Concurrent Decomposition of Multiterminal BDDs[1]

Ilya Levin[1], Osnat Keren[2], Vladimir Ostrovsky[1], George Kolotov[1]
[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

Abstract
The paper deals with the problem of decomposition of logic functions and their implementation in a form of Multi Terminal Binary Decision Diagrams (MTBDD). The new logic decomposition method and the implementation style, called concurrent decomposition, introduced in the paper leads to a compact VLSI layout of locally interconnected logic elements. It is easily amenable to VLSI implementations of custom design in deep submicron technology, where control over interconnect wiring and its delay becomes of primary importance.
The proposed decomposition method is based on the algebra of D-polynomials reviewed in the paper. The resulting decomposition is directly mapable onto special type of binary graph called Concurrent Multi Terminal BDD. The paper describes both the theoretical fundamentals of the decomposition algorithm and its certain implementation. Benchmark results are presented and analyzed. Guidelines for effective using of the proposed technique are provided.

1. Introduction

There have been several attempts to derive logic structures that could efficiently be implemented in VLSI technology. They all strive to derive a regular structure (array or lattice) of identical or similar elements that could be interconnected only by local (neighbor-to-neighbor) interconnections. Some of them are based on direct implementation of various BDD's, with nodes being implemented as MUXes, while recognizing various symmetry properties of logic functions [1]. Others explore decomposition methods based on Reed-Muller expansion and use of XORs [2, 3]. Most of these attempts are still impractical due to a large number of logic elements needed, or their area inefficiency. Often there is a need for duplication of binary variables during logic expansion and extraction of symmetric functions. Finally, current decomposition techniques remain strongly dependent on the ordering of variables during logic expansion making the resulting diagrams/structures too large. An in-depth analysis and extensive survey of logic decomposition methods can be found in [4].
This paper deals with decomposition of multi-output logic functions for their efficient implementation in a form of Multiterminal Binary Decision Diagrams (MTBDDs). Partitioning of the shared BDD representation of multi-output functions was proposed in [5], where the authors suggested to partition the set of outputs of a multi-output function and to optimize the obtained functions separately.
The present paper proposes a different approach. It introduces a so-called concurrent decomposition of multi-output functions, allowing the partitioning of not only the set of outputs of the function, but also the set of its implicants.
The proposed approach may be considered as a new logic decomposition method and an implementation style, which can lead to a compact VLSI layout of locally interconnected logic elements. It is easily amenable to VLSI implementations of custom design using deep submicron technology, where control over interconnect wiring and its delay becomes of primary importance.
The theoretical foundation of our decomposition method is based on algebra of D-polynomials [6, 7]. The result of the decomposition is a direct mapping of the logic function onto the MTBDD.
The decomposition approach described in this paper aims at practical implementations of logic functions as VLSI structures. The main purposes of the proposed technique are to minimize the size of the resulting implementation while maintaining full routability and predictability of interconnect delay. Specifically, the problem discussed here can be stated as follows:
Given a minimized sum-of-product representation of a multiple-output logic function, construct a network of concurrently functioning MTBDD with minimum number of nodes.

[1] The work was supported by BSF under Grant 2002259

The optimized MTBDD network, dominated by local interconnections, has to guarantee easy routability and facilitates a reliable prediction of interconnect delay. Since each node can be readily implemented as a simple logic element/gate, their minimization directly translates into minimum-area, compact layouts. Other criteria, such as performance, power, testability, etc., can be also considered.
The main advantage of the approach proposed in this paper is the locality of interconnects, which is highly desirable in order to achieve the desired design convergence during the physical design phase. Unlike other methods, that lead to regular layouts in form of binary trees or lattices [1, 4], this method does not require repetition of input variables or extraction of symmetric or pseudo-symmetric functions.
It should also be noted that, in practice, logic functions can rarely be presented in a form of a single expression. Usually it is more convenient to describe them as a system of hierarchical functions and concurrent logic statements. The method described here is able to work using such forms of representation of systems of logic functions.
The paper is organized as follows. Section 2 introduces both Concurrent Multi Terminal BDDs (CMTBDDs) and D-polynomials. Section 3 deals with the proposed method of decomposition including its theoretical fundamentals and the decomposition algorithm. Section 4 presents experimental results. Conclusions are provided in Section 5.

