# PIECEWISE LINERIZATION OF LOGIC FUNCTIONS*

*Ilya Levin[1], Osnat Keren[2], George Kolotov[1], Mark Karpovsky[3]*

[1]Tel Aviv University, Ramat Aviv, Tel Aviv 69978, Israel, i.levin@computer.org
[2]Bar Ilan University, Ramat Gan, Ramat Gan 52900, Israel, kereno@eng.biu.ac.il
[3]Boston University, 8 Saint Mary's Street, Boston, MA 02215, markkar@bu.edu

**ABSTRACT**

The paper deals with a problem of linearization of multi-output logic functions. Specifically, the case is discussed, when the functions have a large number of variables and cannot be efficiently linearized by using known techniques. For solution of the problem a so-called piecewise linearization is proposed.

The piecewise linearization comprises decomposition of an initial multi-output function into a network of components, followed by independent linearization of the components. The decomposition is based on the theory of $D$-polynomials described in the paper. The resulting piecewise linearized network is directly mapable onto a special type of a binary graph called Parallel Multi Terminal BDD.

An efficient heuristic algorithm for the piecewise linearization is provided. The presented benchmarks results demonstrate high efficiency of the proposed method in comparison with known linearization approaches. The results also show that integrating linearization techniques with the described decomposition, thus obtaining the piecewise linearization, are a very promising both from practical and from the theoretical points of view.

## 1. INTRODUCTION

The use of the Linear Decomposition as tool for optimization in logic synthesis, particularly, in Binary Decision Diagrams (BDDs) optimization is promising direction of research in the field of logic design [1]. BDDs are a standard part of many CAD systems in logic design, signal processing, and other areas where efficient, in terms of space and time, manipulation of BDD representation for a given function is usually estimated by the number of non-terminal nodes in the BDD. The size of a BDD is very sensitive to the order of variables, ranging from the polynomial to the exponential complexity for the same function for different orders of variables. Therefore, the majority of approaches to the BDD size reduction are related to development of efficient algorithms for reordering of variables see for example, [2], [3]. Linearly transformed BDDs are defined by allowing linear combinations of the variables [4].

The known linearization techniques use representation of logic functions in a form of the truth table. The present paper deals with multi-output logic functions defined by their implicant table. Moreover, we deal where the case when the implicant table is sparse (contains a large number of "don't cares"), and the function is defined by the corresponding implicant table depends on the large number of variables. In such a case the use of known linearization techniques becomes inefficient or even inapplicable due to the great size of the corresponding truth table. The combination of the linearization and decomposition was used for implementation of planar BDDs [5] and for linearization of functions defined in sum-of-product form [6]. The present paper develops the idea of piecewise linearization which is a functional decomposition followed by a linearization procedure. The paper proposes an algorithm for constructing the piecewise linear implementation of an arbitrary logic function defined by its sum-of-products (SOP).

The theoretical foundation of our decomposition method is the algebra of $D$-polynomials [7]. The result of the decomposition is a direct mapping of the logic function onto a special type of a binary graph called Parallel Multi Terminal BDD (PMTBDD) that consists of the specifically connected component MTBDD.

The decomposition approach described in this paper aims at practical implementations of logic functions as specific VLSI structures. The main purposes of the proposed technique are to minimize the size of the resulting VLSI implementation by representing the initial multi-output function as a network of component MTBDDs. Specifically, the above task, named piecewise linearization, can be stated as follows:

Given a minimized sum-of-product representation of a multiple-output logic function, construct a network of component MTBDDs of the minimal size by applying linearization of components.

The paper is organized as follows. Section 2 describes the general structure of PMTBDDs. Section 3 introduces the notion of $D$-polynomials. Section 4 describes the algorithm of decomposition, followed by the linear transformation of components. Experimental results and the corresponding discussion are provided in Section 6. Conclusions are presented in Section 5.

* The work was supported by BSF under Grant 2002259

## 2. PARALLEL MULTI TERMINAL BDDS

The linear decomposition of multi-output functions of large number of input variables, defined by a large set of cubes of a high order, is a significantly hard problem due to its complexity.

Decomposing of an initial function into a network of components followed by their independent linearization can assist in overcoming the problem of complexity for the above cases.

The proposed decomposition approach is based on a new concept of parallel multi-terminal binary decision diagram (PMTBDD) comprising component MTBDDs.

The PMTBDD is constructed by combining component MTBDDs using the parallel and series operations:

1) Series connection: replacing one terminal node of an MTBDD with another MTBDD.
2) Parallel connection: connecting roots of two or more MTBDDs.

Introducing the PMTBDD opens a way for handling multi-output logic functions of a large number of variables, defined by their SOP (implicant table) with a large number of cubes of a high order**.**

## 3. $D$-POLYNOMIALS

This section introduces a notion of $D$-polynomial which provides a convenient mathematical model for the manipulation of MTBDD. We show that the parallel and series connections of the MTBDD can be modeled as a product and substitution of $D$-polynomials representing those MTBDDs.

### 3.1. Representation of Logic Functions

Consider an n-input, m-output completely specified Boolean function $F: X^n \to Z^m$, where $X \in \{0, 1, *\}$ and $Z \in \{0, 1\}$. Let F be initially represented in minimized (prime and irredundant) sum-of-product (SOP) form, where each output $Z_i$ is written as a logical sum (OR) of product terms (implicants):

$$Z_i = \sum_{j \in I(i)} \alpha_j \qquad (1)$$

$I(i)$ denotes an index set of implicants associated with the output $Z_i$. Implicants can be shared between different outputs. Notice that $\alpha$s are functions of input variables $(x_1, x_2, \ldots, x_n)$. We will also refer to them as the $\alpha$-functions.

