First course

Lectured One

1- Number Systems Operation:

1- Decimal Numbers.

2- Binary Numbers.

3 - Hexadecimal Numbers.

1- Decimal Numbers: In the decimal number system each of the ten digits (10digits), 0 through 9 (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9).

Decimal weight … …10^4 10^3 10^2 10^1 10^0. 10^{-1} 10^{-2} 10^{-3} ….

Example (1): $\mathbf{(345)_{10}}$

$\mathbf{300+40+5=10^2*3+10^1*4+10^0*5=345= (345)_{10}}$

↓ ↓ ↓

3 4 5

Example (2): $\mathbf{23.5 = (23.5)_{10}}$

$\mathbf{2*10^1 + 3*10^0 +5*10^{-1} = 20+3+0.5=23.5}$

Where 10^0 =1

2- Binary Numbers: The binary number system its two digits a base-two system. The two binary digits (bits) are 1 and 0 (1,0).

Binary weight 2^3 2^2 2^1 2^0

Weight value 8 4 2 1

A- Binary – to – Decimal Conversion:

***Binary number 1101101 where 2^0 =1**

1 1 0 1 1 0 1

$\mathbf{2^6 \quad 2^5 \quad 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0 = 2^6 *1+ 2^5*1+2^4*0+2^3*1+2^2*1+2^1*0+2^0*1}$

$\mathbf{= 64+32+0+8+4+0+1=96+13=109}$ ➔$\mathbf{(109)_{10}}$

*The fractional binary number 0.1011

$$\begin{array}{lllll} 0. & 1 & 0 & 1 & 1 \\ & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} \end{array} = 1*2^{-1}+ 0*2^{-2}+1*2^{-3}+1*2^{-4} =$$

$$0.5+0+0.125+0.0625=0.6875 \rightarrow (0.6875)_{10}$$

B- Decimal – to – Binary Conversion:

1- Convert a decimal whole number to binary using the repeated division – by – 2 method.

2- Convert a decimal fraction to binary using the repeated Multiplication – by – 2 method.

Example (1):

Number $(58)_{10}$ ====➔$(111010)_2$

2	58	mod	LSB
2	29	==➔0	↑
2	14	==➔1	
2	7	==➔ 0	=======➔ $(111010)_2$
2	3	==➔1	
2	1	==➔1	
	0	==➔1	
			MSB

Example (2):

Number $(0.3125)_{10}$ ======➔$(0101)_2$

MSB	carry	
↓		0.3125*2
	0	0.6250*2
	1	0.2500*2
	0	0.5000*2
	1	0.0000
LSB		

$(0101)_2$

4- Hexadecimal Numbers: The hexadecimal number system has a base of sixteen; it is composed of 16 digits and alphabetic characters.

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

Lecture (2)

A- Binary – to – Hexadecimal conversion:

4-bit groups, starting at the right-most bit.

Example: $(1100101001010111)_2$ =======➔ $(CA57)_{16}$

1100 1010 0101 0111

C A 5 7

B- Hexadecimal – to – Binary Conversion:

Example: $(10A4)_{16}$ =========➔ $(1000010100100)_2$

1 0 A 4

0001 0000 1010 0100

C- *Hexadecimal – to –Decimal Conversion:* By to method

* **First method:**

Example: $(A85)_{16}$ ====➔ $(2693)_{10}$

1- Convert to binary number.

2- Convert from binary number to decimal number.

A 8 5

1010 1000 0101 =

$2^{11}*1+2^{10}*0+2^9*1+2^8*0+2^7*1+2^6*0+2^5*0+2^4*0+2^3*0+2^2*1+2^1*0+2^0*1=$

$2^{11}+2^9+2^7+2^2+2^0=2048+512+128+4+1=2693=(2693)_{10}$

* **Second method:**

Example: $(E5)_{16}$ ========➔ $(229)_{10}$

$(E5)_{16}=E*16^1+5*16^0=14*16+5*1=224+5=229=(229)_{10}$

D- *Decimal – to – Hexadecimal Conversion:*

Example: **Convert the decimal number 650 to hexadecimal by repeated division by 16.**

$(650)_{10}$ =====➔ $(28A)_{16}$

		Mod	LSD
16	650		
16	40	======➔A	
16	2	======➔8	
	0	======➔2	
			MSD

MSD 2 8 A LSD = $(28A)_8$

2-Binary Arithmetic:

1- Binary Addition.

2- Binary Subtraction.

3- Binary Multiplication.

4- Binary Division.

1- ***Binary Addition:*** The four basic rules for adding binary digits (bits) are as follows.

0+0=0	**Sum of 0 with a carry 0**
0+1=1	**Sum of 1 with a carry 0**
1+0=1	**Sum of 1 with a carry 0**
1+1=1 0	**Sum of 0 with a carry 1**

Examples:

$$\begin{array}{r} 110 \\ +\,100 \\ \hline 1010 \end{array} \qquad \begin{array}{r} 6 \\ +4 \\ \hline 10 \end{array} \qquad\qquad \begin{array}{r} 111 \\ +011 \\ \hline 1010 \end{array} \qquad \begin{array}{r} 7 \\ +3 \\ \hline 10 \end{array}$$

$$\begin{array}{r} 1111 \\ +\;\;1100 \\ \hline 11011 \end{array} \qquad \begin{array}{r} 15 \\ +12 \\ \hline 27 \end{array}$$

2- ***Binary Subtraction:*** The four basic rules for subtracting are as follows.

0-0=0

1-1=0

1-0=1

0-1=1 0-1 with a borrow of 1

Examples:

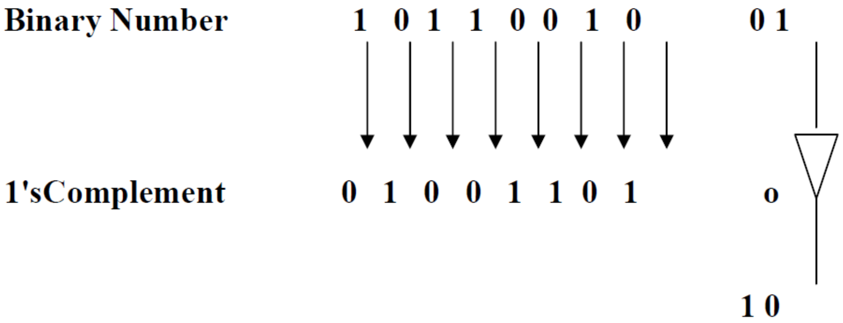
$$\begin{array}{r} 11 \\ -\ 01 \\ \hline 10 \end{array} \qquad \begin{array}{r} 3 \\ -\ 1 \\ \hline 2 \end{array} \qquad \begin{array}{r} 11 \\ -\ 10 \\ \hline 01 \end{array} \qquad \begin{array}{r} 3 \\ -\ 2 \\ \hline 1 \end{array} \qquad \begin{array}{r} 101 \\ -\ 011 \\ \hline 010 \end{array} \qquad \begin{array}{r} 5 \\ -\ 3 \\ \hline 2 \end{array}$$

$$\begin{array}{r} 110 \\ -\ 101 \\ \hline 001 \end{array} \qquad \begin{array}{r} 6 \\ -\ 5 \\ \hline 1 \end{array} \qquad \begin{array}{r} 101101 \\ -\ 001110 \\ \hline 011111 \end{array} \qquad \begin{array}{r} 45 \\ -\ 14 \\ \hline 31 \end{array}$$

Lecture(3)

3- *1's And 2's Complement of Binary Number:*

The 1's complement and the 2's complement of binary number are important because they permit the representation of negative numbers.



2's Complement of a binary number is found by adding 1 to the LSB of the 1's Complement.

2's Complement= (1's Complement) +1

Binary number 10110010

1'scomplement 01001101

Add 1 + 1

2's complement 01001110

In decimal number complement such as:

0====□9

7====□2

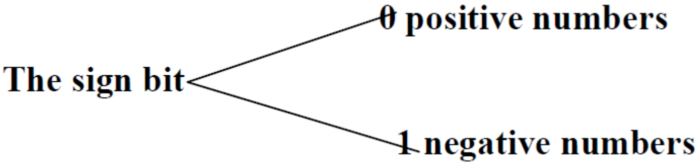
6====□3

9====□0

4====□5

1====□8

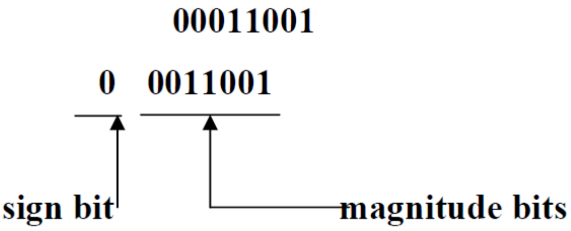
Signed Numbers: Signed binary number consists of both sign and magnitude information.



The sign bit

0 positive numbers

1 negative numbers



00011001

0 0011001

sign bit

magnitude bits

Example: **Express the decimal number - 39 as an 8-bit number in the sign-magnitude, 1's complement, and 2's complement forms.**

Solution:

1- Write the 8-bit number for +39 00100111

2- 1's complement 11011000

3- Add 1 1

1 1011001 = - 39

sign bit negative

4- ***Hexadecimal Addition & Subtraction:***

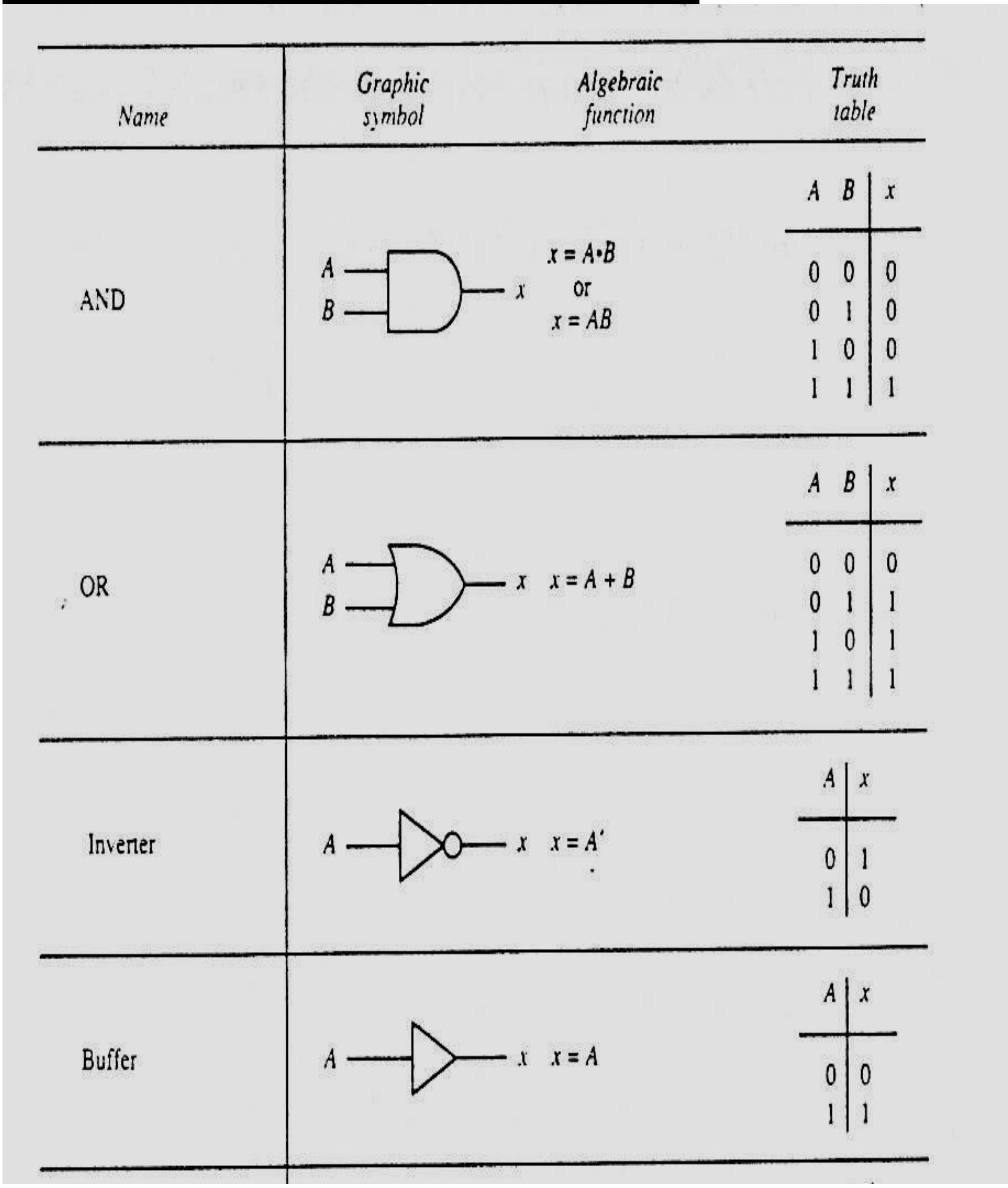
Hexadecimal Addition:

$$\begin{array}{r} 2A7 \\ +\ 317 \\ \hline 5BE \end{array} \qquad \begin{array}{r} 2AB \\ +317 \\ \hline 5C2 \end{array} \qquad \begin{array}{r} 2B \\ +\ 84 \\ \hline AF \end{array}$$

