Module 2
Boolean Algebra And Logic Gates

Motivation:

Boole's algebra is based on logical relations. Hence the variables of Boole take only the values 0 and 1, with 0 standing for false and 1 for truth. Hence Boolean algebra is employed in digital logic design which incorporates binary numbers.

Syllabus:

Lecture no	Content	Duration (Hr)	Self-Study (Hrs)
1	Theorems and Properties of Boolean Algebra, Boolean functions, Boolean function reduction using Boolean laws	1	1
2	Canonical forms, Standard SOP and POS form	1	2
3	**Basic Digital gates:** NOT , AND , OR , NAND , NOR , EXOR , EXNOR, positive and negative logic	1	1
4	K-map method 2 variable, 3 variable	1	2
5	K-map method 4 variable, Don't care condition	1	2
6	Quine-McClusky Method	1	2
7	NAND,NOR Realization	1	2

Learning Objectives:

Learners shall be able to:

1. To explain the theorems and properties of boolean algebra
2. To apply Boolean laws for reduction of Boolean functions.
3. To convert Boolean expressions in canonical form to SOP/POS form.
4. To apply K-map method to reduce logical expressions
5. To apply Quine-McClusky method to reduce logical expressions
6. To realize basic gates using universal gates.

Theoretical Background:

Boolean algebra is a method of simplifying the logic circuits (or sometimes called as logic switching circuits) in digital electronics. So it is also called as “Switching algebra”. We can represent the functioning of logic circuits by using numbers, by following some rules, which are well known as "Laws of Boolean algebra".

We can also make the calculations and logical operations of the circuits even faster by following some theorems, which are known as "Theorems of Boolean Algebra". A Boolean function is a function which represents the relation between the input and output of a logic circuit.

Boolean logic allows only two states of the circuit, such as True and False. These two states are represented by 1 and 0, where 1 represents the state "True" and 0 represents the state "False". The most important thing to remember in Boolean algebra is that it is very much different than regular mathematical algebra and its methods.

Key Definitions:

Karnaugh maps: A mapping technique used to minimize logical expression

Minterm: A logical term consisting of all the literals in the ANDed form.

Maxterm: A logical term consisting of all the literals in the ORed form in the logic function

Canonical Form: A term used to describe a Boolean function that is written either as a sum of minterms, or as a product of maxterms. For example, using three variables A, B, and C, the equation $f(A, B, C) = A(B \sim C) + ABC$

Product of Sum (POS): A logical expression in the form of ORed terms ANDed together

Sum of Product (SOP): A logical expression in the form of ANDed terms ORed together

Literal: A Literal is a Boolean Variables or its complement

Combinational logic: The logic in which the outputs at any instant of time are dependent only on the inputs present at that time

Standard POS: If each term in POS form contains all the literals then the POS form is known as standard or canonical POS form

Standard SOP: If each term in SOP form contains all the literals then the SOP form is known as standard or canonical SOP form.

Don’t care: A minterm/maxterm in a logic function may or may not be included.

Course Content:

Lecture 1

2.1 Boolean Algebra

In Boolean algebra, the variables are represented by English Capital Letter like A, B, C etc and the value of each variable can be either 1 or 0, nothing else. In Boolean algebra an expression given can also be converted into a logic diagram using different logic gates like AND gate, OR gate and NOT gate, NOR gates, NAND gates, XOR gates, XNOR gates etc.

Boolean theorems and **laws** are used to simplify the various logical expressions. In a digital designing problem, a unique logical expression is evolved from the truth table. If this logical expression is simplified the designing becomes easier.

Properties of Boolean algebra:

Annulment law – a variable ANDed with 0 gives 0, while a variable ORed with 1 gives 1, i.e.,

A.0 = 0

A + 1 = 1

Identity law – in this law variable remain unchanged it is ORed with ‘0’ or ANDed with ‘1’, i.e.,

A.1 = A

A + 0 = A

Idempotent law – a variable remain unchanged when it is ORed or ANDed with itself, i.e.,

A + A = A

A.A = A

Complement law – in this Law if a complement is added to a variable it gives one, if a variable is multiplied with its complement it results in ‘0’, i.e.,

A + A’ = 1

A.A’ = 0

Double negation law – a variable with two negation its symbol gets cancelled out and original variable is obtained, i.e.,

((A)’)’=A

Commutative law – a variable order does not matter in this law, i.e.,

A + B = B + A

A.B = B.A

Associative law – the order of operation does not matter if the priority of variables are same like ‘*’ and ‘/’, i.e.,

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

A.(B.C) = (A.B).C

Distributive law – this law governs opening up of brackets, i.e.,

A.(B+C) = (A.B)+(A.C)

A+(B.C) = (A+B).(A+C)

Absorption law –This law involved absorbing the similar variables, i.e.,

A.(A+B) = A

A + AB = A

De Morgan law – the operation of an AND or OR logic circuit is unchanged if all inputs are inverted, the operator is changed from AND to OR, the output is inverted, i.e.,

(A.B)’ = A’ + B’

(A+B)’ = A’.B’

Boolean function reduction using Boolean laws

- Complex combinational logic circuits must be reduced without changing the function of the circuit.
- Reduction of a logic circuit means the same logic function with fewer gates and/or inputs.
- The first step to reducing a logic circuit is to write the Boolean Equation for the logic function.
- The next step is to apply as many rules and laws as possible in order to decrease the number of terms and variables in the expression.
- To apply the rules of Boolean Algebra it is often helpful to first remove any parentheses or brackets.
- After removal of the parentheses, common terms or factors may be removed leaving terms that can be reduced by the rules of Boolean Algebra.
- The final step is to draw the logic diagram for the reduced Boolean Expression.

Examples of Simplification:

Example 1

Perform FOIL (Firt - Outer - Inner - Last)

AA = A (Anything ANDed with itself is itself)

Find a like term (A) and pull it out. (There is an A in A, AC, and AB). Make sure you leave the BC alone at the end.

Anything ORed with a 1 is a 1 (1+C+B=1).

Anthing ANDed with a 1 is itself (A1=A)

$(A + B)(A + C) = AA + AC + AB + BC$

$=A+AC+AB+BC$

$= A(1 + C + B) + BC$

$=A.1+ BC$

$= A + BC$

Example 2

Find like term (B) and pull it out.

Anything ORed with its own complement equals 1.

Anything ANDed with 1 is itself.

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

$= B (1)$

$= B$

Example 3

Find like term and pull them out. Make sure you leave the one.

Anything ORed with a 1 is 1.

Anything ANDed with a 1 is itself

$AC'+ABC' = AC' (1+B)$

$= AC' (1)$

$=AC'$

Example 4

Find like terms and pull them out.

Anything ORed with its own complement equals 1.

Anything ANDed with 1 equals itself.

$AB'D+AB'D' = AB' (D+D')$

$= AB' (1)$

$= AB'$

DeMorgan's Theorem

- De Morgan's theorem allows large bars in a Boolean Expression to be broken up into smaller bars over individual variables.
- De Morgan's theorem says that a large bar over several variables can be broken between the variables if the sign between the variables is changed.
- De Morgan's theorem can be used to prove that a NAND gate is equal to an OR gate with inverted inputs.
- De Morgan's theorem can be used to prove that a NOR gate is equal to an AND gate with inverted inputs.
- In order to reduce expressions with large bars, the bars must first be broken up. This means that in some cases, the first step in reducing an expression is to use De Morgan's theorem.
- It is highly recommended to place parentheses around terms where lines have been broken.

