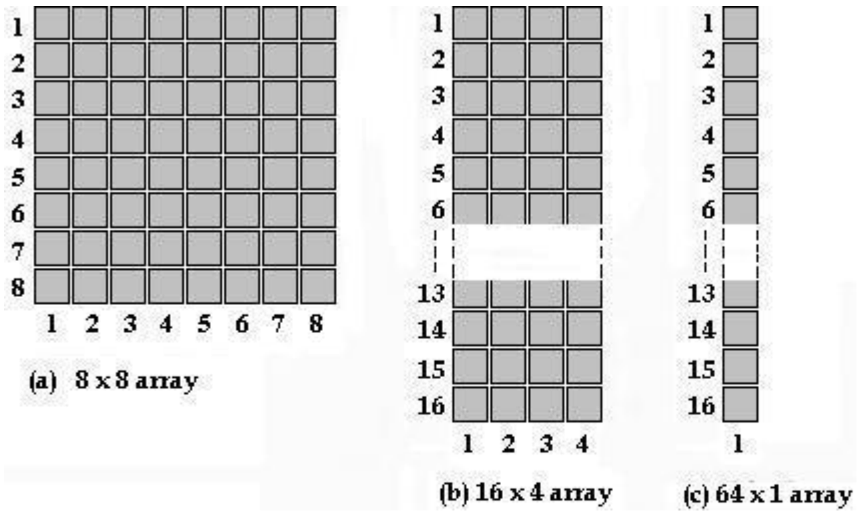
A memory unit is a collection of storage cells with associated circuits needed to transfer information in and out of the device The binary information is transferred for storage and from which information is available when needed for processing When data processing takes place, information from the memory is transferred to selected registers in the processing unit Intermediate and final results obtained in the processing unit are transferred back to be stored in memory

52 Units of Binary Data: Bits, Bytes, Nibbles and Words

As a rule, memories store data in units that have from one to eight bits The smallest unit of binary data is the **bit** In many applications, data are handled in an 8-bit unit called a **byte** or in multiples of 8-bit units The byte can be split into two 4-bit units that are called **nibbles** A complete unit of information is called a **word** and generally consists of one or more bytes Some memories store data in 9-bit groups; a 9-bit group consists of a byte plus a parity bit

53 Basic Semiconductor Memory Array

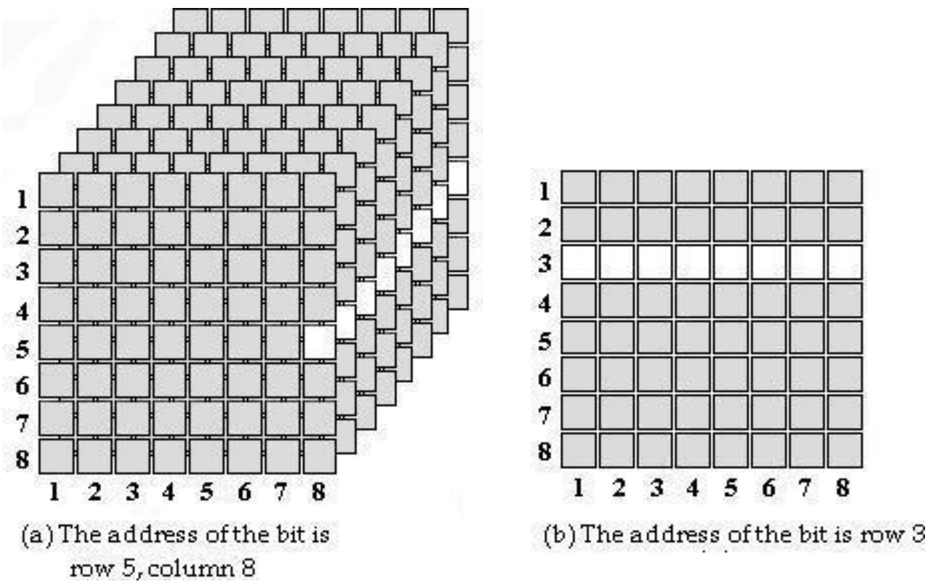
Each storage element in a memory can retain either a 1 or a 0 and is called a **cell** Memories are made up of arrays of cells, as illustrated in Figure below using 64 cells as an example Each block in the memory array represents one storage cell, and its location can be identified by specifying a row and a column



A 64-cell memory array organized in three different ways

54 Memory Address and Capacity

The *location* of a unit of data in a memory array is called its **address** For example, in Figure (a), the address of a bit in the 3-dimensional array is specified by the row and column In Figure (b), the address of a byte is specified only by the row in the 2-dimensional array So, as you can see, the address depends on how the memory is organized into units of data Personal computers have random-access memories organized in bytes This means that the smallest group of bits that can be addressed is eight



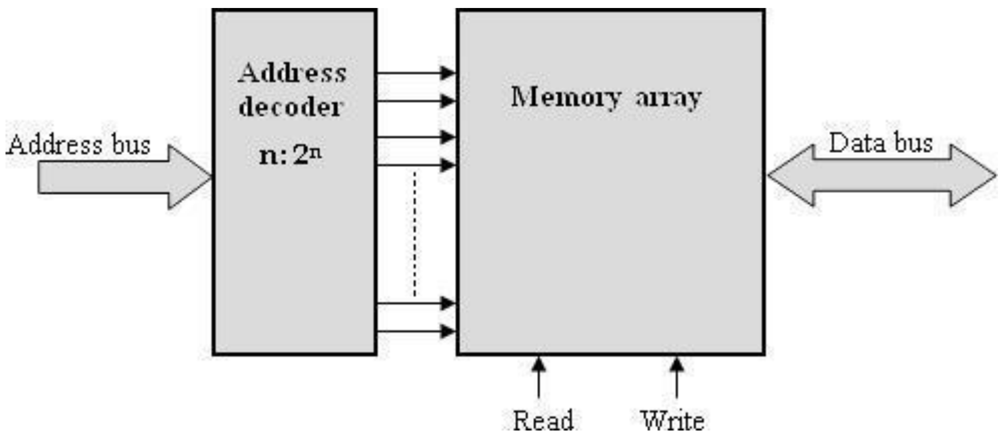
Examples of memory address

The **capacity** of a memory is the total number of data units that can be stored For example, in the bit-organized memory array in Figure (a), the capacity is 64 bits In the byte-organized memory array in Figure (b), the capacity is 8 bytes, which is also 64 bits Computer memories typically have 256 MB (megabyte) or more of internal memory

55 Basic Memory Operations

Since a memory stores binary data, data must be put into the memory and data must be copied from the memory when needed The write operation puts data into a specified address in the memory, and the read operation copies data out of a specified address in the memory The addressing operation, which is part of both the write and the read operations, selects the specified memory address

Data units go into the memory during a write operation and come out of the memory during a read operation on a set of lines called the *data bus* As indicated in Figure, the data bus is bidirectional, which means that data can go in either directional (into the memory or out of the memory)



Block diagram of memory operation

For a write or a read operation, an address is selected by placing a binary code representing the desired address on a set of lines called the address bus The address code is decoded internally and the appropriate address is selected The number of lines in the address bus depends on the capacity of the memory For example, a 15-bit address code can select 32,768 locations (2^{15}) in the memory; a 16-bit address code can select 65,536 locations (2^{16}) in the memory and so on

In personal computers a 32-bit address bus can select 4,294,967,296 locations (2^{32}), expressed as 4GB

551 Write Operation

To store a byte of data in the memory, a code held in the address register is placed on the address bus Once the address code is on the bus, the address decoder decodes the address and selects the specified location in the memory The memory then gets a write command, and the data byte held in the data register is placed on the data bus and stored in the selected memory address, thus completing the write operation When a new data byte is written into a memory address, the current data byte stored at that address is overwritten (replaced with a new data byte)

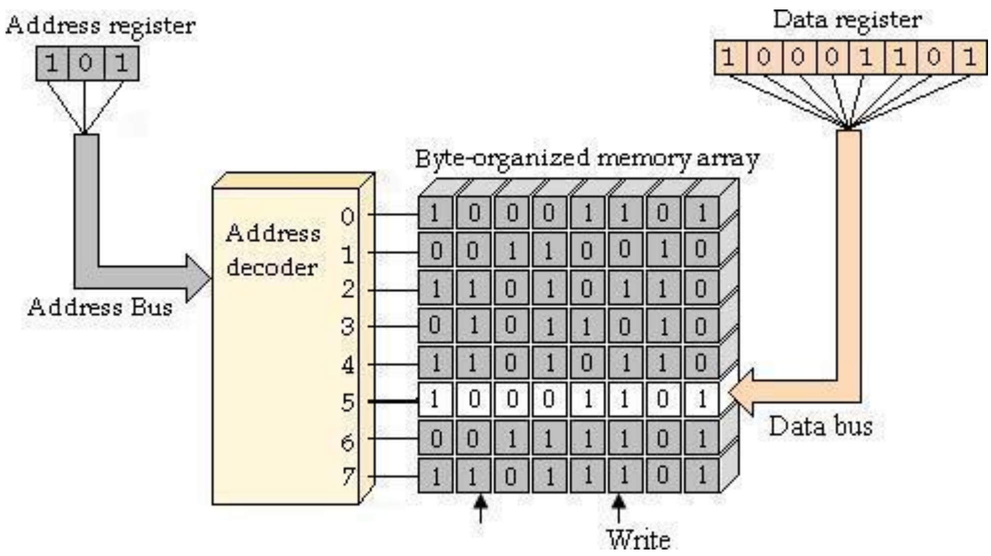


Illustration of the Write operation

552 Read Operation

A code held in the address register is placed on the address bus Once the address code is on the bus, the address decoder decodes the address and selects the specified location in the memory The memory then gets a read command, and a "copy" of the data byte that is stored in the selected memory address is placed on the data bus and loaded into the data register, thus completing the read operation When a data byte is read from a memory address, it also remains stored at that address This is called *nondestructive* read

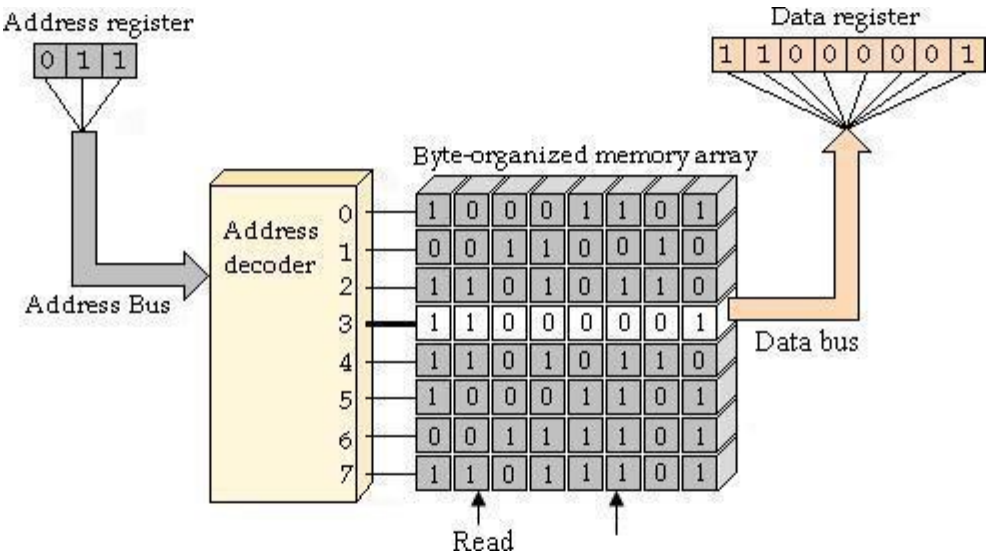


Illustration of the Read operation

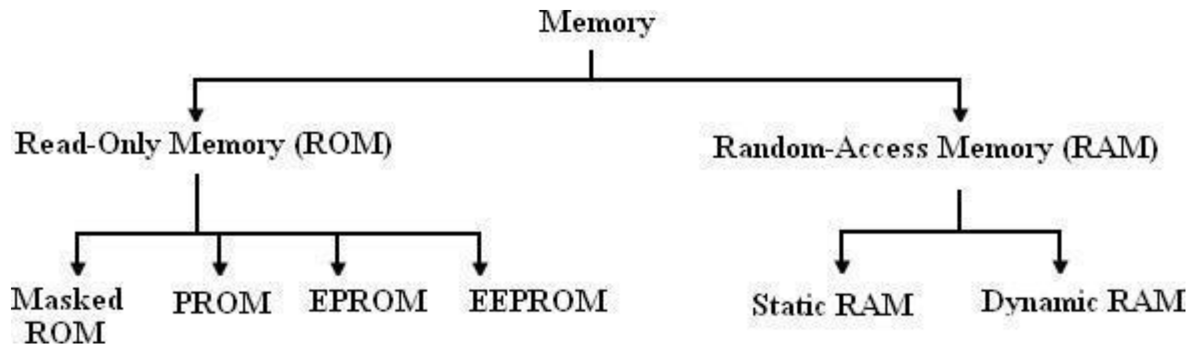
56 Classification of Memories

There are two types of memories that are used in digital systems:

- Random-Access Memory (RAM),
- Read-Only Memory (ROM)

RAM (random-access memory) is a type of memory in which all addresses are accessible in an equal amount of time and can be selected in any order for a read or write operation All RAMs have both read and write capability Because RAMs lose stored data when the power is turned off, they are ***volatile*** memories

ROM (read-only memory) is a type of memory in which data are stored permanently or semi permanently Data can be read from a ROM, but there is no write operation as in the RAM The ROM, like the RAM, is a random-access memory but the term RAM traditionally means a random-access read/write memory Because ROMs retain stored data even if power is turned off, they are ***nonvolatile*** memories



Classification of memories

561 RANDOM-ACCESS MEMORIES (RAMS)

RAMs are read/write memories in which data can be written into or read from any selected address in any sequence When a data unit is written into a given address in the RAM, the data unit previously stored at that address is replaced by the new data unit When a data unit is read from a given address in the RAM, the data unit remains stored and is not erased by the read operation This nondestructive read operation can be viewed as copying the content of an address while leaving the content intact

A RAM is typically used for short-term data storage because it cannot retain stored data when power is turned off

The two categories of RAM are the ***static* RAM** (SRAM) and the ***dynamic* RAM** (DRAM) Static RAMs generally use flip-flops as storage elements and can therefore store data indefinitely *as long as dc power is applied* Dynamic RAMs use capacitors as storage elements and cannot retain data very long without the capacitors being recharged by a process called **refreshing** Both SRAMs and DRAMs will lose stored data when dc power is removed and, therefore, are classified as ***volatile memories***

Data can be read much faster from SRAMs than from DRAMs However, DRAMs can store much more data than SRAMs for a given physical size and cost because the DRAM cell is much simpler, and more cells can be crammed into a given chip area than in the SRAM

5611 Static RAM (SRAM)

Storage Cell:

All static RAMs are characterized by flip-flop memory cells As long as dc power is applied to a static memory cell, it can retain a 1 or 0 state indefinitely If power is removed, the stored data bit is lost

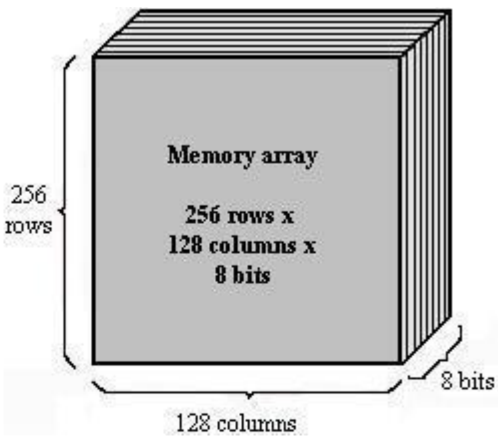
The cell is selected by an active level on the Select line and a data bit (l or 0) is written into the cell by placing it on the Data in line A data bit is read by taking it off the Data out line

Basic SRAM Organization:

Basic Static Memory Cell Array

The memory cells in a SRAM are organized in rows and columns All the cells in a row share the same Row Select line Each set of Data in and Data out lines go to each cell in a given column and are connected to a single data line that serves as both an input and output (Data I/O) through the data input and data output buffers

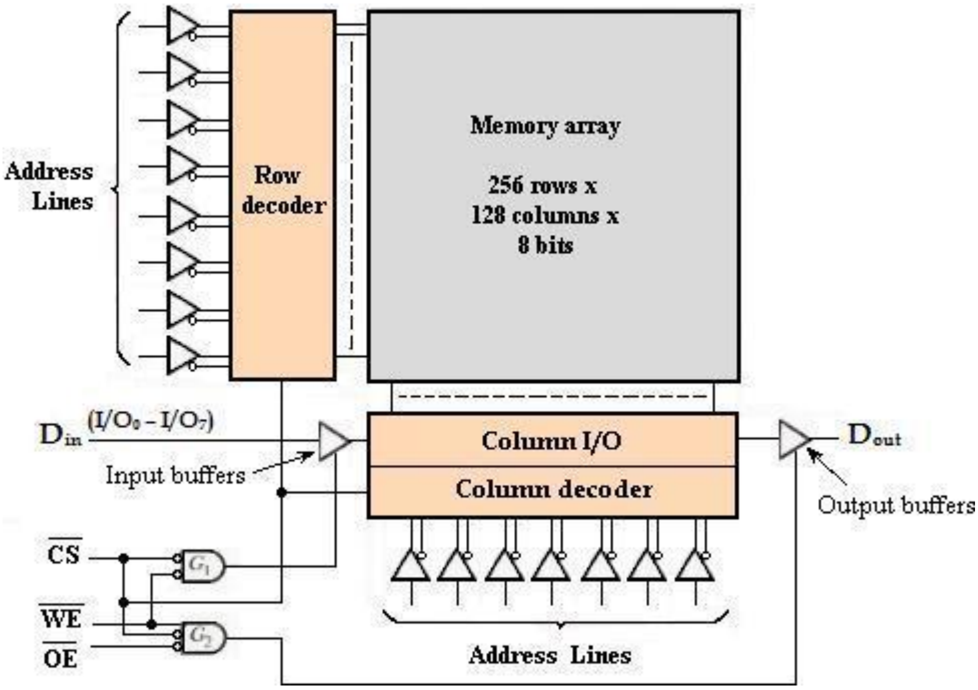
SRAM chips can be organized in single bits, nibbles (4 bits), bytes (8 bits), or multiple bytes (16, 24, 32 bits, etc) The memory cell array is arranged in 256 rows and 128 columns, each with 8 bits as shown below There are actually 2^{15} = 32,768 addresses and each address contains 8 bits The capacity of this example memory is 32,768 bytes (typically expressed as 32 Kbytes)



Memory array configuration

Operation:

The SRAM works as follows First, the chip select, CS, must be LOW for the memory to operate Eight of the fifteen address lines are decoded by the row decoder to select one of the 256 rows Seven of the fifteen address lines are decoded by the column decoder to select one of the 128 8-bit columns



<u>**Memory block diagram**</u>

Read:

In the READ mode, the write enable input, WE' is HIGH and the output enable, OE_ is LOW The input tri state buffers are disabled by gate **G_1**, and the column output tristate buffers are enabled by gate **G_2** Therefore, the eight data bits from the selected address are routed through the column I/O to the data lines (I/O_1 through I/O_7), which are acting as data output lines

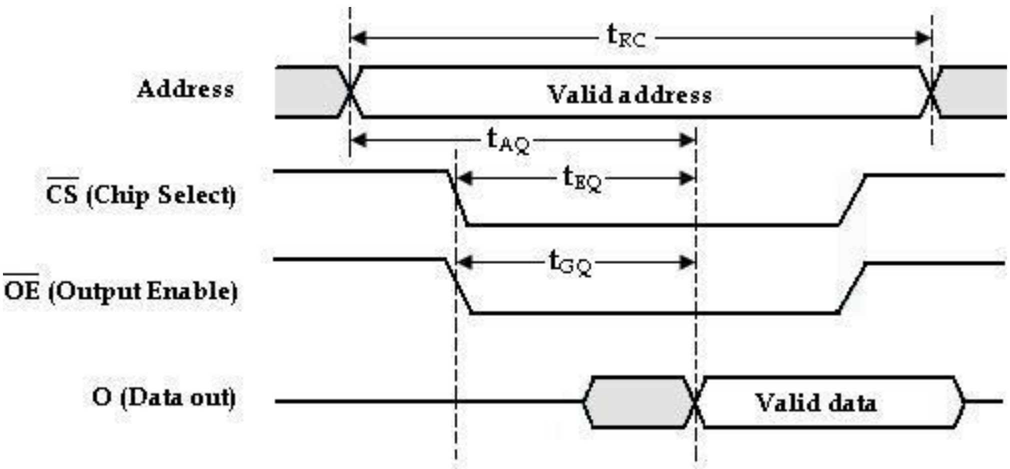
Write:

In the WRITE mode, WE' is LOW and OE' is HIGH The input buffers are enabled by gate G_1, and the output buffers are disabled by gate G_2 Therefore the eight input data bits on the data lines are routed through the input data control and the column I/O to the selected address and stored

Read and Write Cycles:

For the read cycle shown in part (a), a valid address code is applied to the address lines for a specified time interval called the ***read cycle time***, t_{WC} Next, the chip select (CS) and the output enable (DE) inputs go LOW One time interval after the DE input goes LOW; a valid data byte from the selected address appears on the data lines This time interval is called the ***output enable access time***, t_{GQ} Two other access times for the read cycle are the ***address access time***, t_{AQ}, measured from the beginning of a valid address to the appearance of valid data on the data lines and the **chip enable access time**, t_{EQ}, measured from the HIGH-to-LOW transition of CS to the appearance of valid data on the data lines

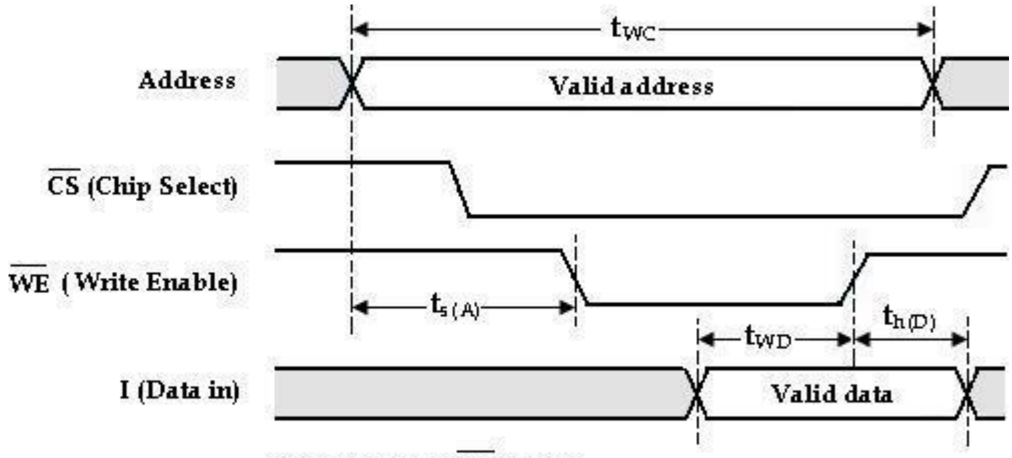
During each read cycle, one unit of data, a byte in this case is read from the memory



(a) Read Cycle ($\overline{WE}$ HIGH)

For the write cycle shown in Figure (b), a valid address code is applied to the address lines for a specified time interval called the ***write cycle time***, t_{WE} Next, the chip select (CS) and the write enable (WE) in puts go LOW The required time interval from the beginning of a valid address until the WE input goes LOW is called the ***address setup time***, $t_{s(A)}$ The time that the WE input must be LOW is the write pulse width The time that the input WE must remain LOW after valid data are applied to the data inputs is designated t_{WD}; the time that the valid input data must remain on the data lines after the WE input goes HIGH is the data hold time, $t_{h(D)}$

During each write cycle, one unit of data is written into the memory



(b) Write cycle ($\overline{WE}$ LOW)

562 READ- ONLY MEMORIES (ROMS)

A ROM contains permanently or semi-permanently stored data, which can be read from the memory but either cannot be changed at all or cannot be changed without specialization equipment A ROM stores data that are used repeatedly in system applications, such as tables, conversions, or programmed instructions for system initialization and operation ROMs retain stored data when the power is OFF and are therefore nonvolatile memories

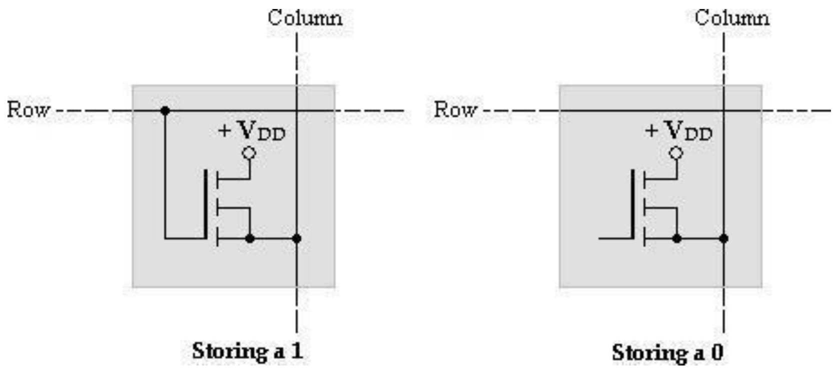
The ROMs are classified as follows:

i. Masked ROM (ROM)
ii. Programmed ROM (PROM)
iii. Erasable PROM (EPROM)
iv. Electrically Erasable PROM (EEPROM)

