Information Theoretic Heuristics to Find the Minimal SOP Expression Considering Don't Care Using Binary Decision Diagrams

N. Padmavathy[1(✉)], K. S. Jhansi[1], K. Ormila[2], and B. Valarmathi[3]

[1] Department of Electronics and Communication Engineering, Vishnu Institute of Technology, Bhimavaram 08544, India
padmavathy.n@vishnu.edu.in

[2] Department of Electrical and Electronics Engineering, AMK Technological Polytechnic College, Chennai, India

[3] Department of Software and Systems Engineering, School of Information Technology and Engineering, Vellore Institute of Technology, Vellore, Tamil Nadu, India

Abstract. Logic minimization plays a significant part in decreasing the complexity of the circuit since the number of Gates will be diminished… Till today, the conventional or traditional approaches like Boolean laws; Karnaugh map; Quine-Mccluskey are in existence for Boolean expression simplification. The above approaches have several drawbacks; to name a few – logical synthesis complexity, multiple solutions using k-maps, the number of cells increases exponentially with number of variables in a Boolean function and more computational time is needed while solving using Quine Mccluskey. Moreover, the implementation of k-map and Quine-Mccluskey method is difficult in logic synthesis of chip design. The solution to all above drawbacks is the use of Binary Decision Diagrams which is faster and its applicability to large circuits is possible. The main purpose of this work is to reduce the Boolean expressions considering DC function and also to calculate the entropy of the simplified Boolean expression using binary decision diagrams.

Keywords: Binary decision diagrams · Boolean expression · Entropy · Logic gates · Karnaugh map · Minimal SOP · Quine McClusky method

1 Introduction

Over several decades, Boolean logic expression or functions had been represented as a sum-of-product (SOP) or cube form. These expressions has been transformed to simple circuits with application of Boolean algebraic rules, to achieve low logic complexity in terms of variables counts in a Boolean function (BF). There had also been other traditional, conventional approaches like k-map, tabulation method, which are being used over many decades to get minimized logic functions. No optimal solution can be achieved with use of k-Map approach because the combination of AND/OR expression varied from

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2023
Published by Springer Nature Switzerland AG 2023. All Rights Reserved
N. Gupta et al. (Eds.): IC4S 2022, LNICST 472, pp. 86–111, 2023.
https://doi.org/10.1007/978-3-031-28975-0_8

person to person. In order to achieve a potential better result, the tabulation method came into existence. This method failed to compete with the Boolean rules method because of the computational complexity with increase in number of literals in a Boolean expression i.e., the minterms size is exponential to the number of input variable [1]. Motivated with the drawbacks, the authors provide a solution in minimizing the Boolean expressions with use of binary decision diagrams (BDD) and information heuristics method for calculating the entropy of the simplified BF. [2] Used BDD CAD applications (logic synthesis, formal verification) and proposed a variable ordered algorithms for micro-canonical optimization using two procedures viz., initialization and sampling.

1.1 Entropy

The idea of self-information is entropy (the data given by a random process about itself). Entropy is sufficient to study signal imitation by quiet environments. Often there are two or more different arbitrary processes. i.e., one random process represents the source of information and another represents the output of a communication medium in which the encoded source is corrupted by another random process called noise. The fundamental quantity of information theory is entropy [3, 4] and is a measure of uncertainty of an unknown or random quantity as defined using (1)

$$H(k) = -\sum_{k=0}^{k-1} p_k log_2 p \tag{1}$$

where, k is a random variable; p_k is the likelihood of occurrence of a random variable and $H\ (k)$ is the entropy

Entropy can be related to the least number of bits it would take an average to communicate information from one location to another location and is always a relative measure to a probability distribution. Entropy is measured in ***bits*** for base-2 logarithms ***Hartley*** for base-10 logarithms, ***trits*** for base-3 logarithms and ***nats*** for natural logarithms base. It may be noted that natural logarithms are usually more convenient for mathematics while the base – 2 logarithms provide more spontaneous similes. Entropy is proportional to $(-log_2 p_k)$, with the proportionality constant determining what base logarithms are taken in. Averaging over all events according to their respective probabilities, the (1) is got. It may be noted that entropy reaches a maximum iff all states are equiprobable.

1.2 Information

Information is the data that can be stored or transmitted as variables that takes different values; say binary variables can take 0 or 1 as its variable values in digital storage and in case of raw data the outcome is either biased or unbiased. Information is quantified as the number of bits it takes to symbolize a variable and is defined using (2)

$$I(k) = -\sum_{k=0}^{k-1} \log_2 \frac{1}{p_k}\ (bits) \tag{2}$$

In other words, information is connected with a result of uncertainty or un-expectance is the likelihood of the occurrence of the event. The more the unexpectedness or uncertainty of an event, higher is the information. Therefore, information is directly related to uncertainty or inversely related to the probability of occurrence of that event.

2 Literature Review

The authors of [5, 6] have proposed that the central problem of logic synthesis, Binary logic minimization, is most appropriate for reliability analysis and automated reasoning. With BDD and Disjoint Sum of Product (DSOP) reduction, the authors present a strategy for reducing the Boolean Sum of Products function [1]. Proposed on algebraic factorization method. It has been very successful and has become the most common approach in logic synthesis. However, whereas arithmetic and XOR-intensive logic functions, which can be more succinctly represented as combination, produce results that are far from satisfactory, AND/OR-intensive control and random logic functions produce results those are close to optimal. Although logic optimization techniques based on Boolean factorization may provide superior results to algebraic techniques, their high computational complexity prevented them from competing with algebraic techniques.

A new method for efficiently calculating various Shannon information measures using BDD was proposed in [7]. Algorithm for BDD reordering that outperforms other reordering methods in terms of the results it produces. The authors of this paper implemented the technique and reordering algorithm, and the results on circuit's benchmarks are analyzed [8]. Provided an exact method for utilizing BDDs to minimize logic functions. The function is mapped to an extended space using this strategy, which gives it special properties that can be used to calculate its prime and minterms. Conceptually creating a covering table whose columns represent the primes is the next step. The use of BDD as an effective method for minimizing DSOP was suggested and discussed in [9]. An advantage of using BDDs is that they implicitly represent terms. This scheme makes the algorithm faster than techniques that use explicit representation and applies it to large circuits.

Directed acyclic graphs (DAG) [10] were used to represent a BF using OBDD. Tests of functional properties like satisfiability and its equivalence are straightforward thanks to their canonical representation. On OBDD data structures, a number of BF operations can be implemented as graph algorithms. Through symbolic analysis, a wide range of issues can be resolved by utilizing OBDDs. Boolean variables are first used to encode any potential variations in the operating conditions and parameters of the system. In VLSI computer aided design, the authors of [11] emphasized that decision diagrams are the current state of the data structure and have been utilized successfully in numerous other fields. Decision diagrams are utilized extensively and are incorporated into commercial tools. A method for looking into a few families of elementary order – 2 matrices was proposed in [12]. When real vectors are used for Boolean function transformations, new transforms are introduced. One-to-one mapping in a binary/ternary vector space is what these transforms do.

A specific set of fundamental functions can be used to conceptualize arithmetic functions as series expansions in the space of complex valued functions on finite dyadic

(binary relations) groups. A new algorithm transforms disjoint cubes, a simplified representation of Boolean functions, into generalized adding and arithmetic spectra [13]. Exclusive SOPs; [14] that are crucial to logic design and synthesis were proposed. An arithmetic expression that can be thought of as Reed-Muller expressions [15], for switching functions was proposed by replacing Boolean expressions with arithmetic equivalents. Most of studies have considered SOP expression minimization without DC using BDD. The overview of the literature survey indicates that BDD have extensive use in several applications that involves complexity is design. In this paper, minimization of SOP circuits has been computed with DC conditions. The methods which are used to minimizing the Boolean expression are the following methods.

