ATME College of Engineering

ACADEMIC YEAR -2025-26 (ODD SEM)

SUBJECT: Digital Design & Computer Organization

SUBJECT CODE: BCS302

MODULE -1

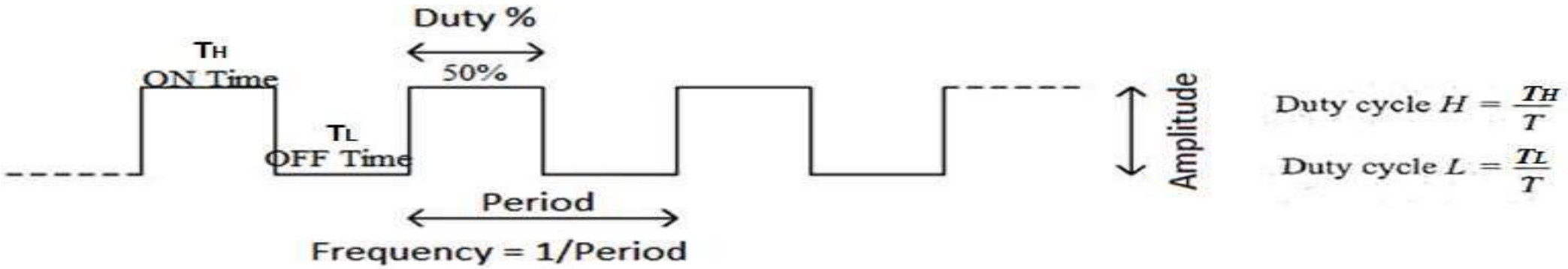
Prepared by:
Mr Kalathma M K,
Asst prof, Dept. of CSE
ATMECE

Contents

Introduction

- **Analog signal** is a signal whose amplitude can take any value between given limits. An **analog circuits** operates on continuous signals.
- **Digital signal** is a signal whose amplitude can have only given discrete values between defined limits. A **digital circuits** operates on discrete signals.
- **Clock** is a periodic, rectangular waveform used as a basic timing signal.
- A **table** that shows all of the input output possibilities of a logic circuit is called **truth table**.

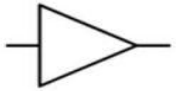
Clock



- A **digital circuit** having one or more input signals but only one output signal is called a gate.
- The most **basic gates** are -the **NOT** gate (inverter), the **OR** gate and the **AND** gate.
- Any logic function/ logic circuit can be implemented using only one kind of gate then such gates are called **universal logic gates**. **NOR** gate and **NAND** gate are called universal logic gates.
- The other logic gates are **XOR** and **XNOR** gate.

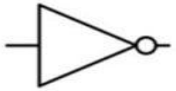
Logic gates

Buffer



Input	Output
0	0
1	1

Inverter



Input	Output
0	1
1	0

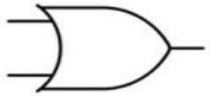
AND

A	B	Output
0	0	0
1	0	0
0	1	0
1	1	1

NAND

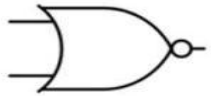
A	B	Output
0	0	1
1	0	1
0	1	1
1	1	0

OR



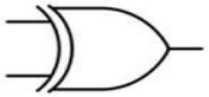
A	B	Output
0	0	0
1	0	1
0	1	1
1	1	1

NOR



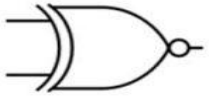
A	B	Output
0	0	1
1	0	0
0	1	0
1	1	0

XOR



A	B	Output
0	0	0
1	0	1
0	1	1
1	1	0

XNOR



A	B	Output
0	0	1
1	0	0
0	1	0
1	1	1

Boolean Rules & Laws

Table 5-3: Summary of Rules and Laws of Boolean Algebra

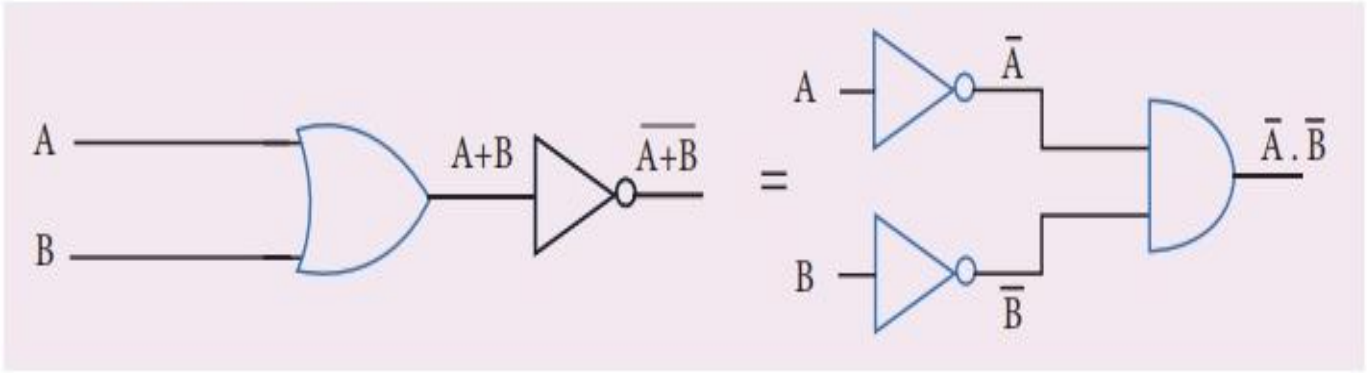
Reference	Rule or Law	Reference	Rule or Law
Rule 1	$A + 0 = A$	Rule 10	$A + AB = A$
Rule 2	$A + 1 = 1$	Rule 11	$A + A'B = A$
Rule 3	$A \cdot 0 = 0$	Rule 12	$(A + B)(A + C) = A + BC$
Rule 4	$A \cdot 1 = A$	Commutative Law	$A + B = B + A$
Rule 5	$A + A = A$		$AB = BA$
Rule 6	$A + A' = 1$	Associative Law	$A + (B + C) = (A + B) + C$
Rule 7	$A \cdot A = A$		$A(BC) = (AB)C$
Rule 8	$A \cdot A' = 0$	Distributive Law	$A(B + C) = AB + AC$
Rule 9	$(A')' = A$		

