BITT POLYTECHNIC

Getlatu, Ranchi-835217

Digital Circuit & Microprocessor

BOOLEAN ALGEBRA

Introduction

Boolean algebra is used to analyse digital gates and circuits. It is a logic to perform mathematical operation on binary numbers i.e., on ‘0’ and ‘1’. Boolean algebra contains basic operators like AND, OR and NOT etc. Operations are represented by ‘.’ for AND, ‘+’ for OR. Operations can be performed on variables which are represented using capital letter e.g. ‘A’ , ‘B’ etc.

Axioms or Postulates of Boolean algebra

Axioms or Postulates of Boolean algebra are a set of logical expressions that we accept without proof and upon which we can build a set of useful theorems.

	AND Operation	**OR Operation**	**NOT Operation**
Axiom1 :	0.0=0	0+0=0	0’=1
Axiom2:	0.1=0	0+1=1	1’=0
Axiom3:	1.0=0	1+0=1	
Axiom4:	1.1=1	1+1=1	

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'

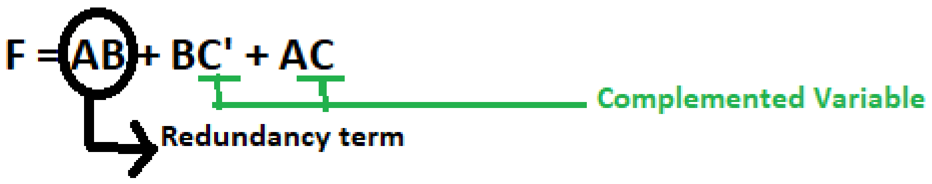
Consensus Theorem

Redundancy theorem is used as a Boolean algebra trick in Digital Electronics. It is also known as Consensus Theorem. this theorem is used to simply the Boolean Algebra. Conditions for applying Redundancy theorem are:

1. Three variables must present in the expression.Here A, B and C are used as variables.
2. Each variables is repeated twice.
3. One variable must present in complemented form. Theorem 1.

F = AB + A'C + BC = AB + A'C

Here, we have three variables A, B and C and all are repeated twice. The variable C is present in complemented form. So, all the conditions are satisfied for applying this theorem.



After applying Redundancy theorem we can write only the terms containing complemented variables (i.e, C) and omit the Redundancy term i.e., AB.

.'. F = AB+BC'+AC=BC' + AC

Consensus Theorem1 Proof:

AB+BC'+AC = AB.1+BC'+AC

= AB(C+C')+BC'+AC

= ABC+ABC'+BC'+AC

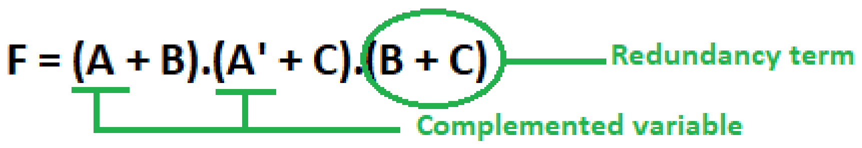
= AC(B+1)+BC'(A+1)

= BC'+AC

The conjunctive dual of this equation is:

Theorem 2:(A+B).(B+C').(A+C) = (B+C').(A+C)

Three variables are present and all are repeated twice. The variable A is present in complemented form. Thus, all the three conditions of this theorem is satisfied.



After applying Redundancy theorem we can write only the terms containing complemented variables (i.e, A) and omit the Redundancy term i.e., (B + C).

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

Principle of Duality

Each postulate consists of two expressions statement one expression is transformed into the other by interchanging the operations(+) and (·) as well as the identity elements 0 and 1.Such expressions are known as duals of each other.If some equivalence is proved, then its dual is also immediately true.If we prove: (x.x)+(x'+x')=1, then we have by duality:(x+x)·(x'.x')=0The Huntington postulates were listed in pairs and designated by part (a) and part (b) in below table.

