Continue

In many digital circuits and practical problems, we need to find expressions with minimum variables. We can minimize Boolean expressions of 3, 4 variables very easily using K-map without using any Boolean algebra theorems. It is a tool which is used in digital logic to simplify boolean expression. It helps to simplify logic into simpler form by organizing grid from truth table values. This helps it to create a minimal Boolean expressions by identifying patterns.K-map can take two forms:Sum of product (SOP) Product of Sum (POS) According to the need of problem. K-map is a table-like representation, but it gives more information than the TABLE. We fill a grid of the K-map with 0's and 1's then solve it by making groups.Steps to Solve Expression using K-mapSelect the K-map according to the number of variables.Identify minterms or maxterms as given in the problem.For SOP put 1's in blocks of K-map respective to the minterms (0's elsewhere).For POS put 0's in blocks of K-map respective to the max terms (1's elsewhere).Make rectangular groups containing total terms in power of two like 2,4,8 ..(except 1) and try to cover as many elements as you can in one group.From the groups made in step 5 find the product terms and sum them up for SOP form.SOP FORM(Sum of Product Form)SOP form is way to simplify and write Boolean expressions using AND to combine inputs and OP to combine the results.1. K-map for 2 variablesIn the 2 variable k-map, four squares are constructed. Each square contains one term of expression with two variables.K-Map for 2 variables 2. K-map of 3 variablesK-map SOP form for 3 variablesZ= ΣA,B,C(1,3,6,7) From red group we get product term: A'C From green group we get product term: AB Summing these product terms we get- Final expression (A'C+AB) 3. K-map for 4 variablesK-map 4 variable SOP formF(A,B,C,D)=Σ(0,2,5,7,8,10,13,15) k map 4 variablesFrom red group we get product term: ABFrom green group we get product term: A'B'Summing these product terms we get- Final expression (AB+A'B').POS FORM (Product of Sum Form) POS form is way to simplify and write Boolean expressions using OR to combine terms inside parentheses and then AND to combine those groups.1.K-map of 2 variablesIn the 2 variable k-map, four squares are constructed. Each square contains one term of expression with two variables.K-map of 2 variables2. K-map of 3 variablesK-map 3 variable POS formF(A,B,C)=Σ(0,3,6,7) From red group we find terms A B Taking complement of these two A' B' Now sum up them (A' + B') From brown group we find terms B C Taking complement of these two terms B' C' Now sum up them (B'+C') From yellow group we find terms A' B' C' Taking complement of these two A B C Now sum up them (A + B + C) We will take product of these three terms : Final expression -(A' + B') (B' + C') (A + B + C) 3. K-map of 4 variables K-map 4 variable POS formF(A,B,C,D)=Σ(3,5,7,8,10,11,12,13) From green group we find terms C' D B Taking their complement and summing them (C+D'+B') From red group we find terms C D A' Taking their complement and summing them (C'+D'+A) From blue group we find terms A C' D' Taking their complement and summing them (A'+C+D) From brown group we find terms A B' C Taking their complement and summing them (A'+B+C') Finally we express these as product -(C+D'+B').(C'+D'+A).(A'+C+D).(A'+B+C') Advantages of K-MAPMakes Logic Simpler: It makes complicated Boolean expressions simpler.Minimizes Logic Gates: Simplifying the logic helps us to use fewer logic gates, making circuits more efficient.Reduce Errors: The visual representation of K-map helps to avoid errors which may happen while simplifying the logic.Time-Saving: It's quicker than traditional methods for simplifying logic.No need for Boolean Laws: K-map doesn't require deep knowledge of Boolean laws, making it easy for beginners.Disadvantages of K-MAPLimited to Fewer Variables: K-maps are best suited for 2 to 4 variables and above it, process becomes hard and complicated to manage. Not suitable for all functions: In some cases, its hard to group terms correctly, leading to errors and making simplification difficult.Space Limitations: As the number of variables increases, the K-map grid becomes too large to handle easily.Requires Careful Grouping: Sometimes incorrect grouping of terms can cause mistakes in logic simplification. Also attempt Quiz on K-MAP . Boolean expressions evaluate to either true or false. In their simplest implementation, precedence constraints use Booleans expressions as gatekeepers to determine whether or not an operation should occur. Within Data Flow operations, Boolean expressions are typically employed in the Conditional Split Transformation to determine whether a row in a Data Flow should be directed to another output. SQL Server Boolean Expression Learn how to use SSIS, from beginner basics to advanced techniques, with online video tutorials taught by industry experts. Enroll for Free "SSIS Training" Demo! For example, a Boolean expression to determine whether a Control Flow step would run only on Friday would require code to parse the day of the week from the current date and compare it to the sixth day, as shown here: DATEPART("dw", GETDATE()) == 6 This is a useful Boolean expression for end of the week activities. To control tasks that run on the first day of the month, use an expression like this: DATEPART ("dd", GETDATE()) == 1 This expression validates as true only when the first day of the month occurs. Boolean expressions don't have to be singular. Compound expressions can be built to test a variety of conditions. Here is an example in which three conditions must all evaluate to true in order for the expression to return a true value: BatchAmount == DepositAmount && @Not_Previously_Deposited == True && BatchAmount > 0.00 The @Not_Previously_Deposited argument in this expression is a variable; the other arguments represent columns in a Data Flow. Of course, an expression can just as easily evaluate alternate conditions, like this: (BatchAmount > 0.00 || BatchAmount < 0.00) && @Not_Previously_Deposited == True In this case, the BatchAmount must not be equal to 0.00. An alternative way to express the same thing is to use the inequality operator: BatchAmount != 0.00 && @Not_Previously_Deposited == True Don't be tripped up by these simple examples. They were defined for packages in which the data had known column data types, so there was no need to take extra precautions with casting conversions. If you are dealing with data from less reliable data sources, however, or you know that two columns have different data types, then take casting precautions with your expression formulas, such as in this expression: (DT_CY)BatchAmount == (DT_CY)DepositAmount && @Not_Previously_Deposited == True && (DT_CY)BatchAmount > (DT_CY)0.00 The Boolean expression examples here are generally the style of expression that are used to enable dynamic SSIS package operations. We have not covered the conditional, date/time, and string-based Boolean expressions, which are in the following sections. String expression development requires a little more information about how to handle a NULL or missing value, which is covered next. You can