**Assignment 1**

Verification of Logic Gates & Boolean Algebra.

Name: Partha Kadam

Branch: AI&DS

DIV: A

Batch: 3

Roll No.: 85

PRN: 12412326

Theory:

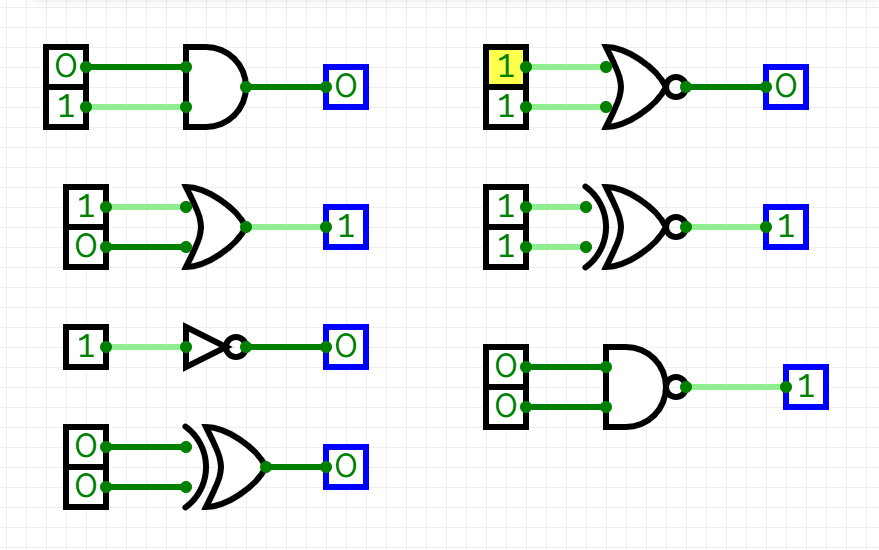
Logic Gates

* We have 7 Logic Gates
* Digital logic is the foundation, not only of computing but also many other electronic devices and control systems found in almost every part of modern life.
* Combinations of logic gates form circuits that can perform specific tasks within larger circuits or systems.
* The process of producing complex circuits using combinations of basic devices is called Combinational Logic.
* A circuit designer may want to design a combinational logic circuit that uses the minimum number of gates, or performs the required task in the least time, or at the minimum cost.
* A logic gate is a small transistor circuit, basically a type of amplifier, which is implemented in different forms within an integrated circuit. Each type of gate has one or more (most often two) inputs and one output.

1. AND Gate:

* Output is at Logic 1 when, & only when all its inputs are at Logic 1, otherwise the output is at Logic 0.
* Boolean Representation –: X=A.B

Circuit Diagram



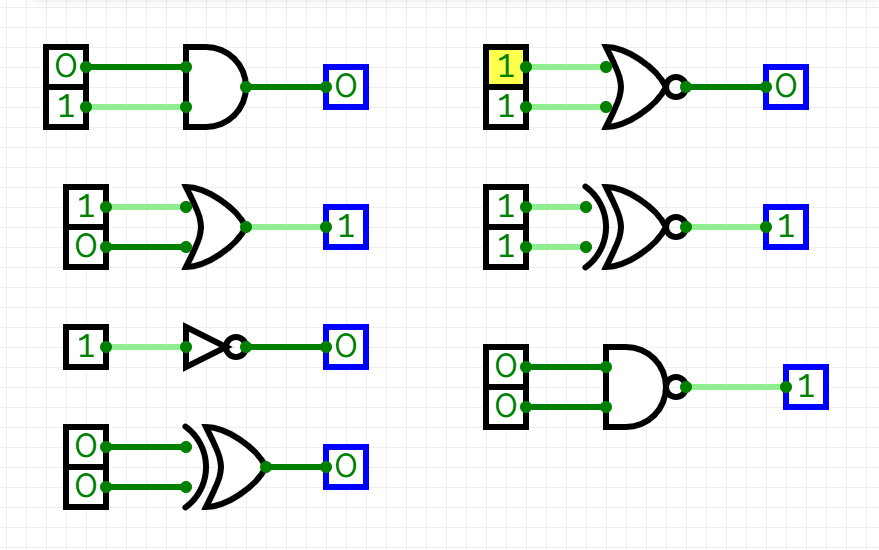
Truth Table

|  |  |  |
| --- | --- | --- |
| Input A | Input B | Output |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