Let $Y_i$ be the name associated with the output $Z_i$. One can also think of $Y_i$ as an operator which has to be performed when the corresponding output $Z_i$ evaluates to 1. $Y_0$ will denote a dummy output function (or an empty operator which does not produce any output).

**Definition 1.** $D$-polynomial is a polynomial defined over a set of operators $Y_i$

$$D = \sum_i Z_i Y_i + \alpha_0 Y_0 \qquad (2)$$

while the coefficients $Z_i$ satisfying the conditions:

a) $\bigcup_i Z_i \vee \alpha_0 = 1$ (completeness),

b) $Z_i \cdot Z_j = 0, \forall i \neq j$ (orthogonality).

Taking (1) into account, we have:

$$D = \sum_i \left( \sum_{j \in I(i)} \alpha_j \right) Y_i + \alpha_0 Y_0 = \sum_{j \in I(i)} \alpha_{ij} Y_i + \alpha_0 Y_0$$

where $\alpha_{ij}$ denotes $j$ th -implicant of function $Z_i$.

In this work we are interested in a class of $D$-polynomials defined over a subset of variables $Y_i$ whose coefficients are implicants of a logic function. Such $D$-polynomials are used to represent logic functions. Coefficients $\alpha_i$ are defined explicitly as the corresponding product terms of the function, while $\alpha_0$ is defined implicitly as a complement of $\bigcup \alpha_i$ (to satisfy the completeness condition).

$D$-binomial is a special case of $D$-polynomial, with exactly two disjoint (orthogonal) implicants, $D = \alpha_1 Y_1 + \alpha_0 Y_0$.

We distinguish between $\alpha$-functions belonging to different $D$-polynomials by labeling them with a super-script index associated with the corresponding polynomial; $\alpha_i^k$ will indicate that implicant $\alpha_i$ is associated with the $D$-polynomial $D_k$. The same implicant can be associated with different polynomials, so that $\alpha_i^k = \alpha_i^j$ for arbitrary values of $k$ and $j$.

**Example 1.** The following is a $D$-polynomial:

$D_1 = x_1 Y_1 + \bar{x}_1 \bar{x}_2 Y_2 + \alpha_0^1 Y_0$

Here, $\alpha_1^1 = x_1$, $\alpha_2^1 = \bar{x}_1 \bar{x}_2$ and $\alpha_0^1 = \bar{x}_1 x_2$. The corresponding MTBDD may be constructed in straightforward way and can be achieved by repeatedly applying the Shannon expansion to $D_1$. Notice that the simplicity of the above transformation is based of the following specific property of the $D_1$: for each application of the Shannon expansion, at least one input variable is present in all the implicants.

Conceptually, a $D$-polynomial $D_i$ can be interpreted as follows. If $\alpha_1^i$ evaluates to 1, then $D_i = Y_j$. If all of the explicit functions $\alpha_1^i$ are equal to 0, then $D_i = Y_0$ which means that no output is produced (or an empty operator is to be performed).

Let us define a product of two $D$-polynomials.

**Definition 2**. Let $D_1 = \sum_{i \in I(i)} \alpha_{ij}^1 Y_i + \alpha_0^1 Y_0$, and $D_2 = \sum_{k \in I(k)} \alpha_{kj}^2 Y_k + \alpha_0^2 Y_0$. The product of $D_1$ and $D_2$, denoted as $D_1 \circ D_2$.is defined by: $D_1 \circ D_2 = \sum \left( \alpha_{ij}^1 \cdot \alpha_{kl}^2 \right) \{ Y_i \circ Y_k \}$,

over each pair of terms from $D_1$ and $D_2$, including the implicit terms $\alpha_{\boldsymbol{0}}^{\boldsymbol{1}}Y_{\boldsymbol{0}}$ and $\alpha_0^2Y_0$. Here $\alpha_{ij}^{\boldsymbol{1}}\cdot\alpha_{kl}^{\boldsymbol{2}}$ is a logic product (AND) of the corresponding $\alpha$-functions and $Y_i\circ Y_j$ is a combination of the respective operators. In other words: when $\alpha_{ij}^{\boldsymbol{1}}\cdot\alpha_{kl}^{\boldsymbol{2}}$ evaluates to 1, both $Y_i$ and $Y_k$ are computed concurrently.

**Lemma 1.** An arbitrary $D$-polynomial $D_i$ can always be represented as follows:

$$D_i=\sum_j Z_j^iY_j+\alpha_{\boldsymbol{0}}^iY_{\boldsymbol{0}}=\prod_j\left(Z_j^iY_j+\alpha_{i\boldsymbol{0}}^jY_{\boldsymbol{0}}\right) \quad (3)$$

where $\alpha_{\boldsymbol{0}}^j=\overline{Z_{\boldsymbol{0}}^i}$, and $\prod_j\alpha_{i0}^j=\alpha_0^i$.