- Boolean Algebra
- Karnaugh Map
- Quine Mc Cluskey

In detail, the concepts with examples on Boolean algebra, K-Map and Tabulation method is available in [16–19]. However, these methods are disadvantageous – like using K-Maps because based on combination of cell, the minimization outcomes may change. In other works, multiple minimized solutions are achievable for single logic expression. Same is the case with QC method but the simplification /minimization approach is time consuming i.e., the procedure is complex and lengthy (Ref example 1). As long as the number of variables does not exceed five or six, the map method of simplification is convenient. It becomes more difficult to determine which combinations form the minimum expression as the number of variables increases. The mapping approach makes it nearly impossible to simplify expressions for complex problems with seven, eight, or even ten variables. The fact that the process of minimization is dependent on human capabilities is another crucial point. The solution to above problems is the use of Binary decision diagrams for the reduction of the Boolean expressions. This paper leads the reader through the minimization process with help of suitable examples. The researcher has considered examples of Boolean expression with and without DCs.

Example 1: Without DC

1. Minimize the BF, f(W,X,Y,Z) = $\sum$m (0, 2, 5, 6, 7, 8, 10, 12, 13, 14, 15) using K -Maps. The solutions are $\overline{X}\overline{Z} + XZ + W\overline{Z} + Y\overline{Z}$ (or) $X'Z' + YZ' + \mathrm{WXY}\prime + W\prime\mathrm{XZ}$
2. Reduce the BF, f(W,X,Y,Z) = $\sum$m (0, 1, 3, 7, 8, 9, 11, 15) using K -Maps. The solution is $\overline{XY} + YZ$
3. Minimize the BF, f(A,B,C,D) = $\sum$m (0, 2, 5, 6, 7, 8, 10, 12, 13, 14, 15) using QC approach. The solutions are $\overline{X}\overline{Z} + XZ + W\overline{Z} + Y\overline{Z}$ (or) $\overline{X}\overline{Z} + XZ + \mathrm{XY} + W\overline{Z}$ *(or)* $\bar{X}\bar{Z} + YZ + \mathrm{WX} + Y\bar{Z}$(or) $\mathrm{X}\prime Z' + \mathrm{XZ} + \mathrm{WX} + \mathrm{XY}$
4. Reduce the BF, f(W, X, Y, Z) = $\sum$m (0, 1, 3, 7, 8, 9, 11, 15) using QC Method.

   The solution is $\overline{XY} + YZ$.

Example 1: With DC

1. Minimize the BF, f(W, X, Y, Z) = ∑m (0, 2, 5, 6, 7, 8, 10, 12, 13, 14, 15) + ∑d (1, 4)and using K -Maps. The solution is W′Y′ + Z′ + X
2. Reduce the BF, f (W, X, Y, Z) = ∑m(0, 1, 3, 7, 8, 9, 11, 15) + ∑d (5, 13) using K -Maps. The solution is X′Y′ + Y′Z + W′X′Z′ + WXY
3. Minimize the BF, f (W, X, Y, Z) = ∑m (0, 2, 5, 6, 7, 8, 10, 12, 13, 14, 15) + ∑d (1, 4) using QC approach. The solution is W′Y′
4. Minimize the Boolean expression f (W, X, Y, Z) = ∑m (0, 1, 3, 7, 8, 9, 11, 15) + d (5, 13) using QC Method. The solution for is X′Y′ + Z′

2.1 Quine-Mccluskey

Quine-Mccluskey method [16–19] is popularly known as Tabulation method. In simplification of Boolean expression the adjacent minterms can be reduced. These minterms are reduced because they differ by only one literal. The minterms whose binary equivalent differs only in one place can be combined to reduce the minterms. This is the fundamental principles of the Quine-Mccluskey method. The minterms are written as their binary equivalents. A horizontal line separates each number of s categories from the number of one-syllable minterms in each group. This separation of minterms helps in searching the binary minterms that differ only in one place. Once the separation is over, each binary number is compared with every term in the next higher category, and if they differ by only one position, a check mark is placed beside each of the two terms and then the term is copied in the second column with a '-' in the position that they differ.

This process of comparison is repeated for every minterm. Once this process is completed the same process is applied to the new resultant terms which are placed in the column. These cycles are repeated until a single cycle passes without removing any more literals. The prime-implicants are the remaining terms, and all terms that did not match during the process. Summing one or more prime implicant gives the simplified Boolean expression. A minimal implicant, such as when a literal product term is removed to produce a non-implicant, is referred to as a prime implicant (PI). Prime implicant is a smallest possible product term got after the removal of all possible literal and further no more removal is possible. The product must be an implicant of a given function for it to be a PI. An essential prime implicant (EPI) is a PI that covers an output of the function that no other PI combination can cover.

2.2 Binary Decision Diagram (BDD)

The Binary Decision Diagram (BDD) is a graphical representation of Shannon's expansion of a logical function, as shown in Fig. 1.The concept of BDD [20] was first proposed and continued by [21]. A BDD is a DAG with two terminal nodes (denoted 0 or 1) of out-degree 0, and a set of variable (branch) nodes of out degree 2. Two successors to the root node are denoted by down sliding lines. In the Fig. 1, these branching nodes indicate a path for each Boolean variable value. The sink node is another name for the '0' and '1' nodes. If the LOW branch (dotted line) is taken from the root, the end point

node '0' is reached, and if the HIGH branch (solid Line) is taken the end point node '1' is reached.

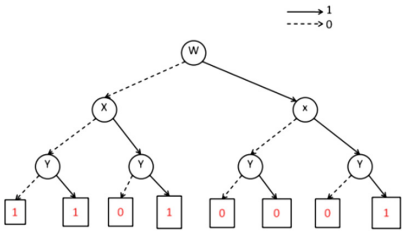

Fig. 1. Binary decision diagram

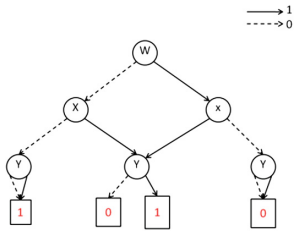

Fig. 2. Reduced ordered BDD

Two crucial rules must be followed by the BDD: first, it must be reduced and then ordered. BDDs are well-known and frequently utilized in logic synthesis and formal verification of integrated circuits.

Ordered Binary Decision Diagram (OBDD). The OBDD extended by [22] (see Fig. 2) are based on a fixed ordering of the variables. If the variables are arranged in a linear manner across all graph paths, the BDD is ordered. OBDD ensures that the variables appear in the same order from the root to the leaves. There are not multiple occurrences of any variable along a path. As an example in the case of n-bit multiplier the number of OBDD representation of the binary multiplier increases exponentially [23]; irrespective of the variable ordering.

Recovering the significant canonicity property for a fixed variable ordering is made possible by the ROBDD representation of each BF, which is a canonical (unique) representation. By constructing their ROBDDs, we can compare BFs and determine whether or not they are equivalent. [24] Has elaborately dealt with the reducing the size of intermediate OBDDs to guarantee a most efficient OBDD in its minimized form having its milestone reference to [25]. In addition, [26] used Evolution algorithms to derive the reduced minimal paths in a BDD.

Uniqueness: Neither the LOW nor HIGH successors of any two dissimilar nodes u and v but share the same variable name. i.e.,

$$var(u) = var(v); low(u) = low(v); high(u) = high(v); implies\ u = v.$$

***Non redundant tests*:** There is no alike LOW and HIGH descendant for variable node u.

The application of BDD is an effective strategy for reducing DSOP. An advantage of using BDD is that terms are implicitly represented. The algorithm is faster than methods based on explicit representations, can be used on large circuits that other methods like Boolean algebra, K-map, and MC can't handle, and gets more complicated when there are more than four variables. In comparison to the methods that are currently in use, the results regarding the size of the reduced DSOPs are also superior. Fault tree analysis (FTA), Bayesian reasoning (BR), product configuration, and private information retrieval (PIR) are just a few of the lesser-known applications of BDD.