De Morgan's Theorem

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

The **First** theorem states that "**The complement of the sum of all the terms is equal to the product of the complement of each term**".

De Morgan's first theorem

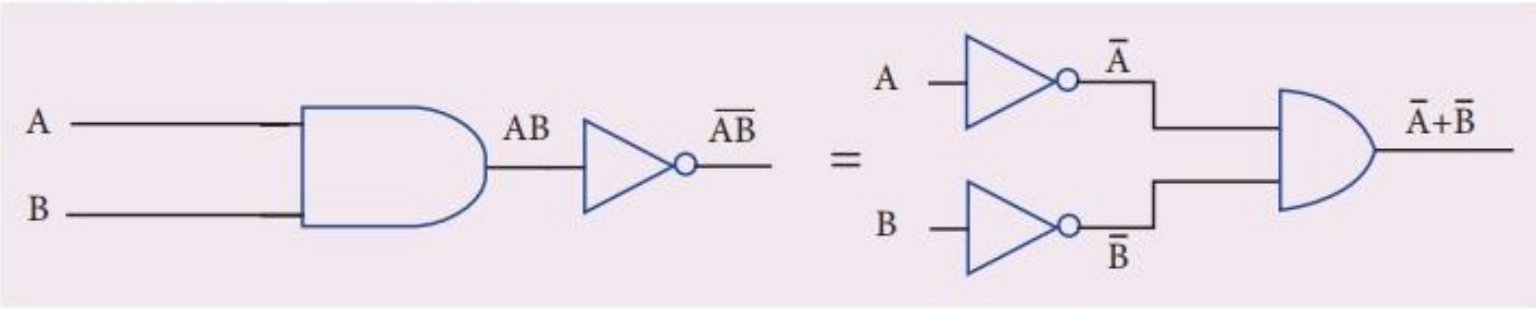


A	B	$\overline{A+B}$	$\overline{A}$	$\overline{B}$	$\overline{A} \cdot \overline{B}$
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

The **First** theorem states that **"The complement of the product of all the terms is equal to the sum of the complement of each term"**.

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

De Morgan's second theorem



A	B	$\overline{AB}$	$\bar{A}$	$\bar{B}$	$\bar{A}+\bar{B}$
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

Minterms and Maxterms for Three Variables

Row No.	A B C	Minterms	Maxterms
0	0 0 0	$A'B'C' = m_0$	$A + B + C = M_0$
1	0 0 1	$A'B'C = m_1$	$A + B + C' = M_1$
2	0 1 0	$A'BC' = m_2$	$A + B' + C = M_2$
3	0 1 1	$A'BC = m_3$	$A + B' + C' = M_3$
4	1 0 0	$AB'C' = m_4$	$A' + B + C = M_4$
5	1 0 1	$AB'C = m_5$	$A' + B + C' = M_5$
6	1 1 0	$ABC' = m_6$	$A' + B' + C = M_6$
7	1 1 1	$ABC = m_7$	$A' + B' + C' = M_7$

- A **minterm** is a **product** of **"n"** literals
- A **maxterm** is a **sum** of **"n"** literals
- **Sum Of Products (SOP)**: A Boolean equation that is the logical sum of logical products. This type of equation applies to an AND-OR circuit.
- **Product Of Sums (POS)**: A Boolean equation that is the logical product of logical sums. This type of equation applies to an OR-AND circuit.