**Proof**. We will demonstrate that the product of $\left(Z_j^iY_j+\alpha_{i\boldsymbol{0}}^jY_{\boldsymbol{0}}\right)$ is equal to the $D$-polynomial with the same coefficients. First, notice that by definition all explicit functions are pairwise orthogonal, so that $Z_j^iY_j\cdot Z_k^iY_k=\boldsymbol{0}$, for $j\neq k$. Furthermore, orthogonality of the functions and the completeness condition $\alpha_{i\boldsymbol{0}}^j=\overline{\alpha_j^i}$ that must be satisfied by each $D$-binomial imply that $Z_j^i\subset\alpha_{i\boldsymbol{0}}^k$, , for $k\neq j$. Hence $Z_j^i\cdot\alpha_{i\boldsymbol{0}}^k=Z_j^i$. Also, notice that the concatenation $\left\{Y_j\circ Y_{\boldsymbol{0}}\right\}$ means that both operators $Y_j$ and $Y_0$ need to be performed simultaneously. Since $Y_0$ is a dummy operator, only $Y_j$ has to be computed; hence $\left\{Y_j\circ Y_{\boldsymbol{0}}\right\}=Y_j$. Finally, orthogonality and completeness conditions of $\alpha$-functions imply that $\bigcup_j\alpha_{i\boldsymbol{0}}^j=\boldsymbol{1}$, $\alpha_{\boldsymbol{0}}^i\subset\alpha_{i\boldsymbol{0}}^j$, $\alpha_{\boldsymbol{0}}^i\subset\alpha_{i\boldsymbol{0}}^j$ & $\alpha_{i\boldsymbol{0}}^k,\ldots,$ $\alpha_{\boldsymbol{0}}^i\subset\prod_j\alpha_{i\boldsymbol{0}}^j$, so that $\prod_j\alpha_{i\boldsymbol{0}}^j=\alpha_{\boldsymbol{0}}^i$,

Therefore, the subsequent multiplication of the consecutive terms of expression (3) yields

$$\left(Z_1^i\cdot Z_2^i\left\{Y_1\circ Y_2\right\}+Z_1^i\cdot\alpha_{i0}^2\left\{Y_1\circ Y_0\right\}+Z_2^i\cdot\alpha_0^i\left\{Y_2\circ Y_0\right\}+\alpha_{i0}^1\cdot\alpha_{i0}^2\left\{Y_0\circ Y_0\right\}\right)$$

$$\prod_{j=1}^{m-2}\left(\alpha_j^iY_j+\alpha_{i0}^jY_0\right)=$$

$$=\left(Z_{\boldsymbol{1}}^iY_{\boldsymbol{1}}+Z_{\boldsymbol{2}}^iY_{\boldsymbol{2}}+\alpha_{i\boldsymbol{0}}^{\boldsymbol{1}}\cdot\alpha_{i\boldsymbol{0}}^{\boldsymbol{2}}Y_{\boldsymbol{0}}\right)\prod_{j=\boldsymbol{1}}^{m-2}\left(Z_j^iY_j+\alpha_{i\boldsymbol{0}}^jY_{\boldsymbol{0}}\right)=\ldots$$

$$=\left(Z_{\boldsymbol{1}}^iY_{\boldsymbol{1}}+Z_{\boldsymbol{2}}^iY_{\boldsymbol{2}}+\ldots+Z_m^iY_m+\alpha_{i\boldsymbol{0}}^{\boldsymbol{1}}\cdot\alpha_{i\boldsymbol{0}}^{\boldsymbol{2}}\cdot\ldots\cdot\alpha_{i\boldsymbol{0}}^mY_{\boldsymbol{0}}\right)=$$

$$=\left(Z_{\boldsymbol{1}}^iY_{\boldsymbol{1}}+Z_{\boldsymbol{2}}^iY_{\boldsymbol{2}}+\ldots+Z_m^iY_m+\alpha_{\boldsymbol{0}}^iY_{\boldsymbol{0}}\right)=\sum_jZ_j^iY_j+\alpha_{\boldsymbol{0}}^iY_{\boldsymbol{0}}.$$

QED

**Theorem 1.** An arbitrary $D$-polynomial $D_i$ can always be represented as a product of $D$-binomials:

$$D_i=\sum_j Z_j^iY_i+\alpha_0^iY_0=\prod_{j\boldsymbol{I}(i)}\left(\alpha_{ij}^iY_i+\alpha_{i0}^iY_0\right), \quad (4)$$

where $\alpha_0^j=\overline{\alpha_0^i}$, and $\prod_j\alpha_{i0}^j=\alpha_0^i$.

**Proof**. Let $D_m=\left(\alpha_{1j}^mY_1+\alpha_{j0}^mY_0\right)\circ\left(\alpha_{1k}^mY_1+\alpha_{k0}^mY_0\right)$.

After performing the multiplication we have:

$$D_m=\alpha_{1j}^m\cdot\alpha_{1k}^m\left\{Y_1\circ Y_1\right\}+\alpha_{1j}^m\cdot\alpha_{k0}^m\left\{Y_1\circ Y_0\right\}+\alpha_0^m\cdot\alpha_{1k}^m\left\{Y_0\circ Y_1\right\}+\alpha_{j0}^m\cdot\alpha_{k0}^mY_0=$$

$$\left(\alpha_{1j}^m\cdot\alpha_{1k}^m+\alpha_{1j}^m\cdot\alpha_{k0}^m+\alpha_{j0}^m\cdot\alpha_{1k}^m\right)Y_1+\alpha_{j0}^m\cdot\alpha_{k0}^mY_0=$$

$$\left(\alpha_{1j}^m\cdot\alpha_{1k}^m+\alpha_{1j}^m\cdot\overline{\alpha_{1k}^m}+\overline{\alpha_{1j}^m}\cdot\alpha_{1k}^m\right)Y_1+\alpha_{j0}^m\cdot\alpha_{k0}^mY_0=\left(\alpha_{1j}^m+\alpha_{1k}^m\right)Y_1+\alpha_0^mY_0$$

Based on this, every logic function $Z_j^i$ of arbitrary $D$-polynomial $D_i$ may be presented as a product of binomials as follows:

$$Z_j^i=\sum_{I(i)}\alpha_{ij}^iY_i+\alpha_0^iY_0=\prod_{I(i)}\left(\alpha_{ij}^iY_i+\alpha_{i0}^iY_0\right) \quad (5)$$

Substituting (5) into (3) results in (4). QED

An important conclusion from this theorem is that the product of $D$-polynomials can be always presented as a product of the corresponding terminal binomials. Subsequently, the terminal binomials can be multiplied to obtain higher level $D$-polynomials. Obviously, there are several ways to group terminal binomials to form a $D$-polynomial. Different grouping of terminal binomials yields different PMTBDD's, resulting in different implementations of Boolean functions. This fact forms the basis of our decomposition approach.