Building BDD. The BDD can be constructed for any the Boolean expression or Boolean function. Initially the decimal numbers are converted into binary equivalents and a truth table as shown in Table 1, variables are assigned to bits in the manner of their binary equivalent.

$$Let\ F(y, z) = \sum m(0, 1, 3) \tag{3}$$

Table 1. Truth table for F(y, z) = ∑ m(0, 1, 3)

Decimal equivalent	y	z	F
0	0	0	1
1	0	1	1
2	1	0	0
3	1	1	1

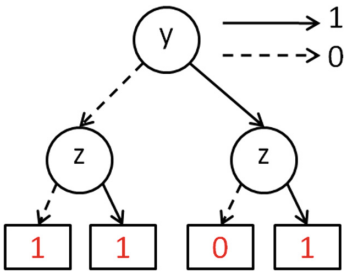

Fig. 3. Building a BDD

The first variable is taken as root *node w a*nd then if the value of the variable is 0 then it is indicated by dotted line to the next literal. If the value of the literal is 1 then it is indicated by solid line. The same procedure is followed by the next variable. In this way the BDD is constructed as shown in Fig. 3.

Reduction Rules of BDD. ROBDD can be obtained from OBDD by repeatedly applying following reduction rules. Figure 4(b) shows the reduction of the BDD using merge isomorphic rule.

Merge Isomorphic Nodes. If two nodes are having the same outputs, then, the corresponding nodes are merged. The Fig. 4(a) is the BDD representation of the BF, F (x, y, z) = ∑ *m*(0, 1, 3, 4, 5, 7). It can be understood that when root node is 'c', the terminal nodes are 1 for *F*(x, y, z) = ∑ *m*(0, 1, 3) with *node w* = 0 and *node x* being a DC condition (i.e., y = 0 or y = 1) as seen in subgraph1 (Fig. 4(a)). A similar type of output (subgraph2) is got for *F*(x, y, z) = ∑ *m*(4, 5, 7) with *node w* = 1 and DC condition (i.e.,*y* = 0 or y = 1). This implies, when *F* = ∑ *m*(0, 1, 3)and *F* = ∑ *m*(4, 5, 7), same

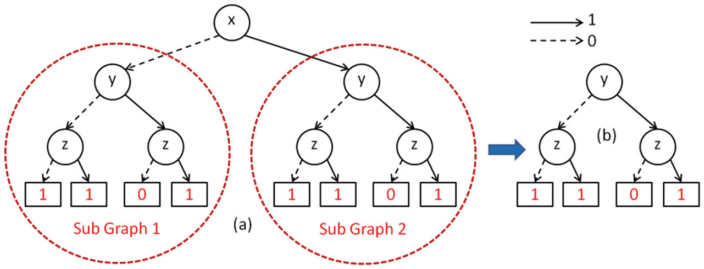

Fig. 4. (a) Isomorphic sub graphs (b) Merged Isomorphic BDD.

terminal *node a*re got. Hence, the subgraphs whose outputs are similar are termed as isomorphic subgraphs.

Isomorphic Subgraphs. The isomorphic subgraphs are the subgraphs which are similar, so the root *node w*nd one of the subgraph is eliminated. Further the elimination of similar subgraphs is done by eliminating the root *node w* and all the subgraphs, retaining one subgraph as shown in Fig. 4 (b). This process of elimination is termed as merging as seen in Fig. 4 (b).

Eliminate Redundant Tests. The two different nodes having same label and same leaves, then such nodes can be eliminated. The root node is removed if its terminal nodes are identical. This elimination is *Bottom-to-top* process. Figure 5(a) is the merged isomorphic graph. The isomorphic graph has a redundancy and can be reduced using redundant test rule. Let *node y* be the root node. According to the Fig. 5 (a), node '*y*' branches out to the terminal node 1 when $w = 0/1$. The sub graph shown within the dotted circles indicate that when *node w* is either 0 or 1, the terminal node output is 1.

Therefore, the terminal node is merged in single *node w* is seen in Fig. 5 (b). The parent (root) *node c*an be removed because the leaf (terminal) nodes are identical, and the leaf node is attached to the next higher root node (see Fig. 5(c)). The merging equivalent node means that the node which has two outputs and that are similar, then merging that node to the one output (see Fig. 5 (b)).

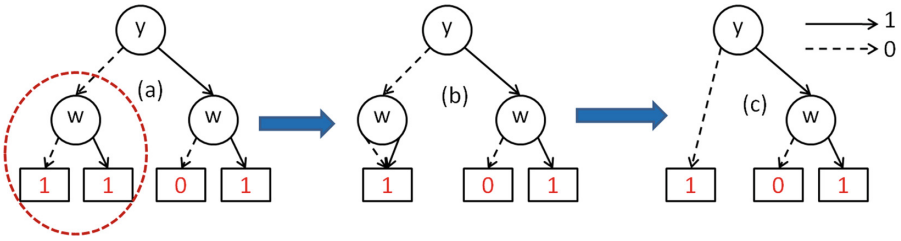

Fig. 5. (a) Merged Isomorphic graph. (b) Redundancy Elimination graph. (c) Reduced BDD

3 Methodology

The methodology involves the procedure of converting the SOP function to minimized SOP function using BDD. Based on the SOP function's variable values, the BDD is created, with the last variable marked with the function output. The SOP function is reduced, by applying the reduction rules like merging equivalent nodes and merging isomorphic sub graphs to the binary decision diagram.

4 Algorithm

The given Boolean expression generates the truth table by converting the given decimal numbers into their respective binary equivalents and obtaining their output function.

4.1 Creating BDD

The BDD can be created by the binary equivalent of a decimal number, which is present in the truth table. In every truth table, the last variable can be represented as a function output.

4.2 Reduction of BDD

The BDD is minimized by applying the reduction rules from bottom level to the top level. The reduction rules are merging equivalent leaves, merging isomorphic nodes and eliminating redundant tests. The reduction rules are thoroughly applied to the BDD to bottom level. After completion of a bottom level, reduction rules have to be applied to the next successive level. The reduction rules must be followed in this manner until the highest level is reached.

4.3 Variable Ordering Using Entropy

Step1: The variable ordering is chosen by calculating the entropy of each variable from truth table for each case i.e. by calculating the information of each node when it becomes '0' and '1'. Then, calculating the entropy by averaging the information of nodes when it is '0' and '1'.
Step 2: This process is continued until the calculation of entropy of all variables is completed. Choosing the variable as the 1^{st}splitting node which has the least entropy among all the variables.
Step 3: Again, calculating the entropy all nodes except 1st splitting node when a 1st splitting node is '0' and '1'. This process is continued until 'n − 1' splitting variables have been obtained.

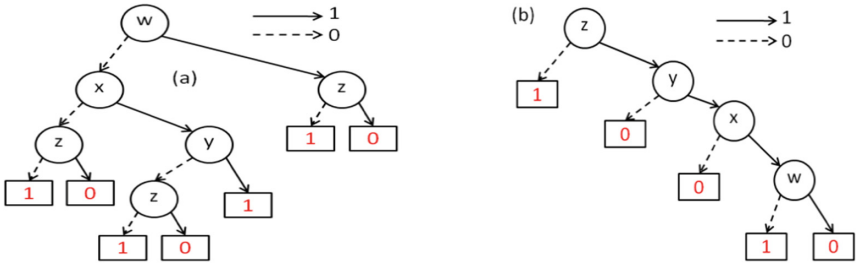

Fig. 6. (a) ROBDD with worst variable ordering. (b) ROBDD with best variable ordering

When building BDD, the right order of variables is very important. Because, as depicted in Fig. 6 (a), the number of 1-paths increases when the worst choice of variable order is chosen, which in turn increases the number of ROBDD nodes. And Fig. 6(b) shows the best variable order with less number of nodes in ROBDD.

