MODULE-II

GATE LEVEL MINIMIZATION

The Map Method, Four Variable K-Map, Product-of-Sums and Sum-of-Products Simplification, Don't Care Conditions, NAND and NOR Implementations, Other Two Level Implementations, Exclusive–OR function.

2.1 The Map Method:

The Map Method, sometimes known as the Mapping Method, is a technique used in various academic disciplines to visually represent data or information.

The intricacy of digital logic gates, which carry out a Boolean function, is directly linked to the intricacy of the algebraic equation utilised to perform that function. While the truth table representation of a function is singular, its algebraic expression can have various equivalent forms. Boolean expressions can be simplified using algebraic approaches. Nevertheless, the method of reduction is seen cumbersome due to its lack of explicit guidelines for anticipating further steps in the manipulation technique.

The mapping method explained in the following section offers a highly efficient and direct approach to minimising Boolean functions. The Karnaugh map, often known as the K-map, is a visual representation of a truth table that is widely recognised in academic circles. A Karnaugh map is a graphical tool consisting of squares, where each square represents a single minterm of the function that needs to be minimised. Given that any Boolean function can be represented as the sum of minterms, it is possible to generate a visual representation of the function using a grid of squares, with each square representing a minterm that is part of the function.

The map visually illustrates the various potential manifestations of a function in its standard form. Through the process of identifying and analysing different patterns, users have the ability to generate multiple algebraic formulations that accurately reflect the same function. Subsequently, they can assess and contrast these expressions in order to select the most straightforward and succinct option. The expressions produced by the map consistently adhere to either the sum of products or product of sums standard forms. An

algebraic expression is said to be in its simplest form when it contains the fewest number of terms and the fewest number of variables in each term. As a consequence, the circuit design achieves a low gate count and minimises the number of inputs to each gate. It will be demonstrated that the most basic form is not always singular; there are cases where two or more forms satisfy the conditions for simplification. In such instances, any of the proposed resolutions would be deemed satisfactory.

Two-variable k-map:

A two-variable Karnaugh map (K-map) can represent a total of four distinct combinations of the input variables A and B, which is equal to 2 raised to the power of 2 (i.e., 2^2 = 4). Each of these pairings, AB,A'B,AB', and A'B'(in SOP form), refers to a specific term in Boolean algebra. The minterms can be denoted by their decimal counterparts:
m_0 relates to A'B' , A'B corresponds to m1, m2 corresponds to $A\overline{B}$, and m3 corresponds to AB, where A is the most significant bit (MSB). The symbol "m" denotes a minterm, with the subscript indicating the decimal value of the minterm. The inclusion or exclusion of a minterm in the expression controls whether the output of the logic circuit is logic 1 or logic 0 for a specific combination of input variables.

Consider the formula $f = \overline{A}B + A\overline{B} + AB$.

This can be equivalently represented using minterm notation as:

The equation F = m1 + m2 + m3 can be expressed as the sum of the masses m(1, 2, 3).
The equation F = m1 + m2 + m3 represents the sum of masses (m1, m2, m3) in the system.

The function F is represented by the sum of the minterms 1, 2, and 3.
The application of a truth table will be employed in this analysis.

Min term	Inputs A	Inputs B	Output F
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

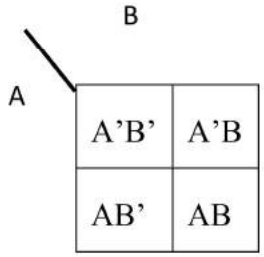
A min-term is a term in Boolean algebra that consists of all variables in their complement the user's inputs. The presence of a 1 in the output signifies the inclusion of a specific min-term in the total, whereas a 0 in same column shows the absence of the particular min-term in the equation for the output. The aforementioned data can alternatively be represented using a 2-variable K-map.

The two-variable K- map comprises a total of 4 cells, each representing a unique min-term. Every every cell on the K- map represents a distinct min-term. The allocation of min-term labels to these cells signifies

the output expressions that the min-terms represent. If a cell has a value of 0 or is blank, it indicates that the corresponding min-term is not part of the expression for the output.

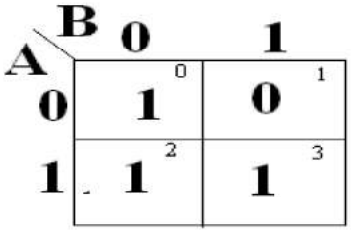
EX: The min-terms of a two-variable K- map f=*A' B*+A*B'* +AB. The mapping of the expressions =∑m(0,2,3) is as follows. The Karnaugh map representation of the sum of min-terms (1,2,3) is requested.

Mapping of SOP Expression's:



The minterms of a two-variable k-map

The mapping of the expressions =∑m(0,2,3)is



k-map of ∑m(0,2,3)

Minimizations of SOP expressions:

In order to reduce Boolean expressions expressed in the Sum of Products (SOP) form using the Karnaugh map (k-map), one should identify adjacent squares with 1's representing minterms that are adjacent to each other. These adjacent squares can then be combined to produce bigger squares, resulting in the elimination of certain variables. Two squares are considered to be nearby when their minterms exhibit a difference in only a single variable. For instance, the sets A and B differ solely in one variable, thus allowing for their combination to generate a 2-square that eliminates the variable B. This principle can be applied to other sets as well.

The essential requirement for the adjacency of minterms is that their decimal representations must exhibit a difference equivalent to a power of 2. A minterm has the capability to be amalgamated with any number of neighboring minterms to construct larger squares. When two minterms are adjacent to one another, they can be merged to create a larger square referred to as a 2-square or a pair. This process effectively eliminates one variable, specifically the variable that is not shared by both minterms.

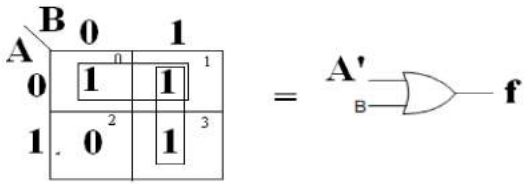
The preceding equation, expressed in terms of minterms, can be reduced as follows: F = m0 + m1 + m2 + m3 = m∑(0, 1, 3). The accompanying picture illustrates the Karnaugh map for f and its subsequent reduction. In one of the 2-square configurations, variable A remains constant at a value of 0, while variable B exhibits variation from 0 to 1. Conversely, in the other 2-square configuration, variable B remains constant

at a value of 1, while variable A exhibits variation from 0 to 1. The phrase A + B can be simplified to its reduced form.

Ex:

The equation f = A + B can be represented in Sum of Products (SOP) form using a Karnaugh map. Additionally, a logic diagram can be constructed to visually depict the logical operations involved in this equation.

It requires two gate inputs for realization as



f=A +B(k-map in SOP form, and logic diagram.)