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

Example 1

Apply DeMorgan’s theorem to remove the overbar covering both terms from the expression

$X = (C'+D)'$

$X = C''.D'$

$X =C.D'$

Example 2

X = C (A+B)’ + D

Applying DeMorgan's theorem and the distribution law:

X = C (A’.B’) + D = A’B’C + D

Let’s check the take away from this lecture

1) Boolean algebra can be used ____________

a) For designing of the digital computers

b) In building logic symbols

c) Circuit theory

d) Building algebraic functions

2) A ________ value is represented by a Boolean expression.

a) Positive

b) Recursive

c) Negative

d) Boolean

3) Which of the following is a Simplification law?

a) M.(~M+N) = M.N

b) M+(N.O) = (M+N)(M+O)

c) ~(M+N) = ~M.~N

d) M.(N.O) = (M.N).O

Exercise
Q.1 Discuss the different properties of Boolean algebra.
Q.2 State DeMorgan's theorem.
Q.3 Simplify the following expression:
F= A+ A'B + A'B'C + A'B'C'D + A'B'C'D'E.

Learning from this lecture: Learners will be able to understand the properties of Boolean algebra and simplify given Boolean expressions using Boolean laws.

Lecture 2

Canonical forms, Standard SOP and POS form

Canonical Form – In Boolean algebra, Boolean function can be expressed as Canonical Disjunctive Normal Form known as **minterm** and some are expressed as Canonical Conjunctive Normal Form known as **maxterm** .

In Minterm, we look for the functions where the output results in "1" while in Maxterm we look for function where the output results in "0".

- We perform **Sum of minterm** also known as Sum of products (SOP).
- We perform **Product of Maxterm** also known as Product of sum (POS).

Boolean functions expressed as a sum of minterms or product of maxterms are said to be in canonical form.

Standard Form – A Boolean variable can be expressed in either true form or complemented form. In standard form Boolean function will contain all the variables in either true form or complemented form while in canonical number of variables depends on the output of SOP or POS.

A Boolean function can be expressed algebraically from a given truth table by forming a :

- minterm for each combination of the variables that produces a 1 in the function and then taking the AND of all those terms.
- maxterm for each combination of the variables that produces a 0 in the function and then taking the OR of all those terms.

Truth table representing minterm and maxterm –

				Minterms		*Maxterms*
X	*Y*	*Z*		*Product Terms*		*Sum Terms*
0	*0*	*0*		$m_o = X'.Y'.Z'$ = min (X',Y',Z')		$M_0 = X+Y+Z$ = max (X,Y,Z)
0	*0*	*1*		$m_l = X'.Y'.Z$ = min (X',Y',Z)		$M_1 = X+Y+Z'$ = max (X,Y,Z')
0	1	*0*		$m_2 = X'.Y.Z'$ = min (X',Y,Z')		$M_2 = X+Y'+Z$ = max (X,Y',Z)
0	1	1		$m_3 = X'.Y.Z = min(X',Y,Z)$		$M_3 = X+Y'+Z'$ = max (X,Y',Z')
1	*0*	*0*		$m_4 = X.Y'.Z'$ = min (X,Y',Z')		$M_4 = X'+Y+Z$ = max (X',Y,Z)
1	*0*	1		$m_a = X.Y'.Z$ = min (X,Y',Z)		$M_5 = X'+Y+Z' = max(X',Y,Z')$
1	1	*0*		$m_6 = X.Y.Z' = min(X,Y,Z')$		$M_6 = X'+Y'+Z$ = max (X',Y',Z)
1	1	1		$m_7 = X.Y.Z$ = min (X,Y,Z)		$M_7 = X'+Y'+Z'$ = max (X',Y',Z')

From the above table it is clear that minterm is expressed in product format and maxterm is expressed in sum format.

Sum of minterms –

The minterms whose sum defines the Boolean function are those which give the 1's of the function in a truth table. Since the function can be either 1 or 0 for each minterm, and since there are 2^n minterms, one can calculate all the functions that can be formed with n variables to be (2^(2^n)). It is sometimes convenient to express a Boolean function in its sum of minterm form.

Example – Express the Boolean function F = A + B'C as standard sum of minterms.

Solution –

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

This function is still missing one variable, so

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

The second term $B'C$ is missing one variable; hence,

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

Combining all terms, we have

$F = A + B'C = ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C$

But $AB'C$ appears twice, and

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

$F = A'B'C + AB'C' + AB'C + ABC' + ABC$

$= m1 + m4 + m5 + m6 + m7$

SOP is represented as $\sum(1, 4, 5, 6, 7)$

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

Solution –

$F = xy + x'z$

$= (xy + x')(xy + z)$

$= (x + x')(y + x')(x + z)(y + z)$

$= (x' + y)(x + z)(y + z)$

$x' + y = x' + y + zz'$

$= (x' + y + z)(x' + y + z')$ $x + z$

$= x + z + yy'$

$= (x + y + z)(x + y' + z)$ $y + z$

$= y + z + xx'$

$= (x + y + z)(x' + y + z)$

$F = (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z')$

$= M0*M2*M4*M5$

POS is represented as $\pi(0, 2, 4, 5)$

Example –

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

$F'(A, B, C) = \sum(0, 2, 3) = m0 + m2 + m3$

Now, if we take the complement of F' by DeMorgan's theorem, we obtain F in a different form:

$F = (m0 + m2 + m3)'$

$= m0'm2'm3'$

$= M0*M2*M3$

$= \pi(0, 2, 3)$

Example – Convert Boolean expression in standard form F=y’+xz’+xyz

Solution – F = (x+x’)y'(z+z’)+x(y+y’)z’ +xyz

F = xy’z+ xy’z’+x’y’z+x’y’z’+ xyz’+xy’z’+xyz

Let’s check the take away from this lecture

1) What are the canonical forms of Boolean Expressions?

a) OR and XOR

b) NOR and XNOR

c) MAX and MIN

d) SOM and POM

2) The ___________ of all the variables in direct or complemented from is a maxterm.

a) addition

b) product

c) moduler

d) subtraction

3) Which of the following expressions is in the sum-of-products (SOP) form?

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

b) (A)B(CD)

c) AB(CD)

d) AB + CD

Exercise

Q.1 What is Canonical form.

Q.2 Define minterm and maxterm.

Q.3 Convert the SOP expression F= xy' + yz' into its POS form.

Learning from this lecture: Learners will be able to understand the meaning of canonical form, SOP and POS expressions.

Lecture 3

2.2 Basic Digital Gates

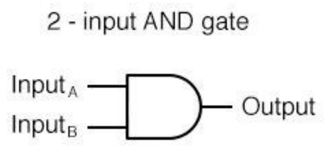
Basic Digital gates: NOT, AND, OR, NAND, NOR, EXOR, EXNOR, positive and negative logic

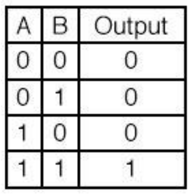
Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on certain logic. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc.

The AND Gate

One of the easiest multiple-input gates to understand is the AND gate, so-called because the output of this gate will be "high" (1) if and only if all inputs (first input and the second input and . . .) are "high" (1). If any input(s) is "low" (0), the output is guaranteed to be in a "low" state as well.

A two-input AND gate's truth table looks like this:





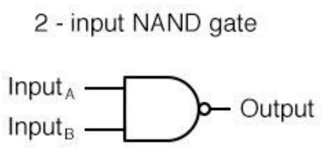
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

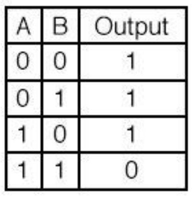
The NAND Gate

A variation on the idea of the AND gate is called the NAND gate. The word "NAND" is a verbal contraction of the words NOT and AND.

Essentially, a NAND gate behaves the same as an AND gate with a NOT (inverter) gate connected to the output terminal. To symbolize this output signal inversion, the NAND gate symbol has a bubble on the output line.