4.4 Binate Select

Finding the DSOP from the ROBDD and finally the ROBDD equation is obtained, which is represented in covering matrix. The Binate select is done with respect to the 1st splitting variable, which is selected from entropy calculation.

4.5 Merging

The merging concept is merging the disjoint cubes from binate select. The merging process is started with the last variable of the binate select variable. This process is continued until the first variable of binate select has merged.

5 Example Considered for Study

The Boolean expression with DC has been considered to minimize the Boolean expression by applying the algorithm for the example is as shown in (4).

$$F(w, x, y, z) = \sum m(0, 1, 3, 7, 8, 9, 11, 15) + d(5, 13) \tag{4}$$

Step 1: Generation of truth table. The Table 2 is generated by converting the decimal numbers into its binary equivalents and function output is TRUE (1) if the decimal number is present in the expression. The DC conditions are marked with an "*" because merging the DC conditions is not required. If the SOP is minimized by taking into account the DC, then it is regarded as 1, and if it is not, it is regarded as 0.

Table 2. Truth table for F(w, x, y, z) = $\sum$ m(0, 1, 3, 7, 8, 9, 11, 15) + d(5, 13)

Binary equivalent	W	X	Y	Z	F	Binary equivalent	W	X	Y	Z	F
0	0	0	0	0	1	8	1	0	0	0	1
1	0	0	0	1	1	9	1	0	0	1	1
2	0	0	1	0	0	10	1	0	1	0	0
3	0	0	1	1	1	11	1	0	1	1	1
4	0	1	0	0	0	12	1	1	0	0	0
5	0	1	0	1	*	13	1	1	0	1	*
6	0	1	1	0	0	14	1	1	1	0	0
7	0	1	1	1	1	15	1	1	1	1	1

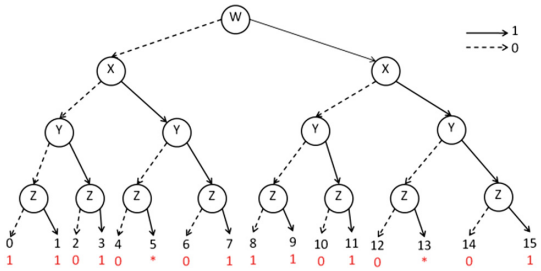

Fig. 7. BDD for **F** = $\sum \boldsymbol{m}(0, 1, 3, 7, 8, 9, 11, 15) + \boldsymbol{d}(5, 13)$.

Step 2: Creating BDD The BDD representation for the BF F = $\sum m(0, 1, 3, 7, 8, 9, 11, 15) + d(5, 13)$ is as shown in Fig. 7.

Step 3: Reducing the BDD: Figure 8, shows a BDD with the sub graphs that are similar. And so, one of the sub graphs is removed and therefore the root node is eliminated and any one of the sub graph is thus considered as depicted in Fig. 9 with the condition that node *w* is 0/1. Figure 9 depict a single x node which is got by merging the two x nodes of Fig. 8. This process is the normal reduction process.

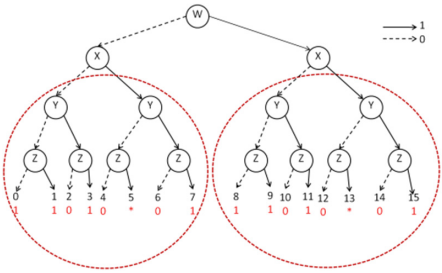

Fig. 8. Reducing the BDD by applying reduction rules

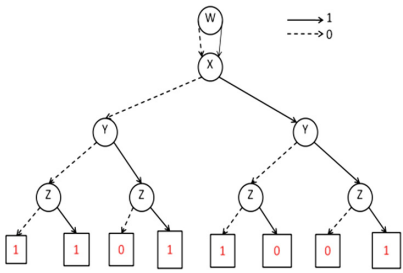

Fig. 9. Merging isomorphic sub graphs

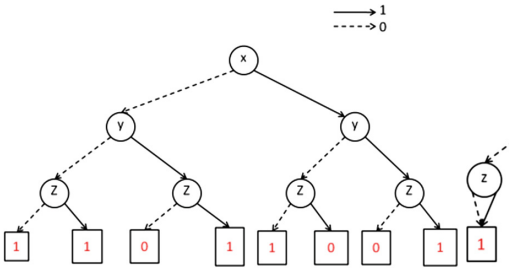

Fig. 10. After merging isomorphic nodes

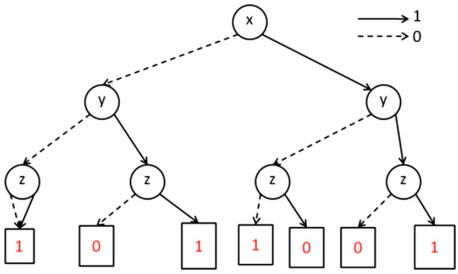

Fig. 11. Merging equivalent nodes

In Fig. 9, as the outputs of *w node* are either 0 or 1, then root *node w* can be eliminated to get Fig. 10. In the left most sub graph, when the value of *y* is 0, the *node z* represents same output 1 then further *node z* is eliminated and after that the *node* y is directly assigned to 1 which results in the reduction of BDD (see Fig. 12). Figure 12, also have two identical sub graphs (see blue dotted circles of Fig. 12) when *node* $y = 0$. So one of them may be eliminated i.e., *node w* is reduced and its corresponding $y = 1$ *is* connected to *node w* (see the brown dotted circles in Fig. 13).

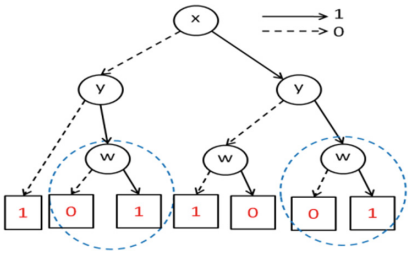

Fig. 12. After merging equivalent and isomorphic nodes in BDD

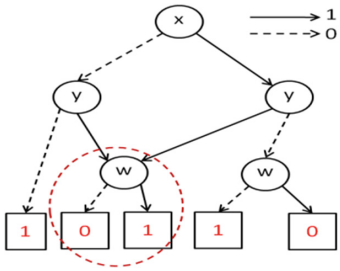

Fig. 13. ROBDD with variable BDD

The output consists of excess usage of nodes which results in increase in the nodes representation. So, therefore the excess usage can be further reduced through node elimination and merging of the isomorphic sub graphs. The output of *node x* & w is same when the value of node *y* is 1. Thus Fig. 13 is the final ROBDD.

Variable ordering by calculating Entropy by considering the DCs:
The information of each node, as well as the times when it reaches 0 and 1, can be used to calculate entropy. And averaging the values for both the cases. In this manner all the variables entropy is calculated.

$$\begin{aligned}
&\mathrm{I}(\mathrm{w}, 0) = (3, 5) = 0.954 \\
&\mathrm{I}(\mathrm{w}, 1) = (3, 5) = 0.954 \quad \mathbf{E(w) = 0.954I} \\
&(\mathrm{x}, 0) = (2, 6) = 0.811 \\
&\mathrm{I}(\mathrm{x}, 1) = (4, 4) = 1 \quad \mathbf{E(x) = 0.905} \\
&\mathrm{I}(\mathrm{y}, 0) = (2, 6) = 0.811 \\
&\mathrm{I}(\mathrm{y}, 1) = (4, 4) = 1 \quad \mathbf{E(y) = 0.905} \\
&\mathrm{I}(\mathrm{z}, 0) = (6, 2) = 0.811 \\
&\mathrm{I}(\mathrm{z}, 1) = (0, 8) = 0 \quad \mathbf{E(z) = 0.4055}
\end{aligned}$$

The first splitting variables are done by observing the entropy of all the nodes and selecting the variable which has least entropy as first splitting node. Here, *node z* has least entropy and selected 'z' as the first splitting variable. The Table 2 is divided into two truth tables depending on the conditions when z is 0 and 1.

