Assignment No.1:

Verification of Logic Gates and Boolean Expressions

Name: Amit Pravin Jain

Branch: Artificial Intelligence & Data Science

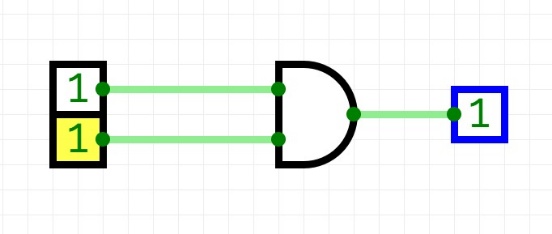
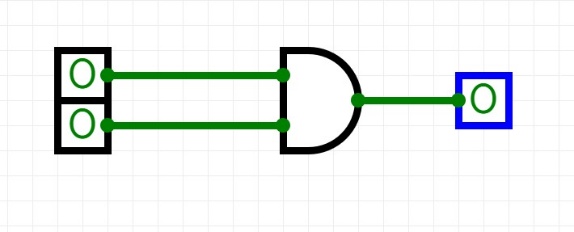
Div- B Roll no.= 66 PRN No.12412777

Subject: FOIDS

**Logic Gates**

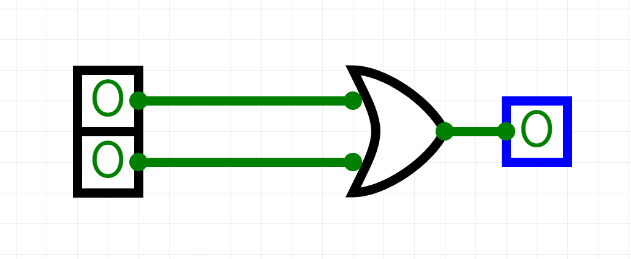
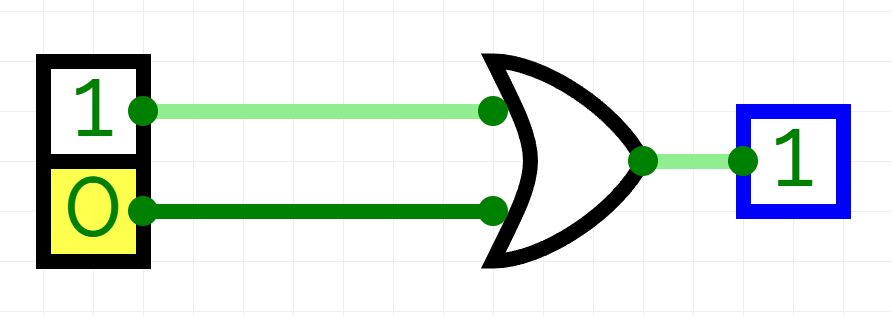
* Logic Gates are the fundamental building blocks in digital electronics. There are basically seven main types of logic gates that are used to perform various logical operations in digital systems.
* Digital logic is the foundation, not only of computing but also many other electronic devices and control 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.
* The Seven Logic gates and their truth tables are as follows:

1. **AND Gate:**
   1. An AND gate is a logic gate.In this gate, if either of the inputs is low (0), then the output is also low. If all of the inputs are high (1), then the output will also be high.
   2. The Boolean Representation of AND Gate : **X=A.B**
   3. Their Visual Representation as:

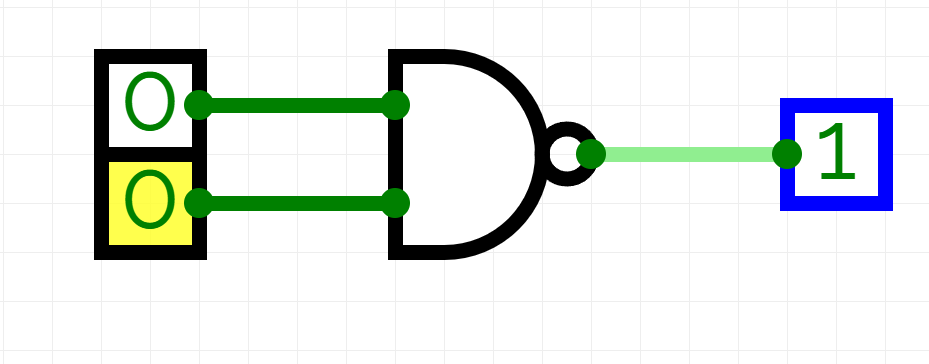
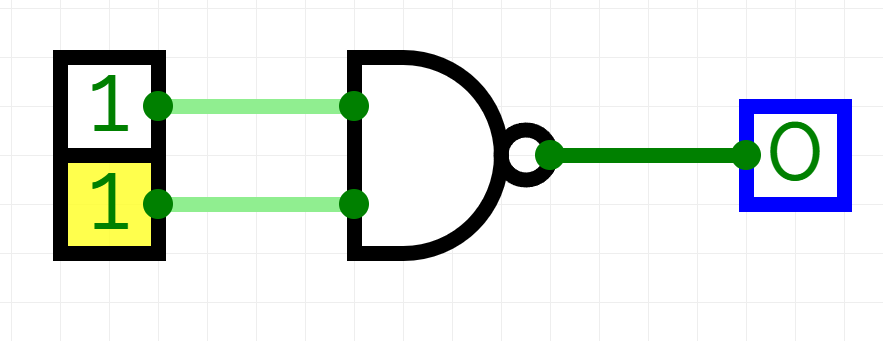


|  |  |  |
| --- | --- | --- |
| Input | | Output |
| A | B | X |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

1. **OR Gate:**
   1. An OR gate is a logic gate.In this gate, if either of the inputs is High (1), then the output is also High. If all of the inputs are low (0), then the output will also be low.
   2. The Boolean Representation of OR Gate : **X=A+B**
   3. Their Visual Representation as:



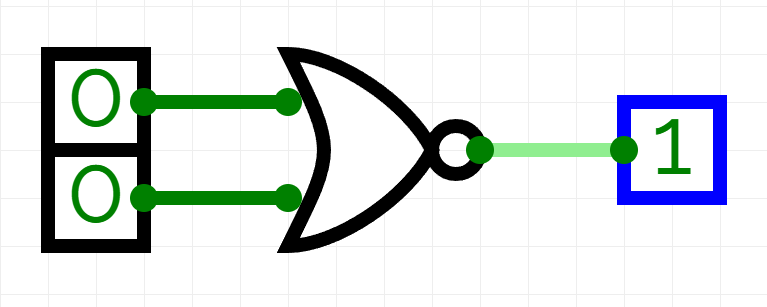
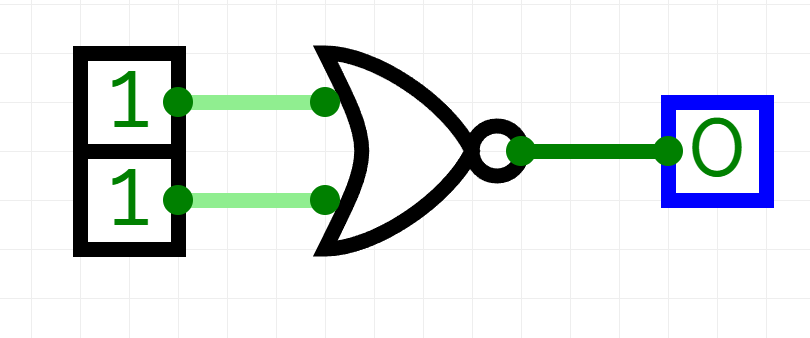
|  |  |  |
| --- | --- | --- |
| Input | | Output |
| A | B | X |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

