Simplification of Boolean Functions Using Boolean Differences

M.B. Ghaznavi-Ghoushchi* and A.R. Nabavi

This paper presents a new method for simplification of Boolean functions based on Boolean differences. The proposed method is applicable to various forms of Boolean functions, including truth tables and Binary Decision Diagrams (BDDs). The Boolean differences are extended to cover the truth tables with don't-care components and cutset graphs in BDDs. The results of simplification agree with Quine-McCluskey and ESPRESSO methods. Experimental tests on MCNC and Berkeley PLA benchmarks show that the proposed method gains a performance of 1.5-10 times faster than ESPRESSO. The algorithms of the proposed method are implemented in Java/Perl/C++, and a toolset for logic function simplification is developed.

INTRODUCTION

Logic synthesis is one of the most important steps in the design process of digital Integrated Circuits (ICs). There are two basic approaches for logic minimization, namely: exact and heuristic approaches. Many minimization approaches for circuit design have been proposed in [1-4]. Also, for two-level realizations, powerful tools for exact minimization have been developed [5-8]. The exact approach computes minimum cover based on the Quine-McCluskey (QM) method [3-5,9-11], which uses minterms as a starting point for minimization and requires computing all prime implicants [12,13]. Therefore, the number of minterms grows exponentially, with the number of input variables [12]. In fact, prime implicants are, at most, $O(2^n)$ and can be, at least, $O(3^n/n)$, while minterms vary from 0 to $2^n$ [13]. Some more modern methods, including the well-known ESPRESSO [5-8], with its later improvements ESPRESSO-EXACT [5] and ESPRESSO-SIGNATURE [4], combine the use of two basic phases known as PI generation and Covering Problem (CP) solution, reducing the number of implicants to be processed. The usual problems in conventional approaches are memory size and time (CPU Time) [4]. There has been some effort made to solve these problems [14,15]. In this paper, a new application scope of Boolean differences for simplification of Boolean functions is introduced. The proposed method can handle various formats of Boolean functions, including the truth table (complete, incomplete without don't-care, incomplete with don't-cares), various types of SOP and BDD-derived cutset graphs. The method has been tested on several different design benchmark circuits. These experiments proved that the new method has, up to an order of magnitude, speed improvement over ESPRESSO.

The remainder of the paper is organized as follows. First, the basic definitions and theorems are presented. Then, the implementation of the proposed algorithms and the theorems needed in simplification are described. After that, the experimental results are given, and finally, the paper is concluded.

BASIC DEFINITIONS AND THEOREMS

In this section, basic definitions and theorems related to Boolean difference and the proposed simplification approach are given.

The Boolean difference of an $n$-variable Boolean function, $f() = f(\nu_1, \nu_2, \cdots, \nu_n)$, with respect to a variable $\nu_i$ is denoted as $\partial f/\partial \nu i$ and defined as [16,17]:

$$\frac{\partial f}{\partial \nu i} = f(\nu_i) \oplus f(\nu_i \oplus 1),$$  (1)
where:

\[ f(y) = f(v_1, v_2, \ldots, v_{n-1}, 1, v_{n+1}, \ldots, v_n) \]  \hspace{1cm} (2)

\[ f(y \oplus 1) = f(v_1, v_2, \ldots, v_{n-1}, 0, v_{n+1}, \ldots, v_n) \]  \hspace{1cm} (3)

and \( \oplus \) denotes the exclusive-or operator. The Boolean difference represents the minterms, for which \( y \) is observable at \( f \). When \( \frac{\partial f}{\partial y} = 0 \), then the function does not depend on \( y \) \cite{17}. In this paper, this property is employed to extend the definition of Boolean difference to truth tables and binary decision diagrams.

**Definitions and Theorems**

**Boolean Vector:** For a given Boolean function:

\[ f(y) = f(v_1, v_2, \ldots, v_{n-1}, y, v_{n+1}, \ldots, v_n). \]  \hspace{1cm} (4)

The corresponding truth table contains \( 2^n \times (n + 1) \) elements, where \( 2^n \times n \) elements are variables and \( 2^n \times 1 \) elements are function values. Each combination of the input variables, \( v_i \), is called a Boolean vector and denoted here by \( L_1, L_2, \ldots, L_n \):

\[ L_1 = (v_1, v_2, \ldots, v_{n-1}, y, v_{n+1}, \ldots, v_n) \]

\[ L_2 = (v_1, v_2, \ldots, v_{n-1}, \bar{y}, v_{n+1}, \ldots, v_n) \]

\[ L_3 = (v_1, v_2, \ldots, v_{n-1}, v_n, v_{n+1}, \ldots, v_n) \]

\[ L_{2n} = (v_1, v_2, \ldots, v_{n-1}, \bar{v}_n, v_{n+1}, \ldots, v_n) \] \hspace{1cm} (5)

These sets are represented by a triplet \( T_2 = \langle L_1, L_{2n}, \bar{y} \rangle \), where \( L \) is the set of Boolean vectors, \( 1 \leq i \leq 2^n \), \( F \) is the vector of Boolean values of \( f \) and \( n \) is the number of input variables. Note that \( f(L_2) \in \{0, 1\} \).

**Definition 1**

**Boolean Vector Complement**

Given a Boolean vector \( L_n = (v_1, v_2, \ldots, v_{n-1}, v_n, v_{n+1}, \ldots, v_k) \), the vector:

\[ L_{n}^{C} = (v_1, v_2, \ldots, v_{n-1}, \bar{v}_n, v_{n+1}, \ldots, v_k) \] \hspace{1cm} (6)

obtained from \( L_n \) by negating the element \( \bar{v}_n \) is defined as the Boolean vector complement of \( L_n \), with respect to \( v_n \). For instance, if \( L_n = (1, 1, 0, 1) \), then \( L_n^{C} = (0, 1, 1, 0) \). \( L_n^{C} = (0, 1, 0, 1, 1, 1) \) and \( L_{n}^{C} = (1, 1, 0, 0, 0) \).

**Definition 2**

**Boolean Vector Difference**

The Boolean difference of a vector \( L_n \), with respect to element \( \bar{v}_n \), is defined as:

\[ \frac{\partial L_n}{\partial \bar{v}_n} = f(L_n) \oplus f(L_n^{C}) \] \hspace{1cm} (7)

**Theorem 1**

Let \( f = f(v_1, v_2, \ldots, v_{n-1}, v_n, v_{n+1}, \ldots, v_k) \) be a Boolean function with truth table \( T_2 = \langle L_1, F, y \rangle \). When the Boolean vector difference of \( L_n = (v_1, v_2, \ldots, v_{n-1}, \bar{v}_n, v_{n+1}, \ldots, v_k) \), with respect to \( v_n \), is zero, then the truth table is independent of \( v_n \):

\[ \frac{\partial L_n}{\partial \bar{v}_n} = 0 \implies T_2 \text{ is independent of } v_n \] \hspace{1cm} (8)

The proof for Theorem 1 is given in Appendix A.