see some examples of these Boolean expressions put to work at the end of this Topic. Frequently Asked SSIS Interview Questions & Answers List of Related Microsoft Certification Courses: What is Karnaugh Map (K-Map)? Karnaugh map or K-map is a map of a function used in a technique used for minimization or simplification of a Boolean expression. It results in less number of logic gates and inputs to be used during the fabrication. Booleans expression can be simplified using Boolean algebraic theorems but there are no specific rules to make the most simplified expression. However, K-map can easily minimize the terms of a Boolean function. Unlike an algebraic method, K-map is a pictorial method and it does not need any Boolean algebraic theorems. K-map is basically a diagram made up of squares. Each of these squares represents a min-term of the variables. If n = number of variables then the number of squares in its K-map will be 2n. K-map is made using the truth table. In fact, it is a special form of the truth table that is folded upon itself like a sphere. Every two adjacent squares of the K-map have a difference of 1-bit including the corners. Karnaugh map can produce Sum of product (SOP) or product of Sum (POS) expression considering which of the two (0,1) outputs are being grouped in it. The grouping of 0's result in Product of Sum expression & the grouping of 1's result in Sum of Product expression. The expression produced by K-map may be the most simplified expression but not unique. There can be more than 1 simplified expression for a single function but they all perform the same. Grey Code In Gray code, every two consecutive number has a difference of 1-bit. As the squares in K-map also differs from its adjacent square by 1-bit which is why the variables in K-map are written in grey code. The gray code ensures that each cell of K-map is 1-bit difference with each other. You may also read: Counter and Types of Electronic Counters BCD to Gray Code Using K-Map The table for BCD to Gray code is given below. Rules of Minimization in K-Map While grouping, you can make groups of 2n number where n=0, 1, 2, 3... You can either make groups of 1's or 0's but not both. Grouping of 1's lead to Sum of Product form and Grouping of 0's lead to Product of Sum form. While grouping, the groups of 1's should not contain any 0 and the group of 0's should not contain any 1. The function output for 0's grouping should be complemented as F'. Groups can be made vertically and horizontally but not diagonally. Groups should be as large as possible even if they overlap. All the like term should be in a group even if they overlap. Uppermost & lowermost squares can be made into a group together as they are adjacent (1-bit difference). Same goes for the corner squares. Each group represents a term in the Boolean expression. Larger the group, smaller and simple the term. Don't care cells "x" should also be included while grouping to make a larger possible group. Karnaugh map of 2 to 4 variables is very easy. However, 5 and 6 variable K-map is a little bit complex. We will discuss one by one in details. You may also read: Digital Flip-Flops – SR, D, JK and T Flip Flops 2 variables have 2n = 22 = 4 minterms. Therefore there are 4 cells (squares) in 2 variable K-map for each minterm. Consider variable A & B as two variables. The rows of the columns will be represented by variable B. The square facing the combination of the variable represents that min term as shown in fig below. Grouping in 2 variables K-map is easy as there are few squares. Example of 2 Variable K-Map Function F (A, B) F = ∑ (m0, m1, m2) = ĀB̄ +ĀB +AB̄ K-map for 2 variables. We made 2 groups of 1's, each group contains 2 minterms. In the first group, variable A is changing & B remains unchanged. So the first term of the output expression will be B̄ (because B = 0 in this group). In the 2nd group, Variable B is changing and variable A remains unchanged. So the second term will be A̅ (because A=0 in this group). Now the simplified expression will be the sum of these two terms as given below, F = Ā + B̄ Compare this expression with the original expression of the function, this expression only uses one gate during its implementation. You may also read: Ripple Carry And Carry Look Ahead Adder 3 Variable K-Map 3 variables make 2n=23=8 min terms, so the Karnaugh map of 3 variables will have 8 squares(cells) as shown in the figure given below. 3 variable K-map can be in both forms. Note the combination of two variables in either form is written in Gray code. So the min terms will not be in a decimal order. The uppermost & lowermost cells are adjacent in the first form of K-map, the leftmost and rightmost cells are also adjacent in the second form of K-map. So they can be made into groups. Some examples of grouping: You can make groups of 2, 4 & 8 cells having same 1s or 0s. Notice the groups of the uppermost & lowermost cells. They are adjacent as there is only one-bit difference. That is why they can be grouped together. Don't make unnecessary groups. All 1s or 0s should be grouped, but not all possible groups of 1s or 0s should be made. Example of 3 Variable K-Map F (A,B,C,E) = ∑ (m0, m2, m5, m7, m8, m10, m14, m6) This example shows that you can make the groups overlap each other to make them as large as possible and cover all the 1s. In this first group (m0, m2, m6, m4), A &B are changing so we will eliminate it. However, C remains unchanged in this group. So the term this group makes is C̄ (because C=0 in this group). In the 2nd group (m0,m1,m4,m5), A and C are changing so it will be eliminated from the term. B remains unchanged in this group. So the term this group produce will be B̄ (because B=0 in this group). The sum of these two terms will make the simplified expression of the function as given below. F = B̄ + C̄ Another example of grouping 2 is given below. It shows how the corner min terms are grouped. In the first group (m0,m4), A is changing. B & C remains unchanged. So the term this group will produce will be B̄C̄ (B=0,C=0 in this group). 2nd group (m3,m7), A is changing. B & C remains unchanged. BC will be the term because B=1,C=1 in this group. So This K-map leads to the expression F = B̄C̄ + BC Some examples shows that a group of 4 cells give a term of 1 literal and a group of 2 cells gives a term of 2 literals and a group of 1 cell gives a term of 3 literals. So the larger the group,the smaller and simple the term gets. 4-variable K-Map 4 variables have 2n=24=16 minterms. So a 4-variable k-map will have 16 cells as shown in the figure given below. Each cell (min term) represent the variables in front of the corresponding row & column. The variables are in gray code (1-bit change). The four cells of the corner are adjacent to each other as there is a 1-bit difference even if they are not touching physically. So they can be grouped together. Some example of grouping in 4-variable k-map is given below. As you can see in the example above the 4 corner cells are grouped. In the second example, leftmost columns can be grouped with rightmost column and uppermost row with the lowermost row. These groups should be as large as possible with