

COURSE MATERIAL

UNIT 1

SEC1207-DIGITAL LOGIC CIRCUITS

SYLLABUS

UNIT I BOOLEAN ALGEBRA AND LOGIC GATES 9 Hrs.

Review of number systems - Binary arithmetic - Binary codes - Boolean algebra and theorems - Boolean functions -Minimization of Boolean functions-Sum of Products(SOP)-Product of Sums(POS)-Simplifications of Boolean functions using Karnaugh map and tabulation methods - Logic gates- NAND and NOR implementation.

TABLE OF TOPICS

S.NO	TOPIC	PAGE NO.
1.1	Review of Number systems	2
1.1.1	Number Systems: Decimal, Binary, Octal, Hexadecimal	2
1.1.2	Conversion from one system to another	3
1.2	Binary Codes	16
1.3	Binary Arithmetic	22
1.4	Boolean Algebra and Theorems	32
1.5	Boolean Functions	36
1.5.1	Minimization of Boolean functions	41
1.5.2	Simplification Using Boolean Functions	47
1.5.2.1	Simplification Using Karnaugh map method	47
1.5.2.2	Simplification Using Tabulation method	67
1.6	Logic gates	74
1.6.1	Universal Gates	83
1.6.2	NAND and NOR implementation	85

1.1 REVIEW OF NUMBER SYSTEMS

Numbers are used to count, communicate, measure etc. The common number systems used in this modern world are decimal and binary system. Decimal system is used by humans and binary system is used in machines. Other human-machine interface systems are octal and hexadecimal.

- Decimal system used by humans has 10 digits ranging from 0 to 9.
- They are 0 1, 2, 3, 4, 5, 6, 7, 8 and 9.
- Since it uses 10 digits, it is named “*decimal*” system, where “*deci*” means “*by 10*”.
- So *Radix* or *Base* of this *Decimal* system is 10.

1.1.1 NUMBER SYSTEMS

The number systems, their base or Radix and the digits used are shown in the below table.

Number Systems	Number of digits (Base OR Radix)	Digits in ascending order															
		0	1														
Binary	2	0	1														
Ternary	3	0	1	2													
Quinary	5	0	1	2	3	4											
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

Note: The systems in BOLD are discussed further and others are for reference only.

Thus radix of binary is 2, that of octal is 8 and that of hexadecimal is 16. The hexadecimal is an exception with 16 digits and uses digits 0-9 (10 digits) and CAPITAL alphabets (A-F) where (A=10, B=11, C=12, D=13, E=14, F=15)

Let us discuss about position and weight of number systems.

Example: Let us consider a Decimal number 8279.312

Decimal Number	8	2	7	9	.	3	1	2
Position	3	2	1	0	↔	-1	-2	-3
Weight = (Radix^{Position})	10 ³	10 ²	10 ¹	10 ⁰		10 ⁻¹	10 ⁻²	10 ⁻³
	1000	100	10	1		0.1	0.01	0.001

As shown in the above table, position increases from floating point to the left and decreases after floating point to the right. Weight of any position is Radix^{position}. For example, weight of position 2 is 10²=100=hundred.

So tables denoting position and weights are shown below for other systems.

Binary Number	1	0	1	1	.	1	0	1
Position	3	2	1	0	↔	-1	-2	-3
Weight = (Radix^{Position})	2 ³	2 ²	2 ¹	2 ⁰		2 ⁻¹	2 ⁻²	2 ⁻³
	8	4	2	1		0.5	0.25	0.125

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

Octal Number	3	6	7	2	.	1	5	4
Position	3	2	1	0	↔	-1	-2	-3
Weight = (Radix^{Position})	8 ³	8 ²	8 ¹	8 ⁰		8 ⁻¹	8 ⁻²	8 ⁻³
	512	64	8	1		0.125	0.0625	0.001953125

Hexadecimal Number	D	6	4	E	.	A	3	9
Position	3	2	1	0	↔	-1	-2	-3
Weight = (Radix^{Position})	16 ³	16 ²	16 ¹	16 ⁰		16 ⁻¹	16 ⁻²	16 ⁻³
	4096	256	16	1		0.5	0.00390625	0.000244140625

Weights and position play important role in conversions from one number system to other.

Note: Each number is called digit in decimal, octal and hexadecimal system but in binary each is called bit.

Solve the questions below:

What is base or radix of a number?

Write the weights of (1.) 1110.1101_b, (2.) 3579.2178_d, (3.) (7543.215)₈ and (4.) 9AD.E347_h

Representation of numbers:

The below table shows two methods of representation of number systems when all are mixed and used in a system

Method 1: Writing the number in braces and using subscript of the number radix at the end of the number as shown in column 2 of the table shown below. (Number)_{radix}

Method 2: Writing the number using alphabets like **b**, **o**, **d** and **h** for binary, octal, decimal and hexadecimal respectively as shown in column 3 of the table shown below. (Number)_{radix}

Writing binary, decimal and hexadecimal is not a problem. But while writing octal, alphabet **o** confuses for number 0, thus method 1 is extensively followed for octal numbers.

NUMBER SYSTEM	Method 1 Representation: USING RADIX AS SUBSCRIPT	Method 2 Representation: USING ALPHABET AT END
Binary	(110101.1101) ₂	110101.1101 b
Octal	(3472.2561) ₈	3472.2561 o (NOT TO BE USED)
Decimal	(4569.2345) ₁₀	4569.2345 d
Hexadecimal	(3A49E.15FC) ₁₆	3A49E.15FC h

Solve the questions below:

How to represent a decimal, a binary, an octal and a hexadecimal number?

1.1.2 NUMBER SYSTEM CONVERSIONS FROM ONE TO ANOTHER:

Three methodologies are discussed here where

1. First one being converting decimal to others,
2. Second being converting others to decimal and

3. The third being Binary to Octal/ Hexadecimal and vice versa

The first one of converting Decimal to others discussed below

1.1.2.1. Decimal to others

For converting decimal number to others, the decimal number is split into integer number before floating point and number after floating number.

For Example: If the Decimal number is 95.215, then the number is split into 95 Integer) and .215 (floating number)

For the integer part use Successive Euclidean Division method and for the fractional part use Successive Euclidean Multiplication method.

Decimal Number	95.215d	
Split into integer and fraction parts	95 (Integer Part)	.215 (Fractional part)
Use Successive Euclidean methods separately for each part	Successive Euclidean Division method	Successive Euclidean Multiplication method

Successive Euclidean Division method: (For Integer Part Only)

	95 (Whole Number Part)																																				
<p>Here for Division, the divisor is R2 (the Radix of the number system to be converted to.) R2 for binary is 2 R2 for octal is 8 R2 for hexadecimal is 16</p>	<table style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Divisor</th> <th>Divident</th> <th>Remainder</th> <th>Direction</th> </tr> </thead> <tbody> <tr> <td>R2</td> <td>Integer</td> <td></td> <td></td> </tr> <tr> <td>R2</td> <td>Quotient1</td> <td>-Remainder1</td> <td style="text-align: center;">↑</td> </tr> <tr> <td>R2</td> <td>Quotient2</td> <td>-Remainder2</td> <td style="text-align: center;">↑</td> </tr> <tr> <td>R2</td> <td>Quotient3</td> <td>-Remainder3</td> <td style="text-align: center;">↑</td> </tr> <tr> <td></td> <td style="text-align: center;">→</td> <td style="text-align: center;">→</td> <td></td> </tr> </tbody> </table> <p style="text-align: center;">Continues till Quotient is less than divisor</p>	Divisor	Divident	Remainder	Direction	R2	Integer			R2	Quotient1	-Remainder1	↑	R2	Quotient2	-Remainder2	↑	R2	Quotient3	-Remainder3	↑		→	→													
Divisor	Divident	Remainder	Direction																																		
R2	Integer																																				
R2	Quotient1	-Remainder1	↑																																		
R2	Quotient2	-Remainder2	↑																																		
R2	Quotient3	-Remainder3	↑																																		
	→	→																																			
<p>The Converted Integer part is</p>	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>Quotient3</td> <td>Remainder3</td> <td>Remainder2</td> <td>Remainder1</td> </tr> </table>	Quotient3	Remainder3	Remainder2	Remainder1																																
Quotient3	Remainder3	Remainder2	Remainder1																																		
<p>Decimal to Binary conversion Suppose it is converted to binary R2 is 2 (Radix)</p>	<table style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Divisor</th> <th>Dividend</th> <th>Remainder</th> <th>Direction</th> </tr> </thead> <tbody> <tr> <td>2</td> <td>95</td> <td></td> <td></td> </tr> <tr> <td>2</td> <td>47</td> <td>- 1</td> <td style="text-align: center;">↑</td> </tr> <tr> <td>2</td> <td>28</td> <td>- 1</td> <td style="text-align: center;">↑</td> </tr> <tr> <td>2</td> <td>14</td> <td>- 0</td> <td style="text-align: center;">↑</td> </tr> <tr> <td>2</td> <td>7</td> <td>- 0</td> <td style="text-align: center;">↑</td> </tr> <tr> <td>2</td> <td>3</td> <td>- 1</td> <td style="text-align: center;">↑</td> </tr> <tr> <td></td> <td>1</td> <td>1</td> <td style="text-align: center;">↑</td> </tr> <tr> <td></td> <td style="text-align: center;">→</td> <td style="text-align: center;">→</td> <td></td> </tr> </tbody> </table>	Divisor	Dividend	Remainder	Direction	2	95			2	47	- 1	↑	2	28	- 1	↑	2	14	- 0	↑	2	7	- 0	↑	2	3	- 1	↑		1	1	↑		→	→	
Divisor	Dividend	Remainder	Direction																																		
2	95																																				
2	47	- 1	↑																																		
2	28	- 1	↑																																		
2	14	- 0	↑																																		
2	7	- 0	↑																																		
2	3	- 1	↑																																		
	1	1	↑																																		
	→	→																																			
<p>The Integer part of converted binary is written in the direction shown</p>	<table style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">→</td> <td style="text-align: center;">→</td> <td style="text-align: center;">→</td> <td style="text-align: center;">→</td> <td style="text-align: center;">→</td> <td style="text-align: center;">→</td> <td style="text-align: center;">→</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> </tr> </table> <p style="text-align: center;">95d=1110011b</p>	→	→	→	→	→	→	→	1	1	1	0	0	1	1																						
→	→	→	→	→	→	→																															
1	1	1	0	0	1	1																															

Successive Euclidean Multiplication method: (For Fractional part only)

	.215 (Fractional part)
	Successive Euclidean Multiplication method

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

<p>Here for multiplication, the multiplicand is R2 (the Radix of the number system to be converted to.) R2 for binary is 2 R2 for octal is 8 R2 for hexadecimal is 16</p>	<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th style="width: 30%;">Multiplication of fraction only</th> <th style="width: 20%;">product</th> <th style="width: 15%;">Integer part</th> <th style="width: 35%;">Direction</th> </tr> </thead> <tbody> <tr> <td>Fractional part x R2=</td> <td>Integer1.Fraction1</td> <td>Integer1</td> <td rowspan="5" style="text-align: center; vertical-align: middle;">↓ ↓ ↓ ↓</td> </tr> <tr> <td>Fraction1 x R2=</td> <td>Integer2.Fraction1</td> <td>Integer2</td> </tr> <tr> <td>Fraction2 x R2=</td> <td>Integer3.Fraction1</td> <td>Integer3</td> </tr> <tr> <td>Fraction3 x R2=</td> <td>Integer4.Fraction1</td> <td>Integer4</td> </tr> <tr> <td>Fraction4 x R2=</td> <td>Integer5.Fraction1</td> <td>Integer5</td> </tr> </tbody> </table>	Multiplication of fraction only	product	Integer part	Direction	Fractional part x R2=	Integer1.Fraction1	Integer1	↓ ↓ ↓ ↓	Fraction1 x R2=	Integer2.Fraction1	Integer2	Fraction2 x R2=	Integer3.Fraction1	Integer3	Fraction3 x R2=	Integer4.Fraction1	Integer4	Fraction4 x R2=	Integer5.Fraction1	Integer5
Multiplication of fraction only	product	Integer part	Direction																		
Fractional part x R2=	Integer1.Fraction1	Integer1	↓ ↓ ↓ ↓																		
Fraction1 x R2=	Integer2.Fraction1	Integer2																			
Fraction2 x R2=	Integer3.Fraction1	Integer3																			
Fraction3 x R2=	Integer4.Fraction1	Integer4																			
Fraction4 x R2=	Integer5.Fraction1	Integer5																			
<p>The fraction part of the converted number is written in the following format with floating point</p>	<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr> <td colspan="6" style="padding: 5px;">→→→→→</td> </tr> <tr> <td style="width: 5%;">.</td> <td style="width: 15%;">Integer1</td> <td style="width: 15%;">Integer2</td> <td style="width: 15%;">Integer3</td> <td style="width: 15%;">Integer4</td> <td style="width: 15%;">Integer5</td> </tr> </table>	→→→→→						.	Integer1	Integer2	Integer3	Integer4	Integer5								
→→→→→																					
.	Integer1	Integer2	Integer3	Integer4	Integer5																
<p>Decimal to Binary conversion Suppose it is converted to binary R2 is 2 (Radix)</p>	<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th style="width: 30%;">Multiplication of fraction only</th> <th style="width: 15%;">product</th> <th style="width: 15%;">Integer part</th> <th style="width: 40%;">Direction</th> </tr> </thead> <tbody> <tr> <td>0.215 x 2 =</td> <td>0.43</td> <td>0</td> <td rowspan="5" style="text-align: center; vertical-align: middle;">↓ ↓ ↓ ↓</td> </tr> <tr> <td>0.43 x 2 =</td> <td>0.86</td> <td>0</td> </tr> <tr> <td>0.86 x 2 =</td> <td>1.72</td> <td>1</td> </tr> <tr> <td>0.72 x 2 =</td> <td>1.44</td> <td>1</td> </tr> <tr> <td>0.44 x 2 =</td> <td>0.88</td> <td>0</td> </tr> </tbody> </table> <p style="text-align: center; margin-top: 10px;">The number can be multiplied till it ends but if it is a recurring number, it can stop at any point</p>	Multiplication of fraction only	product	Integer part	Direction	0.215 x 2 =	0.43	0	↓ ↓ ↓ ↓	0.43 x 2 =	0.86	0	0.86 x 2 =	1.72	1	0.72 x 2 =	1.44	1	0.44 x 2 =	0.88	0
Multiplication of fraction only	product	Integer part	Direction																		
0.215 x 2 =	0.43	0	↓ ↓ ↓ ↓																		
0.43 x 2 =	0.86	0																			
0.86 x 2 =	1.72	1																			
0.72 x 2 =	1.44	1																			
0.44 x 2 =	0.88	0																			
<p>The fraction part of converted binary is written in the direction shown</p>	<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr> <td colspan="6" style="padding: 5px;">→→→→→</td> </tr> <tr> <td style="width: 5%;">.</td> <td style="width: 15%;">0</td> <td style="width: 15%;">0</td> <td style="width: 15%;">1</td> <td style="width: 15%;">1</td> <td style="width: 15%;">0</td> </tr> </table> <p style="text-align: center; margin-top: 10px;">0.215 d=0.0011b</p> <p style="text-align: center; margin-top: 5px;">The answer is absolute answer since it has been stopped at 5th multiplication and the number is recurring. This is a rounded off to 5 answer.</p>	→→→→→						.	0	0	1	1	0								
→→→→→																					
.	0	0	1	1	0																
<p>The integer part and fractional part shall be joint together as single number</p>	<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr> <td style="padding: 5px;">Integer part.Fractional part</td> </tr> <tr> <td style="padding: 5px;">1110011.0011b</td> </tr> </table>	Integer part.Fractional part	1110011.0011b																		
Integer part.Fractional part																					
1110011.0011b																					

Thus binary equivalent to 95.215d=1110011.0011b *(rounded to 4 fractional bits)

a) Decimal to Binary conversion:

	Convert 95.215d to binary Integer part =95 Fractional part = .215																									
<p>Decimal to Binary conversion Suppose Integer part is converted to binary divided by 2 (Binary Radix)</p>	<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 15%;">Divisor</th> <th style="width: 20%;">Divident</th> <th style="width: 10%;"></th> <th style="width: 15%;">Remainder</th> <th style="width: 40%;">Direction</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">2</td> <td style="border-top: 1px solid black; border-bottom: 1px solid black; text-align: center;">95</td> <td></td> <td></td> <td></td> </tr> <tr> <td style="text-align: center;">2</td> <td style="border-bottom: 1px solid black; text-align: center;">47</td> <td style="text-align: center;">-</td> <td style="text-align: center;">1</td> <td style="text-align: center;">↑</td> </tr> <tr> <td style="text-align: center;">2</td> <td style="border-bottom: 1px solid black; text-align: center;">28</td> <td style="text-align: center;">-</td> <td style="text-align: center;">1</td> <td style="text-align: center;">↑</td> </tr> <tr> <td style="text-align: center;">2</td> <td style="border-bottom: 1px solid black; text-align: center;">14</td> <td style="text-align: center;">-</td> <td style="text-align: center;">0</td> <td style="text-align: center;">↑</td> </tr> </tbody> </table>	Divisor	Divident		Remainder	Direction	2	95				2	47	-	1	↑	2	28	-	1	↑	2	14	-	0	↑
Divisor	Divident		Remainder	Direction																						
2	95																									
2	47	-	1	↑																						
2	28	-	1	↑																						
2	14	-	0	↑																						

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

Successive Euclidean Division Method	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr><td style="padding: 0 10px;">2</td><td style="border: 1px solid black; padding: 2px 10px;">7</td><td style="padding: 0 10px;">-</td><td style="padding: 0 10px;">0</td><td style="padding: 0 10px;">↑</td></tr> <tr><td style="padding: 0 10px;">2</td><td style="border: 1px solid black; padding: 2px 10px;">3</td><td style="padding: 0 10px;">-</td><td style="padding: 0 10px;">1</td><td style="padding: 0 10px;">↑</td></tr> <tr><td></td><td style="padding: 2px 10px;">1</td><td></td><td style="padding: 0 10px;">1</td><td style="padding: 0 10px;">↑</td></tr> <tr><td></td><td style="padding: 2px 10px;">→</td><td></td><td style="padding: 0 10px;">→</td><td></td></tr> </table>	2	7	-	0	↑	2	3	-	1	↑		1		1	↑		→		→	
2	7	-	0	↑																	
2	3	-	1	↑																	
	1		1	↑																	
	→		→																		
The Integer part of converted binary is written in the direction shown	<table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="padding: 2px 10px;">1</td><td style="padding: 2px 10px;">1</td><td style="padding: 2px 10px;">1</td><td style="padding: 2px 10px;">0</td><td style="padding: 2px 10px;">0</td><td style="padding: 2px 10px;">1</td><td style="padding: 2px 10px;">1</td> </tr> </table> <p style="text-align: center;">95d=1110011b</p>	1	1	1	0	0	1	1													
1	1	1	0	0	1	1															
Decimal to Binary conversion Suppose fractional part is converted to binary multiplied by 2 (Binary Radix) Successive Euclidean Multiplication Method	<table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <thead> <tr> <th style="padding: 5px;">Multiplication of fraction only</th> <th style="padding: 5px;">product</th> <th style="padding: 5px;">Integer part</th> <th style="padding: 5px;">Direction</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">0.215 x 2 =</td> <td style="padding: 5px;">0.43</td> <td style="padding: 5px;">0</td> <td rowspan="5" style="text-align: center; vertical-align: middle;">↓ ↓ ↓ ↓ ↓</td> </tr> <tr> <td style="padding: 5px;">0.43 x 2 =</td> <td style="padding: 5px;">0.86</td> <td style="padding: 5px;">0</td> </tr> <tr> <td style="padding: 5px;">0.86 x 2 =</td> <td style="padding: 5px;">1.72</td> <td style="padding: 5px;">1</td> </tr> <tr> <td style="padding: 5px;">0.72 x 2 =</td> <td style="padding: 5px;">1.44</td> <td style="padding: 5px;">1</td> </tr> <tr> <td style="padding: 5px;">0.44 x 2 =</td> <td style="padding: 5px;">0.88</td> <td style="padding: 5px;">0</td> </tr> </tbody> </table> <p style="text-align: center;">The number can be multiplied till it ends but if it is a recurring number, it can be stopped at any point</p>	Multiplication of fraction only	product	Integer part	Direction	0.215 x 2 =	0.43	0	↓ ↓ ↓ ↓ ↓	0.43 x 2 =	0.86	0	0.86 x 2 =	1.72	1	0.72 x 2 =	1.44	1	0.44 x 2 =	0.88	0
Multiplication of fraction only	product	Integer part	Direction																		
0.215 x 2 =	0.43	0	↓ ↓ ↓ ↓ ↓																		
0.43 x 2 =	0.86	0																			
0.86 x 2 =	1.72	1																			
0.72 x 2 =	1.44	1																			
0.44 x 2 =	0.88	0																			
The fraction part of converted binary is written in the direction shown	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr><td colspan="6" style="text-align: center;">→→→→→</td></tr> <tr> <td style="padding: 2px 10px;">.</td><td style="padding: 2px 10px;">0</td><td style="padding: 2px 10px;">0</td><td style="padding: 2px 10px;">1</td><td style="padding: 2px 10px;">1</td><td style="padding: 2px 10px;">0</td> </tr> <tr> <td></td><td></td><td></td><td></td><td></td><td style="padding: 2px 10px;">Discarded because 0 end of fraction is valueless</td> </tr> </table> <p style="text-align: center;">0.215 d=0.0011b</p> <p style="text-align: center;">The answer is absolute answer, Since it has been stopped at 5th multiplication and the number is recurring. This answer is rounded off to 5 fractional points.</p>	→→→→→						.	0	0	1	1	0						Discarded because 0 end of fraction is valueless		
→→→→→																					
.	0	0	1	1	0																
					Discarded because 0 end of fraction is valueless																
The integer part and fractional part shall be joint together as single number	<table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="padding: 5px;">Full number = Integer part.fractional part</td> </tr> <tr> <td style="padding: 5px;">95.215d=1110011.0011b</td> </tr> </table>	Full number = Integer part.fractional part	95.215d=1110011.0011b																		
Full number = Integer part.fractional part																					
95.215d=1110011.0011b																					

Thus binary equivalent to 95.215d= 1110011.0011b *(rounded to 4 fractional bits)

b) Decimal to Octal conversion:

	Convert 95.215d to octal Integer part =95 Fractional part = .215																			
Decimal to Octal conversion Suppose Integer part is converted to Octal divided by 8 (Octal)	<table style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <thead> <tr> <th style="padding: 5px;">Divisor</th> <th style="padding: 5px;">Divident</th> <th style="padding: 5px;">Remainder</th> <th style="padding: 5px;">Direction</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">8</td> <td style="border: 1px solid black; padding: 2px 10px;">95</td> <td style="padding: 5px;">-</td> <td style="padding: 5px;">7</td> <td style="padding: 5px;">↑</td> </tr> <tr> <td style="padding: 5px;">8</td> <td style="border: 1px solid black; padding: 2px 10px;">11</td> <td style="padding: 5px;">-</td> <td style="padding: 5px;">3</td> <td style="padding: 5px;">↑</td> </tr> <tr> <td></td> <td style="padding: 2px 10px;">1</td> <td></td> <td></td> <td style="padding: 5px;">↑</td> </tr> </tbody> </table>	Divisor	Divident	Remainder	Direction	8	95	-	7	↑	8	11	-	3	↑		1			↑
Divisor	Divident	Remainder	Direction																	
8	95	-	7	↑																
8	11	-	3	↑																
	1			↑																

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

Radix) Successive Euclidean Division Method	→ →																				
The Integer part of converted Octal is written in the direction shown	<table border="1" style="margin: auto; border-collapse: collapse;"> <tr><td style="text-align: center;">→</td><td style="text-align: center;">→</td><td style="text-align: center;">→</td></tr> <tr><td style="text-align: center;">1</td><td style="text-align: center;">3</td><td style="text-align: center;">7</td></tr> </table> <p style="text-align: center; margin-top: 10px;">95d=(137)₈</p>	→	→	→	1	3	7														
→	→	→																			
1	3	7																			
Decimal to Octal conversion Suppose fractional part is converted to Octal multiplied by 8 (Octal Radix) Successive Euclidean Multiplication Method	<table border="1" style="margin: auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th style="padding: 5px;">Multiplication of fraction only</th> <th style="padding: 5px;">product</th> <th style="padding: 5px;">Integer part</th> <th style="padding: 5px;">Direction</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">0.215 x 8 =</td> <td style="padding: 5px;">1.72</td> <td style="padding: 5px;">1</td> <td rowspan="5" style="padding: 5px; vertical-align: middle;">↓ ↓ ↓ ↓ ↓</td> </tr> <tr> <td style="padding: 5px;">0.72 x 8 =</td> <td style="padding: 5px;">5.76</td> <td style="padding: 5px;">5</td> </tr> <tr> <td style="padding: 5px;">0.76 x 8 =</td> <td style="padding: 5px;">6.08</td> <td style="padding: 5px;">6</td> </tr> <tr> <td style="padding: 5px;">0.08 x 8 =</td> <td style="padding: 5px;">0.64</td> <td style="padding: 5px;">0</td> </tr> <tr> <td style="padding: 5px;">0.64 x 8 =</td> <td style="padding: 5px;">5.12</td> <td style="padding: 5px;">5</td> </tr> </tbody> </table> <p style="text-align: center; margin-top: 10px;">The number can be multiplied till it ends but if it is a recurring number, it can be stopped at any point</p>	Multiplication of fraction only	product	Integer part	Direction	0.215 x 8 =	1.72	1	↓ ↓ ↓ ↓ ↓	0.72 x 8 =	5.76	5	0.76 x 8 =	6.08	6	0.08 x 8 =	0.64	0	0.64 x 8 =	5.12	5
Multiplication of fraction only	product	Integer part	Direction																		
0.215 x 8 =	1.72	1	↓ ↓ ↓ ↓ ↓																		
0.72 x 8 =	5.76	5																			
0.76 x 8 =	6.08	6																			
0.08 x 8 =	0.64	0																			
0.64 x 8 =	5.12	5																			
The fraction part of converted Octal is written in the direction shown	<table border="1" style="margin: auto; border-collapse: collapse; text-align: center;"> <tr><td colspan="6" style="padding: 5px;">→→→→→</td></tr> <tr> <td style="padding: 5px;">.</td> <td style="padding: 5px;">1</td> <td style="padding: 5px;">5</td> <td style="padding: 5px;">6</td> <td style="padding: 5px;">0</td> <td style="padding: 5px;">5</td> </tr> </table> <p style="text-align: center; margin-top: 10px;">0.215 d=(0.15605)₈</p> <p style="text-align: center;">The answer is absolute answer, Since it has been stopped at 5th multiplication and the number is recurring. This answer is rounded off to 5 fractional points.</p>	→→→→→						.	1	5	6	0	5								
→→→→→																					
.	1	5	6	0	5																
The integer part and fractional part shall be joint together as single number	<table border="1" style="margin: auto; border-collapse: collapse; text-align: center;"> <tr> <td style="padding: 5px;">Full number = Integer part.fractional part</td> </tr> <tr> <td style="padding: 5px;">95.215d=(137.15605)₈</td> </tr> </table>	Full number = Integer part.fractional part	95.215d=(137.15605)₈																		
Full number = Integer part.fractional part																					
95.215d=(137.15605)₈																					

Thus Octal equivalent to 95.215d= (137.15605)₈ *(rounded to 5 fractional digits)

c) Decimal to Hexadecimal conversion:

	<p>Convert 95.215d to Hexadecimal Integer part =95 Fractional part = .215</p>																				
<p>Decimal to Hexadecimal conversion Suppose Integer part is converted to Hexadecimal divided by 16 (Hexadecimal Radix) Successive Euclidean Division Method</p>	<table style="margin: auto;"> <tr> <td style="text-align: center;">Divisor 16</td> <td style="text-align: center;">Dividend 95</td> <td style="text-align: center;">-</td> <td style="text-align: center;">Remainder 15 (F)</td> <td style="text-align: center;">Direction ↑</td> </tr> <tr> <td></td> <td style="text-align: center;">5</td> <td></td> <td></td> <td style="text-align: center;">↑</td> </tr> <tr> <td></td> <td style="text-align: center;">→</td> <td></td> <td style="text-align: center;">→</td> <td></td> </tr> </table> <p>15 cannot be written as double digit in hexadecimal because 15d=Fh in hexadecimal (given in bracket) 10d=Ah, 11d=Bh, 12d=Ch, 13d=Dh, 14d=Eh, 15d=Fh</p>	Divisor 16	Dividend 95	-	Remainder 15 (F)	Direction ↑		5			↑		→		→						
Divisor 16	Dividend 95	-	Remainder 15 (F)	Direction ↑																	
	5			↑																	
	→		→																		
<p>The Integer part of converted Hexadecimal is written in the direction shown</p>	<table style="margin: auto; border-collapse: collapse;"> <tr> <td style="border: 1px solid black; padding: 2px;">→</td> <td style="border: 1px solid black; padding: 2px;">→</td> </tr> <tr> <td style="border: 1px solid black; padding: 2px;">5</td> <td style="border: 1px solid black; padding: 2px;">F</td> </tr> </table> <p>95d=5Fh</p>	→	→	5	F																
→	→																				
5	F																				
<p>Decimal to Hexadecimal conversion Suppose fractional part is converted to Hexadecimal multiplied by 16 (Hexadecimal Radix) Successive Euclidean Multiplication Method</p>	<table border="1" style="margin: auto; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Multiplication of fraction only</th> <th style="text-align: center;">product</th> <th style="text-align: center;">Integer part</th> <th style="text-align: center;">Direction</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">0.215 x 16 =</td> <td style="text-align: center;">3.44</td> <td style="text-align: center;">3</td> <td rowspan="5" style="text-align: center;">↓ ↓ ↓ ↓</td> </tr> <tr> <td style="text-align: center;">0.44x 16 =</td> <td style="text-align: center;">7.04</td> <td style="text-align: center;">7</td> </tr> <tr> <td style="text-align: center;">0.04 x 16 =</td> <td style="text-align: center;">0.64</td> <td style="text-align: center;">0</td> </tr> <tr> <td style="text-align: center;">0.64 x 16 =</td> <td style="text-align: center;">10.24</td> <td style="text-align: center;">10(A)</td> </tr> <tr> <td style="text-align: center;">0.24 x 16 =</td> <td style="text-align: center;">3.84</td> <td style="text-align: center;">3</td> </tr> </tbody> </table> <p>The number can be multiplied till it ends but if it is a recurring number, it can be stopped at any point</p>	Multiplication of fraction only	product	Integer part	Direction	0.215 x 16 =	3.44	3	↓ ↓ ↓ ↓	0.44x 16 =	7.04	7	0.04 x 16 =	0.64	0	0.64 x 16 =	10.24	10(A)	0.24 x 16 =	3.84	3
Multiplication of fraction only	product	Integer part	Direction																		
0.215 x 16 =	3.44	3	↓ ↓ ↓ ↓																		
0.44x 16 =	7.04	7																			
0.04 x 16 =	0.64	0																			
0.64 x 16 =	10.24	10(A)																			
0.24 x 16 =	3.84	3																			
<p>The fraction part of converted Hexadecimal is written in the direction shown</p>	<table style="margin: auto; border-collapse: collapse;"> <tr> <td colspan="6" style="text-align: center;">→→→→→</td> </tr> <tr> <td style="border: 1px solid black; padding: 2px;">.</td> <td style="border: 1px solid black; padding: 2px;">3</td> <td style="border: 1px solid black; padding: 2px;">7</td> <td style="border: 1px solid black; padding: 2px;">0</td> <td style="border: 1px solid black; padding: 2px;">A</td> <td style="border: 1px solid black; padding: 2px;">3</td> </tr> </table> <p>0.215 d= 0.370A3h The answer is absolute answer, Since it has been stopped at 5th multiplication and the number is recurring. This answer is rounded off to 5 fractional points.</p>	→→→→→						.	3	7	0	A	3								
→→→→→																					
.	3	7	0	A	3																
<p>The integer part and fractional part shall be joint together as single number</p>	<table border="1" style="margin: auto; border-collapse: collapse;"> <tr> <td style="text-align: center;">Full number = Integer part.fractional part</td> </tr> <tr> <td style="text-align: center;">95.215d=5F.370A3h</td> </tr> </table>	Full number = Integer part.fractional part	95.215d=5F.370A3h																		
Full number = Integer part.fractional part																					
95.215d=5F.370A3h																					

Thus Hexadecimal equivalent to 95.215d= 5F.370A3h *(rounded to 5 fractional digits)

Thus the conversion from decimal to binary/octal/hexadecimal has been discussed.

