ECAP 268 COMP. SYSTEM ARCHITECTURE

CH-2:- Boolean Algebra

Boolean Algebra:- is the branch of algebra in which the values of the variables are the truth values, i.e., true & false, usually denoted by 1 & 0 respectively.

It is like any other mathematical system, which may be defined with a set of elements, a set of operators & a no. of postulates

⇒ Set of elements & operators:-

- A set of elements is any collection of objects having a common property. If S is a set, x & y are certain ~~elem~~ objects, then $x \in S$ denotes that x is a member of set S & $y \notin S$ denotes that y is not an element of S.
- Consider the relation $a * b = c$, we say that $*$ is a binary operator if it specifies a rule for finding c from a pair (a, b) & also if $a, b, c \in S$. However, $*$ is not a binary operator if $a, b \in S$ but $c \notin S$.

⇒ Postulates:-

The postulates of a mathematical system form the basic assumptions from which it is possible to deduce the rules, theorems & properties of the system.

The most common postulates used to formulate various algebraic structures are:-

i) **Closure**:- A set is closed with respect to a binary operator if, for every pair of elements of S, the binary operator specifies a rule for obtaining a unique element of S.

e.g:- The set of natural no. $N = \{1, 2, 3, 4 \ldots\}$ is closed with respect to the binary operator + by the rule of arithmetic addition, since for any $a, b \in N$ we obtain a unique $c \in N$ by the operation of $a + b = c$. But the set of natural no. is not closed with respect to binary operator minus (−) by the rules of arithmetic substraction because $2 - 3 = -1$ & $2, 3 \in N$ while $(-1) \notin N$

ii) **Associative Law**:- Allows the removal of brackets from an expression & regrouping of the variables.

- $A + (B + C) = (A + B) + C = A + B + C$ (OR Associative law)
- $A (B \cdot C) = (A \cdot B) C = A \cdot B \cdot C$ (AND Associate Law)

iii) <u>Commutative Law</u>:- The order of application of 2 separate terms is not important

<u>e.g</u>:- $A \cdot B = B \cdot A$ The order in which 2 variables are AND'ed makes no difference

$A + B = B + A$ The order in which 2 variables are OR'ed makes no difference

iv) <u>Identity element</u>:- A term OR'ed with a "0" or "AND'ed with a "1" will always equal that term

<u>e.g</u>:- $A + 0 = A$ A variable <u>OR'ed</u> with 0 is alway equal to the variable

$A \cdot 1 = A$ A ,, <u>AND'ed</u> ,, 1 ,, ,, ,, ,, ,, ,,

v) <u>Distributive Law</u>:- This law permits the multiplying or factoring out an expression

<u>e.g</u>:- $A(B+C) = A \cdot B + A \cdot C$ (<u>OR</u> Distributive law)

$A + (B \cdot C) = (A + B) \cdot (A + C)$ (<u>AND</u> Distributive law)

vi) <u>Double inversion law</u>:- A term that is inverted twice is equal to the original term

<u>e.g</u>:- $(A')' = A$ A double complement of a variable is always equal to the variable.

vii) <u>Annulment law</u>:- A term AND'ed with "0" equals 0 or OR'ed with "1" will equal 1

<u>e.g</u>:- $A \cdot 0 = 0$ A variable <u>AND'ed</u> with 0 is always equal to 0

$A + 1 = 1$,, ,, <u>OR'ed</u> ,, 1 ,, ,, ,, ,, 1

viii) <u>Idempotent Law</u>:- An input that is AND'ed or OR'ed with itself is equal to that input

<u>e.g</u>:- $A + A = A$ A variable <u>OR'ed</u> with itself is always equal to the variable

$A \cdot A = A$ A variable <u>AND'ed</u> with itself is always equal to the variable.

ix) <u>Complement Law</u>:- A term AND'ed with its complement equals "0" & a term OR'ed with its complement equals "1"

<u>e.g</u>:- $A \cdot A = 0$ A variable <u>AND'ed</u> with its complement is always equal to 0

$A + A = 1$ A variable OR'ed with its complement is always equal to 1.

x) <u>Absorptive law</u>:- This law enables a reduction in a complicated expression to a simpler one by absorbing like terms

<u>e.g</u>:- $A + (A \cdot B) = A$ (<u>OR</u> Absorption Law)

$A(A + B) = A$ (<u>AND</u> Absorption law)

Duality Principle:- If a statement is true, then also it's dual statement is true. We obtain the dual statement by changing + for ., . for +, 1 for 0 & 0 for 1.

e.g:- $0.1=0$ is a true statement asserting that "false & true evaluates to false".