The primary consideration in the design of a digital circuit is to minimize its cost. In order to implement the circuit, it is necessary for the expression employed to be minimum. The cost of the circuit is directly proportional to the number of gate inputs it contains. Therefore, an expression is deemed minimal only if it corresponds to the smallest feasible number of gate inputs. There is no assurance that the Karnaugh map (K-map) in the Sum of Products (SOP) form represents the true minimum expression. In

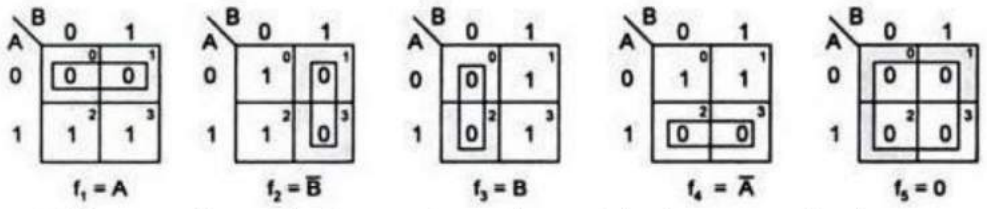
order to achieve the most concise expression, it is necessary to determine the minimal expression in both the Sum of Products (SOP) and Product of Sums (POS) forms by utilizing k-maps. Subsequently, the minimum value between these two minimal expressions should be selected.

The "1's" on the Karnaugh map signify the inclusion of minterms in the output expressions, while the "0's" indicate the exclusion of minterms. Conversely, the absence of a minterm in the sum of products (SOP) expression implies the presence of the corresponding maxterm in the product of sums (POS) expression. When plotting a SOP expression on the Karnaugh map, the "0's" or lack of entries on the map represent the maxterms. In order to get the smallest expression in the Product of Sums (POS) form, it is necessary to analyze the locations of 0s on the Karnaugh map and apply the same approach employed for combining 1s. Additionally, it should be noted that the absence of a maxterm in the product of sums (POS) expression implies the presence of the corresponding minterm in the sum of products (SOP) expression. When a POS expression is graphically represented on a Karnaugh map, the presence of 1s or the absence of entries on the map indicate the minterms.

Mapping of POS expressions:

Each typical POS expression total term is maxterm. A two-variable function (A, B) has four maxterms: A+B, A+*B*', *A* '+B.

M0, M1, M2, and M3. The capital letter M stands for maxterm, and its subscript is its decimal designation, which is achieved by interpreting the non-complemented variable as a 0 and the complemented variable as a 1 and putting them side by side to read the binary number's decimal.



When mapping a POS expression to the k-map, 0s are inserted in the squares for the maxterms in the expression and d1s for the maxterms that are not in the expression. Maxterm and minterm squares share the same decimal. K-maps with two variables maxterms Possible two-variable k-map maxterm groupings

Minimization of POS Expressions:

The term "maxterm" is used to refer to each individual sum term in the standard POS statement. The function in two variables (A, B) exhibits four potential maxterms, namely A+B, A+*B*', *A* '+B, and *A*' +*B*'.

The different representations are denoted as M0, M1, M2, and M3. The uppercase letter M represents a maxterm, with its subscript indicating the decimal value assigned to that maxterm. This value is determined

by considering the non-complemented variable as 0 and the complemented variable as 1, and then combining them to generate a binary number. The decimal equivalent of this binary number is used as the subscript for the maxterm.

When mapping a POS expression onto a Karnaugh map, zeros (0s) are assigned to the squares that correspond to the maxterms included in the expression. On the other hand, ones (1s) are assigned to the squares that correspond to the maxterms not contained in the expression. The decimal representation of the squares of the squares for maxterms is equivalent to that of the minterms. A two-variable Karnaugh map and its corresponding maxterms represent the maxterms of the Karnaugh map.

In order to derive the smallest expression in the form of a product of sums (POS), the provided POS expression should be mapped onto a Karnaugh map. Subsequently, the adjacent 0s should be combined into

the largest possible squares. The squares should be examined, assigning a value of 1 to the complemented variable if it remains constant, and a value of 0 to the non-complemented variable if it remains constant, disregarding variables that do not remain constant within the square. Subsequently, these squares should be expressed as a sum term.

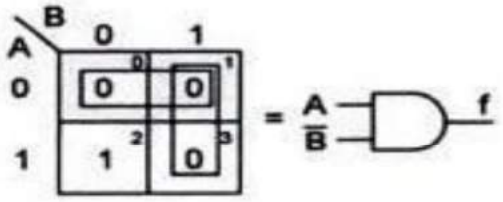
The graphic displays different combinations of maxterms and their corresponding reduced expressions.

In the above scenario, the variable denoted as "f1" is observed to have a consistent value of "A" equal to zero across the entire square. Conversely, the variable "B" undergoes a transition from zero to one. The value of f2 is interpreted as B prime (B') due to the fact that B remains constant with a value of 1, while A transitions from 0 to 1 along the square. The user's text "f5" does not provide any context or information to be rewritten

The value of "read" is considered to be zero in this context due to the simultaneous changes occurring in both variables within the square.

Ex: To simplify the phrase f=(A+B)(A+B')(A'+B'), we can employ the technique of mapping.

The expression f can be represented in terms of maxterms as f = πM(0,1,3). The realization of the reduced expression necessitates the utilization of two gate inputs.



The equation F = AB' represents a logical expression in which F is the result of the Karnaugh map (K-map) is a graphical representation tool used in digital logic design to simplify Boolean expressions. It is particularly useful for transforming Boolean expressions into their product of sums (POS) form. The POS form represents a logical expression as a sum of many product terms. In addition to the K-map, the maxterm M2 is not present in the supplied statement. The presence of a 1 on the k-map signifies this. The SOP expression corresponding to the given expression is represented as the sum of minterms, or AB'. The aforementioned realization is analogous to the one pertaining to the POS form.

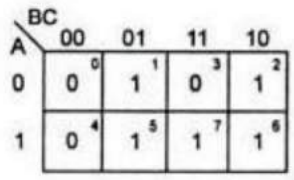
Three-variable K-map:

Minimization of SOP and POS expressions:

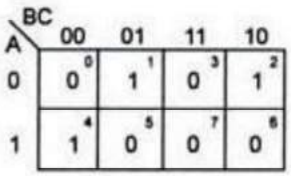
The three-variable Karnaugh map (K-map) is a graphical representation used in digital logic design to simplify Boolean expressions and minimize the standard Sum of Products (SOP) function with three variables (A, B, C) can have eight combinations: A B C, AB C, A BC, A BC, AB C, AB C, ABC, and ABC. Minterms are the permutations m0, m1, m2, m3, m4, m5, m6, and m7. A and C are the minterm designator's most and least important bits.Ex: Map the expression f= (A+B+C),$(A' + B + C')$ $(A' + B' + C')$(A' + B' + C')$(A' + B' + C')$

Let us denote the above expression as f = A'BC + AB'C + A'B'C' + AB'C + ABC. The provided statement can be represented by the following minterms: A' B'C = 001 (m1), AB'C = 101 (m5), A' BC' = 010 (m2), ABC' = 110 (m6), and ABC = 111 (m7).