1.1.2.2. Others to decimal

For converting other system of numbers like binary/octal/hexadecimal to decimal, position and weight of the number is considered. The method used is **position with weight multiplication**.

For Example: Any other number AA.AAA is considered.

Let radix of the number is Y.

Number	A	A	.	A	A	A
Position	1	0	.	-1	-2	-3
Weight	Y^1	Y^0	.	Y^{-1}	Y^{-2}	Y^{-3}
Digit/Bit x Weight	$A \times Y^1 = E^1$	$A \times Y^0 = E^0$.	$A \times Y^{-1} = E^{-1}$	$A \times Y^{-2} = E^{-2}$	$A \times Y^{-3} = E^{-3}$
Converted to Decimal Number	$E^1 + E^0$.	$E^{-1} + E^{-2} + E^{-3}$		
	Sum Integer part separately			.	Sum Fractional part separately	
	Integer part. Fractional part					

As shown in above table, each digit/bit is multiplied by its weight, (Where weight of the number is radix of the number POWER to its position from the fractional point= $\text{Radix}^{\text{position}}$). Then summed integer part and summed fractional part are joined together as shown in the above table. This is the converted decimal number.

Binary to Decimal Number:

Example: Converting binary number (11.011b) to Decimal

Number	1	1	.	0	1	1
Position	1	0	.	-1	-2	-3
Weight	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}
	2	1		0.5	0.25	0.125
Bit x Weight	1×2^1	1×2^0	.	0×2^{-1}	1×2^{-2}	1×2^{-3}
	1×2	1×1		0×0.5	1×0.25	1×0.125
	2	1	.	0	0.25	0.125
Converted to Decimal Number	$2+1=3$.	$0+0.25+0.125=0.375$	
	3.375d					

Hence **Binary number (11.011b) = 3.375d** (Decimal)

Octal to Decimal Number:

Example: Converting Octal number [(73.245)₈] to Decimal

Number	7	3	.	2	4	5
Position	1	0	.	-1	-2	-3
Weight	8^1	8^0	.	8^{-1}	8^{-2}	8^{-3}
	8	1		0.125	0.015625	0.001953125
Digit x Weight	7×8^1	3×8^0	.	2×8^{-1}	4×8^{-2}	5×8^{-3}
	7×8	3×1	.	2×0.125	4×0.015625	5×0.001953125
	56	3	.	0.25	0.0625	0.009765625
Converted to Decimal Number	$56+3=59$.	$0.25+0.0625+0.009765625=0.322265625$		
	59.322265625d					
	59.32227d (Rounded to 5 floating points)					

Hence **Octal number (73.245)₈ = 59.322265625d** (Decimal)

Hexadecimal to Decimal Number:

Example: Converting Hexadecimal number (5F.37Bh) to Decimal

Number	5	F	.	3	7	B
Position	1	0	.	-1	-2	-3
Weight	16¹	16⁰	.	16⁻¹	16⁻²	16⁻³
	16	1	.	0.0625	0.00390625	0.000244140625
Digit x Weight	5 x 16 ¹	F(15) x 16 ⁰	.	3 x 16 ⁻¹	7 x 16 ⁻²	B(11) x 16 ⁻³
	5 x 16	15 x 1	.	3 x 0.0625	7 x 0.00390625	11 x 0.00244140625
	80	15	.	0.1875	0.02734375	0.002685546875
Converted to Decimal Number	80+15=95		.	0.1875+0.02734375+0.002685546875		
	=0.217529296875					
	95.217529296875d					
95.21753d (Rounded to 5 floating points)						

Hence **Hexadecimal number 5F.37Bh =95.217529296875d** (Decimal)

Thus the conversion from binary/octal/hexadecimal to decimal has been discussed.

1.1.2.3 Binary to Octal/Hexadecimal and Vice-versa

(a) Binary to Octal Number

If any binary number is to be converted to octal, group of three (3) bits are formed with both sides of the fractional point. Fractional point is the reference.

Binary Number	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀	.	b ₋₁	b ₋₂	b ₋₃	b ₋₄	b ₋₅
Direction of Grouping	←	←	←	←	←	←	←	←	.	→	→	→	→	→
Group In 3 bits	b ₇ b ₆		b ₅ b ₄ b ₃			b ₂ b ₁ b ₀			.	b ₋₁ b ₋₂ b ₋₃			b ₋₄ b ₋₅	
Add 0 as prefix for Integer part Add 0 as suffix for Fractional part if group is less than 3 bits	0 b ₇ b ₆		b ₅ b ₄ b ₃			b ₂ b ₁ b ₀			.	b ₋₁ b ₋₂ b ₋₃			b ₋₄ b ₋₅ 0	
Each group is converted into a octal number	O ₂		O ₁			O ₀			.	O ₋₁			O ₋₂	
Final Octal Converted number	O₂O₁O₀.O₋₁O₋₂													

Note: Check reference table: 1.1.2.3 at the end of the chapter to convert binary to octal

Thus **Binary number b₇b₆b₅b₄b₃b₂b₁.b₋₁b₋₂b₋₃b₋₄b₋₅** is converted to **octal number (O₂O₁O₀.O₋₁O₋₂)₈**

*Example: convert **1101111.10011b** to **octal** number*

Binary Number	1	1	0	1	1	1	1	.	1	0	0	1	1
Direction of Grouping	←	←	←	←	←	←	←	.	→	→	→	→	→
Group In 3 bits	1	101		111			.	100			11		
Add 0 as prefix for Integer part Add 0 as suffix for Fractional part if group is less than 3 bits	0 0 1		101		111			.	100			11 0	
Each group is converted into a octal number	1		5		7			.	4			6	
Final Octal Converted number	(157.46)₈												

Note: Check reference table: 1.1.2.3 at the end of the chapter to convert octal to binary

Hence 1101111.10011b = (157.46)₈

(b) Binary to Hexadecimal Number

If any binary number is to be converted to hexadecimal, group of four (4) bits are formed with both sides of the fractional point. Fractional point is the reference.

Binary Number	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀	.	b ₋₁	b ₋₂	b ₋₃	b ₋₄	b ₋₅
Direction of Grouping	←	←	←	←	←	←	←	←	.	→	→	→	→	→
Group In 4 bits	b ₇ b ₆ b ₅ b ₄				b ₃ b ₂ b ₁ b ₀				.	b ₋₁ b ₋₂ b ₋₃ b ₋₄				b ₋₅
Add 0 as prefix for integer part Add 0 as suffix for Fractional part if group is less than 4 bits	b ₇ b ₆ b ₅ b ₄				b ₃ b ₂ b ₁ b ₀				.	b ₋₁ b ₋₂ b ₋₃ b ₋₄				b ₋₅ 000
Each group is converted into a Hexadecimal number	H ₁				H ₀				.	H ₋₁				H ₋₂
Final Hexadecimal Converted number	H₂H₁H₀.H₋₁H₋₂													

Note: Check reference table: 1.1.2.3 at the end of the chapter to convert binary to Hexadecimal

Thus **Binary number $b_7b_6b_5b_4b_3b_2b_1.b_{-1}b_{-2}b_{-3}b_{-4}b_{-5}$** is converted to

Hexadecimal number $H_1H_0.H_{-1}H_{-2}$

*Example: convert **110111.10011b** to **Hexadecimal** number*

Binary Number	1	1	0	1	1	1	1	.	1	0	0	1	1
Direction of Grouping	←	←	←	←	←	←	←	.	→	→	→	→	→
Group In 4 bits	110			1111				.	1001			1	
Add 0 as prefix for integer part Add 0 as suffix for Fractional part if group is less than 4 bits	0110			1111				.	1001			1000	
Each group is converted into a Hexadecimal number	6			F(15)				.	9			8	
Final Hexadecimal Converted number	6F.98h												

Hence $110111.10011b = 6F.98h$

Note: Check reference table: 1.1.2.3 at the end of the chapter to convert binary to hexadecimal

(c) Octal to Binary Number

If any octal number is to be converted to binary, each octal digit is replaced by equivalent three (3) binary bits. Fractional point is the reference.

Octal Number to be Converted to Binary number	O_2			O_1			O_0			.	O_{-1}			O_{-2}		
Direction of Grouping	←	←	←	←	←	←	←	←	←	.	→	→	→	→	→	→
Convert each Digit into 3 binary bits	$b_8 b_7 b_6$			$b_5 b_4 b_3$			$b_2 b_1 b_0$.	$b_{-1} b_{-2} b_{-3}$			$b_{-4} b_{-5} b_{-6}$		
Binary Number	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	.	b_{-1}	b_{-2}	b_{-3}	b_{-4}	b_{-5}	b_{-6}

Note: Check reference table: 1.1.2.3 at the end of the chapter to convert binary to octal

Thus **octal number $O_2O_1O_0.O_{-1}O_{-2}$** is converted to **Binary number**

$b_8b_7b_6b_5b_4b_3b_2b_1.b_{-1}b_{-2}b_{-3}b_{-4}b_{-5}b_{-6}$

Example: convert (157.46)₈ to binary number

Octal Number to be Converted to Binary number	1			5			7			.	4			6		
Direction of Grouping	←	←	←	←	←	←	←	←	←	.	→	→	→	→	→	→
Convert each Digit into 3 binary bits	001			101			111			.	100			110		
Binary Number	1101111.10011b															

Note: Check reference table: 1.1.2.3 at the end of the chapter to convert octal to binary

Hence (157.46)₈=1101111.10011b

(d) Hexadecimal to Binary Number

If any hexadecimal number is to be converted to binary, each hexadecimal digit is replaced by equivalent four (4) binary bits. Fractional point is the reference.

Hexadecimal Number to be Converted to Binary number	H ₁				H ₀				.	H ₋₁				H ₋₂			
Direction of Grouping	←	←	←	←	←	←	←	←	.	→	→	→	→	→	→	→	→
Convert each Digit into 4 Equivalent binary bits	b ₇ b ₆ b ₅ b ₄				b ₃ b ₂ b ₁ b ₀				.	b ₋₁ b ₋₂ b ₋₃ b ₋₄				b ₋₅ b ₋₆ b ₋₇ b ₋₈			
Binary Number	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀	.	b ₋₁	b ₋₂	b ₋₃	b ₋₄	b ₋₅	b ₋₆	b ₋₇	b ₋₈

Note: Check reference table: 1.1.2.3 at the end of the chapter to convert Hexadecimal to binary

Thus **Hexadecimal number H₁H₀.H₋₁H₋₂** is converted to

Binary number b₇b₆b₅b₄b₃b₂b₁.b₋₁b₋₂b₋₃b₋₄b₋₅b₋₆b₋₇b₋₈

Example: convert 6F.98h to binary number

Hexadecimal Number to be Converted to Binary	6				F				.	9				8			
---	---	--	--	--	---	--	--	--	---	---	--	--	--	---	--	--	--

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

number									.								
Direction of Grouping	←	←	←	←	←	←	←	←	.	→	→	→	→	→	→	→	→
Convert each Digit into 4 Equivalent binary bits	0110				1111				.	1001				1000			
Binary Number	1101111.10011b																

Note: Check reference table: 1.1.2.3 at the end of the chapter to convert Hexadecimal to binary

Thus **Hexadecimal number 6F.98h** is converted to **Binary number 1101111.10011b**

Table:1.1.2.3.

REFERENCE TABLE-for octal/Hexadecimal to binary conversion and vice versa

OCTAL	Binary Equivalent (3 Bits)
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

HEXADECIMAL	Binary Equivalent (4 Bits)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Solve the questions below:

Convert the following to equivalent binary number

(1.) 564.689d, (2.) (732.672)₈, (3.) FA5.37Bh

Convert the following to equivalent Decimal number

(1.) 110111.111101b, (2.) (732.672)₈, (3.) FA5.37Bh

Convert the following to equivalent Hexadecimal number

(1.) 110111.111101b, (2.) (732.672)₈, (3.) 739.3471d

Binary System

Binary system is extensively used in machines. The language used for communication inside digitized machines is machine language comprising of orderly way of binary system.

Binary system uses two digits 0 and 1 called individually as *bit*. So bit is the basic unit of binary system.

1 0 are single bit numbers, 00, 01, 10, 11 are two-bit numbers and so on.

TECHNICAL NAME	BIT	NIBBLE	BYTE
NO. OF BITS	Single bit	Four bits	Eight bits
EXAMPLE	1 or 0	1110	1110 0101

Group of 4-bit are termed as NIBBLE, group of 8-bit are termed as BYTE.

WORD-SIZE or WORD-LENGTH is a term used in computers which denotes the number of bits; the machine can handle at one instance.

What is a complement?

For bit 0, bit 1 is the complement and for bit 1, bit 0 is the complement.

What is 1's complement?

For a binary word, taking complement for each bit is termed as 1's complement.

For 1001, 1's complement is 0110. We can see each bit is complemented individually.

4-bit number

	MSB			LSB
Weight	8	4	2	1
Binary Number	1	1	1	1

8-bit number

	MSB							LSB
Weight	128	64	32	16	8	4	2	1
Binary Number	1	1	1	1	1	1	1	1

- MSB - MOST SIGNIFICANT BIT- the bit at the left most end of any binary word
- MSB - the bit that has **highest weight** and so termed MOST SIGNIFICANT BIT
- LSB - LEAST SIGNIFICANT BIT - the bit at the Right most end of any binary word
- LSB - the bit that has **lowest weight** and so termed LEAST SIGNIFICANT BIT

1.2 Binary Codes

Binary code is which assigns binary word to each symbol or instruction. Like alphabet 'A' can be assigned a binary word 110111 termed as code for letter 'A'.

Let us see some of the binary codes:

Binary Code 1: BCD (Binary Coded Decimal)

The BCD (Binary Coded decimal) or otherwise termed as '8421' code comprises of four-bit binary equivalent for each decimal digit 0 -9. (Fixed-width) as name states the each decimal digit is encoded in its binary form.

Usually BCD can be represented by using four-bit (Nibble) or eight-bit (Byte) binary equivalent for each decimal digit from 0 to 9. But 4-bit representation is taken as common one.

Table- A 4-bit BCD code for equivalent for each Decimal Digit

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

BCD code for some decimal numbers are shown in the following tables

Decimal	1	0
BCD	0001	0000

Decimal	4	9
BCD	0100	1001

Decimal	3	5	7
BCD	0011	0101	0111

The decimal 10d is written as 0001 0000, 49d as 0100 1001 and 358 as 0011 0101 0111 in BCD. This is termed as 8421 code because the four-bit position weights from MSB to LSB are 8421 respectively.

Decimal	6			
Weights	8	4	2	1
BCD	0	1	1	0

In the above table, when we add the weights where bit 1 appears below the particular weights. Now in the above table, bit 1 appears where weights are 4 and 2. If 4 and 2 are added we get the decimal number.

Example:

Decimal	9			
Weights	8	4	2	1
BCD	1	0	0	1

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

In the above table, adding weights 8 and 1 where bits are 1, we get 9 (=8+1) which is the decimal equivalent of the BCD code 1001.

BCD is a **weighted code** because it uses weights while encoding. Weighted code is one that uses position and its weight while coding. As stated in above tables we can see the addition of respective weights whose bits are 1, it equals the decimal number.

Binary Code 2: Excess-3 Code

It is also nearly a four-bit representation like BCD code. It is just a BCD representation added with binary 3 for each decimal.

Decimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
Excess -3	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100

How excess-3 codes are formed.

	Decimal	5
	BCD	0101
+	Binary Equivalent of 3d	0011
=	Excess -3	1000

Step 1: Convert each decimal digit into its 4-bit BCD equivalent code.

Step 2: Add 0011 with the BCD equivalent code

Step 3: The final 4-bit code is the **Excess-3 Code**. Meaning is each BCD is excess 3 of original number. For decimal 5d, the excess-3 code is four-bit binary equivalent of number 8d, which is excess 3d than 5d.

The knowledge of 9's complement is inevitable to learn about self-complement

Decimal	0	1	2	3	4	5	6	7	8	9
9's Complement	9	8	7	6	5	4	3	2	1	0

The table above shows 9's complement of each decimal. As shown, 9d is 9's complement of decimal 0d and 6d is 9's complement of decimal 3d.

How to obtain 9's complement?

With a decimal number, if the complement is added the final answer is 9d (maximum value digit in decimal). Thus 6 + **3** = 9, 3d is the complement of decimal 6d.

What is self-complementing code?

Self-complementing codes have the property that the 9's **complement** of a decimal number is obtained directly by changing 1's to 0's and 0's to 1's.

Excess-3 is an example of self complementing code.

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

Row	Decimal	Excess -3	1's complement	9's complement
1	6	1001	0110	3
2	3	0110	1001	6
3	4	0111	1000	5
4	5	1000	0111	4

In the above table two examples are shown for self-complementing codes.

Taking rows 1 & 2 to consideration.

Row 1: 1001 is the excess-3 code of decimal 6d, and 0110 is its 1's complement.

Row 2: But 0110 is the excess-3 code of decimal 3d and 1001 is its 1's complement.

It proves that 6 and 3 are 9's complements. Their excess-3 codes are 1's complements.

With rows 3 & 4:

It proves that 4 and 5 are 9's complements. Their excess-3 codes are 1's complements.

This property of complementing is called self-complementing.

This code is not a weighted code, because the weight of binary equivalent is not equal to the decimal equivalent. For decimal 4d, the excess-3 code is 0111b, the weight of the code is 7 not 4. So the code is non-weighted code.

Thus BCD is a **weighted code** and Excess-3 code is a **non-weighted code** and **self-complementing code**.

Binary Code 3: Gray code

Gray code else termed as reflected binary code is one of the non-weighted binary code.

What is a transition?

Assume that when an electrical switch is replaced for each bit in a binary number. Suppose switch ON is considered as bit '1' and switch OFF is considered as '0' as shown in the table below.

	Binary
Switch OFF	0
Switch ON	1

Now the decimal numbers 0d and 1d are converted to 2-bit binary equivalent are shown in table below. As shown in the table, switches A & B are used for each bit. To change from decimal '0' to '1', only switch B should be switched ON (Arrow). Switch B should change from 0 to 1 position, ie.. OFF to ON.

Decimal	2-bit Binary equivalent		Transition 0⇒1, 1⇒0
	Switch	Switch	
	A	B	
0	0	0↓	
1	0	1	0 to 1

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

This change is termed as TRANSITION. If 1 is changed to 0 or 0 to 1 it is called transition.

When decimal 0d is to be changed to 3d, then there needs 2 transitions. Both switches should be changed. Thus it causes two transitions. (Below table)

Decimal	2-bit Binary equivalent		Transition 0⇒1, 1⇒0
	Switch	Switch	
	A	B	
0	0↓	0↓	
3	1	1	0 to 1

When decimal 3d is to be changed to 2d, then there needs 2 transitions. Switch B should be changed from '1' to '0' that is ON to OFF. Thus it causes one transitions. (Below table)

Decimal	2-bit Binary equivalent		Transition 0⇒1, 1⇒0
	Switch	Switch	
	A	B	
3	1	1↓	
2	1	0	1 to 0

So three transitions had been discussed till now to understand what a transition is.

Now in the table below, decimal numbers are converted into their equivalent binary numbers in ascending order.

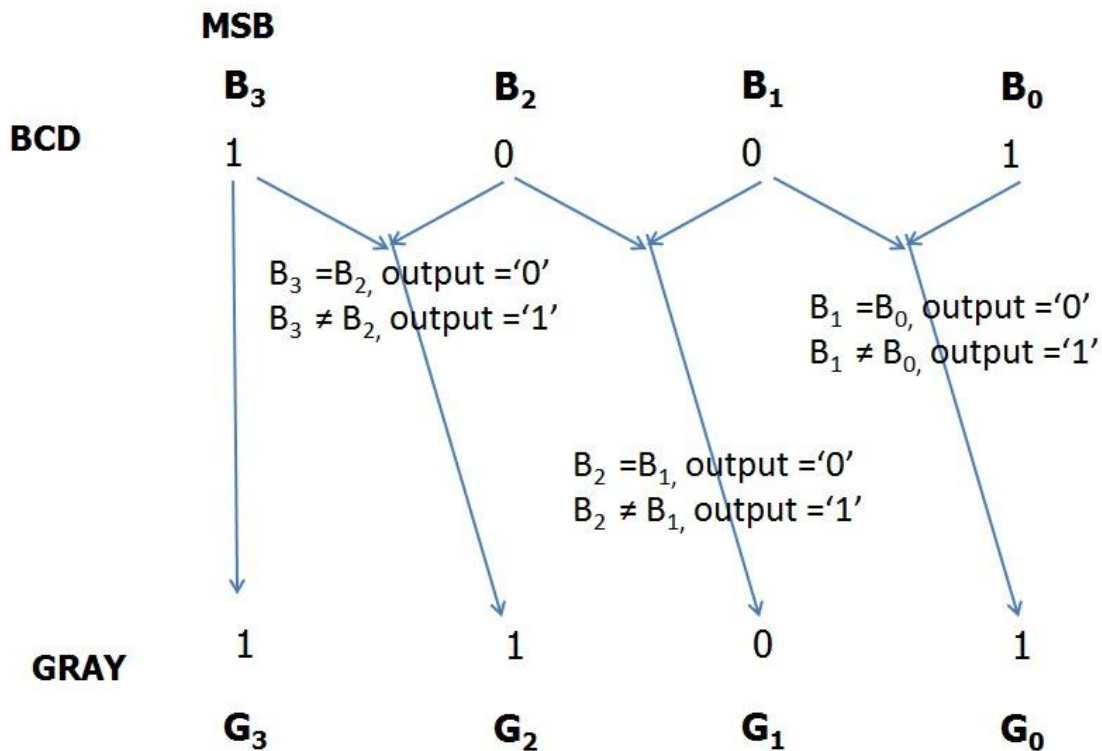
Third column informs about the SWITCHes that transits. Fourth column informs the count of switch transitions.

Decimal	4-bit Binary equivalent				SWITCH Transition	Transition count
	A	B	C	D		
0	0	0	0	0		
1	0	0	0	1	D-0⇒1	1
2	0	0	1	0	C-0⇒1, D-1⇒0	2
3	0	0	1	1	C-0⇒1, D-0⇒1	1
4	0	1	0	0	B-0⇒1, C-1⇒0, D-1⇒0	3
5	0	1	0	1	D-0⇒1	1
6	0	1	1	0	C-0⇒1, D-1⇒0	2
7	0	1	1	1	C-0⇒1, D-0⇒1	1
8	1	0	0	0	A-0⇒1, B-1⇒0, C-0⇒1, D-1⇒0	4
9	1	0	0	1	D-0⇒1	1

It is seen that the transitions are ambiguous and more than one transition are done when switched from one number to another number. So a code that avoids and has only one transition per number change should be devised. The code is GRAY CODE shown in next table.

Gray code was developed by Frank Gray that only has one transition per number change.

How to convert binary number to Gray code?



BCD TO GRAY CODE

Figure: 1.2.1 Binary to Gray Code Conversion

In the figure 1.2.1 above, the BCD (or binary) is converted to Gray code. Binary 4-bit number $B_3B_2B_1B_0$ (1001) is converted to 4-bit Gray code ($G_3G_2G_1G_0$) by using following steps.

Step 1: MSB is used as such. As shown in figure above, bit B_3 is kept as 1. Thus MSB is not changed and the gray code is G_3 .

Step 2: For next position B_2 , MSB bit B_3 and B_2 are compared. When both the bits are same the gray code at position is Binary Bit '0'. If they are different the bit is '1'. In example, both the bits are different, so the gray code at position G_2 is bit '1'.

Step 3: For next position B_1 , Position bit B_2 and B_1 are compared. When both the bits are same the gray code at position is Binary Bit '0'. If they are different the bit is '1'. In example, both the bits are same (bit '0'), so the gray code at position G_1 is bit '0'.