1. OR Gate:

* Output is at Logic 1 when one or more are at Logic 1. If all inputs are at Logic 0, output is at Logic 0.
* Boolean Representation-: X=A+B

Circuit Diagram



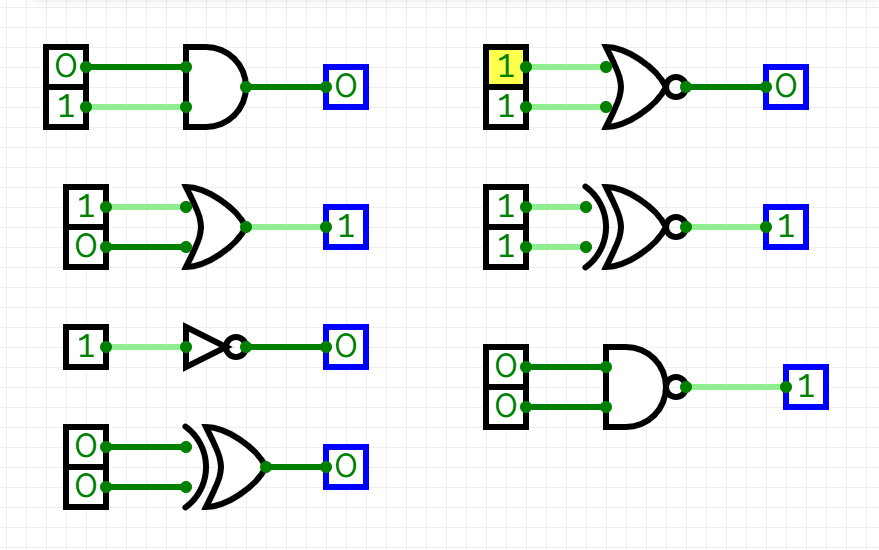
Truth Table

|  |  |  |
| --- | --- | --- |
| Input A | Input B | Output |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

1. NAND Gate:

* Output is at Logic 0 when, and only when all its inputs are at Logic 1, otherwise the output is at Logic 1.
* Boolean Representation-: X=

Circuit Diagram



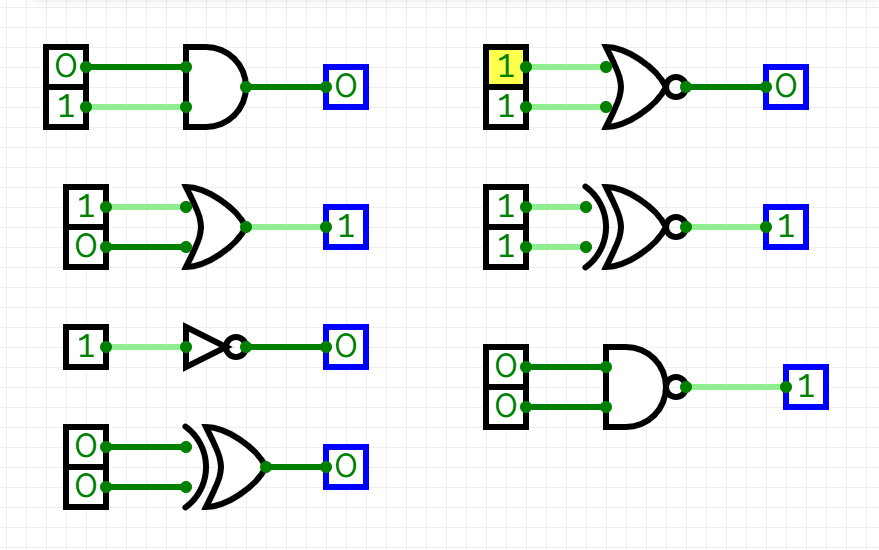
Truth Table

|  |  |  |
| --- | --- | --- |
| Input A | Input B | Output |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