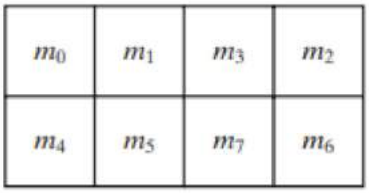
The given expression can be written as f=∑m(1,5,2,6,7), which simplifies to f=∑m(1,2,5,6,7). The Karnaugh map (K-map) is a graphical tool used in digital logic design to simplify Boolean expressions and represent them in Sum of Products (SOP) form.



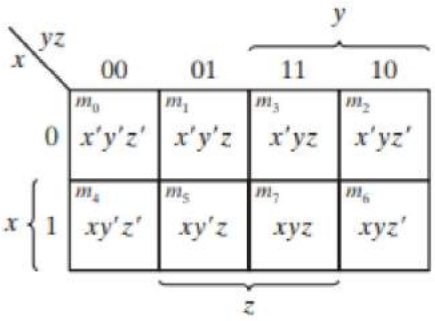
(a) K-map in SOP form



(b) K-map in POS form



(a)



(b)

The formula f can be represented as the logical conjunction of five terms: (A+B+C), (A' + B + C'), (A' + B' + C'), (A' + B' + C), and (A' + B' + C).

Reading the K-maps:

The provided equation can be represented by the following maxterms: A+B+C=000=M0, A' + B + C' = 101=M5, A' + B' + C' = 111=M7, A' + B + C = 011=M3, and A' + B' + C = 110=M6.

The given expression can be written as f = π M (0,5,7,3,6), which is equivalent to f = π M (0,3,5,6,7). The process of mapping the expression is

f5= m0+m1+m2+m3

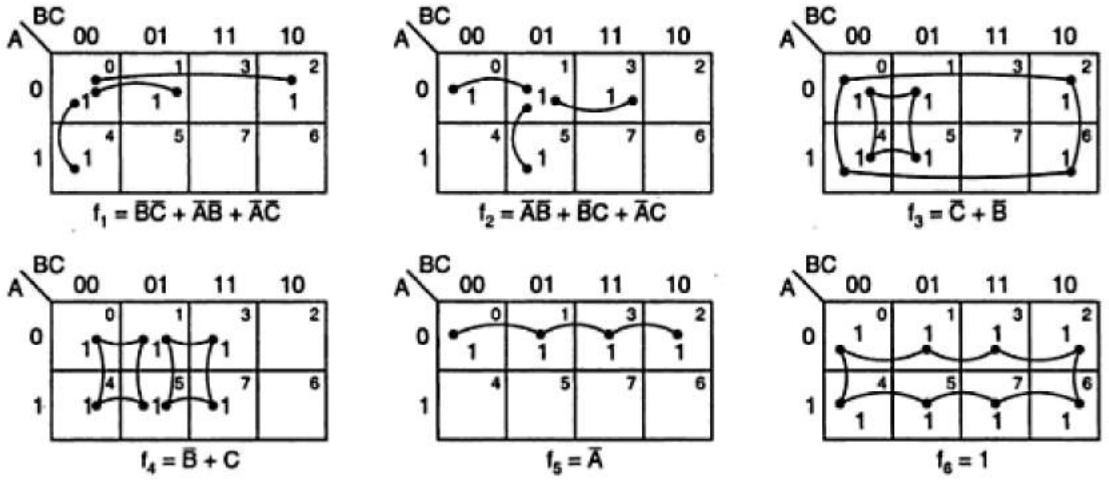
$$= A'B'C' + A'B'C + A'BC' + A'BC$$

$$= A'B'(C' + C) + A'B(C + C')$$

$$=A' B'+A' B$$

$$=A' (B'+B)=A'$$

The Karnaugh map (K-map) is a graphical representation used in digital logic design to simplify Boolean expressions. It is particularly useful for expressing Boolean functions in the product of sums (POS) form.



The objective of this study is to explore the minimization techniques for Sum of Products (SOP) and Product of Sums (POS) expressions.

To simplify the Boolean expressions in Sum of Products (Product of Sums) form shown on the k-map, analyze the presence of 1s (0s) on the map. The aforementioned terms are indicative of the minterms (maxterms). The task at hand involves identifying nearby minterms (maxterms) and subsequently merging them into larger squares. The process of merging neighboring squares in a Karnaugh map that contains either 1s or 0s, with the intention of simplifying a Sum of Products (SOP) or Product of Sums (POS) expression, is referred to as looping. Certain minterms (maxterms) may exhibit a multitude of adjacencies. It is advisable to begin the process by identifying the minterms (or maxterms) that have the fewest amount of adjacencies, and then attempt to construct the largest feasible square. The larger object must be arranged in the shape of a geometric square or rectangle. These structures can be generated through the process of wrapping, but are not achievable by diagonal arrangements. Next, let us evaluate the minterm (or maxterm) that has the second lowest number of adjacencies and attempt to construct the largest feasible square using this term. Continue this process until all of the minterms (or maxterms) have been addressed. A minterm (maxterm) may be included in many squares if it aids in the process of

reduction. The minimal expression can be determined by examining the squares formed in the Karnaugh map. It is possible for there to exist multiple minimum expressions.

Two squares are considered adjacent to each other based on their physical proximity or the possibility of being made adjacent by wrapping around, as indicated by the binary designations along the top and left side of the map being in Gray code. In order for squares to be capable of being combined into larger squares, it is necessary, however not solely reliant, on the condition that their minterm designations exhibit a difference equivalent to a power of two.

The following is a general approach for simplifying Boolean expressions:

Construct the Karnaugh map and allocate the values of 1s (0s) to represent the minterms (maxterms) associated with the Sum of Products (Product of Sums) expression.

Examine the Karnaugh map for instances of isolated 1s (0s) that are not in proximity to any other 1 (0) entries. The above terms can be classified as isolated minterms (or maxterms). The given elements should be interpreted in their original form, as they are unable to be merged even into a two-dimensional arrangement.

Please verify the presence of quadrilaterals (consisting of four squares) and octagons (consisting of eight squares) composed of contiguous 1s (or 0s), even if they include some 1s (or 0s) that have already been merged. The objects in question must conform to a geometric shape that is either a square or a rectangle.

