|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Digital Electronics (116U40C303)** | **Semester:** | **III** |
| **Date of Performance:** | **Kirti Sawalani** | **Batch No:** | **A - 3** |
| **Faculty Name:** | **03 / 08 / 2023** | **Roll No:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** | **\_\_\_ / 25** |

**Experiment No.: 2**

**Title: Implementation of Boolean Expressions**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| 1. Minimize and implement the given Boolean expression using    1. Boolean Algebra   ii. Kmap |

|  |
| --- |
| **COs to be achieved:** |
| **CO1**: Use different techniques of minimization for combinational logic design. |

|  |
| --- |
| **Theory:** |
| Boolean algebra, the Karnaugh map, and the Quine-McClusky method can all be utilized to minimize logic functions. Minimizing these functions leads to efficient circuit design by reducing complexity.  **Algebraic Method of Boolean Expressions:**  The algebraic method involves using Boolean theorems to transform a given boolean expression into an equivalent expression. By applying these theorems, the expression can be simplified and optimized.   1. **Identity Laws:**    * Identity Law 1: A + 0 = A This equation states that the logical OR operation (represented by '+') between any variable 'A' and the logical 0 (which represents false) always yields 'A' as the result.    * Identity Law 2: A · 1 = A This equation states that the logical AND operation (represented by '·') between any variable 'A' and the logical 1 (which represents true) always yields 'A' as the result. 2. **Null Laws:**    * Null Law 1: A + A' = 1 This equation states that the logical OR operation between any variable 'A' and its negation (represented by 'A' with a dash or an apostrophe) always results in true.    * Null Law 2: A · A' = 0 This equation states that the logical AND operation between any variable 'A' and its negation always results in false. 3. **Idempotent Laws:**    * Idempotent Law 1: A + A = A This equation states that the logical OR operation between a variable 'A' and itself always yields 'A' as the result.    * Idempotent Law 2: A · A = A This equation states that the logical AND operation between a variable 'A' and itself always yields 'A' as the result. 4. **Commutative Laws:**    * Commutative Law 1: A + B = B + A This equation states that the order of variables does not matter in the logical OR operation.    * Commutative Law 2: A · B = B · A This equation states that the order of variables does not matter in the logical AND operation. 5. **Associative Laws:**    * Associative Law 1: (A + B) + C = A + (B + C) This equation states that the grouping of variables does not affect the logical OR operation.    * Associative Law 2: (A · B) · C = A · (B · C) This equation states that the grouping of variables does not affect the logical AND operation. 6. **Distributive Laws:**    * Distributive Law 1: A · (B + C) = (A · B) + (A · C) This equation states that the logical AND operation distributes over the logical OR operation.    * Distributive Law 2: A + (B · C) = (A + B) · (A + C) This equation states that the logical OR operation distributes over the logical AND operation. 7. **Absorption Laws:**    * Absorption Law 1: A + (A · B) = A This equation states that the logical OR operation between a variable 'A' and the logical AND operation between 'A' and 'B' is equivalent to 'A' alone.    * Absorption Law 2: A · (A + B) = A This equation states that the logical AND operation between a variable 'A' and the logical OR operation between 'A' and 'B' is equivalent to 'A' alone. 8. **De Morgan's Laws:**    * De Morgan's Law 1: (A + B)' = A' · B' This equation states that the negation of the logical OR operation between 'A' and 'B' is equivalent to the logical AND operation between the negation of 'A' and the negation of 'B'.    * De Morgan's Law 2: (A · B)' = A' + B' This equation states that the negation of the logical AND operation between 'A' and 'B' is equivalent to the logical OR operation between the negation of 'A' and the negation of 'B'. 9. **Double Negation Law:**    * Double Negation Law: (A')' = A This equation states that the negation of the negation of a variable 'A' is equivalent to 'A' itself. 