5621 Masked ROM

The mask ROM is usually referred to simply as a ROM It is permanently programmed during the manufacturing process to provide widely used standard functions, such as popular conversions, or to provide user-specified functions Once the memory is programmed, it cannot be changed

Most IC ROMs utilize the presence or absence of a transistor connection at a row/column junction to represent a 1 or a 0 The presence of a connection from a row line to the gate of a transistor represents a 1 at that location because when the row line is taken HIGH; all transistors with a gate connection to that row line turn on and connect the HIGH (1) to the associated column lines



ROM Cells

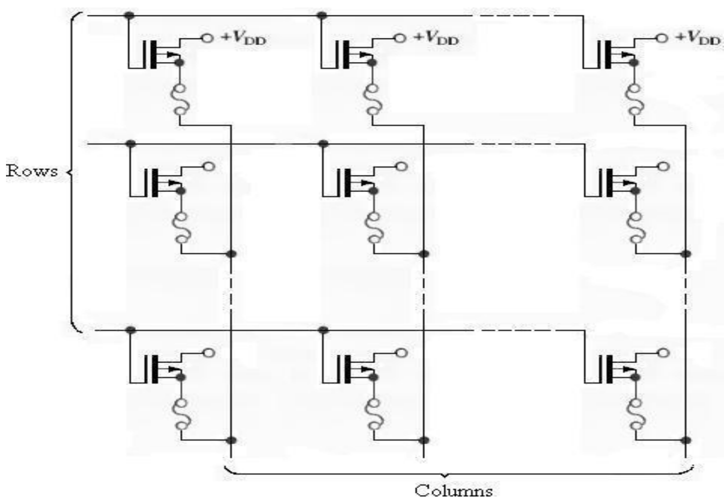
At row/column junctions where there are no gate connections, the column lines remain LOW (0) when the row is addressed

5622 PROM (Programmable Read-Only Memory)

The PROM (Programmable Read-only memory), comes from the manufacturer unprogrammed and are custom programmed in the field to meet the user‘s needs

A PROM uses some type of fusing process to store bits, in which a memory link is burned open or left intact to represent a 0 or a 1 The fusing process is irreversible; once a PROM is programmed, it cannot be changed

The fusible links are manufactured into the PROM between the source of each cell's transistor and its column line In the programming process, a sufficient current is injected through the fusible link to bum it open to create a stored O The link is left intact for a stored 1 All drains are commonly connected to V_{DD}



PROM array with fusible links

Three basic fuse technologies used in PROMs are metal links, silicon links, and pn junctions A brief description of each of these follows

1. **Metal links** are made of a material such as ***nichrome*** Each bit in the memory array is represented by a separate link During programming, the link is either "blown" open or left intact This is done basically by first addressing a given cell and then forcing a sufficient amount of current through the link to cause it to open When the fuse is intact, the memory cell is configured as a logic 1 and when fuse is blown (open circuit) the memory cell is logic 0

2. **Silicon links** are formed by narrow, notched strips of ***polycrystalline silicon*** Programming of these fuses requires melting of the links by passing a sufficient amount of current through them This amount of current causes a high temperature at the fuse location that oxidizes the silicon and forms insulation around the now-open link

3. **Shorted junction**, or avalanche-induced migration, technology consists basically of two pn junctions arranged back-to-back During programming, one of the diode junctions is avalanched, and the resulting voltage and heat cause aluminum ions to migrate and short the junction The remaining junction is then used as a forward- biased diode to represent a data bit

5623 EPROM (Erasable Programmable ROM)

An EPROM is an erasable PROM Unlike an ordinary PROM, an EPROM can be reprogrammed if an existing program in the memory array is erased first

An EPROM uses an NMOSFET array with an isolated-gate structure The isolated transistor gate has no electrical connections and can store an electrical charge for indefinite periods of time The data bits in this type of array are represented by the presence or absence of a stored gate charge Erasure of a data bit is a process that removes the gate charge

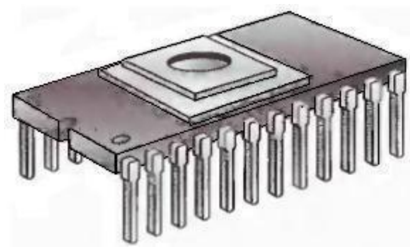
Two basic types of erasable PROMs are the ultraviolet erasable PROM (UV EPROM) and the electrically erasable PROM (EEPROM)

- **UV EPROM:**

You can recognize the UV EPROM device by the transparent quartz lid on the package, as shown in Figure below The isolated gate in the FET of an ultraviolet EPROM is "floating" within an oxide insulating material The programming process causes electrons to be removed from the floating gate Erasure is done by exposure of the memory array chip to high-intensity ultraviolet radiation through the quartz window on top of the package

The positive charge stored on the gate is neutralized after several minutes to an hour of exposure time In EPROM's, it is not possible to erase selective information, when erased the entire information is lost The chip can be reprogrammed

It is ideally suited for product development, college laboratories, etc



Ultraviolet Erasable PROM

During programming, address and datas are applied to address and data pins of the EPROM The program pulse is applied to the program input of the EPROM The program pulse duration is around 50msec and its amplitude depends on EPROM IC It is typically 115V to 25V

In EPROM, it is possible to program any location at any time- either individually, sequentially or at random

5624 EEPROM (Electrically Erasable PROM)

The EEPROM (Electrically Erasable PROM), also uses MOS circuitry Data is stored as charge or no charge on an insulating layer, which is made very thin (< 200Å) Therefore a voltage as low as 20- 25V can be used to move charges across the thin barrier in either direction for programming or erasing ROM

An electrically erasable PROM can be both erased and programmed with electrical pulses Since it can be both electrically written into and electrically erased, the EEPROM can be rapidly programmed and erased in-circuit for reprogramming

It allows selective erasing at the register level rather than erasing all the information, since the information can be changed by using electrical signals

It has chip erase mode by which the entire chip can be erased in 10 msec Hence EEPROM's are most expensive

Advantages of RAM:

1. Fast operating speed **(< 150 nsec**),
2. Low power dissipation **(< 1mW**),
3. Economy,
4. Compatibility,
5. Non-destructive read-out

Advantages of ROM:

1. Ease and speed of design,
2. Faster than MSI devices (PLD and FPGA)
3. The program that generates the ROM contents can easily be structured to handle unusual or undefined cases,
4. A ROM's function is easily modified just by changing the stored pattern, usually without changing any external connections,
5. More economical

Disadvantages of ROM:

1. For functions more than 20 inputs, a ROM based circuit is impractical because of the limit on ROM sizes that are available
2. For simple to moderately complex functions, ROM based circuit may be costly: consume more power; run slower

Comparison between RAM and ROM:

SNo	RAM	ROM
1	RAMs have both read and write	ROMs have only read operation capability
2	RAMs are volatile memories	ROMs are non-volatile memories
3	They lose stored data when the power is turned OFF	They retain stored data even if power is turned off
4	RAMs are available in both bipolar and MOS technologies	RAMs are available in both bipolar and MOS technologies
5	**Types**: SRAM, DRAM, EEPROM	**Types**: PROM, EPROM

Comparison between SRAM and DRAM:

SNo	Static RAM	Dynamic RAM
1	It contains less memory cells per unit area	It contains more memory cells per unit area
2	Its access time is less, hence faster memories	Its access time is greater than static RAM
3	It consists of number of flip-flops Each flip-flop stores one bit	It stores the data as a charge on the capacitor It consists of MOSFET and capacitor for each cell
4	Refreshing circuitry is not required	Refreshing circuitry is required to maintain the charge on the capacitors every time after every few milliseconds Extra hardware is required to control refreshing
5	Cost is more	Cost is less

Comparison of Types of Memories:

Memory type	Non- Volatile	High Density	One- Transistor cell	In-system writability
SRAM	No	No	No	Yes
DRAM	No	Yes	Yes	Yes
ROM	Yes	Yes	Yes	No
EPROM	Yes	Yes	Yes	No
EEPROM	Yes	No	No	Yes

58 PROGRAMMABLE LOGIC DEVICES:

581 INTRODUCTION:

A combinational PLD is an integrated circuit with programmable gates divided into an AND array and an OR array to provide an AND-OR sum of product implementation The PLD's can be reprogrammed in few seconds and hence gives more flexibility to experiment with designs Reprogramming feature of PLDs also makes it possible to accept changes/modifications in the previously design circuits

The advantages of using programmable logic devices are:

1. Reduced space requirements
2. Reduced power requirements
3. Design security
4. Compact circuitry
5. Short design cycle
6. Low development cost
7. Higher switching speed
8. Low production cost for large-quantity production

According to architecture, complexity and flexibility in programming in PLD‘s are classified as —

- PROMs : Programmable Read Only memories,
- PLAs : Programmable Logic Arrays,
- PAL : Programmable Logic Array,
- FPGA : Field Programmable Gate Arrays,
- CPLDs : Complex Programmable Logic Devices

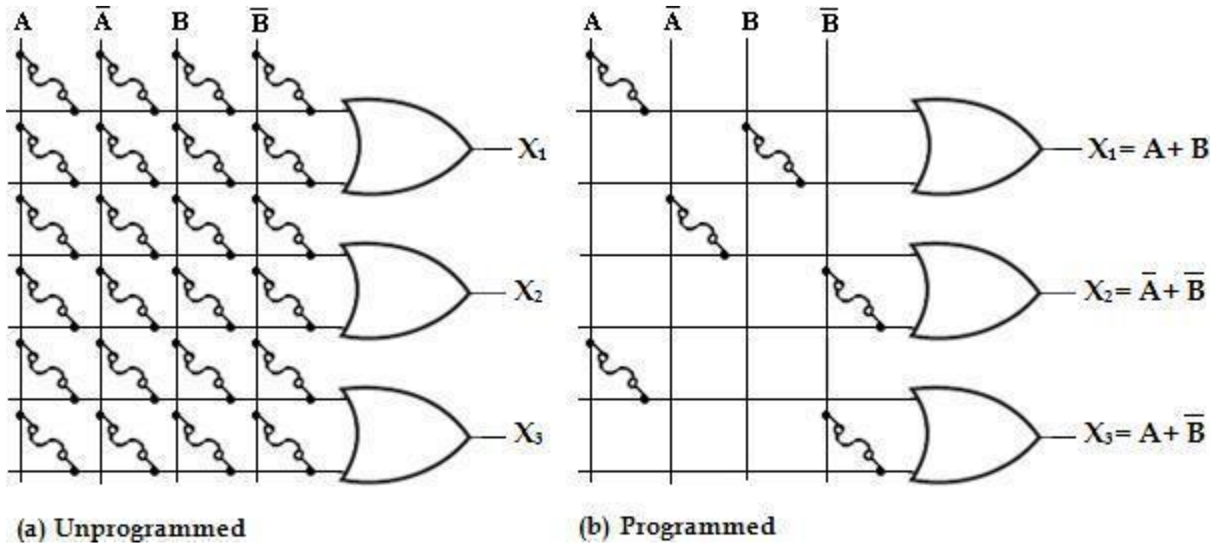
Programmable Arrays:

All PLDs consists of programmable arrays A programmable array is essentially a grid of conductors that form rows and columns with a fusible link at each cross point Arrays can be either fixed or programmable

The OR Array:

It consists of an array of OR gates connected to a programmable matrix with fusible links at each cross point of a row and column, as shown in the figure below The array can be programmed by blowing fuses to eliminate selected variables from the output functions For each input to an OR gate, only one fuse is left intact in order to connect the desired variable to the gate input Once the fuse is blown, it cannot be reconnected