the size of 1,2,4,8 or 16 cells. The terms of the expression depend on these groups. For example, this group contains: One square, then it will give a term of 4 literals Two squares, then it will give a term of 3 literals Four squares, then it will give a term of 2 literals Eight square, then it will give a term of 1 literal Sixteen square which will cover the whole 4-variable k-map which means constant 1 output. Example of 4 Variable K-Map F(A,B,C,D) = ∑(m0, m1, m2, m4, m5, m6, m8, m9, m12, m13, m14) First of all, try to make the biggest possible groups as shown in this example. Corner 1s can also be made into a group of 4. The remaining last 1 should be combined with the pre-made group to make a big overlapping group. Group of 8 will give a term of 1 literal that remains unchanged i.e. C Corner group of 4 will give term with 2 literals that remain unchanged i.e. BD The last group of 4 will give AC because they remained unchanged in the group. So the expression will be F = C̄ + B̄D̄ + ĀC̄ 5 & 6 Variable Karnaugh Maps K-Map is used for minimization or simplification of a Boolean expression. 2-4 variable K-maps are easy to handle. However, the real challenge is 5 and 6 variable K-maps. Visualization of 5 & 6 variable K-map is a bit difficult. When the number of variables increases, the number of the square (cells) increases. And drawing the K-map becomes a bit complicated because of drawing the adjacent cells. 5 Variables K-Map 5 variables have 32 min terms,which mean 5 variable karnaugh map has 32 squares (cells). A 5-variable K-map is made using two 4-variable K-maps. Consider 5 variables A,B,C,D,E. their 5 variable K-map is given below. These both 4-variable Karnaugh map together represents a 5-variable K-map for 5 variable A,B,C,D,E. Notice variable A over the top of each 4-variable K-map. For A=0, the left K-map is selected and right map for A = 1. Each corresponding squares (cells) of these 2 4-variable K-maps are adjacent. Visualize these both K-maps on top of each other. m0 is adjacent to m16, so is m1 to m17 so on until the last square. The rule (method) of grouping is same for each of the 4-variable k-maps. However, you also need to check the corresponding cells in both K-maps as well. A few example of grouping is differentiated using different colors. Example of 5 Variables K-Map F (A,B,C,D,E) = ∑ (m0, m2, m5, m7, m8, m10, m16, m21, m23, m24, m27, m31) This is the 5-variable k-map for the function given above. There are four groups made in this K-map. Each group has been given a different color to differentiate between them. The red color group is a group of 4 min terms made between both 4-variable k-maps because they are adjacent cells and it overlaps the green group. The yellow group is also a group of 4 min terms made between adjacent cells of the 4-variable k-maps. The green group is a group of 4 min terms made in the left 4-variable K-map. The blue group is of 2 min-terms made in the right 4-variable k-map because there are no common adjacent cells in the other k-map. Green color group of 4 min term will produce the term ĀCE. The individual 4-variable k-map will produce C̄E as they are not changing in the group but variable A should also be taken into account because this individual 4-variable k-map is being represented by A. The red color group will produce C̄D̄E. This group is made between both k-maps which means variable A changes and in individual K-map, B changes so these both variables will be eliminated from the term. Only C̄D̄E remains unchanged in this group. The yellow group produce BCE because these literals are not changing in this group. Blue group of 2 min terms will produce the term ABD̄E as it remains unchanged in the group. The simplified expression will be the sum of these 4 terms, which is given below: F = ĀCE + C̄D̄E + BCE + ABD̄E You may also read: Digital Synchronous Counter – Types, Working & Applications 6-Variable Karnaugh Map 6-variable k-map is a complex map which can be drawn. Visualizing 6-variable k-map is a little bit tricky. 6 variables make 64 min terms, this means that the k-map of 6 variables will have 64 cells. Its geometry becomes difficult to draw as these cells are adjacent to each other in all direction in 3-dimensions i.e. a cell is adjacent to upper, lower, left, right, front and back cells at the same time. we will draw it like 5-variable k-map as shown in the figure below. The 6-variable k-map is made from 4 4-variable k-maps. As you can see variable A on the left side select 2 k-maps row-wise between these 4 k-maps. A = 0 for the upper two K-maps and A = 1 for the lower two K-maps. Variable B on top of these K-maps select 2 k-maps column-wise. B = 0 for left 2 K-maps and B = 1 or for right 2 K-maps. Imagine these 4-variable K-maps as a single square, these k-maps are adjacent to each other horizontally and vertically but not diagonally because these cells have 1-bit difference. The groups between these k-maps should be made as done in 5-variable K-map but you cannot make groups between diagonal k-maps. Some examples of grouping in 6-variable k-map are given below. Group of 16 min-terms between 4 k-maps as they are all adjacent. Visualize these k-maps on top of each other. In this example, there are 5 groups of 4 min-terms. Notice the min-terms in the diagonal K-maps. They make a separate group because these K-maps are not adjacent. Example of 6 Variable K-Map maping F = ∑ (m0, m2, m8, m9, m10, m12, m13, m16, m18, m24, m25, m26, m29, m31, m32, m34, m35, m39, m40, m42, m43, m47, m48, m50, m56, m58, m61, m63) Its 6-variable k-map is given below: There are 5 groups in this K-map each colored different. Green group is made of 16 min-terms between all 4 individual K-maps. In this group, AB keeps changing so they will be eliminated from the term. C & E are also changing so they will be eliminated from the term too. So the term will become D̄F because they remain unchanged throughout the group. Red group is made of 4 min-terms. In this group, B is changing so it will be eliminated. D is also changing. So the only remaining unchanged literals will make the term which is ĀCĒF. Blue group is also made of 4 min-terms. The only changing variables are DF throughout this group so they will be eliminated from the term too. Yellow group is also a group of 4 min-terms and the changing variables in this group are AE. The literal that remains unchanged is BCDF in this group. The black group is of 1 min-term only. This group produces the term ABEF. The simplified expression of the function will be the sum of these 5 terms from these groups. The expression is given below: F = D̄F + ĀCĒF + ĀBCĒ + BCDF + ĀBEF You may also read: Share — copy and redistribute the material in any medium or format for any purpose, even commercially. Adapt — remix, transform, and build upon the material for any purpose, even commercially. The licensor cannot revoke these freedoms as long as you follow the license terms. Attribution — You must give appropriate credit , provide a link to the license, and indicate if changes were made . You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original. No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits. You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation . No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material. LearnPracticeImproveSucceedRegistered Students7.1+ CroreStudent Selections4+ Lacs Tests Attempted242+ CroreClasses Attended5.5+ CroreGet unlimited access to the most relevant Mock Tests, on India's Structured Online Test series platformWhat you get with Testbook PassExplore Testbook Pass RRB NTPC Mock Test Series 2024-25 (CBT 1 + CBT 2) 1792 Total Tests | 16 Free Tests English,Hindi + 6 MoreAvailable in: English, Hindi, Bengali, Marathi, Telugu, Tamil, Gujarati, Kannada, 1 Ultimate Live Test191 Most Saved Qs CT78 +1522 more tests SSC CGL (Tier I & Tier II) Mock Test 2025 (New) 1088 Total Tests | 3 Free Tests1 Ultimate Live Test120 2024 PYQ233 Chapter Test (Tier I)+734 more tests RPF Constable Mock Test Series 2024 1197 Total Tests | 9 Free Tests English,Hindi + 6 MoreAvailable in: English, Hindi, Bengali, Marathi, Tamil, Telugu, Gujarati, Oria, 91 ALP & SI 2024 PYP Qs3 ALP & SI 2024 PYP Repeated Qs5 Railway Latest Pattern FT+1098 more tests RPF SI Mock Test Series 2024 1318 Total Tests | 3 Free Tests English,Hindi + 7 MoreAvailable in: English, Hindi, Bengali, Marathi, Telugu, Tamil, Gujarati, Kannada, Oria, 41 Current Affairs Special3 Unexpected/Bouncer FT26 General Science Special+1248 more testsExplore all Test Series With 1.8+ Crore Students and One of the best Selection rate in India amongst online learning platforms, you can surely rely on us to excel.Get Started For FreeLearn from the masters of the subject, in the most engaging yet simplest waysGet a detailed breakdown of your strengths & weaknesses and discover insights to improve your scoreFeel the thrill of a real exam. Improve your time & pressure management skillsLearn in the language you are most comfortable with. Choose from any of our 8 languagesAIR 1|Delhi Judicial 2024Start your learning journey now!LearnPracticeImproveSucceed The article explains the Karnaugh Map (K-Map), a graphical method for simplifying Boolean expressions in digital logic design. It discusses the K-Map's structure, its application for 2-, 3-, and 4-variable problems, techniques for grouping adjacent cells, and utilizing "don't care" entries for further simplification of logic functions. An alternative to the truth table to determine and simplify the logic function for an application is Karnaugh map (K-Map), named after its originator Karnaugh. Karnaugh map abbreviates to K-map offers a simpler solution to find the logic function for applications with two, three, and four inputs. It's applicable up to a maximum of six inputs. For more inputs, other techniques are used. What is a Karnaugh Map (K-Map)? A Karnaugh Map (K-map) is a graphical tool used in digital logic design and simplification of Boolean algebra expressions. It provides a systematic method to simplify logical expressions, reducing the number of terms and operations required to implement a logic circuit. The K-map organizes the truth table into a grid-like structure, grouping together combinations of input variables that differ by only one bit, which helps in identifying common factors. Key Features of a Karnaugh Map (K-Map) Structure __ The K-map is a grid-like representation of truth tables. For an n-variable Boolean function, the K-map consists of 2n cells, each representing a unique minterm (a combination of input values where the output is 1). Arrangement __ The cells are arranged in a way that adjacent cells differ by only one bit (Gray code sequence). This adjacency facilitates combining terms with minimal differences. Simplification __ By grouping adjacent cells with the same value (usually 1), the K-map identifies terms that can be combined. This helps in reducing the Boolean expression. Adjacency __ Cells on the edges of the map are considered adjacent to corresponding cells on the opposite edges (wrapping around), ensuring all possible simplifications are considered. 2-Variable K-Map Example Applications with only two inputs A and B are easy to handle by any method. For easier understanding of how a K-map works, nevertheless, we start with two inputs. The map for this problem, the K-map has four cells. The horizontal cells represent the two states of one of the inputs (B in Figure 1, starting from $\overline{B}$ on the left), and the vertical cells exhibit the states of the other input starting from the NOT value. The correspondence between the truth table and the K-map cells are shown by arrows for two cells. The required values for the output are entered in the corresponding cells (Figure 1b). For example, in the figure shown, it is required that the output be 1 in the two cases: $\overline{B}$ and AB; that is when both inputs have their complement values, and when both inputs have their true values. (Note that for simplicity the multiplication symbol is not shown; thus, AB denotes A.B.) The arrows show the corresponding transfers. The other cells are required to be 0's in this case. Thus, they are filled with 0's. Each cell represents one of the four combinations of the two inputs. For example, the top left cell represents $\overline{A}\overline{B}$. This is the logic expression for that cell. When the cell values in the K-map are identified based on the output requirement and the 1's and 0's are assigned, the logic expression can be derived from the cells containing 1. The sum of the expressions for those cells with 1 inside defines the logic function for the application. In the present two-input example the expression for the output Z is $Z=\overline{A}.\overline{B}+A.B$ Figure 1 Truth table and K-map for a two-variable example. (a) Structure of the K-map. (b) Example of implementation of requirements. This is the expression for the complement of an XOR gate. So, for this application, the required circuit consists of an XOR gate and a NOT gate. For this problem, it was not necessary to use a K-map because it is a trivial case of two inputs. But, for three and four inputs the problem is more involved. One important property of a K-map is that moving from any cell to a neighboring cell implies only one change (and never more than one). One change means that only one of the inputs switches to its complement. You can see that for two-input K-map moving horizontally changes B, only, and moving vertically changes A, only. This property helps to select the correct entry for extension of the two-input K-map to three and four inputs, as can be seen next. Also, take note that in K-map notation all cells have two neighbors vertically and two neighbors horizontally. In other words, the cells are assumed to be seamlessly organized in circles and not in rows and columns. This matter is more important and clearer for three and four inputs and will be seen again. 3-Variable K-Map Example The K-map for two inputs can be extended to three inputs by combining the third input either in the horizontal or vertical direction with the input already placed there. Here we do that horizontally, and the third variable C is combined with B, as it is shown in Figure 2. The K-map for three variables has eight cells, each one of which represents one of the possible eight combinations of three inputs. In Figure 2 the cells are numbered from 1 to 8 for reference. Figure 2 K-map for a three-variable example Note the following important points: 1. Each cell represents the combination of three inputs. Thus, the expression of the function for each cell is the product of inputs or their complements like $\overline{A}B\overline{C}$. 