2. Concurrent Multiterminal BDDs and D-Polynomials

The proposed decomposition approach is based on a concept of CMTBDD, which can be constructed by combining several CMTBDDs 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.

Each node of CMTBDD, associated with a binary variable, has one input and two outputs. An input to a node is a product term generated along the path from the root of the tree to a given variable. Except for the root node, which has a trivial input of 1, each input α to a node comes from the output of the node placed directly one level above it. A node associated with variable x_i adds one literal $(x_i, or\ \overline{x}_i)$ to the product term α. One output of the node represents product term $\alpha\, x_i$, the other $\alpha\, \overline{x}_i$.
Introducing the CMTBDD opens a way of handling systems of logic functions defined by their SOP with a large number of variables and, consequently, reducing the total number of nodes in the resulting MTBDD.
CMTBDDs have there analytical interpretation as so-called D-polynomials [6]. We show below that the parallel and series connections of CMTBDDs can be interpreted as a product and substitution of D-polynomials representing those CMTBDDs. This fact is used as a basic principle of the proposed decomposition approach.

3.1. Representation of Logic Functions

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

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

$I(i)$ denotes an index set of implicants associated with the output Z_i. Implicants can be shared between different outputs. Since the coefficients α are functions of input variables $(x_1, x_2, \ldots, x_n)$, we will also refer to them as the α-functions.
Let Y_i be the label associated with the output Z_i. 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

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

while the coefficients Z_i satisfying the conditions:
a) $\bigcup_i Z_i \vee \alpha_0 = \boldsymbol{1}$ (completeness), b) $Z_i \cdot Z_j = \boldsymbol{0}, \forall i \neq j$ (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 α_{ij} denotes j th -implicant of function Z_i.

Example 1. The following is a D-polynomial: $D_1 = x_1 Y_1 + \overline{x}_1 \overline{x}_2 Y_2 + \alpha_0 Y_0$

Here, $\alpha_1 = x_1$, $\alpha_2 = \overline{x}_1 \overline{x}_2$ and $\alpha_0 = \overline{x}_1 x_2$. 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.

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 α_i are defined explicitly as the corresponding product terms of the function, while α_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 α-functions belonging to different D-polynomials by labeling them with a super-script index associated with the corresponding polynomial; α_i^k will indicate that implicant α_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.

Conceptually, a D-polynomial D_i can be interpreted as follows. If α_1^i evaluates to 1, then $D_i = Y_j$. If all of the explicit functions α_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 $\alpha_0^1 Y_0$ and $\alpha_0^2 Y_0$. Here $\alpha_{ij}^1 \cdot \alpha_{kl}^2$ is a logic product (AND) of the corresponding α-functions and $Y_i \circ Y_j$ is a combination of the respective operators. In other words: when $\alpha_{ij}^1 \cdot \alpha_{kl}^2$ 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:

$$D_i = \sum_j \underset{j}{Z^i} Y_j + \alpha_0^i Y_0 = \prod_j \left(Z_j^i Y_j + \alpha_{i0}^j Y_0 \right) \quad (3)$$

where $\alpha_0^j = \overline{Z_0^i}$, and $\prod_j \alpha_{i0}^j = \alpha_0^i$.

Proof. We will demonstrate that the product of $\left(Z_j^i Y_j + \alpha_{i0}^j Y_0 \right)$ is equal to the D-polynomial with the same coefficients. First, notice that by definition all explicit functions are pairwise orthogonal, so that $Z_j^i Y_j \cdot Z_k^i Y_k = 0$, for $j \neq k$. Furthermore, orthogonality of the functions and the completeness condition $\alpha_{i0}^j = \overline{\alpha_j^i}$ that must be satisfied by each D-binomial imply that $Z_j^i \subset \alpha_{i0}^k$, for $k \neq j$. Hence $Z_j^i \cdot \alpha_{i0}^k = Z_j^i$. Also, notice that the concatenation $\{ Y_j \circ Y_0 \}$ 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 $\{ Y_j \circ Y_0 \} = Y_j$. Finally, orthogonality and completeness conditions of α-functions imply

that $\bigcup_j \alpha_{i0}^j = 1$, $\alpha_0^i \subset \alpha_{i0}^j$, $\alpha_0^i \subset \alpha_{i0}^j \,\&\, \alpha_{i0}^k, \ldots,$ $\alpha_0^i \subset \prod_j \alpha_{i0}^j$, so that $\prod_j \alpha_{i0}^j = \alpha_0^i$,