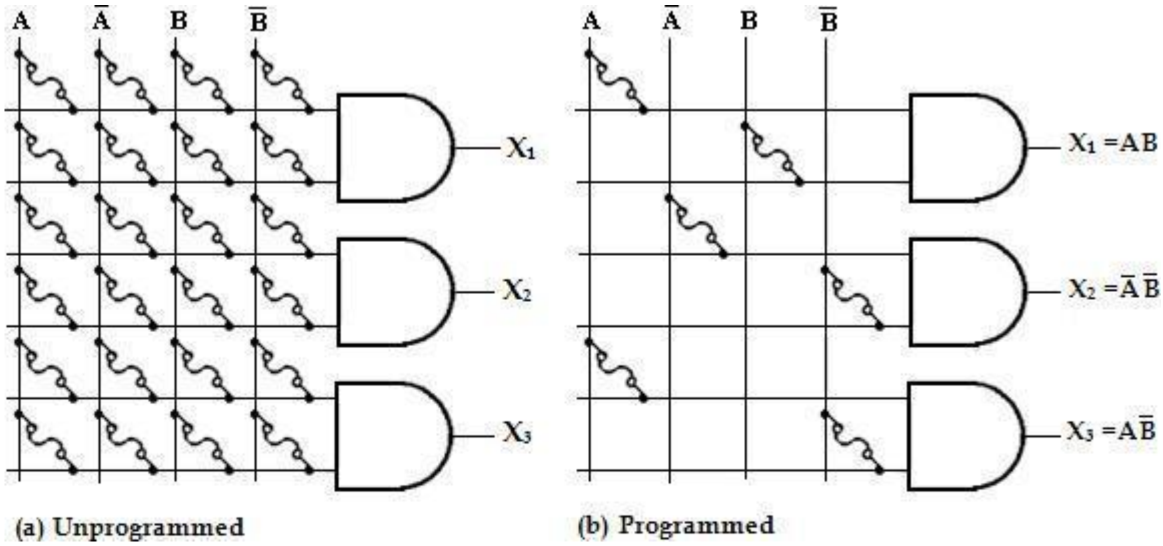
Another method of programming a PLD is the antifuse, which is the opposite of the fuse Instead of a fusible link being broken or opened to program a variable, a normally open contact is shorted by ―melting‖ the antifuse material to form a connection



<u>**An example of a basic programmable OR array**</u>

The AND Array:

This type of array consists of AND gates connected to a programmable matrix with fusible links at each cross points, as shown in the figure below Like the OR array, the AND array can be programmed by blowing fuses to eliminate selected variables from the output functions For each input to an AND gate, only one fuse is left intact in order to connect the desired variable to the gate input Also, like the OR array, the AND array with fusible links or with antifuses is one-time programmable



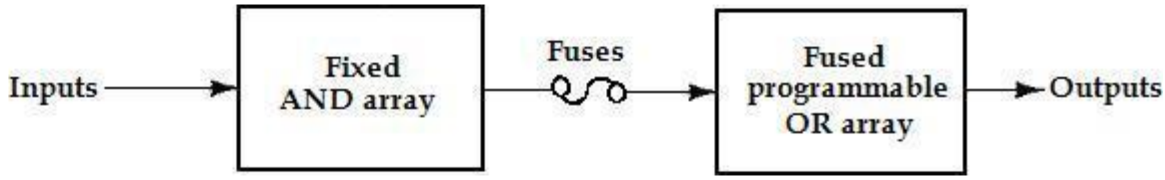
<u>**An example of a basic programmable AND array**</u>

582 Classification of PLDs

There are three major types of combinational PLDs and they differ in the placement of the programmable connections in the AND-OR array The configuration of the three PLDs is shown below

1 Programmable Read-Only Memory (PROM):

A PROM consists of a set of fixed (non-programmable) AND array constructed
as a decoder and a programmable OR array The programmable OR gates implement the Boolean functions in sum of minterms



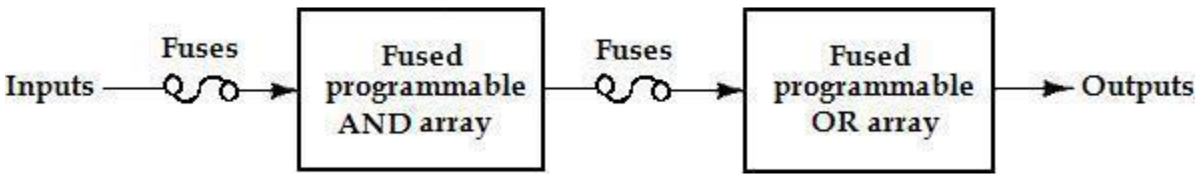
(a) Programmable read- only memory (PROM)

2. Programmable Logic Array (PLA):

A PLA consists of a programmable AND array and a programmable OR array

The product terms in the AND array may be shared by any OR gate to provide the required sum of product implementation

The PLA is developed to overcome some of the limitations of the PROM The PLA is also called an FPLA (Field Programmable Logic Array) because the user in the field, not the manufacturer, programs it

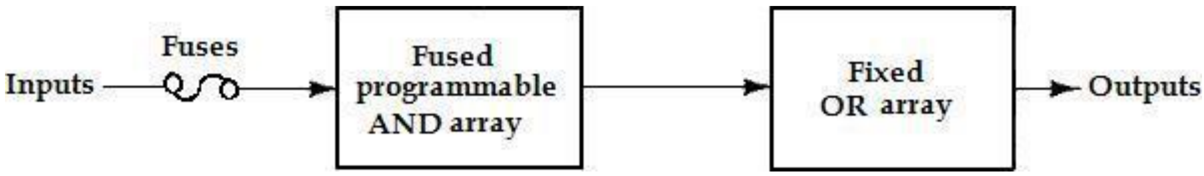


Programmable Logic Array (PLA)

3 Programmable Array Logic (PAL):

The basic PAL consists of a programmable AND array and a fixed OR array The AND gates are programmed to provide the product terms for the Boolean functions, which are logically summed in each OR gate

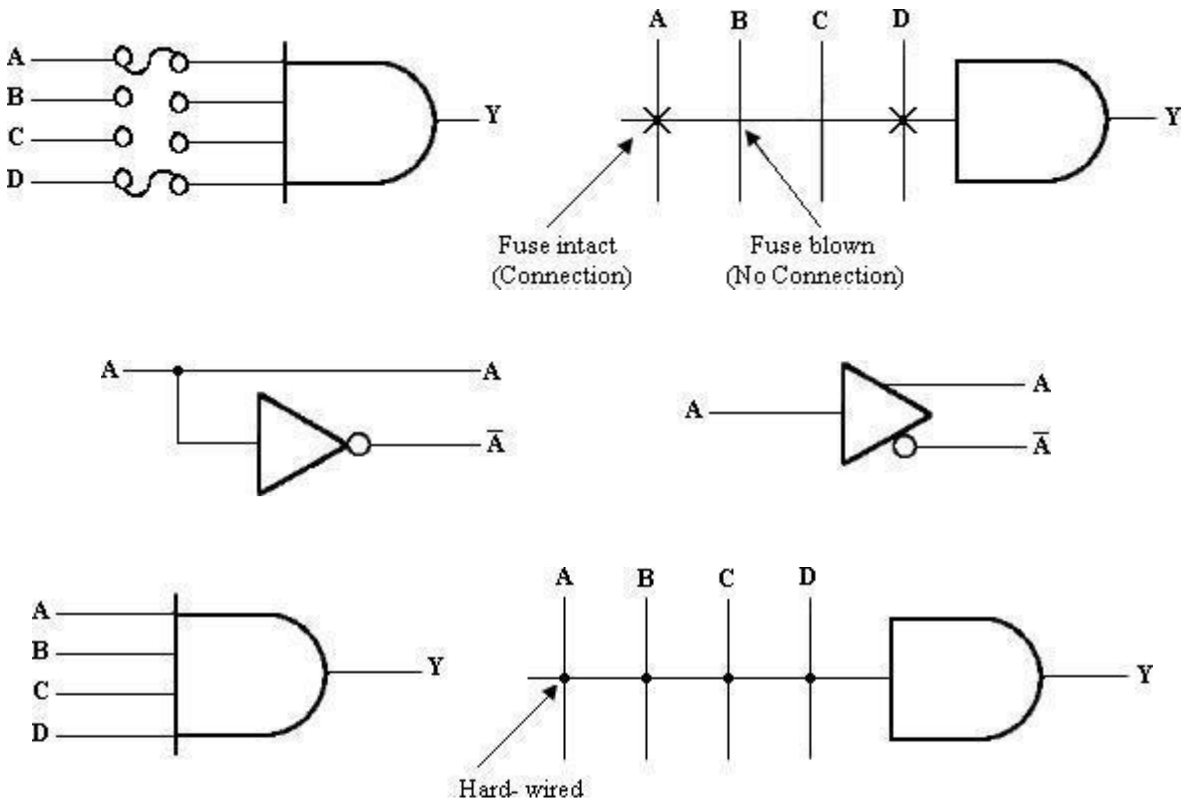
It is developed to overcome certain disadvantages of the PLA, such as longer delays due to the additional fusible links that result from using two programmable arrays and more circuit complexity



Programmable Array Logic (PAL)

Array logic Symbols:

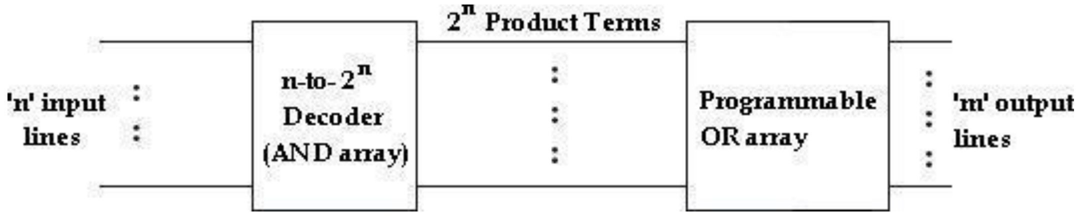
PLDs have hundreds of gates interconnected through hundreds of electronic fuses It is sometimes convenient to draw the internal logic of such device in a compact form referred to as **array logic**



583 PROGRAMMABLE ROM:

PROMs are used for code conversions, generating bit patterns for characters and as look-up tables for arithmetic functions

As a PLD, PROM consists of a fixed AND-array and a programmable OR array The AND array is an n-to-2_n decoder and the OR array is simply a collection of programmable OR gates The OR array is also called the memory array The decoder serves as a minterm generator The n-variable minterms appear on the 2_n lines at the decoder output The 2_n outputs are connected to each of the _m‘ gates in the OR array via programmable fusible links



2_n x m PROM

584 Implementation of Combinational Logic Circuit using PROM

1. Using PROM realize the following expression

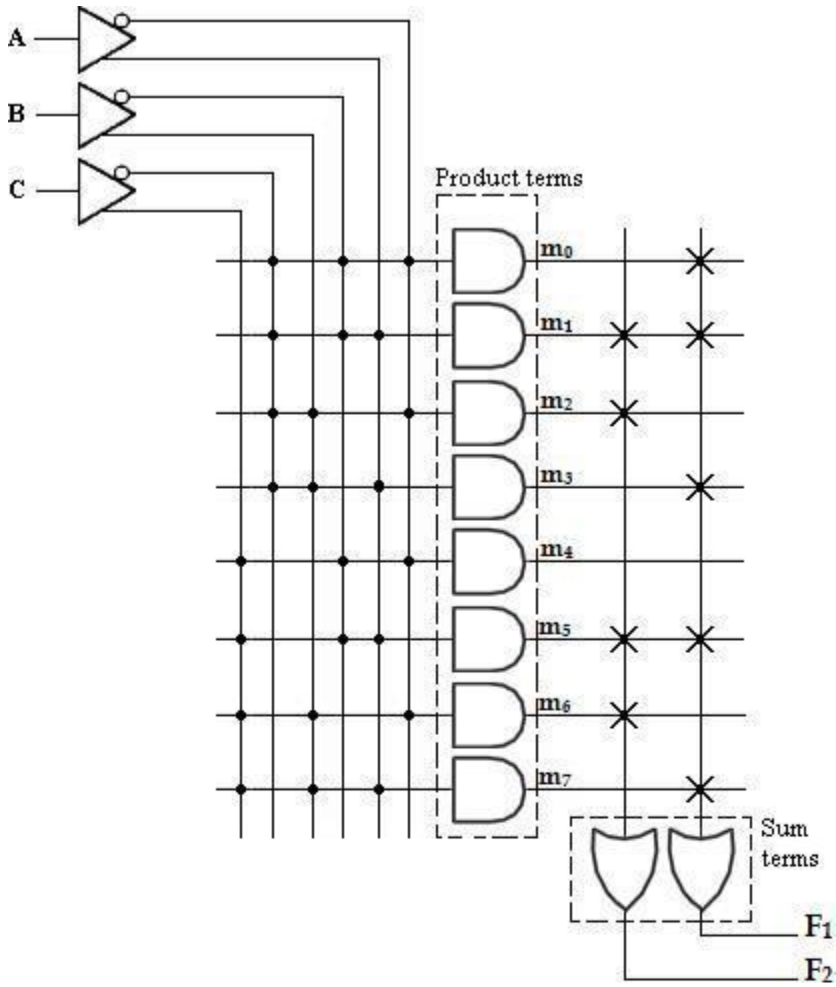
F_1 (A, B, C) = ∑m (0, 1, 3, 5, 7)