For $z = 0$ the truth table is:

Table 3. Generation of truth table considering $z = 0$

w	x	y	F	w	x	y	F
0	0	0	1	1	0	0	1
0	0	1	0	1	0	1	0
0	1	0	0	1	1	0	0
0	1	1	0	1	1	1	0

The truth table when *z* is 0 for all possible combinations of remaining variables is shown in Table 3. The entropy calculation for the remaining variables is shown below. To calculate entropy, refer Entropy calculation of Table 3.

$$\begin{aligned}
&\mathrm{I}(\mathrm{w}, 0) = (3, 1) = 0.811 \\
&\mathrm{I}(\mathrm{w}, 1) = (3, 1) = 0.811 \ \mathbf{E(w) = 0.811} \\
&\mathrm{I}(\mathrm{x}, 0) = (2, 2) = 1 \\
&\mathrm{I}(\mathrm{x}, 1) = (4, 0) = 0 \ \mathbf{E(x) = 0.5} \\
&\mathrm{I}(\mathrm{y}, 0) = (2, 2) = 1 \\
&\mathrm{I}(\mathrm{y}, 1) = (4, 0) = 0 \ \mathbf{E(y) = 0.5}
\end{aligned}$$

For $z = 1$ the truth table is:

The Table 4 is generated for all the combinations of remaining nodes when the *node z* is 1. The entropy value for all remaining variables is as shown below.

Table 4. Generation of truth table considering d = 1.

w	x	y	F	w	x	y	F
0	0	0	1	1	0	0	1
0	0	1	1	1	0	1	1
0	1	0	1	1	1	0	1
0	1	1	1	1	1	1	1

To calculate Entropy, refer Entropy calculation of Table 4.

$$I(w, 0) = (0, 4) = 0$$
$$I(w, 1) = (0, 4) = 0\ \mathbf{E(w) = 0}$$
$$I(x, 0) = (0, 4) = 0$$
$$I(x, 1) = (0, 4) = 0\ \mathbf{E(x) = 0}$$
$$I(y, 0) = (0, 4) = 0$$
$$I\ (y, 1) = (0, 4) = 0\ \mathbf{E\ (y) = 0}$$

The next splitting variable is decided which has the least Entropy for two possible combinations of *node z*. Here, the *node y* is selected as next splitting variable.

For $z = 0; y = 0$ the truth table is:

Table 5. Generation of truth table considering z = 0 and y = 0.

w	x	F	w	x	F
0	0	1	1	0	1
0	1	0	1	1	0

The next splitting variable is selected, based on the entropy calculations of remaining nodes for all possible combination of two *node z* and *y*. The Table 5 is generated for the *node z* is 0 and *y* is 0 for all possible combinations of *node w* and *x*. The entropy calculations of *node w* and *node x* is shown below.

$$I(w, 0) = (1, 1) = 1$$
$$I(w, 1) = (1, 1) = 1\ \mathbf{E(w) = 1}$$
$$I(x, 0) = (0, 2) = 0$$
$$I(x, 1) = (2, 0) = 0\ \mathbf{E(x) = 0}$$

Table 6. Generation of truth table considering z = 0; y = 1.

w	x	F	w	x	F
0	0	0	1	0	0
0	1	0	1	1	0

For $z = 0$; $y = 1$ the truth table is:

The next splitting variable is selected, based on the entropy calculations of remaining nodes for all possible combination of two *node z* and *node x*. The Table 6 is generated for the *node z* is 0 and *y* is 1 for possible permutations of *node w* and *x*. The entropy calculations of *node w* and *node x* is shown below.

$$\begin{aligned} &I(w, 0) = (2, 0) = 0 \\ &I(w, 1) = (2, 0) = 0 \quad \mathbf{E(w) = 0} \\ &I(x, 0) = (2, 0) = 0 \\ &I(x, 1) = (2, 0) = 0 \quad \mathbf{E(x) = 0} \end{aligned}$$

For $z = 1$; $y = 0$ the truth table is:

Table 7. Generation of truth table considering d = 1; c = 0.

w	x	F	w	x	F
0	0	1	1	0	1
0	1	1	1	1	1

The next splitting variable is selected, based on the entropy calculations of remaining nodes for all possible combination of two *node z* and *node y*. The Table 7 is generated for the *node* $z = 1$ and $y = 0$ for all permutations of *w* and *x*. The entropy calculations of *w* and *x* is shown below.

$$\begin{aligned} &I(w, 0) = (0, 2) = 0 \\ &I(w, 1) = (0, 2) = 0 \quad \mathbf{E(w) = 0} \\ &I(x, 0) = (0, 2) = 0 \\ &I(x, 1) = (0, 2) = 0 \quad \mathbf{E(x) = 0} \end{aligned}$$

For $z = 1$; $y = 1$ the truth table is:

The next splitting variable is selected, based on the entropy calculations of remaining nodes for all possible combination of two *node z* and *node y*. The Table 8 is generated for the *node* $z = 1$ and $y = 1$ for all possible permutations of *w* and *x*.

Table 8. Generation of truth table considering d = 1; c = 1.

w	x	F	w	x	F
0	0	1	1	0	1
0	1	1	1	1	1

The Entropy calculations of *w* and *x* is shown below.

$$I(w, 0) = (0, 2) = 0$$
$$I(w, 1) = (0, 2) = 0 \quad \mathbf{E(w) = 0}$$
$$I(x, 0) = (0, 2) = 0$$
$$I(x, 1) = (0, 2) = 0 \quad \mathbf{E(x) = 0}$$

The 3rd splitting variable is selected based up on the entropy values which have the least value in all possible combinations of z and y. Here, *node x* as the next splitting variable. The Table 8 is generated, based on the Table 2. The decimal equivalents may change, because the corresponding binary equivalents changes, when the variables' order is altered. So, the new truth table (Table 9) with its corresponding decimal equivalents is generated from the main truth table.

In general, representation of any node with value 0/1 is shown by a dotted/solid line. Figure 14 is drawn based on the values in the truth table. This procedure is repeated until the final leaves are displayed as outputs and all of the variables have been completed. Figure 15, When *w* = 1and no matter what the values of nodes *x* and *y* are, *z* represent the same output, which is 1.

Therefore, all of the nodes have been eliminated, and the value of *node z* will be directly associated with the value 1 (See Fig. 16). If the *node w* is LOW and the *node y* is HIGH (see dotted small circle in Fig. 15), then irrespective of the values of node *y*; and *z* corresponds to the same output (LOW), then as a result, node *z* can be removed, and the *node y* can be assigned directly to 0.

When node *x* is 0, all other values of node *z* in the leftmost sub tree/sub graph become 1and similarly, if the value of node *x* is 1, all values of node *z* become 0 Then both the leaves are merged into single output leaf. Figure 17 represents the value of *node w* = 1 when *node x* = 0 and *node w* = 0 when the value of *node x* = 1. Finally, the *node w* is eliminated and the terminal nodes are connected to node *x* directly from the OBDD. Figure 17 has three nodes and displays the ROBDD, the final output. Hence, the DSOP from the above BDD are: z + xy'z'.