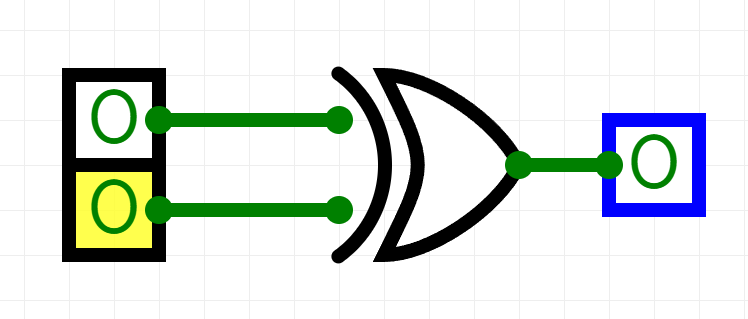
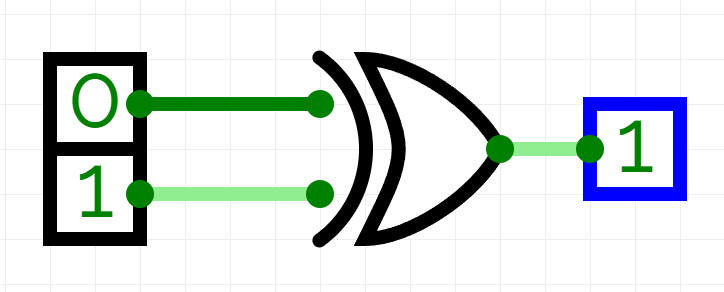
1. **NAND Gate:**
   1. is at Logic 0 when, and only when all its inputs are at Logic 1, otherwise the output is at Logic
   2. The Boolean Representation of NAND Gate : X=
   3. Their Visual Representation as:

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

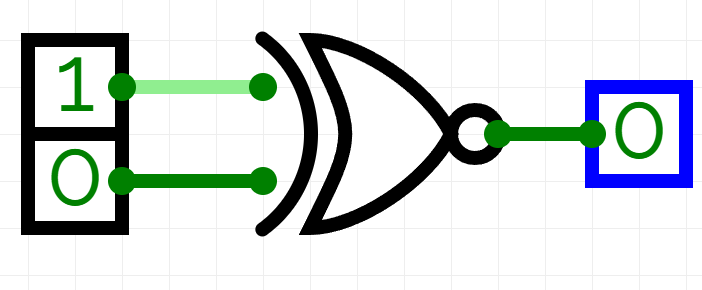
1. **NOR Gate:**
   1. Output is at Logic 0 when one or more inputs are at Logic 1. If all the inputs are at Logic 0, the output is at Logic 1.
   2. The Boolean Representation of NOR Gate :X=

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

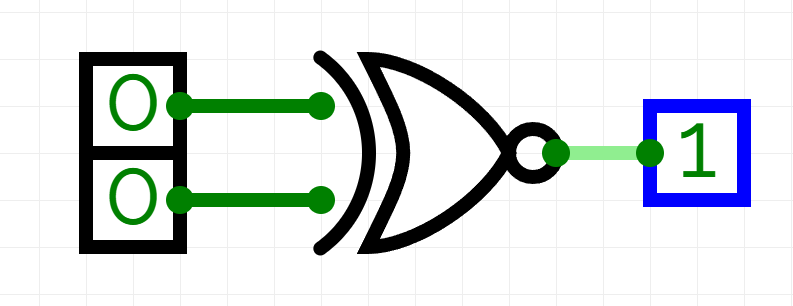
* 1. ****Their Visual Representation:

1. **XOR Gate:**
   1. Output is at Logic 1 when one and only one of its inputs is at Logic 1, otherwise it is at Logic 0.
   2. Boolean Representation-: X=
   3. ****Their Visual Representation:

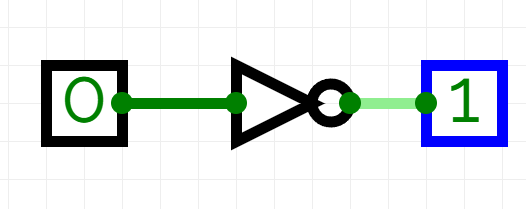
|  |  |  |
| --- | --- | --- |
| Input | | Output |
| A | B | X |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

1. **XNOR Gate:**
   1. 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.
   2. Boolean Representation-: X=
   3. ****Their Visual Representation:

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



1. **Not Gate:**
   1. 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.
   2. Boolean Representation-: X=
   3. Their Visual Representation:



|  |  |
| --- | --- |
| Input | Output |
| 0 | 1 |
| 1 | 0 |

**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.**

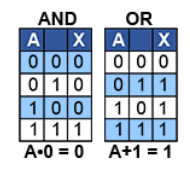
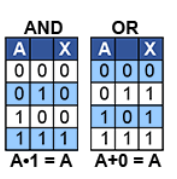
**4a. A•1 = A**

**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**

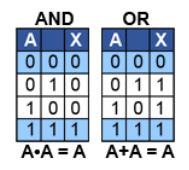
**5b. A+1 = 1**

****

**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**

****

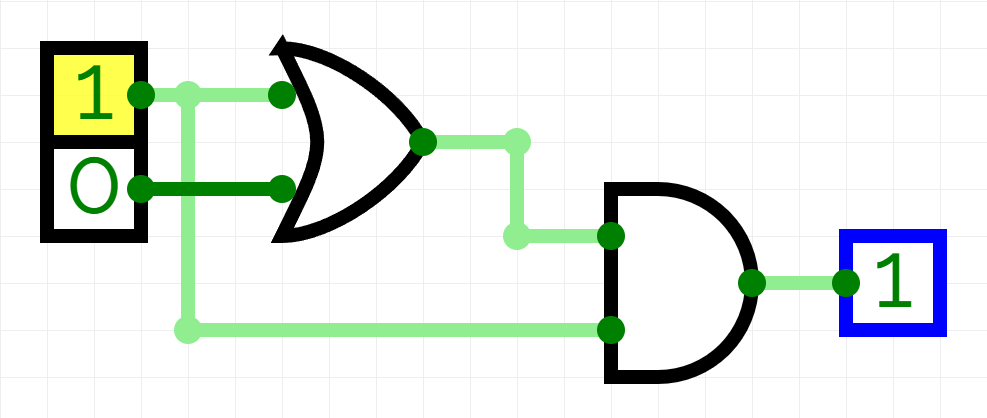
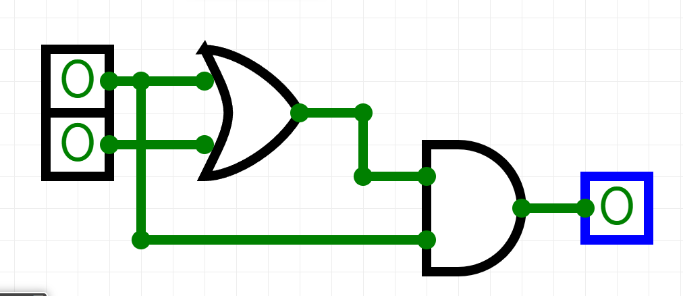
1. **Complement Law-**

**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**

**6.Reduction Laws:**

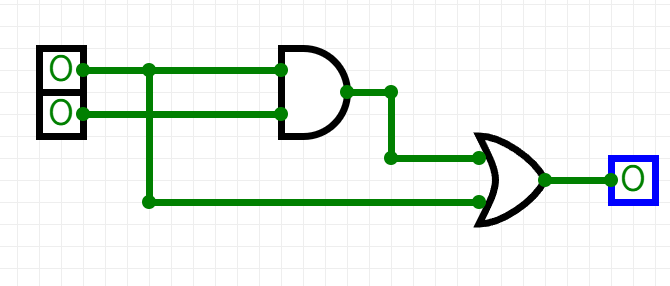
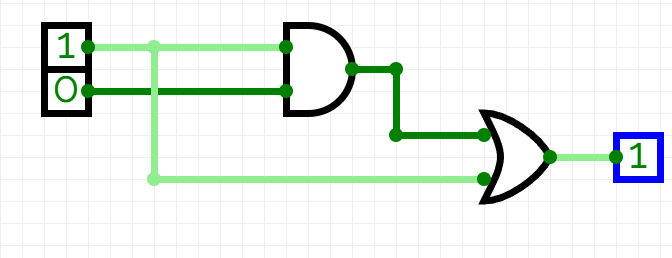
**1. (8a)** When a single variable (A) is AND with itself OR a second variable (A+B), the result is equal to the single variable.