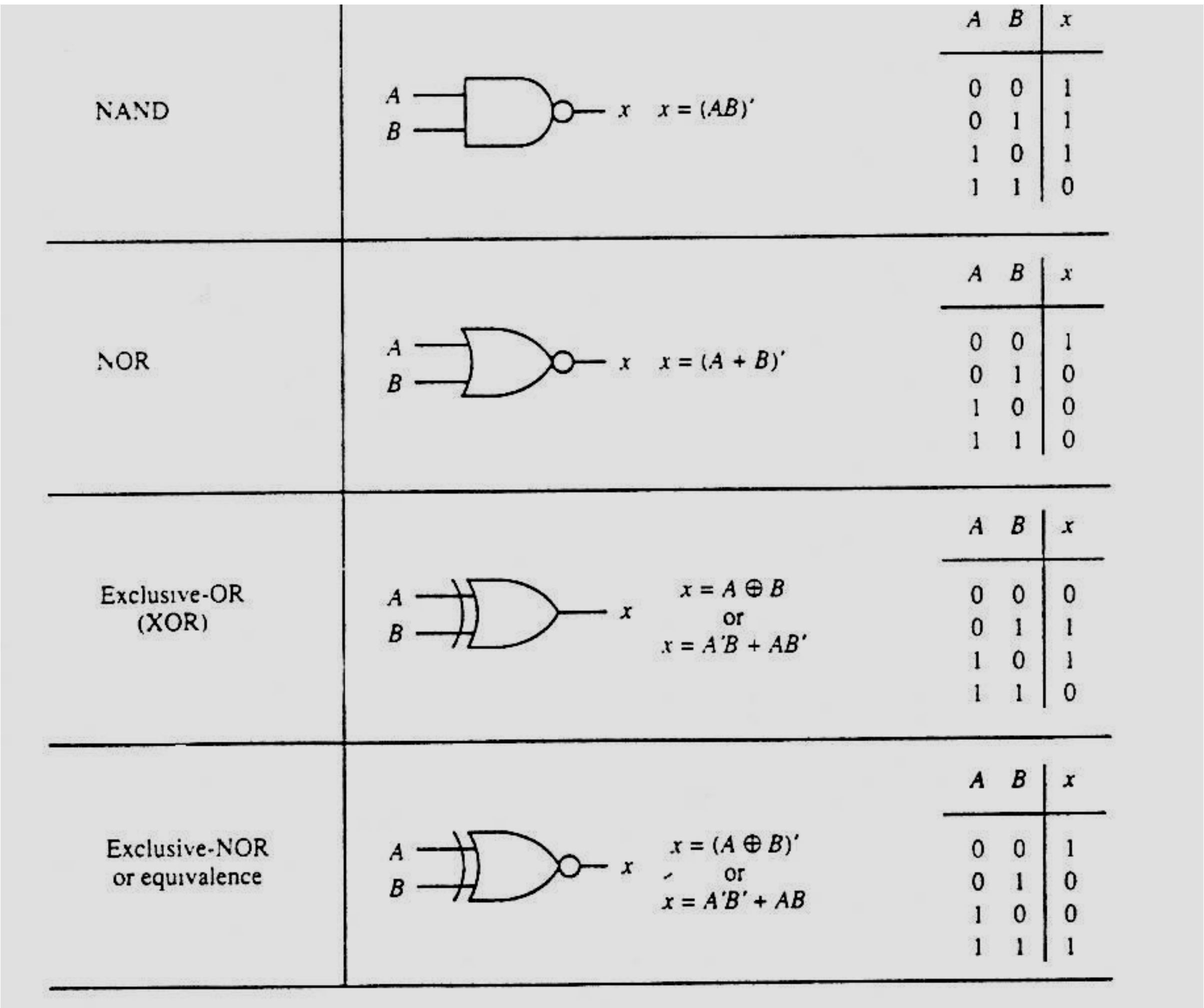
Hexadecimal subtraction:

$$\begin{array}{r} CA2 \\ -\ A1B \\ \hline 287 \end{array} \qquad \begin{array}{r} 47C \\ -\ 2BE \\ \hline 1BE \end{array}$$

Lectured (4) Logic Gats: Set of Gats

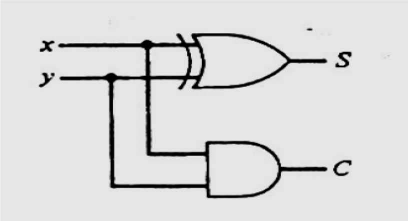


Name	Graphic symbol	Algebraic function	Truth table
AND		$x = A \cdot B$ or $x = AB$	A B | x 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$x = A + B$	A B | x 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$x = A'$	A | x 0 | 1 1 | 0
Buffer		$x = A$	A | x 0 | 0 1 | 1



			A	B	x
NAND	A, B → x	x = (AB)'	0	0	1
			0	1	1
			1	0	1
			1	1	0
NOR	A, B → x	x = (A + B)'	0	0	1
			0	1	0
			1	0	0
			1	1	0
Exclusive-OR (XOR)	A, B → x	x = A ⊕ B or x = A'B + AB'	0	0	0
			0	1	1
			1	0	1
			1	1	0
Exclusive-NOR or equivalence	A, B → x	x = (A ⊕ B)' or x = A'B' + AB	0	0	1
			0	1	0
			1	0	0
			1	1	1

2- Half – Adder: The basic digital arithmetic circuit is the addition of two binary digits. Input variables of a half-adder call augends & addend bits. The output variables the sum & carry.



X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Figure (1-a) Logic diagram for half adder

Figure (1-b) Truth table for half adder

<u>Half- Adder questions:</u>

$S=\overline{X}Y+X\overline{Y}$

S=X (+)Y

C=X*Y

3-Full-Adder: A full - adder is a combinational circuit that forms the arithmetic sum of three input bits. It consists of three inputs &two outputs.

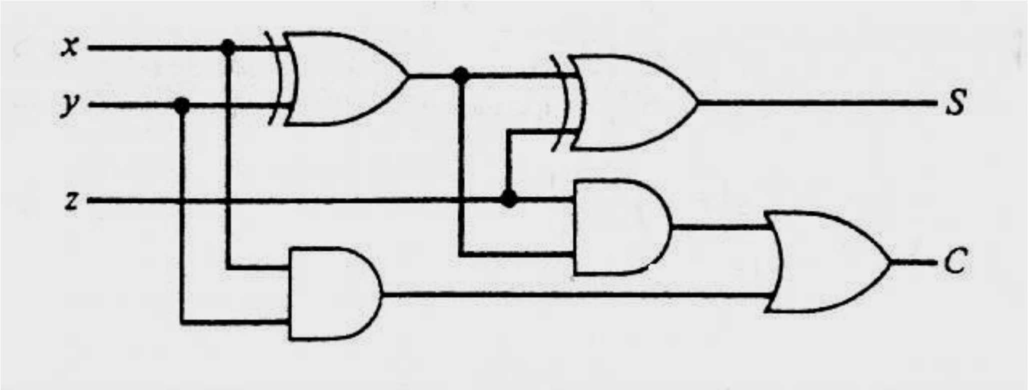


Figure (2-a) Logic diagram for full adder (Logic Diagram)

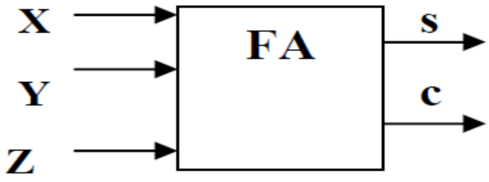


Figure (2-b) Block diagram for full adder

Inputs			Out puts	
X	Y	Z	C	S
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

Figure (2-c) Truth table for full adder

Full - Adder questions:

$S=x\ (+)\ y\ (+)\ z$

$C=XY+\ (XZ\ (+)\ YZ)$

$C=X*Y+\ (X\ (+)\ Y)\ Z$

Lecture (5)

Boolean Algebra &Logic Simplification:

1-Rules of Boolean algebra:

1- $A+0=A$

2- $A+1=1$

3- $A*0=0$

4- $A*1=A$

5- $A+A=A$

6- $A+\overline{A}=1$

7- $A*A=A$

8- $A*\overline{A}=0$

9- $\overline{\overline{A}}=A$ ======➔**Demoragan's theorems**

10- $A+BA=A$

11- $A+\overline{A}B=A+B$

12- $(A+B)(A+C)=A+BC$

2- Examples:

Example 1:

F = X + ýz

Determine the truth table and logic diagram

X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

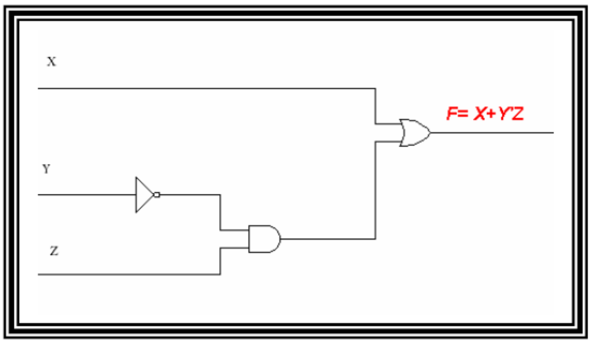


Figure (3-a) Truth table **figure (3-b) Logic diagram**

Example 2:

AB+ A (B+C)+ B(B+C)

1. **AB+AB+AC+BB+BC**
2. **AB+AB+AC+B+BC**
3. **AB+AC+B+BC**
4. **AB+AC+B**
5. **B+AC**

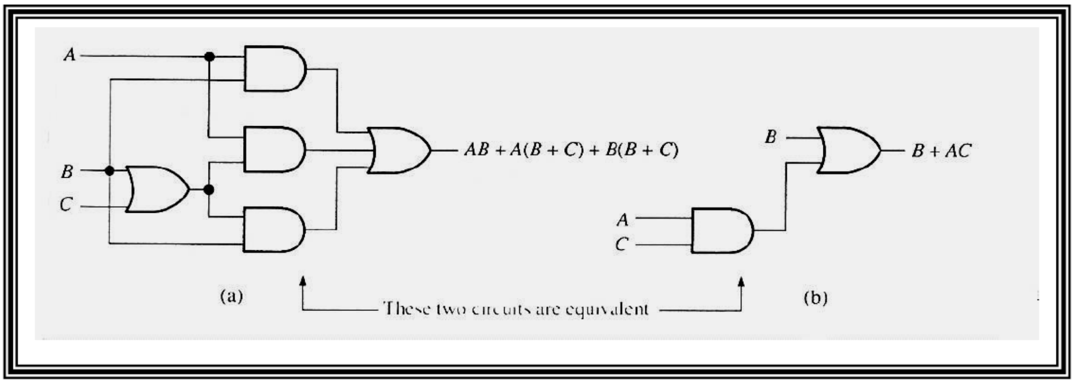


Figure (4)

Example 3:

F=ABC+ABĆ+ĂC

F= AB(C+Ć) +ĂC

F= AB+ĂC

Note: More laws of Boolean algebra

1-Commutative Law: (a) A + B = B + A (b) A B = B A

2-Associate Law: (a) (A + B) + C = A + (B + C) (b) (A B) C = A (B C)

3-Distributive Law: (a) A (B + C) = A B + A C

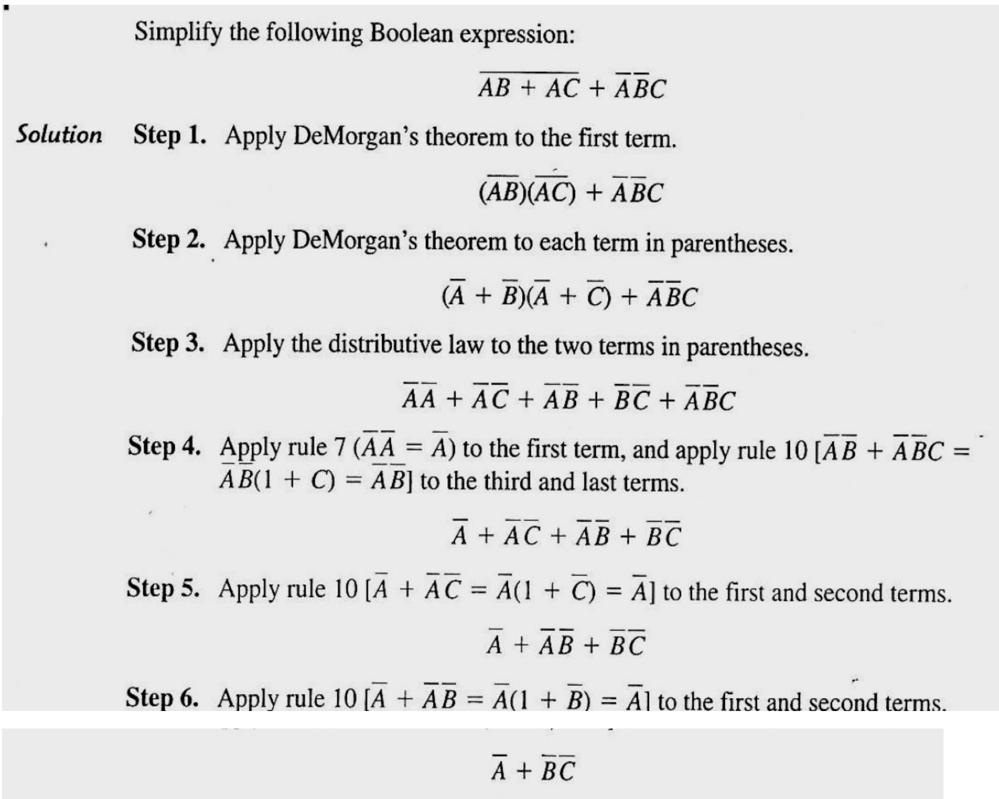
(b) A + (B C) = (A + B) (A + C)

3-De Morgan's Theorem: (a) (A+B)′= A′B′ (b) (AB)′= A′+ B′

5- Absorption: (a) A + A B = A (b) A (A + B) = A

Lecture (6)

Example 4:



Simplify the following Boolean expression:

$$\overline{AB + AC} + \overline{A}\,\overline{B}C$$

Solution **Step 1.** Apply DeMorgan's theorem to the first term.

$$(\overline{AB})(\overline{AC}) + \overline{A}\,\overline{B}C$$

Step 2. Apply DeMorgan's theorem to each term in parentheses.

$$(\overline{A} + \overline{B})(\overline{A} + \overline{C}) + \overline{A}\,\overline{B}C$$

Step 3. Apply the distributive law to the two terms in parentheses.

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

Step 4. Apply rule 7 ($\overline{A}\,\overline{A} = \overline{A}$) to the first term, and apply rule 10 [$\overline{A}\,\overline{B} + \overline{A}\,\overline{B}C = \overline{A}\,\overline{B}(1 + C) = \overline{A}\,\overline{B}$] to the third and last terms.

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

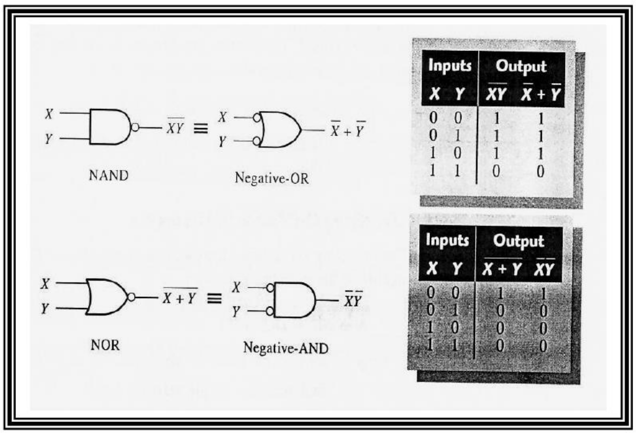
Step 5. Apply rule 10 [$\overline{A} + \overline{A}\,\overline{C} = \overline{A}(1 + \overline{C}) = \overline{A}$] to the first and second terms.

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

Step 6. Apply rule 10 [$\overline{A} + \overline{A}\,\overline{B} = \overline{A}(1 + \overline{B}) = \overline{A}$] to the first and second terms.