### 3.2 Decomposition of $D$-polynomials

We construct an MTBDD corresponding to a system of $D$-polynomials. Different groupings of the terminal binomials lead to different implementations of the logic function represented by this system. It defines a way to decompose the logic function by manipulating the system of $D$-polynomials representing the function.

The starting point of our method is the specification of a logic function in the form of an implicant table.

An initial system of $D$-polynomials can be easily derived from the implicant table by associating a single $D$-binomial with each product term of the table. The function can then be represented as a product of the $D$-binomials.

**Theorem 2**. A multiple-output Boolean function can always be implemented as a product of $D$-polynomials.

**Proof**. Represent the implicant table of the logic function as a product of $D$-binomials and multiply the orthogonal binomials to create the constituent $D$-polynomials.

**Conclusion**: MTBDD corresponding to such an implementation can be realized as a parallel connection of subtrees corresponding to the individual $D$-polynomials.

## 4. PARALLEL DECOMPOSITION

The aim of the proposed parallel decomposition is creating the PMTBDD for an arbitrary multi-output function. The proposed decomposition algorithm is based on partition of the set of product terms representing the ON-set of the function into a set of logic blocks. It is followed by a hierarchical

decomposition of blocks into a common header and a set of block fragments.

This is accomplished by extracting a set of common factors (so-called prefixes) from the subset of product terms of the original implicant table.

**Definition 4**: A product term or a part of the product term, is called a *prefix*.

A set of all prefixes defines a block *header*. A subset of product terms with a common prefix is called a *block*. A set of all blocks is referred to as a *block set*.

**Definition 5:** The set of terms including the given prefix precisely is called its *family*.

The remaining set of product terms, not included in the *block set*, is called a *remainder*.

**Definition 6:** The set of all the rows of the implicant table that do not belong to any of the families is called the *remainder* of the current stage.

A set of product terms obtained by extracting a common prefix from all the members of the block will form a block fragment or *tail*.

**Definition 7:** The prefix's family, after all the prefix's variables are set to "don't care", is called its *tail*.

Header is a fragment (subset of rows and columns) of the implicant table composed of the prefix variables. The header is selected in such a way as to provide minimization of the resulting PMTBDD. We propose to select the header by taking into account the following underlying principle. 1) increase the percentage of "non-don't care" cells (density) of the corresponding fragment the implicant table. 2) ensure the efficiency linearization of the corresponding block.

By construction, the block header is a logic function whose ON-set is a superset of the ON-sets of the logic functions associated with the individual blocks. It will be implemented as an MTBDD whose internal nodes are associated with the prefix variables. The terminal nodes of the tree represent the block fragments, each to be implemented as a separate MTBDD.

The target of the proposed decomposition algorithm is to minimize the total size of the diagram. The algorithm divides the initial function into a block $B_s$ and a remainder $R_s$. The block is a sum of products of simpler sub-functions with prefix terms. The group of prefixes $p_{s,i}$ – the dense fragment chosen for the BDD implementation – forms the header of the block and is, indeed, implemented as a MTBDD, with the sub-functions playing the role of the terminals. Each of the sub-functions and the remainder function $R_s$ can be repeatedly decomposed in the same way, until no further decomposition is possible.

An important feature of the proposed method is its ability to benefit from any other optimization method the user may wish to employ – Sifting, $K$-Procedure, etc [8, 1]. These will be applied to the component MTBDDs, and further reduce the total diagram's size. PMTBDD with linearized blocks is denoted LPMTBDD.

### 4.1 Decomposition Algorithm

The following algorithm details the flow of a particular iteration.

L = Empty list of pairs {Prefix, Tail}
Let R = Set of prodict terms, Y = vector of integer outputs.
Let B = Basic Prefix, $T_F$ = its family, $T_T$=$T_F$\B = its tail
L = [L, (B, $T_T$)]
$T_U$ = R \ ($T_F$ U $T_R$)
C = {t: t∈R and t·B≡0}
$T_R$ = {t: t∈R\(C ∪ $T_F$)}
C = COMMON_PARTS(C)
While |C| > 0,
  Let S = Secondary Prefix
  $T_F$ = its family
  $T_T$ = $T_F$\S = its tail
  L = [L, (S, $T_T$)]
  $T_O$ = {t: t∈C and t·S≡0}
  $T_{NO}$ = {t: t∈C\$T_O$}
  C = $T_O$
  $T_R$ = $T_R$ ∪ $T_{NO}$
End While.
Classify and enumerate the tails in L
Construct the Block's BDD from L
$L_T$ = {l: l∈L, Tail(l) is trivial}
$L_{NT}$ = {l: l∈L\$L_T$}
If the remainder is trivial,
  $L_T$ = [$L_T$, Remainder]
Else
  $L_{NT}$ = [$L_{NT}$, Remainder]
End If
For all the list elements in $L_T$
  Implement the tail
End For
For all the list elements in $L_{NT}$
  Recursively call this procedure
End For

The main part of a particular iteration consists of choosing the prefixes for the block. The prefixes are chosen one by one. Each time a prefix is chosen, all the prefixes not orthogonal to it and belonging to different $Z$-functions are moved to the remainder. Non-orthogonal prefixes belonging to the same function as the prefix in the block are included in the block. This continues until no more suitable prefixes are present. Thereafter the iteration proceeds to enumerate the tails and constructs the block's MTBDD. The non-trivial tails and remainder serve as inputs to the next iterations. The trivial tails are implemented immediately as separate conjunction BDDs and do not require additional iterations.