**Example 1**

For a Boolean function \( f = f(v_1, v_2, v_3) = v_1 + v_2 \), the truth table is shown in Table 1. For \( u = 2 \) and \( i = 3 \):

\[ L_2 = (0, 0, 1, 0, 0) \]

\[ L_3 = (1, 0, 0, 0) \]

\[ f(L_3^C) = f(0, 0, 1, 0, 0) = 0 \]

\[ \frac{\partial L_3}{\partial \bar{v}_2} = 0 \implies (\frac{\partial L_3}{\partial \bar{v}_2} = f(L_3) \oplus f(L_3^C) = 0). \]

Thus, \( T_2 \) is independent of \( v_2 \) in row 2. On the other hand, when \( a = b = 0, f = 0 \).

**Definition 3**

**Incomplete Truth Table**

A complete truth table with \( n \) variables has \( 2^n \times n \) elements. A truth table with less than \( 2^n \times n \) elements is called an incomplete truth table. An incomplete truth table is a truncated truth table, denoted here by \( T_3 = \langle L_1, F_1, y \rangle \) \( 1 \leq m \leq n \), where \( L \) is the set of Boolean Vectors, \( F_1 \) is the vector of Boolean values of \( f \) and \( m \) is the maximum number of input variables. The truth table is not complete, therefore, \( F_1 \) in \( T_3 \) is a subset of \( F \) in \( T_2 \).

**Theorem 2**

Let \( T_3 = \langle L_1, F_1, y \rangle \) \( 1 \leq m \leq n \) be an incomplete truth table. If the Boolean vector difference of \( L_n \) with respect to \( v_n \), is zero, then \( T_3 \) is independent of \( v_n \):

\[ \frac{\partial L_n}{\partial \bar{v}_n} = 0 \implies T_3 \text{ is independent of } v_n \] \hspace{1cm} (9)

Note that in \( T_3 \), despite \( L_n \), \( L_n^{C} \) may not exist. Therefore, the Boolean vector difference in \( T_3 \) is defined only if \( L_n^{C} \) exists. The proof for Theorem 2 is given in Appendix A.

**Table 1. Truth table of \( f = a + b \).**

<table>
<thead>
<tr>
<th>Row</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
**Definition 4**

*Truth Tables with Don’t-Cares*

Truth tables with don’t-cares are denoted here by $T_X = \langle L_X, F_1, m \rangle$; $m \leq n$, where $L_X$ is the set of Boolean vectors with don’t-cares, $F_1$ is the vector of Boolean values of $f$ and $m$ is the maximum number of input variables. Since the truth table contains don’t-care components, $F_1$ in $T_X$ is a subset of $F$ in $T_2$. A don’t-care component (denoted by $X$) can be replaced by 1 or 0.

**Definition 5**

*Boolean Vector Complement in $T_X$*

A Boolean vector with don’t-care is denoted by $L_X = (s_1, s_2, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots, s_n)$, where each element, $s_i$, is 0, 1 or $X$ (i.e., $s_i \in \{0, 1, X\}$). The term Boolean vector complement is defined when the element $s_i$ in row $m$ is either 1 or 0, but not $X$. The Boolean vector complement (with don’t-care) of $L_X$ for element $i$ is $L_X^{\sim i} = (s_1^*, s_2^*, \ldots, s_{i-1}^*, s_i^*, s_{i+1}^*, \ldots, s_n^*)$, where $s_i^* = \begin{cases} s_j^* & \text{if } (j = i), \\ 1 \text{ or } X & \text{if } (s_j = 1 \& \& j \neq i), \\ 0 \text{ or } X & \text{if } (s_j = 0 \& \& j \neq i), \\ 0 \text{ or } 1 \text{ or } X & \text{if } (s_j = X \& \& j \neq i). \end{cases}$ \hspace{1cm} (10)

**Definition 6**

*Boolean Vector Complement Set*

Through this definition, it is seen that $L_X^{\sim i}$ is not unique. The set of all Boolean vectors, which meet the conditions of Equation 10, is called a Boolean vector complement set in this paper.

**Definition 7**

*Boolean Vector Complement Set as a Regular Expression*

For the sake of simplicity, the Boolean vector complement is denoted in a regular expression format [18]. The regular expression representation of $L_X^{\sim i}$ is:

$L_X^{\sim i} = \langle/re/(L_X, i)\rangle = \{\text{Set of all matches for } s_i^*\}$ \hspace{1cm} (11)

where $\langle/re/(L_X, i)\rangle$ is the regular expression for the $i$th bit (element) in $L_X$.

**Example 2**

For $L_X = (0, 0, X), L_X^{\sim 1} = \langle/re/(L_X, 1)\rangle$ and $\langle/re/(L_X, 1)\rangle = \{\text{set of all matches such that their first element is 1, the second element is either 0 or } X \text{ and the third element is } 0, 1, \text{ or } X\}$.

**Definition 8**

*Extended-XOR*

An Extended-XOR operator, denoted here by $\overline{\overline{\cdot}}$, is a two-argument XOR with the following property:

$[L_X \overline{\overline{\cdot}} [L_X \overline{\overline{\cdot}}]_2] = f(L_X_{u1}) \oplus f(L_X_{u2})$. \hspace{1cm} (12)

This operator accepts two Boolean vectors as its arguments.

**Definition 9**

*Overload-XOR*

An overload-XOR operator, denoted here with $\overline{\overline{\cdot}}$, is a two-argument XOR, which accepts a Boolean vector as its first argument and a set of Boolean vectors as its second argument. The final result of overload-XOR is zero when the XOR of individual extended-XOR is 1 and 0 otherwise.

**Definition 10**

*Boolean Difference for $L_X$*

$L_X$ is a single Boolean vector, while $L_X^{\sim i}$ is a set of Boolean vectors. The Boolean difference for $L_X$ is defined as:

$\frac{\partial L_X}{\partial \overline{\overline{\cdot}}_i} = [L_X] \oplus \langle/re/(L_X, i)\rangle$. \hspace{1cm} (13)

By using Definitions 8 and 9, the Boolean difference for $T_X$ is given by:

$\frac{\partial L_X}{\partial \overline{\overline{\cdot}}_i} = [L_X] \oplus \langle/re/(L_X, i)\rangle$

$\frac{\partial L_A}{\partial \overline{\overline{\cdot}}_i} = [L_X] \oplus \{[L_{X1}], [L_{X2}], \ldots, [L_{Xn}]\}$

$= \bigoplus^k \{[L_{X1}] \oplus [L_{X2}], \ldots, [L_{Xk}]\}$

$= \bigoplus^k \{f(L_{X1}) \oplus f(L_{X2}), \ldots, f(L_{Xk})\}$

Note that in $T_X$, despite $L_X, L_X^{\sim i}$ may not exist. Therefore, the Boolean difference in $T_X$ can be defined only if $L_X^{\sim i}$ exists. Also, $\bigoplus^k$ denotes a multi-argument XOR operator.