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

3- Demorgan's theorems:



Inputs		Output	
X	Y	$\overline{XY}$	$\overline{X} + \overline{Y}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Inputs		Output	
X	Y	$\overline{X + Y}$	$\overline{X}\,\overline{Y}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Figure (5) Demorgan's theorems

Example 1:

a- $\overline{\overline{(A+B)}+\overline{C}} = \overline{\overline{(A+B)}}\;\overline{\overline{C}} = (A+B)C$

b- $\overline{(\overline{A}+B)+CD} = \overline{(\overline{A}+B)}\;\underline{CD} = (\underline{\overline{A}}\,\overline{B})\;(\overline{C}+\overline{D}) = A\,\overline{B}\,(\overline{C}+\overline{D})$

c- $\overline{(A+B)\,\overline{C}\;\overline{D}+E+\overline{F}} = ((\overline{(A+B)\,\overline{C}\;\overline{D})}\;(\overline{E+\overline{F})}$

$= (\overline{A}\,\overline{B}+C+D)\;(\overline{E}\,F)$

5- Sum – Of – Products (SOP):

X=AB+BCD+AC

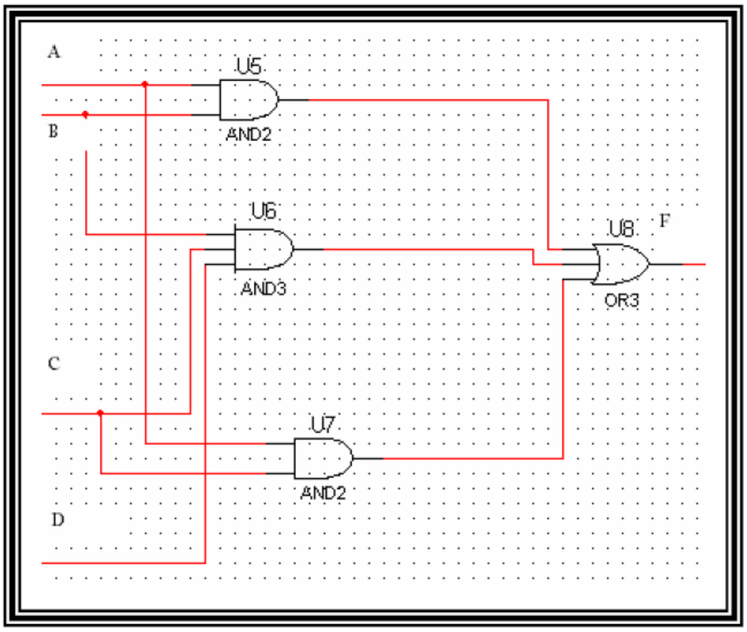


Figure (4) SOP

Examples:

a- $AB+B(CD+EF)=AB+BCD+BEF$

b- $(A+B)(B+C+D)=AB+AC+AD+BB+BC+BD$

c- $\overline{\overline{(A+B)}+C}=\overline{\overline{(A+B)}}*\overline{C}\ \ =(A+B)\overline{C}=A\overline{C}+B\overline{C}$

6- Product – Of – Sum(POS):

(A+B)(B+C+D)(A+C)

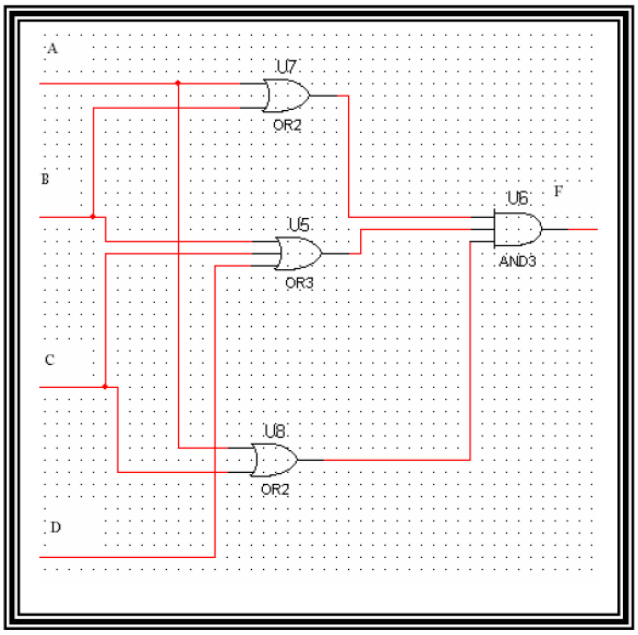
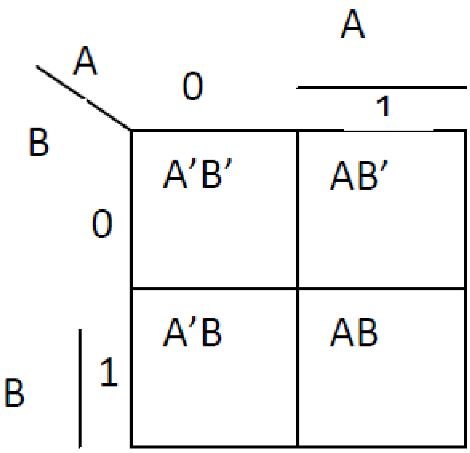


Figure (5) POS

Lecture (7)

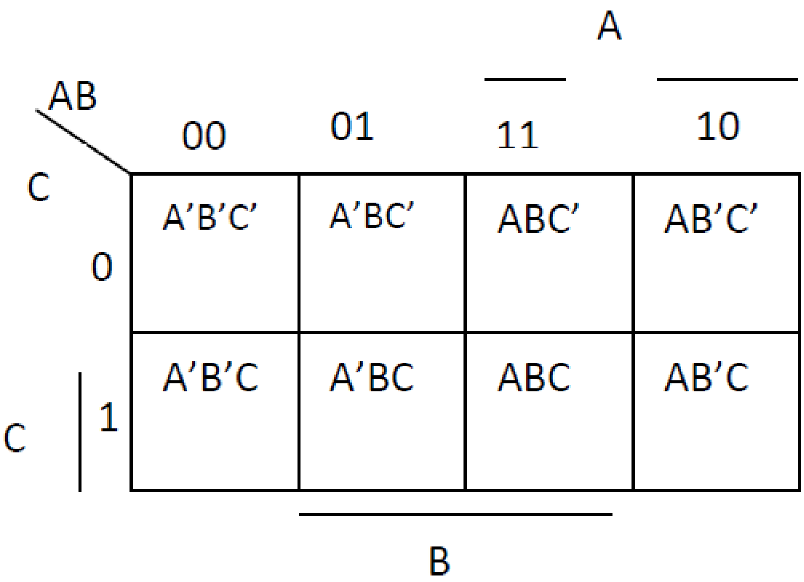
Karnaugh map

The Karnaugh map also known as Veitch diagram or simply as K map is a two dimensional form of the truth table,
drawn in such a way that the simplification of Boolean expression can be immediately be seen from the location of 1's
in the map. The map is a diagram made up of squares , each sqare represent one minterm. Since any Boolean function
can be expressed as a sum of minterms, it follows that a Boolean function is recognised graphically in the map from the
area enclosed by those squares whose minterms are included in the function.
A two variable Boolean function can be represented as follow



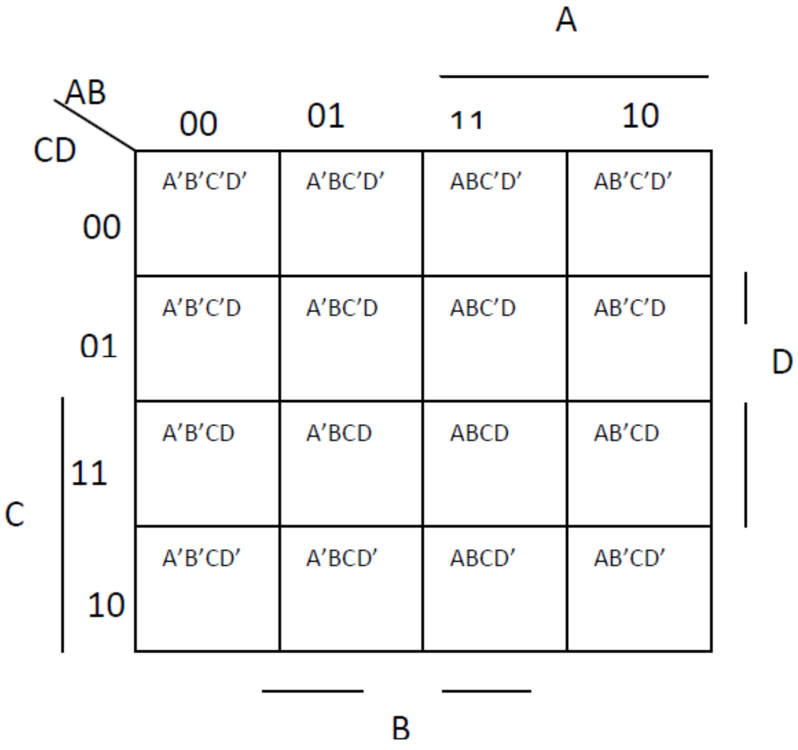
B \ A	0	1
0	A'B'	AB'
1	A'B	AB

A three variable function can be represented as follow



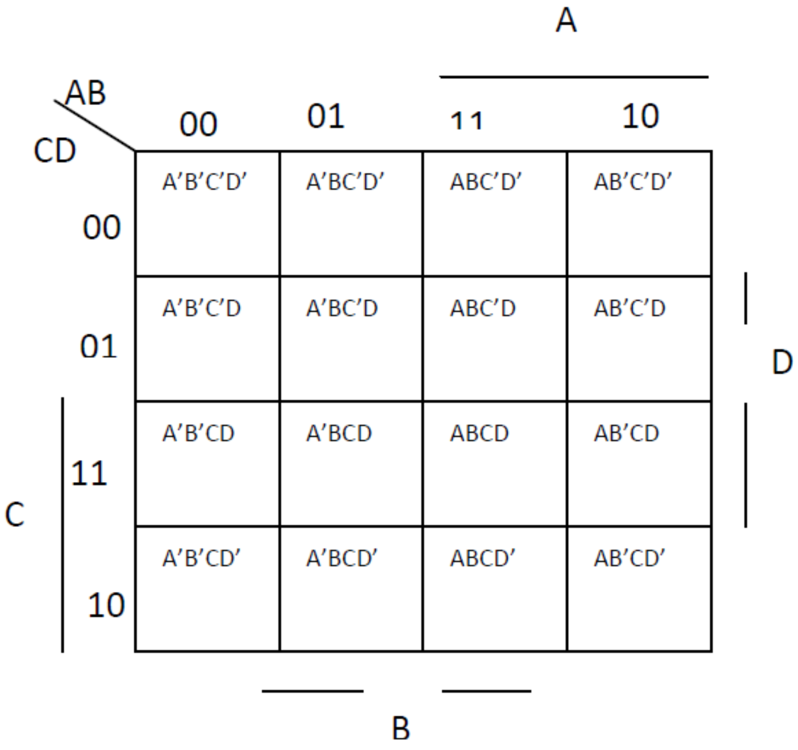
C \ AB	00	01	11	10
0	A'B'C'	A'BC'	ABC'	AB'C'
1	A'B'C	A'BC	ABC	AB'C

A four variable Boolean function can be represented in the map bellow



CD \ AB	00	01	11	10
00	A’B’C’D’	A’BC’D’	ABC’D’	AB’C’D’
01	A’B’C’D	A’BC’D	ABC’D	AB’C’D
11	A’B’CD	A’BCD	ABCD	AB’CD
10	A’B’CD’	A’BCD’	ABCD’	AB’CD’

A four variable Boolean function can be represented in the map bellow



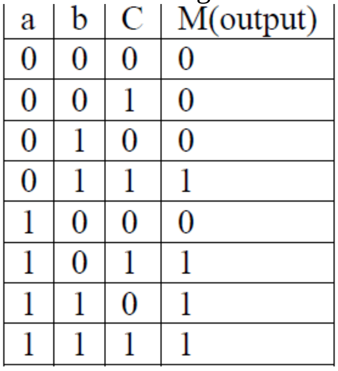
CD \ AB	00	01	11	10
00	A’B’C’D’	A’BC’D’	ABC’D’	AB’C’D’
01	A’B’C’D	A’BC’D	ABC’D	AB’C’D
11	A’B’CD	A’BCD	ABCD	AB’CD
10	A’B’CD’	A’BCD’	ABCD’	AB’CD’

To simplify a Boolean function using karnaugh map, the first step is to plot all ones in the function truth table on the

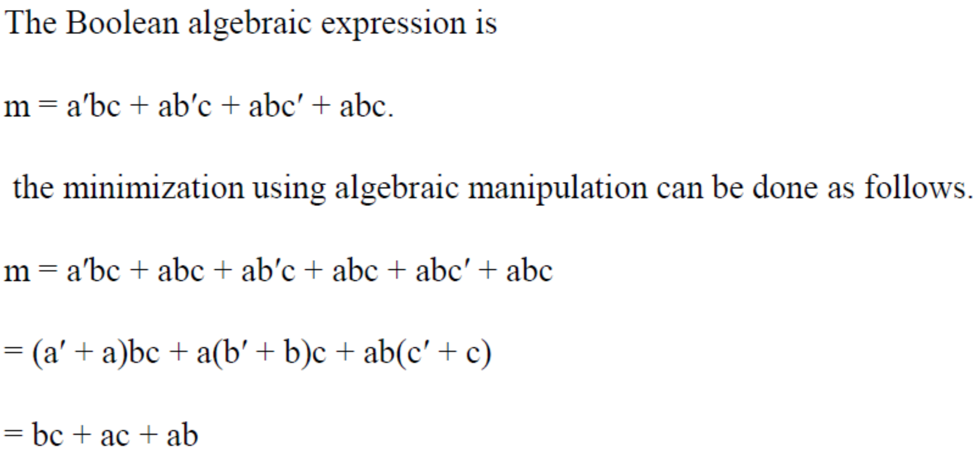
map. The next step is to combine adjacent 1’s into a group of one, two, four, eight, sixteen. The group of minterm
should be as large as possible. A single group of four minterm yields a simpler expression than two groups of two
minterms.
In a four variable karnaugh map
variable product term is obtained if 8 adjacent squares are covered
2 variable product term is obtained if 4 adjacent squares are covered
3 variable product term is obtained if 2 adjacent squares are covered
A square having a 1 may belong to more than one term in the sum of product expression
The final stage is reached when each of the group of minterms are ORded together to form the implified sum of product expression The karnaugh map is not a square or rectangle as it may appear in the diagram. The top edge is adjacent to the bottom edge and the left hand edge adjacent to the right hand edge. Consequent, two squares in karnaugh map are said to be adjacent if they differ by only one variable.

Minimization of Boolean expressions using Karnaugh maps.

Given the following truth table for the majority function.



a	b	C	M(output)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



The Boolean algebraic expression is

$m = a'bc + ab'c + abc' + abc.$

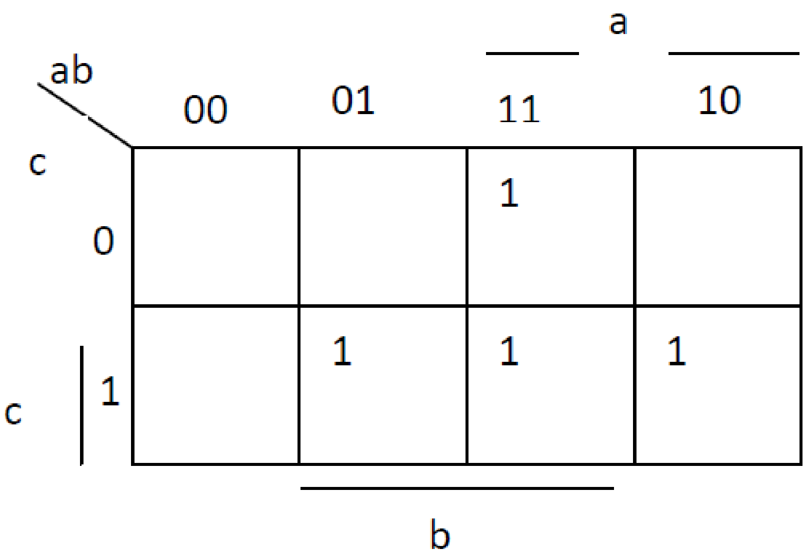
the minimization using algebraic manipulation can be done as follows.