2. Variation of the two inputs is reflected in the horizontal direction. 3. In the horizontal direction moving from each cell to its right or left neighbor implies a change only either in B or in C but not both. This is why in the third column from left BC appears and not $B\overline{C}$. On the basis of the requirement for output, as shown by the asterisks in the truth table, 1's are entered in the corresponding cells in the K-map. The rest of the cells are filled with 0's. The essence of a K-map now can be shown based on the pattern that appears after all cells are filled. Both cells 1 and 5 (in Figure 2) must be selected as part of the output. Cell 1 contains $\overline{A}$, and cell 5 contains A. This implies that changing A to $\overline{A}$ (and vice versa) does not have any effect on the output, meaning that no matter what input A is the output is independent of it. In such a case the output is defined by the common part of these two cells, plus any other cell that contains 1. For this case, $Z=\overline{B}.\overline{C}+\overline{A}.\overline{B}.C$ The process of definition and simplification of the logic function by using a K-map is as follows. 1. Encircle all occurrences of 1's in the cells putting together all those in the neighboring cells in groups of two and four. The neighboring cells can occur in a row, column or rectangle when more than one column (or row) is involved. 2. Write the corresponding expression for each encircled group. 3. Add together the logic expressions for all groups. For step 1, one should pay attention to the fact that moving from cell 4 to cell 1 (or vice versa) and from cell 8 to cell 5 implies only one change; it is a switch between B and its complement. In this respect, cells 1 and 4 are considered neighbors, and cells 5 and 8 are also considered neighbors. Thus, they must be encircled together if they contain 1's. For step 2, note that a row change implies a switch between A and its complement, a change between the first and fourth column implies a switch between B and its complement, and a change between columns 1 and 2 and between columns 3 and 4 implies a switch between C and its complement (see Figure 3). We try to form groups of two or four, whenever possible. For each case, the variable(s) that do not change within the group describe the logic function for that group. Forming groups is the process of simplification of the logic function for an application. Examples of forming groups and encircling neighboring cells are shown in Figure 3. The corresponding logic function for the cases shown is described below. More examples are shown in the next section for four-input problems. Figure 3 Examples of encircling neighboring cells containing 1's in a three-input scenario. (a) Only one group. (b) No grouping is possible. (c) One vertical and one horizontal group. (d) One group of four cells in two rows. (e) One group of four cells in a row. (f) Extreme cells in a row make a group. (g) In this case, there are one pair plus one cell, or two pairs with a shared cell. (h) Two groups, one horizontal and one vertical. (i) Three groups. (j) Two groups. (k) One group of four cells. The logic functions for cases a to k illustrated in Figure 3 are as follows: a. The encircled cells belong to $\overline{A}\overline{B}C$ and $\overline{A}BC$. That is, the output is defined by the sum of $\overline{A}\overline{B}C$ and $\overline{A}BC$. Because a change of $\overline{B}$ to B has no effect on the output (which is shown by having 1 in both cells), then they are removed and the output is defined by $Z=\overline{A}C$ b. The individual encircled cells do not have any common element. The output is, thus, the sum of these two cells. \[Z=\overline{A}\overline{B}C+A\overline{B}C\] c. Although a cell can be shared between the two encircled areas, it is better to divide the cells containing 1's to two groups each one containing a pair of cells. $Z=\overline{B}\overline{C}+\overline{A}C$ d. All the encircled cells have $\overline{B}$ in common and change in A and C do not have any effect. The output is only defined by $\overline{B}$. $Z=\overline{B}$ e. Similar to case d, all encircled cells contain A, and changes for B and C have no effect on the output. $Z=A$ f. There are two encircled pairs in this case; one corresponds to no change in $\overline{C}$ (it exists in both cells) and the other one implies no change in C. \[Z=\overline{A}\overline{C}+AC\] g. When three cells are involved, as encircled, we can consider two pairs with one shared cell. For the first pair,$\overline{B}$ does not change, and for the second pair, C does not change. $Z=\overline{A}\overline{B}+\overline{A}C$ h. For this case, again, we consider a horizontal pair and a vertical pair (one cell is shared between the two). For the horizontal pair, B changes (thus, its function excludes B) and for the vertical pair A changes (so, A is omitted). $Z=\overline{A}C+\overline{B}C$ i. This case can be considered a combination of that in case h plus an additional term for the cells 5 and 8. Thus, $Z=\overline{A}C+\overline{B}C+A\overline{C}$ j. The cells 1 and 4 lead to $\overline{A}\overline{C}$, and the cells 1 and 5 lead to $\overline{B}\overline{C}$; therefore, $Z=\overline{A}\overline{C}+\overline{B}\overline{C}$ k. All the encircled cells contain $\overline{C}$, and changes in A and B do not play a role. $Z=\overline{C}$ 4-Variable K-Map Example The K-map for four variables has 16 cells; that is, twice as many as that for three variables. As was done for the horizontal extension from the two- input case, both horizontal and vertical extensions are implemented. The result is shown in Figure 4. Each cell in this case reflects the value corresponding to one of 16 possible states for combination of four inputs. The logic function of each cell is composed by multiplying (AND'ing) the logic expression of the row and the column in which it lies. For example, ABCD corresponds to the third row (AB) and the third column (CD). Simplification is done in the same manner that was described for the three-input problem. We try to find groups of two, four, and eight neighbor cells that share the same variable(s). Those variables then define the logic function for the group. Figure 5 illustrates a number of examples, whose logic functions are as follows. a. The four encircled cells are independent of each other, and they cannot be grouped. The output, thus, is the sum of all four of them. \[Z=\overline{A}\overline{B}\overline{C}\overline{D}+\overline{A}BCD+\overline{A}B\overline{C}D+\overline{A}\overline{B}C\overline{D}\] Figure 4 K-map for a four-variable example. Figure 5 Examples of four-variable K-maps. b. There are three groups, part of column 1 and 4, the individual circle, and the two in the middle. \[Z=\overline{A}\overline{D}+ABD+\overline{A}\overline{B}\overline{C}D\] c. The