Steps to get SOP

1) Locate each output 1 in truth table
2) Write the respective **minterm**
3) Apply OR operation to the **minterms**

A	B	C	Y	
0	0	0	0	
0	0	1	1	$\overline{A}.\overline{B}.C$
0	1	0	1	$\overline{A}.B.\overline{C}$
0	1	1	1	$\overline{A}.B.C$
1	0	0	0	
1	0	1	0	
1	1	0	1	$A.B.\overline{C}$
1	1	1	0	

SOP:

$\overline{A}.\overline{B}.C + \overline{A}.B.\overline{C} + \overline{A}.B.C + A.B.\overline{C}$ $\quad \Sigma m(1, 2, 3, 6)$

Steps to get POS

1) Locate each output 0 in truth table
2) Write the respective **maxterm**
3) Apply AND operation to the **maxterms**

A	B	C	Y	
0	0	0	0	A+B+C
0	0	1	1	
0	1	0	1	
0	1	1	1	
1	0	0	0	$\overline{A}$+B+C
1	0	1	0	$\overline{A}$+B+$\overline{C}$
1	1	0	1	$\overline{A}$+$\overline{B}$+C
1	1	1	0	$\overline{A}$+$\overline{B}$+$\overline{C}$

POS:

$(A+B+C).(\overline{A}+B+C).(\overline{A}+B+\overline{C}).(\overline{A}+\overline{B}+\overline{C}) = \Pi M(0, 4, 5, 7)$

The various Boolean expression simplification techniques are

1) Algebraic techniques
2) Karnaugh Map/K-Map Method
3) Quine McCluskey Method
4) Entered Variable Map/ MEV/EMV Method

Karnaugh map/K map is a method simplifying and manipulating switching functions.

K map method is faster and easier to apply than other simplification methods.

Two Variable K-Map

The number of cells in 2 variable K-map is four (2X2), since the number of variables is two. The following figure shows 2 variable K-map and location of minterms on a 2 variable K-map

	$\overline{B}$	B
$\overline{A}$	$\overline{A}\,\overline{B}$	$\overline{A}B$
A	$A\overline{B}$	AB

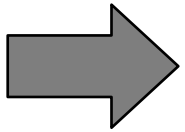
	0	1
0	00	01
1	10	11

	$\overline{B}$	B
$\overline{A}$	0	1
A	2	3

Example: Convert following truth table into K map.

$Y = F(A, B) = \Sigma\, m\,(2, 3)$

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



	$\overline{B}$	B
$\overline{A}$	0 (0)	0 (1)
A	1 (2)	1 (3)

Three Variable K-Map

The number of cells in 3 variable K-map is eight (23), since the number of variables is 3. The following figure shows 3 variable K-map and location of minterms on a 3 variable K-map.

	$\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
$\bar{A}$	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$\bar{A}BC$	$\bar{A}B\bar{C}$
A	$A\bar{B}\bar{C}$	$A\bar{B}C$	ABC	$AB\bar{C}$

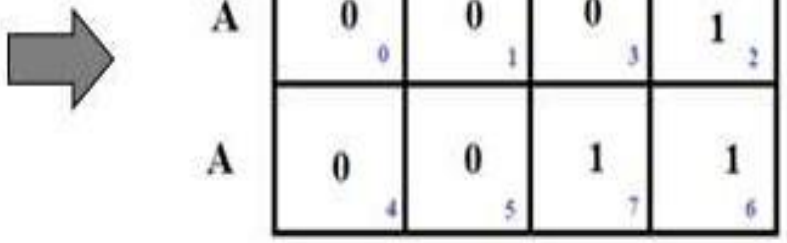
A \ BC	00	01	11	10
0	000	001	011	010
1	100	101	111	110

	$\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
$\bar{A}$	0	1	3	2
A	4	5	7	6

Convert following truth table into K map.