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

$= (a' + a)bc + a(b' + b)c + ab(c' + c)$

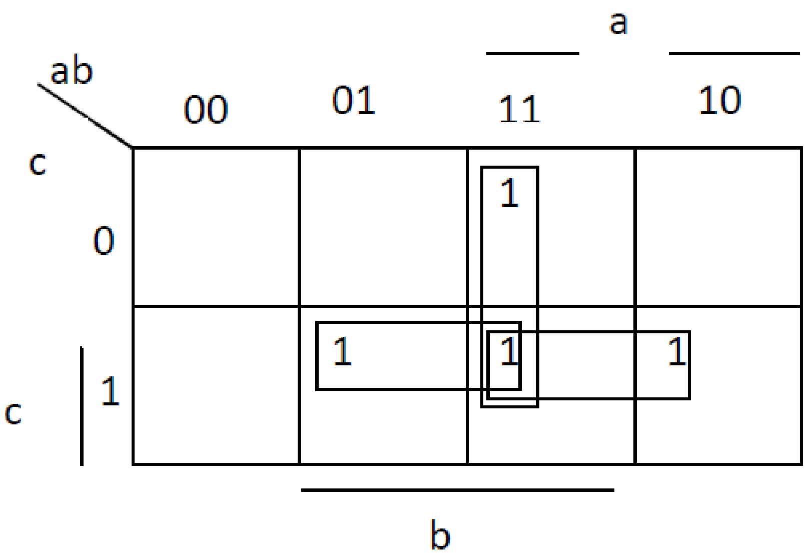
$= bc + ac + ab$

The **abc** term was replicated and combined with the other terms.
To use a Karnaugh map we draw the following map which has a position (square) corresponding to each of the 8
possible combinations of the 3 Boolean variables. The upper left position corresponds to the 000 row of the truth table,
the lower right position corresponds to 101.



The 1s are in the same places as they were in the original truth table. The 1 in the first row is at position 110 (**a** = 1, **b** =
1, **c** = 0).

The minimization is done by drawing circles around sets of adjacent 1s. Adjacency is horizontal, vertical, or both. The
circles must always contain 2_n 1s where n is an integer.

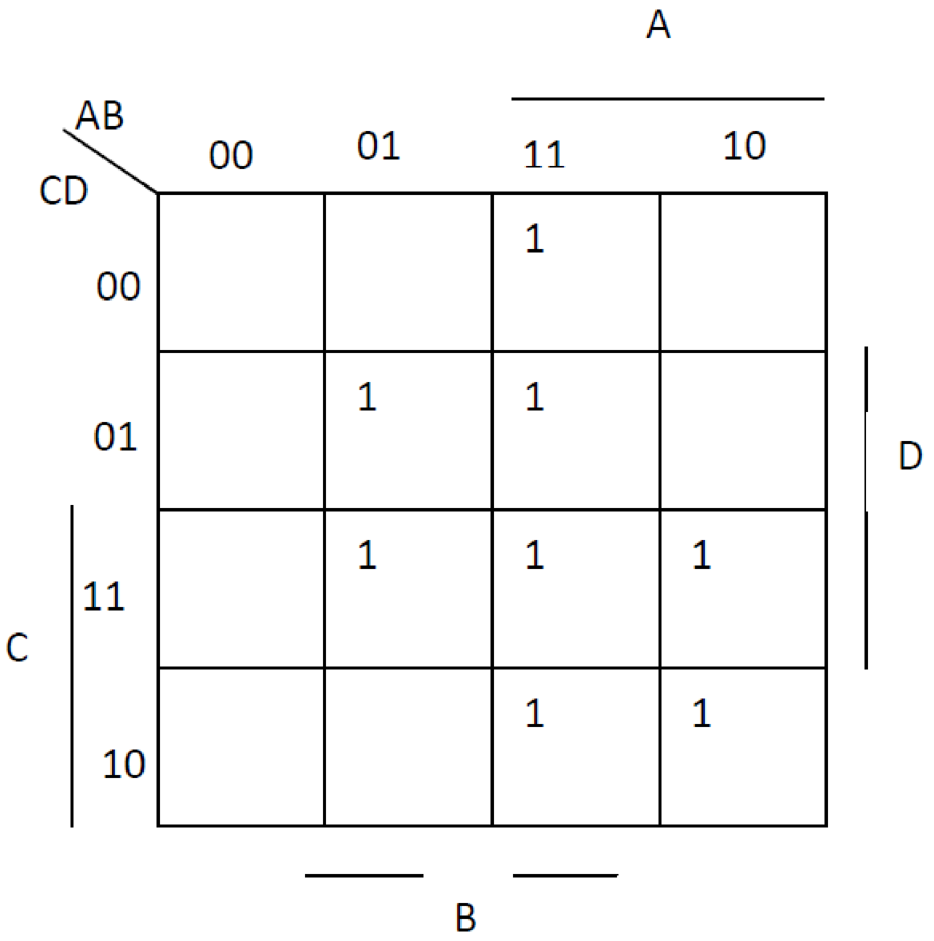


We have circled two 1s. The fact that the circle spans the two possible values of **a**
(0 and 1) means that the **a** term is eliminated from the Boolean expression corresponding to this circle.

Now we have drawn circles around all the 1s. Thus the expression reduces to
bc + ac + ab
as we saw before.

What is happening? What does adjacency and grouping the 1s together have to do with minimization? Notice that the 1
at position 111 was used by all 3 circles. This 1 corresponds to the abc term that was replicated in the original algebraic
minimization. Adjacency of 2 1s means that the terms corresponding to those 1s differ in one variable only. In one case

that variable is negated and in the other it is not.
The map is easier than algebraic minimization because we just have to recognize patterns of 1s in the map instead of
using the algebraic manipulations. Adjacency also applies to the edges of the map.
Now for 4 Boolean variables. The Karnaugh map is drawn as shown below.



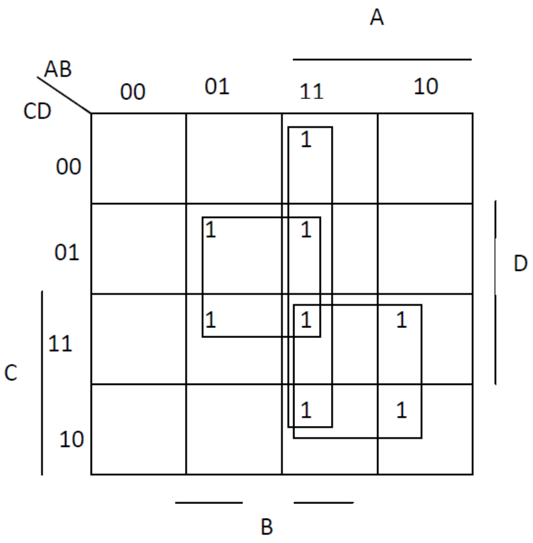
CD \ AB	00	01	11	10
00			1	
01		1	1	
11		1	1	1
10			1	1

The following corresponds to the Boolean expression

Q = A′BC′D + A′BCD + ABC′D′ + ABC′D + ABCD + ABCD′ + AB′CD + AB′CD′

RULE: Minimization is achieved by drawing the smallest possible number of circles, each containing the largest possible number of 1s.

Grouping the 1s together results in the following.



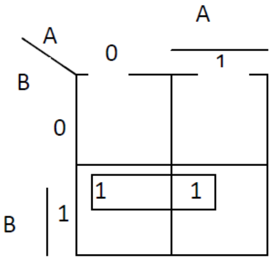
CD \ AB	00	01	11	10
00			1	
01		1	1	
11		1	1	1
10			1	1

The expression for the groupings above is

Q = BD + AC + AB

Other examples

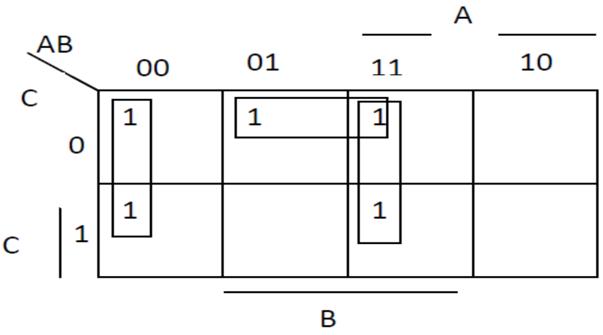
1. **F=A′B+AB**



B \ A	0	1
0		
1	1	1

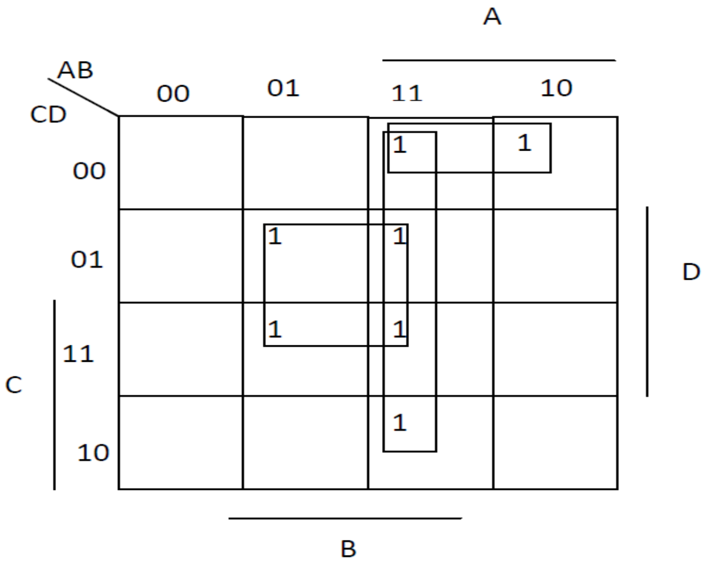
=B

2. F=A′B′C′+A′B′C+A′BC′+ABC′+ABC



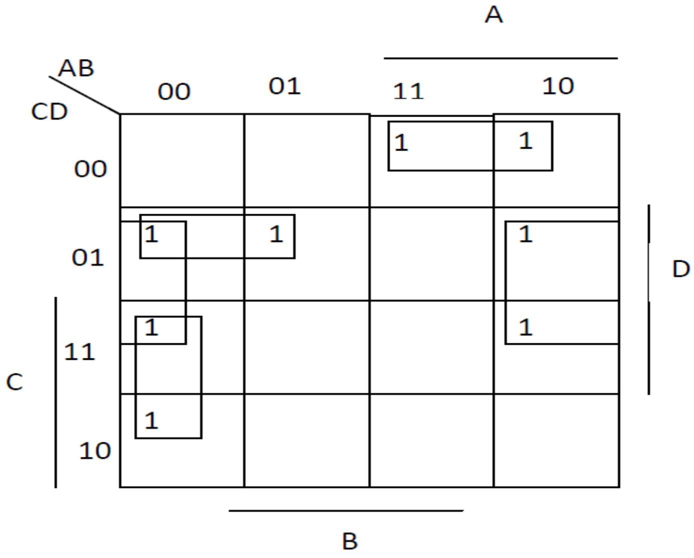
=A’B’+BC’+AB

3. F=AB+A′BC′D+A′BCD+AB′C′D′



=BD+AB+AC’D’

4. F=AC′D′+A′B′C+A′C′D+AB′D

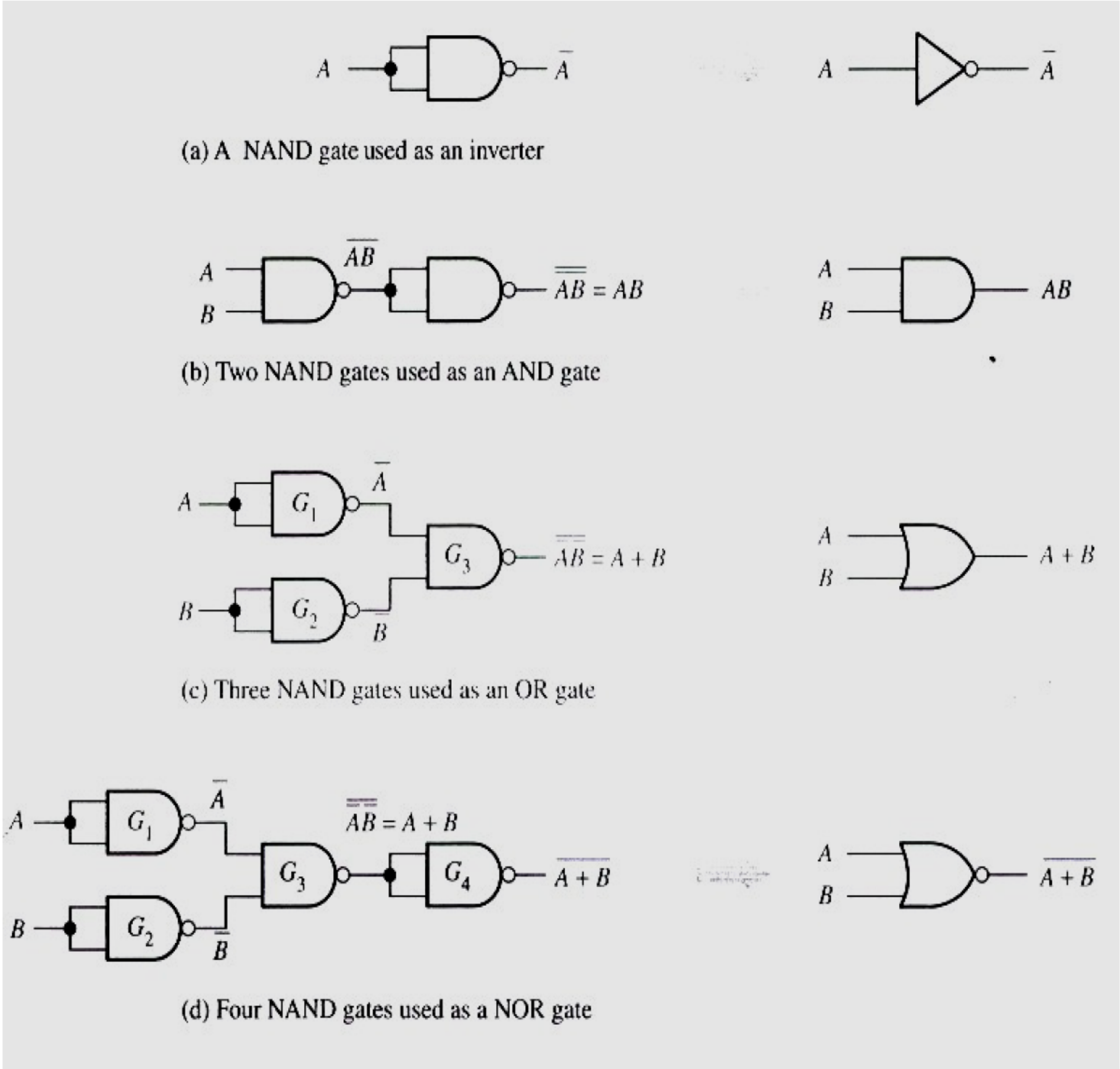


=B’D+AC’D’+A’C’D+A’B’C

Lecture (8)