10. **Universal Bound Laws:**     * Universal Bound Law 1: A + 1 = 1 This equation states that the logical OR operation between any variable 'A' and the logical 1 always results in true.     * Universal Bound Law 2: A · 0 = 0 This equation states that the logical AND operation between any variable 'A' and the logical 0 always results in false. 11. **Negation Laws:**     * Negation Law 1: A + A' = 1 This equation states that the logical OR operation between a variable 'A' and its negation 'A' with a dash or an apostrophe is always true.     * Negation Law 2: A · A' = 0 This equation states that the logical AND operation between a variable 'A' and its negation 'A' with a dash or an apostrophe is always false. 12. **Involution Law:**     * Involution Law: (A')' = A This equation states that the double negation of a variable 'A' is equivalent to 'A' itself. 13. **Null Element Laws:**     * Null Element Law 1: A + 0 = A This equation states that the logical OR operation between any variable 'A' and the logical 0 (representing false) always results in 'A'.     * Null Element Law 2: A · 1 = A This equation states that the logical AND operation between any variable 'A' and the logical 1 (representing true) always results in 'A'. 14. **Domination Laws:**     * Domination Law 1: A + A·B = A This equation states that the logical OR operation between a variable 'A' and the logical AND operation between 'A' and 'B' is always 'A'.     * Domination Law 2: A · (A + B) = A This equation states that the logical AND operation between a variable 'A' and the logical OR operation between 'A' and 'B' is always 'A'. 15. **Consensus Law:**     * Consensus Law: A · B + A' · C + B · C = A · B + A' · C This equation states that in a logical expression, if two terms have a common factor and one term differs in one variable (negated or non-negated), the common factor can be factored out.   **Karnaugh Maps:**  The Karnaugh map, also known as the K-map, is a graphical method used to simplify Boolean expressions and minimize logical functions. It provides a systematic approach to simplifying Boolean algebraic expressions by grouping minterms or maxterms based on their binary representations.  The Karnaugh map is represented as a grid, where each cell corresponds to a unique combination of input variables. The number of rows and columns in the grid is determined by the number of variables in the Boolean expression. For example, if there are two variables, the Karnaugh map would have 2 rows and 2 columns.  To use the Karnaugh map, you need to determine the minterms or maxterms that are true (1) in the Boolean expression and locate them on the K-map. The cells in the Karnaugh map are typically labeled with binary representations based on the variable values. Starting from the top-left cell, the labels progress in a Gray code order, with adjacent cells differing by only one bit.  Once you have located the true minterms or maxterms on the Karnaugh map, you can proceed with simplifying the Boolean expression by identifying groups of adjacent cells that form rectangles or squares with a size of powers of 2 (1, 2, 4, 8, etc.). These groups are called "implicants."  There are two main types of groups in the Karnaugh map:   1. Prime Implicants: These are the largest possible groups that cover as many minterms or maxterms as possible. Prime implicants are formed by grouping cells horizontally or vertically (not diagonally) and must cover adjacent cells. They can span any number of cells that is a power of 2 (1, 2, 4, 8, etc.). 2. Essential Prime Implicants: These are prime implicants that cover at least one minterm or maxterm that is not covered by any other prime implicant. They are essential for obtaining the minimum expression.   Once you have identified the prime implicants and essential prime implicants, you can select a combination of prime implicants to cover all the true minterms or maxterms while minimizing the number of terms in the simplified expression. This selection can be done manually or using algorithms like the Quine-McClusky method.  **Quine-McClusky Method:**  When the number of variables in a Boolean expression exceeds 6, using the Karnaugh map can become challenging. The Quine-McClusky method provides a systematic tabular approach to solving such expressions. Even with a larger number of variables, this method can be efficiently solved using programming techniques. |