**Example 3**

An incomplete truth table, $T_X$, is shown in Table 2. For variable $c$ in $u = 4$ and $i = 3$: 
Table 2. Incomplete truth table $T_X$.

<table>
<thead>
<tr>
<th>Row</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\frac{\partial L_{X_4}}{\partial v_2} = [L_{X_4}] \oplus \langle r \vee (L_{X_4} \oplus 3) \rangle
\]

\[
= [110X] \oplus \langle [1X][1X][1][01X]\rangle
\]

\[
= [110X] \oplus \{[1110],[1111]\}
\]

\[
= 2\{[110X]\oplus[1110],[110X]\oplus[1111]\}
\]

\[
\frac{\partial L_{X_4}}{\partial v_2} = 2(f(110X) \oplus f(1110), f(110X) \oplus f(1111))
\]

\[
= 2\{0 \oplus 0, 0 \oplus 1\} = 2\{0,1\} = 1.
\]

This shows that $c$ for $i = 3$ and $u = 4$ cannot be removed. For variable $c$, $i = 3$ and $u = 3$.

\[
\frac{\partial L_{X_3}}{\partial v_3} = [L_{X_3}] \oplus \langle r \vee (L_{X_3} \oplus 3) \rangle
\]

\[
= [0X10] \oplus \langle [0X][01X][0][0X]\rangle
\]

\[
= [0X10] \oplus \{[0X0X],[110X]\}
\]

\[
\frac{\partial L_{X_3}}{\partial v_3} = 2\{0X10\oplus[0X0X],[0X10]\oplus[110X]\}
\]

\[
= \{0,1\} \oplus f(0X10) \oplus f(0X0X) , f(0X10) \oplus f(110X)
\]

\[
= 2\{0 \oplus 0, 0 \oplus 0\} = 2\{0,0\} = 0.
\]

Therefore, $c$ can be removed for $i = 3$ and $u = 3$.

**Theorem 3**

Suppose $T_X = (L_X, b_1, b_2)$ with don't-care elements is an incomplete truth table. If the Boolean differences of $L_{X_3}$ and $L_{X_3}^{+}$, with respect to $v_2$, are zero, then $T_X$ is independent of $v_2$.

\[
\frac{\partial L_{X_3}}{\partial v_2} = 0 \Rightarrow T_X \text{ is independent of } v_2 \text{ in unit } u.
\]

(14)

**Proof for Theorem 3** is given in Appendix A.

**Binary Decision Diagrams (BDD)**

Binary Decision Diagrams (BDD) [19] are rooted directed acyclic graphs [18], which represent a canonical form of Boolean functions [20,21]. The BDD graph is first decomposed to a cutset graph, which is then simplified by the Boolean difference simplification algorithm.

**Graph-Oriented Realization**

Graph Oriented Realization (GOR) of Boolean functions is a systematic approach for the synthesis of Boolean functions [22]. In GOR, the BDD of the given Boolean function is decomposed to a new graph called a cutset graph, which is a set of all possible paths from root nodes to terminal nodes in the BDD. This cutset graph is then simplified by Logical Path Cardinal Number-Compilation (LPCN-C) rules. This step is called the LPCN-C Phase (LPCN-CP). The basic core of LPCN-CP is implemented with Boolean difference. In the simplification step, redundant nodes and cutsets are removed and a new cutset graph is obtained. The resulted graph is now ready to be employed for implementing further rules and theorems such as Mutual Merging, Minimum Span Rule, Balanced Span Rule and the Cutset Equivalence Rule. Technology mapping on the simplified graph is the last step and results in the final circuit [22].

**Definition 11**

**Cutset in Binary Decision Diagrams (BDD)**

In a BDD, there is always one path from the root node to a terminal node. This path is called a cutset [22]. There is no duplicate node in a cutset. As mentioned above, the set of all possible paths from the root node to the terminal nodes is called a cutset graph. More detail on cutsets is presented in [22].

**Example 4**

For $f = a + b_2c$, the BDD graph and cutset graph are illustrated in Figure 1.

**Definition 12**

**Cutset graph as a $T_X$**

Each cutset graph has some variables and terminators that end in either logic 1 or 0. Converting the cutset graph into a table results in a $T_X$ table, in which
the non-existent variables in each cutset are replaced by don’t-care. The corresponding $T_X$ in the above example is shown in part (d) of Figure 1.

**Definition 13**

*Boolean Difference for a Cutset*

Since a cutset graph corresponds to a $T_X$ table, each cutset also corresponds to a Boolean vector, $LX$, in the $T_X$ table. The Boolean difference for a cutset is defined as the Boolean difference for the corresponding Boolean vector, $LX$.

**Theorem 4**

In the cutset graph of a BDD, if the Boolean difference for a cutset, with respect to a variable $v_i$, is zero, then, the corresponding cutset graph is independent of the variable $v_i$. Proof for Theorem 4 is given in Appendix A.

**Example 5**

For the function given in Example 4, the step-by-step procedure of applying Theorem 4 for $b$ in $u = 4$, and $i = 2$ is given below:

$$
\frac{\partial LX4}{\partial b} = [LX4] \oplus \langle r/e/(LX4, 2) \rangle \\
= [00X] \oplus \langle [0X] \oplus [01X]/ [01X] \rangle \\
= [00X] \oplus \{[011], [010]\} \\
= \bigoplus ([00X] \oplus [011], [00X] \oplus [010]) \\
= \bigoplus \{f(00X) \oplus f(011), f(00X) \oplus f(010)\} \\
= \bigoplus \{0 \oplus 1, 0 \oplus 0\} = \bigoplus \{1, 0\} = 1.
$$

In Appendix C, Theorem 4 is applied to all inputs of the above function. The simplified cutset graph and $T_X$ table are illustrated in Figure 2.

It is seen that the logical depth (maximum number of serially-connected variables) is reduced by the
simplification procedure, which decreases the propagation delay in the corresponding circuit.

**PROPOSED ALGORITHMS AND IMPLEMENTATIONS**

The proposed algorithm on simplification of Boolean functions via Boolean differences is implemented with a combination of Perl/Java programming languages [23-25] and, finally, implemented in C++ for comparing the results on benchmarks. The core module is called the Boolean difference simplification module (BDSM). BSDM supports complete truth tables, incomplete truth tables, incomplete truth tables with don't-cares and cutset graphs. The overall block diagram for BSDM is depicted in Figure 3.

In Figure 4, the flow of the simplification procedures of Boolean functions is illustrated.