Combinational Logic:

1-The NAND Gate as a Universal Logic Element:



(a) A NAND gate used as an inverter

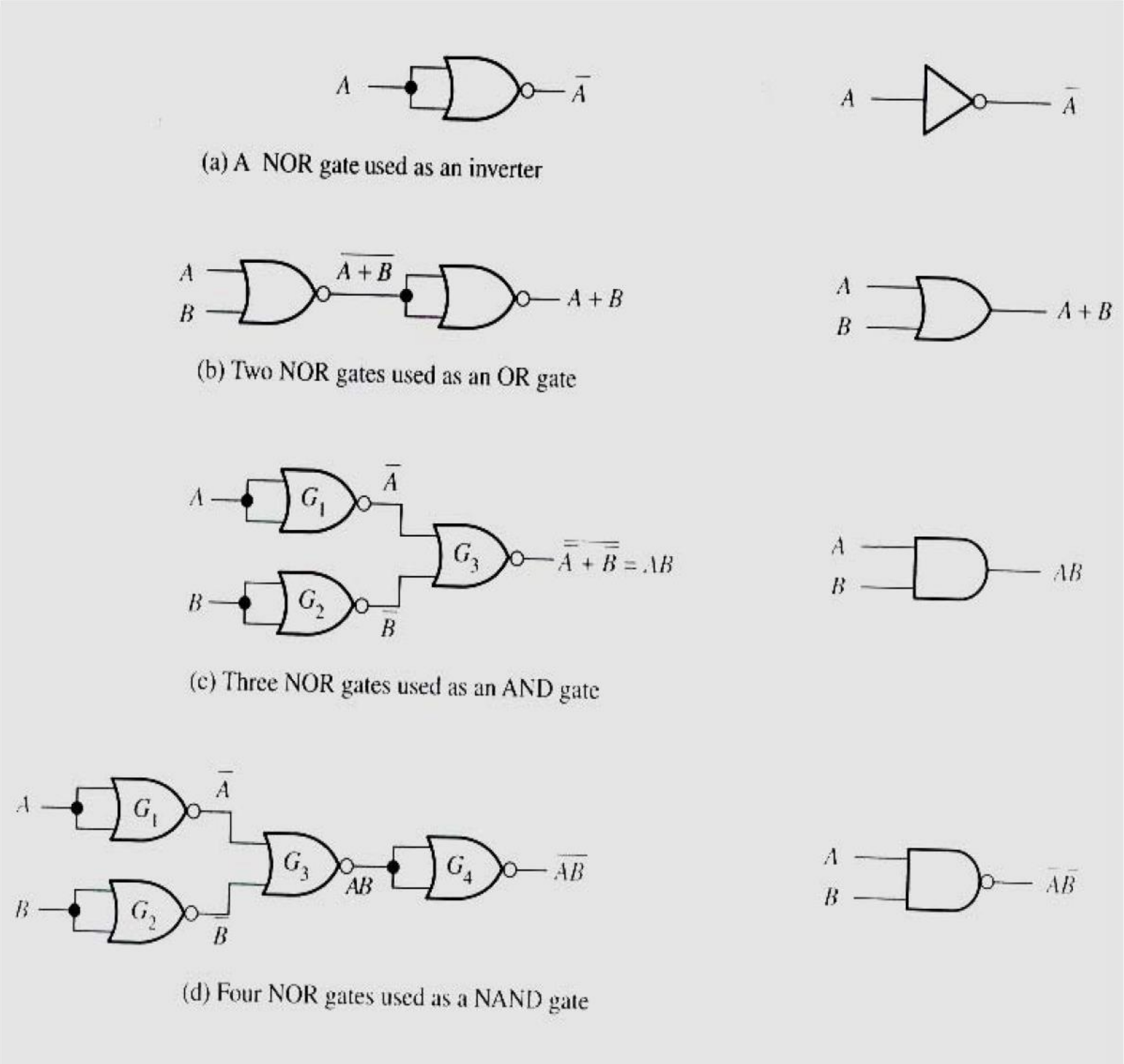
(b) Two NAND gates used as an AND gate

(c) Three NAND gates used as an OR gate

(d) Four NAND gates used as a NOR gate

Figure (9) NAND Gates

2-The NOR Gate as a Universal Logic Element:



(a) A NOR gate used as an inverter

(b) Two NOR gates used as an OR gate

(c) Three NOR gates used as an AND gate

(d) Four NOR gates used as a NAND gate

Figure (10) NOR Gates

3- 4- Bit Parallel Adder (Ripple carry adder)

A group of four bits is a ripple. A basic 4-bit parallel adder implementation with four full adder stages.

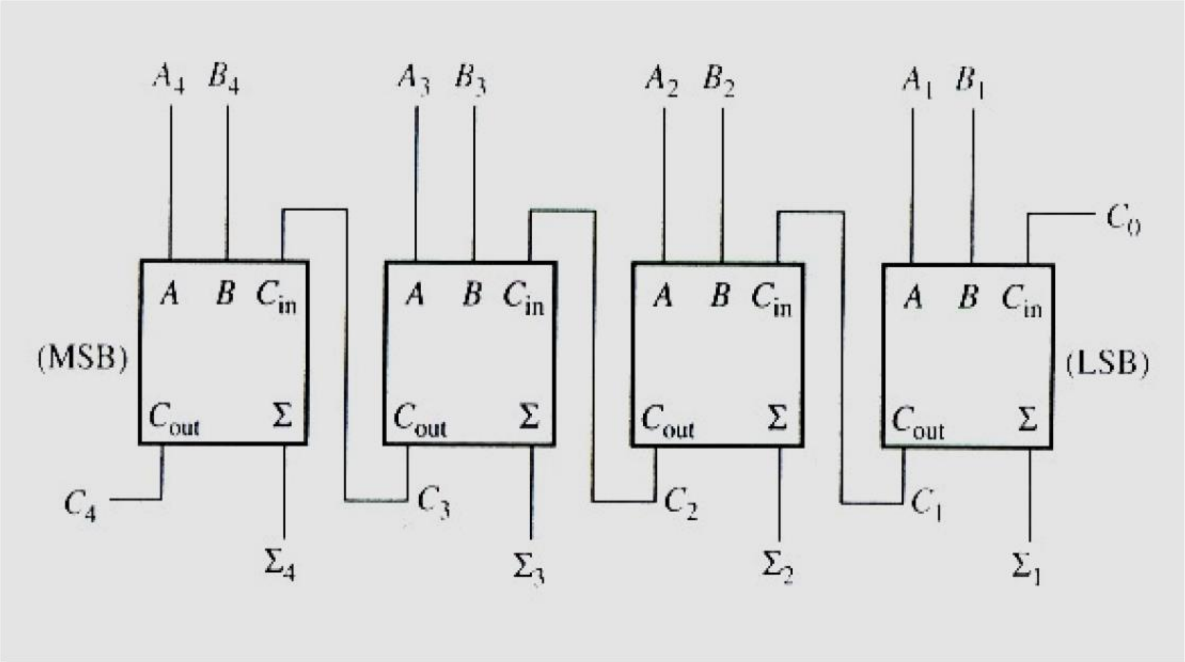


Figure (11) 4-bit parallel adder

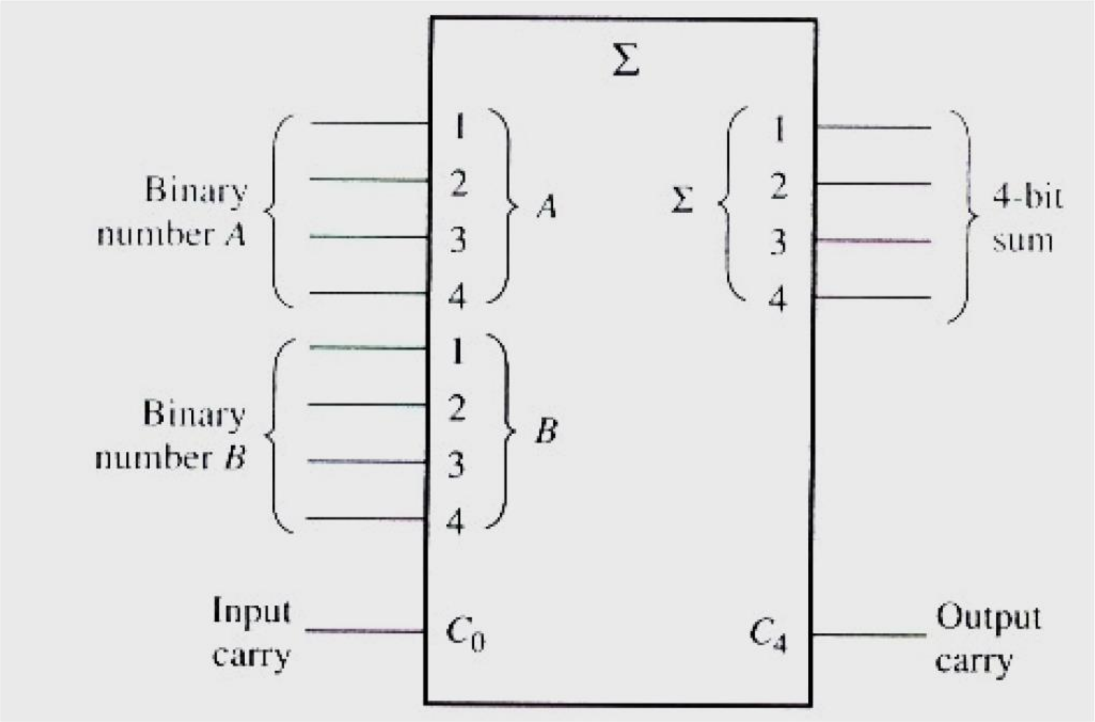


Figure (12) Symbol Logic

4- Example:

Draw the 4-bit parallel adder, find the sum and output carry for the addition of the following two 4-bit numbers if the input carry (C_{n-1}) is 0:

A4A3A2A1=1010 and B4B3B2B1=1011

Solution:

For n=1

A1=0, B1=1, C_{n-1}=0

∑ =1, and C1=0

For n=2

A2=1, B2=1, C_{n-1}=0

∑=0, and C2=1

For n=3

A3=0, B3=0, C_{n-1}=1

∑=1, and C3=0

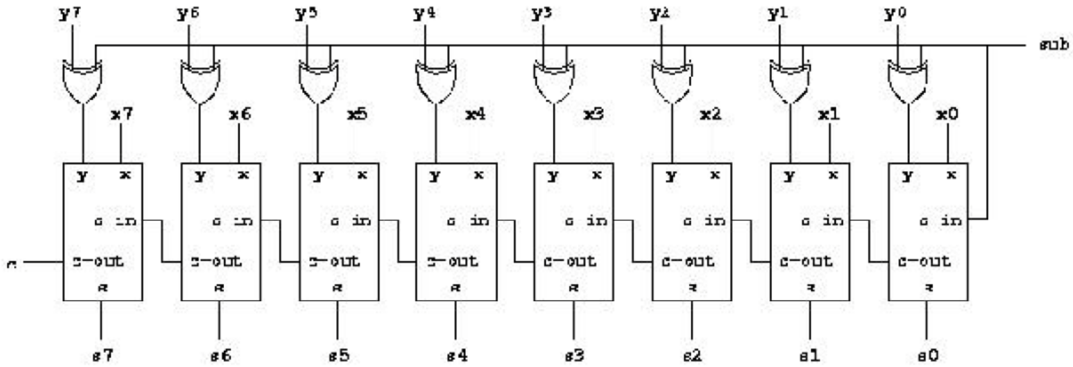
For n=4

A4=1, B4=1, C_{n-1}=0

∑=0, and C4=1

Binary subtraction using adders

We know from the section on binary arithmetic how to negate a number by inverting all the bits and adding 1. Thus, we can compute the expression as *x + inv(y) + 1*. It suffices to invert all the inputs of the second operand before they reach the adder, but how do we add the 1. That seems to require another adder just for that. Luckily, we have an unused carry-in signal to position 0 that we can use. Giving a 1 on this input in effect adds one to the result. The complete circuit with addition and subtraction looks like this:



Medium Scale integration component

The purpose of circuit minimization is to obtain an algebraic expression that, when implemented results in a low cost circuit. Digital circuit are constructed with integrated circuit(IC). An IC is a small silicon semiconductor crystal called chip containing the electronic component for digital gates. The various gates are interconnected inside the chip to form the required circuit. Digital IC are categorized according to their circuit complexity as measured by the number of logic gates in a single packages.

- Small scale integration (SSI). SSi devices contain fewer than 10 gates. The input and output of the gates are connected directly to the pins in the package.
- Medium Scale Integration. MSI devices have the complexity of approximately 10 to 100 gates in a single package
- Large Scale Integration (LSI). LSI devices contain between 100 and a few thousand gates in a single package
- Very Large Scale Integration(VLSI). VLSI devices contain thousand of gates within a single package.

Lecture (9)

Flip-Flop:

The storage elements employed in clocked sequential circuits are called flipflops.

A flip -flops is a binary cell capable of storing one bit of information. It has two outputs, one for the normal value and one for the complement value

of the bit stored in it. Type of flip-flops:

1- S-R flip-flops.

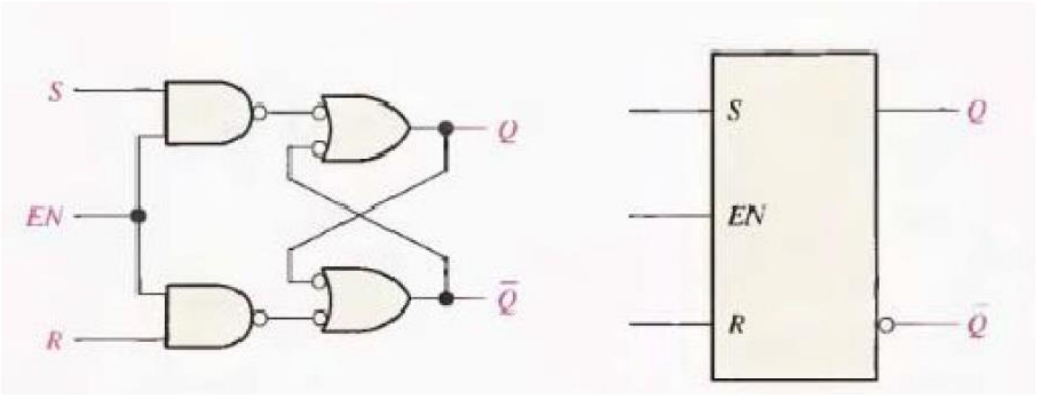
2- D flip-flops.