$1+0=1$ is a true statement asserting that "true or false evaluates true".

⇒ Basic Theorems and Properties:-

i) $A+0=A$
ii) $A+1=1$
iii) $A\cdot 0=0$
iv) $A\cdot 1=A$
v) $A+A=A$
vi) $A+A'=1$
vii) $A\cdot A=A$
viii) $A\cdot A'=0$
ix) $(A')'=A$
x) $A+AB=A$
xi) $A+A'B=A+B$
xii) $(A+B)(A+C)=A+BC$

De-Morgan's Theorem:-

These are basically 2 ~~sets~~ sets of rules or laws developed from the Boolean expressions for AND, OR & NOT using 2 input variables A & B.

These 2 rules or theorems allows the input variables to be negated & converted from one form of a Boolean funcⁿ into an opposite form.

1st Theorem:-

Complementing the result of AND'ing variables together is equivalent to OR'ing the complements of the individual variables.

$\boxed{(A\cdot B)' = A'+B'}$

When 2 (or more) input variables are AND'ed & negated, they are equivalent to the OR of the complements of the individual variables. Thus the equivalent of the NAND funcⁿ & is a -ve -OR funcⁿ proving that $\boxed{A\cdot B = A+B}$

Verification

Inputs			Outputs			
B	A	A·B	(A·B)'	A'	B'	A'+B'
0	0	0	1	1	1	1
0	1	0	1	0	1	1
1	0	0	1	1	0	1
1	1	1	0	0	0	0

2nd Theorem:-

It proves that when 2 (or more) input variables are OR'ed & negated they are equivalent to the AND of the complements of the individual variables.

Thus, the equivalent of the NOR func^n & is a -ve - AND func^n proving that $\boxed{(A+B)' = A' \cdot B'}$ & again we can show this using the following truth table.

Input		Output				
B	A	A+B	(A+B)'	A'	B'	A'.B'
0	0	0	1	1	1	1
0	1	1	0	0	1	0
1	0	1	0	1	0	0
0	1	1	0	0	0	0

Verified

Operator Precedence:-

The operator precedence for evaluating Boolean expression is:-

i) Parentheses
ii) NOT
iii) AND
iv) OR

Boolean Func^n:-

- A binary variable can take the value either 0 or 1
- A boolean function is an expression formed with binary variables, & binary operators AND, OR & one unary operator NOT, parentheses & an equal sign.
- For the given value of variables, the func^n can be either 0 or 1.

Truth Table:- If there are n variables, then there will be 2^n combinations of 1's & 0's

Example:- $F1 = xyz'$ $F2 = x + y'z$ $F3 = x'y'z + x'yz + xy'$ $F4 = xy' + x'z$

x	y	z	F1	F2	F3	F4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

Algebraic Manipulation :-

- A literal is the use of the variable or its complement form in the expression.
- The minimization of no. of literals & the no. of terms results in a circuit with less equipments.
- The no. of literals can be minimized by the algebraic manipulation

Example :- Q.) $x + x'y$
$= (x + x')(x + y)$
$= (1)(x + y)$
$= x + y$

Q.) $x(x' + y)$
$= xx' + xy$
$= 0 + xy$
$= xy$

Q.) $x'y'z + x'yz + xy'$
$= x'z(y' + y) + xy'$
$= x'z(1) + xy'$
$= x'z + xy'$

Q.) $xy + x'z + yz$
$= xy + x'z + yz(x + x')$
$= xy + x'z + xyz + x'yz$
$= xy + xyz + x'z + x'yz$
$= xy(1 + z) + x'z(1 + y)$
$= xy + x'z$

?