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

$$\left(Z_1^i \cdot Z_2^i \{Y_1 \circ Y_2\} + Z_1^i \cdot \alpha_{i0}^2 \{Y_1 \circ Y_0\} + Z_2^i \cdot \alpha_0^i \{Y_2 \circ Y_0\} + \alpha_{i0}^1 \cdot \alpha_{i0}^2 \{Y_0 \circ Y_0\}\right) \prod_{j=1}^{m-2} \left(\alpha_j^i Y_j + \alpha_{i0}^j Y_0\right) =$$

$$= \left(Z_1^i Y_1 + Z_2^i Y_2 + \alpha_{i0}^1 \cdot \alpha_{i0}^2 Y_0\right) \prod_{j=1}^{m-2} \left(Z_j^i Y_j + \alpha_{i0}^j Y_0\right) = \dots$$

$$= \left(Z_1^i Y_1 + Z_2^i Y_2 + \ldots + Z_m^i Y_m + \alpha_{i0}^1 \cdot \alpha_{i0}^2 \cdot \ldots \cdot \alpha_{i0}^m Y_0\right) = \left(Z_1^i Y_1 + Z_2^i Y_2 + \ldots + Z_m^i Y_m + \alpha_0^i Y_0\right) = \sum_j Z_j^i Y_j + \alpha_0^i Y_0$$

QED.

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

$$D_i = \sum_j Z_j^i Y_i + \alpha_0^i Y_0 = \prod_{jI(i)} \left(\alpha_{ij}^i Y_i + \alpha_{i0}^i Y_0\right), \quad (4)$$

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

Proof. Let $D_m = \left(\alpha_{1j}^m Y_1 + \alpha_{j0}^m Y_0\right) \circ \left(\alpha_{1k}^m Y_1 + \alpha_{k0}^m Y_0\right)$. After performing the multiplication we have:

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

$$\left(\alpha_{1j}^m \cdot \alpha_{1k}^m + \alpha_{1j}^m \cdot \overline{\alpha_{1k}^m} + \overline{\alpha_{1j}^m} \cdot \alpha_{1k}^m\right) Y_1 + \alpha_{j0}^m \cdot \alpha_{k0}^m Y_0 = \left(\alpha_{1j}^m + \alpha_{1k}^m\right) Y_1 + \alpha_0^m Y_0$$

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:

$$Z_j^i = \sum_{I(i)} \alpha_{ij}^i Y_i + \alpha_0^i Y_0 = \prod_{I(i)} \left(\alpha_{ij}^i Y_i + \alpha_{i0}^i Y_0\right) \quad (5)$$

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 CMTBDD'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. QED.

In other words, an MTBDD corresponding to the product of D -polynomial can be realized as a parallel connection of subtrees corresponding to the individual D -polynomials.

4. Concurrent Decomposition

The theoretical basic of the proposed decomposition algorithm is the following theorem of non-trivial disjunctive decomposition.

4.1. Non-trivial disjunctive decomposition

Let $y_i = f_i\left(x_1, \ldots, x_n\right), i = 1, \ldots, m$ be a system of logic functions defined by its implicant table $X = \{x_1, \ldots, x_n\}, Y = \{y_1, \ldots, y_m\}$. We deal with a problem of disjunctive decomposition of this system, i.e. we consider a problem of representing the system in the following form: $y_i = \varphi_i\left(X_1\right) + \psi\left(X_2\right)$, where $X_1, X_2 \subset X$, φ and ψ are implicant tables. Notice, that such a decomposition always exists in the case when $X_1 = X$ or $X_2 = X$. We call such decomposition trivial. The following Theorem 2 defines the condition of existence of a non-trivial decomposition

Theorem 2. The non-trivial disjoint decomposition of the function $y_i = f_i(x_1,\ldots,x_n), i = 1,\ldots,m$ exists if and only if the function may be presented as an implicant table where each row includes "don't-cares" in each of a column belonging to one of the following sets of variables: $X \setminus X_1$ or $X \setminus X_2$.

Proof.

Sufficiency. If the implicant table satisfies the theorem conditions, than it may be naturally partitioned into two subtables according to functions $y_i = \varphi_i(X_1)$ and $y_i = \psi(X_2)$.