The BDD generation procedure is implemented with Java (as Java application and stand-alone executable for Win32). The BDSM modules are implemented with Perl (as Perl scripts and stand-alone executables for Win32 and Linux environments). The regular expression of Perl is directly used in the proposed algorithms. This means that each seeking string of bit patterns is first fed into a regex (regular expression) engine [17,23] and results in a regular expression string. This string is then matched with other strings by regular expression matching methods. This is the core of BSDM. Practical Extraction Reporting Language (PERL) has a comprehensive set of regular expression manipulation facilities [23]. BSDM cores are implemented in Perl using its built-in "//re" utilities. In C++ there is a problem on implementation of //re/. First, GREGA [28] was used. GREGA is about 7 times faster than the regex library in boost and about 10 times faster than the regular expression classes in ATL [25]. Most regex engines are based on NFA/NDF (non-deterministic finite state automaton) with iterative execution. This execution is often done with a big, slow switch statement [23]. Also, matching regular expressions with backreferences is an NP-complete problem [23]. Therefore, two reduced methods are employed for generation of this particular regular expression and for matching the specified regular expressions (Xgen/Regex() and Xmatch()). The experimental results show that this makes the CPU time of the final program about 2-3 times faster.

**Example 6**

In string "1010XX10", the regex generated string for the 4th bit ("X") is: "[1X][0X][1X] [1][0X][01X][1X][0X]". This string is a regular expression and is used to match with other strings. In the simplification process, for each individual Boolean vector, the simplified bit is removed. Then, the simplification procedure is continued with the updated (simplified) vectors. In the final stage, redundant elements are removed. The particular implementations of BSDM for each individual case are named by adding a suffix to BSDM. The general syntax for the resulted modules is BSDM\_\_\_\_\_\_\_\_\_\_, where \_\_\_\_\_\_\_\_\_\_ stands for TT, CSG, CFT, IFT and IDT. The sub-modules of BSDM are BSDM4CPTT, BSDM4ITT, BSDM4IDTT and BSDM4CSG. Another algorithm is also implemented for the case of SOP, where only 1-valued terms are considered. This module is also called the Boolean Difference Simplification Module for High-Term SOP (BSDM4HSOP). In the BSDM4HSOP algorithm, during the processing of a single input line, for each matched target vector, the corresponding bit is also marked as don't-care. Then, the current line is modified and the process is carried out for the remaining input vectors. The pseudo code for BSDM4CPTT is presented in Table 3.
CASE STUDIES

The proposed algorithm for Boolean function simplification is examined on a set of Boolean functions. The results of the tests are summarized in the following tables. In these tables, the following notations are used:

| Var:         | number of variables in the desired Boolean function, |
| BDD nodes:   | number of nodes in the BDD graph, |
| $H$:         | number of input nodes with HIGH value, |
| $L$:         | number of input nodes with LOW value, |
| $H+L$:       | total number of input nodes, |
| $LD$:        | logical depth for the given Boolean function (see Appendix B), |
| $ELD$:       | effective (equivalent) logical depth for the given Boolean function (see Appendix B). |

Logical Depth ($LD$) is equal to the maximum number of serially-connected variables (nodes) in the cutset graph. Logical depth represents the signal propagation delay of the synthesized circuit. Therefore, it is used as a figure of merit.

Effective Logical Depth ($ELD$) is defined here as a factor to show incomplete truth tables, $T_A$, by an equivalent complete truth table. In this case, the effective logical depth is equal to the maximum number of serially-connected variables (nodes) of the equivalent complete truth table.

Each test is verified using the Quine-McCluskey method, based on the results obtained for $H$, $L$, $H+L$ terms, and the final results of the simplification. Therefore, the "(.)" columns of Tables are the same for both the proposed algorithm and the Quine-McCluskey method. In these tables, $A(B)$ denotes after (before) simplification. The Quine-McCluskey method is an NP-complete problem [27]. However, the proposed algorithm has a fixed order of $O(n_1 n_0)$, where $n_1 \leq 2^n$ and $n_0 \leq n$ in $T_A$. Since $n_0$ is smaller than $2^n$, the proposed algorithm has less computation complexity than the Quine-McCluskey method. The results of each
test are summarized in two separate tables. The first (second) table shows the absolute values (the reduction in percent).

It is seen in Tables 4 to 7 that there is a simplification performance, but in Table 8, there is no simplification performance for XOR examples. This is mainly due to the symmetric property of these functions.

In Tables 9 to 12, there is simplification in \( L \) or \( H \) or both. In these cases, \( ELD \) is also decreased. The list of functions used for miscellaneous test cases are listed in Table 13. The experimental results of running benchmarks from MCNC and Berkeley PLA sets are presented in Table 14. All the listed benchmarks are tested with both ESPRESSO [28] and the proposed method. Both programs are compiled under Microsoft Visual C++ 6.0 and tested on a PC Windows 2000 Advanced Server with SP3 and performance tuned for application services with Intel P4 1.82GHz, 256M-DDR and a reduced set of background services. It is seen that with the proposed algorithm, the CPU time has a performance between 1.5 - 10 times better than ESPRESSO. The performance of the implemented program slowly degrades when the number of prime implicants increases. This is mainly due to the selected approach of using 2D and 3D matrices in the program. It is hoped to overcome this problem in the next versions with bitwise operations, string-based

<table>
<thead>
<tr>
<th>Table 4. Test results for AND gates.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>and 2</td>
</tr>
<tr>
<td>and 3</td>
</tr>
<tr>
<td>and 4</td>
</tr>
<tr>
<td>and 5</td>
</tr>
<tr>
<td>and 6</td>
</tr>
<tr>
<td>and 7</td>
</tr>
<tr>
<td>and 8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5. Percent of reduction after simplification in AND.</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 6. Test results for OR gates.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>or 2</td>
</tr>
<tr>
<td>or 3</td>
</tr>
<tr>
<td>or 4</td>
</tr>
<tr>
<td>or 5</td>
</tr>
<tr>
<td>or 6</td>
</tr>
<tr>
<td>or 7</td>
</tr>
<tr>
<td>or 8</td>
</tr>
</tbody>
</table>
algorithms and by converting all 3D matrices into 2D matrices.

CONCLUSIONS

In this paper, a new method for simplification of Boolean functions is presented using the well-known concept of Boolean differences and an original extension of the concept covering the Boolean vectors. This method is applicable to truth tables, BDDs, and cutset graphs. The results of the method are verified by the Quine-McCluskey method and compared to that of ESPRESSO on the MCNC and Berkeley PLA benchmarks. The experimental results show a reasonable gain in CPU time (1.5-10 times). The process of simplification with the proposed algorithms has a fixed order of \( O(2^n) \), where \( n \) is the number of variables. While in the Quine-McCluskey method, the order of calculation highly depends on prime implicants. Use of the regular expression core engine of Perl facilitates implementing the proposed method results in a compact code size, but it has some speed disadvantages. Additionally, using the regular expression core engine of C++ degrades the speed performance. Therefore, a simple regex generator and match finder (XgenRegex() and Xmatch()) are developed to speed up the implementation. The algo-