The truth table for a NAND gate is as one might expect, exactly opposite as that of an AND gate:





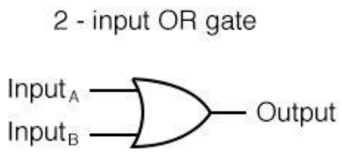
A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

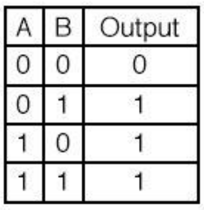
As with AND gates, NAND gates are made with more than two inputs. In such cases, the same general principle applies: the output will be “low” (0) if and only if all inputs are “high” (1). If any input is “low” (0), the output will go “high” (1).

The OR Gate

Our next gate to investigate is the OR gate, so-called because the output of this gate will be “high” (1) if any of the inputs (first input or the second input or . . .) are “high” (1). The output of an OR gate goes “low” (0) if and only if all inputs are “low” (0).

A two-input OR gate’s truth table looks like this:



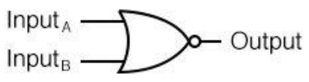


A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

The NOR Gate

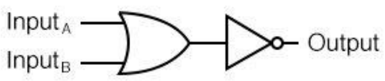
As you might have suspected, the NOR gate is an OR gate with its output inverted, just like a NAND gate is an AND gate with an inverted output.

2 - input NOR gate



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

Equivalent Gate Circuit



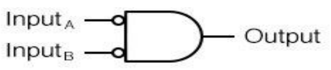
NOR gates, like all the other multiple-input gates seen thus far, can be manufactured with more than two inputs. Still, the same logical principle applies: the output goes “low” (0) if any of the inputs are made “high” (1). The output is “high” (1) only when all inputs are “low” (0).

The Negative-AND Gate

A Negative-AND gate functions the same as an AND gate with all its inputs inverted (connected through NOT gates). In keeping with standard gate symbol convention, these inverted inputs are signified by bubbles.

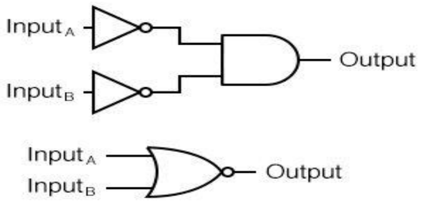
Contrary to most peoples’ first instinct, the logical behavior of a Negative-AND gate is not the same as a NAND gate. Its truth table, actually, is identical to a NOR gate:

2 - input Negative-AND gate



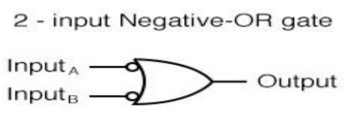
A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

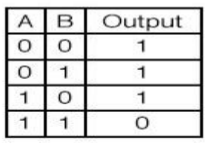
Equivalent Gate Circuits



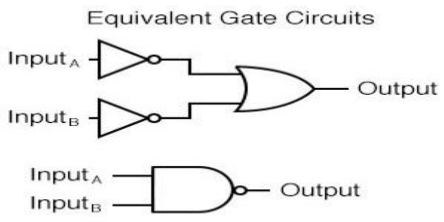
The Negative-OR Gate

Following the same pattern, a Negative-OR gate functions the same as an OR gate with all its inputs inverted. In keeping with standard gate symbol convention, these inverted inputs are signified by bubbles. The behavior and truth table of a Negative-OR gate is the same as for a NAND gate:





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

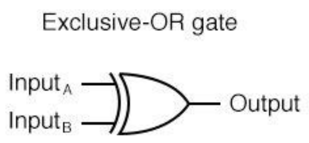


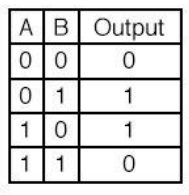
The Exclusive-OR Gate

The last six gate types are all fairly direct variations on three basic functions: AND, OR, and NOT. The Exclusive-OR gate, however, is something quite different.

Exclusive-OR gates output a “high” (1) logic level if the inputs are at different logic levels, either 0 and 1 or 1 and 0. Conversely, they output a “low” (0) logic level if the inputs are at the same logic levels.

The Exclusive-OR (sometimes called XOR) gate has both a symbol and a truth table pattern that is unique:



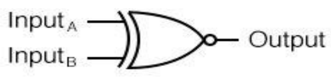


A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

The Exclusive-NOR Gate

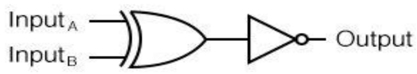
Finally, our last gate for analysis is the Exclusive-NOR gate, otherwise known as the XNOR gate. It is equivalent to an Exclusive-OR gate with an inverted output. The truth table for this gate is exactly opposite as that of the Exclusive-OR gate:

Exclusive-NOR gate



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

Equivalent Gate Circuit



As indicated by the truth table, the purpose of an Exclusive-NOR gate is to output a "high" (1) logic level whenever both inputs are at the same logic levels (either 00 or 11).

Let's check the take away from this lecture

1) Which of the following is/are the universal logic gates?

a) OR and NOR

b) AND

c) NAND and NOR

d) NOT

2) The logic gate that provides high output for same inputs ____________

a) NOT

b) X-NOR

c) AND

d) XOR

3) The NOR gate output will be high if the two inputs are __________

a) 00

b) 01

c) 10

d) 11

Exercise
Q.1 What is Canonical form.
Q.2 Define minterm and maxterm.
Q.3 Convert the SOP expression F= xy' + yz' into its POS form.

Learning from this lecture: Learners will be able to understand the working of logic gates.

Lecture 4

K-map method 2 variable, 3 variable

The Karnaugh map or k-map is a graphical technique for simplifying Boolean function. The k-map in a two dimensional representation of a truth table it provides a simpler method for minimizing logic expressions. The map method is ideally suited for four or less variables. But it becomes cumbersome for five or more variable. A karnaugh map is a diagram consisting of squares. Each square of the map represent either min term or max term. Any logic expression can be written as either sum of product term or product of sum term which is also called sum of min term and product of max term respectively. Therefore a logic expression can be easily represented on a karnaugh map.

A Karnaugh map for n variables made up of 2^n squares. Each square designate a product term in Boolean expression. For product term which are present in the expression. In case of sum of product, 1s can be written in Present Square and in case of product of sum term, 0s can be written in Present Square. Blank Square or opposite square indicate. That then contains nothing.

We can construct Karnaugh map in two different forms for sum of product term and for product of sum term. Because in sum of product contains zero as complement value and one as un-complement value similarly in the product of sum term we use 1 as complement and 0 as un-complemented value function of both sum of product and product of sum about same except that the complement and un-complement value.

2 variable k-map: – In the 2 variable k-map, four squares are constructed. Each square contains one term of expression with two variables. A 2 variable k-map for SOP and POS form are as follows: –

A. SOP: -

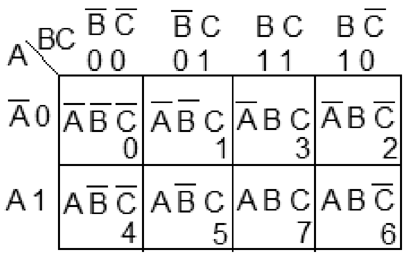
A \ B	$\overline{B}$ 0	B 1
$\overline{A}$ 0	$\overline{A}.\overline{B}$	$\overline{A}.B$
A 1	$A.\overline{B}$	A.B

B. POS: -

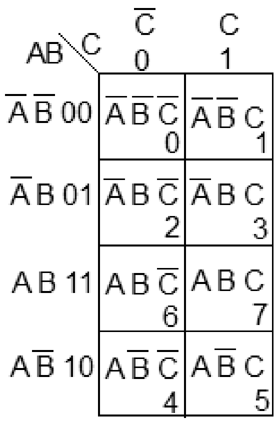
A \ B	B 0	$\overline{B}$ 1
A 0	A+B	$A+\overline{B}$
$\overline{A}$ 1	$\overline{A}+B$	$\overline{A}+\overline{B}$

3 variables k-map: – In the three variable k-map, 8 square is required. The 3-variables k-map can content either horizontally or vertically. An example of 3 variables k-map for SOP and POS form are as follows: –