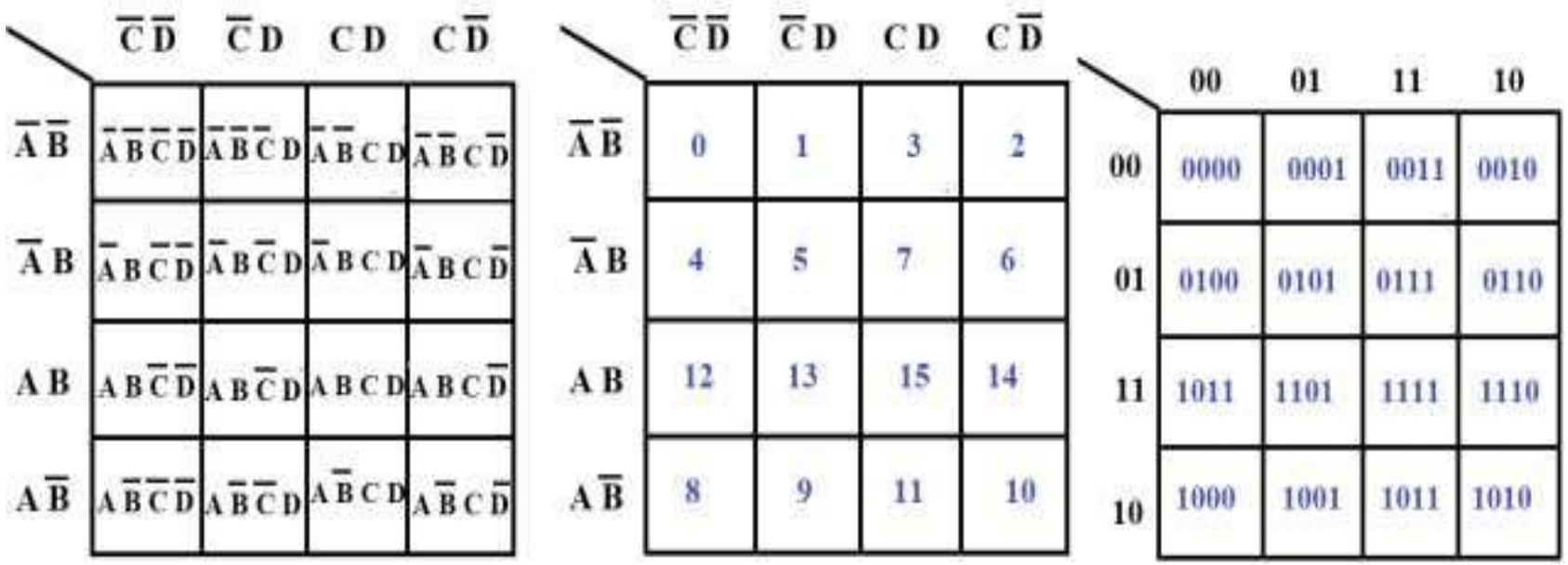
$Y = F(A, B, C) = \Sigma\, m\,(2,6,7)$

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



Four-Variable Karnaugh Maps

The number of cells in 4 variable K-map is sixteen (2^4), since the number of variables is 4. The following figure shows 4 variable K-map and location of min terms on a 4 variable K-map.



Convert following truth table into K map.

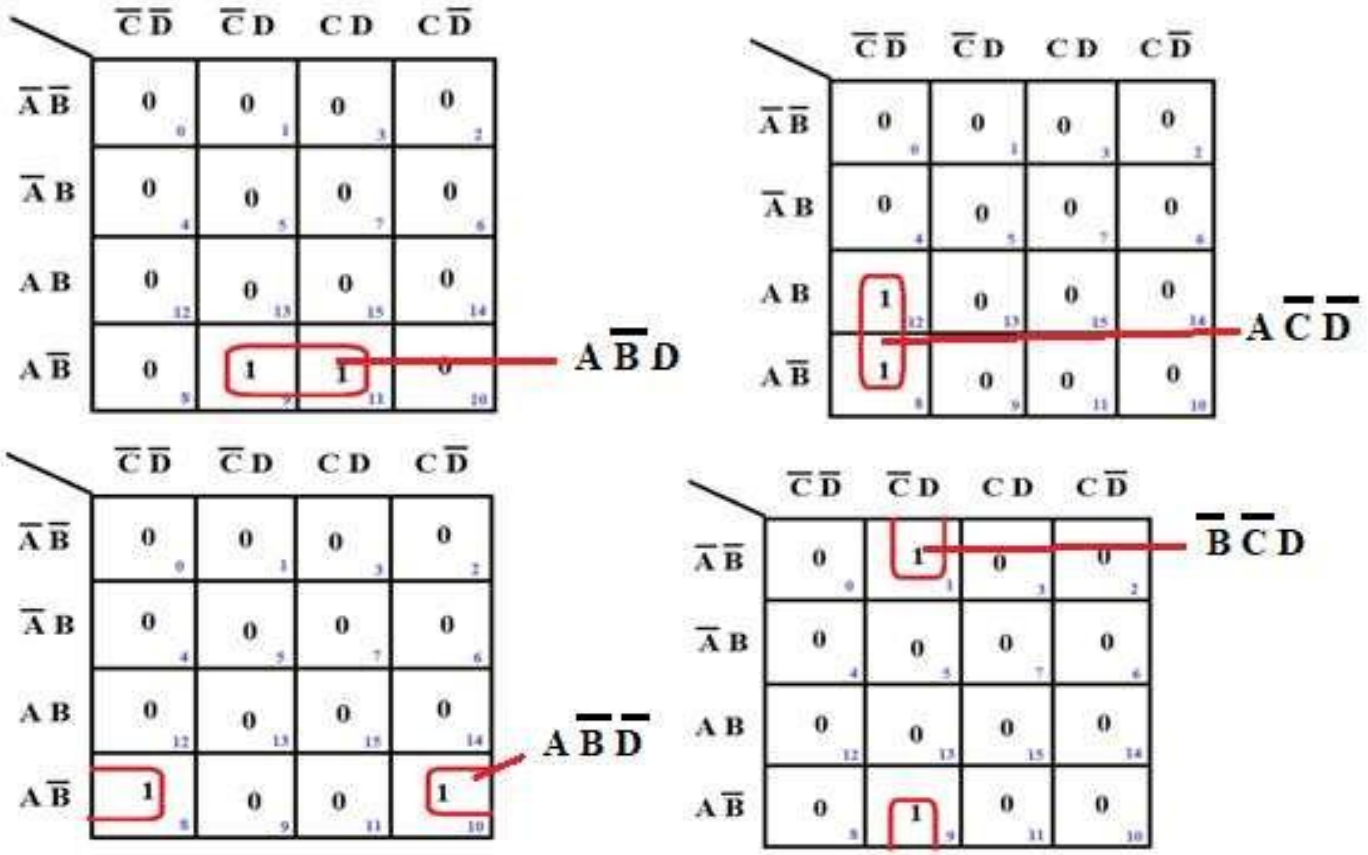
$Y = F(A, B, C, D) = \Sigma\, m(1,6,7,14)$