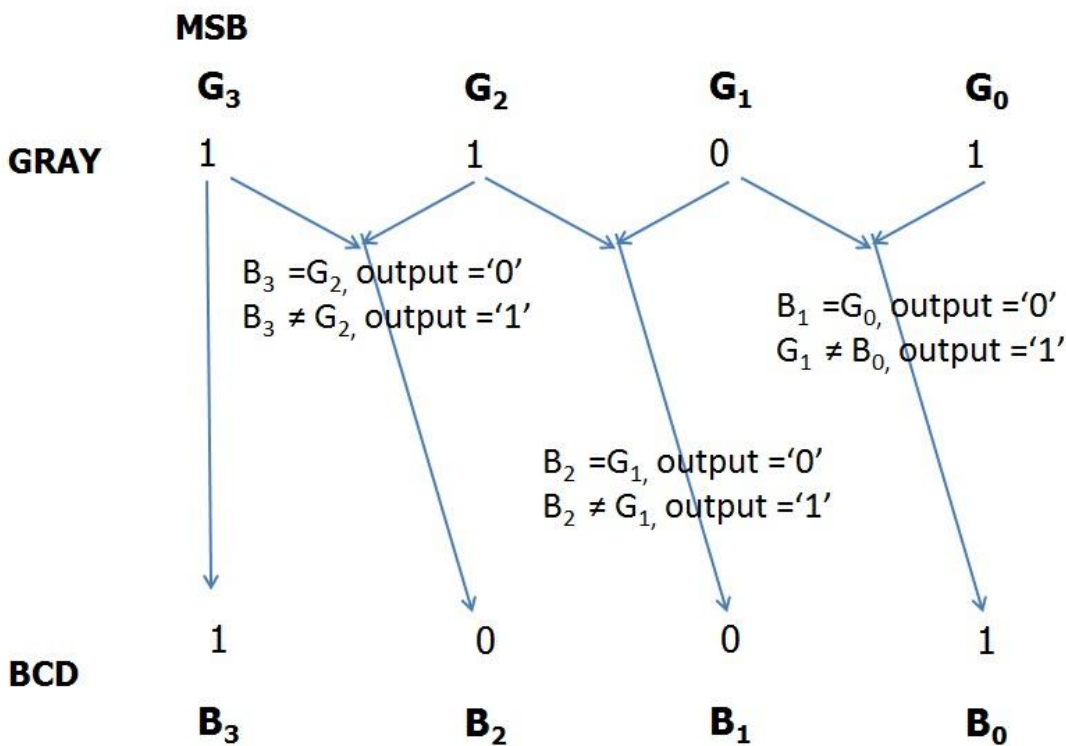
Step 4: For next bits the comparison is continued.

In the table below decimal and relevant Gray code are shown. The transition columns show that only one transition happens per number increment.

Decimal	Gray code				SWITCH Transition	Transition count
	A	B	C	D		
0	0	0	0	0		
1	0	0	0	1	D-0→1	1
2	0	0	1	1	C-0→1	1
3	0	0	1	0	D-1→0	1
4	0	1	1	0	B-0→1	1
5	0	1	1	1	D-0→1	1
6	0	1	0	1	C-1→0	1
7	0	1	0	0	D-1→0	1
8	1	1	0	0	A-0→1	1
9	1	1	0	1	D-0→1	1

How to convert Gray code to binary number?

As shown in the figure 1.2.2 below, the method to convert Gray code to BCD (or Binary) is to be followed.



GRAY TO BCD CODE

Figure: 1.2.2 to Gray Code to Binary Coded Decimal Conversion

It is the same as BCD to Gray code only change is here Binary output and gray code input is compared.

Solve

Try converting the following into Gray code

(1.) 5d, (2.) 9d (3.) 30d

(2.) BCD 1011b, (3.) Binary 11001b

1.3 Binary Arithmetic

Binary arithmetic is also an essential part used in Digital computers.

There are binary addition, subtraction, multiplication and division that shall be discussed further.

1.3.1 Binary addition

Here are some rules for binary addition

Two binary bit (A and B) addition - Table

Rule	A	B	A+B	CARRY	SUM
1	0	0	0+0	0	0
2	0	1	0+1	0	1
3	1	0	1+0	0	1
4	1	1	1+1	1	0

Six binary bit addition (A, B, C, D, E and F) - Table

Rule	A	B	C	D	E	F	A+B+C+D+E+F	CARRY-SECOND	CARRY-FIRST	SUM
5	1	1	1	0	0	0	1+1+1+0+0+0	0	1	1
6	1	1	1	1	0	0	1+1+1+1+0+0	1	0	0
7	1	1	1	1	1	0	1+1+1+1+1+0	1	0	1
8	1	1	1	1	1	1	1+1+1+1+1+1	1	1	0

The above tables are the key to binary addition and multiplication.

Binary Addition				
	2	1	0	Position
	C1	C0		Carry (C)
		A1	A0	Augend
+		B1	B0	Addend
	C1	C0+A1+B1=S1 & C1	A0+B0=S0 & C0	Sum (S)
=	C1	S1	S0	

When two binary numbers A and B are added, their bits are individually added with reference to its positions. Here two bit numbers are shown, in the above table.

The augend number A has bits A0 and A1, the addend number B has bits B0 and B1.

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

The LSB bits A0 and B0 are added first. The sum generated is S0 and the carry generated is C0. The sum S0 is kept at same position 0 and carry C0 is carried to next higher position 1.

In position 1, bits C0, A1 and B1 are added so as to generate a sum of S1 and a carry C1. The sum S1 is kept at same position 1 but carry C1 is carried to next higher position 2.

Since the numbers added have two bits only, the position 2 has only Carry C1. Thus the numbers added sum is C1 S1 S0.

Sample addition:

Row	Decimal	Binary					
1		4	3	2	1	0	Position
2		C3	C2	C1	C0		Carry (C)
3		0	0	1	1		
4	9		1	0	0	1	Augend
5	+3		0	0	1	1	Addend
6			0+1+0	1+0+0	1+0+1	1+1	Sum (S)
7	=12	0	1	1	0	0	
8		C3	S3	S2	S1	S0	

The number 9d and 3d are added in the above sample addition. [9d (Augend) and 3d (addend)]

First Number augend 9d whose binary equivalent is a 4-bit number 1001b but second number addend 3d is a 2-bit number 11b. So both the numbers should be equal in bit size, thus both the numbers are made 4-bit size. Second number 3d shall be made 4-bits by adding '0' as prefix to the binary bits 11 that is 0011b=3d. This is shown in the above table in rows 4 & 5.

First position 0 is considered for addition. The sum S0 is written in the same position 0 in the row 7 and its carry C0 is written in the column C0, row 3.

Then second position 1 is considered for addition. The carry C0 is also considered for generating the sum S1 written in the same position 1 in the row 7 and its carry C1 is written in the column C1, row 3.

The remaining positions are also done as done in position 1. The sums S0, S1, S2 and S3 are generated with a carry C3. Since in position 4, only carry C3 is available which generates the sum S4.

Now the resultant sum is C3 S3 S2 S1 S0 that is 0 1 1 0 0 which can be written as 1100b equivalent to decimal 12d. (Discarding prefix 0)

Summary:

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

- Both the numbers to be added should have same number of bits. If not so, bit 0 is appended as prefix to make both the binary numbers equal.
- The final sum is appended with final carry as a prefix. MSB shall be finally generated carry and discarded if carry is 0.

1.3.2 Binary subtraction

Here are some rules for binary subtraction

Two binary bit (A and B) Subtraction - Table

Rule	A	B	A-B	Borrow	Difference
1	0	0	0-0	0	0
2	0	1	0-1	1	0
3	1	0	1-0	0	1
4	1	1	1-1	0	0

Binary Subtraction				
	2	1	0	Position
	BR3	BR2	BR1	Borrow(BR)
		A1	A0	Minuend
-		B1	B0	Subtrahend
	BR3	(BR2+A1)-B1=D1	(BR1+A0)-B0=D0	Difference (D)
=	BR3	D1	D0	

Concept of binary Borrow is explained below

	2	1	0	Position
	BR3	BR2	BR1	Borrow(BR)
	↷	1 ↷ 1	1 1	
Number	1 0			
	When a bit 1 is donated to next lower position, then bit 0 shall be replaced instead of bit 1	When a bit 1 is borrowed from higher position, then it is equivalent to 2 that is consist 2 bits of 1 each.	When a bit 1 is borrowed from higher position, then it is equivalent to 2 that is consist 2 bits of 1 each.	

The method of subtraction is explained through a sample subtraction below with the methods explained above.

Sample addition:

Row	Decimal	Binary				
1		3	2	1	0	Position
2		B4	B3	B2	B1	Borrow(BR)
3		↩	1↩ 1	1 1		
4	9	10	0	0	1	
5	-3	0	0	1	1	Subtrahend
6		(0+0)-0=0	(1+0)-0=1	(1+1)-1=1	1-1=0	Difference (D)
7	=6	0	1	1	0	
8		D3	D2	D1	D0	

The number 9d and 3d are SUBTRACTED in the above sample addition. [9d (Minuend) and 3d (Subtrahend)]

First Number Minuend 9d whose binary equivalent is a 4-bit number 1001b but second number subtrahend 3d is a 2-bit number 11b. So both the numbers should be equal in bit size, thus both the numbers are made 4-bit size. Second number 3d shall be made 4-bits by adding '0' as prefix to the binary bits 11 that is 0011b=3d. This is shown in the above table in rows 4 & 5.

First position 0 is considered for Subtraction. The difference D0 is 0 as shown in the above table.

Then second position 1 is considered for subtraction. The minuend is smaller (0) than subtrahend (1) and thus direct subtraction cannot be done.

Now for subtraction we need to borrow from higher position 2. But there also the minuend bit 2 is 0 then number can be borrowed from position 3. When borrowed bit 1 from position 3 to position 2, the bit becomes 0 at position 3 and position 2 has two bit 1. Similarly now bit 1 is borrowed from position 2 to position 1. Thus position 1 has two borrow bits of 1 as shown in row 3 of above table.

Now position 1 has two borrow bits position 2 has one borrow bit and Minuend bit at position 3 is 0.

After subtraction [(borrow bits+ Minuend bit)-Subtrahend bit- Difference] we get difference as 0110 which is decimal 6d.

Than this direct subtraction methods, 1's complement addition and 2's complement addition are adapted.

1's complement addition method for Binary subtraction:

What is 1's complement?

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

For a binary word, taking complement for each bit is termed as 1's complement.
 For 1001, 1's complement is 0110. We can see each bit is complemented individually.
 Steps of 1's complement addition instead of subtraction.

Step 1: 1's complement of Subtrahend is taken.

Step 2: Then the 1's complemented subtrahend is added with minuend.

Step 3:

- Here final carry (MSB) generated is 1, and then add the carry to the remaining sum becomes the final difference.
- If the final carry (MSB) generated is 0, then the final difference is the 1's complement of the sum generated.

Sample 1's complement subtraction:

(1.) Subtract 11d – 9d by converting to binary.

(2.) Subtract 9d – 11d 9d by converting to binary.

(1.) Subtract 11d – 9d by converting to binary.

(1) Minuend = 11d, Subtrahend = 9d

11d= 1011b, 9d =1001b

Step 1:

- 1's complement of Subtrahend is taken.
- Subtrahend 9d=1001b is converted to its 1's complement, 1's complement 1001b is 0110b

Step 2: Then the 1's complemented subtrahend is added with minuend.

Step 3: Here final carry (MSB shown in highlight with thick border) generated is 1, then add the carry to the remaining sum becomes the final difference.

Decimal Equivalent		4	3	2	1	0	Position
		1	1	1	0		Carry
11d			1	0	1	1	Minuend
1's Complement of 9d	+		0	1	1	0	1's Complement Subtrahend
1d			0	0	0	1	Sum
	+					1	Add Carry
2d			0	0	1	0	Final Difference

Note: When *minuend* is *greater* than *subtrahend*, the carry generated shall be bit 1.

(2.) Subtract 9d – 11d 9d by converting to binary.

(2) Minuend = 9d, Subtrahend = 11d

9d =1001b, 11d= 1011b

Step 1:

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

- 1's complement of Subtrahend is taken.
- Subtrahend 11d=1011b is converted to its 1's complement, 1's complement 1011b is 0100b
Step 2: Then the 1's complemented subtrahend is added with minuend.

Step 3:

- If the final carry (MSB) generated is 0, then the final difference is the 1's complement of the sum generated.

Decimal Equivalent		4	3	2	1	0	Position
		0	0	0	0		Carry
9d			1	0	0	1	Minuend
1's Complement of 11d	+		0	1	0	0	1's Complement Subtrahend
13d			1	1	0	1	Sum
2d			0	0	1	0	1's Complement of Sum Is the Difference

Note: When *minuend* is *less* than *subtrahend*, the carry generated shall be bit 0.

2's complement addition method for Binary subtraction:

What is 2's complement?

For a binary word, taking complement for each bit is termed as 1's complement and adding bit 1 to it is termed as 2's complement of the number.

For 1001, 1's complement is 0110. We can see each bit is complemented individually.

The bit 1 is added.

1	0	1	1	Number (9d)
0	1	1	0	1's Complement
+			1	Add Bit 1
0	1	1	1	2's Complement

Steps of 2's complement addition instead of subtraction.

Step 1: 2's complement of Subtrahend is taken.

Step 2: Then the 2's complemented subtrahend is added with minuend.

Step 3:

- Here final carry (MSB) generated is 1, and then sum **discarding the carry** shall be the final difference.
- If the final carry (MSB) generated is 0, then the final difference is the 2's complement of the sum generated.

Sample 2's complement subtraction:

(1.) Subtract 11d – 9d by converting to binary.

(2.) Subtract 9d – 11d 9d by converting to binary.

(1.) Subtract 11d – 9d by converting to binary.

(1) Minuend = 11d, Subtrahend = 9d

11d = 1011b, 9d = 1001b

Step 1:

- 2's complement of Subtrahend is taken.
- Subtrahend 9d=1001b is converted to its 2's complement, 2's complement 1001b is 0111b

Step 2: Then the 2's complemented subtrahend is added with minuend.

Step 3:

- Here final carry (MSB shown in highlight with thick border) generated is 1, and then discard carry, remaining sum becomes the final difference.

Decimal Equivalent		4	3	2	1	0	Position
		1	1	1	1		Carry
11d			1	0	1	1	Minuend
2's Complement of 9d	+		0	1	1	1	2's Complement Subtrahend
2d			0	0	1	0	Sum
	+	1					Discard Carry
2d			0	0	1	0	Final Difference

Note: When *minuend* is *greater* than *subtrahend*, the carry generated shall be bit 1.

(2.) Subtract 9d – 11d 9d by converting to binary.

(2) Minuend = 9d, Subtrahend = 11d

9d = 1001b, 11d = 1011b

Step 1:

- 2's complement of Subtrahend is taken.
- Subtrahend 11d=1011b is converted to its 2's complement, 2's complement 1011b is 0101b

Step 2: Then the 2's complemented subtrahend is added with minuend.

Step 3:

- If the final carry (MSB) generated is 0, then the final difference is the 2's complement of the sum generated.

Decimal Equivalent		4	3	2	1	0	Position
		0	0	0	1		Carry
9d			1	0	0	1	Minuend
2's Complement of 11d	+		0	1	0	1	2's Complement Subtrahend
		0					Carry is bit 0 sum shall be 2's complemented
13d			1	1	1	0	Sum
			0	0	0	1	1's Complement of Sum

2d			0	0	1	0	2's Complement of Sum Is the Difference
-----------	--	--	----------	----------	----------	----------	--

Note: When *minuend* is *less* than *subtrahend*, the carry generated shall be bit 0.

1.3.3 Binary multiplication

It is nearly similar to the known decimal multiplication. Here addition upto six bits are shown separately in different successive tables. Carry generated in positions are shown. This may give us pre-requisite study on multiplication when we add upto six bits.

Add two bits- all being 1				
1+1 = 10b=0(sum)1(carry)				
	2	1	0	Position
Here carry generated by position 0 is put in next higher position 1				
		1		Carry
			1	Number 1
+			1	Number 2
			0	Sum
		1	0	Final Sum (including Carry)

Add three bits- all being 1				
1+1+1=11b= 1(sum)1(carry)				
	2	1	0	Position
Here carry generated by position 0 is put in next higher position 1				
		1		Carry
			1	Number 1
			1	Number 2
+			1	Number 3
			1	Sum
		1	1	Final Sum (including Carry)

Add Four bits- all being 1				
1+1+1+1=100b= 0(sum)10(carry)				
	2	1	0	Position
Here carry generated by position 0 is put in next higher positions 2 & 1 as 1 & 0 respectively				
	1	0		Carry
			1	Number 1
			1	Number 2
			1	Number 3
+			1	Number 4
			0	Sum
	1	0	0	Final Sum (including Carry)

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

Add Five bits- all being 1				
1+1+1+1+1=101b= 1(sum)10(carry)				
2	1	0	Position	
Here carry generated by position 0 is put in next higher positions 2 & 1 as 1 & 0 respectively				
1	0		Carry	
		1	Number 1	
		1	Number 2	
		1	Number 3	
		1	Number 4	
+		1	Number 5	
		1	Sum	
1	0	1	Final Sum (including Carry)	

Add Six bits- all being 1				
1+1+1+1+1+1= 110b=0(sum)11(carry)				
2	1	0	Position	
Here carry generated by position 0 is put in next higher positions 2 & 1 as 1 & 1 respectively				
1	1		Carry	
		1	Number 1	
		1	Number 2	
		1	Number 3	
		1	Number 4	
		1	Number 5	
+		1	Number 6	
		0	Sum	
1	1	0	Final Sum (including Carry)	

Sample Multiplication

Multiply 31d x 14d using binary multiplication.

31d = 11111b, 14d=1110b

Decimal equivalent		7	6	5	4	3	2	1	0	Position
15d						1	1	1	1	Multiplicand
x14d						1	1	1	0	Multiplier
	Carry added to same position	1 (5)		1 (3)						<i>Carry from (position shown in brackets) Example: Carry generated in position 3 put in position 5 highlighted</i>
	Carry added to same			1 (4)		1 (2)				<i>Carry from next immediate lower position (position shown in brackets)</i>

	position									
	0 x 1111					0	0	0	0	
	1 x 1111			1	1	1	1			
	1 x 1111		1	1	1	1				
	1 x 1111		1	1	1	1				
210		1	1	0	1	0	0	1	0	Product

1.3.4 Binary Division

This is also similar to decimal division.

Sample Division

Divide 31d x 3d using binary multiplication.

31d = 11111b, 3d=110b

	3 2 1 0	Position
	1 0 1 0	Quotient
1 1	1 1 1 1 1	
	1 1	← Quotient bit 1 is written in position 3
	0 0	← Answer is 0 so Quotient bit 0 is written in position 2
	1 1	
	1 1	← Quotient bit 1 is written in position 1
	0 1	Difference less than the Divisor 11b , so in Quotient bit 0 is written in position 0
		Now 01 is the remainder

Thus 31d ÷ 3d = Quotient 10d=1010b and remainder 1d=01b

Solve the questions below:

Add the following using their equivalent binary numbers

(1.) 25d + 31d, (2.) (76)₈ + (13)₈, (3.) FAh +37h

Subtract the following using their equivalent binary numbers by 2's complement method.

(1.) 45d - 31d, (2.) (76)₈ - (13)₈, (3.) FAh -37h

Add the following using their equivalent binary numbers

(1.) 25.32d + 31.21d, (2.) (7.62)₈ + (1.32)₈, (3.) FA5h +37Bh

Add the following using their equivalent binary numbers

(1.) 25.32d + 31.21d, (2.) (7.62)₈ + (1.32)₈, (3.) FA5h +37Bh

1.4 BOOLEAN ALGEBRA AND THEOREMS

1.4.1. Boolean Algebra:

In computer language, **Boolean** is a **data-type** which only has two values: **true** or **false**. Thus **Boolean variables** have only two values **true (1)** or **false (0)**.

Boolean variable representation:

If a variable is A, then it is considered to be true (1). Thus $A=1$.

If the same variable A is complemented as \bar{A} , then it is considered to be false (0). Thus $\bar{A} = 0$. Thus $A=1$ and $\bar{A} = 0$.

Note: Complement can be used in these formats too: \bar{A} (a bar on variable A) or A' (Apostrophe after variable A).

The variables often used are A, B, C, D, E, W, X, Y, Z

1.4.1.1. Boolean operators:

There are two types of Boolean operators: (1) Basic operators, (2) Derived operators.

Basic boolean operators:

There are three basic operators:

1. Negation (NOT) operator,
2. Disjunction (OR) operator and
3. Conjunction (AND) operator

VENN DIAGRAM FOR OPERATORS

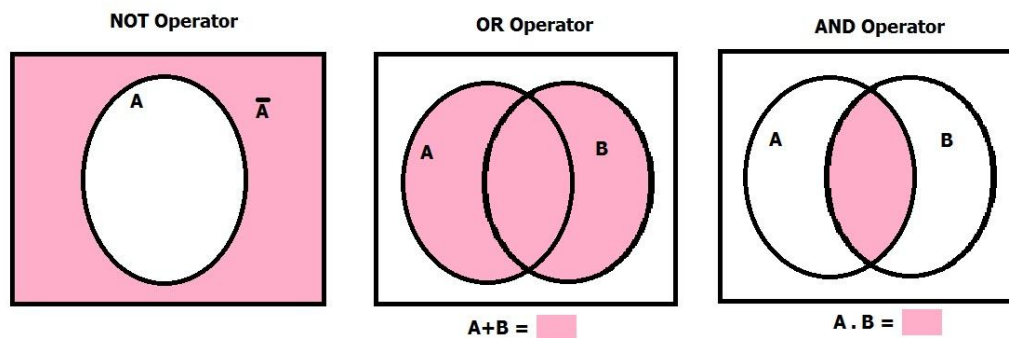


Table 1.4.1: Basic Boolean Operators and operations

BASIC OPERATOR	OPERATION	VARIABLE OPERATION	TRUE OR FALSE	0 OR 1
NOT ($\bar{\quad}$ or \prime)	NOT (A) = \bar{A} (or A')	NOT (A)= A' or \bar{A} NOT (A' or \bar{A})=A	NOT (True)=false NOT (False)=True	NOT (0)=1 NOT (1)=0
OR (+)	A + B		False + False= False False + True = True True + False= True True + True= True	0 + 0= 0 0 + 1 = 1 1 + 0= 1 1 + 1= 1

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

AND (.)	A . B	False . False= False False . True = False True . False= False True . True= True	0 . 0= 0 0 . 1 = 0 1 . 0= 0 1 . 1= 1
------------	-------	--	---

The above table 1.4.1, the operations and operators are shown.

Summary:

NOT operation complements each variable. It's usually a single variable operator.

OR and AND operations are not single variable operators. They need minimum two operators for operation to happen.

OR operation: when one variable of the operation is '1' (True), the output is '1' (True)

Example: $A + B + C = 1$ when only $A=1$, or $B=1$, or $C=1$, and when $A=B=1$, $A=C=1$, $B=C=1$, or $A=B=C=1$.

Output is '1' (=True) when

Case 1: $A=1, B=0, C=0$

Case 2: $A=0, B=1, C=0$

Case 3: $A=0, B=0, C=1$

Case 4: $A=1, B=1, C=0$

Case 5: $A=1, B=0, C=1$

Case 6: $A=0, B=1, C=1$

Case 7: $A=1, B=1, C=1$

Other combinations have output as '0' (=False).

AND operation: when one variable of the operation is '0' (False), the output is '0' (False)

Example: $A . B . C = 0$ when only $A=0$, or $B=0$, or $C=0$, and when $A=B=0$, $A=C=0$, $B=C=0$, or $A=B=C=0$.

Output is '0' (=False) when

Case 1: $A=0, B=0, C=0$

Case 2: $A=1, B=0, C=0$

Case 3: $A=0, B=1, C=0$

Case 4: $A=0, B=0, C=1$

Case 5: $A=1, B=1, C=0$

Case 6: $A=1, B=0, C=1$

Case 7: $A=0, B=1, C=1$

Other combination is when all the variables are '1' (=True). Ie $A=B=C=1$ only the output is '1' (=True)

NOTE: OR operation is termed as "inclusive OR"

1.4.1.2. Derived operators:

One of the main derived operator is *exclusive disjunction* or *exclusive OR (XOR)* operator.

Table 1.4.2: Derived Boolean Operator and operation

DERIVED OPERATOR	OPERATION	VARIABLE OPERATION	TRUE OR FALSE	0 OR 1
XOR (\oplus)	$A \oplus B$		False \oplus False= False False \oplus True = True True \oplus False= True True \oplus True= False	$0 \oplus 0 = 0$ $0 \oplus 1 = 1$ $1 \oplus 0 = 1$ $1 \oplus 1 = 0$

The XOR operation is such that when odd number of variables is True (or '1') then output is True (or '1')

Example: When variables A, B, C, and D are operated with XOR operator, then $A \oplus B \oplus C \oplus D$, the output is '1' (=True) when

Case 1: A=1, B=0, C=0

Case 2: A=0, B=1, C=0

Case 3: A=0, B=0, C=1

Case 4: A=1, B=1, C=1

Other combinations have output as '0' (=False). Thus when one variable is '1' (=True) or all three variables are '1' (=True) have output as '1' (=True)

This operation **XOR** is called **ODD parity Checker** since it gives output when odd number of input variables are '1' (=True)

1.4.2 BOOLEAN THEOREMS:

The below given are the properties and theorems related to Boolean variables.

THEOREM NAME ↓	EXPLANATION	NOT FUNCTION	
		THEOREM	APPLICATION
INVOLUTION	When the complement is of even times (Given Example is 2 times complement), output is the same signal	$\bar{\bar{A}} = A$	$\bar{0} = 0$
			$\bar{1} = 1$
INVOLUTION (odd times)	When the complement is of Odd times (Given Example is 3 times complement), output is the complement signal	$\bar{\bar{\bar{A}}} = \bar{A}$	$\bar{\bar{0}} = 1$
			$\bar{\bar{1}} = 0$

THEOREM NAME ↓	EXPLANATION	OR FUNCTION		AND FUNCTION	
		THEOREM	APPLICATION	THEOREM	APPLICATION
IDEMPOTENT	When identical variables are processed	$A + A = A$	$0 + 0 = 0$	$A.A = A$	$0.0 = 0$
			$1 + 1 = 1$		$1.1 = 1$
COMPLEMENT	When complement variables are processed	$A + \bar{A} = 1$	$0 + 1 = 1$	$A. \bar{A} = 0$	$0.1 = 0$
			$1 + 0 = 1$		$0.1 = 0$

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

COMBINING	When both operators are used, the complement vanishes	$(A.B)+(\bar{A}.B)=B$	$(1.0)+(0.0)=0$	$(A+B).(\bar{A}+B)=B$	$(1+0)+(0+0)=0$
			$(1.1)+(0.1)=1$		$(1+1)+(0+1)=1$
ADSORPTION	Repeated variables is the output	$A+(A.B)=A$	$0+(0.1)=0$	$A.(A+B)=A$	$0.(0+1)=0$
			$1+(1.1)=1$		$1.(1+1)=1$
	Repeated variables is the output	$A+(\bar{A}.B)=A+B$	$0+(1.1)=1$	$A.(\bar{A}+B)=A.B$	$0.(1+1)=0$
			$1.(0+1)=1$		$1.(0+1)=1$

Boolean Property:

PROPERTY NAME ↓	OR FUNCTION		AND FUNCTION	
	THEOREM	APPLICATION	THEOREM	APPLICATION
COMMUTATIVE	$A + B = B + A$	$0+1 = 0+1 = 1$	$A.B = B.A$	$0.1 = 0.1 = 0$
		$1+1 = 1+1 = 1$		$1.1 = 1.1 = 1$
ASSOCIATIVE	$A+(B+C)=(A+B)+C$	$0+(1+1) = (0+1)+1 = 1$	$A.(B.C)=(A.B).C$	$0.(1.1) = (0.1).1 = 0$
		$0+(0+1) = (0+0)+1 = 1$		$0.(0.1) = (0.0).1 = 0$
DISTRIBUTIVE	$A.(B+C)=A.B+A.C$	$0.(1+0)=0.1+0.0=0$	$A+(B.C)=(A+B).(A+C)$	$0+(1.0)=(0+1).(0+0)=0$
		$1.(1+0)=1.1+1.0=1$		$1+(1.0)=(1+1).(1+0)=1$

DEMORGANS THEOREM

THEOREM	PROOF
$\overline{A + B} = \bar{A} . \bar{B}$	$\overline{0 + 0} = \bar{0} . \bar{0} = 0$
	$\overline{0 + 1} = \bar{0} . \bar{1} = 0$
	$\overline{1 + 0} = \bar{1} . \bar{0} = 0$
	$\overline{1 + 1} = \bar{1} . \bar{1} = 1$
$\overline{A . B} = \bar{A} + \bar{B}$	$\overline{0 . 0} = \bar{0} + \bar{0} = 1$
	$\overline{0 . 1} = \bar{0} + \bar{1} = 1$
	$\overline{1 . 0} = \bar{1} + \bar{0} = 1$
	$\overline{1 . 1} = \bar{1} + \bar{1} = 0$

Prove $\overline{A + B} = \bar{A} . \bar{B}$

LHS (Left Hand Side of the equation)

RHS (Right Hand Side of the equation)

			LHS			RHS
A	B	A + B	$\overline{A + B}$	\bar{A}	\bar{B}	$\bar{A} . \bar{B}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Prove $\overline{A . B} = \bar{A} + \bar{B}$

			LHS			RHS
A	B	A . B	$\overline{A . B}$	\bar{A}	\bar{B}	$\bar{A} + \bar{B}$
0	0	0	1	1	1	1

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

The columns LHS and RHS are the same. Thus the theorem is proved. This table is termed as truth table which exhibits truth about the several possible combinations of input variables.