F_2 (A, B, C) = ∑m (1, 2, 5, 6)

Step1: Truth table for the given function

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

Step 2: PROM diagram

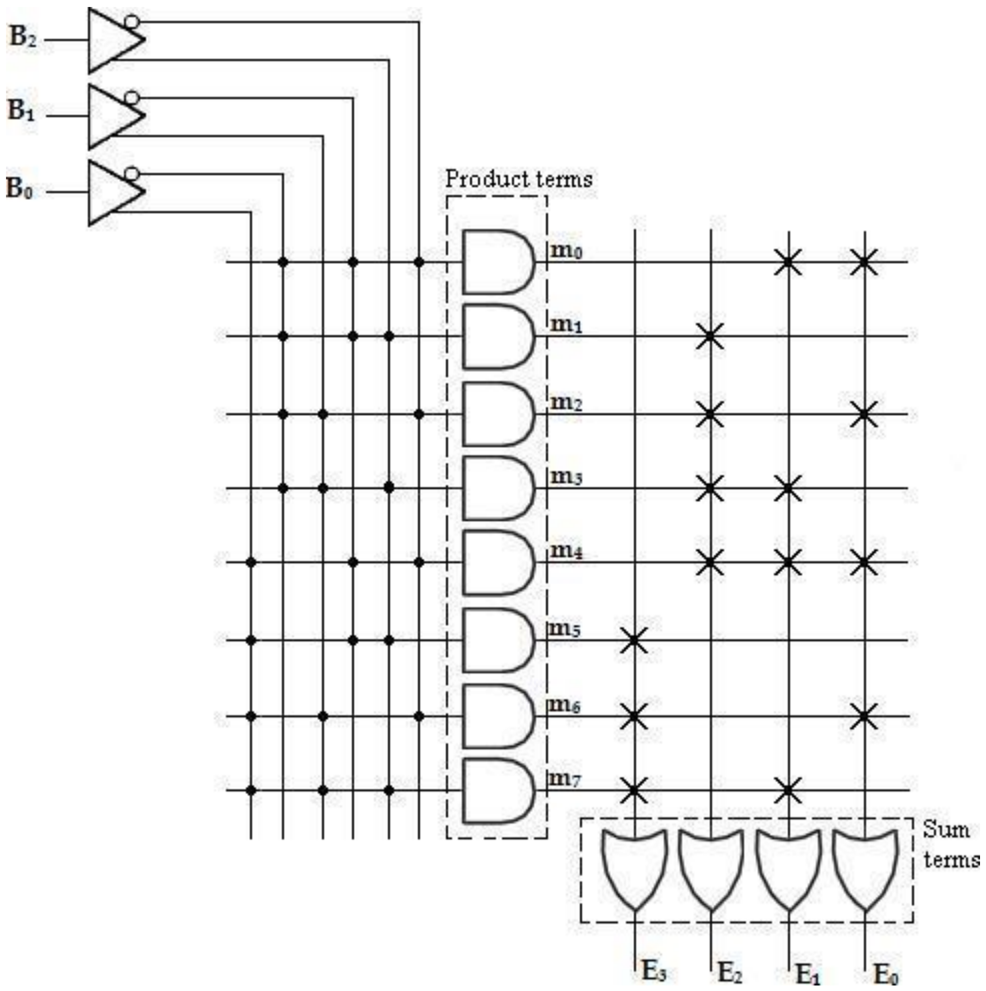


2. Design a combinational circuit using PROM The circuit accepts 3-bit binary and generates its equivalent Excess-3 code

Step1: Truth table for the given function

B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	1	1
0	0	1	0	1	0	0
0	1	0	0	1	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	1
1	0	1	1	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	1	0

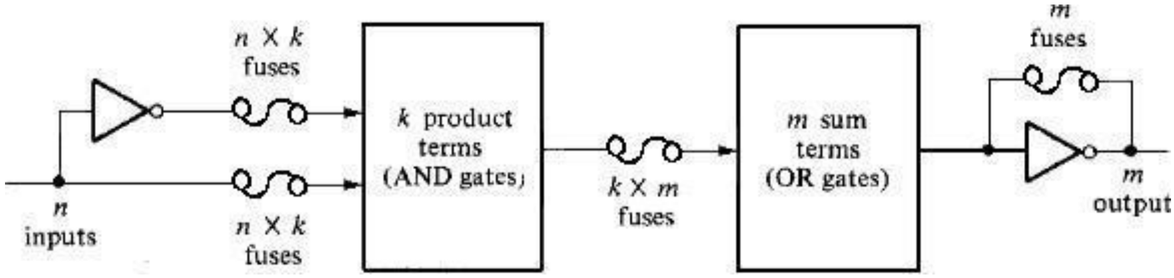
Step 2: PROM diagram



585 PROGRAMMABLE LOGIC ARRAY: (PLA)

The PLA is similar to the PROM in concept except that the PLA does not provide full coding of the variables and does not generate all the minterms

The decoder is replaced by an array of AND gates that can be programmed to generate any product term of the input variables The product term are then connected to OR gates to provide the sum of products for the required Boolean functions The AND gates and OR gates inside the PLA are initially fabricated with fuses among them The specific boolean functions are implemented in sum of products form by blowing the appropriate fuses and leaving the desired connections



PLA block diagram

The block diagram of the PLA is shown above It consists of ‗n‘ inputs, ‗m‘ outputs, ‗k‘ product terms and ‗m‘ sum terms The product terms constitute a group of ‗k‘ AND gates and the sum terms constitute a group of ‗m‘ OR gates Fuses are inserted between all ‗n‘ inputs and their complement values to each of the AND gates Fuses are also provided between the outputs of the AND gate and the inputs of the OR gates

Another set of fuses in the output inverters allow the output function to be generated either in the AND-OR form or in the AND-OR-INVERT form With the inverter fuse in place, the inverter is bypassed, giving an AND-OR implementation With the fuse blown, the inverter becomes part of the circuit and the function is implemented in the AND-OR-INVERT form

586 Implementation of Combinational Logic Circuit using PLA

1. Implement the combinational circuit with a PLA having 3 inputs, 4 product terms and 2 outputs for the functions

F_1 (A, B, C) = ∑m (0, 1, 2, 4)

F_2 (A, B, C) = ∑m (0, 5, 6, 7)

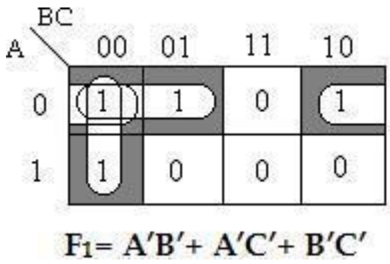
Solution:

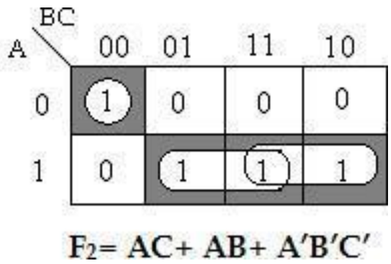
Step 1: Truth table for the given functions

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

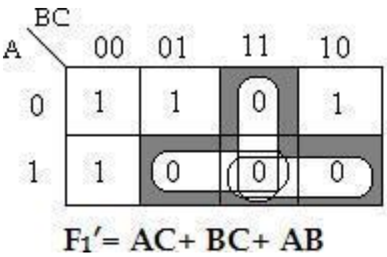
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

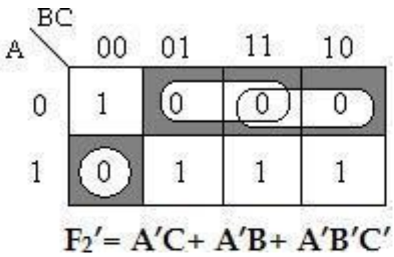
Step 2: K-map Simplification





With this simplification, total number of product term is 6 But we require only 4 product terms Therefore find out F_1‘ and F_2‘





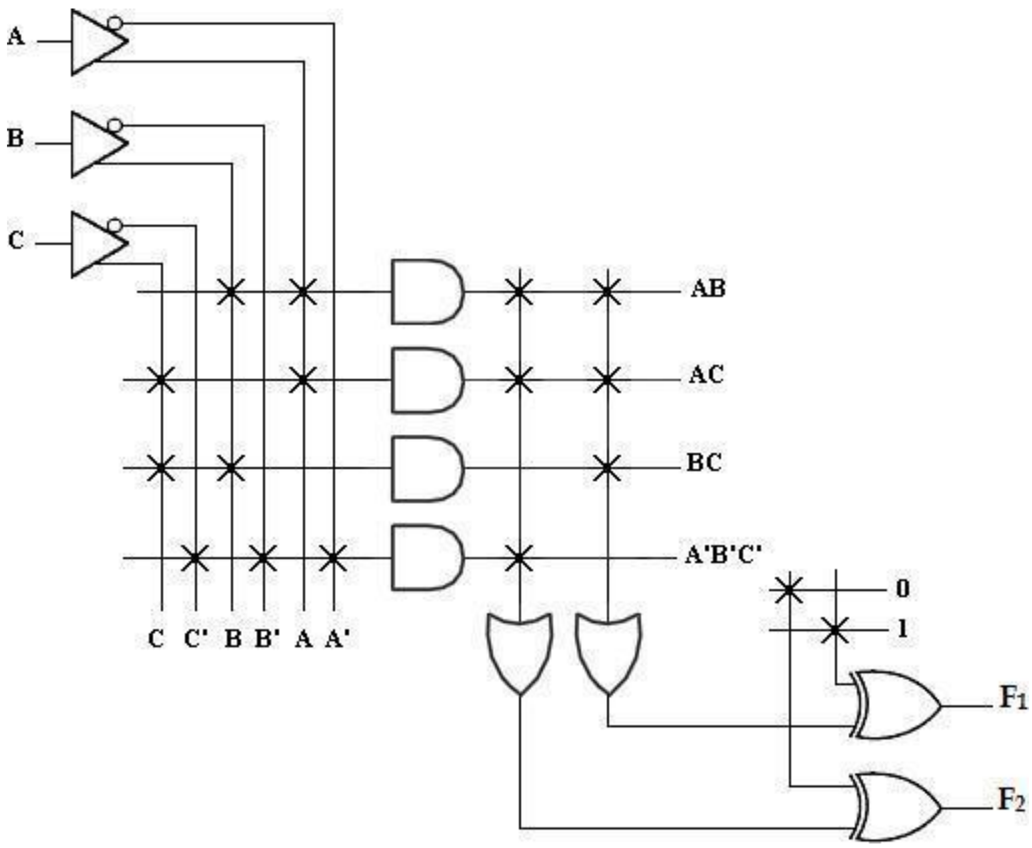
Now select, F_1‘ and F_2, the product terms are AC, AB, BC and A‘B‘C‘

Step 3: PLA Program table:

	Product term	Inputs			Outputs	
		A	B	C	F_1 (C)	F_2 (T)
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

In the PLA program table, first column lists the product terms numerically as 1, 2, 3, and 5 The second column (Inputs) specifies the required paths between the AND gates and the inputs For each product term, the inputs are marked with 1, 0, or - (dash) If a variable in the product form appears in its normal form, the corresponding input variable is marked with a 1 If it appears complemented, the corresponding input variable is marked with a 0 If the variable is absent in the product term, it is marked with a dash (-) The third column (output) specifies the path between the AND gates and the OR gates The output variables are marked with 1's for all those product terms that formulate the required

function **Step 4:** PLA Diagram



The PLA diagram uses the array logic symbols for complex symbols Each input and its complement is connected to the inputs of each AND gate as indicated by the intersections between the vertical and horizontal lines The output of the AND gate are connected to the inputs of each OR gate The output of the OR gate goes to an EX-OR gate where the other input can be programmed to receive a signal equal to either logic 1 or 0

The output is inverted when the EX-OR input is connected to 1 ie, **($x \oplus 1 = x'$)**
The output does not change when the EX-OR input is connected to 0 ie, **($x \oplus 0 = x$)**

2. **Implement the combinational circuit with a PLA having 3 inputs, 4 product terms and 2 outputs for the functions**

   F_1 (A, B, C) = $\sum$m (3, 5, 6, 7)