The task at hand involves identifying any instances of individual units (represented as 1s or 0s) that have not yet been merged, and subsequently consolidating them into larger square formations, provided that such combinations are feasible.Construct the most concise mathematical phrase by adding (multiplying) the product of the terms within each group.

The process of interpreting the Karnaugh maps:

When examining the reduced k-map in SOP (POS) form, the variable that consistently appears as 0 within the square is denoted as the complemented (non-complemented) variable. Conversely, the variable that consistently appears as 1 within the square is denoted as the non-complemented (complemented) variable. Additionally, the term is expressed as a product (sum) term. The summation of all product terms is performed by addition, while the multiplication of all product terms is carried out by multiplication.

Figure 1 displays several potential combinations of minterms and their corresponding minimal expressions extracted from the k-maps. In this context, it is worth noting that f6 is interpreted as 1 since there are no variables that remain constant along the 8-square. The value of F5 is denoted as " " due to the dynamic changes in variables B and C within the 4-square matrix formed by m0, m1, m2, and m3. Meanwhile, variable A remains constant with a value of 0. In terms of algebraic manipulation,

The equation f5 = m0 + m1 + m2 + m3 represents the sum of the masses m0, m1

The expression can be rewritten as follows: A'BC' + A'BC + AB'C' + ABC

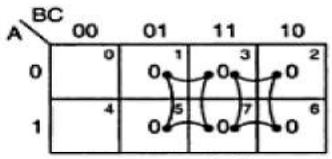
The expression A'B'(C' + C) + A'B(C + C') can be rewritten in an academic manner as follows: A'B'(C' + C) + A'B(C + C').

The expression "A' B' + A' B" can be rewritten in an academic manner as "The sum of the complement of A and the complement of B, the expression A' (B' + B) simplifies to A'.

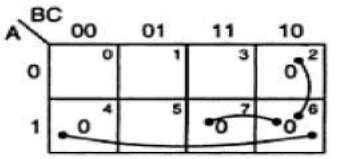
The expression f3 is interpreted as the complement of C (C') added to the complement of B (B'), due to the dynamic changes of variables A and B within the 4-square formed by m0, m2, m6, and m4. Meanwhile, the variable C remains constant with a value of 0. The reading of the given information is denoted as option C. In the 4-square matrix comprising of elements m0, m1, m4, and m5, variables A and C exhibit variability, whereas variable B remains constant at a value of 0. The given text is interpreted as the letter B. The resulting expression for f3 is the summation of the two terms, specifically C' plus B'.

The logical function f1 can be represented as the Boolean expression $B'C' + A' B' + A' C'$. This expression is derived from the observation that in the 2-square formed by m0 and m4, the variable A transitions from a logic 0 to a logic 1. While B' and C' remain constant at a value of 0. The notation is pronounced as "B prime C prime." In the 2-square matrix defined by m0 and m1, the value of C is transitioning from 0 to 1, while the values of A and B remain constant at 0. The notation used is A' B'. Within the 2-square formed by m0 and m2, the value of B transitions from 0 to 1, while A and C remain constant at 0. The notation for the given expression is denoted as A'C'. Hence, the resulting sum of products (SOP) expression can be written as B' C' + A' B' + B' C'.

The k-map provides minimal POS expressions for various maxterm groupings.



(a) $f_1 = (\overline{C})(\overline{B})$



(b) $f_2 = (\overline{A} + \overline{B})(\overline{B} + C)(\overline{A} + C)$

In the depicted diagram, the values of A and B within the 4-square region defined by M1, M3, M7, and M5 are transitioning from 0 to 1, while the value of C remains constant at 1. The notation "C'" is pronounced as "C prime." Within the 4-squad consisting of M3, M2, M7, and M6, variables A and C undergo a transition from a value of 0 to a value of 1. However, the value of B remains consistent at a value of 1. The notation is interpreted as B prime. The minimum expression, denoted as f1, is obtained by multiplying the complement of variable C (C') with the complement of variable B (B'). Additionally, in the above diagram, within the 2-square created by M4 and M6, the variable B transitions from a logic 0 to a logic 1, while variable A remains constant at a logic 1 and variable C remains constant at a logic 0. Please interpret the text as '+C. In a similar vein, the 2-square created by the intersection of M7 and M6 is denoted as A' + B', whereas the 2-square produced by M2 and M6 is denoted as B' + C. The minimum expression, denoted as f2, is obtained by multiplying the sum terms, namely (A' + C), (A' + B'), and (B' + C).

To simplify the phrase f=∑m(0,2,3,4,5,6) utilizing mapping techniques, we will proceed to construct it in AOI logic and NAND logic. The pictures below depict the Sop k-map and its reduction, as well as the implementation of the minimal expression using AOI logic and the related NAND logic.

Ex: In the context of the Sum of Products (SOP) Karnaugh map, the process of reduction is performed in the following manner:

M5 possesses a sole adjacency, namely M4. Consequently, it is recommended to amalgamate M5 and M4 in order to form a square. In this scenario, the value of variable A remains constant at 1, while the value of variable B remains constant at 0. However, the value of variable C varies between the range of 0 to 1. Please interpret the given text as "AB".

M3 possesses a sole adjacency, namely M2. Consequently, it is proposed to amalgamate M3 and M2 in order to form a square configuration. In this scenario, the value of variable A remains consistently at 0, whereas the value of variable B maintains consistently at 1. However, the value of variable C fluctuates between 1 and 0. Please interpret it as "A B."

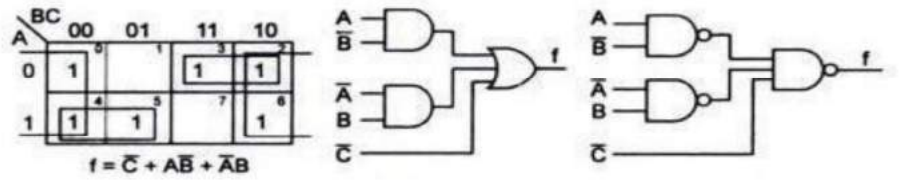
The element m6 has the ability to build a 2-square with m2, whereas m4 can form a 2-square with m0. However, it is important to note that when the map is wrapped from left to right, the elements m0,

m4, m2, and m6 can form a 4-square. Both m2 and m4 have been previously merged, however they can still be exploited once more. Please proceed with the task. Within the context of this 4-square, variable A undergoes a transition from a value of 0 to 1, while variable B similarly undergoes a transition from 0 to 1. However, variable C remains constant at a value of 0. Please read the text as "C."

Please provide a list of all the product terms in Sum of Products (SOP) form. The minimum sum of products (SOP) expression is the minimum value of the function is denoted as fmin.