The output is remembered as "break the line" and "Change the sign"

$$\overline{A + B} = \bar{A} \cdot \bar{B}$$

$$\text{LHS} = \overline{A + B}$$

$$\text{RHS} = \bar{A} \cdot \bar{B}$$

In LHS "**break the line**" or bar on the expression $A + B$, $(\overline{A + B}) = \bar{A} + \bar{B}$. Then "**Change the sign**" OR operator '+' is changed as AND operator '.' Thus it becomes $\bar{A} \cdot \bar{B}$

Same for another theorem $\overline{A \cdot B} = \bar{A} + \bar{B}$.

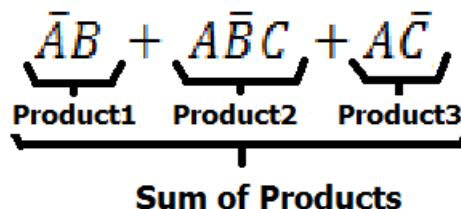
1.5 BOOLEAN FUNCTIONS:

Boolean function: it describes the combinations of Boolean variables that infer particular required output.

- a). When an expression is written in X+Y+Z form, it is called SUM form or using OR variable.
- b). When an expression is written in X.Y.Z form, it is called PRODUCT form or using AND variable.

Sum of products expression (SOP):

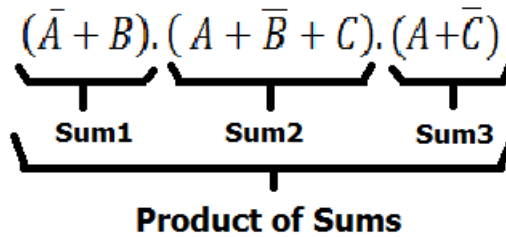
When an expression is written in this form $\bar{A}B + A\bar{B}C + A\bar{C}$, the ANDED variables are ORed later. The ANDED are Products and ORing (SUMming) the products is termed as SUM of PRODUCTS (SoP or SOP).



As shown in the above figure product1, product2 and product3 are AND expressions. The sum is an OR expression, being the OR (Sum) expression of all AND (Products).

Product of Sums expression (POS):

When an expression is written in this form $(\bar{A} + B).(A + \bar{B} + C).(A + \bar{C})$, the ORed variables are ANDed later. The ORed are Sums and ANDing (PRODUCTing) the sums is termed as PRODUCT of SUMS (PoS or POS).



As shown in the above figure, sum1, sum2 and sum3 are OR expressions. The product is an AND expression, being the AND (Product) expression of all OR (Sums).

Maxterms and minterms:

Table 1.5.1: Two variable truth table

Input		Output	Minterm	Maxterm
A	B	F		
0	0	0	$m_0 = \bar{A}\bar{B}$	$M_0 = (A + B)$
0	1	1	$m_1 = \bar{A}B$	$M_1 = (A + \bar{B})$
1	0	1	$m_2 = A\bar{B}$	$M_2 = (\bar{A} + B)$
1	1	0	$m_3 = AB$	$M_3 = (\bar{A} + \bar{B})$

Minterm: In the above table termed as truth table, two inputs A & B are exciting an output F. Since two variables, four combinations 00($\bar{A}\bar{B}$), 01($\bar{A}B$), 10($A\bar{B}$) and 11(AB) and their respective output (F) are shown.

The combinations 01($\bar{A}B$) and 10($A\bar{B}$) excite the output F to True (1). The expressions 01($\bar{A}B$) and 10($A\bar{B}$) are termed as **Minterm**. **Minterm** is a product expression

It is expressed as $F(A, B) = \bar{A}B + A\bar{B}$ or as $F(A, B) = \sum_m(1, 2)$

where

Σ (Sigma) Represents Sum of Products

m represents minterm

m1 & m2 represents 01($\bar{A}B$) and 10($A\bar{B}$) respectively. They are decimal equivalents of the respective expressions.

Thus **minterms** lead to **Sum of products**.(SoP)

Maxterm: The combinations $00(\overline{A}\overline{B})$ and $11(AB)$ excite the output F to False (0). The expressions $00(\overline{A}\overline{B})$ and $11(AB)$ are to be complemented.

Complement of $00(\overline{A}\overline{B}) = \overline{00} = 1+1$ (Demorgan's theorem)

$$1+1 = A + B$$

Likewise complement of $11(AB) = 0 + 0 = \overline{A} + \overline{B}$

Thus $F(A, B) = (A + B) \cdot (\overline{A} + \overline{B})$

The product expressions $(A + B)$ and $(\overline{A} + \overline{B})$ are termed as **Maxterms**.

Maxterm is a sum expression.

It is expressed as $(A, B) = (A + B) \cdot (\overline{A} + \overline{B})$ or as $F(A, B) = \prod_M(0, 3)$

where

Π (π) Represents Product of Sums

M represents Maxterm

M0 & M3 represents $(A + B)$ and $(\overline{A} + \overline{B})$ respectively. They are decimal equivalents of the respective expressions.

Thus **maxterms** lead to **Product of Sums** (PoS)

Duality of Minterm and Maxterms:

1) Maxterm = Complement of Minterm

$$F(A, B) = \overline{AB} + \overline{A\overline{B}} = \overline{F(A, B)} = \overline{\overline{AB} + \overline{A\overline{B}}} = m1, m2$$

$$\overline{F(A, B)} = \overline{\overline{AB} + \overline{A\overline{B}}} = \overline{\overline{AB}} \cdot \overline{\overline{A\overline{B}}} \quad \text{(Demorgan's Theorem)}$$

$$\overline{F(A, B)} = \overline{\overline{AB}} \cdot \overline{\overline{A\overline{B}}} = (A + \overline{B}) \cdot (\overline{A} + B) \quad \text{(Demorgan's Theorem)}$$

$$\overline{F(A, B)} = (A + \overline{B}) \cdot (\overline{A} + B) = \overline{m1}, \overline{m2} = M1, M2$$

$$M1 = (A + \overline{B}) = \overline{m1} = \overline{\overline{A} \cdot B}$$

$$M2 = (\overline{A} + B) = \overline{m2} = \overline{A \cdot \overline{B}}$$

2) Minterm = Complement of Maxterm

$$F(A, B) = (A + B) \cdot (\overline{A} + \overline{B}) = \overline{F(A, B)} = \overline{(A + B) \cdot (\overline{A} + \overline{B})} = M0, M3$$

$$\overline{F(A, B)} = \overline{(A + B) \cdot (\overline{A} + \overline{B})} = \overline{(A + B)} + \overline{(\overline{A} + \overline{B})} \quad \text{(Demorgan's Theorem)}$$

$$\overline{F(A, B)} = \overline{(A + B)} + \overline{(\overline{A} + \overline{B})} = (\overline{A} \cdot \overline{B}) + (A \cdot B) \quad \text{(Demorgan's Theorem)}$$

$$\overline{F(A, B)} = (\overline{A} \cdot \overline{B}) + (A \cdot B) = \overline{M0}, \overline{M3} = m0, m3$$

$$M0 = (A + B) = \overline{m0} = \overline{\overline{A} \cdot \overline{B}}$$

$$M3 = (\overline{A} + \overline{B}) = \overline{m3} = \overline{A \cdot B}$$

EXAMPLE 2: list the minterms and max terms of given Three variable

Table 2.5.2: Three variable truth table

Input			Output	Minterm	Maxterm
A	B	C	F		
0	0	0	0	$m_0 = \bar{A}\bar{B}\bar{C}$	$M_0 = (A + B + C)$
0	0	1	1	$m_1 = \bar{A}\bar{B}C$	$M_1 = (A + B + \bar{C})$
0	1	0	1	$m_2 = \bar{A}B\bar{C}$	$M_2 = (\bar{A} + B + \bar{C})$
0	1	1	0	$m_3 = \bar{A}BC$	$M_3 = (A + \bar{B} + \bar{C})$
1	0	0	0	$m_4 = A\bar{B}\bar{C}$	$M_4 = (\bar{A} + B + C)$
1	0	1	0	$m_5 = A\bar{B}C$	$M_5 = (\bar{A} + B + \bar{C})$
1	1	0	1	$m_6 = AB\bar{C}$	$M_6 = (\bar{A} + \bar{B} + C)$
1	1	1	0	$m_7 = ABC$	$M_7 = (\bar{A} + \bar{B} + \bar{C})$

Using output F=True (1):

$F(A, B, C) = \sum_m(1, 2, 6)$ using output F is true (1) [minterm]

$$F(A, B, C) = m_1 + m_2 + m_6$$

$$F(A, B, C) = \bar{A}\bar{B}C + \bar{A}B\bar{C} + AB\bar{C} \text{ [SUM OF PRODUCTS]}$$

This SOP Function excites output to true (1)

Complement of F

$$\bar{F}(A, B, C) = \overline{m_1 + m_2 + m_6}$$

$$\bar{F}(A, B, C) = \overline{\bar{A}\bar{B}C + \bar{A}B\bar{C} + AB\bar{C}}$$

$$\bar{F}(A, B, C) = \overline{\bar{A}\bar{B}C} \cdot \overline{\bar{A}B\bar{C}} \cdot \overline{AB\bar{C}}$$

$$\bar{F}(A, B, C) = (A + B + \bar{C}) \cdot (A + \bar{B} + C) \cdot (\bar{A} + \bar{B} + C)$$

$$\bar{F}(A, B, C) = (M_1) \cdot (M_2) \cdot (M_6)$$

The above function \bar{F} is true(1) when F is false(0). The maxterms M1, M2, and M6 excites output to \bar{F} which is true(1).

Using output F=False(0):

Products of sums (POS)-[Maxterms]:

$F(A, B, C) = \prod_M(0, 3, 4, 5, 7)$ using output F is False (0)[Maxterm]

$$F(A, B, C) = (M_0) \cdot (M_3) \cdot (M_4) \cdot (M_5) \cdot (M_7)$$

$$F(A, B, C) = (A + B + C) \cdot (A + \bar{B} + \bar{C}) \cdot (\bar{A} + B + C) \cdot (\bar{A} + B + \bar{C}) \cdot (\bar{A} + \bar{B} + \bar{C}) \text{ [PRODUCTS OF SUM]}$$

Complement of F

$$\bar{F}(A, B, C) = \overline{(M_0) \cdot (M_3) \cdot (M_4) \cdot (M_5) \cdot (M_7)}$$

$$\bar{F}(A, B, C) = \overline{(A + B + C) \cdot (A + \bar{B} + \bar{C}) \cdot (\bar{A} + B + C) \cdot (\bar{A} + B + \bar{C}) \cdot (\bar{A} + \bar{B} + \bar{C})}$$

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

$$\bar{F}(A, B, C) = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C} + A\bar{B}C + ABC$$

$$\bar{F}(A, B, C) = (m_0) + (m_3) + (m_4) + (m_5) + (m_7)$$

The above function \bar{F} is False(0) when F is True(1). The minterms $m_0, m_3, m_4, m_5,$ and m_7 excites output to \bar{F} which is False(0).

Summary:

1. $F(A, B, C) = \sum_m(1, 2, 6)$ **(Minterm form)**

2. $F(A, B, C) = \prod_M(0, 3, 4, 5, 7)$ **(Maxterm form)**

3. $F(A, B, C) = \bar{A}\bar{B}C + \bar{A}B\bar{C} + AB\bar{C}$ **[SUM OF PRODUCTS]**

4. $F(A, B, C) = (A + B + C). (A + \bar{B} + \bar{C}). (\bar{A} + B + C). (\bar{A} + B + \bar{C}). (\bar{A} + \bar{B} + \bar{C})$ **[PRODUCTS OF SUM]**

DON'T CARE CONDITIONS

What is a DON'T care condition?

The condition that can be predicted as true(1) or False(0) as per circumstances is termed as DON'T care condition.

Example: Consider representing days of a week with numbers starting from 0 to 7 as shown in the table.

Days of week	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Number	0	1	2	3	4	5	6

As per binary representation, 1 variable represent $2^1=2$ expressions, 2 variables represent $2^2=4$ expressions, 3 variables represent $2^3=8$ expressions, 4 variables represent $2^4=16$ expressions and so on.

The formula for Representation is **BASE^{VARIABLE}=TOTAL EXPRESSIONS.**

According to the above representation rule, weekdays can be expressed by 3 variables. 3 variables can represent maximum 8 expressions. But only days are to be expressed. Thus the function is written as

$$F(A, B, C) = \sum m(0, 1, 2, 3, 4, 5, 6) + d(7)$$

In the above switching function F, $d(7)$ can be termed as DON'T care because it can be wither used or not.

How to use? Represent the numbers from '1' to '7' omitting '0'. Now number '7' is used and number '0' becomes unused. Thus don't care means it may or may not be considered. When we use 1 to 7, then we could say '0' becomes don't care.

Another example:

We have five fingers in one hand (normal Human). Suppose we should show to a child count upto 4. That is, we should show the counts 1, 2, 3 & 4 to the child.

There are many options to show number 4 by using the stated combinations of fingers.

1. Index, middle, ring and little,

2. *thumb*, index, middle and ring,
3. *thumb*, middle, ring and little,
4. *thumb*, index, middle and little,
5. *thumb*, index, ring and little and
6. *thumb*, middle, ring and little.

But usually we show number 4 by using option 1 stated above. Thus

F (Function to show number 4 fingers in one hand)= show by (index, middle, ring, little) + don't care (Thumb)

So now thumb finger is stated to be DON'T care. Don't care shall be extensively used while designing digital circuits.

Standard and Canonical forms:

Canonical form is formed directly from the truth table by using minterms or maxterms.

Canonical form consists of all variables in each expression of the function.

$$1. F(A, B, C) = \bar{A}\bar{B}C + \bar{A}B\bar{C} + AB\bar{C}$$

[SUM OF PRODUCTS]

$$2. F(A, B, C) = (A + B + C). (A + \bar{B} + \bar{C}). (\bar{A} + B + C). (\bar{A} + B + \bar{C}). (\bar{A} + \bar{B} + \bar{C})$$

[PRODUCTS OF SUM]

For example, in the above three variable (variables A, B & C) functions, all the variables are occurring in each expression.

In function 1, three (3) expressions $\bar{A}\bar{B}C$, $\bar{A}B\bar{C}$ and $AB\bar{C}$ exist. It can be noticed three variables existing in all the expressions. In function 2 also same can be noticed in all 5 expressions.

$$3. F(A, B, C) = \bar{A}C + \bar{A}B + AB\bar{C}$$

[SUM OF PRODUCTS]

But in the function 3 above it is noticed that only variables A & C are found in expression $\bar{A}C$ and variables A & B are found in expression $\bar{A}B$. This is supposed to be termed as **standard form**.

Thus function f with three variables A, B and C can be written in four forms.

$$1. f(A, B, C) = \bar{A}B + A\bar{B}C + A\bar{C} \quad \text{(Standard form)}$$

$$2. f(A, B, C) = \bar{A}BC + \bar{A}B\bar{C} + A\bar{B}C + AB\bar{C} + AB\bar{C} \quad \text{(Canonical form)}$$

$$3. f(A, B, C) = \Sigma m(0, 2, 3, 6, 7) \quad \text{(Minterm form)}$$

$$4. f(A, B, C) = \Pi m(1, 4, 5) \quad \text{(Maxterm form)}$$

1.5.1 MINIMIZATION OF BOOLEAN FUNCTIONS:

Usually converting a canonical form function to a standard form function is termed as **minimization**. This minimization can be done by using various methods.

We shall discuss three methods of boolean minimization

1. Using Boolean theorems.
2. Using Karnaugh map (or K-Map) and

3. Using Tabulation method (or Quine Mclusky Method)

Why minimization is done?

Minimization is done to reduce number of redundant (reoccurring) expressions without which the desired output can be derived.

Suppose 5 expressions are there in a function initially and the output is true(1). By using minimization, the function consists only 3 expressions but the output is true(0), then the minimization technique is said to be appropriate. This reduces the usage of less physical components in the circuit.

Thus if the same original output is derived from the minimized or reduced function, the function can be considered to be minimized.

Minimization using Boolean theorems:

It is one of the basic methods to minimize Boolean function.

Example 1:

Minimize the function $F(A, B) = AB + A\bar{B} + AB$

Minimization using Boolean theorems:

$F(A, B) = AB + A\bar{B} + AB$ **(Equation 1)**

This is a two variable function in canonical form. The two variables are A & B.

by taking common identity A, the function can be rewritten as

$F(A, B) = AB + A(\bar{B} + B)$ **(Equation 2)**

by using complement OR theorem $\bar{B} + B = 1$, the function can be rewritten as

$F(A, B) = AB + A$ **(Equation 3)**

by using Adsorption theorem $A + AB = A$, the function can be rewritten as

$F(A, B) = A$ **(Equation 4)**

Thus the function which had 3 variables has been reduced to one variable and it is now a standard form.

$F(A, B) = AB + A\bar{B} + AB$ can be written as $F(A, B) = A$ in minimized form.

Example 2:

Minimize the function $F(A, B, C) = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + ABC + ABC$

Minimization using Boolean theorems:

$F(A, B, C) = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + ABC + ABC$ **(Equation 1)**

This is a three variable function in canonical form. The two variables are A, B & C.

by taking common identity $\bar{A}C$ in 1st and 2nd expressions (shown in color) and AC in the 3rd and 5th expressions, the function $F(A, B, C) = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + ABC + ABC$ can be rewritten as

$$F(A, B, C) = \bar{A}C(\bar{B} + B) + AC(\bar{B} + B) + ABC\bar{C} \text{ (Equation 2)}$$

by using complement OR theorem $\bar{B} + B = 1$, the function can be rewritten as

$$F(A, B, C) = \bar{A}C + AC + ABC\bar{C} \text{ (Equation 3)}$$

by taking common C of expression 1 & 2 of equation 3 we get

$$F(A, B, C) = C(\bar{A} + A) + ABC\bar{C} = C + ABC\bar{C} \text{ (Equation 4)}$$

By using adsorption theorem $A + \bar{A}B = A + B$ in equation 4, we get

$$F(A, B, C) = C + ABC\bar{C} = C + AB \text{ (Equation 5)}$$

Thus the function which had 5 variables has been reduced to two variables and it is now a standard form.

$F(A, B, C) = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$ can be written as $F(A, B, C) = AB + C$ in minimized form.

Note: As stated earlier, variables can also be w, x, y, z.

Example

Express the Boolean function $F = A + \bar{B}C$ in a sum of minterms (SOP).

Solution

The term A is missing two variables because the domain of F is (A, B, C)

$$A = A(B + \bar{B}) = AB + A\bar{B} \quad \text{because } B + \bar{B} = 1$$

$\bar{B}C$ missing A, so

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

$$AB(C + \bar{C}) = ABC + AB\bar{C}$$

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

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

Because $A + A = A$

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

$$F = m_7 + m_6 + m_5 + m_4 + m_1$$

In short notation

$$F(A, B, C) = \sum(1, 4, 5, 6, 7)$$

$$\bar{F}(A, B, C) = \sum(0, 2, 3)$$

Example

Express $F = xy + \bar{x}z$ in a product of maxterms form.

Solution

$$F = xy + \bar{x}z = (xy + \bar{x})(xy + z) = (x + \bar{x})(y + \bar{x})(x + z)(y + z)$$

remember $x + \bar{x} = 1$

$$F = (y + \bar{x})(x + z)(y + z)$$

$$F = (\bar{x} + y + z\bar{z})(x + y\bar{y} + z)(x\bar{x} + y + z)$$

$$F = \underline{(\bar{x} + y + z)}(\bar{x} + y + \bar{z})(x + y + z)(x + \bar{y} + z)\underline{(\bar{x} + y + z)}$$

$$F = (\bar{x} + y + z)(\bar{x} + y + \bar{z})(x + y + z)(x + \bar{y} + z)$$

$$F = M_4 M_5 M_0 M_2$$

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

$$\bar{F}(x, y, z) = \prod(1, 3, 6, 7)$$

Example

Convert the following Boolean expression into standard POS form:

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

Solution

The domain of this POS expression is A, B, C, D. Take one term at a time.

The first term, $\bar{A} + B + C$, is missing variable D or \bar{D} , so add $D\bar{D}$ and apply rule 12 as follows:

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

The second term, $\bar{B} + C + \bar{D}$, is missing variable A or \bar{A} , so add $A\bar{A}$ and apply rule 12 as follows:

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

The third term, $A + B + \bar{C} + \bar{D}$, is already in standard form. The standard POS form of the original expression is as follows:

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

This method is very tedious to work with when the variables and expressions are more in number. Thus next method Karnaugh map method is used to minimize the functions.

Disadvantage of minimization using Boolean theorems:

1. If function has more variables or expressions, it is tedious to minimize it accurately.
2. Redundancy should be analyzed with toughness.
3. Theorems should be remembered for reduction

COURSE MATERIAL

UNIT 1

SEC1207-DIGITAL LOGIC CIRCUITS

SYLLABUS

UNIT I BOOLEAN ALGEBRA AND LOGIC GATES

9 Hrs.

Review of number systems - Binary arithmetic - Binary codes - Boolean algebra and theorems - Boolean functions -Minimization of Boolean functions-Sum of Products(SOP)-Product of Sums(POS)-Simplifications of Boolean functions using Karnaugh map and tabulation methods - Logic gates- NAND and NOR implementation.

TABLE OF TOPICS

S.NO	TOPIC	PAGE NO.
1.1	Review of Number systems	In part 1
1.1.1	Number Systems: Decimal, Binary, Octal, Hexadecimal	In part 1
1.1.2	Conversion from one system to another	In part 1
1.2	Binary Codes	In part 1
1.3	Binary Arithmetic	In part 1
1.4	Boolean Algebra and Theorems	In part 1
1.5	Boolean Functions	In part 1
1.5.1	Minimization of Boolean functions	In part 1
1.5.2	Simplification Using Boolean Functions	46
1.5.2.1	Simplification Using Karnaugh map method	46
1.5.2.2	Simplification Using Tabulation method	66
1.6	Logic gates	74
1.6.1	Universal gates	82
1.6.2	NAND and NOR implementation	84

1.5.2 SIMPLIFICATION OF BOOLEAN FUNCTIONS

1.5.2.1 SIMPLIFICATION OF BOOLEAN FUNCTIONS USING K-MAP

K-map introduction:

K-Map or Karnaugh map is a graphical representation of Boolean logic system directly drawn from minterm (SOP) or maxterm (POS) expressions.

It is used to minimize the SOP or POS expressions in simplified form without altering the output.

Construction of K-map:

A 'n' variable K-map is represented by 2^n squares. Each minterm or maxterm is allotted a square.

- For representing **SOP (minterm)**, if the output of the *minterm* is '1'(True) then the square is represented by '1' else represented by '0'
- For representing **POS (maxterm)**, if the output of the *maxterm* is '0'(False) then the square is represented by '0' else represented by '1'
- For representing **DON'T CARE**, the respective square is marked by **X** both in SOP and POS.

Steps involved in minimizing with K-map:

Step 1: Map representation using number of variables.

Step 2: Plotting the map using truth table, minterm (SOP) function and Maxterm (POS) function.

Step 3: Grouping the function

Step 4: Rewriting the variables in minimized minterm form or minimized maxterm form according to the groups.

1.5.1.2. Two-variable K-map

A '2' variable K-map is represented by 2^2 squares=4 squares. Each minterm or maxterm is allotted a square. [Number of squares in K-Map= $2^{\text{number of variables}}$]

Step 1: Map representation using number of variables:

SOP: If 'F' is a function with two a variables A & B are represented by

$$F(A, B) = \bar{A}\bar{B} + \bar{A}B + A\bar{B} + AB$$

2 variables, $2^2 = 4$ Squares [Number of squares in K-Map= $2^{\text{number of variables}}$]

K-Map is done with 2 rows and 2 columns

Rows hold the variable A and Columns hold the variable B

Row 1= \bar{A}

Row 2= A

Column 1= \bar{B}

Column 2= B

Each square shows minterm in binary form and in variable form.

BINARY PLOTTING (SOP)

F	B	0	1
A		\bar{B}	B
0	\bar{A}	00	10
1	A	10	11

DECIMAL PLOTTING (SOP)

F	B	0	1
A		\bar{B}	B
0	\bar{A}	0	1
1	A	2	3

VARIABLE PLOTTING (SOP)

F	B	0	1
A		\bar{B}	B
0	\bar{A}	$\bar{A}\bar{B}$	$\bar{A}B$
1	A	$A\bar{B}$	AB

MINTERM PLOTTING (SOP)

F	B	0	1
A		\bar{B}	B
0	\bar{A}	m0	m1
1	A	m2	m3

In the above figures,

- we can see that **first one (Binary Plot)** shows, if a function is given as

$$F(A, B) = 00 + 01 + 10 + 11$$

Then the function is plotted in each respective square (discussed in next section)

- we can see that **Second one (Decimal Plot)** shows if a function is given as

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

Then the function is plotted in each respective square (discussed in next section)

- we can see that **Third one (Variable Plot)** shows if a function is given as

$$F(A, B) = \bar{A}\bar{B} + \bar{A}B + A\bar{B} + AB$$

Then the function is plotted in each respective square (discussed in next section)

- we can see that **Fourth one (Minterm Plot)** shows if a function is given as

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

Then the function is plotted in each respective square (discussed in next section)

POS: If 'F' is a function with two variables A & B are represented by

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

$$2 \text{ variables} = 2^2 = 4 \text{ Squares}$$

K-Map is done with 2 rows and 2 columns

Rows hold the variable A and Columns hold the variable B

$$\text{Row 1} = A$$

$$\text{Row 2} = \bar{A}$$

$$\text{Column 1} = B$$

$$\text{Column 2} = \bar{B}$$

Each square shows maxterm in binary form and in variable form.

BINARY PLOTTING (POS)

F	B	1	0
A		B	\bar{B}
1	A	1+1	0+1
0	\bar{A}	0+1	0+0

DECIMAL PLOTTING (POS)

F	B	1	0
A		B	\bar{B}
1	A	0	1
0	\bar{A}	2	3

VARIABLE PLOTTING (POS)

F	B	1	0
A		B	\bar{B}
1	A	$A + B$	$A + \bar{B}$
0	\bar{A}	$\bar{A} + B$	$\bar{A} + \bar{B}$

MAXTERM PLOTTING (POS)

F	B	1	0
A		B	\bar{B}
1	A	M0	M1
0	\bar{A}	M2	M3

In the above figures,

- we can see that **first one (Binary Plot)** shows, if a function is given as

$$F(A, B) = (1+1).(1+0).(0+1).(0+0)$$

Then the function is plotted in each respective square (discussed in next section)

- we can see that **Second one (Decimal Plot)** shows if a function is given as

$$F(A, B) = \Pi M(0, 1, 2, 3)$$

Then the function is plotted in each respective square (discussed in next section)

- we can see that **Third one (Variable Plot)** shows if a function is given as

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

Then the function is plotted in each respective square (discussed in next section)

Step 2: Plotting the variables:

(a) Plotting from simple minterms (SOP):

To understand more, consider the below function:

$$F(A, B) = \bar{A}\bar{B} + \bar{A}B$$

F	B	0	1
A		\bar{B}	B
0	\bar{A}	1	1
1	A	0	0

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

Binary '1' is plotted in the map wherever $\bar{A}\bar{B}$ and $\bar{A}B$ represented (refer last section). Others are represented by binary '0'.

(b) Plotting from minterms (SOP) with don't care:

For another function,

$$F(A, B) = \sum m(0, 1) + d(2)$$

$\sum m(0,1)$ represents sum of products (SOP) or minterms represented by binary '1'. $d(2)$ represents that minterm 2 is a DON'T CARE plotted in the table by using 'X'. others are plotted by using binary '0'.

This is shown in the two-variable K-map below.

$$0 = 00 = \bar{A}\bar{B},$$

$$1 = 01 = \bar{A}B \text{ and}$$

$$2 = 10 = A\bar{B}.$$

F	B	0	1
A		\bar{B}	B
0	\bar{A}	1	1
1	A	X	0

Binary '1' is plotted in the map wherever $\bar{A}\bar{B}$ and $\bar{A}B$ represented. Don't care $A\bar{B}$ is plotted using 'X'. Others are represented by binary '0'.

(c) Plotting from Maxterms (POS) with don't care:

For another function, $F(A, B) = \prod M(3) + d(2)$

$\prod M(3)$ represents products of sum (POS) or Maxterms represented by binary '1'. $d(2)$ represents that Maxterm 2 is a DON'T CARE plotted in the table by using 'X'. Others are plotted by using binary '0'.

This is shown in the two variable K-map below. They are all complements of minterms.

$$3 = 11(\text{minterm}) == +0(\text{Maxterm}) = \bar{A} + \bar{B} \text{ and}$$

$$2 = 10(\text{minterm}) = 0 + 1(\text{Maxterm}) = \bar{A} + B. (\text{DON'T CARE})$$

F	B	1	0
A		B	\bar{B}
1	A	1	1
0	\bar{A}	X	0

Binary '0' is plotted in the map wherever $\bar{A} + \bar{B}$ represented. Don't care $\bar{A} + B$ is plotted using 'X'. Others are represented by binary '1'.