**Example 2**: The imlicant table used in the example is presented in Table 1.

Table 1: The imlicant table for Example 2

| # | X0 | X1 | X2 | X3 | X4 | F0 | F1 | F2 | F3 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | – | 0 | – | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | – | 1 | – | 0 | 1 | 1 | 0 |
| 2 | – | – | 1 | – | 0 | 0 | 0 | 1 | 1 |
| 3 | – | – | – | – | 1 | 0 | 0 | 0 | 1 |
| 4 | 1 | 0 | – | 1 | – | 1 | 1 | 0 | 0 |

BDD, LTBDD, PMTBDD and LPMTBDD for the example are presented in Figures 1-4 correspondingly.

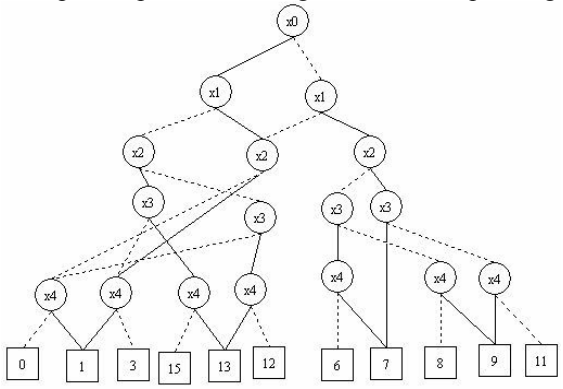


Figure 1. Straightforward implementation of the MTBDD for the Example 2.

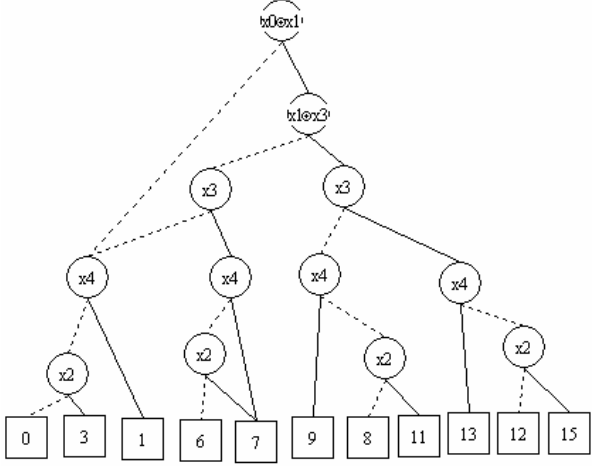


Figure 2. Linearly Transformed MTBDD for the Example 2

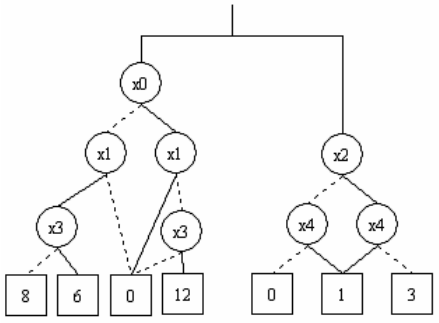


Figure 3. PMTBDD corresponding to the Example 2.

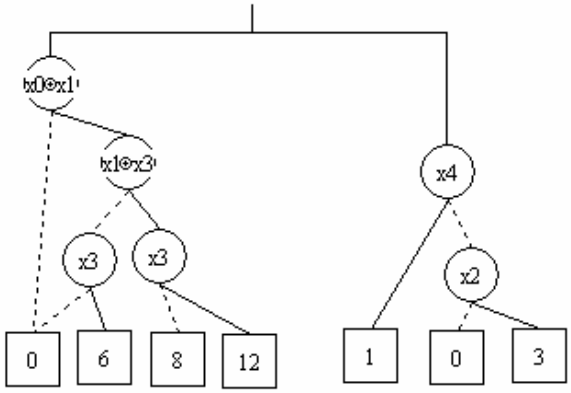


Figure 4. Linearly Transformed PMTBDD for the Example 2.

Terminal nodes of MTBDDs in Figures 1-4 are marked by decimal numbers of corresponding outputs. A standard implementation of the exemplary MTBDD, as an ordered MTBDD, is presented in Figure 1a. Figure 1b shows an implementation based on the proposed decomposition approach, in a form of PMTBDD. The PMTBDD comprises two portions – the block (left) and the remainder (right). The portions are assembled by the newly introduced parallel connection of MTBDDs. Notice that according to the product operation introduced in Definition 2, sets of terminal nodes in the PMTBDD and in the standard MTBDD are not the same. It is the result of the concatenation operation between the original terminal nodes. The concatenation is calculated as the OR function between corresponding output vectors (see Definition 2). For example, terminal node 7 in the MTBDD (Figure 1) corresponds to two terminal nodes 3 and 6 in the PMTBDD (Figure 3). Such cases reflect the non-disjoint property, as in cubes 1 and 2 from Table 1.

The proposed approach, presented by Example 2, allows achieving significant improvement: indeed, the standard MTBDD has 17 non-terminal nodes (NTNs), after linearization this is reduced to 12 NTNs (and 2 XOR gates). PMTBDD has 8 NTNs, linearization further reduces it to 6 NTNs (and 2 XOR gates).