A. SOP: -

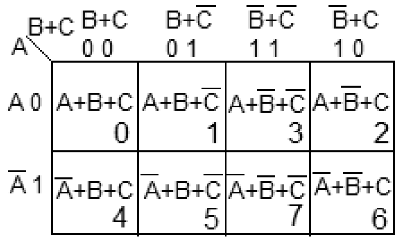


A \ BC	$\overline{B}\,\overline{C}$ 00	$\overline{B}C$ 01	BC 11	$B\overline{C}$ 10
$\overline{A}$ 0	$\overline{A}\,\overline{B}\,\overline{C}$ 0	$\overline{A}\,\overline{B}C$ 1	$\overline{A}BC$ 3	$\overline{A}B\overline{C}$ 2
A 1	$A\overline{B}\,\overline{C}$ 4	$A\overline{B}C$ 5	ABC 7	$AB\overline{C}$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

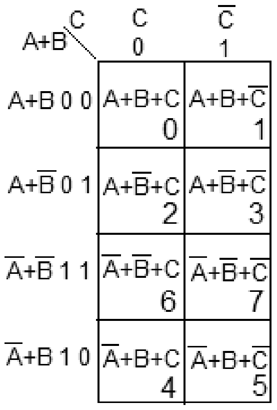


AB \ C	$\overline{C}$ 0	C 1
$\overline{A}\,\overline{B}$ 00	$\overline{A}\,\overline{B}\,\overline{C}$ 0	$\overline{A}\,\overline{B}C$ 1
$\overline{A}B$ 01	$\overline{A}B\overline{C}$ 2	$\overline{A}BC$ 3
AB 11	$AB\overline{C}$ 6	ABC 7
$A\overline{B}$ 10	$A\overline{B}\,\overline{C}$ 4	$A\overline{B}C$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

B. POS: -



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



A+B \ C	C 0	$\overline{C}$ 1
$A+B$ 0 0	$A+B+C$ 0	$A+B+\overline{C}$ 1
$A+\overline{B}$ 0 1	$A+\overline{B}+C$ 2	$A+\overline{B}+\overline{C}$ 3
$\overline{A}+\overline{B}$ 1 1	$\overline{A}+\overline{B}+C$ 6	$\overline{A}+\overline{B}+\overline{C}$ 7
$\overline{A}+B$ 1 0	$\overline{A}+B+C$ 4	$\overline{A}+B+\overline{C}$ 5

Let's check the take away from this lecture

1) A Karnaugh map (K-map) is an abstract form of ____________ diagram organized as a matrix of squares.

a) Venn Diagram

b) Cycle Diagram

c) Block diagram

d) Triangular Diagram

2) Each "1" entry in a K-map square represents:

a) HIGH for each input Truth Table condition that produces a HIGH output.

b) LOW output for all possible HIGH input conditions.

c) HIGH output on the Truth Table for all LOW input combinations.

d) DON'T CARE condition for all possible input Truth Table combinations.

3) An n variable K-map can have

a) n^2 cells

b) 2^n cells

c) n^n cells

d) $n2^n$ cells

Exercise

Q.1 What is the use of K-map in digital logic design.

Q.2 Minimize the following Boolean function using sum of products (SOP): f (A,B) = ∑m(0,1,3)

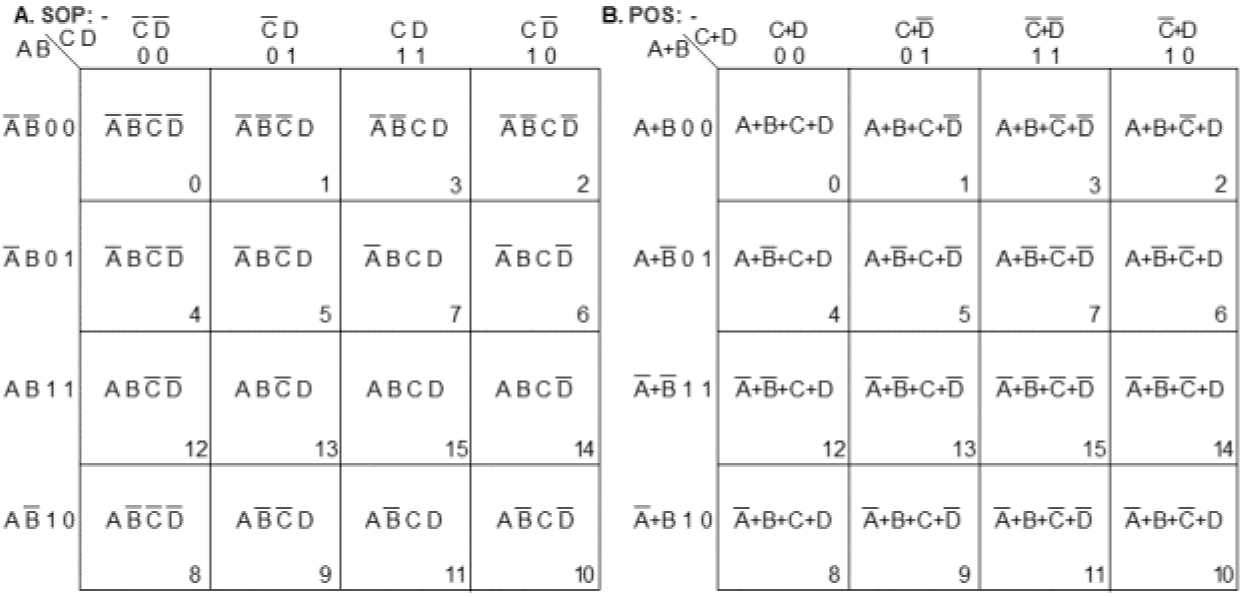
Q.3 Minimize the following Boolean function using sum of products (SOP): f(A,B,C) = ∑m(0,3,5).

Learning from this lecture: Learners will be able to understand how Boolean expressions can be reduced using K-map for 2 variable and 3 variable.

Lecture 5

K-map method 4 variable, Don’t care condition

4 variable k-map: – In the 4 variable k-map, 16 square are required. **An example of 4 variable k-map for SOP and POS form are as follows: –**



A. SOP: -

AB \ CD	$\bar C\bar D$ 00	$\bar C D$ 01	CD 11	$C\bar D$ 10
$\bar A\bar B$ 00	$\bar A\bar B\bar C\bar D$ 0	$\bar A\bar B\bar C D$ 1	$\bar A\bar B CD$ 3	$\bar A\bar B C\bar D$ 2
$\bar A B$ 01	$\bar A B\bar C\bar D$ 4	$\bar A B\bar C D$ 5	$\bar A BCD$ 7	$\bar A BC\bar D$ 6
AB 11	$AB\bar C\bar D$ 12	$AB\bar C D$ 13	$ABCD$ 15	$ABC\bar D$ 14
$A\bar B$ 10	$A\bar B\bar C\bar D$ 8	$A\bar B\bar C D$ 9	$A\bar B CD$ 11	$A\bar B C\bar D$ 10

B. POS: -

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

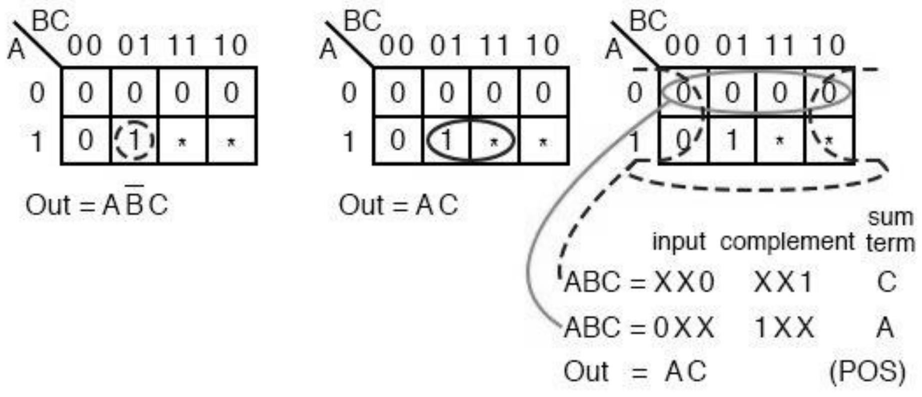
Don’t Cares

Don’t cares in a Karnaugh map, or truth table, may be either **1**s or **0**s, as long as we don’t care what the output is for an input condition we never expect to see. We plot these cells with an asterisk, *, among the normal **1**s and **0**s.