(d) Plotting from truth table:

		INPUT		OUTPUT
		A	B	Y
0	0	0	0	1
1	0	1	1	1
2	1	0	0	X
3	1	1	1	0

minterm – in **BOLD**, *Maxterm* -in italics, Don't Care - X

FROM THE TRUTH TABLE ABOVE, the function using **SOP or minterm** is written as

$$F(A, B) = \sum m(0, 1) + d(2)$$

In the above function,

$\sum m(0,1)$ represents sum of products (SOP) or minterms represented by binary '1'. $d(2)$ represents that minterm 2 is a DON'T CARE plotted in the table by using 'X'. Others are plotted by using binary '0'.

This is shown in the two-variable K-map below.

$$0 = 00 = \bar{A}\bar{B},$$

$$1 = 01 = \bar{A}B \text{ and}$$

$$2 = 10 = A\bar{B}. \text{ (DON'T CARE)}$$

SOP 2-variable K-Map

F	B	0	1
A	\	\bar{B}	B
0	\bar{A}	1	1
1	A	X	0

Binary '1' is plotted in the map wherever $\bar{A}\bar{B}$ and $\bar{A}B$ represented. Don't care $A\bar{B}$ is plotted using 'X'. Others are represented by binary '0'.

FROM THE TRUTH TABLE ABOVE, the function using **POS or maxterm** is written as

$$F(A, B) = \prod M(3) + d(2)$$

In the above function,

$\prod M(3)$ represents sum of products (POS) or maxterms represented by binary '1'. $d(2)$ represents that maxterm 2 is a DON'T CARE plotted in the table by using 'X'. Others are plotted by using binary '0'.

This is shown in the two variable K-map below.

$$2 = 0 + 1 = \bar{A} + B \text{ (DON'T CARE)}$$

$$3 = 0 + 0 = \bar{A} + \bar{B},$$

POS 2-variable K-Map

F	B	1	0
A		B	\bar{B}
1	A	1	1
0	\bar{A}	X	0

Binary '0' is plotted in the map wherever $\bar{A} + \bar{B}$ represented. Don't care $\bar{A} + B$ is plotted using 'X'. Others are represented by binary '1'.

Step 3: Grouping variables for minimization:

Grouping similar Boolean terms is the first step for minimizing the function.

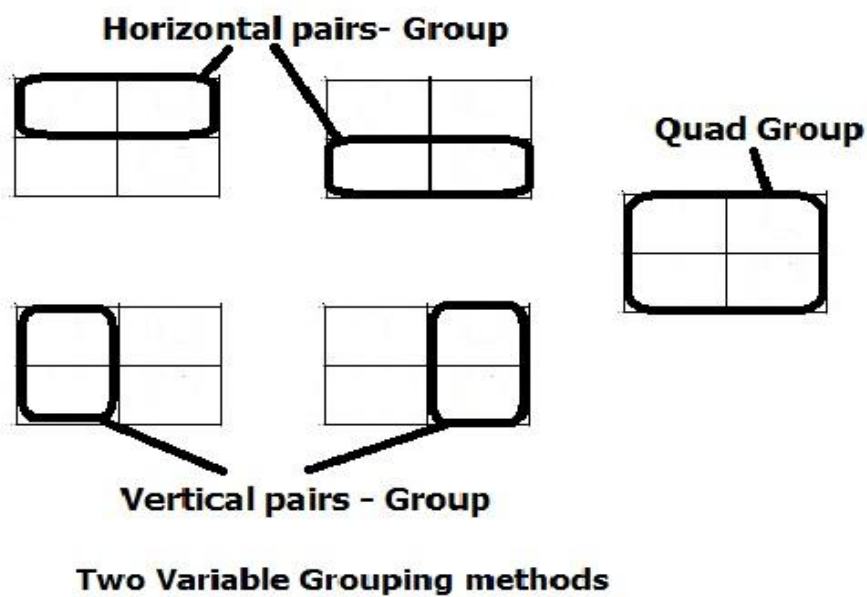


Figure 1.5.1.1 Methods of plotting for two-variables K-Map

The above figure 1.5.1.1, grouping of two adjacent similar variables horizontally and vertically and also grouping all four squares .

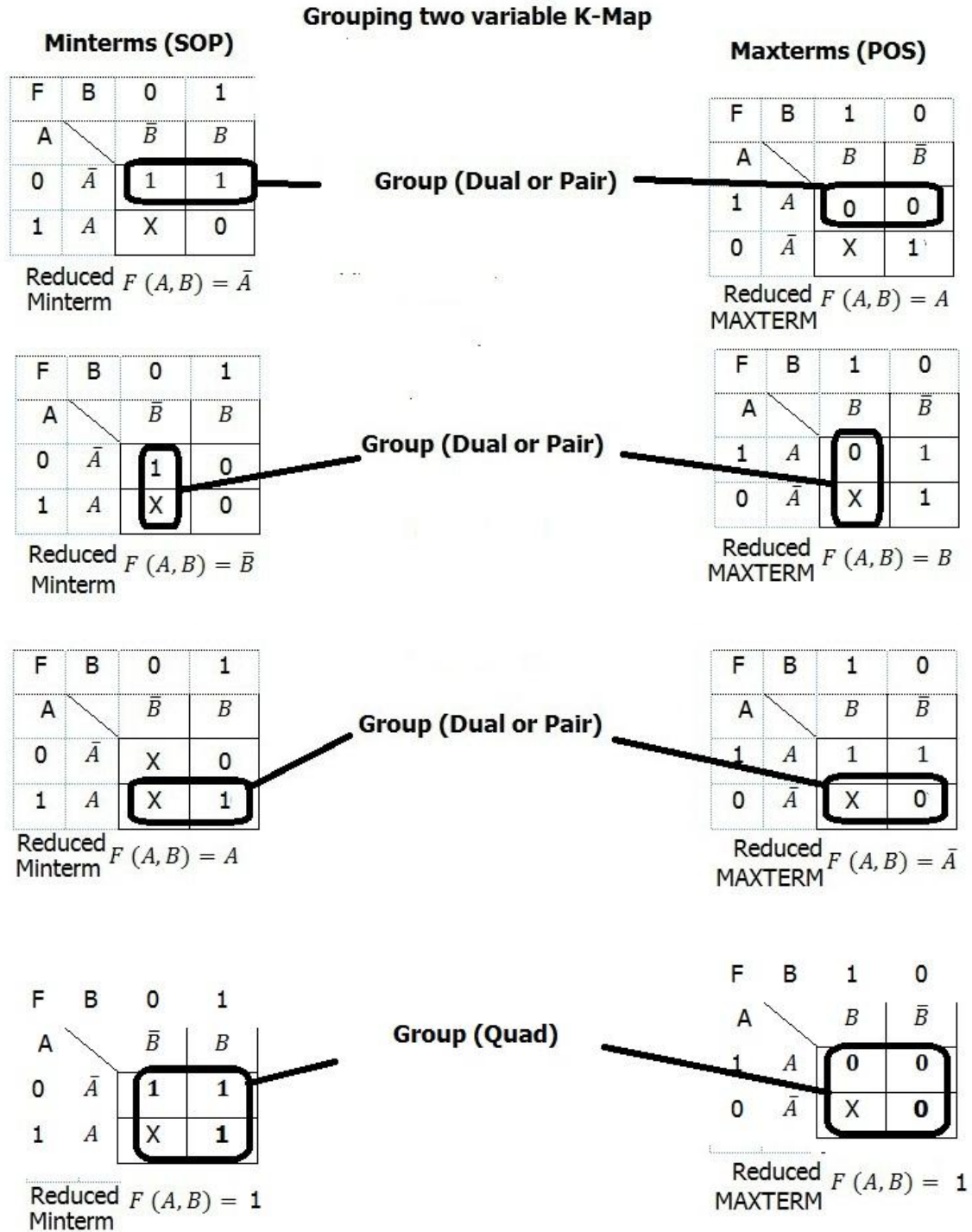


Figure 1.5.1.2 K-Map plot for two-variables

In the above figures, various grouping techniques for two-variable K-map are shown.

Rule 1: A group can be formed by joining adjacent similar terms, whose total is in the order of 2^n where n is any whole number. That is a group can consist of 1 term = 2^0 , 2 terms = 2^1 , 4 terms = 2^2 , 8 terms = 2^3 or 16 terms = 2^4 and so on which are the powers of two.

A group with **2 similar** is called **Pair** or **Dual** (as shown in the above figure 1.5.1.2)

A group with **4 terms** is called **Pair** or **Quad** (as shown in the above figure 1.5.1.2)

A group with **8 terms** is called **Pair** or **Octet (shall be discussed in next session)**

Rule 2: Only similar terms are to be grouped. Like **binary '1'** or **'0'** which are adjacent can be grouped. The group **cannot** contain both **'1'** and **'0'**. The group can contain **don't-care terms** but **at least one term should be '1' or '0'**. (as shown in the above figure 1.5.1.2)

Rule 3: The groups are formed by joining adjacent squares either in horizontal or vertical directions. (**Diagonal direction** of grouping is **not** to be done to use basic gates) as shown in next figure 1.5.1.3 below.

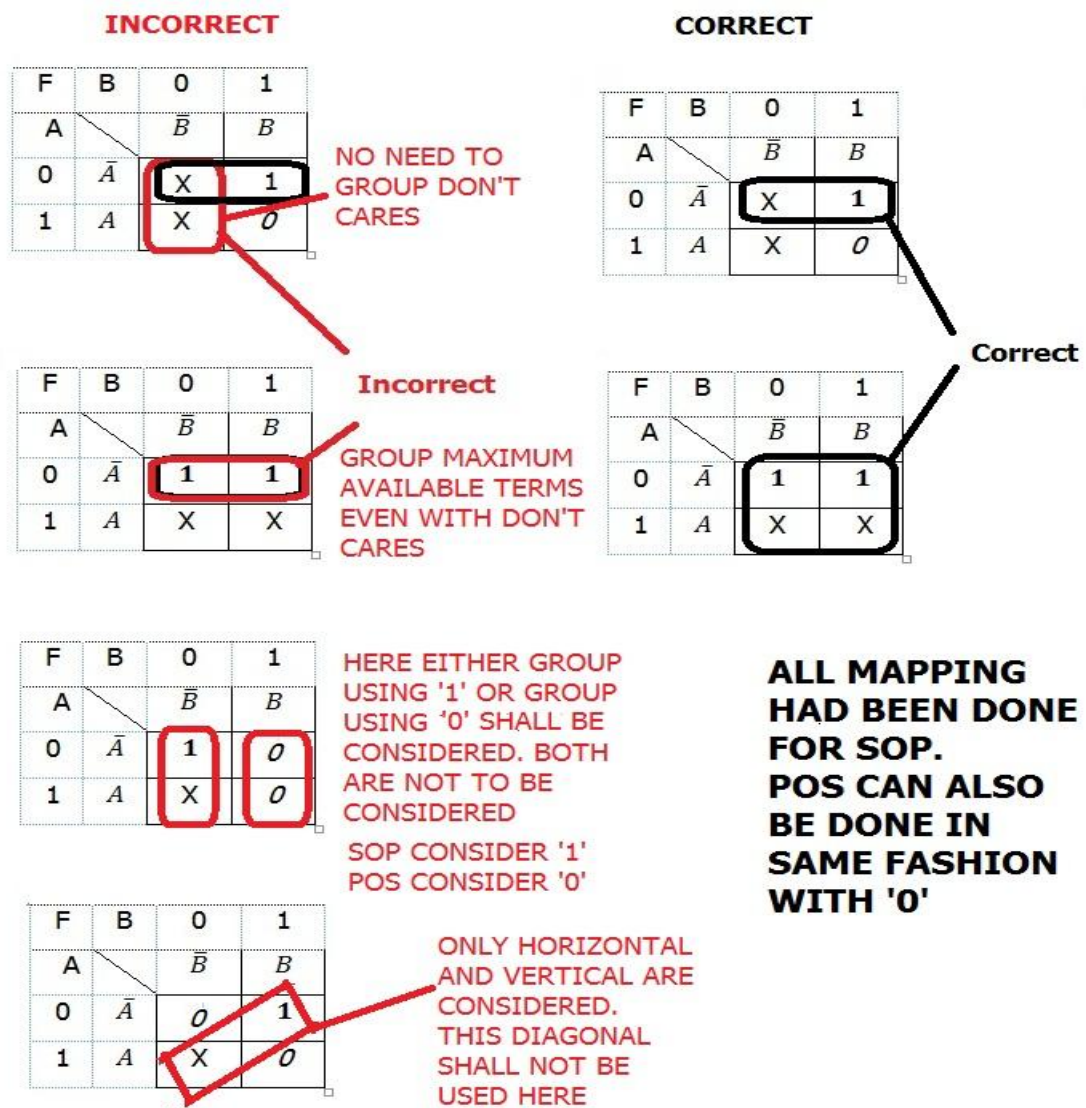


Figure 1.5.1.3 Methods of Grouping in Two-variable K-map.

Step 4: Prime implicants :

In the next below figure 1.5.1.4, a two-variable function is being reduced or minimized by using K-map.

Both the function and truth table show the same expressions. (both are given to make both system understandable).

'0' is a minterm expression but '2' is a don't care expression.

Minterm '0' is $\bar{A}\bar{B}$ and don't-care '2' is $A\bar{B}$ and marked in the K-map.

FUNCTION TO BE REDUCED

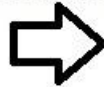
$$F(A, B) = \sum m(0) + d(2)$$

OR

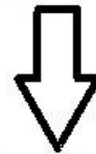
TRUTH TABLE TO BE REDUCED

Input		Output
A	B	F(A, B)
0	0	1
0	1	0
1	0	X
1	1	0

PLOTTING MAP



F	B	0	1
A		\bar{B}	B
0	\bar{A}	1	0
1	A	X	0

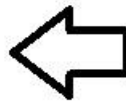


GROUPING

F	B	0	1
A		\bar{B}	B
0	\bar{A}	1	0
1	A	X	0

REDUCED FUNCTION

$$F(A, B) = \bar{B}$$



For the group, vertically \bar{A} and A are uncommon and so it is discarded variable B which is a vertically common identity

Figure 1.5.1.4 Example of Two-variable K-map (SOP).

Then the vertical group is formed between $\bar{A}\bar{B}$ and $A\bar{B}$. They both have common term as \bar{B} and uncommon terms \bar{A} and A. **Uncommon terms shall be discarded (neglected) and common term is only to be taken.** Thus \bar{B} is the only reduced term termed as **Prime implicant**.

Prime implicant is $F(A, B) = \bar{B}$

Thus K-map has reduced two minterms with 2-variables into a single variable. This shows how easy and efficient this method is, than minimizing using Boolean theorems.

In The below Figure 1.5.1.5, K-map of two-variables A and B using POS terms is shown. Same method but grouping is done using '0'.

FUNCTION TO BE REDUCED (POS)

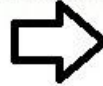
$$F(A, B) = \prod M(1, 3) + d(2)$$

OR

TRUTH TABLE TO BE REDUCED

Input		Output
A	B	F(A, B)
0	0	1
0	1	0
1	0	X
1	1	0

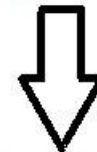
PLOTTING MAP



F	B	1	0
A		B	\bar{B}
1	A	1	0
0	\bar{A}	X	0

$A + \bar{B}$

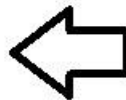
$\bar{A} + \bar{B}$



GROUPING

REDUCED FUNCTION

$$F(A, B) = \bar{B}$$



F	B	1	0
A		B	\bar{B}
1	A	1	0
0	\bar{A}	X	0

\bar{B}

For the group, vertically \bar{A} and A are uncommon and so it is discarded
variable B which is a vertically common identity

Figure 1.5.1.5 Example of Two-variable K-map (POS).

Output is same for POS and SOP as \bar{B} and thus in two-variable K-map, grouping using '1' by SOP or grouping using '0' in POS is the same.

Prime implicant is $F(A, B) = \bar{B}$

1.5.1.3. Three-variable K-map

A '3'-variable K-map is represented by 2^3 squares=8 squares. Each minterm or maxterm is allotted a square.

Here K-Map plotting for 3-variables is shown in step-by-step.

Step 1: since 3-variables are available 2^3 squares are needed.

3 variables, so $2^3 = 8$ Squares [Number of squares in K-Map= $2^{\text{number of variables}}$]

Map can be drawn as shown in the below four figures from 1.5.1.2.a to d.

- Figure 1.5.1.2.a shows mapping with 2 rows and 4 columns plotted with binary number equivalent to variable.

BINARY PLOTTING

F	BC	00	01	11	10
A		$\overline{B}\overline{C}$	$\overline{B}C$	BC	$B\overline{C}$
0	\overline{A}	000	001	011	010
1	A	100	101	111	110

Rows = 2
Columns = 4
Rows x columns = 8 squares

Row 1:	\overline{A}	0	Column 1:	$\overline{B}\overline{C}$	00
Row 2:	A	1	Column 2:	$\overline{B}C$	01
			Column 3:	BC	11
			Column 4:	$B\overline{C}$	10

Fig.1.5.1.2.a. Binary Plot- method 1

DECIMAL EQUIVALENT

F	BC	00	01	11	10
A		$\overline{B}\overline{C}$	$\overline{B}C$	BC	$B\overline{C}$
0	\overline{A}	0	1	3	2
1	A	4	5	7	6

Rows = 2
Columns = 4
Rows x columns = 8 squares

Columns 3 & 4 (shown in Bold) are written in GRAY code. Even numbers 2,3 and 6,7 are written in reverse. (Discussed below)

Fig.1.5.1.2.b. Decimal Plot- method 1

F	C	0	1
AB		\overline{C}	C
00	$\overline{A}\overline{B}$	000	001
01	$\overline{A}B$	010	011
11	AB	111	110
10	$A\overline{B}$	100	101

Rows = 4
Columns = 2
Rows x columns = 8 squares

Column 1:	$\overline{A}\overline{B}$	00	Row 1:	\overline{C}	0
Column 2:	$\overline{A}B$	01	Row 2:	C	1
Column 3:	AB	11			
Column 4:	$A\overline{B}$	10			

Fig.1.5.1.2.c. Binary Plot- method 2

F	C	0	1
AB		\overline{C}	C
00	$\overline{A}\overline{B}$	0	1
01	$\overline{A}B$	2	3
11	AB	7	6
10	$A\overline{B}$	4	5

Rows = 4
Columns = 2
Rows x columns = 8 squares

Rows 3 & 4 (shown in Bold) are written in GRAY code. Even numbers 4,5 and 6,7 are written in reverse. (Discussed below)

Fig.1.5.1.2.d. Decimal Plot- method 2

Here as shown in the above figure 1.5.1.2, different plotting is shown. When 4-rows or 4-columns are marked with **GRAY code**. 00(0), 01(1), 11(3) and 10(2) NOT binary order 00(0), 01(1), 10(2), and 11(3).

Why GRAY CODE is considered?

Gray code has only one transition **00→01, 01→11, 11→10, 10→00** (shown in bold) [refer gray code section]

But binary code has multi-transition **00→01, 01→10, 10→11, 11→00** (shown in bold-italics).

Since only one transition is there in the order of Gray code, 00 01, 11 and 10, it is used while writing for four rows or columns.

So the columns 3 & 4 in figure 1.5.1.2.b. and rows 3 & 4 in figure.1.5.1.2.d are reversed.

NOTE: Either (4 rows and 2 columns) OR (2 rows and 4 columns) map is considered. Both methods yield the same minimized output function. We shall discuss 2 rows and 4 column map.

VARIABLE PLOTTING

F	BC	00	01	11	10
A		$\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
0	\bar{A}	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$\bar{A}BC$	$\bar{A}B\bar{C}$
1	A	$A\bar{B}\bar{C}$	$A\bar{B}C$	ABC	$AB\bar{C}$

F	BC	1+1	1+0	0+0	0+1
A		$B+C$	$B+\bar{C}$	$\bar{B}+\bar{C}$	$\bar{B}+C$
1	A	$A+B+C$	$A+B+\bar{C}$	$A+\bar{B}+\bar{C}$	$A+\bar{B}+C$
0	\bar{A}	$\bar{A}+B+C$	$\bar{A}+B+\bar{C}$	$\bar{A}+\bar{B}+\bar{C}$	$\bar{A}+\bar{B}+C$

SOP PLOTTING
2 Rows and 4 Columns

F	C	0	1
AB		\bar{C}	C
00	$\bar{A}\bar{B}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$
01	$\bar{A}B$	$\bar{A}B\bar{C}$	$\bar{A}BC$
11	AB	$AB\bar{C}$	ABC
10	$A\bar{B}$	$A\bar{B}\bar{C}$	$A\bar{B}C$

POS PLOTTING
2 Rows and 4 Columns

F	C	1	0
AB		C	\bar{C}
1+1	$A+B$	$A+B+C$	$A+B+\bar{C}$
1+0	$A+\bar{B}$	$A+\bar{B}+C$	$A+\bar{B}+\bar{C}$
0+0	$\bar{A}+\bar{B}$	$\bar{A}+\bar{B}+C$	$\bar{A}+\bar{B}+\bar{C}$
0+1	$\bar{A}+B$	$\bar{A}+B+C$	$\bar{A}+B+\bar{C}$

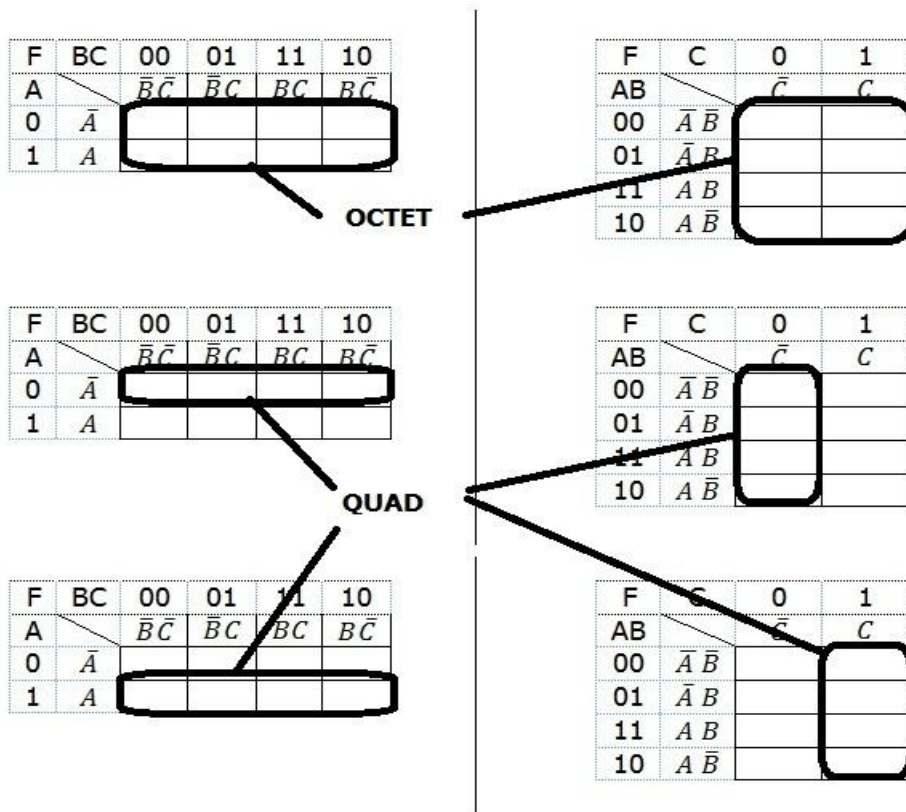
SOP PLOTTING
4 Rows and 2 Columns

POS PLOTTING
4 Rows and 2 Columns

Fig.1.5.1.3. Variable Plotting – SOP & POS

The above figure 1.5.1.3 shows different plotting using SOP or POS forms.

The figure BELOW SHOW different GROUPING in 3-variable K-map.



SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

F	BC	00	01	11	10
A		$\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
0	\bar{A}				
1	A				

F	C	0	1
AB		\bar{C}	C
00	$\bar{A}\bar{B}$		
01	$\bar{A}B$		
11	AB		
10	$A\bar{B}$		

QUADS

F	BC	00	01	11	10
A		$\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
0	\bar{A}				
1	A				

F	C	0	1
AB		\bar{C}	C
00	$\bar{A}\bar{B}$		
01	$\bar{A}B$		
11	AB		
10	$A\bar{B}$		

F	BC	00	01	11	10
A		$\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
0	\bar{A}				
1	A				

F	C	0	1
AB		\bar{C}	C
00	$\bar{A}\bar{B}$		
01	$\bar{A}B$		
11	AB		
10	$A\bar{B}$		

**FOLDING
QUAD**

**FOLDING
QUAD**

F	BC	00	01	11	10
A		$\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
0	\bar{A}				
1	A				

F	C	0	1
AB		\bar{C}	C
00	$\bar{A}\bar{B}$		
01	$\bar{A}B$		
11	AB		
10	$A\bar{B}$		

PAIRS

QUAD

F	BC	00	01	11	10
A		$\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
0	\bar{A}				
1	A				

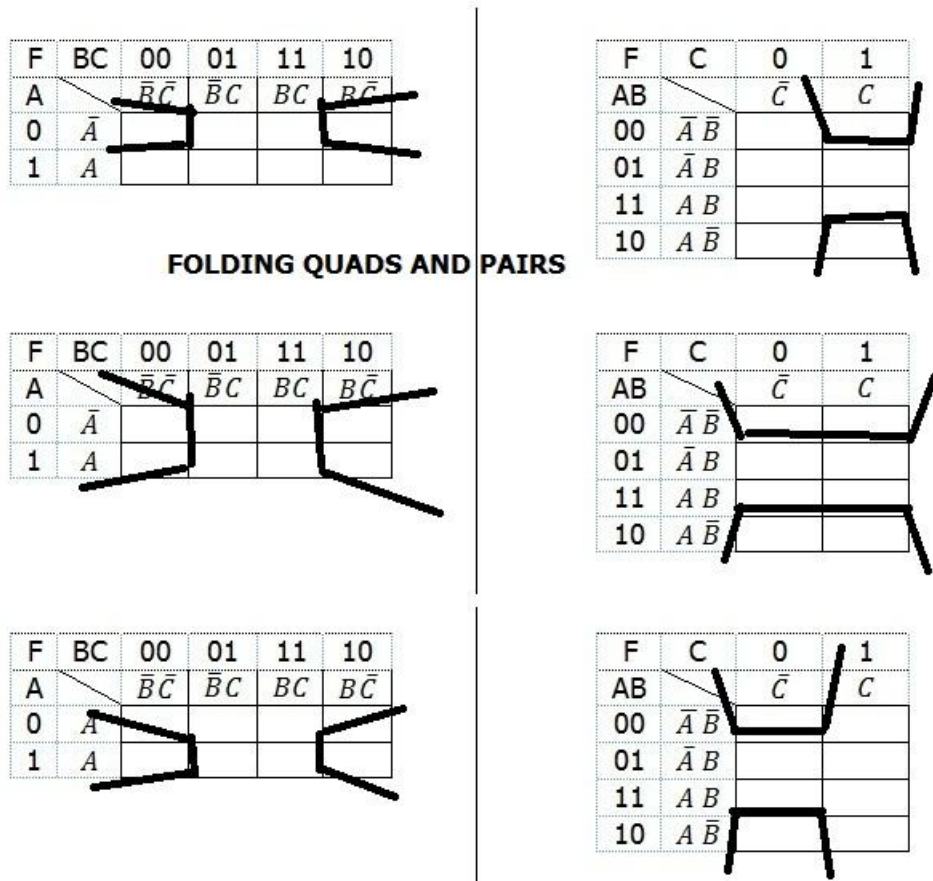
F	C	0	1
AB		\bar{C}	C
00	$\bar{A}\bar{B}$		
01	$\bar{A}B$		
11	AB		
10	$A\bar{B}$		

QUADS

F	BC	00	01	11	10
A		$\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
0	\bar{A}				
1	A				

F	C	0	1
AB		\bar{C}	C
00	$\bar{A}\bar{B}$		
01	$\bar{A}B$		
11	AB		
10	$A\bar{B}$		

QUADS



The technique called folding is shown in the above FIGURES for grouping quads and pairs in 3-variable K-map. This shall be explained while using it in a sample 3-variable K-map.