### 4.2. Choosing the Prefixes

In this section the process of choosing the suitable block header of the iteration is discussed. The requirements for the first (basic) prefix and all other (secondary) prefixes are given and the criteria the choosing them are suggested.

The choice is performed by grading all the potential candidates, computing a weighted average of the different grades for each candidate, and then selecting the candidate with the highest grade.

The following notations are used: $P$ – the prefix under consideration. $R$ – the set of rows of the implicant table.

For a selected prefix: $T_F$ – the family of the prefix. $T_R$ – the set of the terms that don't depend on any of the prefix's variables. $T_U = R \setminus (T_F \cup T_R)$ – the set of terms depending on some of the prefix's variables. It is called the "undecided" set, since these terms are neither the prefix's family nor its remainder. $T_0$ – the set of terms orthogonal to the prefix. $T_0 \subseteq T_U$. $M$ – the number of variables in the prefix. $N = |T_F|$.

For any set $A$ of terms:

$X(A) = \{x_i : \exists t \in A, t(x_i) \neq "-"\}$. In other words, $X(A)$ is the set of variables in all the terms of $A$.

$S(A) = \sum_{t \in A} |\{x_i : t(x_i) \neq "-"\}|$. In other words, $S(A)$ is the number of literals in all the terms of A.

$Y(A) = sup \bigcup_{\{t \in A\}} S_t$, where $S_t$ is the row in the Sums matrix of the implicant table corresponding to $t$. In other words, $Y(A)$ is the set of outputs corresponding to all the terms of $A$.

*4.2.1 List of Candidates*

The prefix can be either a complete row of the Products matrix of the implicant table, or a subset of literals common to several terms. A straightforward procedure is proposed below for constructing the list of the candidates from the list of terms of the implicant table.

Let $x, y$ be variables with values from {0, 1, –}. The operator $\psi(x,y)$ compares the two Boolean variables and returns the value of one of them if they are equal, "don't care" otherwise:

$$\psi(x,y) = xI(x = y) + "don't\ care" I(x \neq y) \quad (5)$$

The function Common($T_1$, $T_2$) accepts the two terms $T_1$ and $T_2$ and applies ψ in a bitwise manner to each of the variables in the set $X(T_1) \cup X(T_2)$. Finally, the following method summarizes the suggested procedure.

```
Constructing the List of Candidates.
INPUT: List of prodict terms.
OUTPUT: List of candidates.
Output ← Input
Temp1 ← Output
REPEAT
    Temp2 ← Empty set
    Apply Common() to every pair of terms from Temp1.
If the result is not empty, add it to Temp2.
    Output ← Output U Temp2
    Temp1 ← Temp2
UNTIL |Temp1| = 0
```

*4.2.2 Choosing the Basic Prefix*

The basic prefix is the foundation of a block. It is chosen so as to make the block header the most suitable for a BDD implementation. For this, the basic prefix has to attract the secondary prefixes "close" to it and repel those "far" from it.

There are three main concerns to consider here: the input variables, the output functions and the length of the prefix. In addition, since the secondary prefixes will be chosen from the set $T_O$, it is imperative to measure the self-orthogonality of $T_U$.

The first criterion answers the "Inputs" requirement:

$$\alpha_X = 1 - \frac{|X(T_F) \cap X(T_R)|}{|(X(T_F) \setminus X(P)) \cup X(T_R)|} \quad (6)$$

It counts the variables common to the tail and the remainder corresponding to the prefix. The ratio has to be reduced as much as possible, in order to separate the block (with its tails) from the remainder. This criterion has values in [0, 1] interval, 0 corresponding to the case when all the remainder's variables are present in the tail and 1 to the opposite.

The second criterion answers the "Outputs" requirement:

$$\alpha_Y = 1 - \frac{|Y(T_F) \cap Y(T_R)|}{|Y(T_F) \cup Y(T_R)|} \quad (7)$$

It counts the outputs common to the tail and the remainder corresponding to the prefix. The rationale here is the same as for the "Inputs" requirement.

The third criterion, called "Prefix Area", measures the percentage of the area covered by the prefix within its family.

$$\alpha_P = \frac{MN}{S(T_F)} \quad (8)$$

The reasoning is simple: the longer the basic prefix, the longer the list of candidates for the secondary prefixes.

The final criterion, called "Orthogonality" answers the additional requirement.

$$\alpha_T = \frac{S_b(T_O)}{S_b(T_U)} \quad (9)$$

It counts the number of literals in the terms orthogonal to the prefix relative to the number of literals in all the candidates.

*4.2.3 Choosing the Secondary Prefix*

In selecting the basic prefix it's important to establish a solid foundation for the block. For secondary prefixes the goal is different, and the requirements change accordingly. The prefixes already chosen into the block have to be taken into account.

In the following equations, the superscript indices $i$ and $i+1$ stand for "current situation" and "after adding the considered prefix", respectively.