The Karnaugh map (K-map) is a graphical representation used in digital logic design to simplify Boolean expressions. It is a visual tool that aids in the implementation of logic functions. The AND-OR-INVERT (AOI) logic is

fmin=

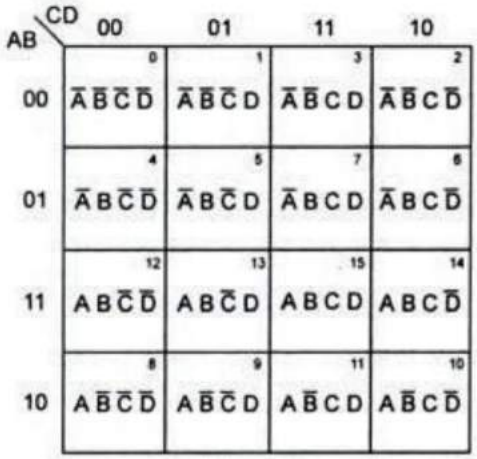


k-map **AOI logic** **NAND logic**

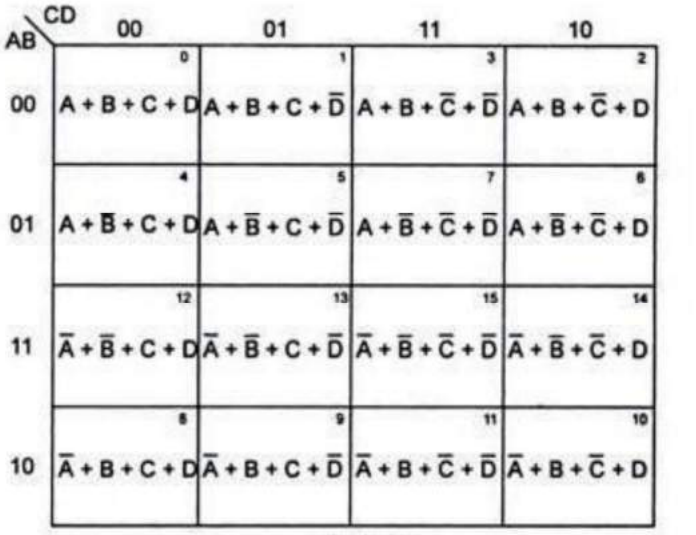
2.2 Four variable k-maps:

Four-variable Karnaugh map expressions can exhibit a total of 16 possible combinations of input variables, denoted as A, B, C, and D. These combinations can be represented in sum-of-products (SOP) form using minterm designations m0, m1, ..., m15, respectively. Additionally, the same combinations can be expressed in product-of-sums (POS) form using maxterm designations M0, M1, ..., M15, respectively. The object in question possesses a total of 24 squares or cells, which can be further simplified to 16. The rows and columns in the designations of binary numbers are represented using the gray code. The concept of Adjacency ordering involves the arrangement of numbers in a sequence, where the numbers 01 and 10 are followed by the number 11.

The subject of this discussion pertains to the Standard Operating Procedure (SOP) form and the Point of Sale (POS) form.

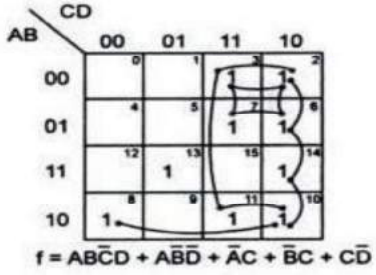


(a)SOP form



(b)POS form

EX: Reduce using mapping the expression $\sum(2,3,6,7,8,10,11,13,14)$



The subject of this discussion pertains to the Standard Operating Procedure (SOP) form and the Point of Sale (POS) form.

2.3 Product- of - Sums Simplification

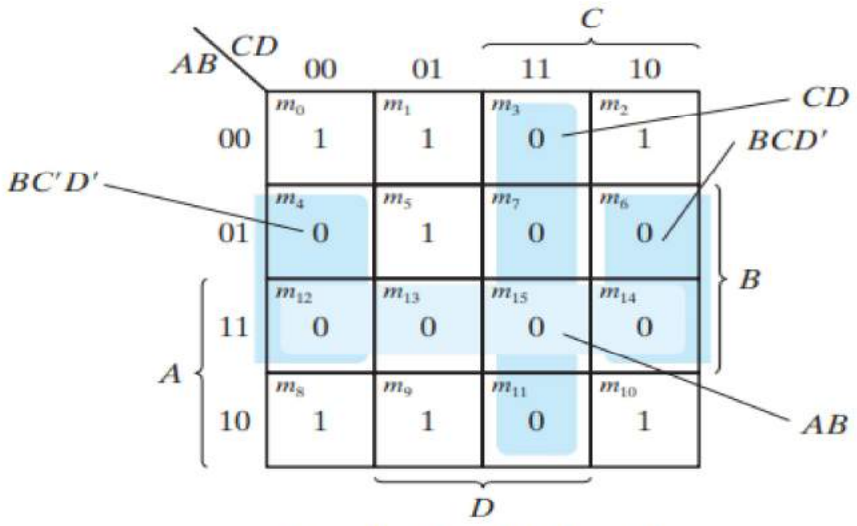
The process of acquiring a minimized function in the form of a product-of-sums is derived from the fundamental characteristics of Boolean functions. The numerical value of "1" assigned to each square on the map corresponds to the minterms of the function. The minterms that are absent from the conventional sum-of-products representation of a function indicate the complement of said function. Based on this discovery, it can be observed that the complement of a function is visually depicted in the map as the squares that are not indicated by the presence of 1's. By assigning the value of 0 to the empty squares and merging them with neighboring squares that form valid combinations, we may get a simplified sum-of-products formula for the complement of the function, denoted as F'. The function F in product-of-sums form can be obtained by taking the complement of F' (as a consequence of DeMorgan's theorem). The function obtained is automatically in product-of-sums form according to the generalized DeMorgan's theorem.

Ex: The user's text will be rewritten to be more academic without adding any additional information. The Boolean function F (A, B, C, D) =0, 1, 2, 5, 8, 9, 10) can be simplified into (a) sum-of-products form and (b) product-of-sums form.

The proposed solution is as follows:

The locations denoted as "1" in the map seen in Figure 1. Please provide a comprehensive representation of all the minterms associated with the given function. The squares labeled with 0's indicate the minterms that are not part of the function F, thereby representing the complement of F. The simplified function in sum-of-products form, obtained by combining the squares with 1's, can be expressed as follows: (a) F = B'D' + B'C' + A'C'D.

By combining the squares labeled with 0's, as seen in the diagram, we derive the simplified complemented function: F' = AB + CD + BD'.