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

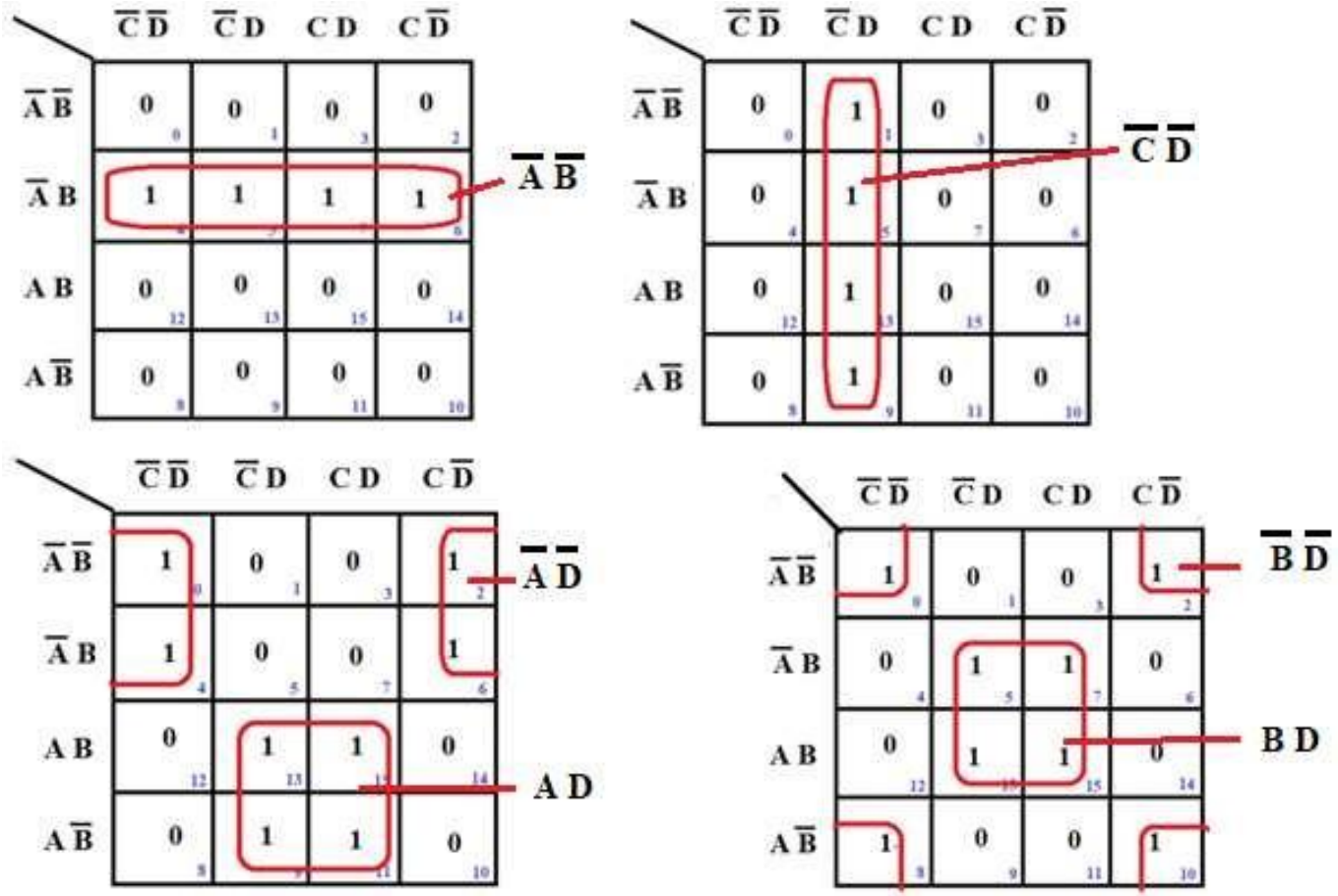
	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	0 (0)	1 (1)	0 (3)	0 (2)
$\overline{A}B$	0 (4)	0 (5)	1 (7)	1 (6)
AB	0 (12)	0 (13)	0 (15)	1 (14)
$A\overline{B}$	0 (8)	0 (9)	0 (11)	0 (10)