1. NOR Gate:

* Output is at Logic 0 when one or more inputs are at Logic 1. If all the inputs are at Logic 0, the output is at Logic 1.
* Boolean Representation-:X=

Circuit Diagram



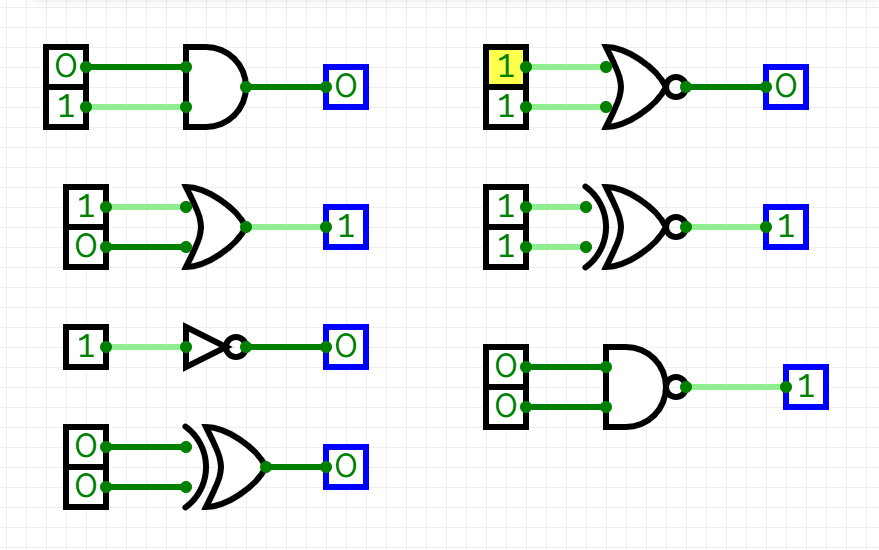
Truth Table

|  |  |  |
| --- | --- | --- |
| Input A | Input B | Output |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

1. XOR Gate:

* Output is at Logic 1 when one and only one of its inputs is at Logic 1, otherwise it is at Logic 0.
* Note. In the case of 3 input XOR gate, if all inputs are at Logic 1, the output is at Logic 1.
* Boolean Representation-:X=

Circuit Diagram



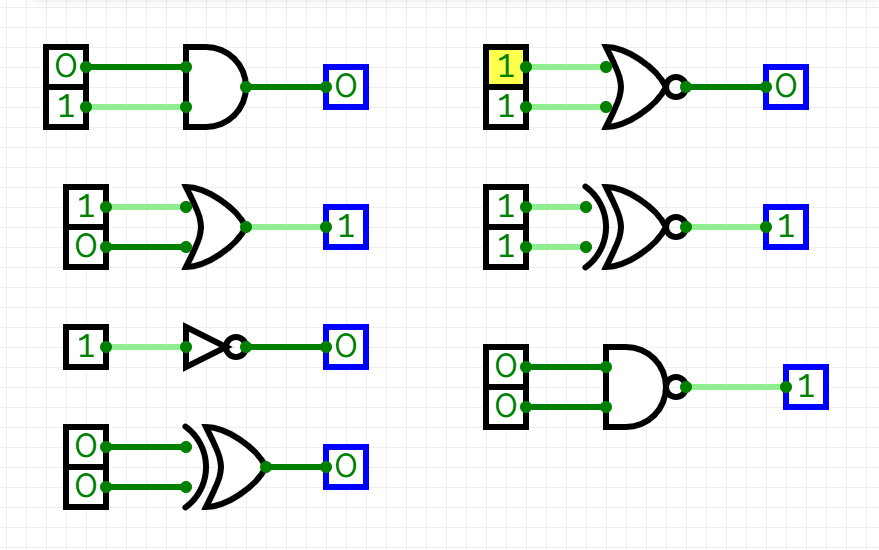
Truth Table

|  |  |  |
| --- | --- | --- |
| Input A | Input B | Output |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