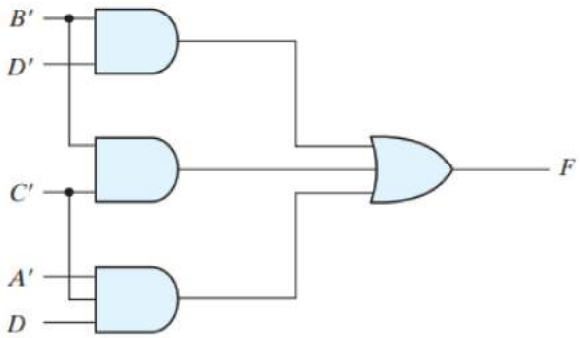


Note: $BC'D' + BCD' = BD'$

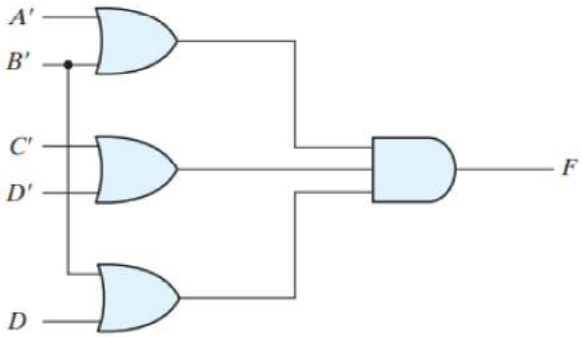
By employing De Morgan's theorem, the simplified function in product-of-sums form can be derived by taking the dual and complementing each literal.

The formula F is derived from the logical expression (A' + B') (C' + D') (B' + D).

The gate-level implementation of the simpler expressions derived in the previous example is depicted in Figure below.



(a) $F = B'D' + B'C' + A'C'D$



(b) $F = (A' + B')(C' + D')(B' + D)$

The implementation of the sum-of-products expression involves utilizing a collection of AND gates, where each AND gate corresponds to an individual AND term. The outputs of the AND gates are thereafter linked to the inputs of a singular OR gate. The function is also implemented in (b) using a product-of-sums form, where a set of OR gates is employed, with each OR gate corresponding to an individual OR term. The outputs of the OR gates are interconnected with the inputs of a singular AND gate.

2.4 Don’t care combinations:

Sometimes the output value is unknown owing to poor input combinations or insignificance. Combinations without experiment values are called "don't care" combinations. These combinations are invalid or insignificant due to the lack of output value definition. Combinations without expression values are called "don't care" or optional. These expressions are incomplete. Invalid combinations produce a "don't care" value.

The Boolean function F (w, x, y, z) = 1, 3, 7, 11, 15) with the associated don't-care conditions d (w, x, y, z) = 0, 2, 5) is to be simplified.

The proposed solution is as follows:

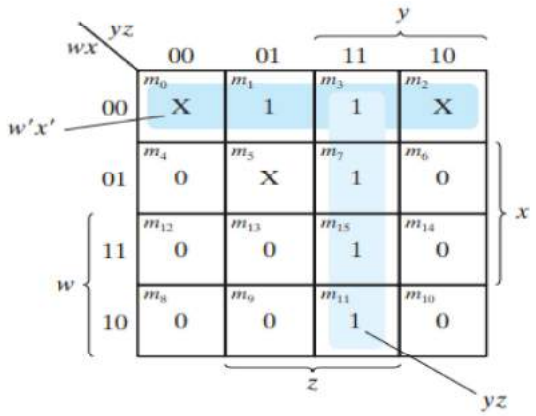
The minterms of the function F represent the specific combinations of variables that result in the function evaluating to a logical value of 1. The minterms of variable d represent the don't-care conditions, which can be assigned either a value of 0 or 1. The process of map simplification is illustrated in Figure 1. The minterms of the function F are denoted by the presence of 1's, while the minterms of the function d are represented by X's. The remaining squares are filled with 0's. In order to obtain the simplified expression in sum-of-products form, it is necessary to incorporate all five instances of the value 1 in the map. However, the inclusion or exclusion of any of the X's is contingent upon the specific simplification method employed for the function. The term "yz" encompasses the four minterms located in the third column. The minterm m1 can be logically paired with minterm m3 to get the three-literal term w’x’z. Nevertheless, the incorporation of one or two contiguous X's enables the amalgamation of four contiguous squares, resulting in a two-literal word. In Figure.In the simplified function, the don't-care minterms 0 and 2 are incorporated alongside the 1's.

The equation F = yz + w'x' is a mathematical expression involving variables y, z, w, and x.

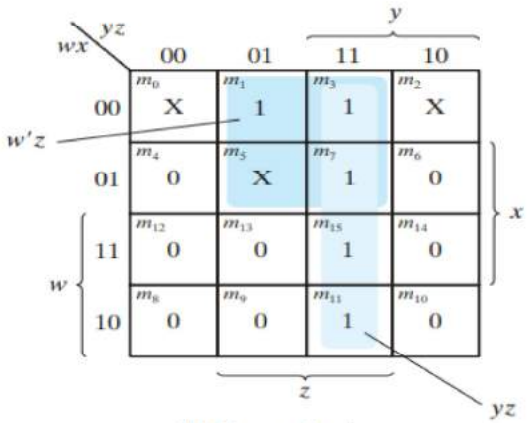
In Figure (b), the don't-care minterm 5 is incorporated with the 1's, resulting in the simplified function.

The equation F = yz + w'z is a mathematical expression using variables y, z, and w.

$$F = yz + w'z$$



(a) $F = yz + w'x'$



(b) $F = yz + w'z$

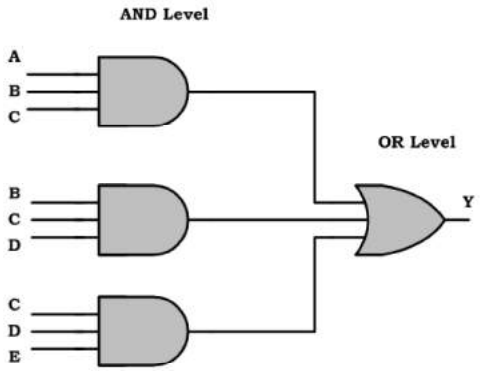
2.4 NAND–NOR implementations

It is possible to implement any Boolean expression using NAND-NOR gates. To begin, it is necessary to convert the given expression into either Sum of Products (SOP) or Product of Sums (POS) form. NAND and NOR gates are commonly referred to as universal gates due to their ability to implement any Boolean statement without the need for additional gates.

Logic circuits are commonly transformed into NAND or NOR circuits due to their extensive commercial utilization and their inherent logical properties. The concept of wired logic involves the utilization of wires to achieve logical operations, specifically the NOR gate, without the need for its physical implementation. The utilization of NAND-NOR implementation results in a reduction in the quantity of gates employed and a decrease in the overall circuit dimensions. Consequently, this leads to a decrease in both the cost of implementation and the amount of electricity consumed. Another factor to consider is the simplicity and cost-effectiveness of the fabrication process for NAND and NOR gates.