Pairs, Quads and Octets

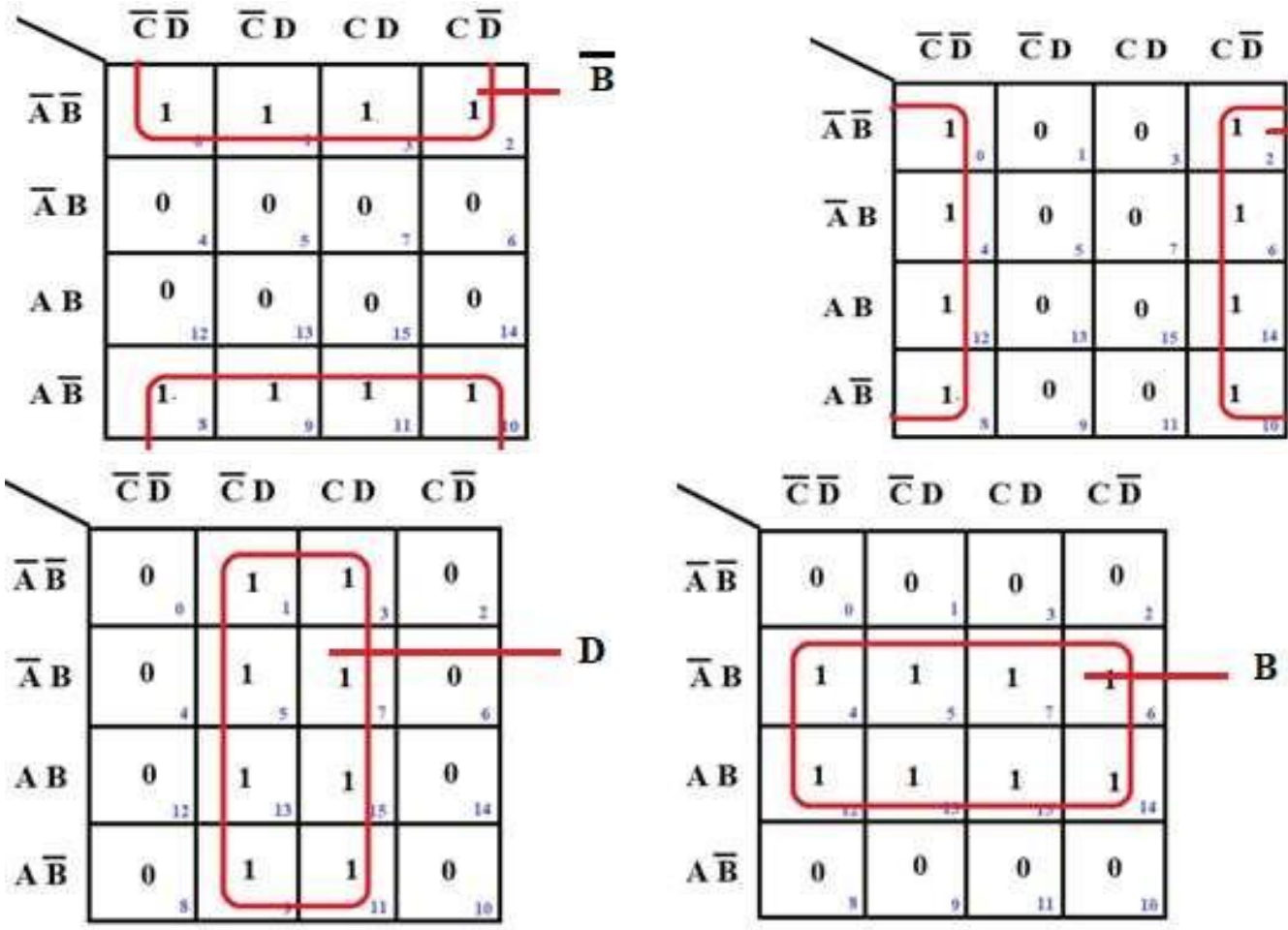
- Two adjacent 1's in the K-map is called a ***pair*** and it eliminate one ***variable.***



- A ***quad*** is a group of four l's that are horizontally or vertically adjacent and a quad eliminates two variables and their complements.

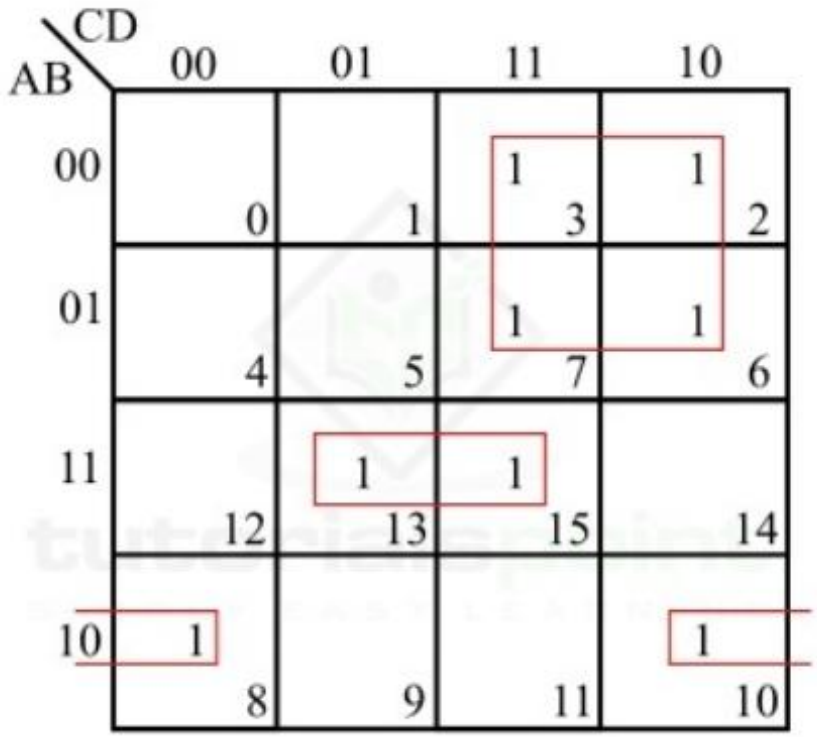


- An ***octet*** is a group of 8 ls that are horizontally or vertically adjacent and an octet eliminates three variables and their complements.