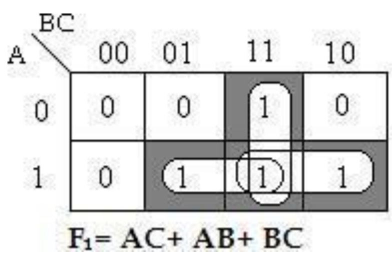
   F_2 (A, B, C) = $\sum$m (0, 2, 4, 7)

   <u>Solution:</u>

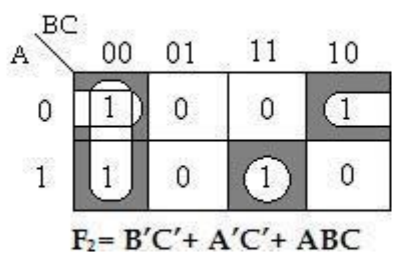
Step 1: Truth table for the given functions

A	B	C	F_1	F_2
0	0	0	0	1
0	0	1	0	0
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: K-map Simplification

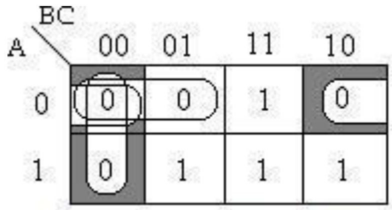


$F_1 = AC + AB + BC$

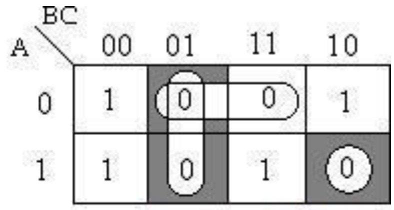


$F_2 = B'C' + A'C' + ABC$

With this simplification, total number of product term is 6 But we require only 4 product terms Therefore find out F_1' and F_2'



$F_1' = B'C' + A'C' + A'B'$



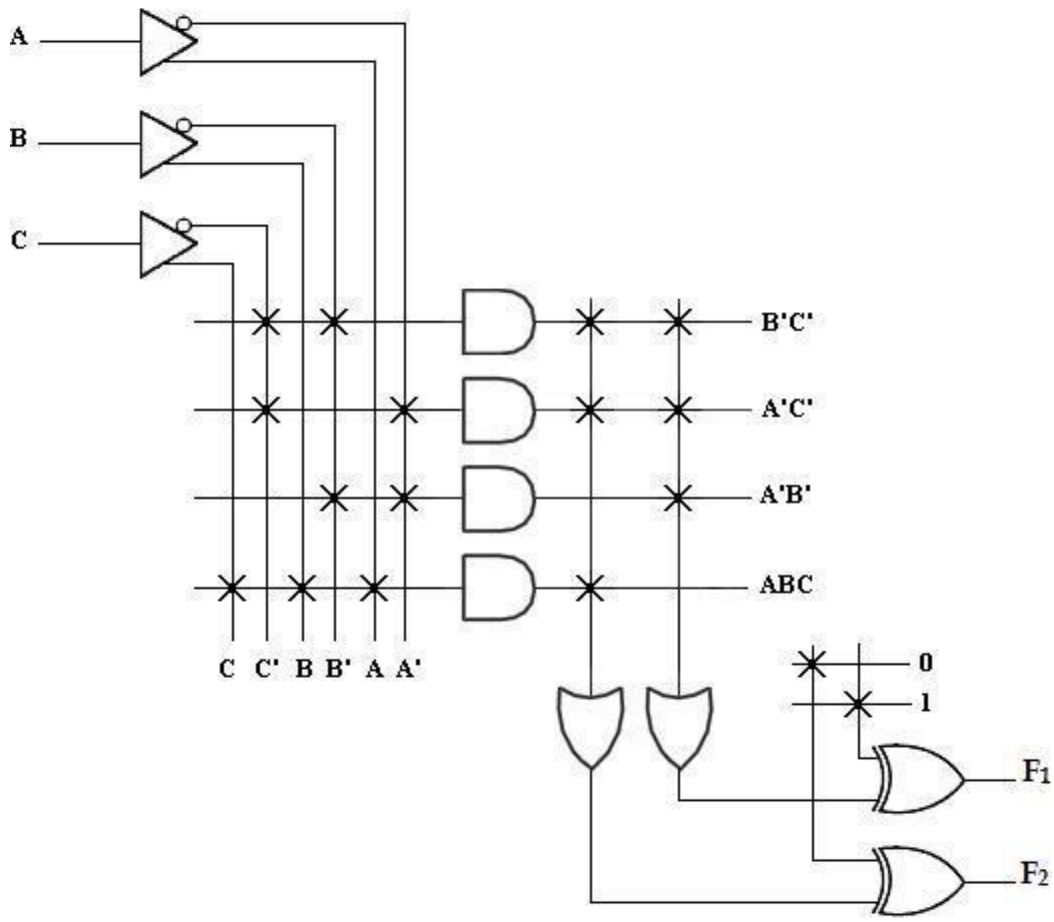
$F_2' = A'C + B'C + ABC'$

Now select, F_1' and F_2, the product terms are **B′C′, A′C′, A′B′ and ABC**

Step 3: PLA Program table

	Product term	Input s			Outputs	
		A	B	C	F_1 (C)	F_2 (T)
B′C′	1	-	0	0	1	1
A′C′	2	0	-	0	1	1
A′B′	3	0	0	-	1	-
ABC	4	1	1	1	-	1

Step 4: PLA Diagram



3. Implement the following functions using PLA

F_1 (A, B, C) = $\sum$m (1, 2, 4, 6)

F_2 (A, B, C) = $\sum$m (0, 1, 6, 7)

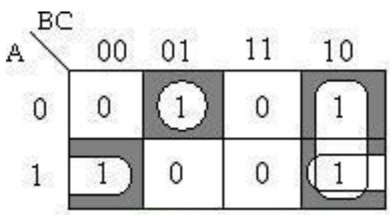
F_3 (A, B, C) = $\sum$m (2, 6)

<u>Solution:</u>

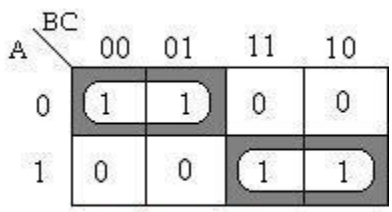
Step 1: Truth table for the given functions

A	B	C	F_1	F_2	F_3
0	0	0	0	1	0
0	0	1	1	1	0
0	1	0	1	0	1
0	1	1	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	1	0

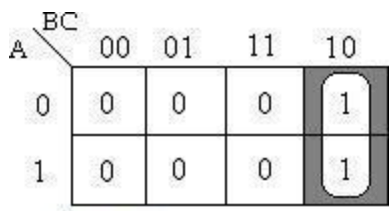
Step 2: K-map Simplification



F_1= A′B′C+ AC′+ BC′



F_2= A′B′+ AB

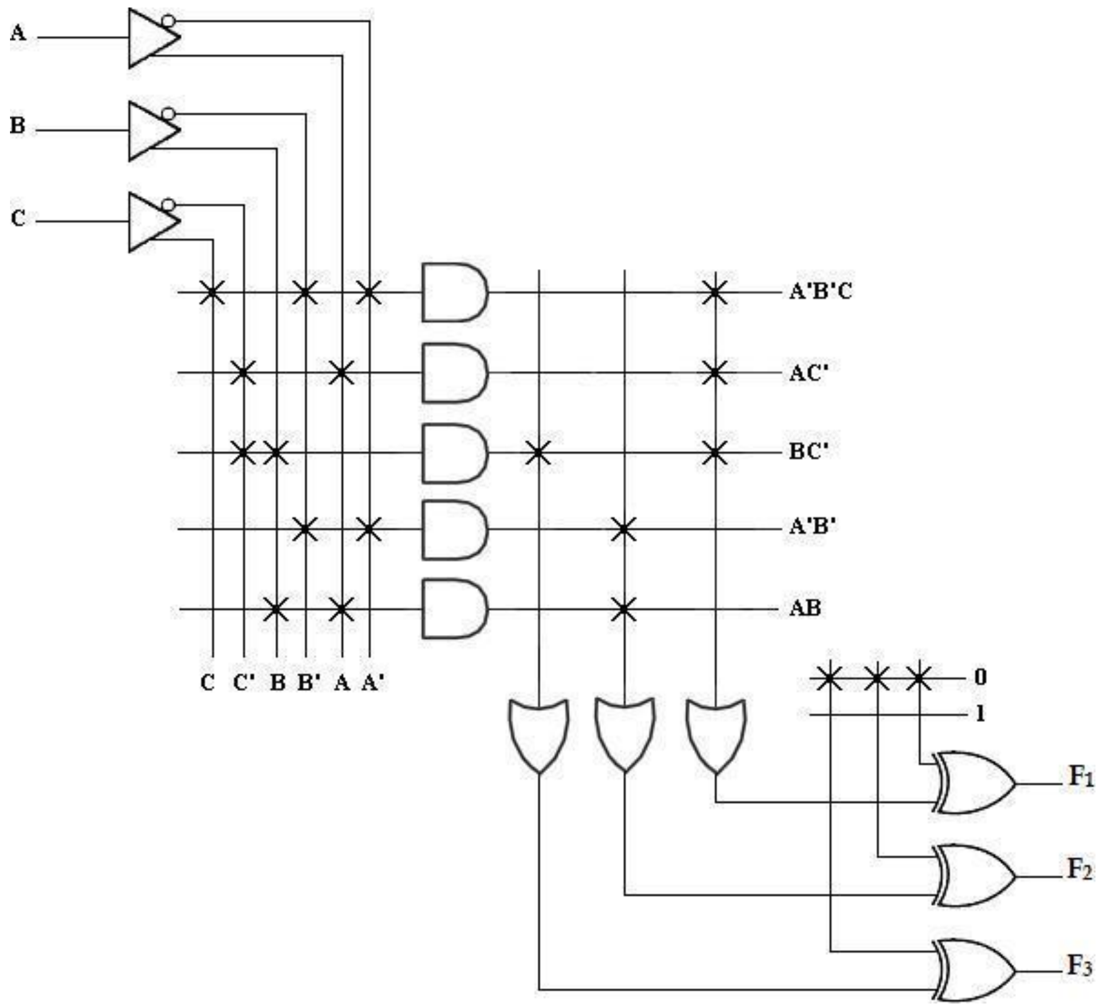


F_3= BC′

Step 3: PLA Program table

	Product term	Inputs			Outputs		
		A	B	C	F_1 (T)	F_2 (T)	F_3 (T)
A'B'C	1	0	0	1	1	-	-
AC'	2	1	-	0	1	-	-
BC'	3	-	1	0	1	-	1
A'B'	4	0	0	-	-	1	-
AB	5	1	1	-	-	1	-

Step 4: PLA Diagram



4. A combinational circuit is designed by the function F_1 (A, B, C) = ∑m (3, 5, 7)

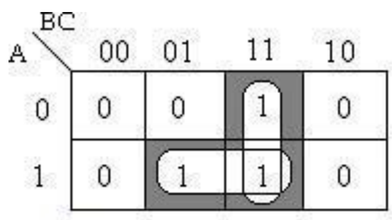
F_2 (A, B, C) = ∑m (4, 5, 7)

<u>Solution:</u>

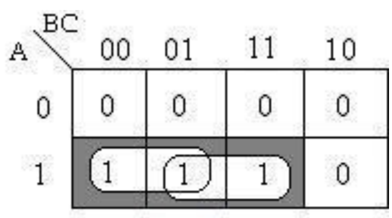
Step 1: Truth table for the given functions