1. XNOR Gate:

* Output is at Logic 0 when one and only one of its inputs is at Logic 1, otherwise it is at Logic 1. Similar to XOR but inverted.
* Note. If 3 input XNOR gate, all inputs are at Logic 1 the output is at Logic 0.
* Boolean Representation-: X=

Circuit Diagram



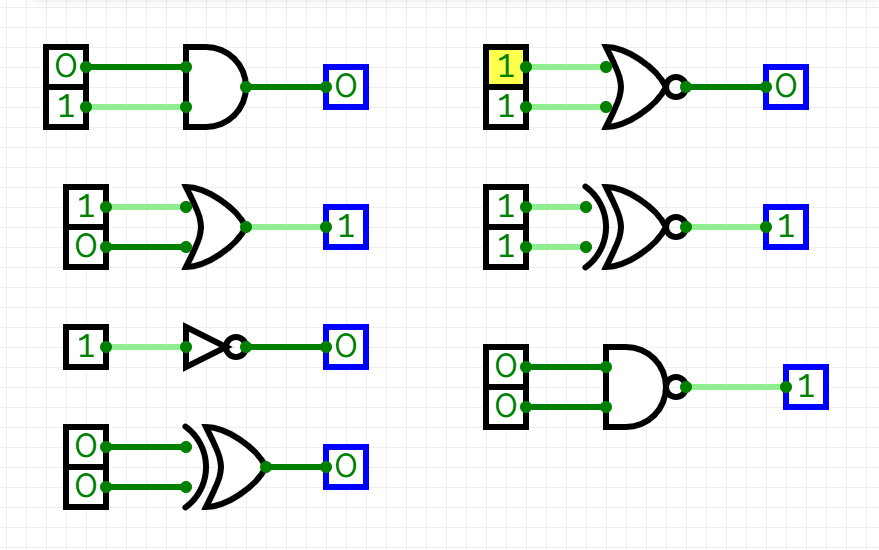
Truth Table

|  |  |  |
| --- | --- | --- |
| Input A | Input B | Output |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

1. NOT Gate:

* Output is at Logic 0 when its only input is at Logic 1, and at Logic 1 when its only input is at Logic 0, Also called Inverter.
* Boolean Representation-: X=

Circuit Diagram



Truth Table

|  |  |  |
| --- | --- | --- |
| Input A | Input B | Output |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Boolean Algebra

Circuit Simplification Using Boolean Algebra-

The algebraic method used to simplify digital circuits applies a number of Boolean laws to successively simplify complex equations. Selected laws and rules are applied, step by step, to the original equation, so as to eventually arrive at a simplified version that can be implemented with a smaller number of gates and therefore lead to a simpler circuit.

Boolean Laws –

The laws of Boolean algebra are similar in some ways to those of standard algebra, but in some cases Boolean laws are unique. This is because when logic is applied to digital circuits, any variable such as A can only have two values 1 or 0, whereas in standard algebra A can have many values.

1. Commutative Laws- In a group of variables connected by operators AND or OR, the order of the variables does not matter.

1a. Boolean addition (OR): A+B = B+A

1b. Boolean multiplication (AND): A•B = B•A

1. Associative Laws- The order of calculation can be changed without affecting the result (Change which terms are in brackets, or remove brackets). Note: This is only OK so long as all signs (+ or •) are the same.

2a. Boolean addition (OR): (A+B)+C = A+(B+C) = A+B+C

2b. Boolean Multiplication (AND): (A•B)•C = A•(B•C) = A•B•C = ABC

1. Distributive Laws- The same answer is arrived at when multiplying (ANDing) a variable by a group of bracketed variables added (ORed) together, as when each multiplication (AND) is performed separately.Law 3a is similar to factoring in normal algebra, but law 3b is unique to Boolean algebra because unlike normal algebra, where A x A=A2, in Boolean algebra A•A = A

3a. A•(B+C) = A•B+A•C

3b. A+(B•C) = (A+B) • (A+C)

1. Identity Elements- In rule 4a, when the variable A is ANDed with logic 1 (called the Identity Element for the AND operator). The variable ANDed with 1 retains its identity.