Simplify the following Boolean expression using the 4-variable K-map.

$$f(A,B,C,D) = \sum m(2,3,6,7,8,10,13,15)$$

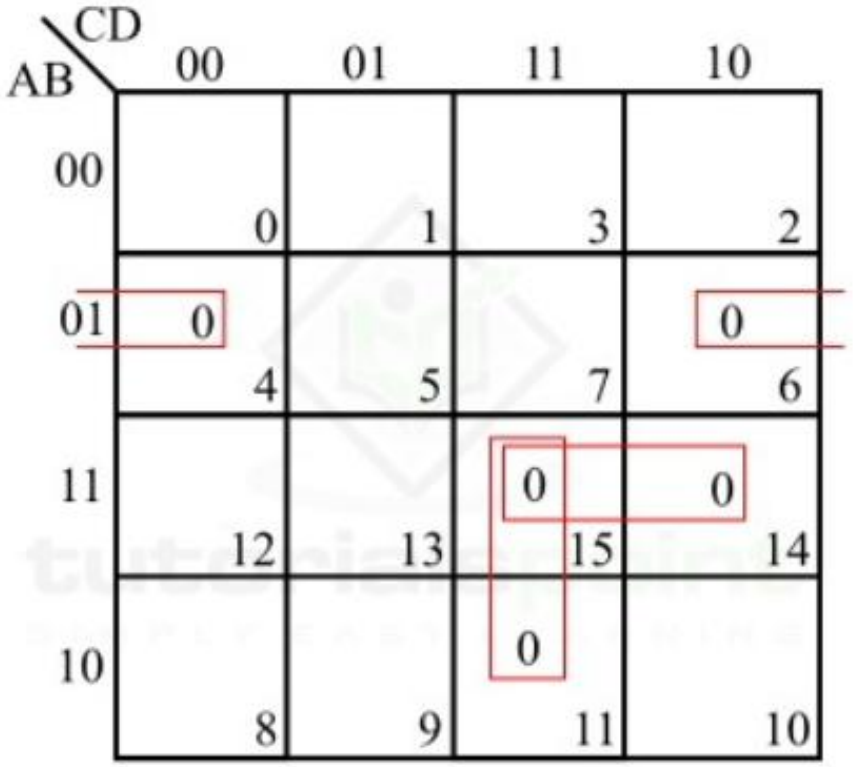


So the simplified SOP expression is,

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

Minimize the following Boolean expression using the 4-variable K-map.

$$f(A,B,C,D) = \prod M(4,6,11,14,15)$$



So, the reduced POS expression of the given Boolean function is,

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

Determination of Minimum Expressions using Essential Prime Implicants

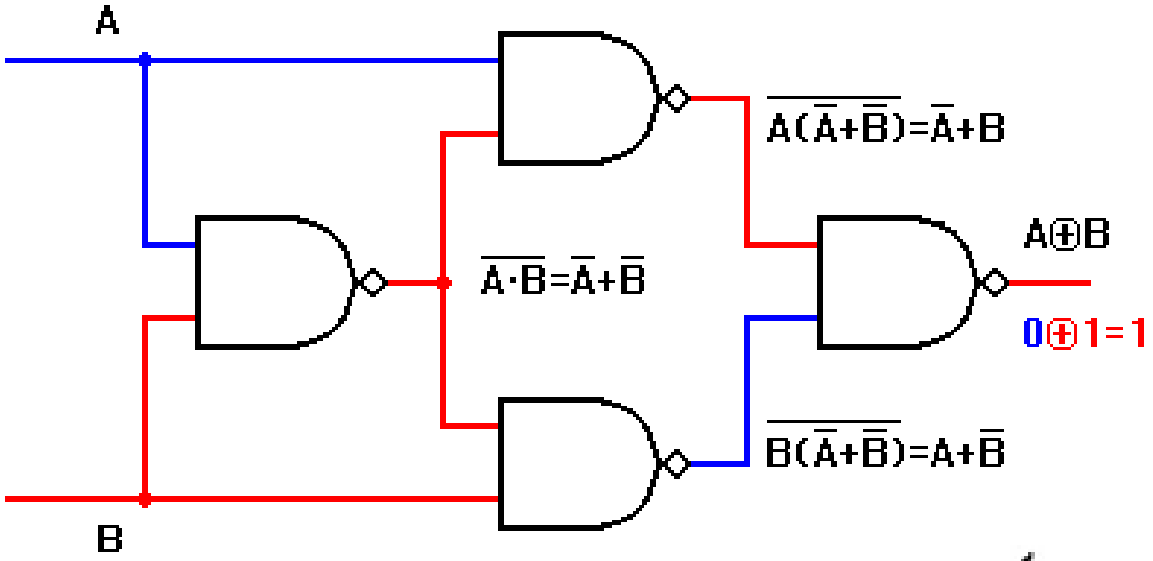
- Any single 1 or any group of 1's which can be combined together on a map of the function F represents a product term which is called an implicant of F.