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

Step 2: K-map Simplification



$F_1 = AC + BC$

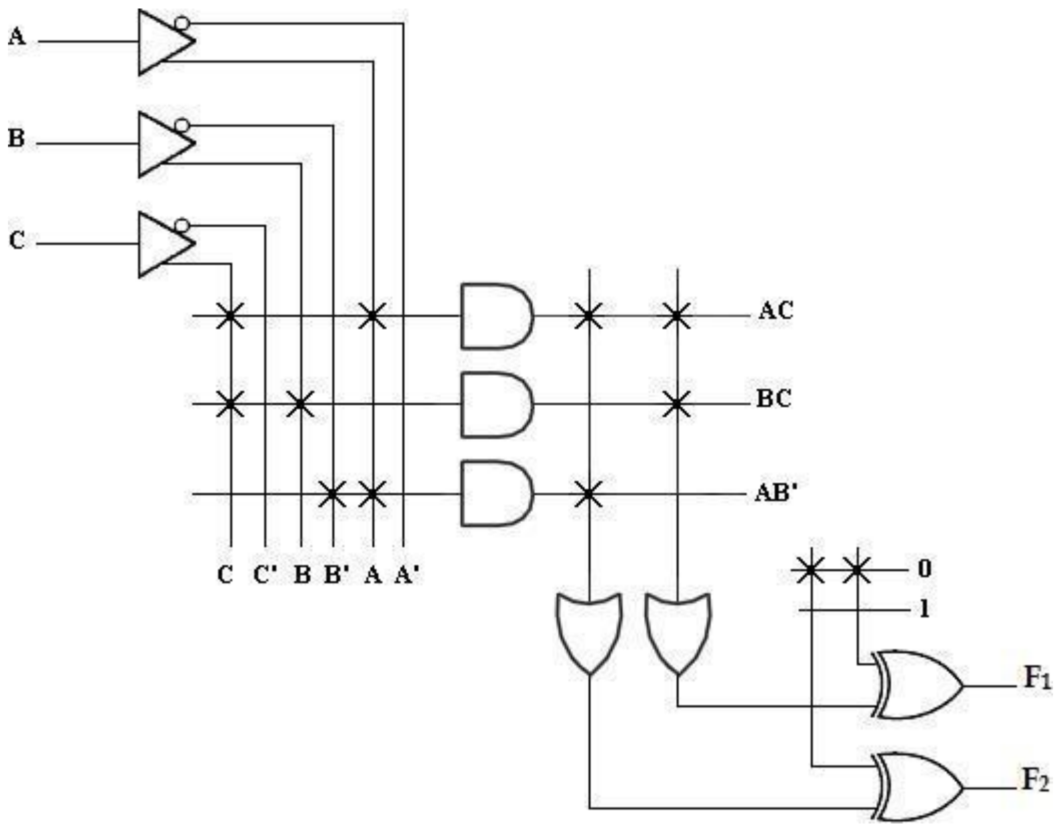


$F_2 = AB' + AC$

Step 3: PLA Program table

	Product term	Inputs			Outputs	
		A	B	C	F_1 (C)	F_2 (T)
AC	1	1	-	1	1	1
BC	2	-	1	1	1	-
AB′	3	1	0	-	-	1

Step 4: PLA Diagram



5. **A combinational circuit is defined by the functions, F_1 (A, B, C) = $\sum$m (1, 3, 5) F_2 (A, B, C) = $\sum$m (5, 6, 7)**

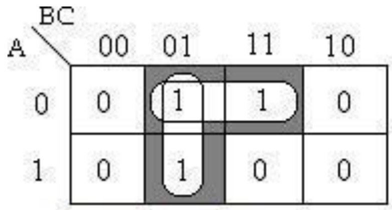
   Implement the circuit with a PLA having 3 inputs, 3 product terms and 2 outputs

Solution:

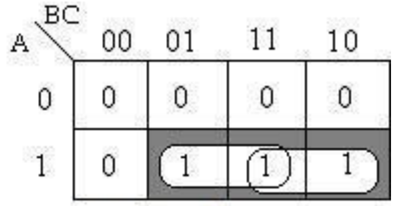
Step 1: Truth table for the given functions

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

Step 2: K-map Simplification

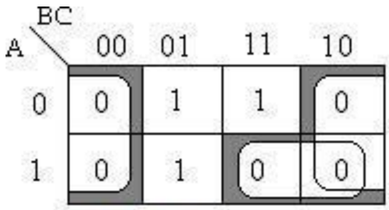


$F_1 = B'C + A'C$

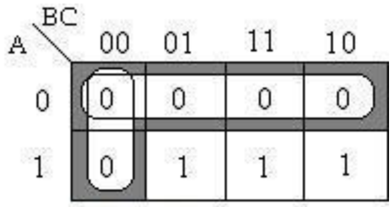


$F_2 = AC + AB$

With this simplification, total number of product term is 5 But we require only 3 product terms Therefore find out F_1' and F_2'



$F_1' = C' + AB$



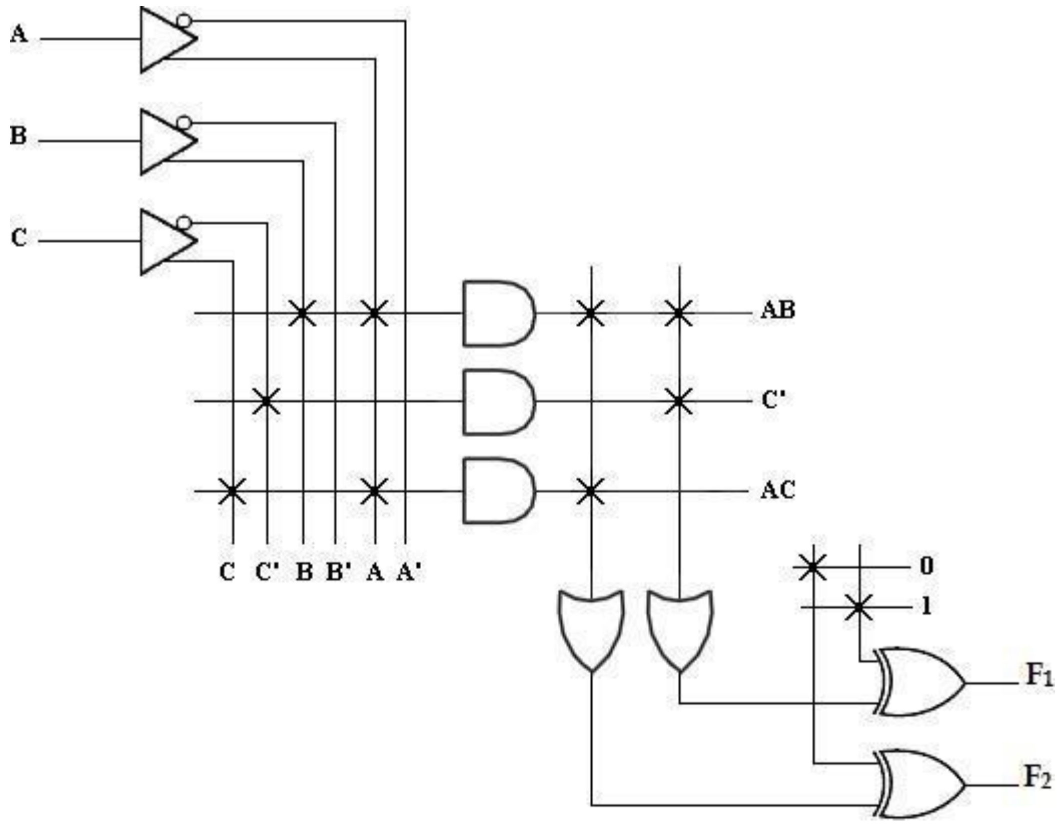
$F_2' = A' + B'C'$

Now select, F_1' and F_2, the product terms are **AC, AB and C′**

Step 3: PLA Program table

	Product term	Inputs			Outputs	
		A	B	C	F_1 (C)	F_2 (T)
AB	1	1	1	-	1	1
C′	2	-	-	0	1	-
AC	3	1	-	1	-	1

Step 4: PLA Diagram



6. **A combinational circuit is defined by the functions, F_1 (A, B, C) = $\sum$m (0, 1, 3, 4)**

   F_2 (A, B, C) = $\sum$m (1, 2, 3, 4, 5)

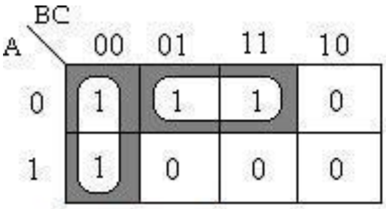
   Implement the circuit with a PLA having 3 inputs, 4 product terms and 2 outputs

<u>Solution:</u>

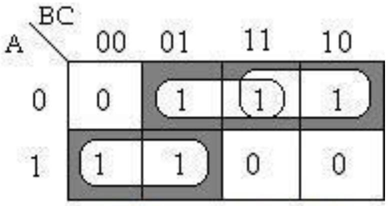
Step 1: Truth table for the given functions

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

Step 2: K-map Simplification



$F_1 = B'C' + A'C$



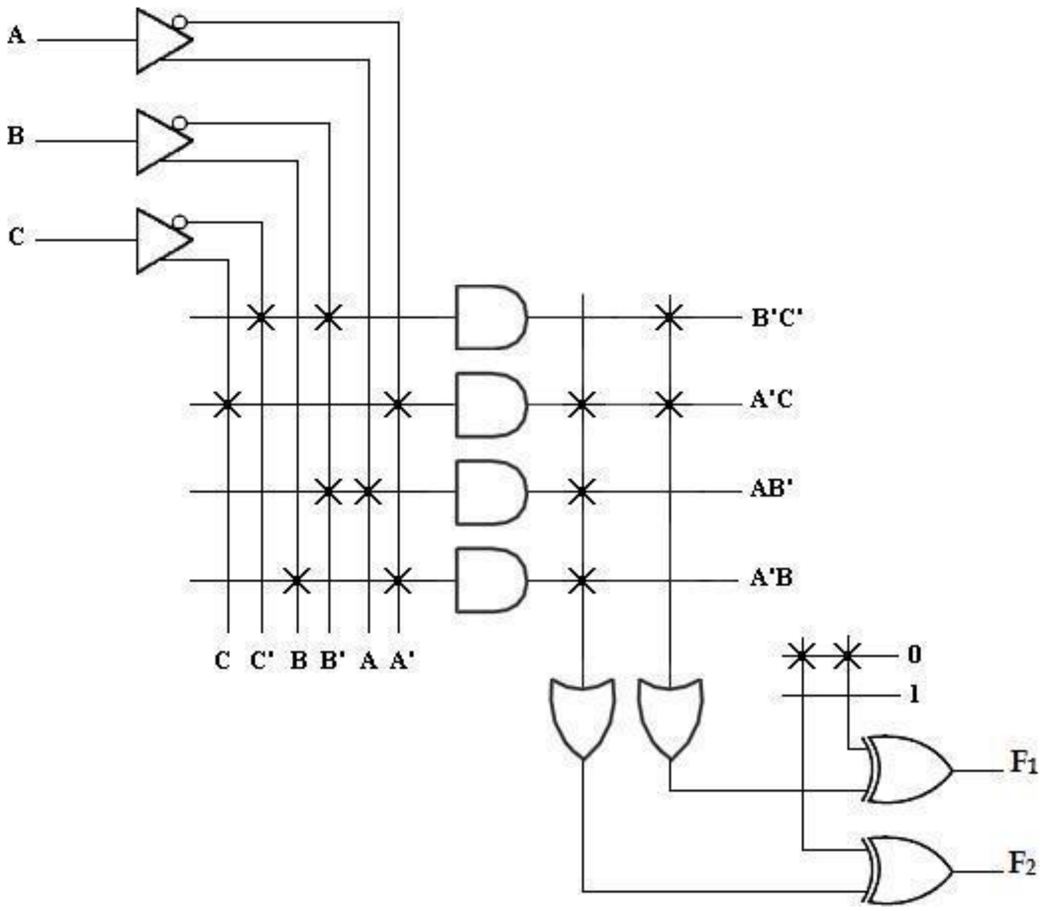
$F_2 = AB' + A'C + A'B$

The product terms are **B′C′, A′C, AB′ and A′B**

Step 3: PLA Program table

	Product term	Input s			Outputs	
		A	B	C	F_1 (T)	F_2 (T)
B′C′	1	-	0	0	1	-
A′C	2	0	-	1	1	1
AB′	3	1	0	-	-	1
A′B	4	0	1	-	-	1

Step 4: PLA Diagram



7. A combinational logic circuit is defined by the function,

F (A, B, C, D) = ∑m (3, 4, 5, 7, 10, 14, 15)

G (A, B, C, D) = ∑m (1, 5, 7, 11, 15)

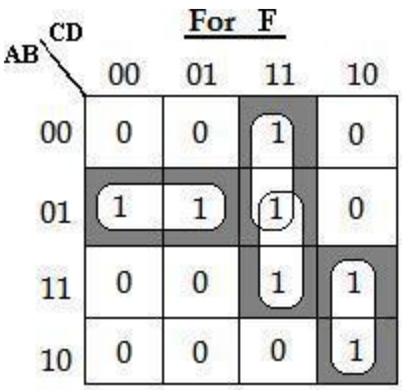
Implement the circuit with a PLA having 4 inputs, 6 product terms and 2 outputs

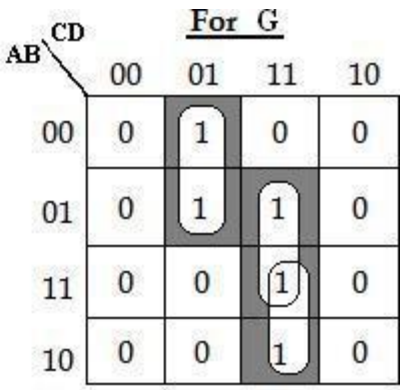
Solution:

Step 1: Truth table for the given functions

A	B	C	D	F	G
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	0
0	0	1	1	1	0
0	1	0	0	1	0
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	1	0
1	0	1	1	0	1
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	1	0
1	1	1	1	1	1

Step 2: K-map Simplification





F= A'BC'+ A'CD+ BCD+ ACD'

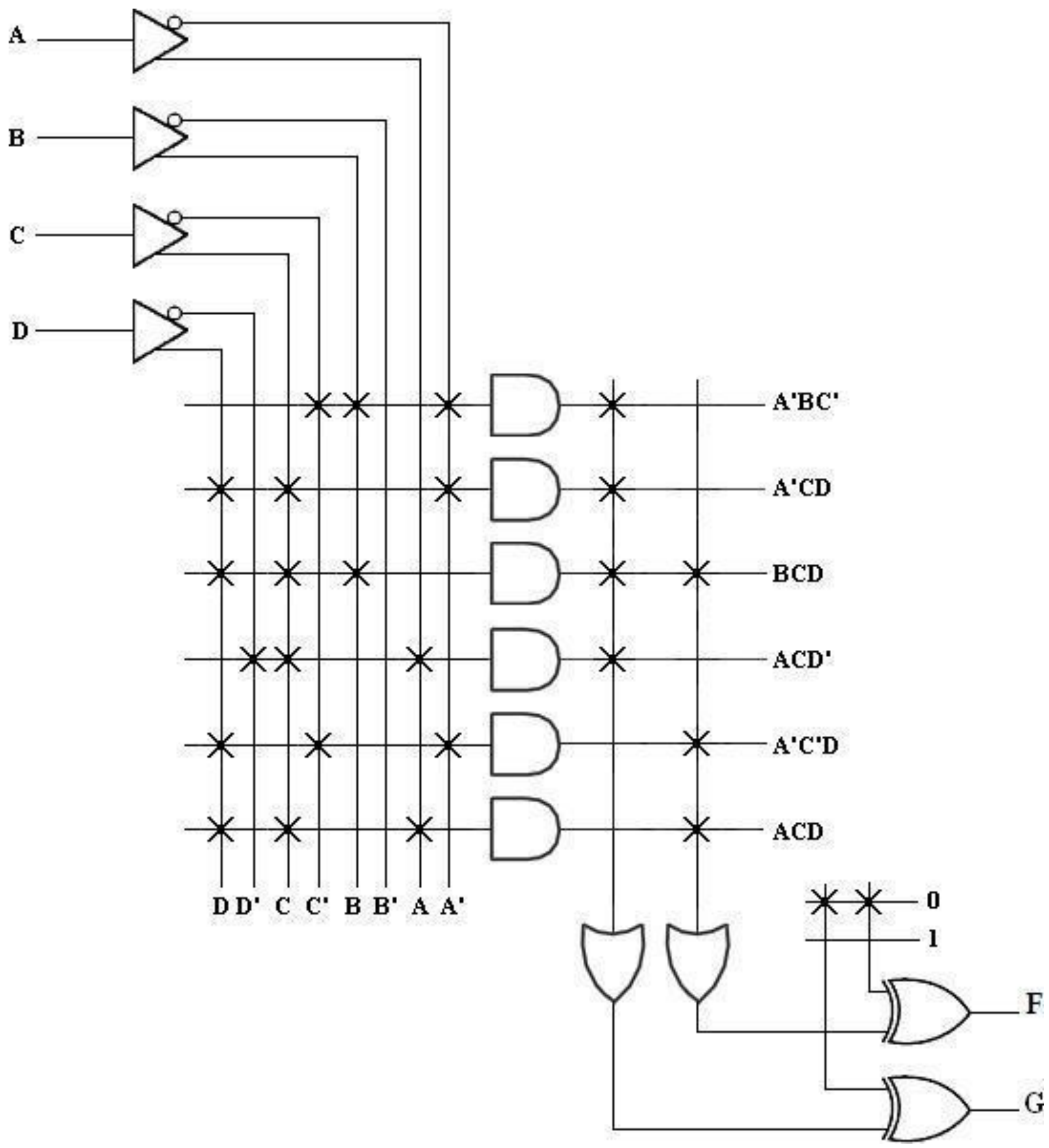
G= A'C'D+ BCD+ ACD

The product terms are A‘BC‘, A‘CD, BCD, ACD‘, A‘C‘D, ACD

Step 3: PLA Program table

	Product term	Inputs				Outputs	
		A	B	C	D	F (T)	G (T)
A‘BC‘	1	0	1	0	-	1	-
A‘CD	2	0	-	1	1	1	-
BCD	3	-	1	1	1	1	1
ACD‘	4	1	-	1	0	1	-
A‘C‘D	5	0	-	0	1	-	1
ACD	6	1	-	1	1	-	1

Step 4: PLA Diagram



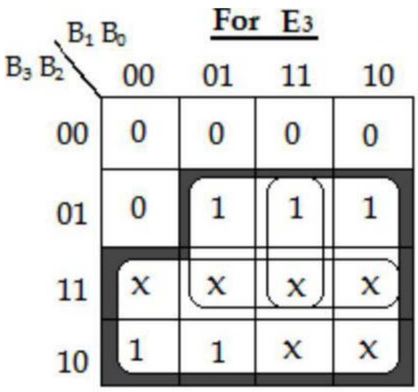
8 Design a BCD to Excess-3 code converter and implement using suitable PLA

Solution:

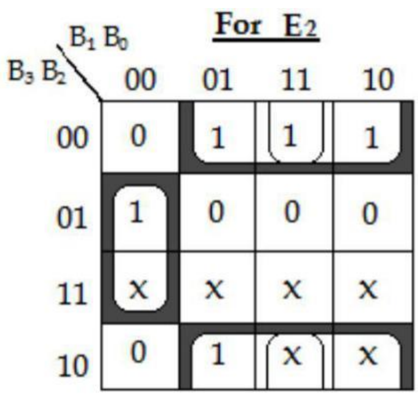
Step 1: Truth table of BCD to Excess-3 converter is shown below,

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

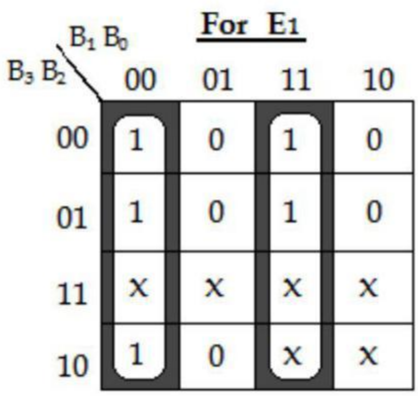
Step 2: K-map Simplification



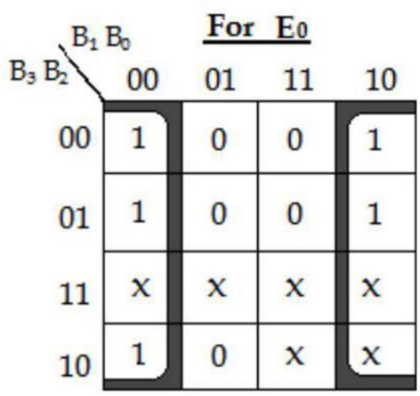
$E_3 = B_3 + B_2B_0 + B_2B_1$



$E_2 = B_2B_1'B_0' + B_2'B_0 + B_2'B_1$



$E_1 = B_1'B_0' + B_1B_0$



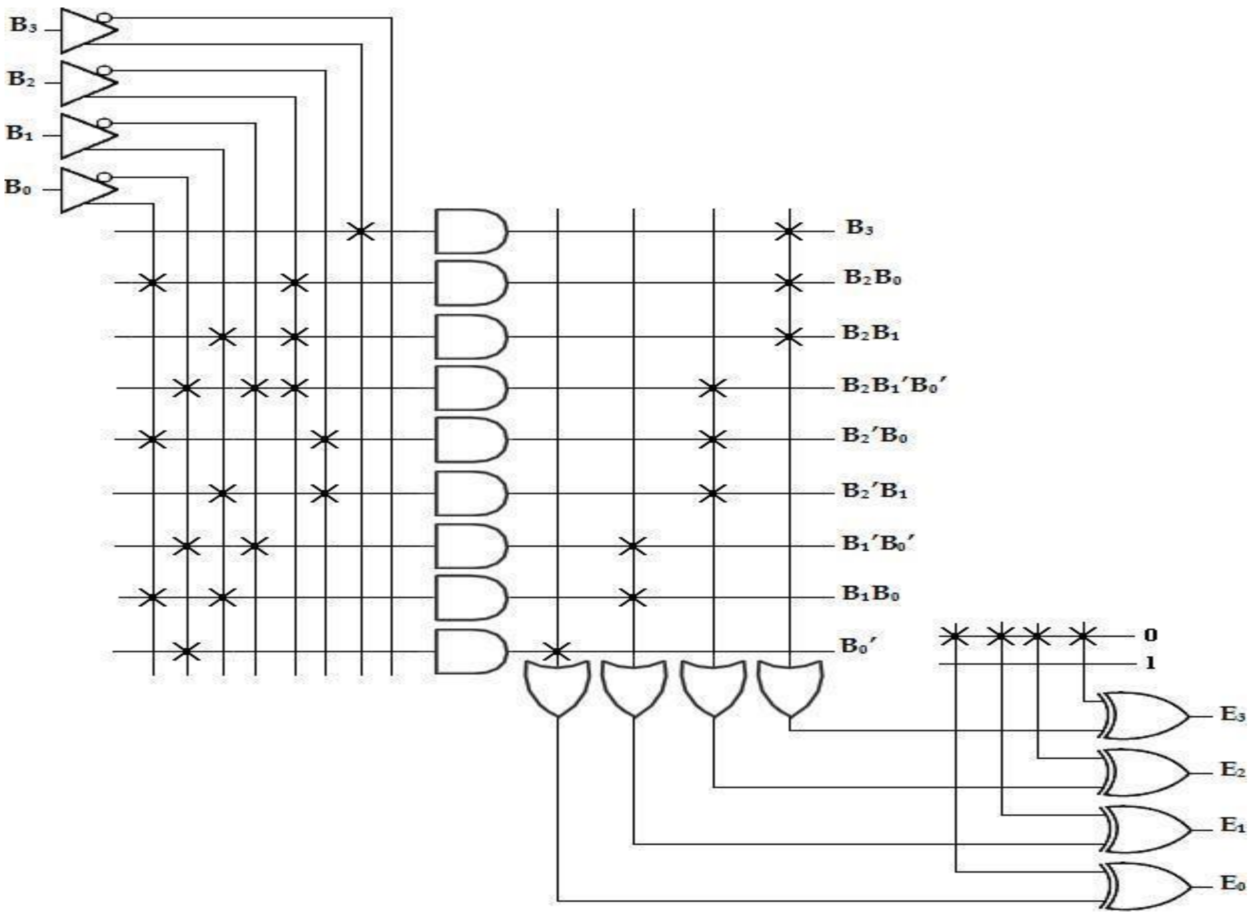
$E_0 = B_0'$

The product terms are B_3, B_2B_0, B_2B_1, $B_2B_1'B_0'$, $B_2'B_0$, $B_2'B_1$, $B_1'B_0'$, B_1B_0, B_0'

Step 3: PLA Program table

	Product terms	Inputs				Outputs			
		B_3	B_2	B_1	B_0	E_3 (T)	E_2 (T)	E_1 (T)	E_0 (T)
B_3	1	1	-	-	-	1	-	-	-
B_2B_0	2	-	1	-	1	1	-	-	-
B_2B_1	3	-	1	1	-	1	-	-	-
$B_2B_1'B_0'$	4	-	1	0	0	-	1	-	-
$B_2'B_0$	5	-	0	-	1	-	1	-	-
$B_2'B_1$	6	-	0	1	-	-	1	-	-
$B_1'B_0'$	7	-	-	0	0	-	-	1	-
B_1B_0	8	-	-	1	1	-	-	1	-
B_0'	9	-	-	-	0	-	-	-	1

Step 4: PLA Diagram



Comparison between PROM, PLA, and PAL:

SNo	PROM	PLA	PAL
1	AND array is fixed and OR array is programmable	Both AND and OR arrays are programmable	OR array is fixed and AND array is programmable
2	Cheaper and simpler to use	Costliest and complex	Cheaper and simpler
3	All minterms are decoded	AND array can be programmed to get desired minterms	AND array can be programmed to get desired minterms
4	Only Boolean functions in standard SOP form can be implemented using PROM	Any Boolean functions in SOP form can be implemented using PLA	Any Boolean functions in SOP form can be implemented using PLA