function of a row or column is always that written in front of the row or on the top of the column. Thus, \[Z=\overline{A}\overline{B}+AB\] d. All the 1's in the four corners can be grouped together. The only two variables that do not change are $\overline{B}$ and $\overline{D}$. The individual cell is $\overline{A}\overline{B}\overline{C}D$. \[Z\text{ }=\overline{B}\overline{D}\text{ }+\overline{A}\overline{B}\overline{C}D\] e. A group of four and a pair can be identified here. \[Z=B\overline{C}+BCD\] f. Two independent groups exist. In the top group, change of B and D does not play any role; in the bottom one, again, change of B and D with no effect is involved. Therefore, $Z=\overline{A}\overline{C}+AC$ g. This case is similar to case c with the exception that the two groups share one cell, but that does not affect the result. \[Z=\overline{A}\overline{B}+\overline{C}\overline{D}\] h. The answer based on the three pairs of cells forming three groups is \[Z=\overline{B}\overline{C}\overline{D}+B\overline{C}D+\overline{B}CD\] Don't Care Entries The entries of a K-map are not always 1's and 0's. The application requirement determines those cells that must contain 1. Some of the cells also must contain 0 as a requirement, but, also, it can happen that for certain cells the content either is not defined or its value is immaterial. In such a case, it does not matter if a value of 1 or 0 is assigned to the cell. For a case like this an X is entered in the cell instead of 1 or 0. An example is shown in Figure 6a. These cells are called "don't care." Figure 6 Benefitting from do not care entries. (a) All X's are considered 0 and do not appear in the output. (b) In order to group with 1's, some X's are considered 1. One can benefit from the "don't care" entries in simplifying the logic function. This advantage is not available in the example in Figure 6a. On the contrary, in the example shown in Figure 6b, the logic function can be quite simplified if certain of these "X" cells are assumed to contain 1. One option, based on the encircled groups as shown and according to the definition of cells in Figure 4, is \[Z=\overline{B}D+BC\] Karnaugh Map (K-Map) Key Takeaways The Karnaugh Map (K-Map) plays a crucial role in simplifying Boolean expressions, which directly impacts the efficiency and reliability of digital logic design. By reducing complex logic functions into simpler forms, K-Maps help minimize the number of logic gates required in a circuit. This leads to lower power consumption, faster processing, and cost-effective implementation—factors that are critical in designing hardware for computers, embedded systems, industrial automation, and consumer electronics. Additionally, the ability to handle "don't care" conditions makes K-Maps highly adaptable in real-world scenarios, where flexibility in output logic is often needed. The K-Map method is a simple and efficient method for simplifying Boolean Expressions. In this lecture, we will learn to solve two and three variables Boolean functions using K-MAP. This method is also known as the Karnaugh Map. K-MAP is represented in the form of a Truth Table. In a truth table, a function of n variables will have 2^n min-terms. For example, there will be four min-terms for two variables, i.e.(x,y), and 8 min-terms for 3 variables. Note: Min-terms and Max-terms are forms of Literals. Two-Variable K-Map The 0 and 1 represent the row and column. X and Y are variables of a Boolean function. The following diagram shows the arrangement of two variables of the Boolean function in the form of K-MAP. According to Pic A, the Square in Row 0 and Column 0 is m0. Square in Row 0 and Column 1 is m1. M0, m1, m2... m (n) are the min-terms. According to Pic B, the Square in Row 0 and Column 0 is X'.Y' and so on. Example 01: Simplify the Boolean Function F(X, Y) = x.y+x'y Step 01: Represent the function in the form of K-MAP Represent the corresponding boxes of (x.y) and (x'.y) with 1 and the remaining with 0. Step 02: Mark any group of two or for adjacent 1 as shown below Step 03: write a simplified expression for each group X'.y+x.y = (x'+x).y = 1.y =y Step 04: write the final simplified form as a sum of products F(X, Y) = y Example 02: Simplify the Boolean Function F(X, Y) = x.y'+x'.y+x.y Step 01: Represent the function in the form of K-MAP Represent the corresponding boxes of (x.y'), (x'.y), and (x.y) with 1 and remaining with 0. Step 02: Mark any group of two or for adjacent 1 as shown below Step 03: write a simplified expression for each group Group 01: X'.y+x.y = (x'+x).y = 1.y =y Group 02: X.y'+x.y = (y'+y).x = 1.x =x Step 04: write the final simplified form as a sum of products F(X, Y) = x+y Example 03: Simplify the Boolean Function F(X, Y) = x'.y'+ x.y'+x'.y+x.y Step 01: Represent the function in the form of K-MAP Represent the corresponding boxes of (x', y'),(x.y'), (x'.y), and (x.y) with 1 and remaining with 0. Step 02: Mark any group of two or for adjacent 1 as shown below Step 03: write a simplified expression for each group All boxes are filled with one, so there will be only one group. Group 01: x'.y'+ x.y'+x'.y+x.y = y'(x+x')+ y(x+x')= y'.1+y.1= y'+y = 1 Step 04: write the final simplified form as a sum of products F(X, Y) = 1 Three-Variable K-Maps There are eight min-terms in a three-variable map. Therefore, the map consists of eight squares. Using the K-Map method, let's explain some examples of three-variables Boolean algebra simplifications. Example 01: Simplify the Boolean Function F(X, Y, Z) = x.y.z'+ x'.y'.z'+ x.y'.z'+x'.y.z Step 01: Represent the function in the form of K-MAP Represent the corresponding boxes of x.y.z', x'.y'.z', x.y'.z', and x'.y.z with 1 and remaining with 0. Step 02: Mark any group of two or for adjacent 1 as shown below Step 03: write a simplified expression for each group Group 01: x'.y'.z' + x.y'.z' = y'.z'(x+x') = y'.z'(1)= y'.z' Group 01: x.y'.z' + x.y.z' = x.z'(x+x') = x.z'(1) = x.z' Step 04: write the final simplified form as a sum of products F(X, Y) = y'.z'+ x.z' Privacy Policy, Terms and Conditions, DMCA Policy and Compliant, Contact, Help, about Copyright © 2018-2026 BrainKart.com; All Rights Reserved. Developed by Therithal info, Chennai. 