Table for Postulates and Theorems of Boolean algebra

Part-A	Part-B
A+0=A	A.0=0
A+1=1	A.1=A
A+A=A (Impotence law)	A.A=A (Impotence law
A+A'=1	A.A'=0
Commutative law: A+B=B+A	A.B=B.A
Associative law: A + (B +C) = (A +B)+C	A(B.C) = (A.B)C
Distributive law: A.(B + C) = AB+AC	A + BC = (A + B).(A +C)
Absorption law: A +AB=A	A(A +B) = A
DeMorgan Theorem: (A+B)' = A'. B'	(A.B)' = A'+ B'

Redundant Literal Rule: A+(A'.B)'=A+B	A.(A'+B)'=AB
Consensus Theorem: AB+ A'C + BC = AB + A'C	(A+B). (A'+C).(B+C) =(A+B). (A'+C)

Canonical & Standard Forms

We will get four Boolean product terms by combining two variables x and y with logical AND operation. These Boolean product terms are called as **min terms** or **standard product terms**. The min terms are x'y', x'y, xy' and xy.

Similarly, we will get four Boolean sum terms by combining two variables x and y with logical OR operation. These Boolean sum terms are called as **Max terms** or **standard sum terms**. The Max terms are:
x + y, x + y', x' + y and x' + y'.

The following table shows the representation of min terms and MAX terms for 2 variables.

x	**y**	**Min terms**	**Max terms**
0	0	m_o=x'.y'	M_0=x+y
0	1	m_1=x'.y	M_1=x+y'
1	0	m_2=x.y'	M_2=x'+y
1	1	m_3=x.y	M_3=x'+y'

If the binary variable is '0', then it is represented as complement of variable in min term and as the variable itself in Max term. Similarly, if the binary variable is '1', then it is represented as complement of variable in Max term and as the variable itself in min term.

From the above table, we can easily notice that min terms and Max terms are complement of each other. If there are 'n' Boolean variables, then there will be 2^n min terms and 2^n Max terms.

Canonical forms

A truth table consists of a set of inputs and outputs

If there are 'n' input variables, then there will be 2^n possible combinations with 0's and 1's. So the value of each output variable depends on the combination of input variables. So, each output variable will have '1' for some combination of input variables and '0' for some other combination of input variables.

Therefore, we can express each output variable in following two ways.

- Canonical SoP form
- Canonical PoS form

Canonical SoP form

Canonical SoP form means **Canonical Sum of Products** form. In this form, each product term contains all literals. So, these product terms are nothing but the min terms. Hence, canonical SoP form is also called as **sum of min terms** form.

First, identify the min terms for which, the output variable is 1 and then do the logical OR of those min terms in order to get the Boolean expression *function* corresponding to that output variable. This Boolean function will be in the form of sum of min terms.

Follow the same procedure for other output variables also, if there is more than one output variable.

Example

Consider the following **truth table**.

Input			**Output**
p	**q**	**r**	**f**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Here, the output *f* is '1' for four combinations of inputs. The corresponding min terms are p'qr, pq'r, pqr', pqr. By doing logical OR of these four min terms, we will get the Boolean function of output *f*

Therefore, the Boolean function of output is, f = p'qr + pq'r + pqr' + pqr. This is the **canonical SoP form** of output, f. We can also represent this function in following two notations.

$f=m3+m5+m6+m7$

$f=\sum m(3,5,6,7)$

In one equation, we represented the function as sum of respective min terms. In other equation, we used the symbol for summation of those min terms.

<u>**Canonical PoS form**</u>

Canonical PoS form means **Canonical Product of Sums** form. In this form, each sum term contains all literals. So, these sum terms are nothing but the Max terms. Hence, canonical PoS form is also called as **product of Max terms** form.

First, identify the Max terms for which, the output variable is 0 and then do the logical AND of those Max terms in order to get the Boolean expression *function* corresponding to that output variable. This Boolean function will be in the form of product of Max terms.
Follow the same procedure for other output variables also, if there ismore than one output variable.

Example

Consider the same truth table of previous example. Here, the output *f*

is '0' for four combinations of inputs. The corresponding Max terms are: p + q + r, p + q + r', p + q' + r, p' + q + r. By doing logical AND of these four Max terms, we will get the Boolean function of output *f*

Therefore, the Boolean function of output is, f = $(p+q+r).(p+q+r').(p+q'+r).(p'+q+r)$ This is the **canonical PoS form** of output, f. We can also represent this function in following two notations.

$f=M0.M1.M2.M4$

$f=\prod M(0,1,2,4)$

In one equation, we represented the function as product of respective Max terms. In other equation, we used the symbol for multiplication of those Max terms.

The Boolean function, f = $(p+q+r).(p+q+r').(p+q'+r).(p'+q+r)$ is the dual of the Boolean function, f = p'qr + pq'r + pqr' + pqr.