Necessity. If the system is presented in a form $y_i = \varphi_i(X_1) + \psi(X_2)$, than each of output variables y_i is equal to 0 only when both of the functions are equal to 0. Thus, if $\varphi_i(X_1) = 1$ and $\psi(X_2) = 1$, than $y_i = 1$ on these same vectors independently from values of $\varphi_i(X_1)$ and $\psi(X_2)$ and, consequently, independently from values of the variables from the set $X \setminus X_1$ $(X \setminus X_2)$, QED.

4.2 Decomposition algorithm

Based on Theorem 2, the proposed decomposition algorithm uses a parallel sharing 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.

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

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 CMTBDD. We propose to select the header by taking into account high percentage of "non-don't care" cells (density) of the corresponding fragment the implicant table.

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 algorithm divides the initial function into a block and a remainder. The block is a sum of products of simpler sub-functions with prefix terms. The group of prefixes – 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 can be repeatedly decomposed in the same way, until no further decomposition is possible.

The main part of a particular iteration of the decomposition algorithm 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 output 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 BDDs and do not require additional iterations. Example 2 illustrates the idea of the algorithm.

Example 2. The exemplary implicant table is presented in Table 1.

Table 1: The implicant 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

MTBDD and CMTBDD for the example are presented in Figures 1a and 1b correspondingly.

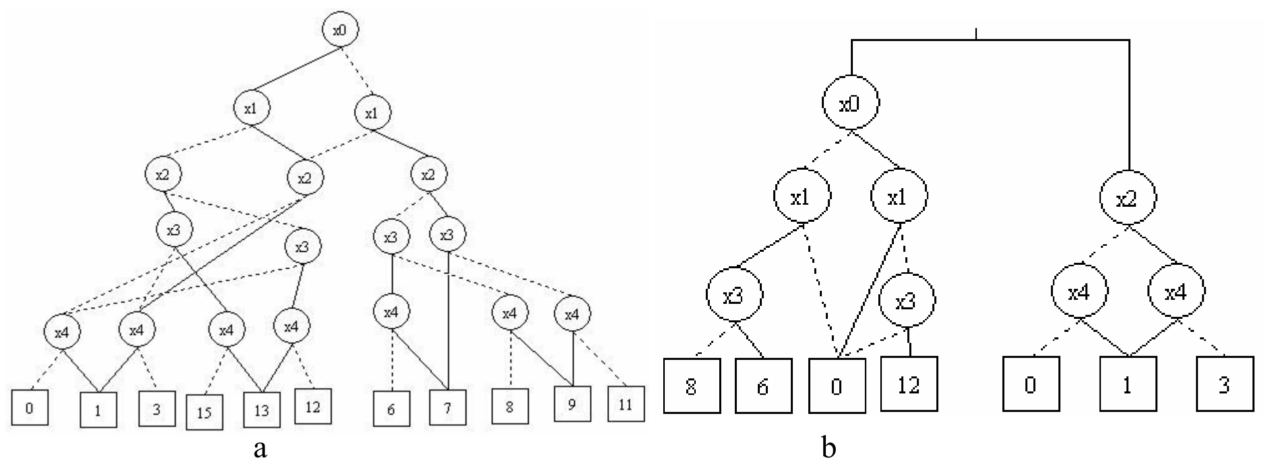

Figure 1. MTBDD (a) and the CMTBDD (b) for the Example 2.

Terminal nodes of MTBDDs in Figure 1 are marked by decimal numbers of corresponding outputs. A standard implementation of the exemplary MTBDD, as an ordered BDD, is presented in Figure 1a. Figure 1b shows an implementation based on the proposed decomposition approach, in a form of CMTBDD. The CMTBDD 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 CMBDD 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 form (a) corresponds to two terminal nodes 3 and 6 in the CMTBDD form (b). Obviously, 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), while CMTBDD has only 8 NTNs.

5. Experiments

The experiments demonstrate that the proposed decomposition, when successful, greatly reduces the size of the MTBDD. 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 goal of the conducted experiments was comparing the effectiveness of the straightforward implementation of the MTBDD with the proposed concurrent decomposition.

In the experiments the implicant table representations of the standard combinatorial-circuit benchmarks (LGSYNTH93) were used.