Binate Covering with Recursion Without DC
The Covering Matrix is:
In the covering matrix, if any variable is not present in disjoint cubes, then that variable is represented with the symbol '2' and is as shown in Table 10 and the corresponding binate select is shown in Fig. 19. The literals can be represented by a mathematical

Table 9. Variable ordered truth table for 4 Variable BF.

z y x w	w x y z	Binary equivalents	F
0 0 0 0	0 0 0 0	0	1
0 0 0 1	1 0 0 0	8	1
0 0 1 0	0 1 0 0	4	1
0 0 1 1	1 1 0 0	12	1
0 1 0 0	0 0 1 0	2	0
0 1 0 1	1 0 1 0	10	0
0 1 1 0	0 1 1 0	6	1
0 1 1 1	1 1 1 0	14	1
1 0 0 0	0 0 0 1	1	0
1 0 0 1	1 0 0 1	9	0
1 0 1 0	0 1 0 1	5	0
1 0 1 1	1 1 0 1	13	0
1 1 0 0	0 0 1 1	3	0
1 1 0 1	1 0 1 1	11	0
1 1 1 0	0 1 1 1	7	1
1 1 1 1	1 1 1 1	15	1

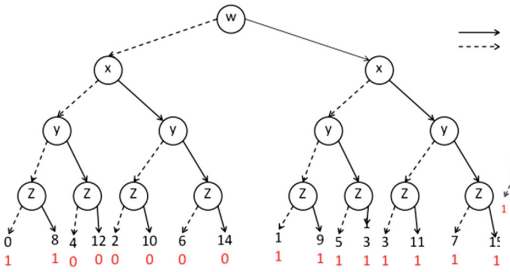

Fig. 14. Generation of variable ordered BDD with the DCs

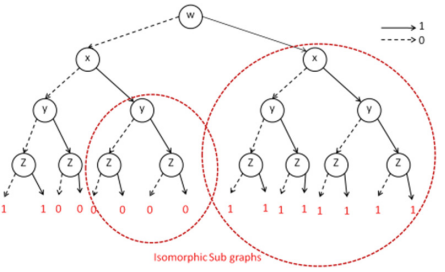

Fig. 15. Merging the equivalent nodes in variable ordered BDD with the DCs

equation as expressed in (4)

$$u = \begin{cases} 2 \; if \;\; u \; is \; missing \\ 0 \; if \;\; u \; is \; a \; complement \\ 1 \; if \;\; u \; is \; normal \end{cases} \tag{4}$$

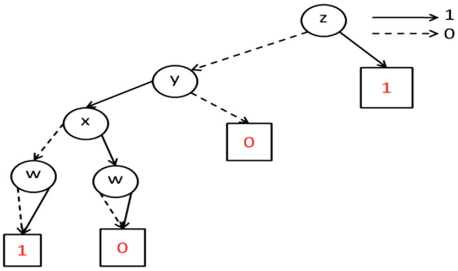

Fig. 16. After merging equivalent invariable OBDD

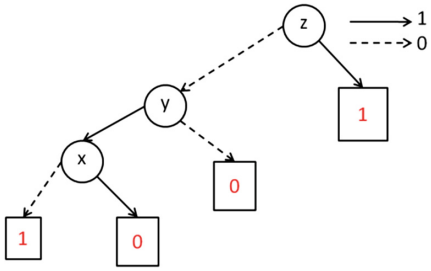

Fig. 17. ROBDD by considering the DCs

Table 10. Covering matrix for DSOP from variable ordered BDD with DC

				W	x	y	z
	x'	y'		2	0	0	2
x'	y	z		2	0	1	1
x	y	z		2	1	1	1

Binate Select

The binate select is a process in which the first node is the first splitting node (for definitions refer Subsect. 4.4), variable order, and this procedure follows up to the 3 variables because, the example is considered for the 4 variables and cannot be further be minimized. Figure 18 gives an idea of how the binate select is performed with reference to Table 10. The binate select is further continued by the merging process (See Fig. 22) to obtain the reduced SOP.

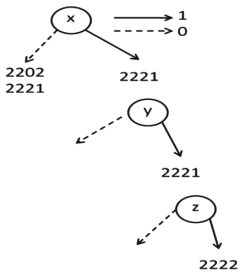

Fig. 18. Binate selecting for DSOP from OBDD without DC

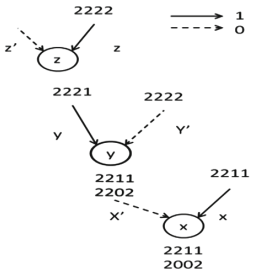

Fig. 19. Merging DSOP from OBDD without DC

Merging

The merging process is done for the nodes which is last node (z) in binate select process. The output of last node (z) in binate select is input to the next node (y/x) of merging. Then, the output changes depending upon the indicating line (dotted line (0)/solid line (1)). In this way, merging is done for all the splitting variables from last node z to the

first node x. If the input of node in merged BDD (see Fig. 19) is same as the output of the binate select (see Fig. 19), then the output is same as the input with no change.

Finally, after simulation the obtained simplified expression is $\boldsymbol{F = yz + x'y'}$. To summarize, the given BF with 4 input variables with 8 product terms require 15 nodes for construction of a BDD with 15 logic gates (AND, OR, NOT). There is a possibility that fewer gates will be used when universal gates are used. The given function is further reduced to an expression with two product terms using BDD, requiring three nodes and five gates. From the obtained results it is clear that the cost of SOP implementation reduces, propagation delay decreases as the number of gates reduces. The same procedure is done by considering DC as explained in preceding session.

Binate Covering with Recursion with DC

The Covering Matrix

The Table 11 represents the disjoint cube variables. The literals can be represented by a mathematical equation as expressed in (5).

Table 11. Covering matrix for DSOP from variable Ordered BDD with the DC

					w	x	y	z
			z		2	2	2	1
	x′	y′	z′		2	0	0	0

Binate Select

The binate select is a process in which the first node is the first splitting node that determines the order of the variables. This procedure continues until there are 3 variables because the example takes into account the DCs for 4 variables.

Merging

The merging process is first done for the node which is the last node (x) in binate select process. The output of the last node x in binate select is the input to the next node (y/z) of merging. Then, the output changes depending on the indicating line (dotted line (0)/solid line (1)). In this way, merging is done for all the splitting variables from last node x to the first node z. If the input of node in merged BDD (see Fig. 21) is same as the output of the binate select (see Fig. 20), and then the output is same as the input with no change.

Finally, after simulation the obtained simplified expression is $F = z + x'y'$. To summarize, the given Boolean function with 4 input variables with 8 product terms and two DC product terms require 15 nodes for construction of a BDD with 17 logic gates (AND, OR, NOT). Using of BDD, the given function reduces to an expression with two product terms and this requires 3 nodes with 4 gates. From the obtained results it is clear that the cost of SOP implementation reduces, propagation delay decreases as the number of gates reduces with the use of BDD than the conventional approaches.

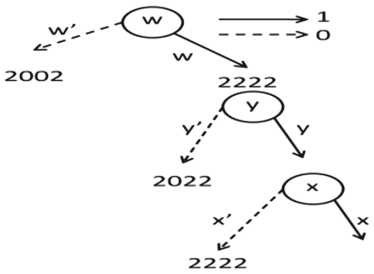

Fig. 20. Binate select for DSOP from BDD with DC

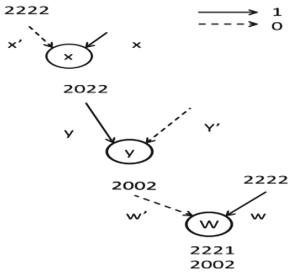

Fig. 21. Merging DSOP from BDD with DC

6 Simulation Results

The simulation results are the results obtained from the written code for the examples considered with and without DCs in session 2. The process of applying the steps to Binary decision diagram to convert to the reduced binary decision diagram is provided as follows.

6.1 Boolean Expression with the DC

The BF with DC (5) is considered as an example

$$F = (0, 1, 3, 7, 8, 9, 11, 15) + d(5, 13) \tag{5}$$

$$\begin{aligned} F =& w'x'y'z' + w'x'y'z + w'x'yz + w'xyz + wx'y'z' + wx'y'z \\ &+ wx'yz + wxyz + w'xy'z + wxy'z \end{aligned} \tag{6}$$

The Boolean function is expressed as Boolean expression (5). The Fig. 22 shows the BDD of the given SOP created for the all possible combinations of input variables as 0 and 1. The output of the sink node indicates the function outputs.