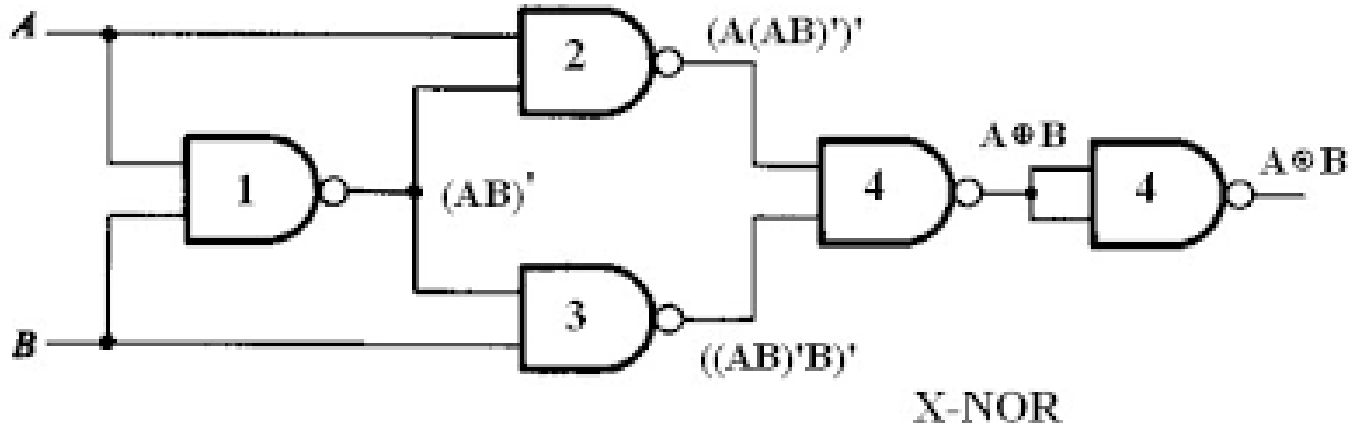
- A product term implicant is called a prime implicant if it cannot be combined with another term to eliminate a variable.

The following procedure can then be used to obtain a minimum sum of products from a Karnaugh map

1) Choose a minterm (a 1) which has not yet been covered.
2) Find all 1's and X's adjacent to that minterm.
3) If a single term covers the minterm and all of the adjacent 1's and X's, then that term is an essential prime implicant, so select that term
4) Repeat steps 1, 2, and 3 until all essential prime implicants have been chosen.
5) Find a minimum set of prime implicants which cover the remaining 1's on the map.

XOR & XNOR gates using NAND gates





VERILOG
Hardware Description Language

About Verilog

- Along with VHDL, Verilog is among the most widely used HDLs.
- Main differences:
  - VHDL was designed to support system-level design and specification.
  - Verilog was designed primarily for digital hardware designers developing FPGAs and ASICs.

Concept of Verilog "Module"

In Verilog, the basic unit of hardware is called a *module*.

Basic Syntax of Module Definition

```
module  module_name  (list_of_ports);

        input/output declarations;

        local net declarations;

        parallel statements;

endmodule
```

Example 1 :: simple AND gate

```
module  simpleand (f, x, y);
   input  x, y;
   output  f;
   assign  f = x & y;
endmodule
```

Verilog Language Features

Operators

Arithmetic Operators:

+	unary (sign) plus
–	unary (sign) minus
+	binary plus (add)
–	binary minus (subtract)
*	multiply
/	divide
%	modulus
**	exponentiation

Examples:

– (b + c)

(a – b) + (c * d)

(a + b) / (a – b)

a % b

a ** 3

Operators

Logical Operators:

!	logical negation
&&	logical AND
||	logical OR

Examples:

```
(done && ack)
(a || b)
! (a && b)
((a > b) || (c ==0))
((a > b) && ! (b > c))
```

Operators

Bitwise Operators:

~	bitwise NOT
&	bitwise AND
|	bitwise OR
^	bitwise exclusive-OR
~^	bitwise exclusive-NOR

Examples:

```
wire  a, b, c, d, f1, f2, f3, f4;
assign f1 = ~a | b;
assign f2 = (a & b) | (b & c) | (c & a);
assign f3 = a ^ b ^ c;
assign f4 = (a & ~b) | (b & c & ~d);
```

Operators

Relational Operators:

!=	not equal
==	equal
>=	greater or equal
<=	less or equal
>	greater
<	less

Examples:

(a != b)

((a + b) == (c – d))

((a > b) && (c < d))

(count <= 0)

Operators

Shift Operators:

>>	shift right
<<	shift left
>>>	arithmetic shift right

Examples:

```
wire [15:0] data, target;
assign target = data >> 3;
assign target = data >>> 2;
```

AND Gate Using Data Flow Modelling

```
module and_gate_d(a,b,y);
input a,b;
output y;

assign y = a & b;

endmodule
```

AND Gate Using Behavioral Modelling

```
module AND_gate_b(a,b,y);
input a,b;
output y;

always @ (a,b)
y = a & b;

endmodule
```

AND Gate Using Structural Modelling

```
module and_gate_s(a,b,y);
input a,b;
output y;

and(y,a,b);

endmodule
```

Data Flow

```
module and_gate_d(a,b,y);
input a,b;
output y;

assign y = a & b;

endmodule
```

Behavioral

```
module AND_gate_b(a,b,y);
input a,b;
output y;

always @ (a,b)
y = a & b;

endmodule
```

Structural

```
module and_gate_s(a,b,y);
input a,b;
output y;

and(y,a,b);

endmodule
```

Thank you