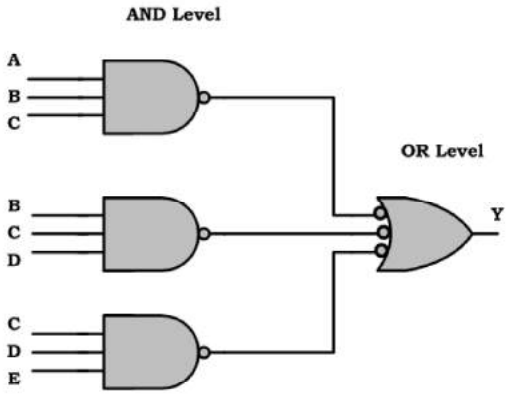
The equation can be expressed as Y = ABC + BCD + CDE.

The first step



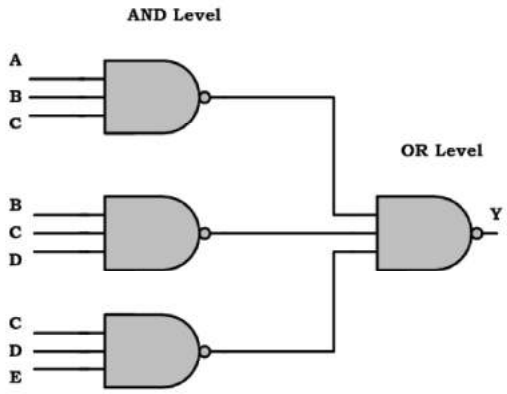
The second step

Bubbles are injected at the output of the AND gate and at the input of the OR gate, respectively.



The third step

The utilization of a NAND gate as a replacement for a bubbled input OR gate is being considered. Currently, all of the gates have been replaced with NAND gates.

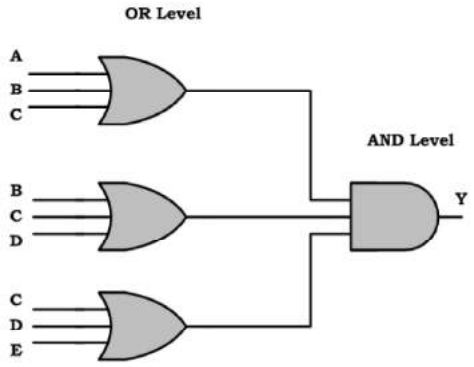


Example 2:

To realize the above statement using NOR gates, the appropriate logic gates need to be interconnected in a specific configuration.

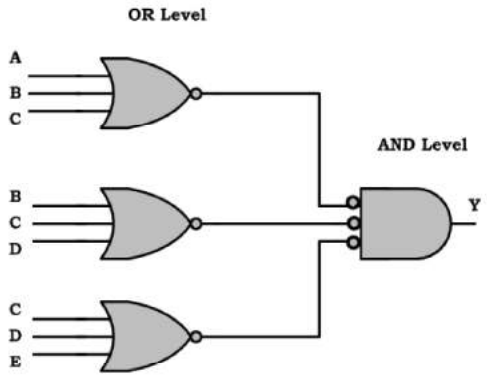
The expression Y is the result of the multiplication of three terms, namely Y=(A+B+C).(B+C+D).(C+D+E)

Step 1:



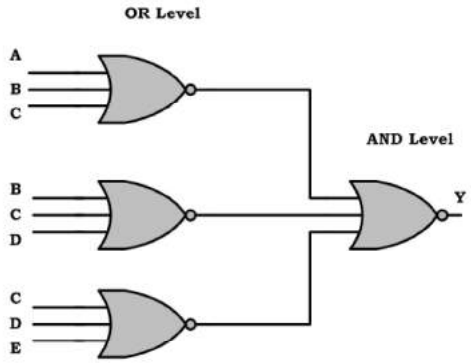
Step 2:

Bubbles are injected at the output of the OR gate and at the input of the AND gate, respectively.



Step 3:

The utilization of a NOR gate as a replacement for a bubbled input AND gate is observed. Currently, all of the gates have been replaced with NOR gates.

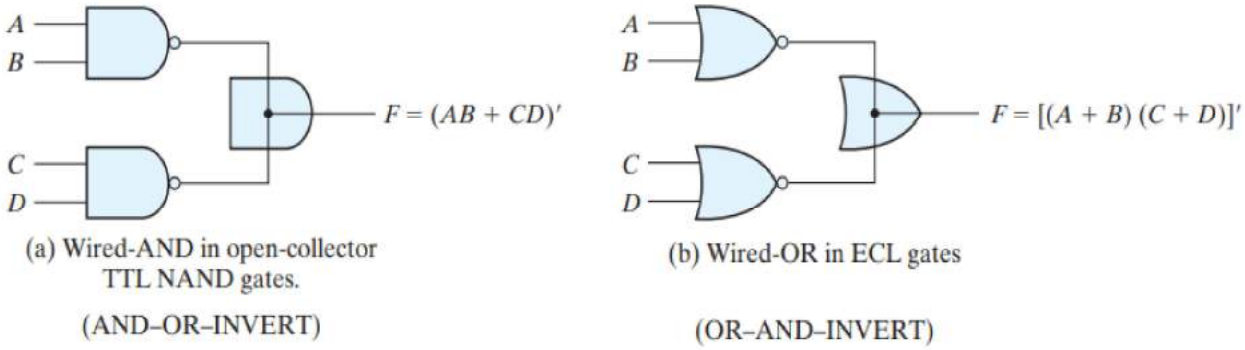


2.5 Other Two Level Implementations:

The predominant gate types encountered in integrated circuits are NAND and NOR gates. From a practical standpoint, the implementations of NAND and NOR logic hold significant importance. Certain NAND or NOR gates have the potential to establish a wire connection between their outputs, hence enabling the realization of a certain logic function. This particular form of reasoning is commonly referred to as wired logic. An instance of wired-AND logic can be achieved by connecting open-collector TTL NAND gates together. The wired-AND logic operation, implemented using two NAND gates, is illustrated in Figure 1. (a) The AND gate is visually represented by lines passing through its center, serving as a distinguishing feature from other conventional gates. The wired-AND gate is not a tangible gate, but rather a symbolic representation used to denote the function achieved by the specified wiring connection. The logical function executed by the circuit depicted in Figure 1. The user's text is not sufficient to rewrite in an academic manner.

$$F = (AB)' \cdots (CD)' = (AB + CD)' = (A' + B')(C' + D')$$

The function in question is commonly referred to as an AND–OR–INVERT function.



(a) Wired-AND in open-collector TTL NAND gates.