The first criterion, called "Additional Inputs", counts the number of variables common to the tail and to the remainder of the considered prefix, but only those not hitherto present in the block.

$$\beta_X = 1 - \frac{|X^{i+1}(T_F) \cap X^{i+1}(T_R)| - |X^i(T_F) \cap X^i(T_R)|}{|(X^{i+1}(T_F) \setminus X^{i+1}(P)) \cup X^{i+1}(T_R)|} \quad (10)$$

The second criterion, called "Additional Outputs", counts the number of output functions common to the tail and to the remainder of the considered prefix. Here, like in the previous criterion, only the newly added outputs are considered.

$$\beta_Y = 1 - \frac{|Y^{i+1}(T_F) \cap Y^{i+1}(T_R)| - |Y^i(T_F) \cap Y^i(T_R)|}{|Y^{i+1}(T_F) \cup Y^{i+1}(T_R)|} \quad (11)$$

The third criterion, called "Additional Area", measures the additional area brought to the block and to the remainder by selecting the considered prefix.

$$\beta_S = \frac{S^{i+1}(T_F) - S^i(T_F)}{S^{i+1}(T_F)} - \frac{S^{i+1}(T_R) - S^i(T_R)}{S^{i+1}(T_R)} \quad (12)$$

This equation can be rewritten as follows:

$$\beta_S = 1 - \frac{S^i(T_F)}{S^{i+1}(T_F)} - \left(1 - \frac{S^i(T_R)}{S^{i+1}(T_R)}\right) = \frac{S^i(T_R)}{S^{i+1}(T_R)} - \frac{S^i(T_F)}{S^{i+1}(T_F)} \quad (13)$$

Each of the two fractions is limited to the interval [0, 1], but the total value of $\beta_S$ is in the interval [–1, 1]. In

this it differs from all the other criteria and ruins somewhat the elegance of the total, but does not bear any serious impact on the results.

*4.2.4 Combining the Criteria*

The four criteria and the three criteria for the basic prefix are combined in the following weighted average.

$$\alpha = a_X\alpha_X + a_Y\alpha_Y + a_P\alpha_P + a_T\alpha_T \quad (14)$$
$$\beta = b_X\beta_X + b_Y\beta_Y + b_S\beta_S$$

When choosing the basic prefix, the candidate with the highest $\alpha$ is taken. Likewise, when choosing the secondary prefix, the candidate with the highest $\beta$ is taken. The coefficients of the criteria have to be chosen so as to get the optimal result.

## 5. EXPERIMENTS

The experiments demonstrate that the proposed decomposition, when successful, greatly reduces the size of the BDD. Its success strongly depends on the density of the implicant table. Therefore, its effectiveness can be predicted quite reliably by making some preliminary study of the implicant table functions' representation.

The goals of the conducted experiments are as follows:

1. Comparing the effectiveness of the complete linearization (performed with $K$-Procedure) and that of the Parallel decomposition.
2. Investigating the relative importance of the coefficients and formulating the guidelines concerning the weights.

During the experiments the implicant table representations of the standard combinatorial-circuit benchmarks (LGSYNTH93) were used. The experiments were limited to the following subset of benchmarks:

– Number of terms the implicant table is limited by 200. This limitation may seem somewhat severe, and, indeed, the Parallel decomposition can be performed in certain cases for implicant tables of over 500 terms. In general, however, since the search for common parts grows polynomially in the number of input terms, it is suggested that this value is kept below 200.

– Number of input variables limited by 23. This is a necessary condition for running the Exhaustive $K$-Procedure in its tabular form.

– Number of output functions limited by 31. This is the weakest of all requirements and is dictated by the treatment of the outputs as integers.

Both the Parallel decomposition and the $K$-Procedure based linearization of resulting blocks were checked on standard combinatorial-circuit benchmarks (LGSYNTH93). For each benchmark, the sizes of the following diagrams were recorded and compared: original BDD; LTBDD; PMTBDD; and LPMTBDD.

The results are shown in the Tables 2 and 3. The first lists the benchmarks for which the Parallel decomposition is more effective than the linearization of the initial implicant table without decomposition. The second shows the failures.

The columns in the tables are as follows. For each benchmark, the number of the input variables and the implicant table density are followed by the four columns of the results. The two additional columns show the improvement/degradation ratios: from BDD to PMTBDD, with and without the linearization. The ratios are given in per cents. Both tables are sorted by the ascending the implicant table density.

Table 2. Benchmarks results, |LPMTBDD| < |LTBDD|

| Title | \|X\| | Density [%] | BDD | LTBDD | PBDD | LTPBDD | PBDD / BDD [%] | LTPBDD / LTBDD [%] |
|---|---|---|---|---|---|---|---|---|
| ALU1 | 12 | 18 | 982 | 632 | 25 | 25 | 2.55 | 3.96 |
| B12 | 15 | 29 | 155 | 155 | 145 | 139 | 93.55 | 89.68 |
| DK48 | 15 | 31 | 3428 | 1877 | 58 | 57 | 1.69 | 3.04 |
| DK27 | 9 | 34 | 79 | 42 | 22 | 19 | 27.85 | 45.24 |
| CON1 | 7 | 37 | 16 | 16 | 15 | 14 | 93.75 | 87.5 |
| ALU2 | 10 | 39 | 264 | 264 | 150 | 147 | 56.82 | 55.68 |
| DUKE2 | 22 | 40 | 1435 | 457 | 326 | 226 | 22.72 | 49.45 |
| ALU3 | 10 | 42 | 278 | 278 | 151 | 132 | 54.32 | 47.48 |
| MISEX3C | 14 | 43 | 10875 | 6722 | 705 | 534 | 6.48 | 7.94 |
| WIM | 4 | 50 | 15 | 12 | 10 | 9 | 66.67 | 75 |
| F51M | 8 | 53 | 255 | 255 | 155 | 135 | 60.78 | 52.94 |
| DK17 | 10 | 57 | 160 | 83 | 55 | 54 | 34.38 | 65.06 |
| APLA | 10 | 64 | 128 | 94 | 85 | 77 | 66.41 | 81.91 |
| INC | 7 | 79 | 39 | 37 | 35 | 34 | 89.74 | 91.89 |
| | | | | | MEAN RATIO | | 48.41 | 54.06 |