The figure below 1.5.1.4 shows how a 3- VARIABLE truth table is plotted in K-map (SOP)

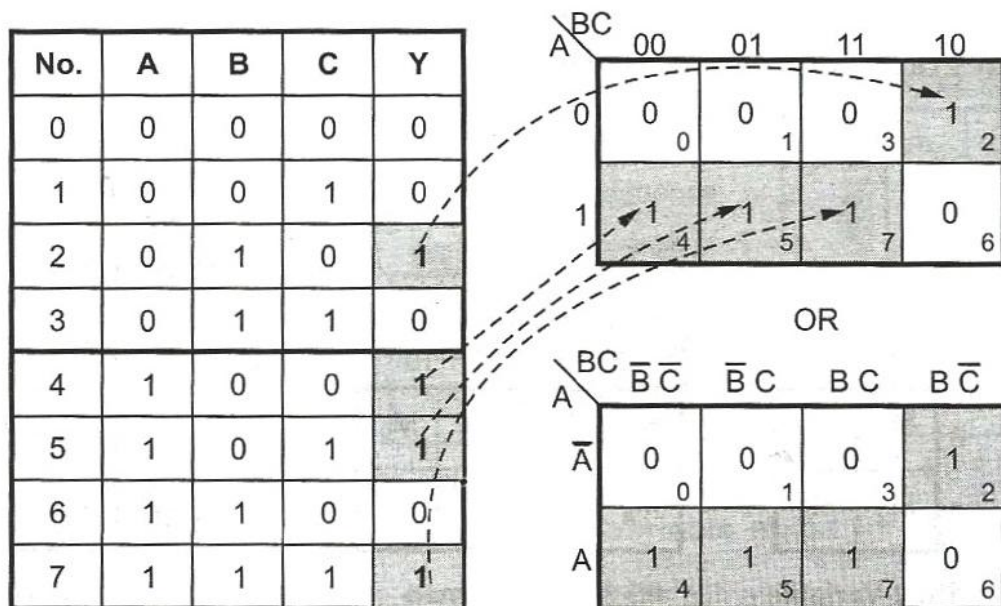


Fig.1.5.1.4. Plotting 3-variable K-Map from Truth table (SOP)

Example: Plot the Boolean FUNCTION $F(A, B, C) = \sum m(1, 6, 7)$ or EXPRESSION $F(A, B, C) = \bar{A}\bar{B}C + A\bar{B}\bar{C} + ABC$

Solution:

The above is plotted in the figure.1.5.1.5 (SOP)

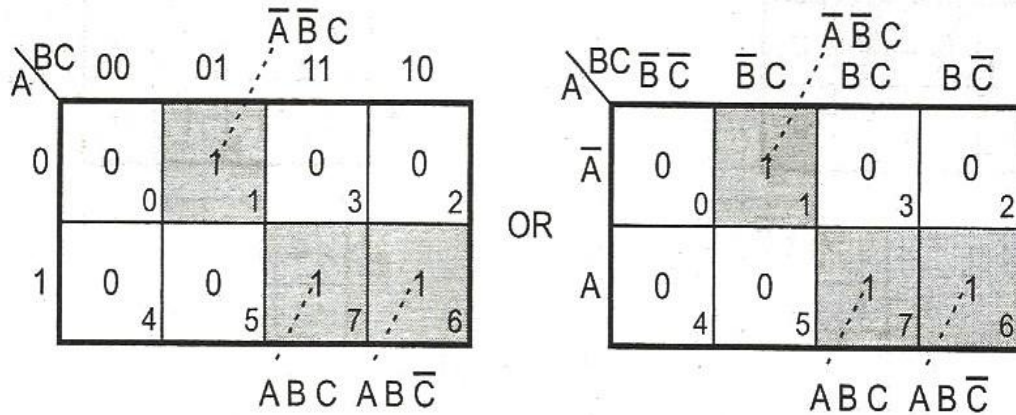


Fig.1.5.1.5. Plotting 3–variable K-Map from function (SOP)

Example: Plot the Boolean FUNCTION $F(A, B, C) = \prod M(1, 2, 3, 6)$ or EXPRESSION $F(A, B, C) = (A + B + \bar{C})(A + \bar{B} + C)(A + \bar{B} + \bar{C})(\bar{A} + \bar{B} + C)$

Solution:

The above is plotted in the figure.1.5.1.6 (POS)

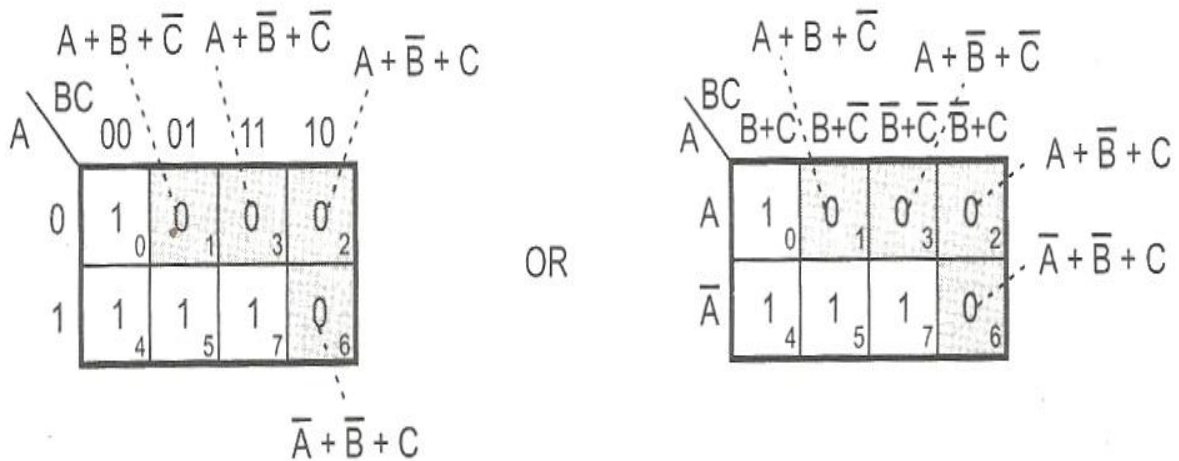


Fig.1.5.1.6. Plotting 3–variable K-Map from function (POS)

Example: Group and reduce the Boolean FUNCTION $F(A, B, C) = \sum m(0, 1, 3, 7) + d(2, 5)$

Solution:

The above is plotted, grouped and reduced in the figure.1.5.1.7 (SOP WITH DON'T CARE)

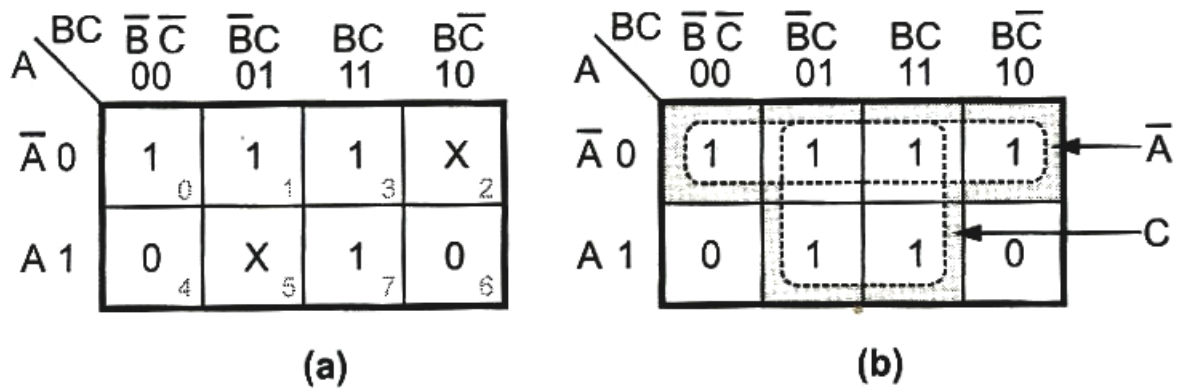


Fig.1.5.1.7. Grouping 3–variable K-Map from function (SOP with Don't care)

Figure 1.5.1.7.a shows, don't care conditions X mark in the squares representing minterms 2 and 5. Since the don't care falls inside the grouping of Quads shown in figure 1.5.1.7.b it is considered as binary 1.

The reduced SOP function is $F(A, B, C) = \bar{A} + C$

Example: Using K-map minimize the Boolean FUNCTION $F(A, B, C) = \sum m(0, 1, 3, 4, 5)$

Solution:

The above is plotted, grouped and reduced in the table.1.5.1.1 (SOP)

Table 1.5.1.1. Minimizing 3–variable K-Map from function (SOP)

<p style="text-align: center;">Step 1</p> <p>Plot the minterms as shown in the beside figure (a) <i>0, 1, 3, 4, and 5 with '1' and other squares with binary '0'</i></p>	<p style="text-align: center;">(a)</p>
--	--

<p style="text-align: center;">Step 2</p> <p>Isolated 1's are not available.</p> <p>Squares or Cells 1 & 3 are grouped together as DUAL as shown in the figure (b) beside. Here variable \bar{A} is common. Then between 2nd and 3rd column, variable C is common but \bar{B} & B is uncommon. Thus $\bar{A}C$ is taken and variable B is omitted.</p>	<p style="text-align: center;">(b)</p>
<p style="text-align: center;">Step 3</p> <p>Squares or Cells 0, 1, 4 and 5 are grouped together as QUAD as shown in the figure (c) beside. Here variable A is uncommon, C is uncommon but \bar{B} is common. Thus \bar{B} is only taken.</p> <p>Thus the reduced minterm is shown in the figure as $F(A, B, C) = \bar{A}C + \bar{B}$. This is done by omitting the uncommon variables.</p>	<p style="text-align: center;">(c)</p>

Thus the minimized function is $F(A, B, C) = \bar{A}C + \bar{B}$.

Example: Using K-map minimize the Boolean FUNCTION $F(A, B, C) = \prod M(0, 1, 3, 4, 7)$

Solution:

The above is plotted, grouped and reduced in the table.1.5.1.2 (POS)

Table 1.5.1.2. Minimizing 3–variable K-Map from function (POS)

<p style="text-align: center;">Step 1</p> <p>Plot the maxterms as shown in the beside figure (a)</p> <p><i>0, 1, 3, 4, and 7 with '0'.</i></p>	<p style="text-align: center;">(a)</p>
---	---

<p>Step 2 Isolated 1's are not available.</p> <p>Squares or Cells 0 & 4 and also 3 & 7 are grouped together as DUALS as shown in the figure (b) beside. Here variable $B + C$ (1 & 4) and $\bar{B} + \bar{C}$ (3 & 7) are common. Thus $B + C$ and $\bar{B} + \bar{C}$ are prime implicants.</p>	<p style="text-align: center;">(b)</p>
<p>Step 3 Squares or Cells 1 & 3 are grouped together as DUAL as shown in the figure (c) beside. Here variable A and \bar{C} is common. Thus $A + \bar{C}$ is common term.</p> <p>Thus the reduced maxterm is shown in the figure as $F(A, B, C) = (B + C)(\bar{B} + \bar{C})(A + \bar{C})$. This is done by omitting the uncommon variables.</p>	<p style="text-align: center;">(c)</p>

Thus the minimized function is $F(A, B, C) = (B + C)(\bar{B} + \bar{C})(A + \bar{C})$

1.5.1.4. Four variable K-map

A '4' variable K-map is represented by 2^4 squares=16 squares. Each minterm or maxterm is allotted a square.

SOP 4-variable mapping is shown below.

Y	CD	00	01	11	10
AB	$\bar{C}\bar{D}$	$\bar{C}D$	$C\bar{D}$	CD	$\bar{C}\bar{D}$
00	$\bar{A}\bar{B}$	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}\bar{B}C\bar{D}$	$\bar{A}\bar{B}CD$
01	$\bar{A}B$	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}B\bar{C}D$	$\bar{A}BC\bar{D}$	$\bar{A}BCD$
11	AB	$AB\bar{C}\bar{D}$	$AB\bar{C}D$	$ABC\bar{D}$	$ABCD$
10	$A\bar{B}$	$A\bar{B}\bar{C}\bar{D}$	$A\bar{B}\bar{C}D$	$A\bar{B}C\bar{D}$	$A\bar{B}CD$

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

POS 4-variable mapping is shown below.

Y	CD	1+1 $C + D$	1+0 $C + \bar{D}$	0+0 $\bar{C} + \bar{D}$	0+1 $\bar{C} + D$
1+1 $A + B$		$A + B + C + D$	$A + B + C + \bar{D}$	$A + B + \bar{C} + \bar{D}$	$A + B + \bar{C} + D$
1+0 $A + \bar{B}$		$A + \bar{B} + C + D$	$A + \bar{B} + C + \bar{D}$	$A + \bar{B} + \bar{C} + \bar{D}$	$A + \bar{B} + \bar{C} + D$
0+0 $\bar{A} + \bar{B}$		$\bar{A} + \bar{B} + C + D$	$\bar{A} + \bar{B} + C + \bar{D}$	$\bar{A} + \bar{B} + \bar{C} + \bar{D}$	$\bar{A} + \bar{B} + \bar{C} + D$
0+1 $\bar{A} + B$		$\bar{A} + B + C + D$	$\bar{A} + B + C + \bar{D}$	$\bar{A} + B + \bar{C} + \bar{D}$	$\bar{A} + B + \bar{C} + D$

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

Same way as 3-variable mapping, 4-variable mapping is also done.

Plotting of 4-variable K-map:

SOP form:

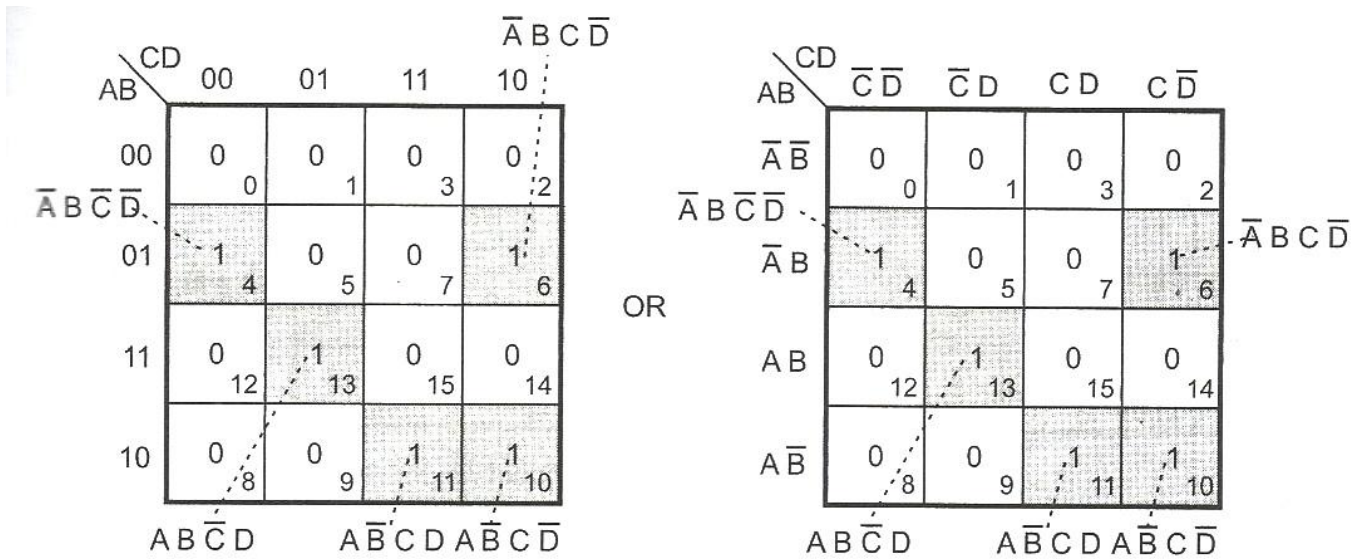


Figure 1.5.1.4.1. plotting 4-variable K-map.(SOP)

An example SOP:

Minimize the function using K-map.

$$F(A, B, C, D) = \sum m(2, 4, 5, 9, 12, 13)$$

Solution:

Below in figure 1.5.1.4.2.a, the minterms 2, 4, 5, 9, 12 and 13 are marked as '1' in their respective cells. Others are marked '0'

Below in figure 1.5.1.4.2.b, the minterms 2 is found isolated and so it is encircled. The term is not minimized and is $\bar{A}\bar{B}C\bar{D}$

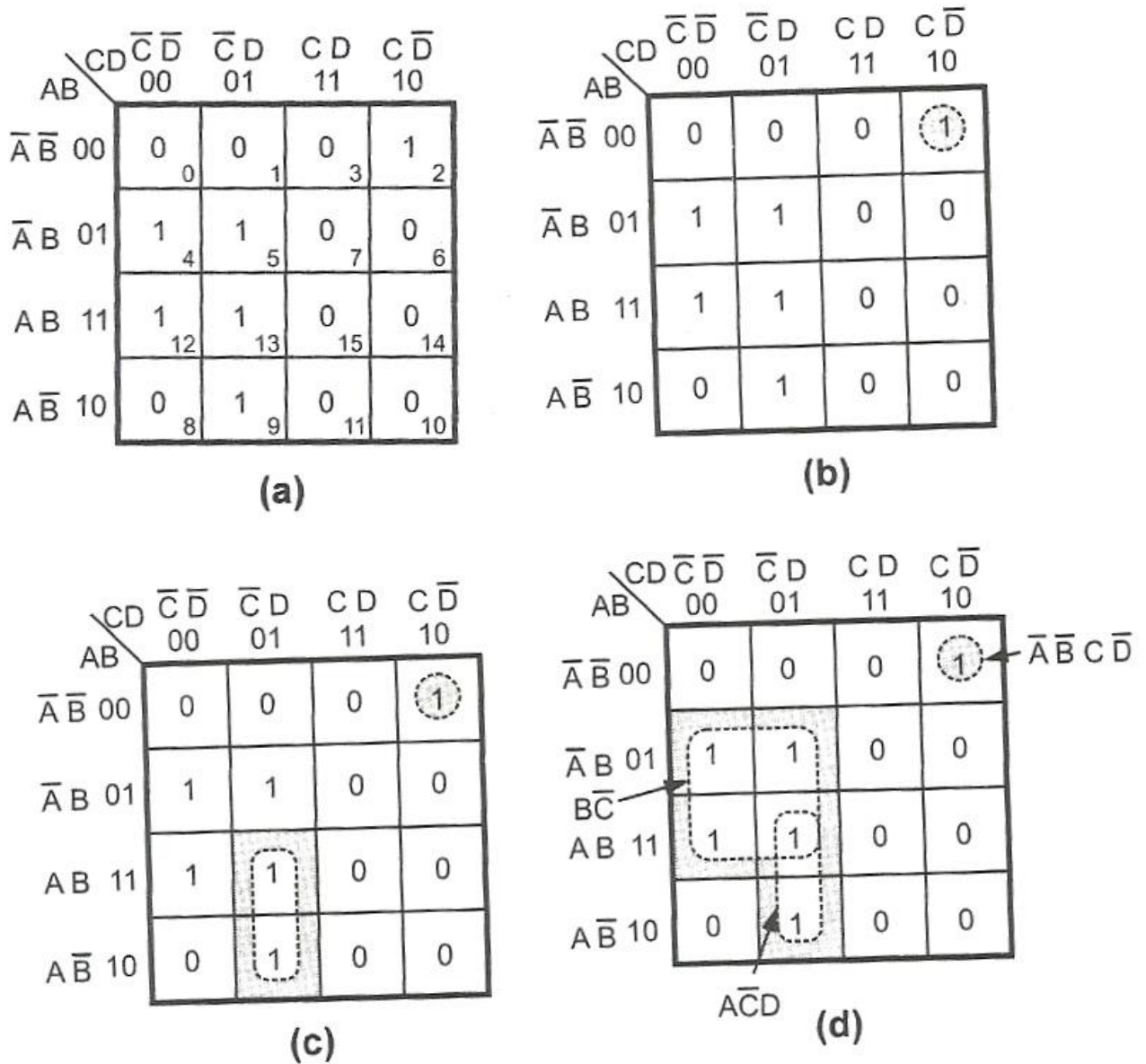


Figure 1.5.1.4.2. Minimizing a 4-variable K-map. (SOP)

Above in figure 1.5.1.4.2.c, the minterms 9 & 13 are grouped as DUAL (PAIR). The term is minimized as $A\bar{C}D$.

Above in figure 1.5.1.4.2.d, the minterms 4, 5, 12 and 13 are grouped as QUAD. The term is minimized as $B\bar{C}$.

Thus the final minimized PRIME IMPLICANTS are $F = \bar{A}\bar{B}C\bar{D} + A\bar{C}D + B\bar{C}$

An example SOP:

Minimize the function using K-map.

$$F(A, B, C, D) = \sum m(0, 1, 2, 4, 5, 6, 8, 9, 10, 12, 13)$$

Solution:

Below in figure 1.5.1.4.3, the minterms are marked as '1' in their respective cells.

Then they are grouped as shown in the same figure.

The minimized function is

$$F(A, B, C, D) = A\bar{B}\bar{D} + \bar{A}\bar{D} + \bar{C}$$

It can also be

$$F(A, B, C, D) = \bar{B}\bar{D} + \bar{A}\bar{D} + \bar{C}$$

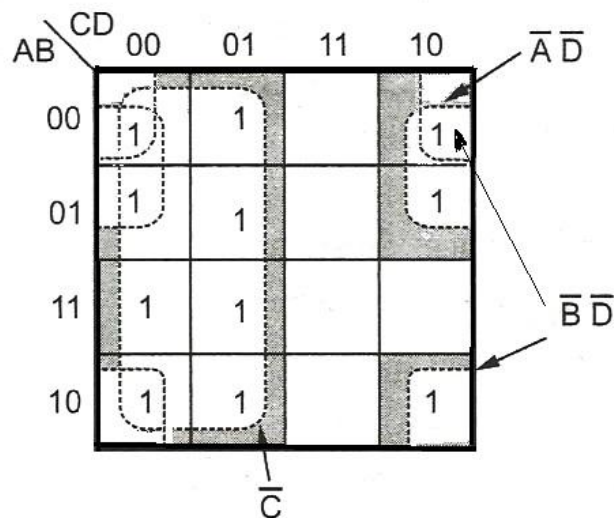


Figure 2.5.1.4.3. Grouping & Minimizing a 4-variable K-map. (SOP)

Limitations of K-map:

1. As the number of variables increases the method becomes complex.
2. Simplification depends upon human abilities.

1.5.2.2 SIMPLIFICATION OF BOOLEAN FUNCTIONS USING TABULATION METHOD

This method of minimization is also called Quine-Mcclusky method.

Example: Solve the function $F(A, B, C, D) = \sum m(0, 1, 3, 5, 7, 9, 11, 13, 15) + d(2, 6)$ using tabulation method

Step 1: Since the function has four variables A, B, C and D, each minterm is to be written with its equivalent binary number. As shown in the table below, three columns are shown. First column is

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

about the given minterms, second columns their equivalent binary of given variables and third column is of the number of 1's in the binary equivalent.

Note: *Second column refers to the binary equivalent with word length equal to number of variables.*

Here it is a four variable function, so 4-bit equivalent is written.

Minterm	Four bit equivalent	Number of 1's
m0	0000	0
m1	0001	1
m3	0011	2
m5	0101	2
m7	0111	3
m9	1001	2
m11	1011	3
m13	1101	3
m15	1111	4
m2	0010	1
m6	0110	2

Step 2: Sort the table in ascending order according to number of 1's in it.

Minterm	Four bit equivalent	Number of 1's
m0	0000	0
m1	0001	1
m2	0010	1
m3	0011	2
m5	0101	2
M6	0110	2
M9	1001	2
m7	0111	3
m11	1011	3
m13	1101	3
m15	1111	4

Step 3: After sorting group it accordingly to the number of '1's given in it.

Group 0 represents no '1's (minterm m0),

Group 1 represents variables with one '1' (minterms m1, m2),

Group 2 represents variables with two '1'(minterms m3, m5, m6, m9),

Group 3 represents variables with three '1' (minterms m7, m11, m13) and

Group 4 represents variables with four '1' (minterms m15).

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

Reduction 1						
Group	Minterm	4-bit Binary Equivalent				Minimized
		A	B	C	D	
0	m0	0	0	0	0	☑
1	m1	0	0	0	1	☑
	m2	0	0	1	0	☑
2	m3	0	0	1	1	☑
	m5	0	1	0	1	☑
	m6	0	1	1	0	☑
	m9	1	0	0	1	☑
3	m7	0	1	1	1	☑
	m11	1	0	1	1	☑
	m13	1	1	0	1	☑
4	m15	1	1	1	1	☑

Step 4: Compare adjacent groups only for minimization.

That is considering groups 0 and 1.

- Check between elements of group 0 and 1, whether one transition (change) is there in the binary equivalent of the variables. If so mark the change as dash(-).
 - Example: As we can consider, reduction 1 table above, the elements m0(0000) and m1(0001) have one transition.

Group	Minterm	Variables			
		A	B	C	D
Group 0	m0	0	0	0	0
Group 1	m1	0	0	0	1
Reduced Group	m(0,1)	0	0	0	-

As shown in the table column representing variable D, (highlighted) has uncommon binary numbers. It is supposed to have transition. Other variables A, B, and C are common. Thus in **reduced group (5)** represented in below table becomes minimized and represented by – (dash)

- Note: only adjacent groups 0 & 1, 1 & 2, 2 & 3 and 3 & 4 can be reduced. **Never** attempt to reduce using group 2 & 4, because they may more than one change or transition.
- The reduced group is marked as minimized in the above table. Suppose m0 and m1 are reduce in next table, they are marked as reduced by using ☑ in the column minimized.

Reduction 2						
Group	Minterm	4-bit Binary Equivalent				Minimized
		A	B	C	D	

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

0 & 1 (5)	m(0, 1)	0	0	0	-	✓
	m(0, 2)	0	0	-	0	✓
1 & 2 (6)	m(1, 3)	0	0	-	1	✓
	m(1, 5)	0	-	0	1	✓
	m(1, 9)	-	0	0	1	✓
	m(2, 3)	0	0	1	-	✓
	m(2, 6)	0	-	1	0	✓
2 & 3 (7)	m(3, 7)	0	-	1	1	✓
	m(3, 11)	-	0	1	1	✓
	m(5, 7)	0	1	-	1	✓
	m(5, 13)	-	1	0	1	✓
	m(6, 7)	0	1	1	-	✓
	m(9, 11)	1	0	-	1	✓
	m(9, 13)	1	-	0	1	✓
3 & 4 (8)	m(7, 15)	-	1	1	1	✓
	m(11, 15)	1	-	1	1	✓
	m(13, 15)	1	1	-	1	✓

Reduction 3						
Group	Minterm	4-bit Binary Equivalent				Minimized
		A	B	C	D	
5 & 6 (9)	m(0, 2, 1, 3)	0	0	-	-	same
	m(0, 1, 2, 3)	0	0	-	-	
6 & 7 (10)	m(1, 3, 5, 7)	0	-	-	1	Same ✓
	m(1, 5, 3, 7)	0	-	-	1	
	m(1, 3, 9, 11)	-	0	-	1	Same ✓
	m(1, 9, 3, 11)	-	0	-	1	
	m(1, 5, 9, 13)	-	-	0	1	Same ✓
	m(1, 9, 5, 13)	-	-	0	1	
	m(2, 3, 6, 7)	0	-	1	-	same
	m(2, 6, 3, 7)	0	-	1	-	
	m(2, 6, 9, 13)	-	-	1	0	✓
7 & 8 (11)	m(3, 7, 11, 15)	-	-	1	1	Same ✓
	m(3, 11, 7, 15)	-	-	1	1	
	m(5, 7, 13, 15)	-	1	-	1	Same ✓
	m(5, 13, 7, 15)	-	1	-	1	
	m(9, 11, 13, 15)	1	-	-	1	Same ✓
	m(9, 13, 11, 15)	1	-	-	1	

Reduction 4						
Group	Minterm	4-bit Binary Equivalent				Minimized
		A	B	C	D	
10 & 11	m(1, 3, 5, 7, 9, 13, 11, 15)	-	-	-	1	Same
	m(1, 3, 9, 11, 5, 7, 13, 15)	-	-	-	1	
	m(1, 9, 5, 13, 3, 11, 7, 15)	-	-	-	1	
	m(2, 6, 9, 13, 3, 7, 11, 15)	-	-	1	-	

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

Thus this process of reduction table goes on till reduction cannot be done.

Step 5: Now prime implicants are selected from all the reduction tables. The minterms which are not minimized (unmarked) are written in the below table of prime implicants.

The **prime implicants** are the **bold** ones in the above tables. There are four (4) prime implicants.

Row 1	0, 2, 1, 3	00-- = A'B'
Row 2	2, 6, 3, 7	0-1- = A'C
Row 3	1, 3, 9, 11, 5, 7, 13, 15	---1 = D
Row 4	2, 6, 9, 13, 3, 7, 11, 15	--1- = C

How to convert binary representation to Boolean variables?