**A• (A+B) = A**

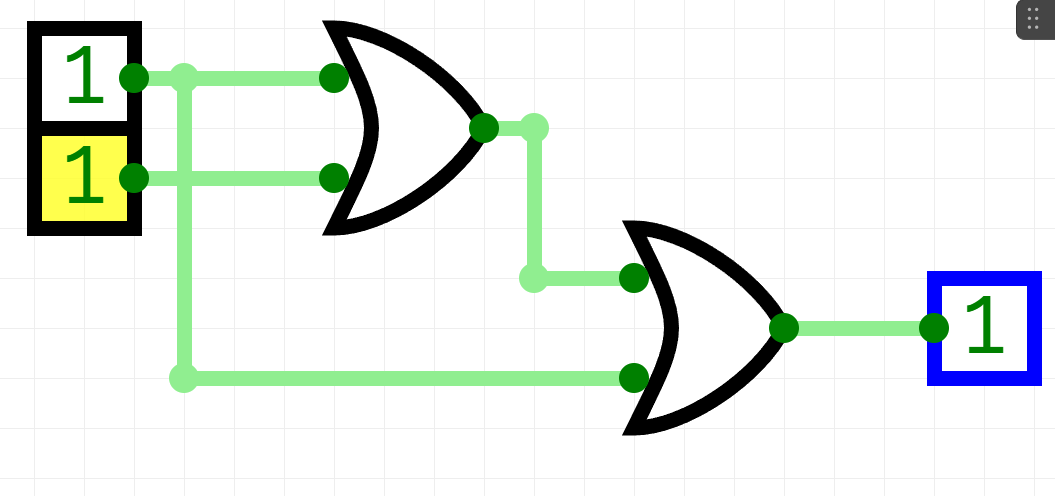
|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **A+B** | **A.(A+B)** |
| **0** | **0** | **0** | **0** |
| **0** | **1** | **1** | **0** |
| **1** | **0** | **1** | **1** |
| **1** | **1** | **1** | **1** |

**2.(8b)** When a single variable (A) is OR with itself AND a second variable (A•B), the result is equal to the single variable.

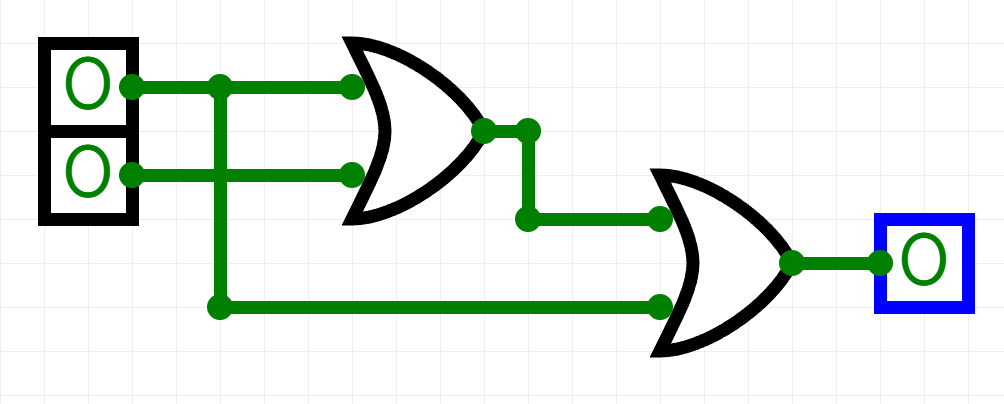
|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **A.B** | **A+ (A•B)** |
| **0** | **0** | **0** | **0** |
| **0** | **1** | **0** | **0** |
| **1** | **0** | **0** | **1** |
| **1** | **1** | **1** | **1** |

**A+ (A•B) = A**

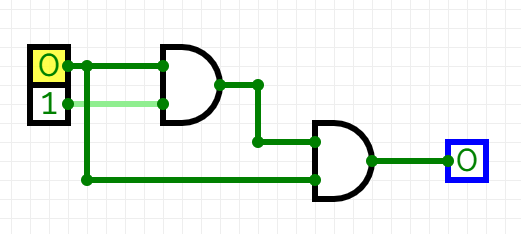
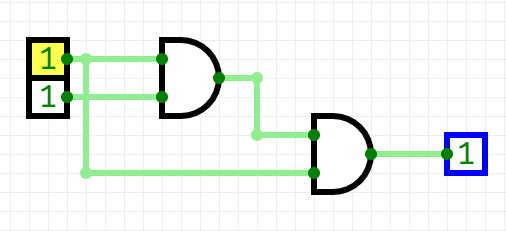
**3.(8c)** When a single variable (A) is ORed with itself OR a second variable (A+B), the single variable disappears.