When forming groups of cells, treat the don’t care cell as either a **1** or a **0**, or ignore the don’t cares.

This is helpful if it allows us to form a larger group than would otherwise be possible without the don’t cares. There is no requirement to group all or any of the don’t cares.

Only use them in a group if it simplifies the logic.



Above is an example of a logic function where the desired output is **1** for input **ABC = 101** over the range from **000 to 101**. We do not care what the output is for the other possible inputs (**110, 111)**. Map those two as don't cares. We show two solutions.

The solution on the right Out = AB'C is the more complex solution since we did not use the don't care cells. The solution in the middle, Out=AC, is less complex because we grouped a don't care cell with the single **1** to form a group of two.

The third solution, a Product-Of-Sums on the right, results from grouping a don't care with three zeros forming a group of four **0**s. This is the same, less complex, **Out=AC**.

Let's check the take away from this lecture

1) In K-map a quad is a group of ------------- 1's.

a) 2

b) 6

c) 4

d) 8

2) It should be kept in mind that don't care terms should be used along with the terms that are present in ____________

a) Minterms

b) Expressions

c) K-Map

d) Latches

3) There are ______ cells in a 4-variable K-map.

a) 12

b) 16

c) 18

d) 8

Exercise
Q.1 State the role of don’t care in K-map.
Q.2 Simplify the expression Y=A'B'C' D'+A' B' CD'+A' BCD'+A' BCD+AB' C' D'+ABCD'+ABCD using K-map
Q.3 Minimize f = m(1,5,6,12,13,14) + d(4) in SOP minimal form.

Learning from this lecture: Learners will be able to understand how Boolean expressions can be reduced using K-map for 4 variable and don’t care condition.

Lecture 6

Quine-McClusky Method

Quine-McClukey tabular method is a tabular method based on the concept of prime implicants. We know that prime implicant is a product or sum term, which can’t be further reduced by combining with any other product or sum terms of the given Boolean function.

This tabular method is useful to get the prime implicants by repeatedly using the following Boolean identity.

xy + xy’ = xy+y′y+y′ = x.1 = x

Procedure of Quine-McCluskey Tabular Method

Follow these steps for simplifying Boolean functions using Quine-McClukey tabular method.

Step 1 − Arrange the given min terms in an ascending order and make the groups based on the number of ones present in their binary representations. So, there will be at most ‘n+1’ groups if there are ‘n’ Boolean variables in a Boolean function or ‘n’ bits in the binary equivalent of min terms.

Step 2 − Compare the min terms present in successive groups. If there is a change in only one-bit position, then take the pair of those two min terms. Place this symbol ‘_’ in the differed bit position and keep the remaining bits as it is.

Step 3 − Repeat step2 with newly formed terms till we get all prime implicants.

Step 4 − Formulate the prime implicant table. It consists of set of rows and columns. Prime implicants can be placed in row wise and min terms can be placed in column wise. Place ‘1’ in the cells corresponding to the min terms that are covered in each prime implicant.

Step 5 − Find the essential prime implicants by observing each column. If the min term is covered only by one prime implicant, then it is essential prime implicant. Those essential prime implicants will be part of the simplified Boolean function.

Step 6 − Reduce the prime implicant table by removing the row of each essential prime implicant and the columns corresponding to the min terms that are covered in that essential prime implicant. Repeat step 5 for Reduced prime implicant table. Stop this process when all min terms of given Boolean function are over.

Example

Let us simplify the following Boolean function, f(W,X,Y,Z)=∑m(2,6,8,9,10,11,14,15)f(W,X,Y,Z)=∑m(2,6,8,9,10,11,14,15) using Quine-McClukey tabular method.

The given Boolean function is in sum of min terms form. It is having 4 variables W, X, Y & Z. The given min terms are 2, 6, 8, 9, 10, 11, 14 and 15. The ascending order of these min terms based on the number of ones present in their binary equivalent is 2, 8, 6, 9, 10, 11, 14 and 15. The following table shows these min terms and their equivalent binary representations.

Group Name	**Min terms**	**W**	**X**	**Y**	**Z**
GA1	2	0	0	1	0
	8	1	0	0	0
GA2	6	0	1	1	0
	9	1	0	0	1
	10	1	0	1	0
GA3	11	1	0	1	1
	14	1	1	1	0
GA4	15	1	1	1	1

The given min terms are arranged into 4 groups based on the number of ones present in their binary equivalents. The following table shows the possible merging of min terms from adjacent groups.

Group Name	**Min terms**	**W**	**X**	**Y**	**Z**
GB1	2,6	0	-	1	0
	2,10	-	0	1	0
	8,9	1	0	0	-
	8,10	1	0	-	0

GB2	6,14	-	1	1	0
	9,11	1	0	-	1
	10,11	1	0	1	-
	10,14	1	-	1	0
GB3	11,15	1	-	1	1
	14,15	1	1	1	-

The min terms, which are differed in only one-bit position from adjacent groups are merged. That differed bit is represented with this symbol, '-'. In this case, there are three groups and each group contains combinations of two min terms. The following table shows the possible merging of min term pairs from adjacent groups.

Group Name	**Min terms**	**W**	**X**	**Y**	**Z**
GB1	2,6,10,14	-	-	1	0
	2,10,6,14	-	-	1	0
	8,9,10,11	1	0	-	-
	8,10,9,11	1	0	-	-
GB2	10,11,14,15	1	-	1	-
	10,14,11,15	1	-	1	-

The successive groups of min term pairs, which are differed in only one-bit position are merged. That differed bit is represented with this symbol, '-'. In this case, there are two groups and each group contains combinations of four min terms. Here, these combinations of 4 min terms are available in two rows. So, we can remove the repeated rows. The reduced table after removing the redundant rows is shown below.

Group Name	**Min terms**	**W**	**X**	**Y**	**Z**
GC1	2,6,10,14	-	-	1	0
	8,9,10,11	1	0	-	-
GC2	10,11,14,15	1	-	1	-

Further merging of the combinations of min terms from adjacent groups is not possible, since they are differed in more than one-bit position. There are three rows in the above table. So, each row will give one prime implicant. Therefore, the prime implicants are YZ’, WX’ & WY.

The prime implicant table is shown below.

Min terms / Prime Implicants	**2**	**6**	**8**	**9**	**10**	**11**	**14**	**15**
YZ’	1	1			1		1	
WX’			1	1	1	1		
WY					1	1	1	1

The prime implicants are placed in row wise and min terms are placed in column wise. 1s are placed in the common cells of prime implicant rows and the corresponding min term columns.

The min terms 2 and 6 are covered only by one prime implicant YZ’. So, it is an essential prime implicant. This will be part of simplified Boolean function. Now, remove this prime implicant row and the corresponding min term columns. The reduced prime implicant table is shown below.

Min terms / Prime Implicants	**8**	**9**	**11**	**15**
WX’	1	1	1	
WY			1	1

The min terms 8 and 9 are covered only by one prime implicant WX’. So, it is an essential prime implicant. This will be part of simplified Boolean function. Now, remove this prime implicant row and the corresponding min term columns. The reduced prime implicant table is shown below.

Min terms / Prime Implicants	**15**
WY	1

The min term 15 is covered only by one prime implicant WY. So, it is an essential prime implicant. This will be part of simplified Boolean function.