3- J-K flip-flops.

R-S flip flop

The most foundational flip-flop is called Reset-Set (R-S)flip-flop, the (R-S) flip-flop has three inputs and two outputs, one of the input is denoted by C and is normally a clock input. The two output are always in opposite states from each other and denoted Q and Q' because the R and S input are both ANDed with the clock (enable), they have no effect on the state of the flip flop

while the clock is 0, the following figure (14) is (a) logical diagram and (b) logical symbol of R-S flip-flop and figure (15) illustrate truth table for R-S flip_flop.



a- Logic diagram b-Logic symbol

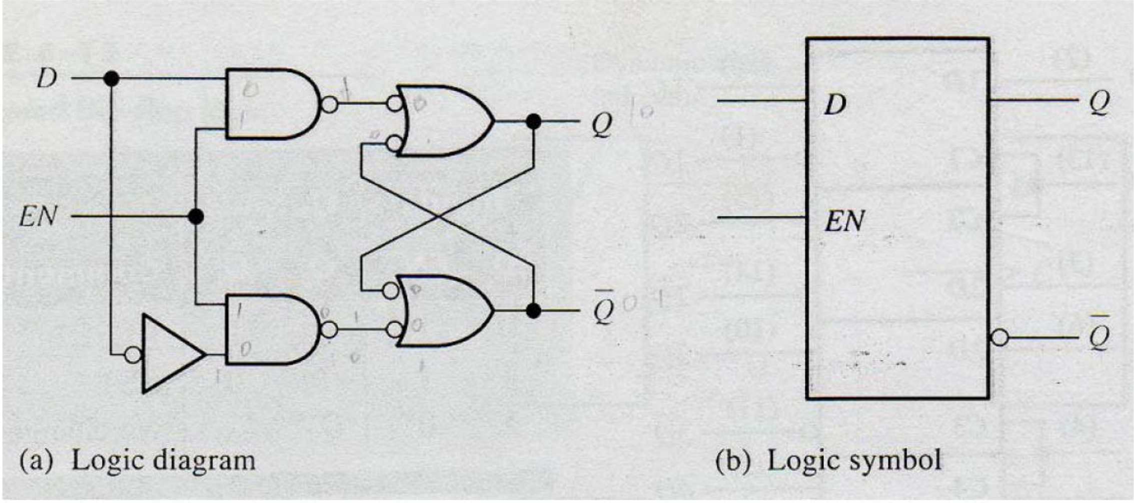
Figure (14) : R-S flip-flop

S	R	Q_n	Q_{n+1}	Comments
0	0	0	0	No change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0		unknown
1	1	1		

Figure (15): truth table for R-S flip flop

D flip-flop

A data flip-flop (D flip-flop) is one with two inputs, a clock input and input labeled D. It is easily constructed from R-S flip flop by letting D be the S and connecting R to D through an inverter, a logical diagram and logical symbol are following figure (16), and figure (17) illustrated truth table for D flip-flop.



(a) Logic diagram

(b) Logic symbol

Figure(16): D flip-flop

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

Figure(17):truth table for D flip-flop

J-K flip- flop

A J-K flip flop is an R-S flip flop that has been modified by feeding the outputs back and ANDing them with the inputs. The deference is that the J-K flip-flop has no unknown state as does the S-R flip-flop as show in following figure (18), and figure (19) illustrated a truth table for J-K flip flop.

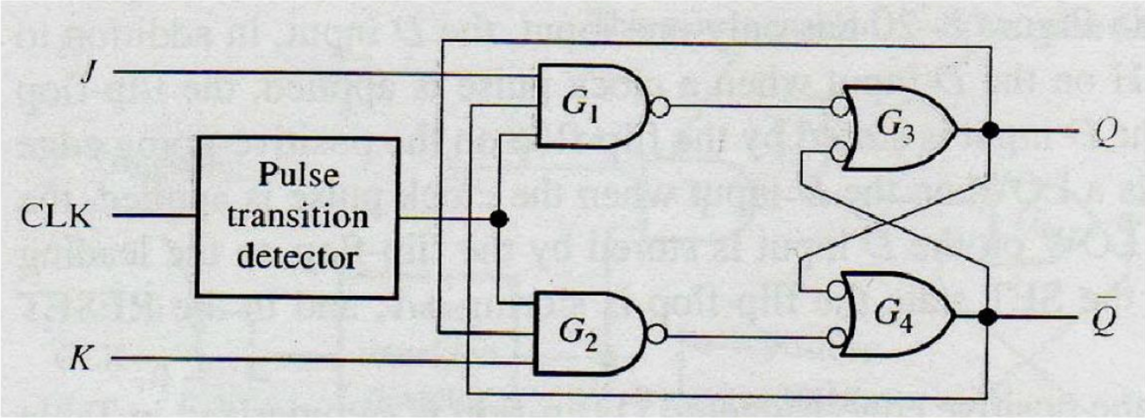


Figure (18): J-K flip flop

S	R	Q_n	Q_{n+1}	Comments
0	0	0	0	No change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	1	
1	1	1	0	

Figure(19):truth table for J-K flip-flop

Lectured (10)

Shift Register: A register is a digital circuit with two basic functions: 1- data storage, 2- data movement.

The storage capability of a register makes it an important type of memory device. The concept of storing a 1 or 0 in a D flip flop. A 1 is applied to the data input, and clock puls is applied that stores the 1 by setting the flip-flop when the 1 on the input is removed, the flip-flop remains in the set state, there by storing the 1. A similar procedure applies to the storage of a 0 by resetting the flip-flop.

Type of shift register:

1- Serial in\ Serial out shift right.

2- Serial in\ Serial out shift left.

3- Parallel in\Serial out.

4- Serial in\Parallel out.

5-Parallel in\ Parallel out.

6 Rotate right.

7- Rotate left.

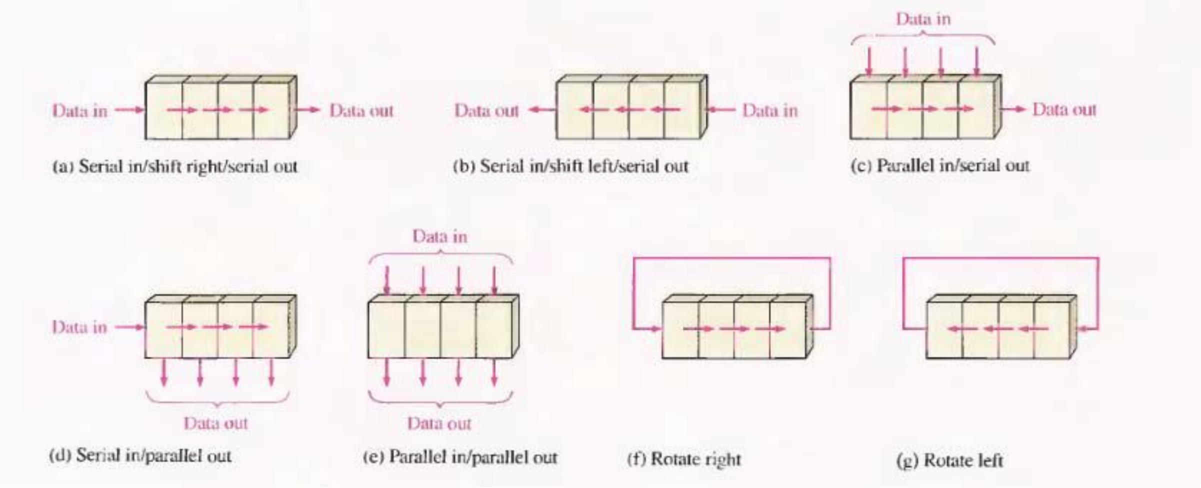

Data in
Data out
(a) Serial in/shift right/serial out
Data out
Data in
(b) Serial in/shift left/serial out
Data in
Data out
(c) Parallel in/serial out
Data in
Data out
(d) Serial in/parallel out
Data in
Data out
(e) Parallel in/parallel out
(f) Rotate right
(g) Rotate left

Figure (20) Type of shift register

1- Serial in \ Serial out shift Register:

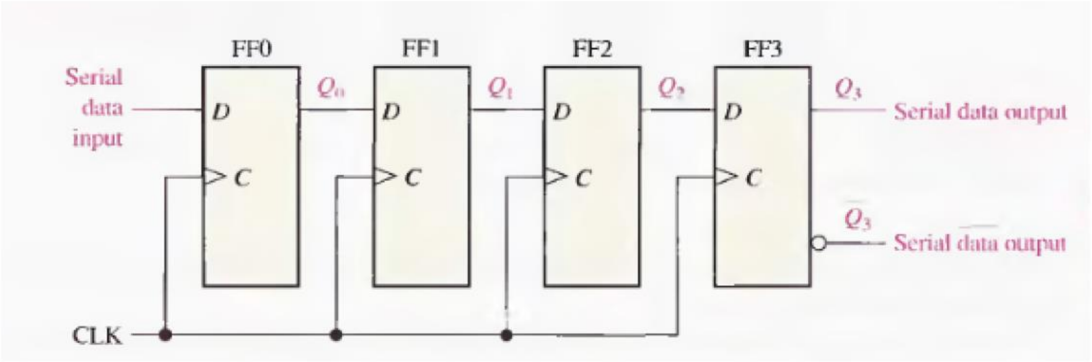


Figure (21) shift register 4-bit

Example: 1

Shift Register 4-bit

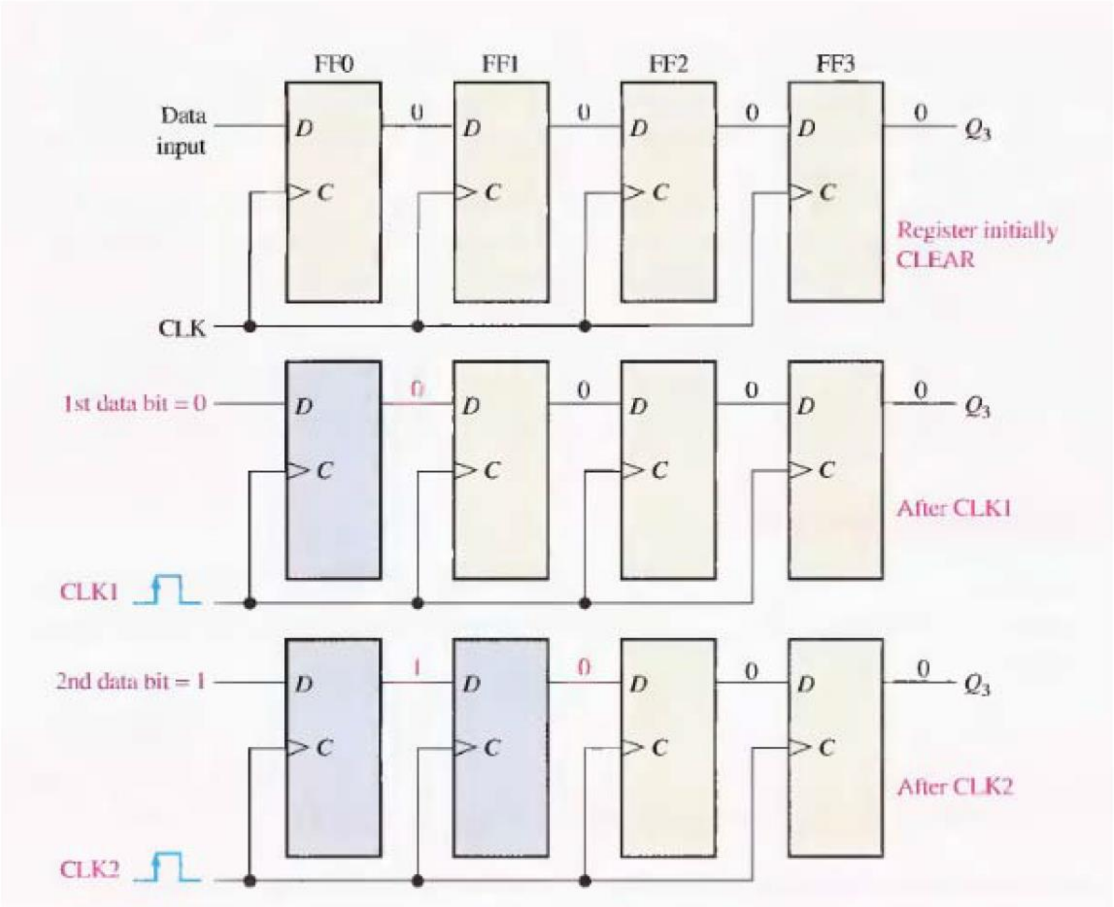


Figure (22) 4-bit shift register

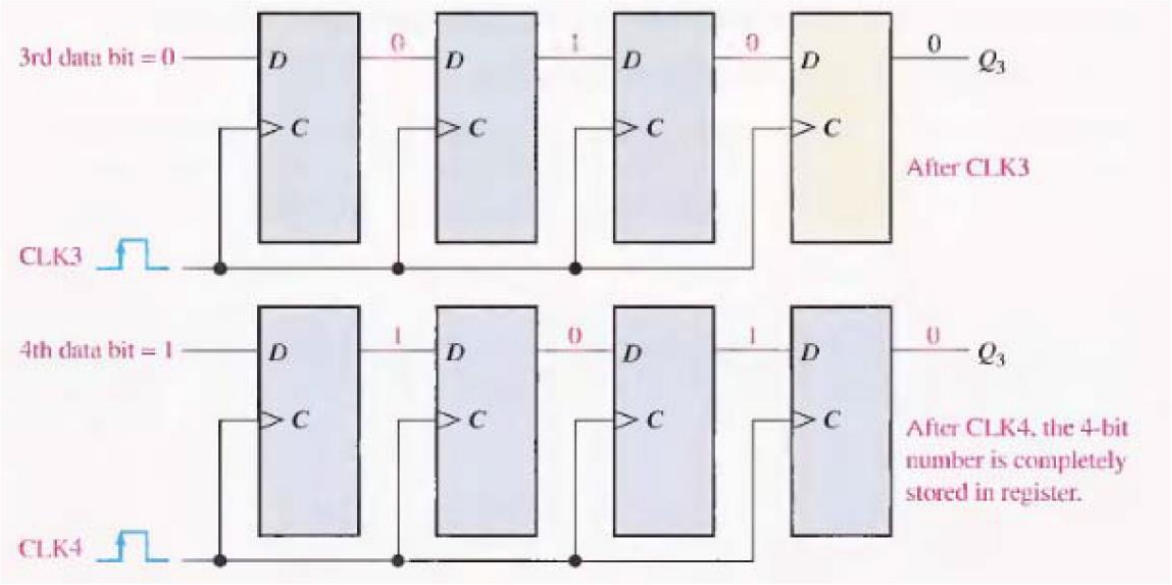


Figure (23) 4-bit shift register

Example: 2

Draw 5-bit shift register and write wave form?