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

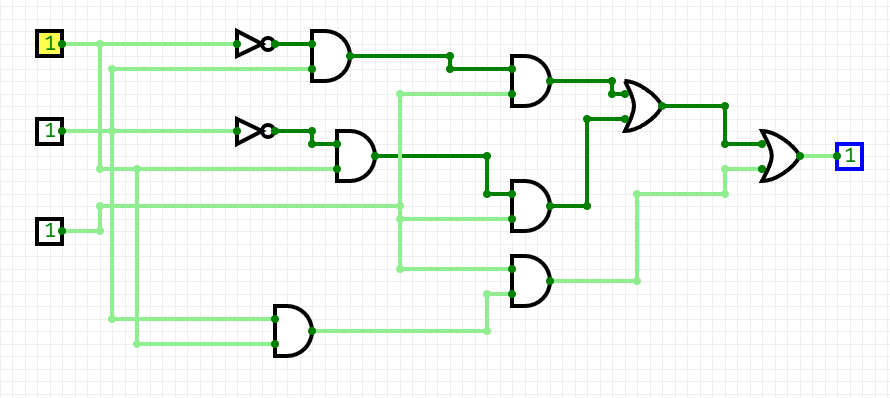
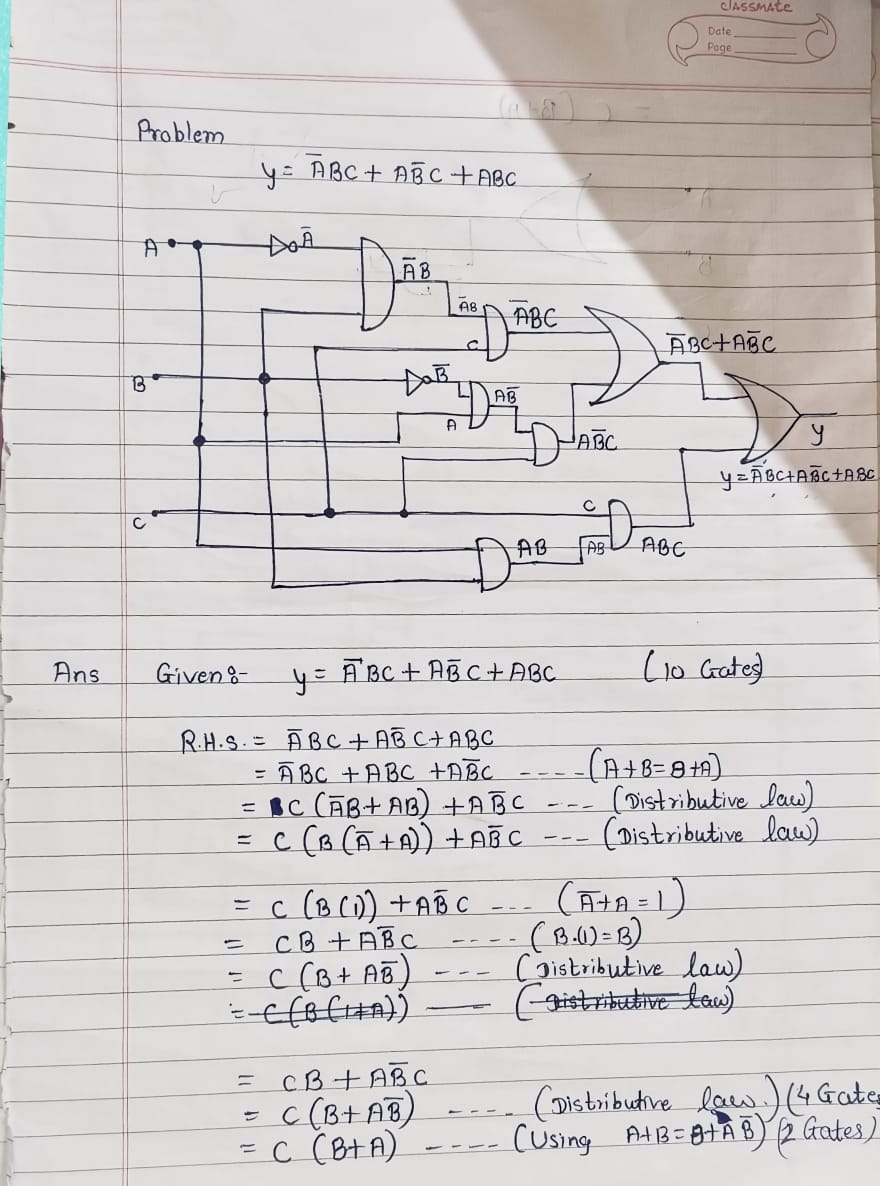
|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **A+B** | **A+ (A+B)** |
| **0** | **0** | **0** | **0** |
| **0** | **1** | **1** | **1** |
| **1** | **0** | **1** | **1** |
| **1** | **1** | **1** | **1** |