### Table 7. Simplification in OR.

<table>
<thead>
<tr>
<th>#</th>
<th>( f )</th>
<th>( %H )</th>
<th>( %L )</th>
<th>( %H + L )</th>
<th>( %CS )</th>
<th>( %LD )</th>
<th>( %ELD )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>or\textsubscript{2}</td>
<td>0</td>
<td>33.3333</td>
<td>20.0000</td>
<td>0</td>
<td>0</td>
<td>20.0000</td>
</tr>
<tr>
<td>2</td>
<td>or\textsubscript{3}</td>
<td>0</td>
<td>50</td>
<td>33.3333</td>
<td>0</td>
<td>0</td>
<td>33.3333</td>
</tr>
<tr>
<td>3</td>
<td>or\textsubscript{4}</td>
<td>0</td>
<td>60</td>
<td>42.8571</td>
<td>0</td>
<td>0</td>
<td>42.8571</td>
</tr>
<tr>
<td>4</td>
<td>or\textsubscript{5}</td>
<td>0</td>
<td>66.6667</td>
<td>33.3333</td>
<td>0</td>
<td>0</td>
<td>33.3333</td>
</tr>
<tr>
<td>5</td>
<td>or\textsubscript{6}</td>
<td>0</td>
<td>71.4286</td>
<td>55.5555</td>
<td>0</td>
<td>0</td>
<td>55.5555</td>
</tr>
<tr>
<td>6</td>
<td>or\textsubscript{7}</td>
<td>0</td>
<td>75</td>
<td>60.0000</td>
<td>0</td>
<td>0</td>
<td>60</td>
</tr>
<tr>
<td>7</td>
<td>or\textsubscript{8}</td>
<td>0</td>
<td>77.7778</td>
<td>63.3333</td>
<td>0</td>
<td>0</td>
<td>63.3333</td>
</tr>
</tbody>
</table>

### Table 8. Test results for XOR gates.

<table>
<thead>
<tr>
<th>( f )</th>
<th>Var/N#</th>
<th>( H_B/L_B )</th>
<th>( (H + L)_B )</th>
<th>( CS_B )</th>
<th>( LD_B/ELD_B )</th>
<th>( H_A/L_A )</th>
<th>( (H + L)_A )</th>
<th>( CS_A )</th>
<th>( LD_A/ELD_A )</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{xor\textsubscript{2}}</td>
<td>2/3</td>
<td>4/4</td>
<td>8</td>
<td>4</td>
<td>2/2</td>
<td>4/4</td>
<td>8</td>
<td>4</td>
<td>2/2</td>
</tr>
<tr>
<td>\textit{xor\textsubscript{3}}</td>
<td>3/7</td>
<td>12/12</td>
<td>24</td>
<td>8</td>
<td>3/3</td>
<td>12/12</td>
<td>24</td>
<td>8</td>
<td>3/3</td>
</tr>
<tr>
<td>\textit{xor\textsubscript{4}}</td>
<td>4/9</td>
<td>32/32</td>
<td>64</td>
<td>16</td>
<td>4/4</td>
<td>32/32</td>
<td>64</td>
<td>16</td>
<td>4/4</td>
</tr>
<tr>
<td>\textit{xor\textsubscript{5}}</td>
<td>5/11</td>
<td>80/80</td>
<td>160</td>
<td>32</td>
<td>5/5</td>
<td>80/80</td>
<td>160</td>
<td>32</td>
<td>5/5</td>
</tr>
<tr>
<td>\textit{xor\textsubscript{6}}</td>
<td>6/13</td>
<td>192/192</td>
<td>384</td>
<td>64</td>
<td>6/6</td>
<td>192/192</td>
<td>384</td>
<td>64</td>
<td>6/6</td>
</tr>
<tr>
<td>\textit{xor\textsubscript{7}}</td>
<td>7/15</td>
<td>448/448</td>
<td>896</td>
<td>128</td>
<td>7/7</td>
<td>448/448</td>
<td>896</td>
<td>128</td>
<td>7/7</td>
</tr>
<tr>
<td>\textit{xor\textsubscript{8}}</td>
<td>8/17</td>
<td>1024/1024</td>
<td>2048</td>
<td>256</td>
<td>8/8</td>
<td>1024/1024</td>
<td>2048</td>
<td>256</td>
<td>8/8</td>
</tr>
</tbody>
</table>

### Table 9. Test results for AND-OR-INV gates.

<table>
<thead>
<tr>
<th>( f )</th>
<th>Var/N#</th>
<th>( H_B/L_B )</th>
<th>( (H + L)_B )</th>
<th>( CS_B )</th>
<th>( LD_B/ELD_B )</th>
<th>( H_A/L_A )</th>
<th>( (H + L)_A )</th>
<th>( CS_A )</th>
<th>( LD_A/ELD_A )</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{aci\textsubscript{2x2}}</td>
<td>4/6</td>
<td>11/10</td>
<td>21</td>
<td>7</td>
<td>4/3</td>
<td>7/8</td>
<td>15</td>
<td>7</td>
<td>3/2.1429</td>
</tr>
<tr>
<td>\textit{aci\textsubscript{2x3}}</td>
<td>6/8</td>
<td>33/21</td>
<td>54</td>
<td>13</td>
<td>6/4.1388</td>
<td>15/18</td>
<td>33</td>
<td>13</td>
<td>5/2.5385</td>
</tr>
<tr>
<td>\textit{aci\textsubscript{2x4}}</td>
<td>8/10</td>
<td>74/36</td>
<td>110</td>
<td>21</td>
<td>8/5.2381</td>
<td>26/32</td>
<td>58</td>
<td>21</td>
<td>7/2.7619</td>
</tr>
<tr>
<td>\textit{aci\textsubscript{2x5}}</td>
<td>10/12</td>
<td>140/55</td>
<td>195</td>
<td>31</td>
<td>10/6.2003</td>
<td>40/50</td>
<td>90</td>
<td>31</td>
<td>9/2.9032</td>
</tr>
<tr>
<td>\textit{aci\textsubscript{2x6}}</td>
<td>12/14</td>
<td>237/78</td>
<td>315</td>
<td>43</td>
<td>12/7.3256</td>
<td>57/72</td>
<td>129</td>
<td>43</td>
<td>11/3</td>
</tr>
<tr>
<td>\textit{aci\textsubscript{2x7}}</td>
<td>14/16</td>
<td>371/105</td>
<td>476</td>
<td>57</td>
<td>14/8.3500</td>
<td>77/88</td>
<td>175</td>
<td>57</td>
<td>13/3.0702</td>
</tr>
<tr>
<td>\textit{aci\textsubscript{2x8}}</td>
<td>16/18</td>
<td>548/136</td>
<td>684</td>
<td>73</td>
<td>16/9.3000</td>
<td>100/128</td>
<td>238</td>
<td>73</td>
<td>15/3.1233</td>
</tr>
</tbody>
</table>
### Table 10. Simplification in AND-OR-INV.