- Consider the **Row 2** for an example:
 - **Row 2 :**
 - if binary is '0' corresponding variable is represented in its complement form **A' or B' or C' or D' etc.**
 - if binary is '1' corresponding variable is represented in un-complemented form **A or B or C or D etc.**
 - if binary is not available '-' then it is not represented as shown in below table.
 - In the table A is represented in complement form A' because binary equivalent of A is '0'.
 - And C is represented in true form C because binary equivalent of C is '1'.
 - Others B and D are not represented because they are minimized (-)
 - This expression is **A'C**

Variables	A	B	C	D
Binary	0	-	1	-
Variable Expression	A'	-	C	-
	A'C			

- Like this all rows were done and the below stated function is derived

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

Step 6: Then essential prime implicants should be chosen.

- List the minterms in rows and their respective Boolean variable expressions.
- In columns write all the minterm numbers given in the function (the function was $F(A, B, C, D) = \sum m(0, 1, 3, 5, 7, 9, 11, 13, 15) + d(2, 6)$, except the don't cares. Only highlighted ones shall be written.
- Now mark the respective columns in which the corresponding minterms are found. For example in the below table in first row the minterm is 0,1,2,3. The columns with the

SATHYABAMA UNIVERSITY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGG.
COURSE MATERIAL – SEC1207 – DIGITAL LOGIC CIRCUITS – UNIT 1

numbers 0, 1, 3 are marked as shown (but 2 is not marked because it is a DON'T care variable)

- Whichever column is having single mark is encircled or highlighted as shown in the figure. Here column representing number **0** and **5** are encircled or highlighted.
- The corresponding row is highlighted as essential as shown in essential column in the below table.
- Check for that the essential prime implicants cover all the columns. If covered, the other minterms are **not** to be considered as **essential prime implicants**. Here both the rows cover all the columns. Thus essential prime implicants are marked with .

Essential Prime implicants:

Essential	Minterm	0	1	3	5	7	9	11	13	15
<input checked="" type="checkbox"/>	0, 2, 1, 3 (A'B')	<input checked="" type="checkbox"/>	x	x						
	2, 6, 3, 7 (A'C)			x		x				
<input checked="" type="checkbox"/>	1, 3, 9, 11, 5, 7, 13, 15 (D)		x	x	<input checked="" type="checkbox"/>	x	x	x	x	x
	2, 6, 9, 13, 3, 7, 11, 15 (C)			x		x	x	x	x	x

D – Don't care

Now the essential prime implicant's variable expression is written in SOP form because the function was given in SOP form.

Thus the essential prime implicants = **A'B' + D**

Note: For POS form: If the function is given in POS form, then the same steps of SOP to be followed. But at the end the function to be written in POS form.

Just complement the SOP term and use De-Morgan's theorem to write it in POS form.

Advantages of Tabulation method:

1. Any number of variables can be used for minimization.
2. It can run on a computer using algorithm for running any number of variables.

Disadvantages of Tabulation method:

As variable number increases it becomes tedious to complete minimization manually.

Other Examples for Tabulation method:

Example 1: SOP

Simplify the following expression to sum of product using Tabulation Method

$$F(a, b, c, d) = m(0,4,8,10,12,13,15) + d(1,2)$$

Solution:

a. Determination of Prime Implicants

Group 0	m0: 0000	✓	(0,4)	0-00	✓	(0,4,8,12)	-00
			(0,8)	-000	✓	(0,8,4,12)	-00 redundant
			(0,d1)	000-		(0,8,d2,10)	-0-0
			(0,d2)	00-0	✓	(0,d2,8,10)	-0-0 redundant

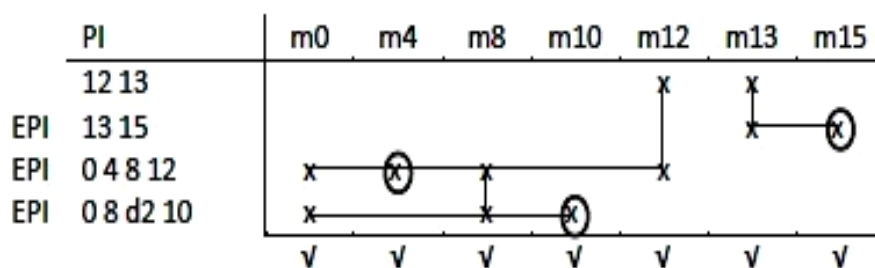
Group 1	m4: 0100	✓	(4,12)	-100	✓		
	m8: 1000	✓	(8,10)	10-0	✓		
	d1: 0001	✓	(8,12)	1-00	✓		
	d2: 0010	✓	(d2,10)	-010	✓		

Group 2	m10: 1010	✓	(12,13)	110-			
	m12: 1100	✓					

Group 3	m13: 1101	✓	(13,15)	11-1			
					✓		

Group 4	m15: 1111	✓					
---------	-----------	---	--	--	--	--	--

b. Prime Implicant Chart:



$$f(a, b, c, d) = bc'd' + a'b' + cd + a'd'$$

Example 2: POS

Simplify the following expression to product of sum using Tabulation Method

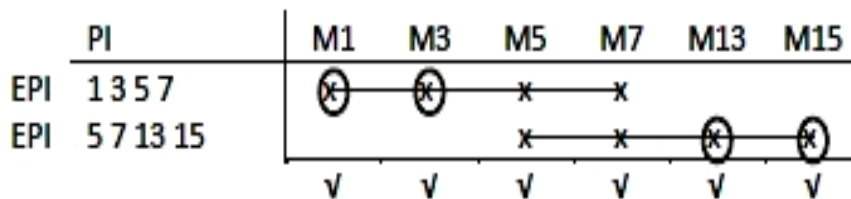
$$F(a, b, c, d) = \prod (1,3,5,7,13,15)$$

Solution:

a. Determination of Prime Implicants

Group 0			
Group 1	M1: 0001	✓ (1,3) 00-1	✓ (1,3,5,7) 0-1
		(1,5) 0-01	✓ (1,5,3,7) 0-1 redundant
Group 2			
Group 2	M3: 0011	✓ (3,7) 0-11	✓ (5,7,13,15) -1-1
	M5: 0101	✓ (5,7) 01-1	✓ (5,13,7,15) -1-1 redundant
		(5,13) -101	✓
Group 3			
Group 3	M7: 0111	✓ (7,15) -111	✓
	M13: 1101	✓ (13,15) 11-1	✓
Group 4			
Group 4	M15: 1111	✓	

b. Prime Implicant Chart:



$$f(a, b, c, d) = (a + d')(b' + d')$$

1.6 LOGIC GATES

Gates are devices that implements basic Boolean logical operations. So they are termed as **Logical Gates**.

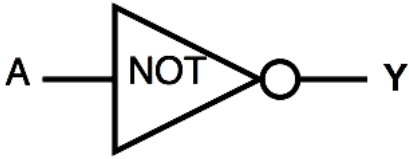
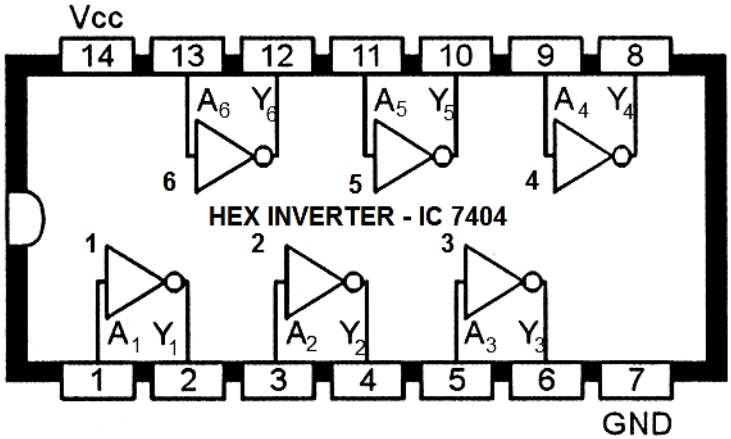
In digital electronics, the gates can be classified into the following types:

1. Basic Logic gates
2. Complemented Logic gates
3. Derived Logic Gates
4. Special Gates

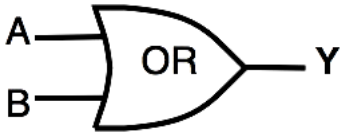
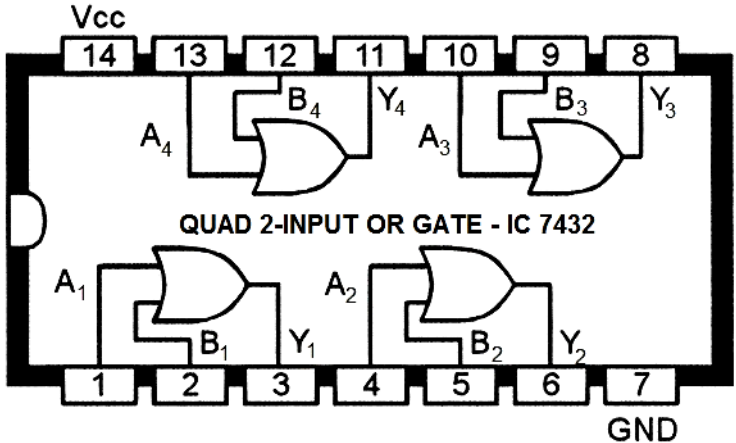
1. The **basic logic gates** implements the basic logical operations like NOT, OR, and AND.

The symbol, truth table, the logical statement, and the sample IC Pin diagram are shown below for each gate. (NOT, OR and AND gates)

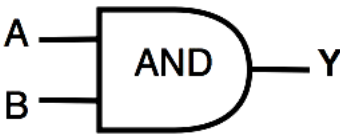
GATE NUMBER 1- INVERTER OR NOT GATE

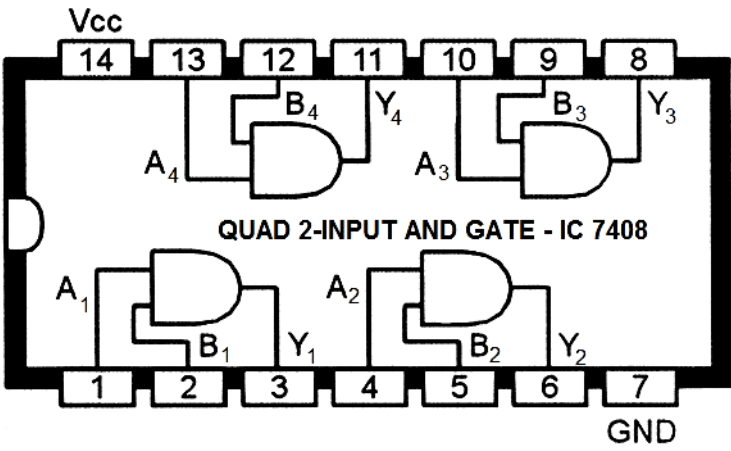
<p style="text-align: center;"><u>INVERTER (OR) NOT GATE- SYMBOL</u></p> <div style="text-align: center;">  </div> <hr style="width: 80%; margin: 10px auto;"/> <p style="text-align: center;"><u>INVERTER (OR) NOT GATE STATEMENT</u></p> <p style="text-align: center;">$Y = \bar{A}$</p>	<p style="text-align: center;"><u>INVERTER (OR) NOT GATE - TRUTH TABLE</u></p> <table border="1" style="margin: 10px auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th style="padding: 5px;">INPUT A</th> <th style="padding: 5px;">OUTPUT Y</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">0 (False)</td> <td style="padding: 5px;">1 (True)</td> </tr> <tr> <td style="padding: 5px;">1 (True)</td> <td style="padding: 5px;">0 (False)</td> </tr> </tbody> </table> <p style="margin-top: 10px;">Case 1: If the input A is false (0), the output Y is True (1). Case 2: If the input A is True (1), the output Y is False (0).</p>	INPUT A	OUTPUT Y	0 (False)	1 (True)	1 (True)	0 (False)
INPUT A	OUTPUT Y						
0 (False)	1 (True)						
1 (True)	0 (False)						
<p>NOT gate is a basic logical gate for inverting operation. So it shall be termed as Inverter.</p> <p>This gate is simple single-input and single output gate as shown in the above symbol.</p> <p>When a Boolean A is given as input, the output Y is the complement of the input. ($Y = A'$)</p>	<p style="text-align: center;"><u>INVERTER (OR) NOT GATE IC 7404 PIN DIAGRAM</u></p> <div style="text-align: center;">  </div>						

GATE NUMBER 2- OR GATE

<p style="text-align: center;"><u>OR GATE- SYMBOL</u></p> <div style="text-align: center;">  </div> <hr style="width: 80%; margin: 10px auto;"/> <p style="text-align: center;"><u>OR GATE STATEMENT</u></p> <p style="text-align: center;">$Y=A+B$</p>	<p style="text-align: center;"><u>OR GATE TRUTH TABLE</u></p> <table border="1" style="margin: 10px auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th colspan="2">INPUTS</th> <th>OUTPUT</th> </tr> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table> <p>0 – False, 1 - True Case 1: If the inputs A & B are false (0), the output Y is false (0). Case 2: If any or all of the inputs are true (1), then output Y is True (1)</p>	INPUTS		OUTPUT	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	1
INPUTS		OUTPUT																	
A	B	Y																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	1																	
<p>OR gate is a basic logical gate for ORing operation.</p> <p>This gate is multi-input and single output gate as shown in the above symbol.</p> <p>When a Boolean A & B are inputs, the output Y is the OR of the input. ($Y = A+B$)</p>	<p style="text-align: center;"><u>OR GATE IC 7432 PIN DIAGRAM</u></p> <div style="text-align: center;">  </div> <p style="text-align: center;">QUAD 2-INPUT OR GATE - IC 7432</p>																		


GATE NUMBER 3- AND GATE

<p style="text-align: center;"><u>AND GATE- SYMBOL</u></p> <div style="text-align: center;">  </div> <hr style="width: 80%; margin: 10px auto;"/> <p style="text-align: center;"><u>AND GATE STATEMENT</u></p> <p style="text-align: center;">$Y=A.B$ or AB</p>	<p style="text-align: center;"><u>AND GATE TRUTH TABLE</u></p> <table border="1" style="margin: 10px auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th colspan="2">INPUTS</th> <th>OUTPUT</th> </tr> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table> <p>0 – False, 1 - True Case 1: If any (or) both the input A & B are false (0), then output Y is false (0).</p>	INPUTS		OUTPUT	A	B	Y	0	0	0	0	1	0	1	0	0	1	1	1
INPUTS		OUTPUT																	
A	B	Y																	
0	0	0																	
0	1	0																	
1	0	0																	
1	1	1																	

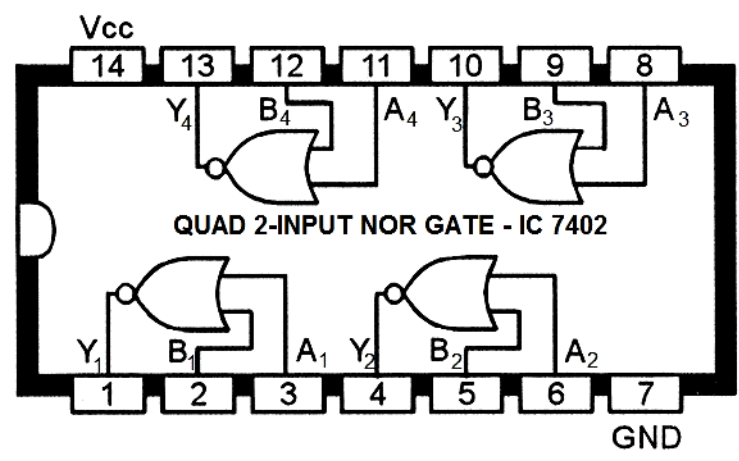
<p>AND gate is a basic logical gate for ANDing operation.</p> <p>This gate is multi-input and single output gate as shown in the above symbol.</p> <p>When a Boolean A & B are inputs, the output Y is the AND of the input. ($Y = A.B$)</p>	<p>Case 2: If both (or) all the inputs are true (1), then output Y is True (1)</p> <p style="text-align: center;">AND GATE IC 7408 PIN DIAGRAM</p>  <p style="text-align: center;">QUAD 2-INPUT AND GATE - IC 7408</p>
---	---

2. The **complemented logic gates** implements the complements of some basic logical operations like OR, and AND. The complements are NOR and NAND for OR and AND respectively. The symbol, truth table, the logical statement, and the sample IC Pin diagram are shown below for each gate. (NAND and NOR gates)

GATE NUMBER 4- NOR GATE

<p style="text-align: center;"><u>NOR GATE- SYMBOL</u></p>  <hr style="width: 80%; margin: 10px auto;"/> <p style="text-align: center;"><u>NOR GATE STATEMENT</u></p> <p style="text-align: center;">$Y = \overline{A + B}$</p>	<p style="text-align: center;"><u>NOR GATE TRUTH TABLE</u></p> <table border="1" style="margin: auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th colspan="2">INPUTS</th> <th>OUTPUT</th> </tr> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table> <p>0 – False, 1 - True Case 1: If all (or) both the input A & B are false (0), then output Y is True (1). Case 2: If any (or) all of the inputs are true (1), then output Y is False (0)</p>	INPUTS		OUTPUT	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	0
INPUTS		OUTPUT																	
A	B	Y																	
0	0	1																	
0	1	0																	
1	0	0																	
1	1	0																	
<p>NOR gate is a complemented logical gate for ORing & NOT operation.</p> <p>This gate is multi-input and single output gate as shown in the above symbol.</p> <p>When a Boolean A & B are inputs,</p>	<p style="text-align: center;"><u>NOR GATE</u> <u>IC 7402 PIN DIAGRAM</u> <u>(THE GATE DIRECTION IS OPPOSITE OF OTHER GATES)</u></p>																		

the output Y is the OR with NOT of the input. ($Y = \overline{A + B}$)



GATE NUMBER 5- NAND GATE

NAND GATE- SYMBOL



NAND GATE STATEMENT

$$Y = \overline{A \cdot B}$$

NAND GATE TRUTH TABLE

INPUTS		OUTPUT
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

0 – False, 1 - True

Case 1: If any (or) all the input A & B are false (0), then output Y is True (1).

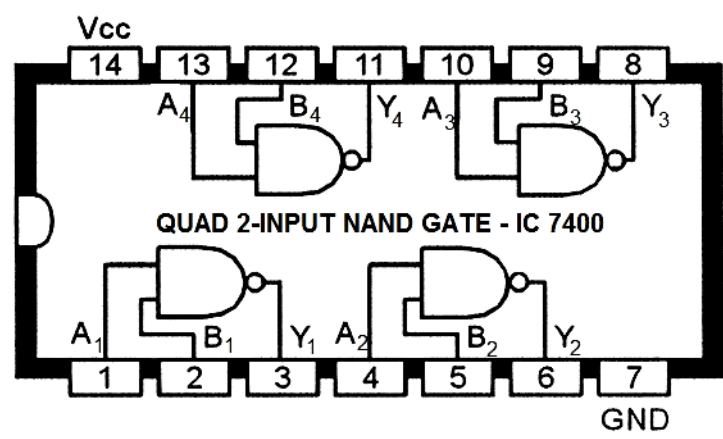
Case 2: If all of the inputs are true (1), then output Y is False (0)

NAND gate is a complemented logical gate for ANDing & NOT operation.

This gate is multi-input and single output gate as shown in the above symbol.

When a Boolean A & B are inputs, the output Y is the AND with NOT of the input. ($Y = \overline{A \cdot B}$)

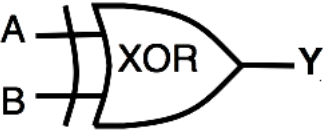
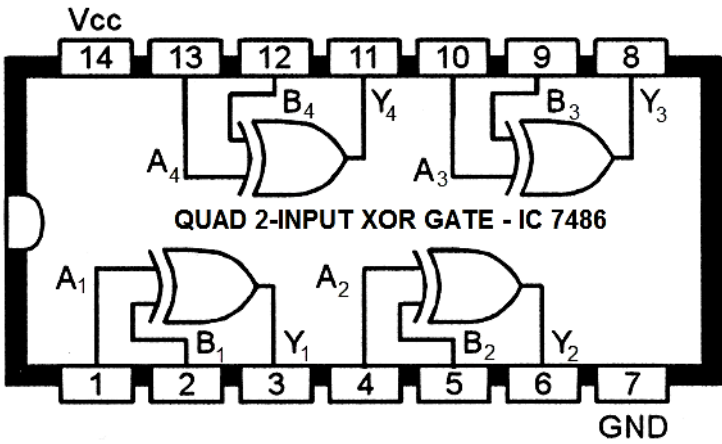
NAND GATE IC 7400 PIN DIAGRAM



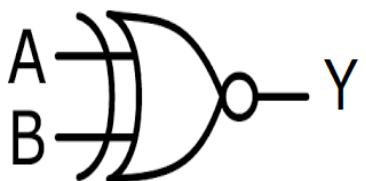
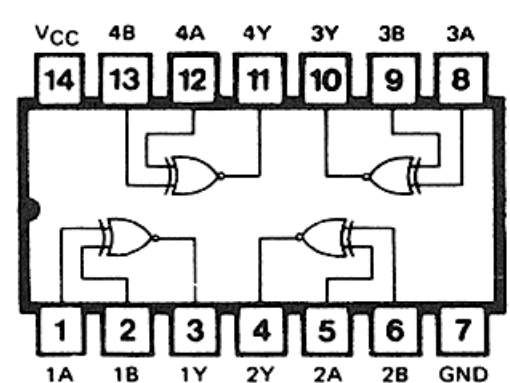
3. The **Derived logic gates** implements the derivation of some basic logical operations like EXCLUSIVE-OR and EXCLUSIVE NOR.

The symbol, truth table, the logical statement, and the sample IC Pin diagram are shown below for each gate. (XOR and XNOR gates)

GATE NUMBER 6- EXCLUSIVE OR GATE

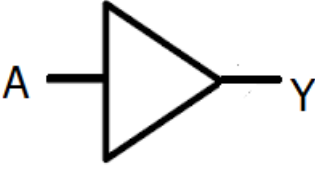
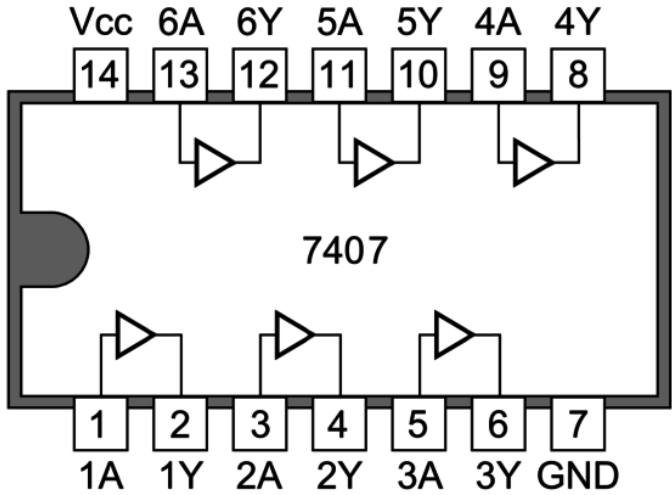
<p style="text-align: center;"><u>XOR GATE- SYMBOL</u></p> <div style="text-align: center;">  </div> <hr style="width: 80%; margin: 10px auto;"/> <p style="text-align: center;"><u>XOR GATE STATEMENT</u></p> <p style="text-align: center;">$Y = A \oplus B$ $Y = \bar{A}B + A\bar{B}$</p> <p style="text-align: center;"><i>both statements are the same</i></p>	<p style="text-align: center;"><u>XOR GATE TRUTH TABLE</u></p> <table border="1" style="margin: 10px auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th colspan="2">INPUTS</th> <th>OUTPUT</th> </tr> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table> <p>0 – False, 1 - True Case 1: If the inputs are of ODD parity, that is, have odd number of ones (1), and then output Y is True (1). Case 2: If the inputs are of EVEN parity, that is, have EVEN number of ones (1), and then output Y is False (0).</p>	INPUTS		OUTPUT	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	0
INPUTS		OUTPUT																	
A	B	Y																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	0																	
<p>Exclusive OR gate or XOR gate is a derived logical gate that uses XOR operation.</p> <p>This gate is multi-input and single output gate as shown in the above symbol.</p> <p>When a Boolean A & B are inputs, the output Y is the XOR of the input. ($Y = A \oplus B$)</p> <p>This gate is otherwise called as ODD PARITY CHECKER, because when inputs are of ODD parity, it gives output True (1).</p>	<p style="text-align: center;"><u>XOR GATE IC 7486 PIN DIAGRAM</u></p> <div style="text-align: center;">  </div> <p style="text-align: center;">QUAD 2-INPUT XOR GATE - IC 7486</p>																		

GATE NUMBER 7- EXCLUSIVE NOR GATE

<u>XNOR GATE- SYMBOL</u>	<u>XNOR GATE TRUTH TABLE</u>																		
<div style="text-align: center;">  </div> <p style="text-align: center; margin-top: 10px;"><u>XNOR GATE STATEMENT</u></p> <p style="text-align: center;">$Y = A \odot B$ $Y = AB + \overline{AB}$</p> <p style="text-align: center;"><i>both statements are the same</i></p>	<table border="1" style="margin: 0 auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th colspan="2">INPUTS</th> <th>OUTPUT</th> </tr> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table> <p>0 – False, 1 - True Case 1: If the inputs are of EVEN parity, that is, have EVEN number of ones (1), and then output Y is True (1). Case 2: If the inputs are of ODD parity, that is, have ODD number of ones (1), and then output Y is False (0).</p>	INPUTS		OUTPUT	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	1
INPUTS		OUTPUT																	
A	B	Y																	
0	0	1																	
0	1	0																	
1	0	0																	
1	1	1																	
<p>Exclusive NOR gate or XNOR gate is a derived logical gate that uses XNOR operation.</p> <p>This gate is multi-input and single output gate as shown in the above symbol.</p> <p>When a Boolean A & B are inputs, the output Y is the XNOR of the input. ($Y = A \odot B$)</p> <p>This gate is otherwise called as EVEN PARITY CHECKER, because when inputs are of EVEN parity, it gives output True (1).</p>	<p style="text-align: center;"><u>XNOR GATE IC 74266 PIN DIAGRAM</u></p> 																		

4. The **Special logic gates** as the name states, used in special situations and for special purpose. They are Buffer Gate, Tristate Logic gates (Discussed in Unit 4) and so on. The symbol, truth table, the logical statement, and the sample IC Pin diagram are shown below for each gate. (BUFFER gates)

GATE NUMBER 8- BUFFER GATE

<p style="text-align: center;"><u>BUFFER GATE- SYMBOL</u></p> <div style="text-align: center;">  </div> <p style="text-align: center;"><u>BUFFER GATE STATEMENT</u></p> <p style="text-align: center;">Y = A</p>	<p style="text-align: center;"><u>BUFFER GATE TRUTH TABLE</u></p> <table border="1" style="margin: auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th>INPUT A</th> <th>OUTPUT Y</th> </tr> </thead> <tbody> <tr> <td>0 (False)</td> <td>0 (False)</td> </tr> <tr> <td>1 (True)</td> <td>1 (true)</td> </tr> </tbody> </table> <p>0 – False, 1 - True Case 1: If the input A is False(0), then output Y is False (0). Case 2: If the input A is True(1), then output Y is True(1).</p>	INPUT A	OUTPUT Y	0 (False)	0 (False)	1 (True)	1 (true)
INPUT A	OUTPUT Y						
0 (False)	0 (False)						
1 (True)	1 (true)						
<p>Buffer gate is a special logical gate for the purpose of isolation. Isolation stops input being loaded by output.</p> <p>This gate is simple single-input and single output gate as shown in the above symbol.</p> <p>When a Boolean A is given as input, the output Y is the same as the input. (Y = A)</p>	<p style="text-align: center;"><u>BUFFER GATE IC 7407 PIN DIAGRAM</u></p> <div style="text-align: center;">  </div>						

NOTE: HOW TO IMPLEMENT BOOLEAN EXPRESSION USING GATES?

- Let us consider a switching function $F = AB + CD + E$
- The function has three expressions AB, CD and E ORed together.
- Each expression can be realized into gate.
- AB is realized by using a AND gate with two inputs A & B. The Output of the first AND gate is AB. CD is realized by using a AND gate with two inputs C & D. The Output of the second AND gate is CD. Third expression E is a single expression and doesn't need any gate to realize. This part of circuit is termed LEVEL 1.
- Since as said earlier all expressions are ORed together, thus an OR gate is used to receive the outputs of Level 1 and propagate it to next level. This level OR gate is in LEVEL 2.