****

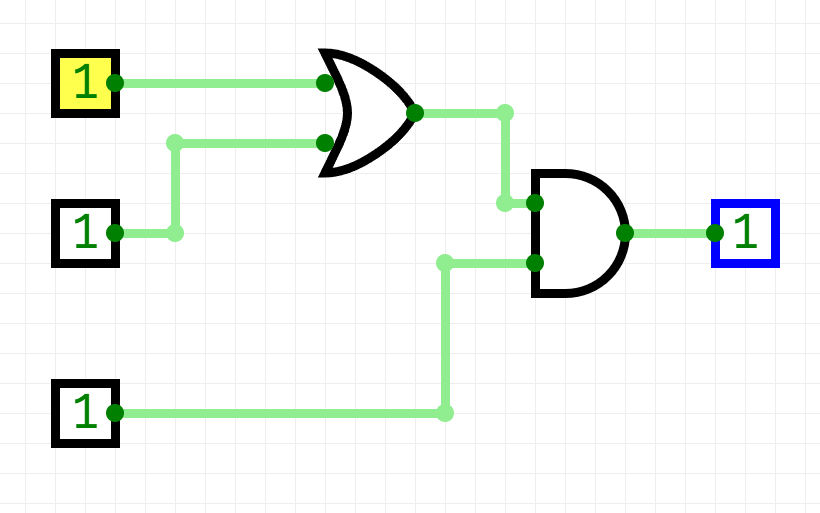
**4.(8d)** When a single variable (A) is ANDed with itself AND a second variable (A•B), the single variable disappears.

**A• (A•B) = (A•B)**

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **A.B** | **A.(A.B)** |
| **0** | **0** | **0** | **0** |
| **0** | **1** | **0** | **0** |
| **1** | **0** | **0** | **0** |
| **1** | **1** | **1** | **1** |

******Before Simplification:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | | | **Output** | |
| **A** | **B** | **C** | **Y=A’BC+AB’C+ABC** | **Y=C.(A+B)** |
| **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **1** | **0** | **0** |
| **0** | **1** | **0** | **0** | **0** |
| **0** | **1** | **1** | **1** | **1** |
| **1** | **0** | **0** | **0** | **0** |
| **1** | **0** | **1** | **1** | **1** |
| **1** | **1** | **0** | **0** | **0** |
| **1** | **1** | **1** | **1** | **1** |

**Conclusions:**

The Boolean expression Y = ¬A·B·C + A·B·¬C + A·B·C was simplified using basic Boolean laws to a much simpler form: Y = C(A + B). This simplification cuts down the number of logic gates from 10 to just 2, making the circuit easier to build and more efficient to run. The truth table clearly shows that both versions give the same result for all input combinations,