Rule 4b, shows that the Identity Element for the OR operator is 0, and any variable (e.g. A) ORed with 0 it retains its identity.

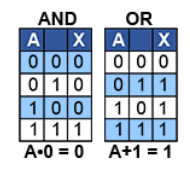
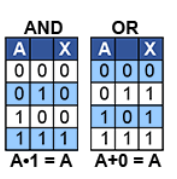
4a. A•1 = A

4b. A+0 = A

5a and 5b show how by ‘forcing the Identity Element’, (in B column of the truth tables) to the opposite states to those used in 4a and 4b, produces an output that is the same as the Identity Element.

5a. A•0 = 0

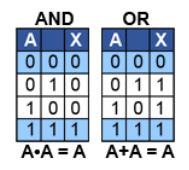
5b. A+1 = 1



6a and 6b show that ANDing or ORing two identical variables, produces an output equal to a single variable, showing that one of the variables is redundant, a useful rule when simplifying Boolean equations.

6a. A•A = A

6b. A+A = A



1. Complement Law-

7a. A + = 1 Any variable ORed with its inverse is 1 7b. A • = 0 Any variable ANDed with its inverse is 0

NOTE – = A Double inversion (NOT NOT) returns the variable to its previous state.

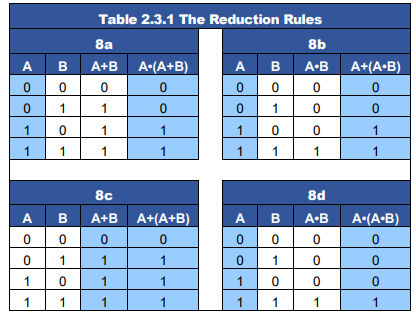
1. Reduction-

8a. When a single variable (A) is ANDed with itself OR a second variable (A+B), the result is equal to the single variable.

8a A• (A+B) = A

8b. When a single variable (A) is ORed with itself AND a second variable (A•B), the result is equal to the single variable.

8b A+ (A•B) = A

8c. When a single variable (A) is ORed with itself OR a second variable (A+B), the single variable disappears.

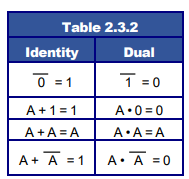
8c A+ (A+B) = (A+B)

8d. When a single variable (A) is ANDed with itself AND a second variable (A•B), the single variable disappears.

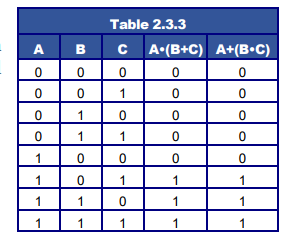
8d A• (A•B) = (A•B)