The reduction rules reduce the redundant nodes and merge equivalent nodes and are applied to the BDD (Fig. 22), and then the resultant output (Fig. 23) is the ROBDD. The variable OBDD is obtained in the same manner as discussed in afore mention sections to get variable ordered BDD (Fig. 24); reduced variable ordered BDD (Fig. 25).

6.2 Expressions Considered for 3 and 4 Variables (with & Without DC)

The expressions considered for 3, 4 variables with and without DC has been taken to generate the Tables 12, 13, 14 and 15 for number of nodes and logic gates for all possible variable orders.

Expression for 3 Variables Without DC. Boolean Function: $F = \sum m(0, 1, 2, 4, 6)$; Boolean Expression: $F = x'y'z' + x'y'z + x'yz' + xy'z' + xyz'$

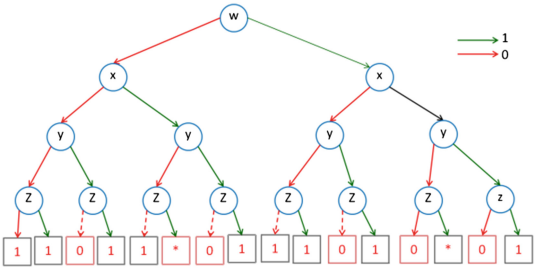

Fig. 22. BDD for SOP with DC.

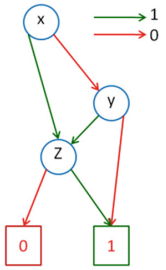

Fig. 23. ROBDD with DC

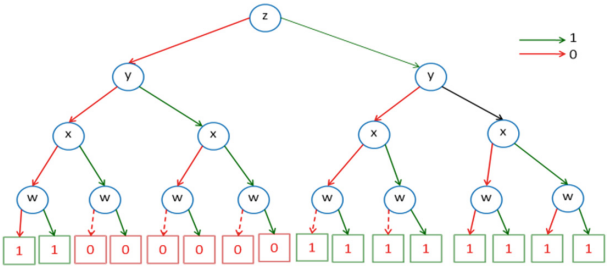

Fig. 24. Variable OBDD with DC

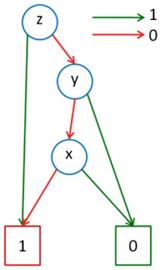

Fig. 25. ROBDD of Fig. 24 with DC

Table 12. Results for 3 variable Boolean expression without dc

$F = x'y'z' + x'y'z + x'yz' + xy'z' + xyz'$		
Variable order	Minimization using BDD	
	Number of nodes in ROBDD	Number of logic gates
xyz	5	7
xzy	5	7
yzx	5	7
yxz	5	7
zxy	5	7
zyx	5	7

Without simplifying, the number of nodes and logic gates required for the given Boolean function is as follows:

$$\text{Number of Gates} = 11; \text{Number of Nodes} = 7$$

Table 12 shows that the variable order used in the BDD approach that determines the number of logic gates and nodes in ROBDD. For all possible combinations (No. of combinations $3! = 6$) of 3 variables SOP without the DC the order of the number of

nodes and logic gates in ROBDD is as mentioned in Table 12. Table 12 exhibits that the variable order has no effect on the number of logic gates and nodes. Hence, any variable order may be considered for this particular case.

Expression for 3 Variables with DC. Boolean Function: $F = \sum m(0, 1, 2, 4, 6) + d(5)$; Boolean Expression $F = x'y'z' + x'y'z + x'yz' + xy'z' + xy'z + xyz'$.

Without simplifying, the number of nodes and logic gates required for the given Boolean function is as follows:

$$\text{Number of gates} = 12; \quad \text{Number of nodes} = 7$$

According to Table 13, the number of logic gates and nodes in ROBDD are proportional to the variable order when employing the BDD approach. For all possible combinations (No. of combinations $3! = 6$) of 3 variables SOP with the DC, for different variable order, the number of nodes and logic gates in ROBDD are same as referenced in Table 13. According to Table 13, the number of logic gates and nodes remains constant regardless of the variable order. Moreover, it is understood that the variable order (*xyz*; *xzy*; *yzx*; *yxz*) the logic circuit may need 4 gates with 4 nodes in the ROBDD. Hence, any variable order may be considered for this particular case and would be the best order. The variable order (*zxy*; *zyx*) are coined as *worst order* because more nodes are present after reduction of BDD though the number of logic gates are same.

Table 13. Results for 3 variable Boolean expression by considering DCs

$F = x'y'z' + x'y'z + x'yz' + xy'z' + xyz'$		
Variable order	Minimization using BDD	
	Number of nodes in ROBDD	Number of logic gates
xyz	4	4
xzy	4	4
yzx	4	4
yxz	4	4
zxy	4	4
zyx	5	4

Expression 4-variable without DC. Boolean Function: $F = \sum m(7, 10, 12, 14, 15)$.

Boolean expression $F = w'xyz + wx'yz' + wxy'z' + wxyz' + wxyz$.

Without simplifying, the number of nodes and logic gates required for the given Boolean function is as follows:

$$\text{Number of gates} = 11; \quad \text{Number of nodes} = 15$$

For every possible combination (No. of combinations $4! = 24$) of the variables, the number of nodes and logic gates in ROBDD required are mentioned in the Table 14. The

best variable order obtained using entropy calculation is '*wxyz*' because it requires less number of nodes (6) and logic gates (4) compared to other combinations in ROBDD. The worst variable order is '*wxyz*' and '*wyzx*' because it requires more number of logic gates (8) and nodes (8) for expression without DC. While remaining all other combinations mostly the number of gates (6) with 7 ROBDD nodes.

Table 14. Results for 4 variable Boolean expression without DCs

$F = w'xyz + wx'yz' + wxy'z' + wxyz' + wxyz$					
Variable order	Minimization using BDD		Variable order	Minimization using BDD	
	No. of nodes in ROBDD	No. of Logic Gates		No. of nodes in ROBDD	No. of Logic Gates
w x y z	8	8	y w z x	7	6
w x z y	6	4	y w x z	7	6
w yx z	7	5	y x w z	8	7
w y z x	8	8	y x z w	7	6
w zx y	7	7	y z x w	7	6
w z y x	8	4	y z w x	7	6
x y z w	8	7	z w x y	7	6
x y w z	7	6	z w y x	7	6
x wy z	7	6	z x w y	7	6
x w z y	7	6	z x y w	7	6
x z w y	7	6	z y w x	7	5
x z y w	7	6	z y x w	7	7

Expression 4-variable with DC. Boolean Function:

$$F = \sum m(7, 10, 12, 14, 15) + d(13).$$

Boolean expression $F = w'xyz + wx'yz' + wxy'z' + wxyz' + wxyz + wxy'z$.

The number of nodes and logic gates required for the given Boolean function without simplification is as given below:

Number of gates = 12; Number of nodes = 15

For every possible combination (No. of combinations 4! = 24), the number of nodes and logic gates in ROBDD required are obtained as mentioned in the Table 15 with DC. The best variable order obtained using entropy calculation is 'wxyz' and 'wyxz' because it requires less number of nodes (6) and logic gates (4) compared to other combinations in ROBDD. The worst variable order are at most 9 combinations (xyzw, xywz; xzwy; xzyw; yxzw; yzxw; yzwx; zxyw and zywx) because it requires more number of logic gates (7) and nodes (8) for expression without DC.