17 Mar 2025 | 3 min readAs we know that K-map takes both SOP and POS forms. So, there are two possible solutions for K-map, i.e., minterm and maxterm solution. Let's start and learn about how we can find the minterm and maxterm solution of K-map.Minterm Solution of K MapThere are the following steps to find the minterm solution or K-map:Step 1:Firstly, we define the given expression in its canonical form.Step 2:Next, we create the K-map by entering 1 to each product-term into the K-map cell and fill the remaining cells with zeros.Step 3:Next, we form the groups by considering each one in the K-map.Notice that each group should have the largest number of 'ones'. A group cannot contain an empty cell or cell that contains 0.In a group, there is a total of 2n number of ones. Here, n=0, 1, 2, ...n.Example: 20=1, 21=2, 22=4, 23=8, or 24=16.We group the number of ones in the decreasing order. First, we have to try to make the group of eight, then for four, after that two and lastly for 1.In horizontally or vertically manner, the groups of ones are formed in shape of rectangle and square. We cannot perform the diagonal grouping in K-map.The elements in one group can also be used in different groups only when the size of the group is increased.The elements located at the edges of the table are considered to be adjacent. So, we can group these elements.We can consider the 'don't care condition' only when they aid in increasing the group-size. Otherwise, 'don't care' elements are discarded.Step 4:In the next step, we find the boolean expression for each group. By looking at the common variables in cell-labeling, we define the groups in terms of input variables. In the below example, there is a total of two groups, i.e., group 1 and group 2, with two and one number of 'ones'.In the first group, the ones are present in the row for which the value of A is 0. Thus, they contain the complement of variable A. Remaining two 'ones' are present in adjacent columns. In these columns, only B term in common is the product term corresponding to the group as A'B. Just like group 1, in group 2, the one's are present in a row for which the value of A is 1. So, the corresponding variables of this column are B'C'. The overall product term of this group is AB'C'.Step 5:Lastly, we find the boolean expression for the Output. To find the simplified boolean expression in the SOP form, we combine the product-terms of all individual groups. So the simplified expression of the above k-map is as follows:A'+AB'C'Let's take some examples of 2-variable, 3-variable, 4-variable, and 5-variable K-map examples.Example 1: Y=A'B' + A'B+ABSimplified expression: Y=A'+BExample 2: Y=A'B'C'+A' BC'+AB' C'+AB' C+ABC'+ABCSimplified expression: Y=A+C'Example 3: Y=A'B'C' D'+A' B' CD'+A' BCD'+A' BCD+AB' C' D'+ABCD'+ABCDSimplified expression: Y=BD+B'D'Maxterm Solution of K-MapTo find the simplified maxterm solution using K-map is the same as to find for the minterm solution. There are some minor changes in the maxterm solution, which are as follows:We will populate the K-map by entering the value of 0 to each sum-term into the K-map cell and fill the remaining cells with one's.We will make the groups of 'zeros' not for 'ones'.Now, we will define the boolean expressions for each group as sum-terms.At last, to find the simplified boolean expression in the POS form, we will combine the sum-terms of all individual groups.Let's take some example of 2-variable, 3-variable, 4-variable and 5-variable K-map examplesExample 1: Y=(A'+B')+(A'+B)+(A+B)Simplified expression: A'BExample 2: Y=(A + B + C') + (A + B' + C') + (A' + B' + C) + (A' + B' + C')Simplified expression: Y=(A + C') .(A' + B')Example 3: F(A,B,C,D)=π(3,5,7,8,10,11,12,13)Simplified expression: Y=(A + C') .(A' + B')Next TopicDe-Morgan's Theorem The website is currently getting the required resources.If it takes longer than 30 seconds then please refresh unless you have slow internet. MindMajix conducted Octopus Deploy training for our team successfully. Even though the initially assigned instructor not available before the training for health reasons, MindMajix assigned another trainer immediately from their pool of trainers. We see no issues interacting with MindMajix as they are proactive and very responsible. Bayswater, Victoria, AustraliaRating: 4.5 In the world of digital logic and circuit design, simplifying boolean expressions is a fundamental task. One of the most effective tools for this purpose is the Karnaugh Map, often referred to as a K-map. K-maps provide a graphical and systematic approach to minimize boolean expressions, making them easier to implement in digital circuits. This article explores the concept of boolean expressions, introduces the Karnaugh Map, and demonstrates how it is used to simplify complex logical statements. What is K Map? Karnaugh Map, or K Map, is a graphical representation of a Boolean function. It is used in digital electronics and computer science to simplify Boolean expressions and minimize the number of gates required to implement a logic circuit. The K Map is a two-dimensional grid of cells, each of which represents a possible combination of inputs to the function. The cells are either marked with a 0 or 1, indicating the value of the function for that particular input combination. K Maps are used to simplify Boolean expressions by grouping together adjacent cells that have the same value and reducing them to a single expression. The resulting expression is a minimal representation of the original function, and it can then be implemented using fewer gates, reducing the complexity and cost of the logic circuit. This simplification process is known as minimizing the Boolean function. K Maps can be used for functions with up to four inputs, although for functions with more inputs, the map becomes too complex to be useful. Before, learning how to solve a K Map, let us first look at some terms related to K Map. Terms Related to K Map Here are some terms related to Karnaugh Maps (K Maps) that are commonly used in digital electronics and computer science: Cell: A cell is defined as a single square on the K Map that represents a particular combination of inputs to the function. Group: A collection of adjacent cells that have the same value (either 0 or 1). Prime Implicant: A group of cells that cannot be reduced further without changing its value. Essential Prime Implicant: A prime implicant is an implicant that must be included in the minimized expression for the function. Don't-Care Term: A term in the minimized expression for the function that does not affect the output, and can therefore be ignored. Implicant: A term in the minimized expression for the function that is used to simplify the original Boolean expression. Quine-McCluskey Algorithm: A method for simplifying Boolean expressions, based on the K Map, that can be used to obtain the minimized expression for a function. These are some of the key terms used in K Maps and the simplification of Boolean expressions. Understanding these terms is essential for using K Maps effectively and for simplifying Boolean expressions in digital electronics and computer science. Types of K Map Karnaugh Maps (K Maps) can be classified based on the number of variables used to represent a Boolean function: 2-Variable K Map A 2-variable K Map is used to simplify Boolean functions that depend on only two inputs. A 2-variable K Map is a simple square grid of cells, with each cell representing a combination of the two inputs. The representation 2-variable K Map in SOP and POS form is shown below: 3-Variable K Map A 3-variable K Map is used to simplify Boolean functions that depend on three inputs. A 3-variable K Map is a cube with each cell representing a combination of the three inputs. The representation 3-variable K Map in SOP and POS form is shown below: 4-Variable K Map A 4-variable K Map is used to simplify Boolean functions that depend on four inputs. A 4-variable K Map is a grid of cells with four dimensions, each dimension representing one of the four inputs. The representation 4-variable K Map in SOP and POS form is shown below: n-Variable K Map An n-variable