(AND–OR–INVERT)

(b) Wired-OR in ECL gates

(OR–AND–INVERT)

Figure. (a)The utilization of wired logic involves the implementation of a wired-AND logic configuration utilizing two NAND gates. (b)The utilization of Wired-OR in emitter-coupled logic (ECL) gates.

In a same manner, the NOR outputs of ECL gates can be interconnected to execute a wired-OR operation. The logical function executed by the circuit depicted in Figure 1. The second option, denoted as (b), is

$$F = (A + B)' + (C + D)' = [(A + B)(C + D)]'$$

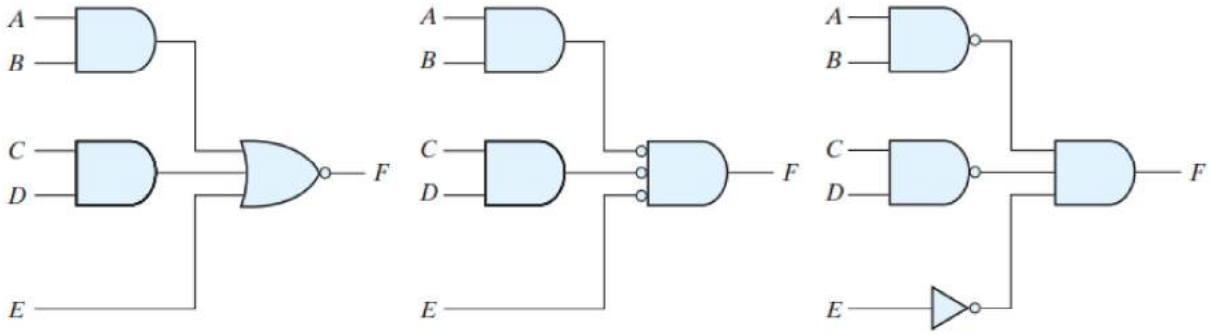
The function is commonly referred to as an OR–AND–INVERT function. A wired-logic gate does not manifest a tangible second-level gate as it solely constitutes a wired connection. The initial tier comprises NAND (or NOR) gates, while the subsequent tier features a solitary AND (or OR) gate.

AND–OR–INVERT Implementation:

The two forms, NAND-AND and AND-NOR, exhibit equivalence and can be considered collectively. Both of these components are responsible for performing the AND–OR–INVERT operation. The AND-NOR form exhibits similarities to the AND-OR form, although with an inversion achieved through the utilization of a bubble in the output of the NOR gate. The aforementioned expression is the implementation of the function F, where F is the complement of the logical OR operation performed on the logical AND operations of AB, CD, and E.

The diagram presented in Figure 1 is derived by employing the alternate graphic symbol for the NOR gate. I would like to request a revision of the user's text to adhere to academic standards. It should be noted that the variable E, in this context, is not complemented. This is due to the fact that the only alteration performed is in the graphical

representation of the NOR gate. The bubble is relocated from the input terminal of the second-level gate to the output terminals of the first-level gates. In order to mitigate the effects of the bubble, an inverter is required for the single variable. In an alternative approach, the removal of the inverter can be achieved by complementing the input E. The circuit depicted in Figure (c) follows a NAND-AND structure and was previously illustrated in the aforementioned figure to execute the AND-OR-INVERT operation. In order to create an AND-OR logic circuit, it is necessary to have an expression that is in the sum-of-products form. The implementation of the AND-OR-INVERT logic is largely similar, with the exception of the inversion component. Hence, if the complement of the function is condensed into sum-of-products form through the consolidation of the 0's in the map, it becomes feasible to execute F' using the AND-OR component of the function. When the input F travels through the output inversion, which is always present in the function, it will produce the output F.



(a) AND–NOR (b) AND–NOR (c) NAND–AND

2.6 Exclusive–OR function

The exclusive-OR function, often known as XOR, is a logical operation that outputs true only when the number of true inputs is odd.

The exclusive-OR (XOR), represented by the symbol $\oplus$, is a logical operation that executes the subsequent Boolean operation:

$$x \oplus y = xy' + x'y$$

The exclusive-OR (XOR) operation yields a result of 1 when either x or y is equal to 1, but not when both are equal to 1 or when both are equal to 0. The exclusiveNOR, which is sometimes referred to as equivalency, executes the subsequent Boolean operation:

$$(x \oplus y)' = xy + x'y'$$

The exclusive-NOR operation yields a result of 1 when both x and y have the same value, either 1 or 0. The complementarity between the exclusive-NOR and the exclusive-OR can be demonstrated by the utilization of a truth table or through algebraic manipulation.

$$(x \oplus y)' = (xy' + x'y)' = (x' + y)(x + y') = xy + x'y'$$

The subsequent IDs are applicable to the exclusive-OR operation:

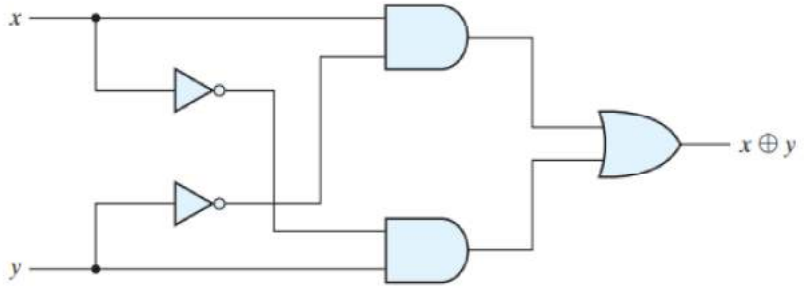
$$\begin{aligned} x \oplus 0 &= x \\ x \oplus 1 &= x' \\ x \oplus x &= 0 \\ x \oplus x' &= 1 \\ x \oplus y' &= x' \oplus y = (x \oplus y)' \end{aligned}$$

Any of these identities can be demonstrated through the use of a truth table or by substituting the { operator with its corresponding Boolean expression. Furthermore, it can be demonstrated that the exclusive-OR operation exhibits both commutativity and associativity properties. In other words,

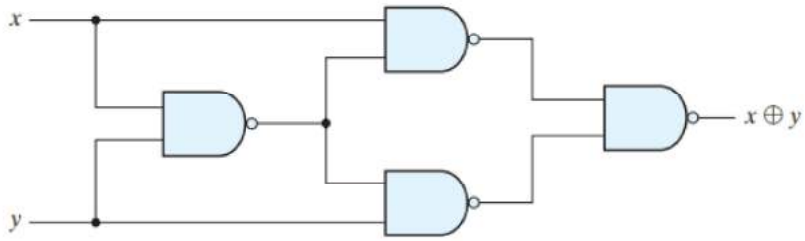
$$A \oplus B = B \oplus A$$

and

$$(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$$



(a) Exclusive-OR with AND–OR–NOT gates



(b) Exclusive-OR with NAND gates

Fig. Exclusive-OR implementations