The results are shown in the Tables 2. The first lists the benchmarks for which the Concurrent decomposition is more effective than the MTBDD 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 MTBDD to CMTBDD. The ratios are given in percents. Both tables are sorted by the ascending the implicant table density.

Table 2. Benchmarks results, where: |CMTBDD|<|MTBDD| (left); |CMTBDD|>|MTBDD| (right)

Title	|X|	Density [%]	MTBDD	CMTBDD	CMTBDD /MTBDD [%]
ALU1	12	18	982	25	2.55
B12	15	29	155	145	93.55
DK48	15	31	3428	58	1.69
DK27	9	34	79	22	27.85
CON1	7	37	16	15	93.75
ALU2	10	39	264	150	56.82
DUKE2	22	40	1435	326	22.72
ALU3	10	42	278	151	54.32
MISEX3C	14	43	10875	705	6.48
WIM	4	50	15	10	66.67
F51M	8	53	255	155	60.78
DK17	10	57	160	55	34.38
APLA	10	64	128	85	66.41
INC	7	79	39	35	89.74

Title	|X|	Density [%]	MTBDD	CMTBDD	CMTBDD / MTBDD [%]
ADD6	12	52	504	731	145.04
RADD	8	57	90	143	158.89
CLIP	9	59	189	376	198.94
Z4	7	61	52	101	194.23
ROOT	8	65	72	134	186.11
SQR6	6	67	63	85	134.92
SQN	7	69	81	116	143.21
MLP4	8	73	240	345	143.75
SAO2	10	73	95	157	165.26
DIST	8	73	125	326	260.8
BW	5	80	25	58	232
RD53	5	90	15	53	353.33

The above results show that the density of the implicant table is a consistent indicator of the success of the decomposition. The successful cases 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 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.
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, a sparse implicant table can be easily dealt with by splitting them into a network of concurrently working MTBDDs. With dense implicant tables, choosing suitable blocks is difficult, and arbitrary choices lead to ineffective implementations.

6. Conclusions

The proposed in the paper decomposition approach is based on parallel connecting of multi terminal BDDs. By introducing the parallel connecting we have achieved an implementation of logic functions in a form of a structure desirable for effective routing. This structure called Concurrent MTBDD has advantages in comparison with conventional MTBDDs both from the point of locality of interconnected logic elements and from the point of the number of required nodes of the resulting diagram. It is easily transformable to VLSI implementations using deep submicron technology, where control over interconnect wiring and its delay becomes of primary importance.
The theoretical background of the proposed approach is algebra of D-polynomials and the Theorem of the existence of the non-trivial disjunctive decomposition. The theorem serves as a justification of the proposed decomposition algorithm. Efficiency of the algorithm is evaluated on a number of benchmarks. The experiments show that the main parameter that has to be taken into account when using the proposed approach is the density of the corresponding implicant table (the percent of "non-don't-care" cells). The proposed technique is highly effective if the density is relatively low (up to 45%).
Notice that the proposed technique broadens the plurality of multi-output functions that can be implemented in the form of MTBDD. Indeed, benchmarks with a huge number of implicants, which cannot be handled by traditional techniques due to high complexity of these functions, can be now successfully implemented in the form of the proposed CMTBDD since the suggested technique performs partitioning of such multi-output functions and implements them as a net of interconnected fragments.
The authors believe that the proposed structure and decomposition approach will facilitate the layout design and make this approach amenable to various implementations.

References

[1] Akers, S.B: A Rectangular Logic Array, IEEE Trans. on Computers, Aug. 1972, pp. 848-857.

[2] Perkowski, M. Chrzanowska-Jeske, and Y. Xu: Lattice Diagrams Using Reed-Muller Logic, Proc. RM'97 Conference, Oxford Univ., U.K., Sept. 1997, pp. 85 – 102.

[3] Song, N, M. Perkowski, M. Charzanowska-Jeske: A New Design Methodology for Two-Dimensional Logic Arrays, VLSI Design, Vol.3, 1995, pp.315-332.

[4] Perkowski, M., S. Grygiel: A Survey on Function Decomposition, Version IV, 1995. Available on the web.

[5] M. Matsuura, T. Sasao, J.T. Butler, and Y. Iguchi: Bi-partition of shared binary decision diagrams, IEICE Transactions on Fundamentals of Electronics, Vol.E85-A, No.12, Dec. 2002, pp. 2693-2700.

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

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