K Map can be used to simplify Boolean functions that depend on n inputs, where n is any positive integer. An n-variable K Map is a grid of cells with n dimensions, each dimension representing one of the n inputs. The number of variables used in a K Map determines the number of dimensions in the grid and the number of cells in the map. The goal of using a K Map is to obtain a minimized expression for the Boolean function that is as simple as possible, regardless of the number of variables involved. Let's see how to make k map from boolean expression. How to Solve Boolean Expression K Map Solving a Boolean Expression using Karnaugh Map (K Map) involves the following steps: Step 1 – Create the K Map: Create a two-dimensional grid of cells, with each cell representing a possible combination of inputs to the function. List all possible combinations of inputs, and evaluate the function for each combination, marking the corresponding cell on the K Map with the resulting value. Step 2 – Group cells with the same value: Group together adjacent cells that have the same value (either 0 or 1), forming groups of cells. The goal is to form the largest possible groups, as each group can be reduced to a single term in the simplified expression. Step 3 – Identify prime implicants: Prime implicants are groups of cells that cannot be reduced further without changing their value. Identify all prime implicants in the K Map. Step 4 – Identify essential prime implicants: Essential prime implicants are prime implicants that must be included in the minimized expression for the function. Identify all essential prime implicants in the K Map. Step 5 – Write the minimized expression: Write an expression for each group, using the variable that corresponds to the row or column that the group covers. The resulting expression is the minimized representation of the original Boolean function. Step 6 – Check for don't-care terms: Don't-care terms are terms in the minimized expression that do not affect the output and can be ignored. Check the minimized expression for don't-care terms and remove them if they are present. The above steps provide a general outline for solving a K Map. There are various methods for simplifying Boolean expressions, including the Quine-McCluskey Algorithm, which is based on the K Map and can be used to obtain the minimized expression for a function. Example of K Map Here is an example of how to solve a boolean expression using a K Map. Let us consider the following Boolean Expression, Z = ∑P, Q, R (1, 3, 6, 7) K Map for the given expression can be constructed as The product term from the Red Group: P'R The product term from the Green Group: PQ So, the final minimized boolean expression is: (P'R + PQ) In this example, we can see how the K-Map provides a simple and intuitive method for visualizing and simplifying Boolean functions. By grouping together adjacent cells with a value of 1, we can simplify the Boolean expression and reduce the complexity of the resulting digital circuit. Advantages of K Map Karnaugh Maps (K Maps) have several advantages over other methods of simplifying Boolean expressions and designing digital circuits: K Maps are simple to use and understand, making them accessible to students and professionals with limited experience in digital electronics. K Maps provides an intuitive visual representation of Boolean functions, allowing users to quickly identify patterns and simplify expressions. K Maps is easy to use and can be created quickly, even for complex functions with many inputs. K Maps can be used to find the minimized expression for a Boolean function, reducing the complexity of the expression and the cost of the resulting digital circuit. K Maps allow for the consideration of don't-care terms, which are terms in the Boolean expression that do not affect the output and can be ignored, reducing the complexity of the expression even further. K Maps can be used with other methods of simplifying Boolean expressions and designing digital circuits, providing a complementary tool that can be used in conjunction with other techniques. So, K Maps provides a simple, intuitive, and effective method of simplifying Boolean expressions and designing digital circuits. The use of K-Maps can result in reduced complexity, cost, and design time, making them valuable tools in digital electronics and computer science. Now Let's have a look at the limitations of k map. Disadvantages of K Map Despite their many advantages, Karnaugh Maps (K Maps) also have some disadvantages: K Maps are limited to Boolean functions with a small number of inputs and are not well-suited for functions with a large number of inputs. As the number of inputs increases, the complexity of K Maps increases, making them more difficult to use and understand for complex functions. Creating K Maps can be time-consuming, especially for functions with many inputs or for functions that require multiple iterations to reach the minimized expression. The process of creating and using K Maps can be prone to human error, especially if the user is not familiar with the method or is not careful when identifying minterms and max terms. So, we can say that K Maps is a useful tool for simplifying Boolean functions and designing digital circuits, they are not without limitations and challenges. It is important to carefully consider the advantages and disadvantages of K Maps and to use them appropriately in the context of a particular design problem. Conclusion Karnaugh Maps are invaluable tools for simplifying boolean expressions in digital circuit design. They offer a systematic approach to reducing complexity and optimizing logical statements, which is crucial for efficient circuit implementation. By visualizing and organizing truth table data, engineers and designers can streamline the development of digital systems, ultimately leading to more reliable and cost-effective solutions. Frequently Asked Questions (FAQs) Related to boolean expression using K map Some FAQs on K Map are listed below. 1. When should I use a Karnaugh Map to simplify a boolean expression? Karnaugh Maps are particularly useful for simplifying boolean expressions with up to six variables. For larger expressions, alternative methods like Quine-McCluskey or computer-based algorithms may be more practical. 2. Can Karnaugh Maps handle don't-care conditions? Yes, Karnaugh Maps can accommodate don't-care conditions, which are often represented as "X" in the K-map. These conditions allow for further optimization by considering certain input combinations as irrelevant. 3. Are Karnaugh Maps used in digital circuit design beyond simplifying boolean expressions? Karnaugh Maps are primarily used for simplification, but they can also assist in verifying the correctness of logical designs and identifying redundancies or inconsistencies. 4. Are there software tools available for generating Karnaugh Maps and simplifying expressions automatically? Yes, many digital design and logic simulation software packages include tools for generating Karnaugh Maps and simplifying boolean expressions. These tools can streamline the design process and reduce the potential for human error.