Canonical & Standard forms :-

All boolean expressions regardless of their form can be converted to either of these 2 forms:-
i) SOP :- Sum of Products (~~minterm~~)
ii) POS :- Products of Sum

i) SOP :- A product term is a term consisting of the boolean multiplication of literals. It is known as minterms. When 2 or more minterms are summed by boolean addition, the resulting expression is a SOP form.

e.g:- $AB + ABC$; $ABC + CDE + B'CD'$; $A'B + A'BC' + AC$

SOP can also ~~to~~ contain a single variable term like :-
$A' + A'B'C + BCD'$

In SOP form, single bar cannot ~~be~~ extend over more than one variable however more than one variable in a term can have an over-bar. e.g:- SOP expression can have $A'B'C$ but it cannot have $(ABC)'$.

ii) POS :- A sum is defined as a term consisting of boolean addition of the literals. It is also known as Max term. When 2 or more sum terms are multiplied, the resulting expression is a product of sum form.

e.g:- $(A' + B)(A + B' + C)$; $(A' + B' + C')(C + D + E)(B' + C + D)$;
$(A + B)(A + B' + C)(A' + C)$

It can also contain single variable :- $A(A'+B'+C')(C+D'+E)(B'+C+D)$

In POS form, single bar cannot extend over more than one variable however more than one variable in a term can have an over bar.

e.g:- a POS expression can have the term $A'+B'+C'$ but not $(A+B+C)'$.

K-Map :- Karnaugh Map

convenient as long as it doesn't exceed 5 or 6.

- It is a systematic method for simplifying the boolean expressions.
- It produce simplest POS or SOP which is known as minimum expression
- It is similar to TT (Truth Table) because it represents all the possible value of IP (input) variables & the resulting OP (output) for each value

⇒ Array of Cells :-

- K-Map is an array of cells in which each cell represents a binary value of IP variables.
- It can be used for expressions with 2, 3, 4 & 5 variables.

Number of Cells :-

For 2 variables, no. of cells $= 2^2 = 4$ & so on...

⇒ Cell adjacency :-

- The cell in a K-map are arranged so that there is only a single variable change b/w (between) adjacent cells.
- Adjacency :- It is defined by a single variable change. Cells that differ by only one variable are adjacent cells.
- e.g:- For 3 variable K-map
    - 010 is adjacent to 000, 011, 110
    - 010 is not adjacent to 001, 111
- So, physically each cell is adjacent to the cells that are immediately next to it on any of its 4 sides.
- A cell is not adjacent to the cells that diagonally touch any of its corner.

⇒ Wrap around adjacency :-

The cells in the top row are adjacent to the corresponding cells in the bottom row & the cells in the outer left column are adjacent to the corresponding cells in the outer right column.

⇒ 2-variable K-map :- There are 4 min-terms

A \ B	0	1
0	00	01
1	10	11

⇒ 3-variable K-Map :- 8 min terms

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

⇒ 4-Variable K-Map :- 16 min terms

AB \ CD	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

?

K-Map SOP Minimization :-

A minimized SOP contains the fewest possible terms with the fewest possible variables per term.

After SOP expression is mapped, there are 3 steps in obtaining a minimum SOP expression.

i) Grouping of ~~the~~ 1's

ii) Determine the product term of each group.

iii) Summing the resulting product term

Example :- $A'B'C + A'BC' + ABC' + ABC$ (Simplify this funcn)

A \ BC	00	01	11	10
0		1		1
1			1	1

3

2

Group:-1

001
010
110
111

Group 1 → AB because these values are not changing in this group

Group 2 → BC' Complement of C is taken because the value of C is 0

Group 3 → A'B'C

Ans :- AB + BC' + A'B'C

Q.) $A'B'CD + A'BC'D' + ABC'D + ABCD + ABC'D' + A'B'C'D + AB'CD'$

0011   0100   1101   1111   1100   0001   1010

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

Now,
Group 1:- 1100, 1101
Group 2:- 1101, 1111
Group 3:- 1010
Group 4:- 0100, 1100
Group 5:- 0001, 0011
(Min)

Group 1:- ABC'
Group 2:- ABD
Group 3:- $AB'CD'$
Group 4:- $BC'D'$
Group 5:- $A'B'D$