In this example problem, we got three prime implicants and all the three are essential. Therefore, the simplified Boolean function is

f W,X,Y,Z = YZ’ + WX’ + WY.

Let’s check the take away from this lecture

1) The Quine-McCluskey method can be used to

a) replace the K-map method

b) simplify expressions with 5 or more variables

c) both a and b

d) none of the above

2) Tabular method is also known as ___________

a) Karnaugh map

b) **Quine-McCluskey**

c) Prime Implicant

d) None of the above

3) The Quine– McClusky method of minimization of a logic expression is a (i) graphical method (ii) algebraic method (iii) tabular method (iv) a computer-oriented algorithm The correct answers are

a) (iii) and (iv)

b) (ii) and (iv)

c) (i) and (iii)

d) (i) and (ii)

Exercise

Q.1 Compare Quine-McCluskey method with K-map method.

Q.2 State the advantages of Quine-McCluskey method.

Q.3 Minimize f = m(1,5,6,12,13,14) + d(4) using Quine-McCluskey method.

Learning from this lecture: Learners will be able to understand how Boolean expressions can be reduced using Quine-McCluskey method.

Lecture 7

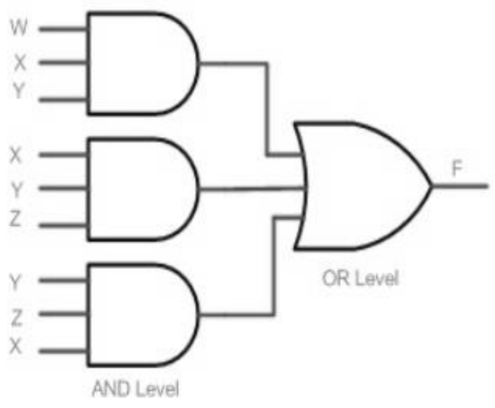
NAND, NOR Realization

Any logic function can be implemented using NAND gates. To achieve this, first the logic function has to be written in Sum of Product (SOP) form. Once logic function is converted to SOP, then is very easy to implement using NAND gate. In other words any logic circuit with AND gates in first level and OR gates in second level can be converted into a NAND-NAND gate circuit.

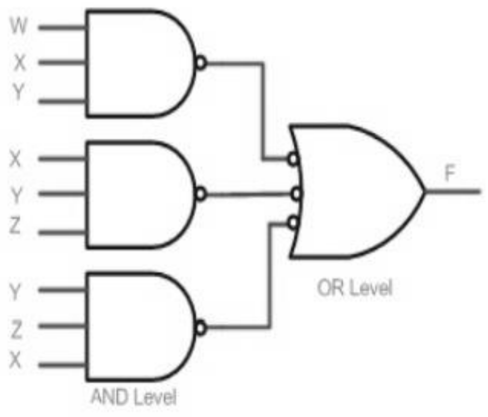
Consider the following SOP expression

F = W.X.Y + X.Y.Z + Y.Z.W

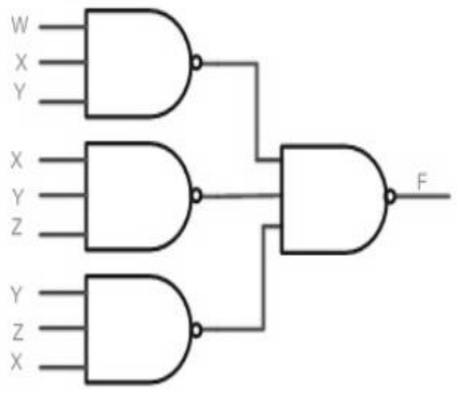
The above expression can be implemented with three AND gates in first stage and one OR gate in second stage as shown in figure.



If bubbles are introduced at AND gates output and OR gates inputs (the same for NOR gates), the above circuit becomes as shown in figure.



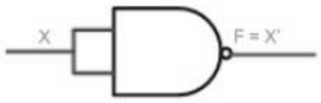
Now replace OR gate with input bubble with the NAND gate. Now we have circuit which is fully implemented with just NAND gates.



i) Realization of logic gates using NAND gates

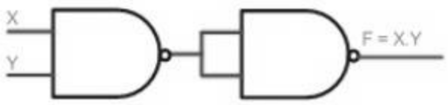
Implementing an inverter using NAND gate

Input	Output	Rule
(X.X)'	= X'	Idempotent



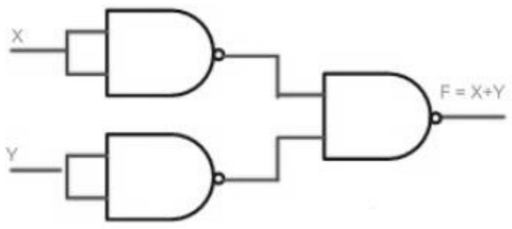
Implementing AND using NAND gates

Input	Output	Rule
((XY)'(XY)')'	= ((XY)')'	Idempotent
	= (XY)	Involution



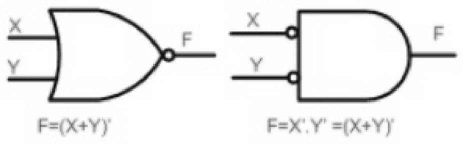
Implementing OR using NAND gates

Input	Output	Rule
((XX)'(YY)')'	= (X'Y')'	Idempotent
	= X''+Y''	DeMorgan
	= X+Y	Involution



Implementing NOR using NAND gates

Input	Output	Rule
((XX)'(YY)')'	=(X'Y')'	Idempotent
	=X''+Y''	DeMorgan
	=X+Y	Involution
	=(X+Y)'	Idempotent



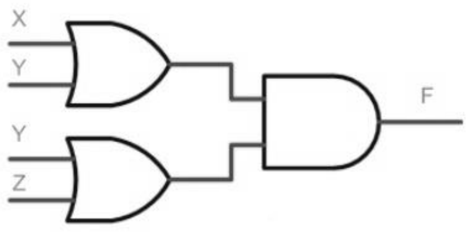
ii) Realization of logic function using NOR gates

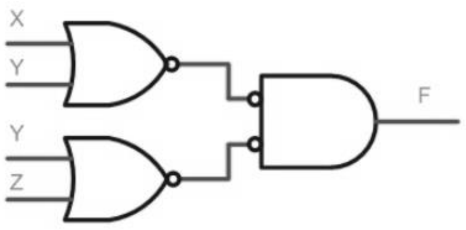
Any logic function can be implemented using NOR gates. To achieve this, first the logic function has to be written in Product of Sum (POS) form. Once it is converted to POS, then it's very easy to implement using NOR gate. In other words any logic circuit with OR gates in first level and AND gates in second level can be converted into a NOR-NOR gate circuit.

Consider the following POS expression

F = (X+Y) . (Y+Z)

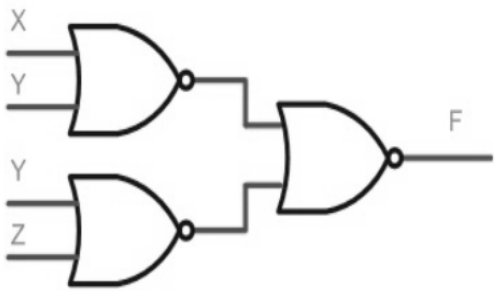
The above expression can be implemented with three OR gates in first stage and one AND gate in second stage as shown in figure.





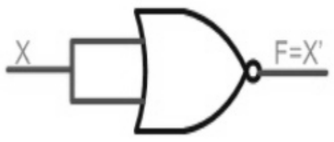
If bubble are introduced at the output of the OR gates and the inputs of AND gate, the above circuit becomes as shown in figure.

Now replace AND gate with input bubble with the NOR gate. Now we have circuit which is fully implemented with just NOR gates.