Table 15. Results for 4 variable Boolean expression with DCs

$F = w'xyz + wx'yz' + wxy'z' + wxyz' + wxyz$					
Variable order	Minimization using BDD		Variable order	Minimization using BDD No. of nodes in ROBDD	No. of logic gates
	No. of nodes in ROBDD	No. of logic gates			
w x y z	7	6	y w z x	7	6
w x z y	**6**	**4**	y wx z	7	6
w yx z	**6**	**4**	y x w z	7	6
w y z x	7	5	y x z w	8	7
w zx y	7	5	y z x w	8	7
w z y x	7	6	y zw x	8	7
x y z w	8	7	z w x y	7	6
x y w z	8	7	z w y x	7	6
x wy z	7	6	z x w y	8	6
x wz y	7	6	z x y w	8	7
x z w y	8	7	z y w x	8	7
x z y w	8	7	z y x w	6	5

6.3 Graphical Representation of Results

The program was implemented in Intel Corei3, Inspiron 14, 6006U CPU, 2 GHz processor with 4 GB RAM with Python 2.0.1Student Edition platform. The simulation results are explained in detail. As the number of input variables increased, the amount of time required in generating the BDD increased as well (the graph was not provided) and Figs. 26 and 27 depict the reduction in number of logic gates and nodes using BDD for 3 and 4 Variable BF.

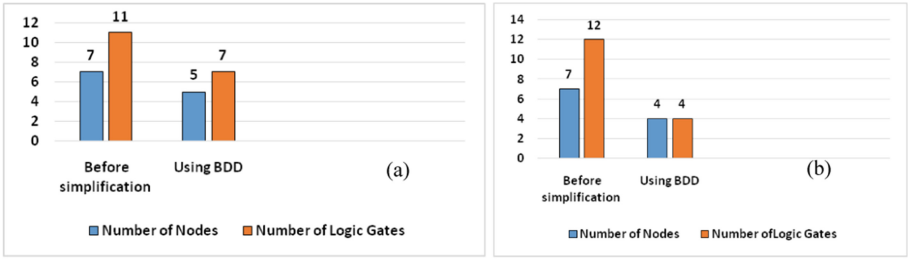

Fig. 26. Comparison of No. of Nodes and No. of Logic gates for a 3 variable Boolean function before simplification and after using BDD (a) without DC (b) with DC

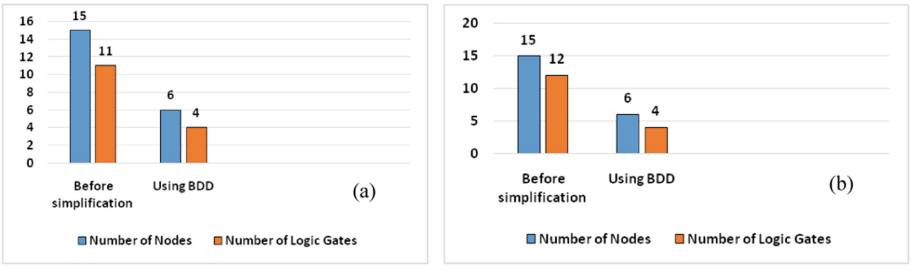

Fig. 27. Comparison of No. of Nodes and No. of Logic gates for a 4 variable Boolean function before simplification and after using BDD (a) without DC (b) with DC

7 Conclusion

The paper provides a detailed understanding of the minimization of the Boolean functions (a) without DC (b) with DC using BDD. The results indicate that BDD is a good alternative for SOP Boolean function reduction as it takes less number of nodes for the logic circuit implementation; less propagation delay, low power consumption. In this work, an algorithm has been developed to minimize the Boolean expression using BDD with the aid of reduction and merging rules. The number of nodes is counted, each node's entropy is calculated, and the variable is then arranged in ascending entropy order. Using, covering matrix, binate select and merging techniques, the ROBDD is obtained based on the best variable order. The observations of the obtained results are summarized as follows- The more input variables there are in the BF, the longer it takes to generate the BDD. The number of nodes and logic gates are almost same for the lower inputs (say 3 variable BF) for any variable order, while 50% reduction in number of gates are got when 4- input variables is considered. To learn more about the propagation delay, power consumption, this approach may be validated using a suitable VLSI tool to obtain optimized results.

References

1. Yang, C., Ciesielski, M.: BDS-a BDD-Based Logic Optimization System, TR-CSE-00-1, pp. 1–25 (2000)
2. Minna, L., Jo, S-Y.: A minimization technique for BDD based on micro canonical optimization. The Kips Trans.: Part A **8**(1), 48–55 (2001)
3. Shannon, C.E.: A mathematical theory of communication. The Bell Syst. Tech. Jour. **27**(3), 379–423 (1948)
4. Shannon, C.E., Weaver, W.W.: The Mathematical Theory of Communication. University of Illinois Press (1949)
5. Sensarma, D., Subhashis, B., Krishnendu, B., Saptarshi, N., Samar, S.: On an optimization technique using binary decision diagram. Int. J. Comput. Sci., Eng. Appl.. **2**(1), 73–86 (2012)
6. Banerjee, S., Sensarma, D., Basuli, K., Naskar, S., Sarma, S.S.: The reconstruction conjecture. The Second Int. Conf. Comput. Sci. Inform. Technol. **86**(3), 17–25 (2011)
7. Popel, D.: Towards Efficient Calculation of Information Measures for Reordering of Binary Decision Diagrams. In: Proceedings IEEE International Symposium on Signals Circuits and Systems, pp. 509–512 (2001)

8. Swamy, G.M.: An exact logic minimizer using implicit binary decision diagram based methods. In: ICCAD'94 Proceedings of the 1994 IEEE/ACM international conference on Computer aided design (1994)
9. Fey, G., Drechsler, R.: Utilizing BDDs for disjoint SOP minimization. In: the 2002 45th Midwest Symposium on Circuits and Systems, vol. 2, pp. 306–309 (2002)
10. Bryant, R.E.: Symbolic boolean manipulation with ordered binary-decision diagrams. ACM Comput. Surv. **24**(3), 293–318 (1992)
11. Drechsler, R., Sieling, D.: Binary decision diagrams in theory and practice. Software Tools Technol. Transfer. **3**, 112–136 (2001)
12. Falkowski, B.J.: Spectral Methods for Boolean and Multi-valued Input Logic Functions, Dissertation. Portland State University (1991)
13. Falkowski, B.J., Schafer, I., Chang, C.-H.: An effective computer algorithm for the calculation of disjoint cube representation of boolean functions. In: Proceedings of 36th Midwest Symposium on Circuits and Systems, vol. 2, pp. 1308–1311 (1993)
14. Mishchenko, A., Perkowski, M.: Fast Heuristic Minimization of Exclusive-Sum-of-Products. In: International workshop on Applications of the Reed-Muller Expansion in Circuit Design, pp. 242–250 (2001)
15. Astola, J., Stankovic, R.: Fundamentals of Switching Theory and Logic Design – A Hands on Approach. Springer (2006)
16. Mano, M., Ciletti, M.D.: Digital Logic And Computer Design, 4th edn. Pearson PHI (2016)
17. Kohavi, Z., Jha, N.K.: Switching And Finite Automata Theory, 3rd edn. Cambridge University Press (2009)
18. Godse, A.P., Godse, D.A.: Switching Theory and Logic Design. Technical publications Pune (2009)
19. Kumar, A.A.: Fundamentals of Digital Circuits, 4th edn. PHI Learning Publisher (2016)
20. Lee, C.Y.: Representation of switching circuits by binary decision programs. Bell Syst. Technol. **38**(4), 985–999 (1959)
21. Akers, S.B.: Binary decision diagrams. IEEE Trans. Comput. **C–27**(6), 509–516 (1978)
22. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Comput. **35**(8), 677–691 (1986)
23. Bryant, R.E.: On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. IEEE Trans. Comput. **40**(2), 205–213 (1991)
24. Coudert, O., Madre, J.C., Berthet, C.: Verifying temporal properties of sequential machines without building their state diagrams. In: Clarke, E.M., Kurshan, R.P. (eds.) CAV 1990. LNCS, vol. 531, pp. 23–32. Springer, Heidelberg (1991). https://doi.org/10.1007/BFb0023716
25. Clarke, E.M., Kurshan, R.P. (eds.), American Mathematical Society, pp. 75–84 (1990)
26. Bryant R.E.: Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, vol. 24(3), 273–318 (1992)