Table 3. Benchmarks results, |LPMTBDD| > |LTBDD|

| Title | \|X\| | Density [%] | BDD | LTBDD | PBDD | LTPBDD | PBDD / BDD [%] | LTPBDD / LTBDD [%] |
|---|---|---|---|---|---|---|---|---|
| ADD6 | 12 | 52 | 504 | 384 | 731 | 569 | 145.04 | 148.18 |
| RADD | 8 | 57 | 90 | 64 | 143 | 106 | 158.89 | 165.63 |
| CLIP | 9 | 59 | 189 | 143 | 376 | 318 | 198.94 | 222.38 |
| Z4 | 7 | 61 | 52 | 29 | 101 | 79 | 194.23 | 272.41 |
| ROOT | 8 | 65 | 72 | 71 | 134 | 124 | 186.11 | 174.65 |
| SQR6 | 6 | 67 | 63 | 63 | 85 | 75 | 134.92 | 119.05 |
| SQN | 7 | 69 | 81 | 40 | 116 | 92 | 143.21 | 230.00 |
| MLP4 | 8 | 73 | 240 | 202 | 345 | 305 | 143.75 | 150.99 |
| SAO2 | 10 | 73 | 95 | 58 | 157 | 122 | 165.26 | 210.34 |
| DIST | 8 | 73 | 125 | 108 | 326 | 292 | 260.80 | 270.37 |
| BW | 5 | 80 | 25 | 23 | 58 | 56 | 232.00 | 243.48 |
| RD53 | 5 | 90 | 15 | 13 | 53 | 44 | 353.33 | 338.46 |
| | | | | | MEAN RATIO | | 193.04 | 212.16 |

The analysis of these results shows that the density of the implicant table is, indeed, a reliable indicator of the success of the function's Parallelization. The successful cases (|PMTBDD|<|MTBDD| and/or |LTPBDD|<|LTBDD|) are mostly in the low-density area (Density up to 45%) and the unsuccessful ones are mostly in the high-density area (Density at least 60%). The middle or gray area functions (Density within 40-60%) are divided more or less evenly between the successes and the failures. Moreover, there are several examples where the high-density functions are successfully decomposed, and no examples where the method failed to work on low-density functions.

The Parallel decomposition, on the other hand, relies upon extracting dense fragments from the given implicant table, and treating the sparse remainders and tails separately. Therefore, sparse implicant table can be easily dealt with by splitting them into a network of concurrently working BDDs. With dense implicant tables, choosing suitable blocks is difficult, and arbitrary choices lead to ineffective implementations. This is summarized in Table 4.

Table 4. Analysis of the results.

| | | Density | |
|---|---|---|---|
| | | Low | High |
| Method | PLA→BDD | Lots of "don't cares". Lots of replications. BDD not compact. | Few "don't cares". Number of replications small. BDD compact. |
| | Parallelization | Several component fragments. Easy choice of component diagrams. Suitable for PBDD. | Dense PLA – difficult to determine separate fragments. Unsuitable for PBDD. |

## 6. CONCLUSIONS

The paper describes a new method of a so-called piecewise linearization of logic functions, which comprises a) decomposing the initial implicant table of a logical function into a network of component Multi Terminal BDDs, and b) separate linearization of these BDDs. Two connecting operations are used in the decomposition network, i.e., serial and parallel connections of the component BDDs. A new mathematical basis of the parallel connection of the BDDs is formed by a newly introduced algebra of $D$-polynomials. This new algebra, as well as a number of theoretical statements proven in the paper are used as a theoretical background of the proposed decomposition algorithm.

The decomposition algorithm is presented in details. Benchmark results demonstrate efficiency of the proposed approach in comparison with straightforward implementations of MTBDDs (including or not including linearization). solutions with linearization and without linearization. The proposed piecewise linearization opens a way for using the linearization technique to implement logic functions having a great number of variables.

## 7. REFERENCES


[1] M.G.Karpovsky, R.Stancovic, J. Aastola, "Reduction of Sizes of Decision Diagrams by Autocorrelation Functions", *IEEE Trans on Computers, May, 2003, pp.592-607.*

[2] M. Fujita, Y. Kukimoto, R. K. Brayton, "BDD minimization by truth table permutation", *Proc. Int. Symp. on Circuits and Systems, ISCAS'96, May 12-15, 1996, Vol. 4, 596-599.*

[3] Rudell, R., "Dynamic variable ordering for ordered binary decision diagrams", *Proc. IEEE Conf. Computer Aided Design, Santa Clara, CA, 1993, 42-47.*

[4] W. Gunther, R. Drechsler, "Linear transformations and exact minimization of BDDs", *Proc. 8th Great lake Symp. on VLSI, February 19-21, 1998, 325-330.*

[5] Levin I., Stankovic R., Karpovsky M., Astola J., (2005) "Construction of Planar BDDs by Using Linearization and Decomposition". *Proceedings of Fourteenth International Workshop on Logic and Synthesis, Lake Arrowhead, California, pp. 132-139.*

[6] Keren O., Levin I., "Linearization of the Logic Functions Defined in SOP Form". *Procs. of the Work in Progress Session, DSD 2005, Porto (Portugal), September 2005, E. Grosspietsch, K. Klockner (eds.), SEA-Publications: SEA-SR-09, July 2005.*

[7] Levin, I., Levit, V. (1998). "Controlware for Learning with Mobile Robots. *Computer Science Education", 8(3), 181-196.*

[8] Meinel, C. Somenzi, F. Theobald, T. "Linear Sifting of Decision Diagrams and its Application in Synthesis". *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems 2000, Vol 19; part 5, pages 521-533.*