- Thus the circuit is two level AND-OR logic circuit for the function $F = AB + CD + E$

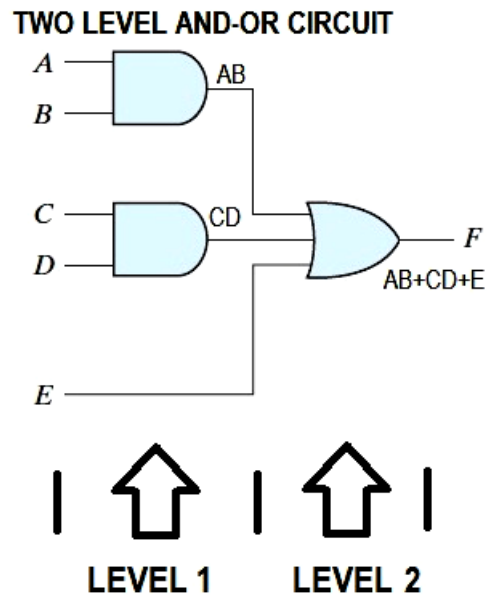


Figure 3.5.1.a TWO LEVEL AND-OR CIRCUIT

1.6.1 UNIVERSAL GATES

Universal gates are the gates which can be used to implement any gate.

NAND and NOR gates are to be UNIVERSAL gates. They both can be used to implement any of the basic gates such as NOT, OR and AND gates. In this section, the implementation using NAND and NOR shall be individually discussed.

NAND gate as Universal gate:

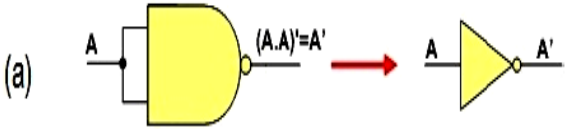
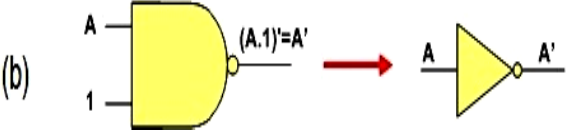
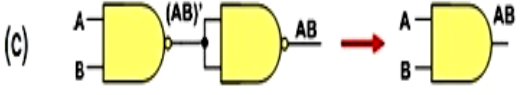
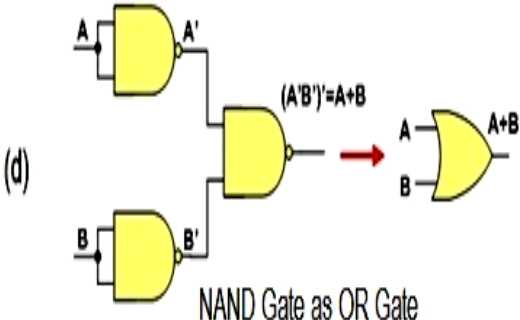
<p>NAND gate as NOT gate: Consider the truth table of NAND gate</p> <table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th colspan="2">INPUTS</th> <th>OUTPUT</th> </tr> <tr> <th>A</th> <th>B</th> <th>Y</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table> <p>When the above truth table of NAND is checked, when both inputs are the same, the output is complement of the input. 1. when both inputs are '0', output is '1' and when both inputs are '1', output is '0' acting like NOT gate as shown in <i>fig a</i> beside.</p>	INPUTS		OUTPUT	A	B	Y	0	0	1	0	1	1	1	0	1	1	1	0	<p>(a) </p> <p>(b) </p> <p style="text-align: center;">NAND gate used as NOT gate</p>
INPUTS		OUTPUT																	
A	B	Y																	
0	0	1																	
0	1	1																	
1	0	1																	
1	1	0																	
<p>NAND gate as AND gate: This becomes simple because when output of NAND gate is inverted, it becomes the output of AND gate. Thus an inverter or NOT gate is added to the NAND gate output. (<i>Figure c</i>)</p>	<p>(c) </p> <p style="text-align: center;">NAND Gate as AND Gate</p>																		
<p>NAND gate as OR gate: 1. NAND Gate statement: $Y = \overline{A \cdot B}$ 2. Using Demorgan's Theorem $Y = \overline{(\overline{A} \cdot \overline{B})} = \overline{\overline{A}} + \overline{\overline{B}}$ 3. The above statement is ORing \overline{A} and \overline{B} 4. So an inverter (NOT gate) for converting A to \overline{A} and B to \overline{B} is added before another NAND gate inputs as shown in the <i>figure d</i> beside.</p>	<p>(d) </p> <p style="text-align: center;">NAND Gate as OR Gate</p>																		

Figure 1.6.1.1. (a, b) NOT, (c) OR & (d) AND gates implemented using NAND Gate

The above figure 1.6.1.1, shows the details of NAND gate being used as NOT gate, AND gate and OR gate.

NOR gate as Universal gate:

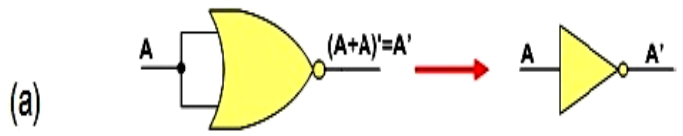
NOR gate as NOT gate:

Consider the truth table of NOR gate

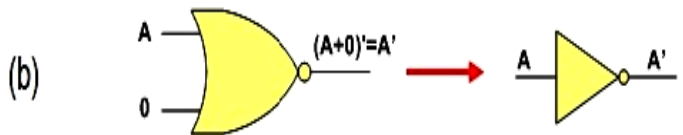
INPUTS		OUTPUT
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

When the above truth table of NOR is checked, when both inputs are the same, the output is complement of the input.

1. when both inputs are '0', output is '1' and when both inputs are '1', output is '0' acting like NOT gate as shown in *fig a* beside.

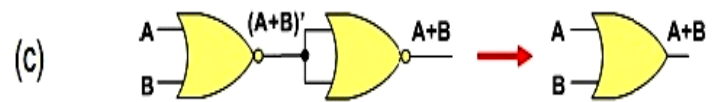


NOR Gate as NOT gate



NOR gate as OR gate:

This becomes simple because when output of NOR gate is inverted, it becomes the output of OR gate. Thus an inverter or NOT gate is added to the NOR gate output. (*Figure c*)



NOR Gate as OR Gate

NOR gate as AND gate:

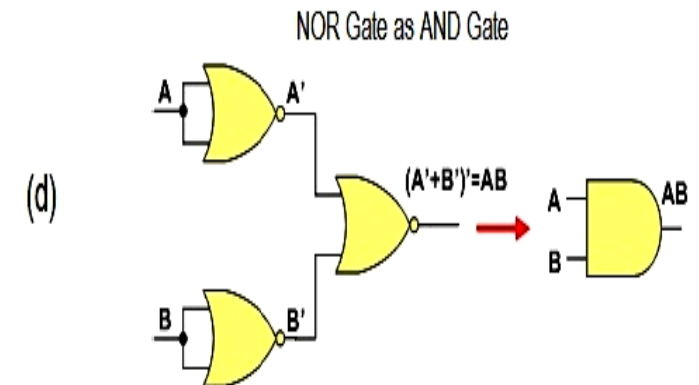
1. NOR Gate statement: $Y = \overline{A + B}$

2. Using Demorgan's Theorem

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

3. The above statement is ORing \bar{A} and \bar{B}

4. So an inverter (NOT gate) for converting A to \bar{A} and B to \bar{B} is added before another NOR gate inputs as shown in the *figure d* beside.



NOR Gate as AND Gate

Figure 1.6.1.2. (a, b) NOT, (c) OR & (d) AND gates implemented using NOR Gate

The above figure 1.6.1.2, shows the details of NOR gate being used as NOT gate, AND gate and OR gate.

1.6.2 NAND IMPLEMENTATION & NOR IMPLEMENTATION

SOP expression to be implemented using NAND gates:

When SOP is implemented using Gates, the final circuit has AND gates at first level (connected to the input) and then an OR gate at next level as shown in the below figure.

The switching function $F = AB + CD$ has been implemented using AND-OR gates as shown in figure 1.6.2.1(a) below.

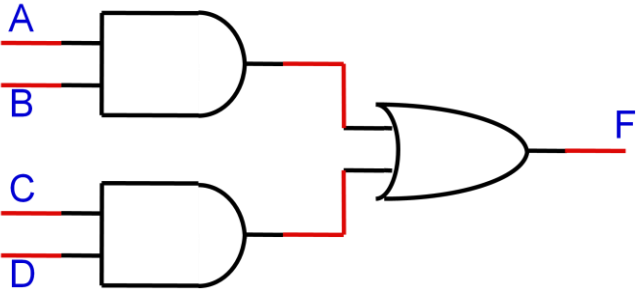
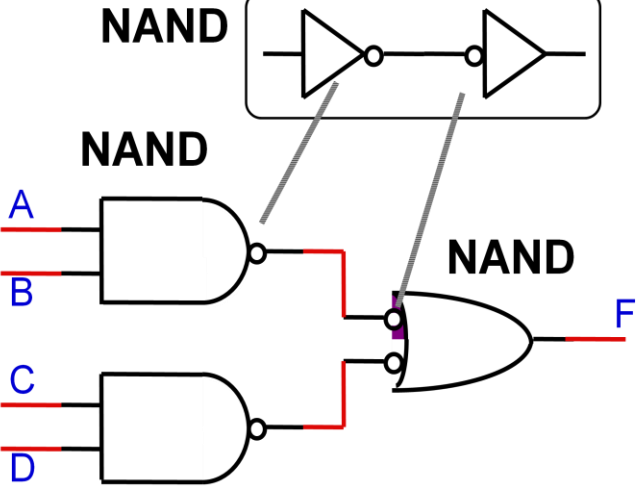
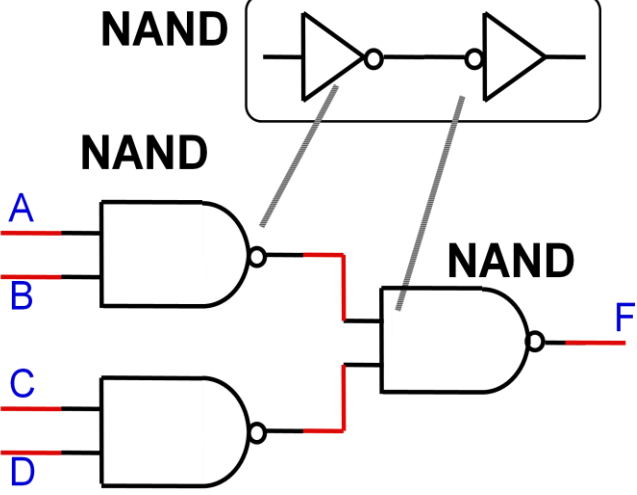
<p style="text-align: center;">AND-OR IMPLEMENTATION</p> <p style="text-align: center;">SOP $F = AB + CD$</p> <p>Two input AND gates for AB and CD as LEVEL 1 gates. An OR gate for ORing $AB + CD$ as LEVEL 2 gate.</p>	
<p>1. Add bubbles, to the outputs of FIRST LEVEL AND gates.</p> <p>2. Add bubbles in the inputs of OR gates of SECOND LEVEL.</p>	<p style="text-align: center;">NAND</p> 
<p>NAND-NAND IMPLEMENTATION</p> <p>1. As shown in the figure, BUBBLED OR gate is replaced by NAND gate.</p> <p>2. Thus circuit is converted to NAND-NAND logic circuit.</p>	<p style="text-align: center;">NAND</p> 

Figure 1.6.2.1.a. Two level NAND-NAND implementation from AND-OR circuit

POS Expression to be implemented using NOR gates:

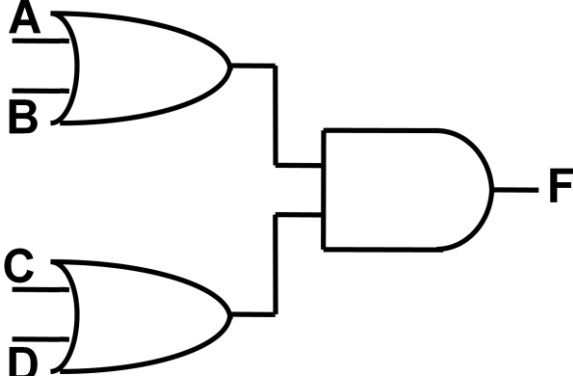
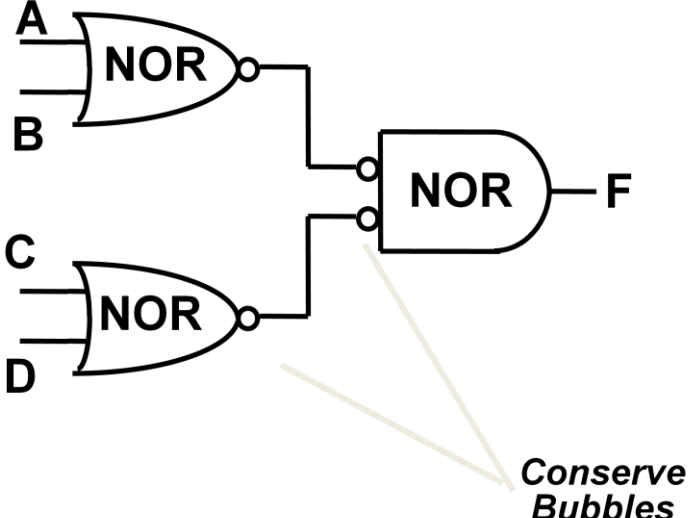
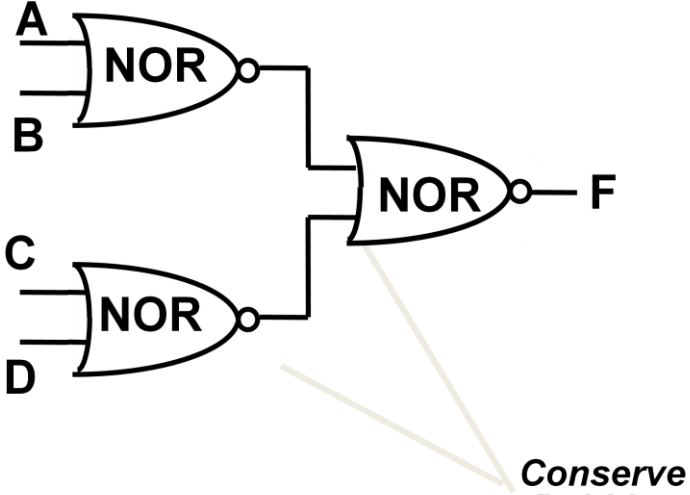
<p><u>OR-AND IMPLEMENTATION</u></p> <p style="text-align: center;"><u>POS</u> $F = (A + B).(C + D)$</p> <p>Two input OR gates for $A + B$ and $C + D$ as LEVEL 1 gates. An AND gate for ANDing $(A + B).(C + D)$ as LEVEL 2 gate.</p>	
<p>1. Add bubbles, to the outputs of FIRST LEVEL OR gates.</p> <p>2. Add bubbles in the inputs of AND gates of SECOND LEVEL.</p>	 <p style="text-align: right;"><i>Conserve Bubbles</i></p>
<p><u>NOR-NOR IMPLEMENTATION</u></p> <p>1. As shown in the figure, BUBBLED AND gate is replaced by NOR gate.</p> <p>2. Thus circuit is converted to NOR-NOR logic circuit.</p>	 <p style="text-align: right;"><i>Conserve Bubbles</i></p>

Figure 1.6.2.1.b Two level NOR-NOR implementation from OR-AND circuit

SOP expression to be implemented using NAND gates:

<p style="text-align: center;">AND-OR IMPLEMENTATION</p> <p style="text-align: center;"><u>SOP</u> $F = AB + CD$</p> <p>Two input AND gates for AB and CD as LEVEL 1 gates. An OR gate for ORing $AB + CD$ as LEVEL 2 gate.</p>	
<p>Step 1: Add bubbles, to the outputs of FIRST LEVEL AND gates and an inverter (NOT) gate to the input as shown in the figure beside.</p>	<p style="text-align: center;">Step 1</p> <p style="text-align: center;">Conserve "Bubbles"</p>
<p>Step 2: Output OR gate is added with a bubble and a NOT gate (INVERTER) following it.</p> <p>Thus as shown in the figure beside, the OR-AND gate logic is converted into NAND-NAND logic.</p> <p>Thus the above are the steps to convert POS to NAND-NAND gate logic.</p>	<p style="text-align: center;">Step 2</p> <p style="text-align: right;">Conserve "Bubbles"</p>

Figure 1.6.2.1.c Two level NOR-NOR implementation from AND-OR circuit

POS expression to be implemented using NAND gates:

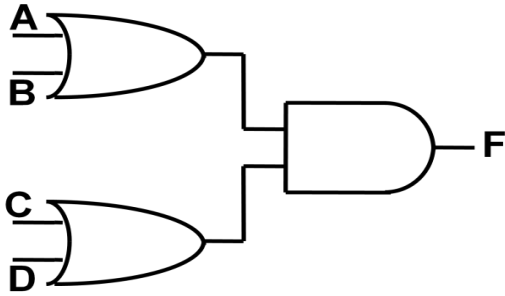
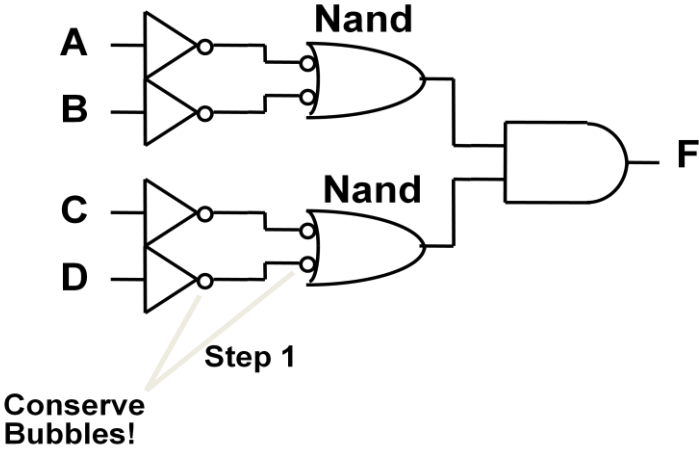
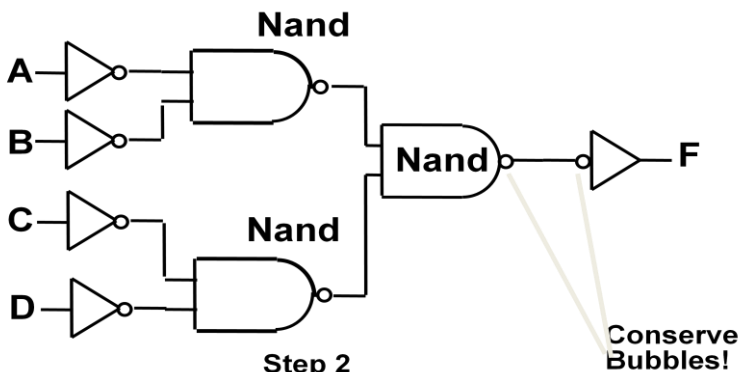
<p>OR-AND IMPLEMENTATION</p> <p style="text-align: center;"><u>POS:</u> $F = (A + B).(C + D)$</p> <p>Two input OR gates for $A + B$ and $C + D$ as LEVEL 1 gates. An AND gate for ANDing $(A + B).(C + D)$ as LEVEL 2 gate.</p>	
<p>Step 1: Add bubbles, to the outputs of FIRST LEVEL OR gates and an inverter (NOT) gate to the input as shown in the figure beside.</p>	 <p style="text-align: center;">Step 1</p> <p>Conserve Bubbles!</p>
<p>Step 2: Output AND gate is added with a bubble and a NOT gate (INVERTER) following it.</p> <p>Thus as shown in the figure beside, the OR-AND gate logic is converted into NAND-NAND logic.</p> <p>Thus the above are the steps to convert POS to NAND-NAND gate logic.</p>	 <p style="text-align: center;">Step 2</p> <p>Conserve Bubbles!</p>

Figure 1.6.2.1.d Two level NAND-NAND implementation from OR-AND circuit

Examples:

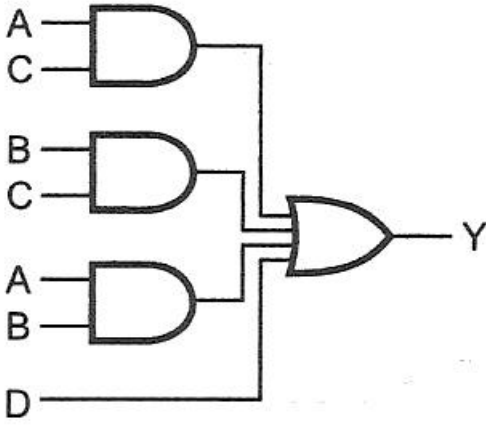
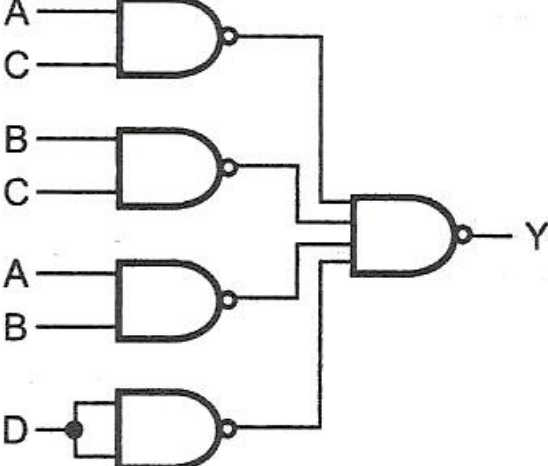
<p>AND-OR IMPLEMENTATION <u>SOP</u> $Y = AC + ABC + \bar{A}BC + AB + D$</p> <p>$Y = AC + ABC(A + \bar{A}) + AB + D$</p> <p>$Y = AC + ABC + AB + D$</p> <p>Step 1: The above reduced function is implemented using AND-OR Logic as shown in the figure (a) beside.</p>	 <p style="text-align: center;">(a)</p>
<p>NAND-NAND IMPLEMENTATION</p> <p>Step 2: Add bubbles, to the outputs of FIRST LEVEL AND gates and an inverter (NOT) gate to the input of SECOND level OR gate that becomes NAND gate (Bubbled OR) as shown in the figure beside.</p>	 <p style="text-align: center;">(b)</p>

Figure 1.6.3.1 Two level NAND-NAND implementation from AND-OR circuit

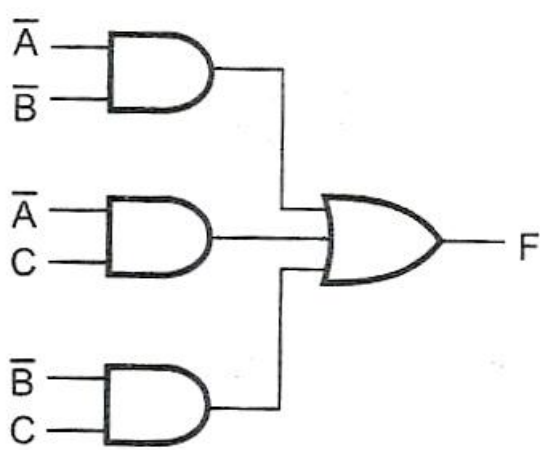
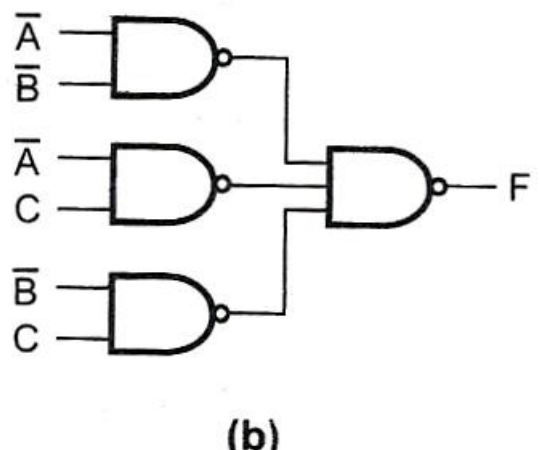
<p>AND-OR IMPLEMENTATION</p> <p style="text-align: center;"><u>SOP</u> $Y = \bar{A}\bar{B} + \bar{A}C + \bar{B}C$</p> <p>Step 1: The above function is implemented using AND-OR Logic as shown in the figure (a) beside.</p>	 <p style="text-align: center;">(a)</p>
<p>NAND-NAND IMPLEMENTATION</p> <p>Step 2: Add bubbles, to the outputs of FIRST LEVEL AND gates and an inverter (NOT) gate to the input of SECOND level OR gate that becomes NAND gate (Bubbled OR) as shown in the figure beside.</p>	 <p style="text-align: center;">(b)</p>

Figure 1.6.3.2 Two level NAND-NAND implementation from AND-OR circuit

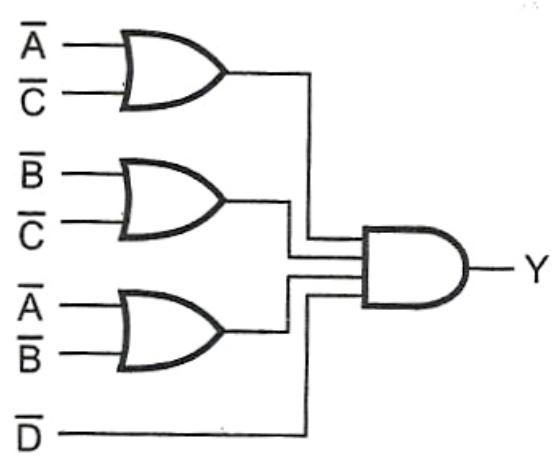
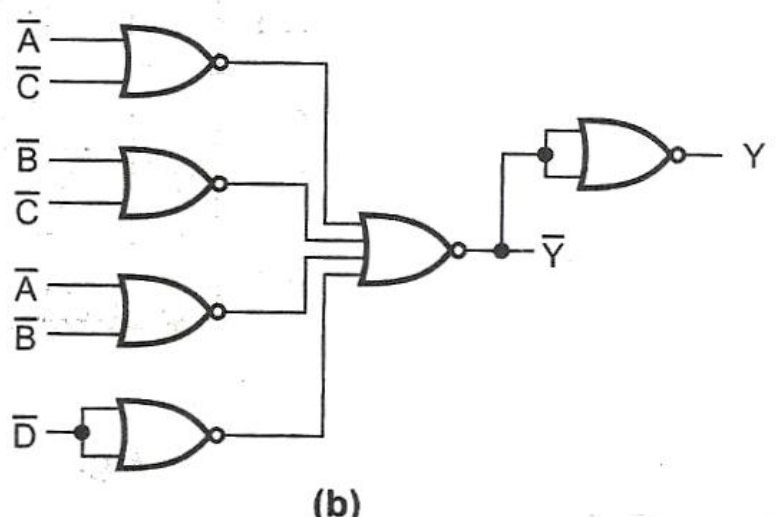
<p><u>OR-AND IMPLEMENTATION</u></p> <p style="text-align: center;"><u>POS</u></p> $Y = (\bar{A} + \bar{C}).(\bar{A} + \bar{B}).(\bar{B} + \bar{C}).\bar{D}$ <p>Step 1: The above function is implemented using OR-AND Logic as shown in the figure (a) beside.</p>	 <p>(a)</p>
<p><u>NOR-NOR IMPLEMENTATION</u></p> <p>Step 2: Add bubbles, to the outputs of FIRST LEVEL OR gates and an inverter (NOT) gate to the input of SECOND level AND gate that becomes NOR gate (Bubbled AND) as shown in the figure beside.</p>	 <p>(b)</p>

Figure 1.6.3.3 Two level NOR-NOR implementation from OR-AND circuit

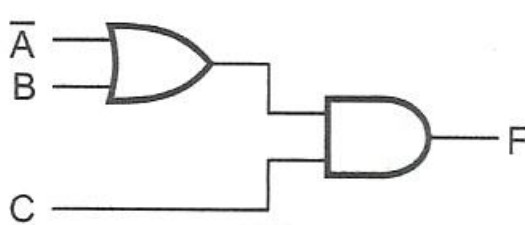
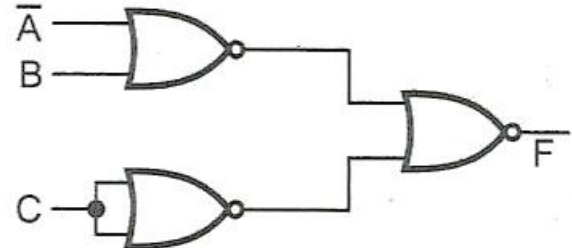
<p style="text-align: center;"><u>OR-AND IMPLEMENTATION</u> <u>POS</u> $Y = (\bar{A} + B).C$</p> <p>Step 1: The above function is implemented using OR-AND Logic as shown in the figure (a) beside.</p>	 <p style="text-align: center;">(a)</p>
<p style="text-align: center;"><u>NOR-NOR IMPLEMENTATION</u></p> <p>Step 2: Add bubbles, to the outputs of FIRST LEVEL OR gates and an inverter (NOT) gate to the input of SECOND level AND gate that becomes NOR gate (Bubbled AND) as shown in the figure beside.</p>	 <p style="text-align: center;">(b)</p>

Figure 1.6.3.4 Two level NOR-NOR implementation from OR-AND circuit