**BOOLEAN ALGEBRA AND LOGIC GATES**

**Properties of Boolean Algebra**

* Switching algebra = Boolean algebra

**Switching Laws**

* **Annulment law:**

**A.0 = 0**

**A+1 = 1**

* **Identity law:**

**A.1 = A**

**A+0 = A**

* **Idempotent law:**

**A+A = A**

**A.A = A**

* **Complement law:**

**A+A’ = 1**

**A.A’ = 0**

* **Double negation law:**

**(A’)’ = A**

* **Commutative law:**
  + Order doesn’t matter.

**A+B = B+A**

**A.B = B.A**

* **Associative law:**
  + Order doesn’t matter when priority of symbols is same.

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

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

* **Distributive law:**

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

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

* **Absorption law:**

**A.(A+B) = A**

**A + AB = A**

**A+ A'B = A+B**

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

* **De Morgan’s law:**
  + Inverting equation on one side results in variable and operator inversion.

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

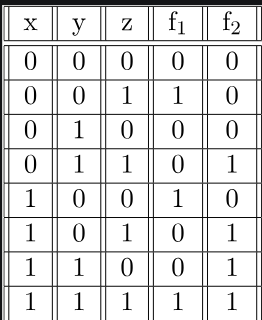
**(A+B)' = A'.B'**

* **Consensus theorem:**

**AB + A’C + BC = AB + A’C**

**Representation of Boolean Functions**

* **Boolean function:** The expression of a switch circuit.
* **Standard form:** True form
* **Canonical form:** Complemented form
* Every binary variable can take **2 forms** (standard or canonical).
* So, **n** variables involved means **2n** possible **combinations**.
* **Minterm/standard product (mi):** All possible combinations of binary variables connected by AND logic.
* **For example:** x.y, x’.y, x.y’, x’.y’
* Complemented variables in **minterms** are 0, **true forms** are 1.
* A variable is **un-primed** in **minterm** if it is 1.
* A variable is **primed** in **minterm** if it is 0.
* **Maxterm/ standard sum (Mi):** All possible combinations of binary variables connected by OR logic.
* **For example:** x+y, x’+y, x+y’, x’+y’
* Complemented variables in **maxterms** are 1, **true** **forms** are 0.
* A variable is **un-primed** in **maxterm** if it is 0.
* A variable is **primed** in **maxterm** if it is 1.
* Each minterm is **complement** of its corresponding maxterm.



**f1(x,y,z) = m1 + m4 + m7 = M1.M4.M7**

**f2(x,y,z) = m3 + m5 + m6 + m7 = M3.M5.M6.M7**

* Boolean functions are represented as **sum of minterms** or **product of maxterms** is called **canonical form**.

**POS and SOP**

Expression conversions to POS and SOP example:-

**Given:** F = x + y’z

**F = x(y+y’) + y’z**

**F = xy + xy’ + y’z**

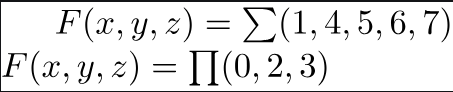
**F = xy(z+z’) + xy’(z+z’) + (x+x’)y’z**

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

**F = m1 + m4 + m5 + m6 + m7**

**F = M0.M2.M3**

* We write the given equation in form of **minterms** for getting **SOP**.
* And for **POS** we write it in terms of **maxterms**.



* **Sigma** represents **minterms**.
* And **Pi** represents **maxterms**.

**Canonical and Standard Form**

* Canonical **disjunctive** normal – minterms
* Canonical **conjunctive** normal – maxterms
* Boolean functions represented as SOP or POS are known as **canonical form**.
* In standard form, Boolean function is expressed as **either in true** form or **complemented** form.

**Forming Truth Table**

* Minterms in OR combination, where value of function is 1.
* Maxterms in AND combination, where value of function is 0.
* When converting a function to **maxterm**, **distributing law** is used.

**Example:-**

If

**F = m1 + m3 + m6 + m7 = Sigma(1,3,6,7)**

Then,

**F’ = m2 + m4 + m5 = Sigma(2,4,5)**

**Canonical Form**

1. **Advantages:**

* Uniqueness
* Clarity
* Completeness (can represent function despite being complex)

1. **Disadvantages:**
   * Complex (for functions with many variables)
   * Computation
   * Redundancy (unrequired variables left out)

**Standard Form**

1. **Advantages:**
   * Simplicity
   * Efficiency (fewer logic gates)
   * Flexibility (can be derived to other equations)
2. **Disadvantages:**
   * Non-uniqueness
   * Incompleteness
   * Ambiguous (multiple terms of same value)

**Functional Completeness**

* **Set of operations:** A set containing all required operators for a given function.
* **Functionally complete set:** {+,\*} or {+,’} or {\*,’} involved.

**Post’s Functional Completion Theorem**

* **T0 (only 0 preserving functions)**

**F(0,0,0,…,0) = 0**

* **T1**
* **S (self dual functions)**

**F(x1,…,xn) = ~F(~x1,…,~xn)**

* **M (monotonic functions)**

**{x1,…,xn} <= {x1,…,xn} because F(x1,…,xn) <= F(x1,…,xn)**

* **L (linear functions)**

**F(x1,…,xn) = a0 + a1x1 + … + anxn**

**Theorem:** It says that for a given function F, there is a member which doesn’t follow each class’ rules above.

* **Procedure:-**
  + Put all variables as A, then all as B, then C etc.
  + Then make some **bullshit** **derivations** using those equations.
  + And **fetch** all the involved operators.
  + And conclude if the function is functionally complete or not.

**Functional Completeness**

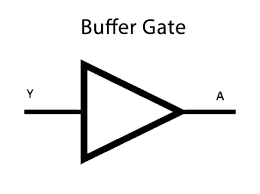
1. **Advantages:**
   * Flexibility (can represent any Boolean function)
   * Efficient (construction using less basic gates)
   * Universality (can be used anywhere)
2. **Disadvantages:**
   * Complexity (difficult to understand)
   * Limited applicability (at some situations, may not be suitable to use)
   * Non-intuitive (abstract maths, not intuitive)

**Propositional Logic**

* **+** is disjunction
* **.** is conjunction
* **‘** is negation

**Logic Gates**

* **Gate:**
  + Digital circuit
  + Allows electric current
* **XOR:** For **n** inputs, if number of **1s** are **odd**, then output is **1**, else **0**.
* **Buffer gate:** Opposite of **NOT** gate (same as input).



**Universal Gates**

* NAND and NOR are called universal gates because **other gates can be made** using them.

**Two serial NAND gates = a AND gate**

* XOR using NAND:-