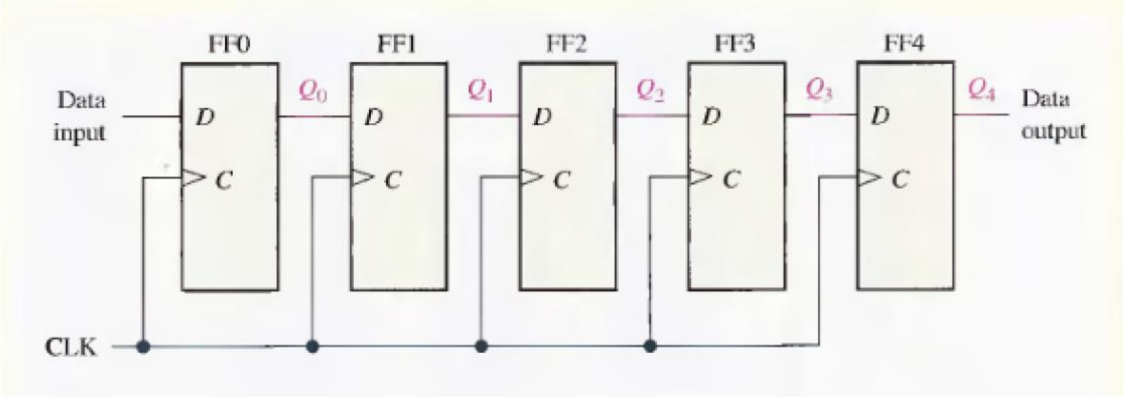


Figure (24) 5-bit shift register

Lectured (11)

Decoders & encoders:

1- Decoder:

A decoders is combinational circuit that converts binary information form the n coded inputs to a maximum of 2_n unique outputs.

That decoders are called n-to-m line decoders where m <=2_n.

The logic diagram of a 3-to-8 line decoder is three data inputs, A0, A1, and A2 are decoded into eight out puts, each out puts representing one of the combinations of the three binary input variables.

This decoder is a binary – to – octal conversion.

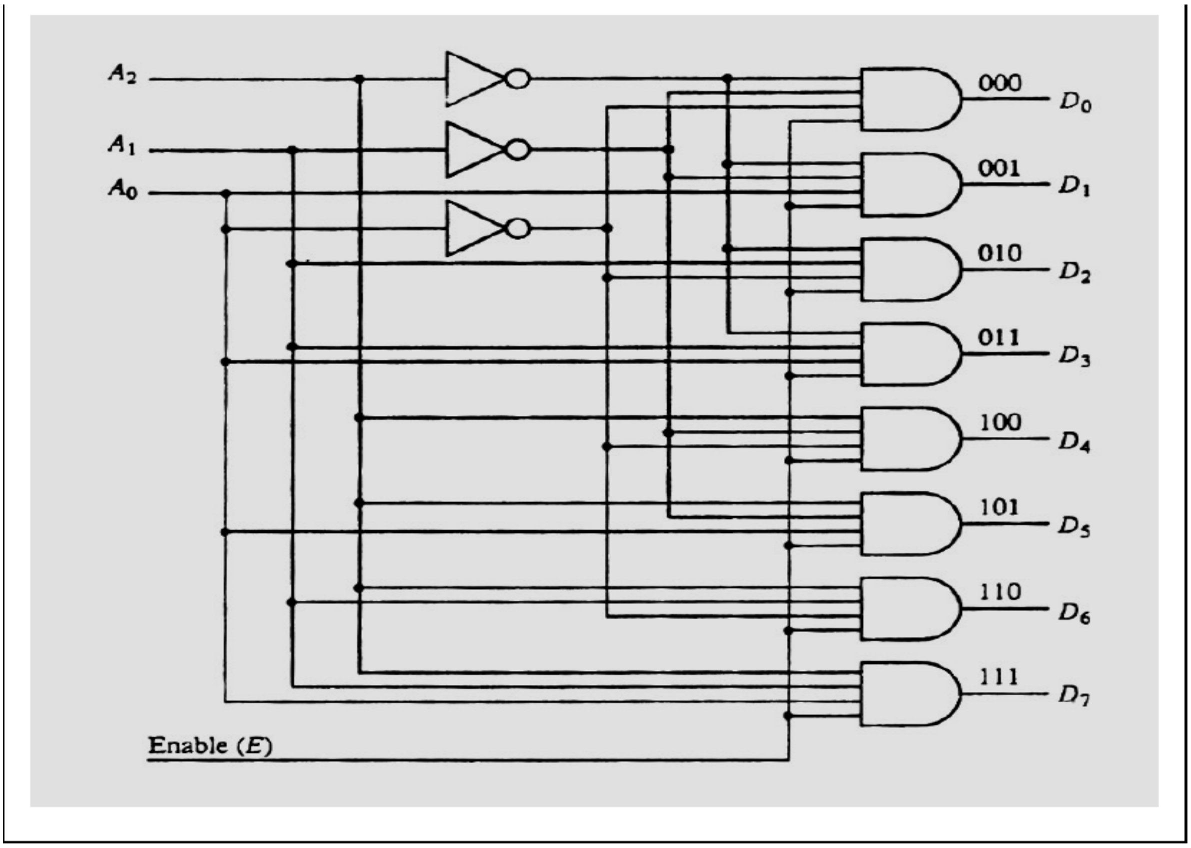


Figure (14-a) 3-to-8 line decoder (Logic Diagram)

Enable	Inputs			Outputs							
E	**A2**	**A1**	**A0**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0		0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Figure (14-b)Truth table for 3-to-8 line decoder

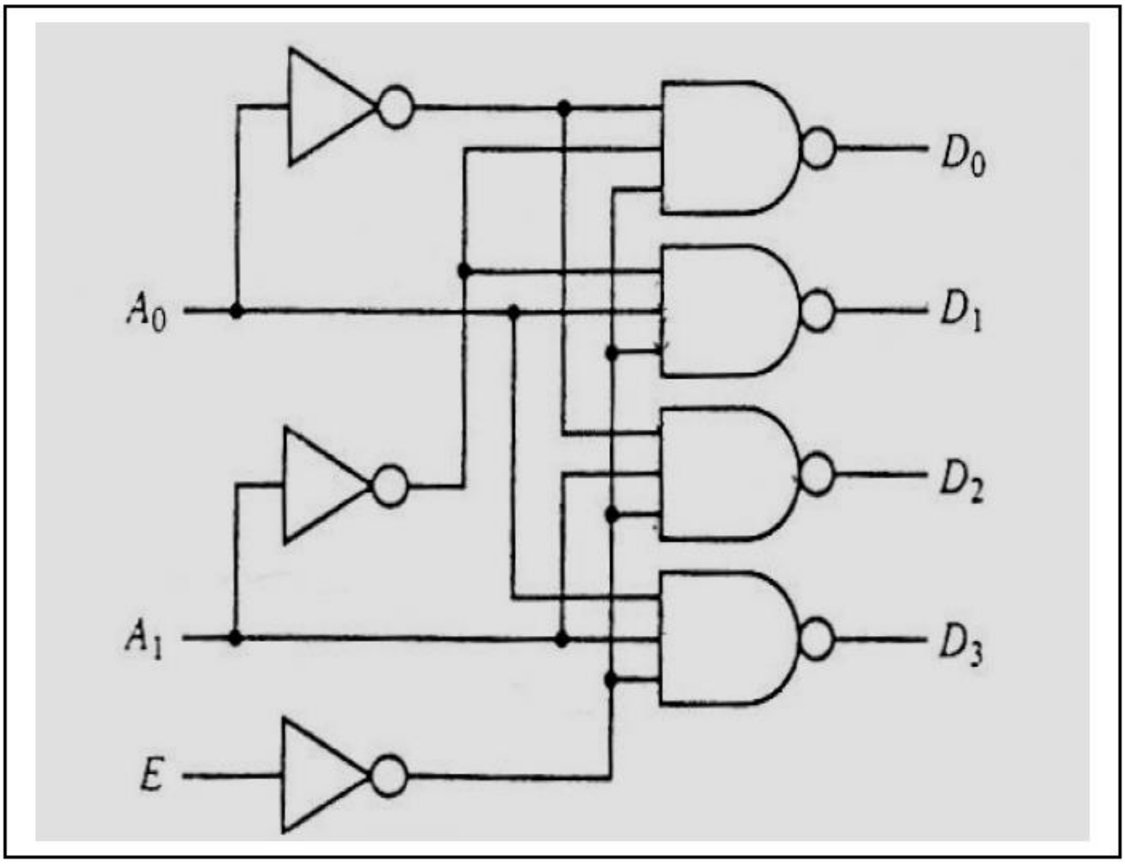


Figure (15-a) 2-to-4 line decoder (Logic Diagram)

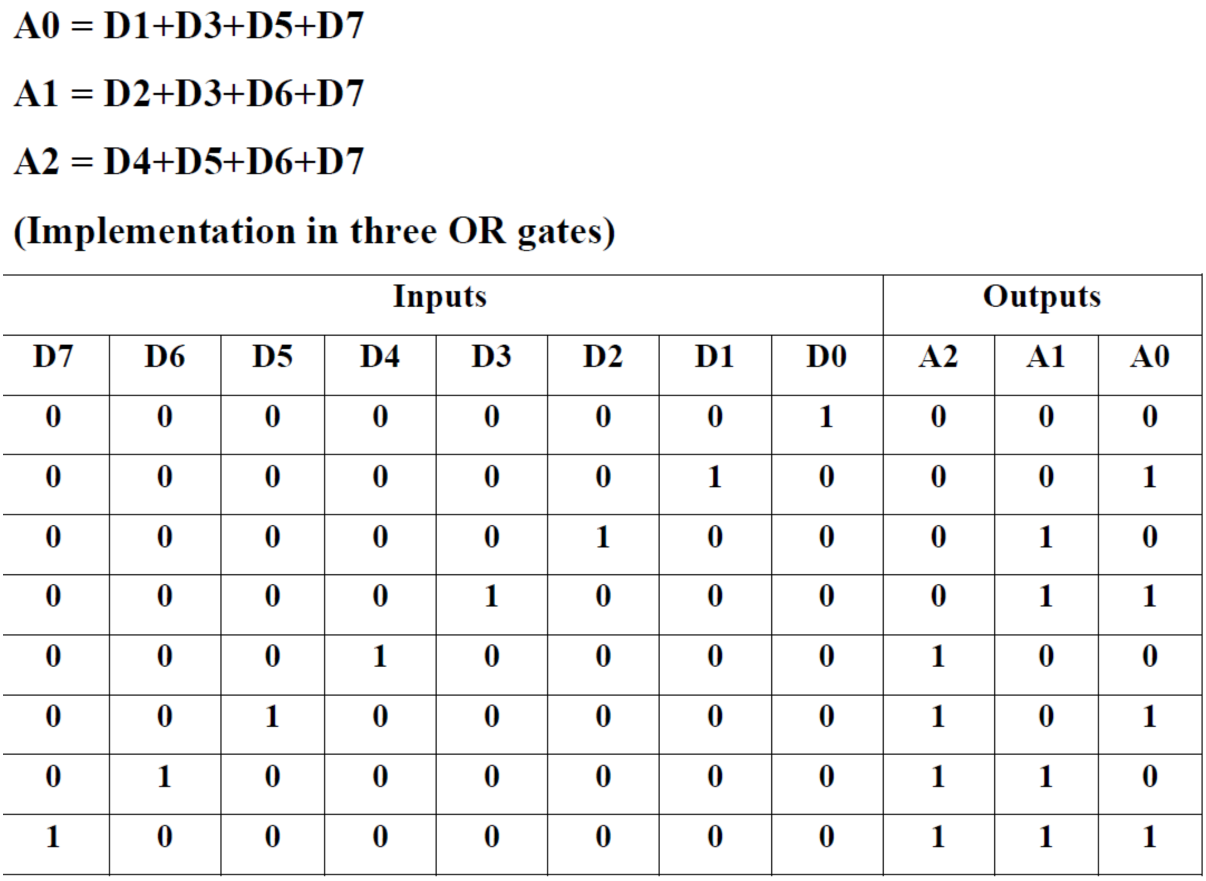
Enable	Inputs		Outputs			
E	A1	A0	D0	D1	D2	D3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
0	X	X	1	1	1	1

Figure (15-b)Truth table for 2-to-4 line decoder

2- Encoder:

An encoder is a digit circuit that performs the inverse operation of a decoder. An encoder has 2_n (or less) input lines and n output lines. An encoder is the octal – to – binary encoder.

It has eight inputs, one for each of the octal digits, and three outputs that generate the corresponding binary number.



A0 = D1+D3+D5+D7

A1 = D2+D3+D6+D7

A2 = D4+D5+D6+D7

(Implementation in three OR gates)

Inputs								Outputs		
D7	D6	D5	D4	D3	D2	D1	D0	A2	A1	A0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Figure (16-a) Truth table for octal – to – binary encoder

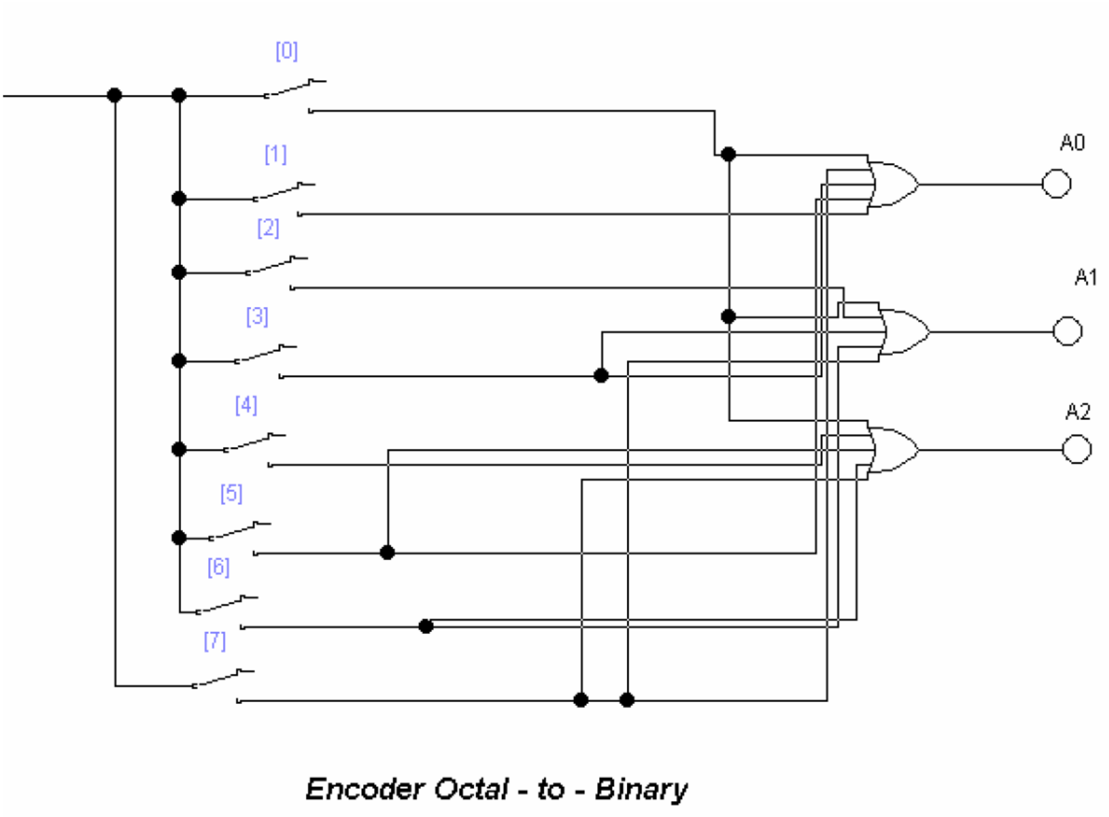


Figure (16-b) 8 – to – 3 lines Encoder (Logic Diagram)

3- Multiplexers:

A multiplexer is a combinational circuit that receiver binary information form one of 2_n input data lines and directs it to a single out put line.