$\rightarrow$ Ans:- $ABC' + ABD + AB'CD' + BC'D' + A'B'D$

⇒ For non-Standard expression:-

Q.) $A' + AB' + ABC'$ ~~max~~

0	10	110
000	100	
001	101	
010		
011		

A \ BC	00	01	11	10
0	1	1	1	1
1	1	1		1

Group 1:- 000, 001, 011, 010
Group 2:- 000, 001, 100, 101
Group 3:- 000, 100, 010, 110
(max)

Group 1:- A'
Group 2:- B'
Group 3:- C'

$\Rightarrow A' + B' + C'$

Q.) $B'C' + AB' + ABC' + AB'CD' + A'B'C'D + AB'CD$

$B'C'$	AB'	ABC'	$AB'CD'$	$A'B'C'D$	$AB'CD$
0000	1000	1100	1010	0001	1011
0001	1001	1101			
1001	1011				
1000	1010				

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

Group 1:- 0000, 0001
2:- 1100, 1101, 1000, 1001
3:- 1000, 1001, 1011, 1010
4:- 0000, 0001, 1000, 1001

Group 1:- $A'B'C'$
2:- AC'
3:- AB'
4:- $B'C'$

$\Rightarrow A'B'C' + AC' + AB' + B'C'$

K-Map POS Minimization:-

Q.) ~~(A+B+C)(A+B+C')(A+B'+C)~~ (A

After POS expression is mapped, there are 3 steps in obtaining a minimum POS expression.

i) Grouping of 0's
ii) Determine the sum term of each group
iii) Multiplying the resulting sum ~~term~~ terms.

Q.) $(A+B+C)(A+B+C')(A+B'+C)(A+B'+C')(A+B'+C)$
(0 0 0) (0 0 1) (0 1 0) (0 1 1) (1 1 0)

A \ BC	00	01	11	10
0	0	0	0	0
1				0

→ 1
→ 2

Group 1:- A
2:- B'+C

⇒ (A)(B'+C) ~~Ans~~

Non-Std

Q.) $(B+C+D)(A+B+C'+D)(A'+B'+C+D')(A+B'+C+D)(A'+B'+C+D)$
(0 0 0 0) (0 0 1 0) (1 1 0 1) (0 1 0 0) (1 1 0 0)
(1 0 0 0)

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

Group 1:- ~~CD~~ C+D
2:- A'+B'+C
3:- A+B+D

Ans:- (C+D)(A'+B'+C)(A+B+D)

Q.) $(A'+B'+C+D)(A+B'+C+D)(A+B+C+D')(A+B+C'+D')(A'+B+C+D')$
(1 1 0 0) (0 1 0 0) (0 0 0 1) (0 0 1 1) (1 0 0 1)
(A+B+C'+D)
(0 0 1 0)

~~For SOP~~:-

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

For SOP :-

Group 1 :- 0101, 0111, 1101, 1111 :- BD

2 :- 0111, 0110, 1111, 1110 :- BC

3 :- 1111, 1110, 1011, 1010 :- AC

4 :- 0000, 1000 :- ~~C'D'~~ B'C'D'

Ans :- BD + BC + AC + B'C'D'

For POS :-

Group 1 :- 0001, 0011 :- A+B+D'

2 :- 0011, 0010 :- A+B+C'

3 :- 0100, 1100 :- B'+C+D

4 :- 0001, 1001 :- B+C+D'

Ans :- (A+B+D')(A+B+C')(B'C+D)(B+C+D')

5-Variable Kmap :-

Q.) A'B'C'D'E' + A'B'CD'E' + A'BCD'E' + A'BC'D'E' + A'B'C'D'E +
00000 00100 01100 01000 00001

A'BCD'E + A'BCDE + AB'C'D'E' + AB'C'D'E + ABCD'E + AB'CDE +
01101 01111 10000 10001 11101 10111

+ ABCDE
11111

BC \ DE	00	01	11	10
00	1	1		
01	1			
11	1	1	1	
10	1			

A=0

BC \ DE	00	01	11	10
00	1	1		
01			1	
11		1	1	
10				

A=1

Group 1 :- A'D'E'

2 :- BCE

3 :- B'C'D'

4 :- ACDE

⟹ (A'D'E) + (BCE) + (B'C'D') + (ACDE)

Q:) Simplify the Boolean func^n :-

$F(A,B,C,D,E) = \{0, 2, 4, 6, 9, 13, 21, 23, 25, 29, 31\}$.