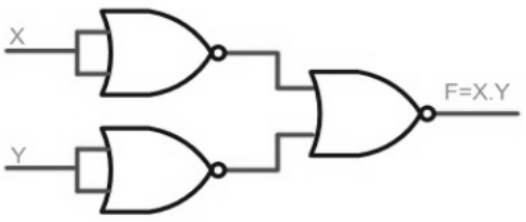
Implementing an inverter using NOR gate

Input	Output	Rule
(X+X)'	= X'	Idempotent



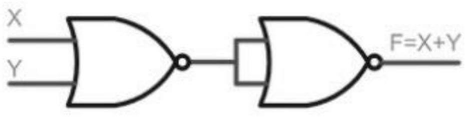
Implementing AND using NOR gates

Input	Output	Rule
((X+X)'+(Y+Y)')'	=(X'+Y')'	Idempotent
	= X''.Y''	DeMorgan
	= (X.Y)	Involution



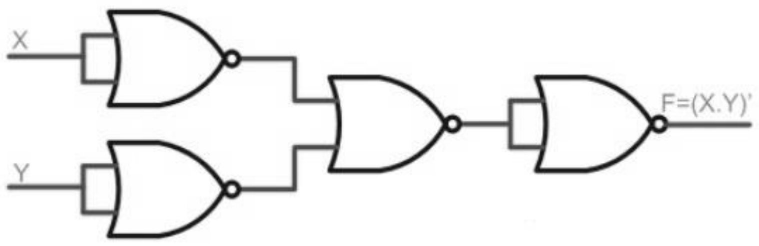
Implementing OR using NOR gates

Input	Output	Rule
((X+Y)'+(X+Y)')'	= ((X+Y)')'	Idempotent
	= X+Y	Involution



Implementing NAND using NOR gates

Input	Output	Rule
((X+Y)'+(X+Y)')'	= ((X+Y)')'	Idempotent
	= X+Y	Involution
	= (X+Y)'	Idempotent



Let's check the take away from this lecture

1) The inputs of a NAND gate are connected together. The resulting circuit is ………….

a) OR gate

b) AND gate

c) NOT gate

d) None of the above

2) The NOR gate is OR gate followed by ………………

a) AND gate

b) NAND gate

c) NOT gate

d) None of the above

3) The NAND gate is AND gate followed by …………………

a) NOT gate

b) OR gate

c) AND gate

d) None of the above

Exercise

Q.1 Realize AND gate using NAND gates only.

Q.2 Realize AND gate using NOR gates only.

Q.3 Realize OR gate using NOR gates only.

Learning from this lecture: Learners will be able to understand how basic gates can be implemented using universal gates.

Conclusion

The study of Boolean Algebra and Logic gates helps to understand how Boolean algebraic concepts are applied in digital logic design. Also the study of basic gates and universal gates provides the base for digital logic design.

Short Answer Questions:

1. What is Boolean algebra?

Ans) Boolean Algebra is used to analyze and simplify the digital (logic) circuits. It uses only the binary numbers i.e. 0 and 1. It is also called as Binary Algebra or logical Algebra. Boolean algebra was invented by George Boole in 1854.

2. What is Boolean expression?

Ans) A Boolean expression always produces a Boolean value. A Boolean expression is composed of a combination of the Boolean constants (True or False), Boolean variables and logical connectives. Each Boolean expression represents a Boolean function.

3. What is a logic gate?

Ans) Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on certain logic.

4. List the basic gates?

Ans)

i) OR Gate

ii) AND Gate

iii) NOT Gate

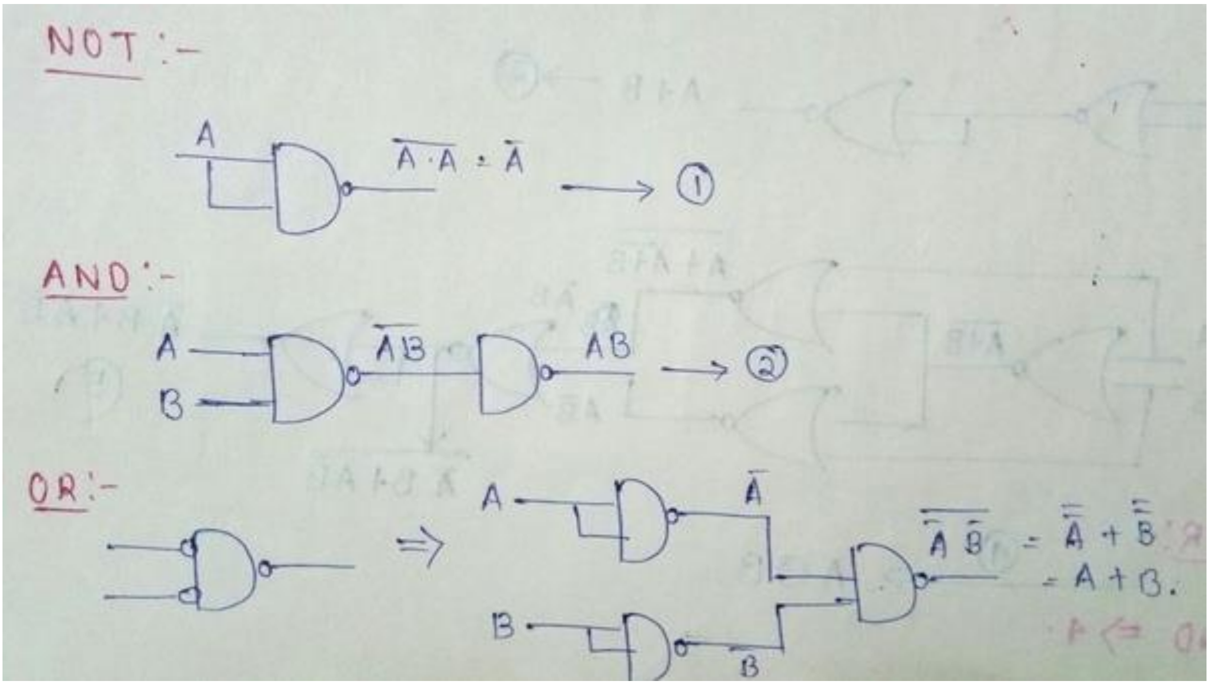
iv) XOR Gate

5. Why NAND and NOR are called universal gates?

Ans) NAND and NOR are called universal gates because all the other gates like AND, OR, NOT, XOR and XNOR can be derived from it.

6. Realize AND using NAND gate?

Ans) (Refer 2 in the figure given below)

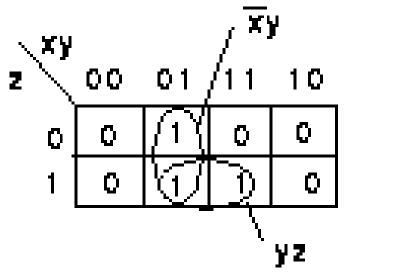


7. Simplify the expression f(x,y,z) = ∑(2,3,7).

Ans)

$$
\begin{aligned}
f &= \overline{x}y\overline{z} + \overline{x}yz + xyz \\
&= \overline{x}y\overline{z} + \overline{x}yz + \overline{x}yz + xyz \\
&= \overline{x}y + yz
\end{aligned}
$$

K-map:



8. Simplify the expression Y = C + (BC)’ using Boolean laws.

Ans)

Expression	Rule(s) Used
C + (BC)’	Original Expression
C + (B’ + C’)	DeMorgan's Law.
(C + C’) + B	Commutative, Associative Laws.
1 + B’	Complement Law.
1	Identity Law.

Long Answer Questions:

1. Simplify the following expressions using Boolean laws:

i) (A + C)(AD + AD’) + AC + C

ii A’(A + B) + (B + AA)(A + B’)

Ans) i)

Expression	Rule(s) Used
(A + C)(AD + AD’) + AC + C	Original Expression
(A + C)A(D + D’) + AC + C	Distributive.
(A + C)A + AC + C	Complement, Identity.
A((A + C) + C) + C	Commutative, Distributive.
A(A + C) + C	Associative, Idempotent.
AA + AC + C	Distributive.
A + (A + 1)C	Idempotent, Identity, Distributive.
A + C	Identity, twice.