The selection of a particular input data line for the output is determined by a set of selection inputs. A 2_n- to- 1 , A 4-to-1. Multiplexer is called Data Selector.

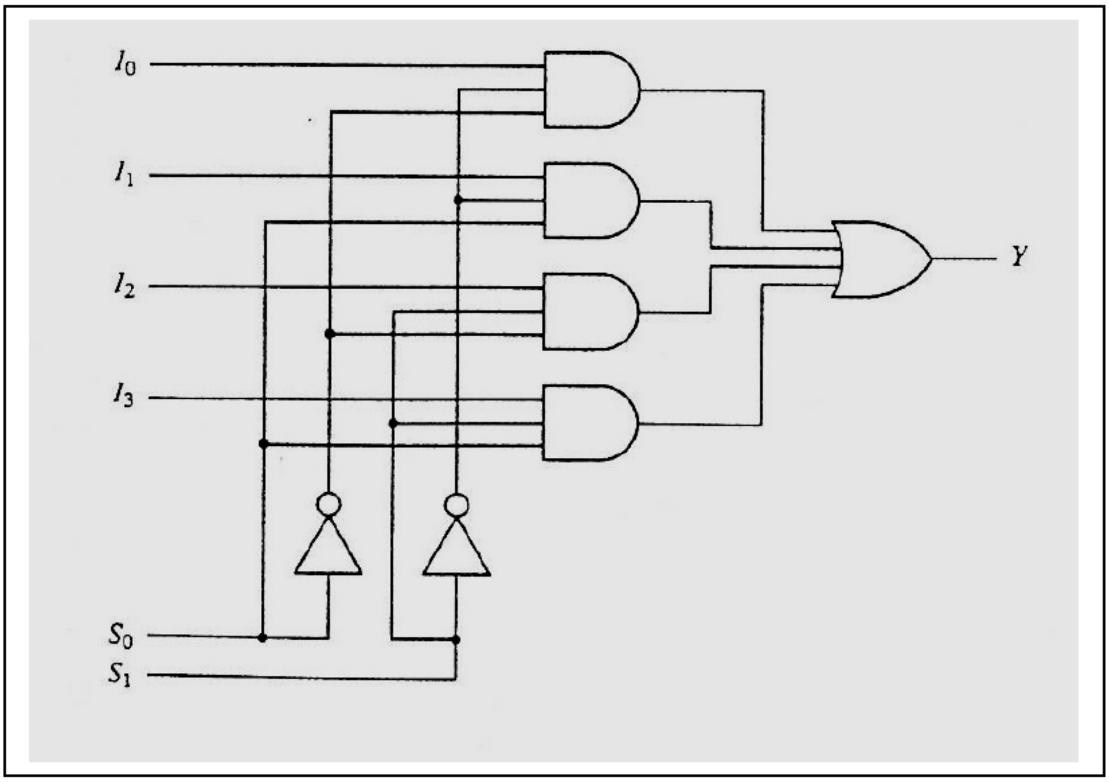
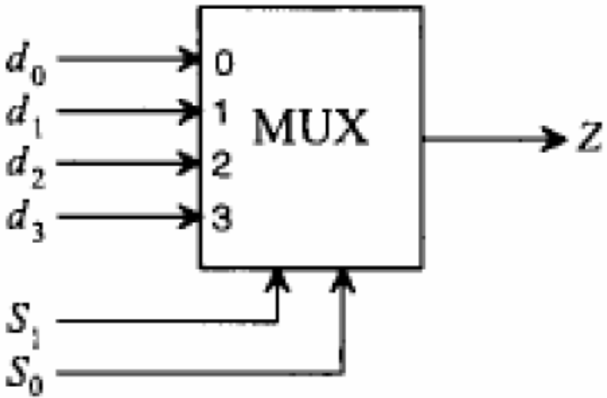


Figure (17-a) 4-to-1 line multiplexer (Logic Diagram)

Inputs		Outputs
S0	S1	Y
0	0	Y1
0	1	Y2
1	0	Y3
1	1	Y4

Figure (17-b) Truth table for 4-to-1 multiplexer



Lectured (12)

4-Demultiplexers:

A demultiplexer (DEMUX) basically reverses the multiplexing function. It takes digital information from one line and distributes it to a given number of output lines. For this reason, the demultiplexer is also known as a data distributor. As you will learn, decoders can also be used as demultiplexers.

A 1 to 4 lines demultiplexer (DEMUX) circuit. The data input line goes to all of the AND gates. The two data select lines enable only one gate at a time, and the data appearing on the data input line will pass through the selected gate to the associated data output line.

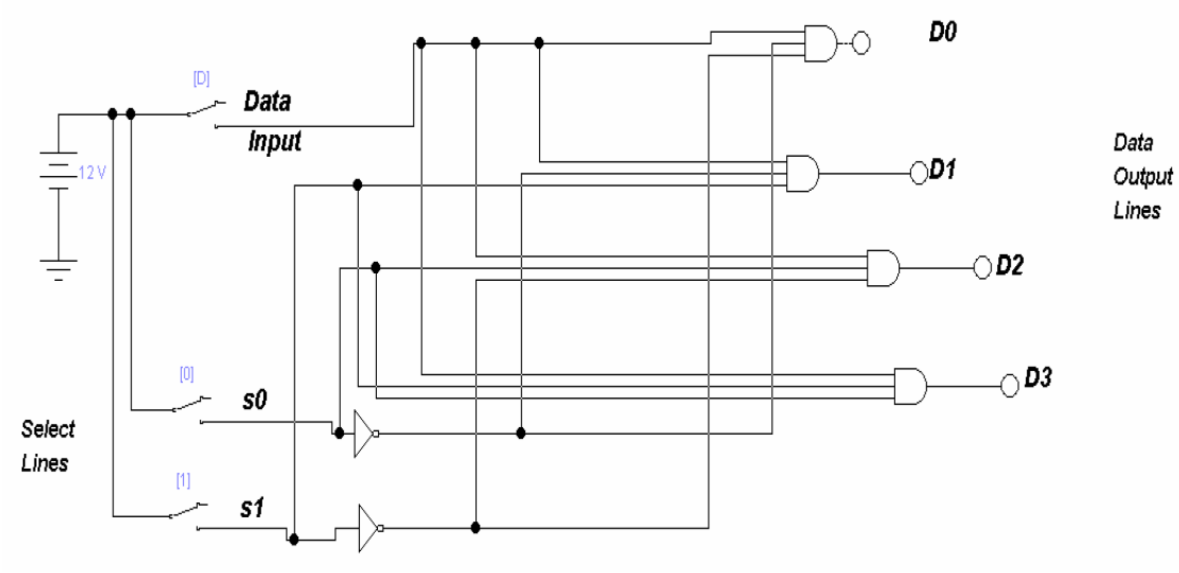


Figure (18-a) Demultiplexer 1 to 4 lines (Logic Diagram)

Inputs			Outputs			
Data	S0	S1	D4	D3	D2	D1
0	0	0	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Figure (18-b) Truth table for 4-to-1 Demultiplexer

Lectured (13)

Binary Counter: The binary counter is consist two types.
1- Asynchronous counter operation.
2- Synchronous counter operation.
1- Asynchronous counter operation:
In figure (25-a, b, c) shows a 2-bit counter connected for asynchronous operation. Notice that the clock (CLK) is applied to the clock input (C) of only the first flip-flop, FF0, which is always the least significant bit (LSB). The second flip-flop, FF1, is triggered by the Q0 output of FF0. FF0 changes state at the positive-going edge of each clock pulse, but FF1 changes only when triggered by a positive-going transition of the Q0 output of FF0. Because of the inherent propagation delay time through a flip-flop, a transition of the input clock pulse (CLK) and transition of the output of FF0 can never occur at exactly the same time. Therefore, the two flip-flops are never simultaneously triggered, so the counter operation is asynchronous.

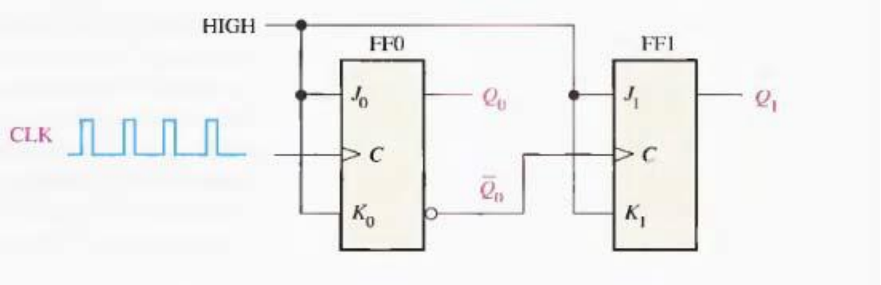


Figure (25-a) 2-Bit Asynchronous Binary Counter

2- Synchronous counter operation:
The term synchronous refers to events that have a fixed time relationship with each other. A synchronous counter is one in which all the flip-flops in the counter are clocked at the same time by a common clock pulse.
A 3-bit synchronous binary counter is shown in figure (26-a) and timing diagram is shown (26-b) you can understand this counter operation by examining its sequence of states as shown in truth table (26-c).

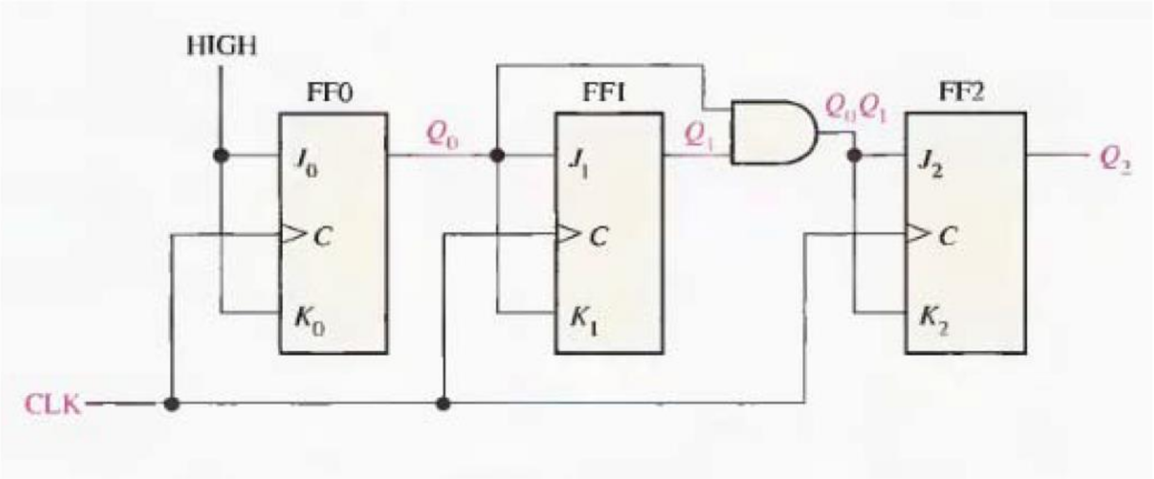


Figure (26-a) 3-Bit Synchronous Binary Counter

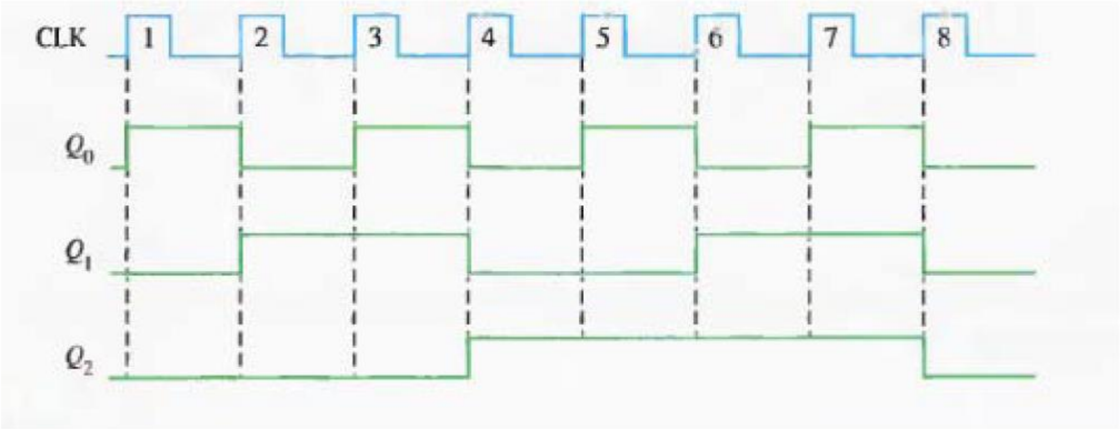
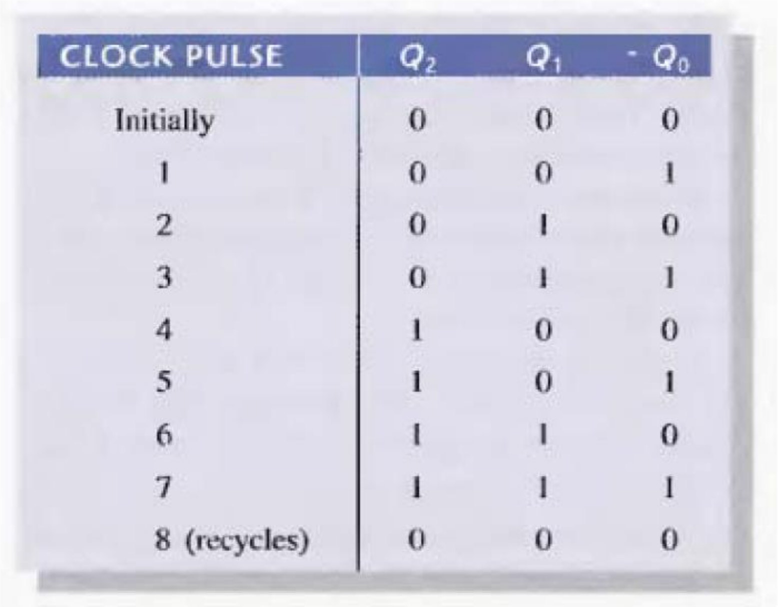


Figure (26-b) Time diagram 3-Bit Synchronous Binary Counter



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

Figure (26-c) truth table for 3-Bit Synchronous Binary Counter