A = 0

BC \ DE	00	01	11	10
00	1			1
01	1			1
11		1		
10		1		

A = 1

BC \ DE	00	01	11	10
00				
01		1	1	
11		1	1	
10		1		

Group 1 :- A'B'E'
2 :- BD'E
3 :- ACE
4 :- ACD'E

$\Rightarrow$ A'B'E + BD'E + ACE + ACD'E

Tabulation Method :- Quine-McCluskey method

- The K-Map method of simplification is convenient as long as the variables doesn't exceed 5 or 6.
- As the no. of variables increases, the excessive no. of squares prevents a reasonable selection of adjacent squares.

Disadvantage of K-Map :-

This method relies on the ability of human user to recognize certain patterns
For func^n of 5 or more variables, it is difficult to be sure that the best sol^n has been made

Solution :- Tabulation Method

- It is a step-by-step procedure that is guaranteed to produce a simplified standard form expression for a func^n.
- It can be applied to problems with many variables.
- It is also known as Quine-McCluskey Method

This simplification method consists of 2 parts :-

A) The 1st is to find by an exhaustive search all the terms that are candidates for inclusion in the simplified func^n. These terms are known as prime implications.

B) The 2nd operation is to choose among the ~~prime implication~~ implicants those that give an expression with the least number of literals.

Part 1:- Determination of Prime Implicants:-

i) Write down the list of minterms that specify the func^n.

ii) Compare each minterm with every other minterm. If two minterms differ in only one variable, that variable is removed & a term with one less literal is found.

iii) This process is repeated for every minterm until the exhaustive search is completed.

iv) The matching process cycle is repeated for those new terms just found.

v) Further cycles are continued until a single pass through a cycle yields no further elimination of literals. The remaining terms & all the terms that is not match during the process comprise the prime implicants.

Example:- Simplify the following Boolean func^n by using the tabulation method.

$$F = \{0, 1, 2, 8, 10, 11, 14, 15\}$$

a)

		W	x	y	z	
i)	0	0	0	0	0	✓
	~~4~~					
ii)	1	0	0	0	1	✓
	2	0	0	1	0	✓
	8	1	0	0	0	✓
iii)	10	1	0	1	0	✓
iv)	11	1	0	1	1	✓
	14	1	1	1	0	✓
v)	15	1	1	1	1	✓

b)

	W	x	y	z	
0,1	0	0	0	—	
0,2	0	0	—	0	✓
0,8	—	0	0	0	✓
2,10	—	0	1	0	✓
8,10	1	0	—	0	✓
10,11	1	0	1	—	✓
10,14	1	—	1	0	✓
11,15	1	—	1	1	✓
14,15	1	1	1	—	✓

c)

	W	x	y	z
0,2,8,10	—	0	—	0
0,8,2,10	—	0	—	0
10,11,14,15	1	—	1	—
10,14,11,15	1	—	1	—

- The term of column b has only 3 variables. A 1 under the variable means it is unprimed & a 0 means it is primed, & a – means the variable is not included in the term.
- The searching & comparing process is repeated for the terms in column (b) to form the 2 variable terms of column (c)
- The unchecked terms in the table forms the prime implicants so, $F = w'x'y' + x'z' + wy$ ~~Ans~~

Another method: – the decimal equivalent of minterms are listed.

(a)		(b)			(c)	
0	✓	0,1	(1)		0,2,8,10	(2,8)
		0,2	(2)	✓	0,8,2,10	(2,8)
1	✓	0,8	(8)	✓		
2	✓				10,11,14,15	(1,4)
8	✓	2,10	(8)	✓	10,14,11,15	(1,4)
		8,10	(2)	✓		
10	✓					
		10,11	(1)	✓		
11	✓	10,14	(4)	✓		
14	✓					
		11,15	(4)	✓		
15	✓	14,15	(1)	✓		

Part 2:- Selection of Prime Implicants :-

- In the prime implicants table, each prime implicant is represented in a row & each minterm in a column
- A minimum set of prime implicants is then chosen that covers all the minterms in the func^n.