|  |
| --- |
| **Circuit Diagram/ Block Diagram:** |
| **Solve using Boolean algebra:**  xy + xz + yz = xy + x\_z + yz(x + x\_)  = xy + x\_z + xyz + x\_yz  = xy(1 + z) + x\_z(1 + y)  = xy + xz.  Implement both sides and prove that the result is the same. Also, implement using universal gates.  ***One sample question is solved here. Each faculty will give the problem.***  **Solve using K map:**  *F = A’B’C’+ B’CD’ + A’BCD’+ AB’C’*      Reduced expression is: B’C’ + A’CD’ + B’D’  Implement the circuit using basic gates and universal gates. |

|  |
| --- |
| **Step-wise Procedure:** |
| 1. Verify all gates. 2. Verify all Logic equations. 3. Reduce the expression using Boolean algebra. 4. Implement /Simulate using basic gates and universal gates. 5. Reduce the given expression using the K map. 6. Implement/simulate using basic gates and universal gates. |

|  |
| --- |
| **Implementation:** |
| * **NOT Gate (7404) –**  |  |  | | --- | --- | | **A** | **Output** | | 0 | 1 | | 1 | 0 |  * **AND Gate –**  |  |  |  | | --- | --- | --- | | **A** | **B** | **A • B** | | 0 | 0 | 0 | | 0 | 1 | 0 | | 1 | 0 | 0 | | 1 | 1 | 1 |      * **OR Gate –**  |  |  |  | | --- | --- | --- | | **A** | **B** | **A + B** | | 0 | 0 | 0 | | 0 | 1 | 1 | | 1 | 0 | 1 | | 1 | 1 | 1 |      * **NAND Gate (7400) –**  |  |  |  | | --- | --- | --- | | **A** | **B** |  | | 0 | 0 | 1 | | 0 | 1 | 1 | | 1 | 0 | 1 | | 1 | 1 | 0 |      * **NOR Gate (7402) –**  |  |  |  | | --- | --- | --- | | **A** | **B** |  | | 0 | 0 | 1 | | 0 | 1 | 0 | | 1 | 0 | 0 | | 1 | 1 | 0 |      * **XOR Gate (7486) –**  |  |  |  | | --- | --- | --- | | **A** | **B** | **O/P** | | 0 | 0 | 0 | | 0 | 1 | 1 | | 1 | 0 | 1 | | 1 | 1 | 0 |      * **De Morgan's Laws (Boolean laws) Verification:**     Circuit Implementation    Truth Table   * ***F = A’B’C’+ B’CD’ + A’BCD’+ AB’C’*** :     Circuit Implementation    Circuit Diagram   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | A | B | C | D | A’ | B’ | C’ | D’ | A’B’ | A’B’C’ | B’C | B’CD’ | A’BCD’ | AB’C’ | X | | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |   Truth table for F |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| 1. **A safe has five locks, *v*, *w*, *x*, *y*, and *z*, all of which must be unlocked for the safe to open. The keys to the locks are distributed among five executives in the following manner:**   ***A has keys for locks v and x;***  ***B has keys for locks v and y;***  ***C has keys for locks w and y;***  ***D has keys for locks x and z;***  ***E has keys for locks v and z.***   1. **What is the minimum number of executives needed to unlock all five locks on the safe?** 2. **What are the different combinations of executives that can unlock the safe? Write an expression, f(A, B, C, D, E), to indicate when the safe can be opened based on the presence of executives.** 3. **Who is the executive that is necessary in every combination to successfully open the safe?** 4. **In a game, there are six keys labeled A, B, C, D, E, and F, and six locks labeled 1, 2, 3, 4, 5, and 6. Each lock must be unlocked to complete the game. The keys are distributed among five players as follows:**   ***Player 1 has keys for locks A, B, and C.***  ***Player 2 has keys for locks B, C, and D.***  ***Player 3 has keys for locks C, D, and E.***  ***Player 4 has keys for locks D, E, and F.***  ***Player 5 has keys for locks E, F, and A.***   1. **Determine the minimum number of players required to unlock all the locks.** 2. **Find all the combinations of players that can unlock the locks. Write an expression, f(P1, P2, P3, P4, P5), to indicate when all locks can be opened based on the presence of players.** 3. **Identify the player who is essential in every combination to successfully unlock all the locks.** 4. **A puzzle consists of four colored switches: Red (R), Green (G), Blue (B), and Yellow (Y). To solve the puzzle, all four switches must be turned on. Four friends, Alex, Ben, Chris, and Dan, each have control over one or more switches as follows:**   ***Alex can control switches R and G.***  ***Ben can control switches G and B.***  ***Chris can control switches B and Y.***  ***Dan can control switches Y and R.***   1. **Determine the minimum number of friends needed to solve the puzzle.** 2. **Find all the combinations of friends that can solve the puzzle.** 3. **Write an expression, f(A, B, C, D), to indicate when the puzzle can be solved based on the presence of friends.** 4. **Identify the friend who is essential in every combination to successfully solve the puzzle.** |

|  |
| --- |
| **Conclusion:** |
| Learnt to assemble circuits using basic gates and verified them using truth table. |

**Signature of faculty in-charge with Date:**