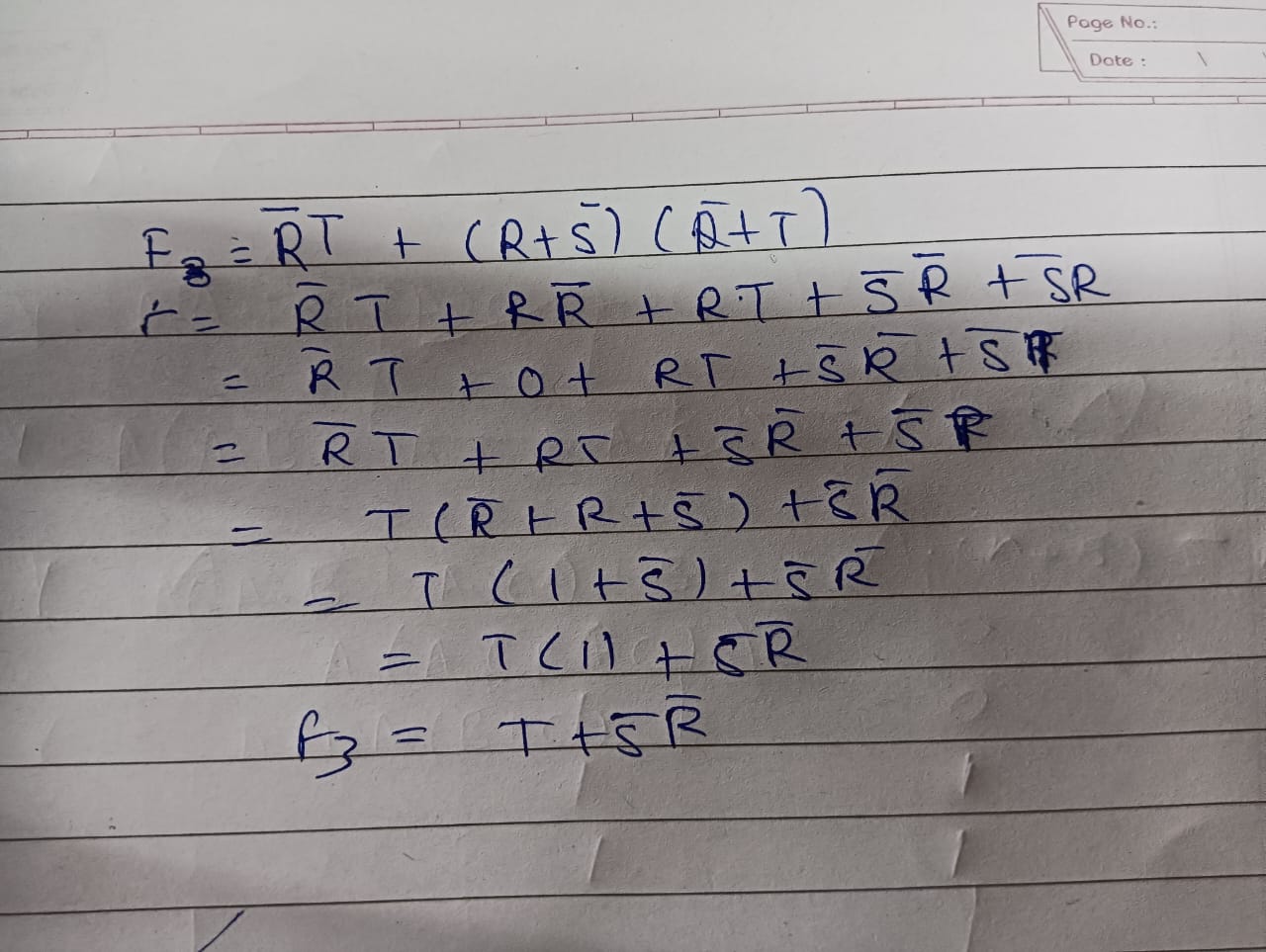
1. Duality Rules-

It is possible to derive additional identities by obtaining the Dual of an identity. This involves changing the AND operators to OR and the OR operators to AND. Additionally any 0s are changed to 1s and 1s to 0s as shown in Table 2.3.2.



The duality rule can be used to change a logic expression containing both AND and OR elements to its equivalent dual expression. Table 2.3.3 shows that A•(B+C) is the same as A+(B•C).





Before simplification

A diagram of a computer network

AI-generated content may be incorrect.

A diagram of a fish

AI-generated content may be incorrect.

Truth Table:

|  |  |  |  |
| --- | --- | --- | --- |
| Input R | Input S | Input T | Output |
|  | 0 | 0 | 0 |
|  | 0 | 1 | 1 |
|  | 1 | 0 | 1 |
|  | 1 | 1 | 1 |

**Conclusion:**

In this assignment, I learned about the different laws of Boolean algebra and their applications. I also studied various logic gates, implemented them, and verified their functions using CircuitVerse. To apply what I learned, I simplified a complex Boolean equation using these laws. This helped me understand the concepts better and see how they are used in digital logic design.