Therefore, both canonical SoP and canonical PoS forms are **Dual** to each other. Functionally, these two forms are same. Based on the requirement, we can use one of these two forms.

Standard forms

We discussed two canonical forms of representing the Boolean outputs. Similarly, there are two standard forms of representing the Boolean outputs. These are the simplified version of canonical forms.

- Standard SoP form
- Standard PoS form

The main **advantage** of standard forms is that the number of inputs applied to logic gates can be minimized. Sometimes, there will be reduction in the total number of logic gates required.

<u>Standard SoP form</u>

Standard SoP form means **Standard Sum of Products** form. In this form, each product term need not contain all literals. So, the product terms may or may not be the min terms. Therefore, the Standard SoP form is the simplified form of canonical SoP form.

We will get Standard SoP form of output variable in two steps.

- Get the canonical SoP form of output variable
- Simplify the above Boolean function, which is in canonical SoP form.

Follow the same procedure for other output variables also, if there is more than one output variable. Sometimes, it may not possible to simplify the canonical SoP form. In that case, both canonical and standard SoP forms are same.

Example

Convert the following Boolean function into Standard SoP form. f = p'qr + pq'r + pqr' +

pqr

The given Boolean function is in canonical SoP form. Now, we have to simplify this Boolean function in order to get standard SoP form.

Step 1 − Use the **Boolean postulate**, x + x = x. That means, the Logical OR operation with any Boolean variable 'n' times will be equal to the same variable. So, we can write the last term pqr two more times.

⇒ f = p'qr + pq'r + pqr' + pqr + pqr + pqr

Step 2 − Use **Distributive law** for 1st and 4th terms, 2nd and 5th terms, 3rd and 6th terms.

⇒ f = qr(*p′*+*p)*+ pr(*q′*+*q)* + pq(*r′*+*r)*

Step 3 − Use **Boolean postulate**, x + x' = 1 for simplifying the terms present in each parenthesis.

⇒ f = qr(1)+ pr(1) + pq(1)

Step 4 − Use **Boolean postulate**, x.1 = x for simplifying above three terms.

⇒ f = qr + pr + pq

⇒ f = pq + qr + pr

This is the simplified Boolean function. Therefore, the **standard SoP form** corresponding to given canonical SoP form is **f = pq + qr + pr**

<u>Standard PoS form</u>

Standard PoS form means **Standard Product of Sums** form. In this form, each sum term need not contain all literals. So, the sum terms may or may not be the Max terms. Therefore, the Standard PoS form is the simplified form of canonical PoS form.

We will get Standard PoS form of output variable in two steps.

- Get the canonical PoS form of output variable

- Simplify the above Boolean function, which is in canonical PoS form.

  Follow the same procedure for other output variables also, if there is more than one output variable. Sometimes, it may not possible to simplify the canonical PoS form. In that case, both canonical and standard PoS forms are same.

Example

Convert the following Boolean function into Standard PoS form. f =

$(p+q+r).(p+q+r').(p+q'+r).(p'+q+r)$

The given Boolean function is in canonical PoS form. Now, we have to simplify this Boolean function in order to get standard PoS form.

Step 1 − Use the **Boolean postulate**, x.x = x. That means, the Logical AND operation with any Boolean variable 'n' times will be equal to the same variable. So, we can write the first term p+q+r two more times.

⇒ f = $(p+q+r).(p+q+r).(p+q+r).(p+q+r').(p+q'+r).(p'+q+r)$

Step 2 − Use **Distributive law,** x + $(y.z)=(x+y).(x+z)$

for 1st and 4th parenthesis, 2nd and 5th parenthesis, 3rd and 6th parenthesis.

⇒ f = $(p+q+rr').(p+r+qq').(q+r+pp')$

Step 3 − Use **Boolean postulate**, x.x'=0 for simplifying the terms present in each parenthesis.

⇒ f = $(p+q+0).(p+r+0).(q+r+0)$

Step 4 − Use **Boolean postulate**, x + 0 = x for simplifying the terms present in each parenthesis

⇒ f = $(p+q).(p+r).(q+r)$
⇒ f = $(p+q).(q+r).(p+r)$

This is the simplified Boolean function. Therefore, the **standard PoS form** corresponding to given canonical PoS form is **f = $(p+q).(q+r).(p+r)$**. This is the **dual** of the Boolean function, f = pq + qr + pr.

Therefore, both Standard SoP and Standard PoS forms are Dual to each other.