<table>
<thead>
<tr>
<th>#</th>
<th>$f$</th>
<th>%H</th>
<th>%L</th>
<th>%H + L</th>
<th>%CS</th>
<th>%LD</th>
<th>%ELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>aoi$_2$x 2</td>
<td>36.3636</td>
<td>20</td>
<td>26.5714</td>
<td>0</td>
<td>25</td>
<td>28.57</td>
</tr>
<tr>
<td>2</td>
<td>aoi$_2$x 3</td>
<td>54.5455</td>
<td>14.2857</td>
<td>38.8888</td>
<td>0</td>
<td>16.0000</td>
<td>38.8873</td>
</tr>
<tr>
<td>3</td>
<td>aoi$_2$x 4</td>
<td>64.8649</td>
<td>11.1111</td>
<td>47.2727</td>
<td>0</td>
<td>12.5</td>
<td>47.2729</td>
</tr>
<tr>
<td>4</td>
<td>aoi$_2$x 5</td>
<td>71.4286</td>
<td>9.0909</td>
<td>53.8461</td>
<td>0</td>
<td>10</td>
<td>53.8464</td>
</tr>
<tr>
<td>5</td>
<td>aoi$_2$x 6</td>
<td>75.9384</td>
<td>7.6923</td>
<td>59.0476</td>
<td>0</td>
<td>8.3333</td>
<td>59.0477</td>
</tr>
<tr>
<td>6</td>
<td>aoi$_2$x 7</td>
<td>79.2453</td>
<td>6.6667</td>
<td>63.2352</td>
<td>0</td>
<td>7.1429</td>
<td>63.2351</td>
</tr>
<tr>
<td>7</td>
<td>aoi$_2$x 8</td>
<td>81.7518</td>
<td>5.8824</td>
<td>66.6666</td>
<td>0</td>
<td>6.25</td>
<td>66.6667</td>
</tr>
</tbody>
</table>

### Table 11. Test results for MISC functions.

<table>
<thead>
<tr>
<th>$f$</th>
<th>Var/N#</th>
<th>$H_B/L_B$</th>
<th>$(H + L)_B$</th>
<th>$CS_B$</th>
<th>$LD_B/ELD_B$</th>
<th>$H_A/L_A$</th>
<th>$(H + L)_A$</th>
<th>$CS_A</th>
<th>LD_A/ELD_A</th>
</tr>
</thead>
<tbody>
<tr>
<td>gfi$_{1}x8a$</td>
<td>8/14</td>
<td>91/129</td>
<td>220</td>
<td>34</td>
<td>8/6.4706</td>
<td>79/95</td>
<td>174</td>
<td>34</td>
<td>7/5.1176</td>
</tr>
<tr>
<td>gfi$_{1}x8b$</td>
<td>8/16</td>
<td>246/230</td>
<td>476</td>
<td>68</td>
<td>8/7</td>
<td>134/124</td>
<td>258</td>
<td>44</td>
<td>7/5.8636</td>
</tr>
<tr>
<td>gfi$_{1}x10$</td>
<td>10/16</td>
<td>227/289</td>
<td>516</td>
<td>66</td>
<td>10/7.8182</td>
<td>144/185</td>
<td>329</td>
<td>57</td>
<td>9/5.7179</td>
</tr>
<tr>
<td>gfi$_{1}x12$</td>
<td>12/18</td>
<td>531/673</td>
<td>1204</td>
<td>130</td>
<td>12/9.2615</td>
<td>264/390</td>
<td>654</td>
<td>100</td>
<td>10/6.54</td>
</tr>
<tr>
<td>gfi$_{1}x14$</td>
<td>14/20</td>
<td>1203/1569</td>
<td>2772</td>
<td>258</td>
<td>14/10.7442</td>
<td>485/847</td>
<td>1332</td>
<td>178</td>
<td>11/7.4831</td>
</tr>
<tr>
<td>gfi$_{1}x16a$</td>
<td>16/22</td>
<td>2975/3617</td>
<td>6292</td>
<td>514</td>
<td>16/12.2412</td>
<td>930/1875</td>
<td>2805</td>
<td>327</td>
<td>12/8.578</td>
</tr>
<tr>
<td>gfi$_{1}x16b$</td>
<td>16/26</td>
<td>2999/4305</td>
<td>7304</td>
<td>562</td>
<td>16/12.9004</td>
<td>1900/2574</td>
<td>4564</td>
<td>461</td>
<td>13/9.9002</td>
</tr>
<tr>
<td>gfi$_{1}x24a$</td>
<td>38/14</td>
<td>72759/104913</td>
<td>177672</td>
<td>9010</td>
<td>24/19.7194</td>
<td>37410/32762</td>
<td>90172</td>
<td>5406</td>
<td>23/16.68</td>
</tr>
</tbody>
</table>

### Table 12. Simplification in MISC functions.

<table>
<thead>
<tr>
<th>#</th>
<th>$f$</th>
<th>%H</th>
<th>%L</th>
<th>%H + L</th>
<th>%CS</th>
<th>%LD</th>
<th>%ELD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>gfi$_{1}x8a$</td>
<td>13.1808</td>
<td>23.3566</td>
<td>20.9500</td>
<td>0</td>
<td>12.5</td>
<td>2091</td>
</tr>
<tr>
<td>2</td>
<td>gfi$_{1}x8b$</td>
<td>45.3285</td>
<td>46.087</td>
<td>73.4355</td>
<td>35.2941</td>
<td>12.5</td>
<td>16.2343</td>
</tr>
<tr>
<td>3</td>
<td>gfi$_{1}x10$</td>
<td>36.5093</td>
<td>35.9862</td>
<td>36.2403</td>
<td>15.3654</td>
<td>10</td>
<td>26.1735</td>
</tr>
<tr>
<td>4</td>
<td>gfi$_{1}x12$</td>
<td>50.2825</td>
<td>42.0505</td>
<td>45.6810</td>
<td>23.0769</td>
<td>16.6937</td>
<td>29.3551</td>
</tr>
<tr>
<td>5</td>
<td>gfi$_{1}x14$</td>
<td>59.6841</td>
<td>45.0166</td>
<td>51.9480</td>
<td>31.0078</td>
<td>21.4286</td>
<td>30.3522</td>
</tr>
<tr>
<td>6</td>
<td>gfi$_{1}x16a$</td>
<td>65.2336</td>
<td>48.1615</td>
<td>55.4195</td>
<td>36.3813</td>
<td>25</td>
<td>29.9252</td>
</tr>
<tr>
<td>7</td>
<td>gfi$_{1}x16b$</td>
<td>33.6445</td>
<td>49.2091</td>
<td>37.5136</td>
<td>17.9715</td>
<td>6.25</td>
<td>23.8235</td>
</tr>
<tr>
<td>8</td>
<td>gfi$_{1}x24a$</td>
<td>48.5837</td>
<td>49.7088</td>
<td>49.2480</td>
<td>40</td>
<td>4.1667</td>
<td>15.4132</td>
</tr>
</tbody>
</table>