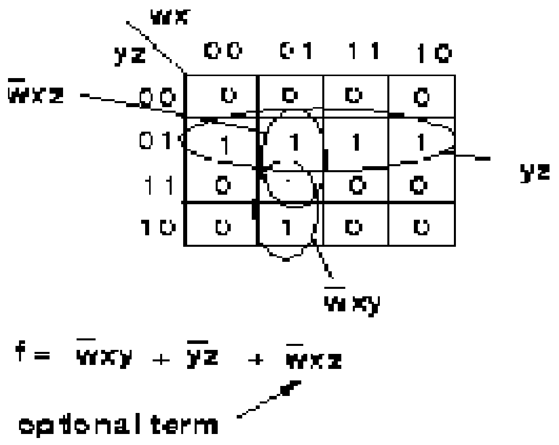
ii)

Expression	Rule(s) Used
A’(A + B) + (B + AA)(A + B’)	Original Expression
A’A + AB + (B + A)A + (B + A)B’	Idempotent (AA to A), then Distributive, used twice.
A’B + (B + A)A + (B + A)B’	Complement, then Identity. (Strictly speaking, we also used the Commutative Law for each of these applications.)
A’B + BA + AA + BB’ + AB’	Distributive, two places.
A’B + BA + A + AB’	Idempotent (for the A's), then Complement and Identity to remove BB.
A’B + AB + A(1) + AB’	Commutative, Identity; setting up for the next step.

A’B + A(B + 1 + B’)	Distributive.
A’B + A	Identity, twice (depending how you count it).
A + A’B	Commutative.
(A + A’)(A + B)	Distributive.
A + B	Complement, Identity.

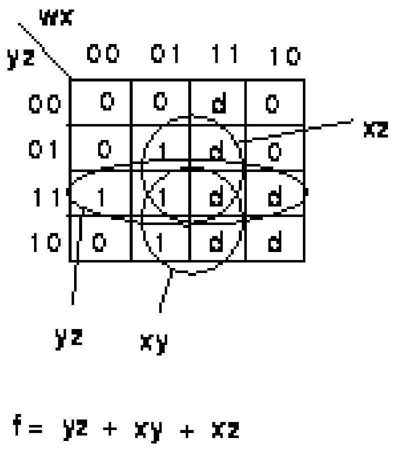
2. Simplify the expression f(w,x,y,z) = ∑(1,5,6,7,9,13)

Ans)



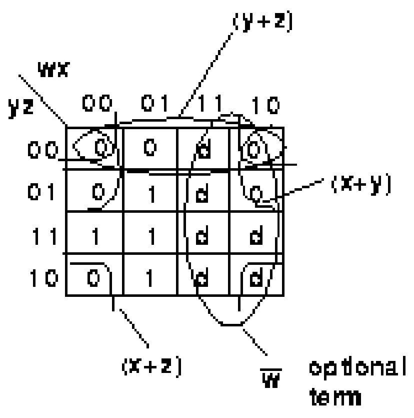
3. Simplify the expression f(w,x,y,z)=∑(3,5,6,7) + d (10,11,12,13,14,15).

Ans)



4. Simplify the expression f(w,x,y,z)=π(3,5,6,7) + d (10,11,12,13,14,15).

Ans)



f = (x+z)(x+y)(y+z)

5. Simplify the expression Z= f(A,B,C) = A'B'C'+ A'B'C + AB'C' + AB'C using Quine McCluskey method.

Ans)

-Change the function into binary notation with index value and decimal value.

f(A, B , C) = $\sum$ (000, 001, 100, 101) ———— Binary notation

0 1 1 2 ———— Index

0 1 4 5 ———— Decimal value

-Tabulate the index groups in a column and insert the decimal value alongside.

	First List		Second List		Third List	
		A B C		A B C		A B C
Index 0 →	0	0 0 0 ✓	0,1	0 0 – ✓	0,1,4,5	– 0 –
Index 1 →	1	0 0 1 ✓	0,4	– 0 0 ✓	0,4,1,5	– 0 –
Index 1 →	4	1 0 0 ✓	1,5	– 0 1 ✓		
Index 2 →	5	1 0 0 ✓	4,5	1 0 – ✓		

-From the first list, we combine terms that differ by 1 digit only from one index group to the next. These terms from the first list are then separated into groups in the second list. Note that the ticks are just there to show that one term has been combined with another term. From the second list we can see that the expression is now reduced to: Z =A'B' + B'C' + B'C + AB'

-From the second list note that the term having an index of 0 can be combined with the terms of index 1. Note that the dash indicates a missing variable and must line up in order to get a third list. The final simplified expression is: **Z = B'**

Set of Questions for FA/IA/ESE

Q. 1) State DeMorgan's laws.

Q. 2) Simplify the expression (AB)'(A' + B)(B' + B).

Q. 3) Realize AND gate using NOR gates only.

Q.4) Convert the canonical SOP form F= A’B’C+ A’BC’ + A’BC+ AB’C into POS form

Q. 5) Simplify the expression Pi(0,2,3,5,7) using K-map and realize using basic gates.

Q. 6) Simplify the expression ∑(0,4,5) using K-map and realize using NOR gates only

Q. 6) Simplify the expression f(w,x,y,z)=π(3,5,6,7) + d (10,11,12,13,14,15) using Quine McCluskey method.

Q. 7) Simplify the expression D=A’BC’+ABC’+AB’C’+A’B’C+A’BC+ABC using K-map and realize using NAND gates.

Q. 8) List the different laws used in Boolean algebra.

References:

1) Modern Digital Electronics By R. P. Jain.

2) Digital Logic and Computer Design By M. Morris Mano.

3) Digital Principles and Applications By Donald p Leach, Albert Paul Malvino

Practice for Module-02

Q.1) Simplify the following expressions i) F(A,B,C)=A′B+BC′+BC+AB′C′ ii) F(A,B,C)=(A+B)(A+C) using Boolean laws (10 marks)

Q.2) a) Convert the SOP F= AC’+AB+BC into canonical SOP (5 marks)

b) Convert the POS F = (A+B). (B+C). (A+C) into canonical POS (5 marks)

Q.3) Simplify the expression ∑(2,5,7) using K-map and realize using basic gates. (5 marks)

Q.4) Simplify the expression ∑(2,7,9,10,11,12,14,15) using K-map and realize using NOR gates. (10 marks)

Q.5) Simplify the expression Pi (2,3,4,6,9,11,12,13) using K-map and realize using basic gates. (10 marks)

Q.6) Simplify the expression ∑ (0,1,6,7,810,14,15) using Quine McCluskey method. (10 marks)

Self-assessment

Q.1) What is Boolean Algebra? Define the role of Boolean algebra in digital logic design.

Q. 2) Recall the Properties of Boolean Algebra.

Q. 3) Explain how Quine McCluskey method can overcome the shortfalls of K-map method.

Q. 4) Explain the steps used in K-map method to reduce Boolean expression.

Self-evaluation

Name of Student		
Class		
Roll No.		
Subject		
Module No.		
S.No		Tick Your choice
1.	Do you understand how Boolean algebra can be used in digital logic design?	○ Yes ○ No
2.	Do you understand the difference between canonical SOP/POS and Standard SOP/POS?	○ Yes ○ No
3.	Do you understand the K-map technique for reducing boolean expression?	○ Yes ○ No
4.	Do you understand the Quine McCluskey method for reducing boolean expression?	○ Yes ○ No
5.	Do you understand how the universal gates can be used to implement basic gates ?	○ Yes ○ No
6.	Do you understand module 2 ?	○ Yes, Completely. ○ Partialy. ○ No, Not at all.