### Table 13. List of functions used as MISC test functions.

<table>
<thead>
<tr>
<th>Name</th>
<th>$f$</th>
<th># Var.</th>
</tr>
</thead>
<tbody>
<tr>
<td>gfi$_{1}x8a$</td>
<td>$a + bx + (d + e + f_{g}(g + h))y'$</td>
<td>8</td>
</tr>
<tr>
<td>gfi$_{1}x8b$</td>
<td>$a + bx + (d + e + f_{g}(g + h))y'$</td>
<td>8</td>
</tr>
<tr>
<td>gfi$_{1}x10$</td>
<td>$a + bx + (d + e + f_{g}(g + h))y' + i, j$</td>
<td>10</td>
</tr>
<tr>
<td>gfi$_{1}x12$</td>
<td>$a + bx + (d + e + f_{g}(g + h))y' + i, j + k, l$</td>
<td>12</td>
</tr>
<tr>
<td>gfi$_{1}x14$</td>
<td>$a + bx + (d + e + f_{g}(g + h))y' + i, j + k, l + m, n$</td>
<td>14</td>
</tr>
<tr>
<td>gfi$_{1}x16b$</td>
<td>$a + bx + (d + e + f_{g}(g + h))y' + i, j + k, l + m, n$</td>
<td>16</td>
</tr>
<tr>
<td>gfi$_{1}x24a$</td>
<td>$a + bx + (d + e + f_{g}(g + h))y' + i, j + k, l + m, n_{o}(o + p)$</td>
<td>24</td>
</tr>
</tbody>
</table>
Tables of Boolean Functions are implemented with Java/Perl/C++, which can be employed for a two-level sum of product realization (PLA). Further research will be directed towards multi-level realization, multi-valued logic synthesis and code optimization for more speed improvement.

ACKNOWLEDGMENT

The authors wish to thank Mr. Mohammad Reza Tayfeh and Mr. Hassan Ghelipour Khatir for their great help. The authors also wish to thank anonymous reviewers for their comments and suggestions.

REFERENCES


**APPENDIX A**

**Proof of Theorem 1**

Rewriting $L_{2n}$ and $L_{2n}^{C_2}$:

\[ L_{2n} = (v_1, v_2, \ldots, v_{2n})_{A} \]

\[ L_{2n}^{C_2} = (v_1, v_2, \ldots, v_{2n})_{A} \]

Since $T_{2} = \{L_{2n}, L_{2n}^{C_2}\}$ is a complete truth table (with $2^{2n}$ elements), both $L_{2n}$ and $L_{2n}^{C_2}$ belong to $T_{2}$.

Rearranging $L_{2n}$ and $L_{2n}^{C_2}$ : $L_{2n} = (L_{2n_1}, v_{2n_2}) L_{2n}^{C_2} = (L_{2n_1}, v_{2n_2})$ where $L_{2n} = (v_1, v_2, \ldots, v_{2n-1}, v_{2n})_{A}$.

Since $\frac{d}{dx} f(L_{2n}) = 0$, $f(L_{2n}) = f(L_{2n_1}) f(L_{2n_1}, v_{2n}) = f(L_{2n_1}, v_{2n})$. By rewriting the Shannon expansion theorem for $v_{2n}$ in unit $u$, the following is obtained:

\[ f = v_{2n} f(L_{2n_1}, 1)_{u} + v_{2n} f(L_{2n_1}, 0)_{u} = (v_{2n} + v_{2n}) f(L_{2n_1}, 1)_{u} = f(L_{2n_1}, 1)_{u} \]

This shows that $T_{2}$ in row $u$ is independent of $v_{2n}$ and the proof is complete.

**Proof of Theorem 2**

Since both $L_{2n}$ and $L_{2n}^{C_2}$ are in $T_{2}$, the proof is similar to that of Theorem 1.

**Proof of Theorem 3**

A truth table $T_{X}$, with don't-care components is given. $L_{X}$ is a Boolean vector of binary values 1, 0 and X. The value of $j$th bit in $L_{X}$ is denoted by $t_{Xj} \in \{0,1\}$.

Don't-care components can be replaced by either 1 or 0, which are used in the process of simplification. The set of $(/re/(L_{X_1}, J))$ has $k$ Boolean vectors \{(1,2,\ldots, k)\}.

The combination of $L_{2n}$ with $L_{2n}^{C_2}$ gives a set of $k$ pairs \{(0,1), (0,2), \ldots, (0,k)\}. There are two possible cases:

1. When at least one of the Boolean differences of $L_{2n}$ and $L_{2n}^{C_2}$ is not zero
2. When all Boolean vector differences of $L_{2n}$ and $L_{2n}^{C_2}$ are zero

In case 1, the Boolean vector difference for pair $(0, p)$ is assumed to be 1 and all other Boolean vector differences $(0, j), j = 1, 2, \ldots, k$, $j \neq p$ are zero. The corresponding situation is illustrated in Figure A1. In Figure A2 the sub-vectors are equal.

$L_{0u} = L_{Bu}$ and $L_{0u} = L_{Bu}$.

The Shannon expansion for $\nu_{2}$ is:

\[ f = \nu_{2} f(\nu_{2}) + \nu_{2} f(\nu_{2}) \]

\[ f(\nu_{2}) \text{ and } f(\nu_{2}) \text{ have different values, e.g. either (0,1) or (1,0)} \]  

\[ f = \nu_{2} f(\nu_{2}) + \nu_{2} f(\nu_{2}) \]  \hspace{1cm} (A1)

![Figure A1. Set of matches for regular expression bit patterns.](image-url)
This shows that \( f \) is not independent of \( \nu_j \) and, therefore, \( \nu_j \) is not removable. In case 1, all Boolean vector differences are zero. This situation is illustrated in Figure A3, where all sub-vectors are equal:

\[
L_{0a} = L_{1a}, \quad L_{2a} = L_{0b}, \quad w = 1, 2, \ldots, k.
\]

The Shannon theorem implies that the functions \( f \) and \( T_X \) are independent of \( \nu_j \), and the proof is complete.

**Proof of Theorem 4**

Each cutset graph corresponds to a \( T_X \) table and each cutset corresponds to a Boolean vector in the \( T_X \) table. Therefore, the proof of Theorem 3 is valid for Theorem 4.

**APPENDIX B**

**Logical Depth and Effective Logical Depth**

For a given Boolean function \( f = f(\nu_1, \nu_2, \ldots, \nu_i, \nu_{i+1}, \ldots, \nu_n) \), the corresponding complete truth table,

\[
<table>
<thead>
<tr>
<th>U</th>
<th>\nu_1, \nu_2, \nu_3, \ldots, \nu_{i-1}, \nu_i, \nu_{i+1}, \ldots, \nu_n</th>
<th>\quad</th>
<th>f</th>
</tr>
</thead>
</table>
| 0 | \nu_j | \nu_j | 0 | 1
| 1 | \nu_j | \nu_j | 0 | 1
| 2 | \nu_j | \nu_j | 0 | 1
| 3 | \nu_j | \nu_j | 0 | 1

has \( 2^l \times (n + 1) \) elements, where \( 2^l \times n \) elements are variables and \( 2^l \times 1 \) elements are function values. In the complete truth table each row has \( n \) variables. Therefore, the logical depth (i.e., the path of serially-connected variables) is \( n \). If the truth table is assumed as a rectangle with \( 2^n \times 1 \) as its height, then its width is \( n \).

Incompletely truth table without don’t-care is \( m^* \times (n^* + 1) \), where \( m^* \leq 2^n \) and \( n^* \leq n \). An incomplete truth table without don’t-cares is a complete truth table with at least one row deleted.

In the incomplete truth tables without don’t-cares, each row has \( n \) variables. Therefore, the logical depth (i.e., the path of serially-connected variables) is \( n \).

An incomplete truth table with don’t-cares has \( m^* \times n^* \), where \( m^* \leq 2^n \). It is a truth table with at least one variable, which is dropped in one or more rows.

Therefore, the number of serially-connected variables in rows is not constant and varies from one row to the next. The sum of all elements in this case (truth table with don’t-cares) is denoted by \( PL \).

Effective Logical Depth (\( ELD \)) is defined here as:

\[
ELD = \frac{PL}{m^*}.
\]

Figure A4 illustrates the relation of \( LD \) and \( ELD \) in truth tables. As seen in Figure A4, \( ELD \) times the number of rows makes a rectangle with a height of \( m^* \) and a width of \( ELD \).

Generally, in the process of simplification for a truth table, the simplified truth table has don’t-care components. In this case, \( ELD \) is also defined. In the case of an incomplete truth table with don’t-care components, \( ELD \) is used instead of \( LD \).

\( LD \) is equivalent to the number of serially-connected variables. Therefore, in the process of synthesis, this is equivalent to the number of serially-connected transistors [22]. The less \( LD \), the less is the propagation delay time.

**APPENDIX C**

**Complete Steps of Applying Boolean Difference**

For \( b \) in \( U = 3, i = 2 \):

\[
\frac{\partial L X_3}{\partial b} = [L X_3] \oplus ([010] \oplus ([00X]) [00X])
\]

\[
= [010] \oplus ([00X]) [00X]
\]

\[
= 010 \oplus [[00X]]
\]

\[
= 010 \oplus [00X]
\]

\[
= 1/([010] \oplus [00X]).
\]
\[
\frac{\partial L_{X_2}}{\partial b} = \bigoplus \{ f(010) \oplus f(00X) \} \\
= \bigoplus \{ 0 \oplus 0 \} = \bigoplus \{ 0 \} = 0.
\]

For \( b \) in \( U = 2, i = 1 \):

\[
\frac{\partial L_{X_2}}{\partial a} = [L_{X_2}] \oplus \langle /re/(L_{X_2}, 1) \rangle \\
= [010] \oplus \langle [1][1X][1X]/ \rangle \\
= [011] \oplus \{ [11X] \} \\
= \bigoplus \{ [011] \oplus [11X] \} \\
= \bigoplus \{ f(011) \oplus f(11X) \} \\
= \bigoplus \{ 1 \oplus 1 \} = \bigoplus \{ 0 \} = 0.
\]

For \( b \) in \( U = 2, i = 2 \):

\[
\frac{\partial L_{X_2}}{\partial b} = [L_{X_2}] \oplus \langle /re/(L_{X_2}, 2) \rangle \\
= [011] \oplus \langle [0X][0][1X]/ \rangle \\
= [011] \oplus \{ [00X] \} \\
= \bigoplus \{ [011] \oplus [00X] \} \\
= \bigoplus \{ f(011) \oplus f(00X) \} \\
= \bigoplus \{ 1 \oplus 0 \} = \bigoplus \{ 1 \} = 1.
\]

For \( b \) in \( U = 2, i = 3 \):

\[
\frac{\partial L_{X_2}}{\partial c} = [L_{X_2}] \oplus \langle /re/(L_{X_2}, 3) \rangle \\
= [011] \oplus \langle [0X][1X][c]/ \rangle \\
= [011] \oplus \{ [010] \} \\
= \bigoplus \{ [011] \oplus [010] \} \\
= \bigoplus \{ f(011) \oplus f(010) \} \\
= \bigoplus \{ 1 \oplus 0 \} = \bigoplus \{ 1 \} = 1.
\]
\[
\frac{\partial L_{X_3}}{\partial a} = [L_{X_3}] \oplus \langle \rho e/(L_{X_3},1) \rangle = [010] \oplus \langle [1][X][0X] \rangle = [011] \oplus \{(1X)\}.
\]

For \( b \) in \( U = 3, i = 1 \):
\[
\frac{\partial L_{X_3}}{\partial a} = 1 \oplus \{(011) \oplus [011]\}
= 1 \oplus \{f(011) \oplus f(1X)\}
= 1 \oplus \{0 \oplus 1\} = 1 \oplus \{1\} = 1.
\]

For \( b \) in \( U = 4, i = 2 \):
\[
\frac{\partial L_{X_3}}{\partial b} = [L_{X_3}] \oplus \langle \rho e/(L_{X_3},2) \rangle = [010] \oplus \langle [0X][0X][01X] \rangle = [010] \oplus \{(00X)\}
= 1 \oplus \{(010) \oplus [00X]\}
= 1 \oplus \{f(010) \oplus f(00X)\}
= 1 \oplus \{0 \oplus 0\} = 1 \oplus \{0\} = 0.
\]

For \( b \) in \( U = 3, i = 3 \):
\[
\frac{\partial L_{X_3}}{\partial c} = [L_{X_3}] \oplus \langle \rho e/(L_{X_3},3) \rangle = [010] \oplus \langle [0X][1X][1] \rangle = [010] \oplus \{(011)\}
\]

For \( b \) in \( U = 4, i = 2 \):
\[
\frac{\partial L_{X_4}}{\partial b} = [L_{X_4}] \oplus \langle \rho e/(L_{X_4},2) \rangle = [001X] \oplus \langle [00X][01X] \rangle = [00X] \oplus \{(1X)\}
= 1 \oplus \{(00X) \oplus f(1X)\}
= 1 \oplus \{0 \oplus 1\} = 1 \oplus \{1\} = 1.
\]

For \( b \) in \( U = 3, i = 1 \):
\[
\frac{\partial L_{X_4}}{\partial b} = [L_{X_4}] \oplus \langle \rho e/(L_{X_4},1) \rangle = [00X] \oplus \langle [1][0X][01X] \rangle = [00X] \oplus \{(1X)\}
= 1 \oplus \{(00X) \oplus f(1X)\}
= 1 \oplus \{0 \oplus 1\} = 1 \oplus \{1\} = 1.
\]

For \( b \) in \( U = 4, i = 2 \):
\[
\frac{\partial L_{X_4}}{\partial b} = [L_{X_4}] \oplus \langle \rho e/(L_{X_4},2) \rangle = [001X] \oplus \langle [00X][01X] \rangle = [00X] \oplus \{(1X)\}
= 1 \oplus \{(00X) \oplus f(1X)\}
= 1 \oplus \{0 \oplus 1\} = 1 \oplus \{1\} = 1.
\]