2011 14th Euromicro Conference on Digital System Design

Generalized If-Then-Else Operator for Compact Polynomial Representation of Multi Output Functions

Ilya Levin
School of Education
Tel-Aviv University
Te-Aviv, Israel
ilia1@post.tau.ac.il

Osnat Keren
School of Engineering
Bar-Ilan University
Ramat-Gan, Israel
kereno@macs.biu.ac.il

***Abstract*—The paper studies a new polynomial representation of Multi Output Functions (MOFs). The new representation, called GITE-polynomials, is based on a newly introduced Generalized If-Then-Else (GITE) function. Being a compact form of representation of MOFs, the GITE-polynomials allow efficient manipulation with a set of functions. The paper introduces algebra of GITE-polynomials. Properties of this algebra are used for solving the MOF-decomposition problem. The solution provides a compact representation of MOFs.**

***Keywords*-Multi-output function, Binary Decision Diagram (BDD), decomposition, If-Then-Else operator.**

I. Introduction

Many problems in VLSI-CAD and other fields of computer science can be formulated in terms of Boolean functions. The central issue in providing computer-aided solutions to such problems is to find a compact representation for a set of Boolean functions, for which basic Boolean operations and equivalence check can be efficiently performed. Decision diagram based representation of a set of Boolean functions can be useful in verification and synthesis [13], [16], [6].

The requirements of compactness and manipulability usually are conflicting requirements. Currently, Binary Decision Diagrams (BDDs) serve the most popular compromise between these conflicting requirements. In a large number of practical cases, a conventional shared BDD and multi-terminal BDD representation of a set of Boolean function is exponential in the number of primary inputs. This fact limits complexity of the problems, which can be solved by using BDDs and, as a result, rises a problem of finding new compact representations of a set of logic functions.

Logic decomposition provides an efficient tool for obtaining compact representations. Algorithms for disjunctive and non disjunctive decomposition of Boolean function by using BDDs were presented in [3], [23]. A method for decomposition of a MOF into two functions with intermediate outputs using BDD was presented in [20]. In [18], the MOF representation using Galois field was studied. The authors use decomposition of Ordered Binary Decision Diagrams (OBDD) in their method. A unified logic optimization method, which handles both AND/OR and XOR function based on various dominators is presented in [22].

A compact representation of Multi Output Functions (MOFs) is in the focus of the present paper. Our approach is based on a newly introduced Generalized If-Then-Else (GITE) operator. A system of logic functions can be described as a GITE-formula (GITE-polynomial). In this sense, the conventional algebra of Boolean formula is a particular case of the GITE-algebra.

A particular case of the GITE representation (called D-polynomials) has been studied in [14], [15] where a multi-output function has been defined in a form of disjoint cubes having a specific structure. The GITE notation presented in this paper supports a wide variety of multi-output representations.

The paper is organized as follows. Section II provides a necessary background. The GITE operator and the algebra of GITE-polynomials are introduced in Section III. The decomposition algorithm is given in Section IV. Section V includes experimental results. Conclusions are given in Section VI.

II. Preliminaries

One of the popular forms for representation of a logic function is a Binary Decision Diagram (BDD). A fundamental operator allowing operations between BDDs is *If-Then-Else (ITE)* operator [9]. The operator has the following form:

$$ITE(f,g,h) = \begin{cases} g & if \quad f=1 \\ h & if \quad f=0 \end{cases},$$

where f is a Boolean function, g and h are BDDs corresponding to a specific logic functions. The ITE operator allows to implement any function of two variables [7], [9] and consequently may be considered as $universal$ basis on a set of logic functions. In our paper, we use the ITE operator as an operator on the *set of logic functions*.

A set of logic functions can be considered as a single function of n input binary variables and m binary output variables [10], [17], [19]. The domain and range of such a MOF are the n-dimensional and m-dimensional Boolean

978-0-7695-4494-6/11 $26.00 © 2011 IEEE
DOI 10.1109/DSD.2011.7

cubes, respectively. We refer to a vertex of the Boolean cube by its corresponding integer value.

The two main operations on MOF are the ITE and the $Apply$ [1]. The Apply may be used to accomplish a large number of matrix operations. Its definition is: $Apply(f, g, \circ) = f \circ g$, where f and g are MOFs, and $\circ$ is any binary operation on two operands, for example, $+, -$, min, max, etc.

A number of representations of MOFs were intensively studied. For example, a matrix representation (which is actually a Karnaugh-like form), and the Multi-Terminal Binary Decision Diagram (MTBDD) representation [1], [12], [21], [24]. This paper suggests an *analytical* representation of MOFs.

III. Generalized ITE operator

The first argument in the definition of the ITE operator can be interpreted as a Boolean function or as a certain *two-block* partition on the Boolean cube. In the present paper, we consider an arbitrary *n-block* partition instead of the two-block partition of the ITE operator. Such kind of generalization leads to a new *Generalized ITE (GITE)* operator.

Definition 1: Let π be a partition of the n-dimensional Boolean cube comprising w disjoint blocks B_i, $\pi = \{B_i\}_{i=0}^{w-1}$. Let $X = (x_{n-1}, \dots x_0)$ be an input vector of length n. Let H be a set of values. Denote by $h(B_i) \in H$ the value associated with the i'th block. Then, the GITE operator is

$$\begin{aligned} g(X) &= GITE(\pi, h(B_0), \dots h(B_{w-1})) \\ &= \begin{cases} h(B_i) & if \quad X \in B_i \\ 0 & if \quad X \notin B_i \end{cases} \end{aligned}$$

■

The GITE comprises a $partition$ (input) portion and a $functional$ (output) portion. The GITE operation maps a partition of the Boolean cube on the predefined set of output values.

In this paper, we use a Sum-of-Products (SOP) based analytical form for representing GITE formulas of MOFs. We call this analytical representation a GITE-polynomial representation.

Definition 2: Let $\pi = \{B_i\}_{i=0}^{w-1}$ be a partition where $\{B_i\}_{i=1}^{w-1}$ is a set of pairwise-disjoint Boolean functions expressed in a SOP form and $B_0 = \overline{\vee_{i=1}^{w-1} B_i}$. Denote by Y_i the output value of the MOF associated with the i'th block. A GITE-polynomial is defined as follows,

$$\begin{aligned} &D \stackrel{def}{=} \textstyle\sum_{i=1}^{w-1} B_i Y_i + B_0 Y_0 = GITE(\pi, Y) \\ &= GITE(B_1, \dots, B_{w-1}, B_0; Y_1, \dots, Y_{w-1}, Y_0) \end{aligned}$$

where Y_0 is the Null output vector, that is, $Y_0 \circ Y_i = Y_i$ for all Y_i. ■

Next we introduce two operators of the GITE-polynomials, product (apply) and factorization.

The $Apply$ operation corresponds to a *product* operation between GITE-polynomials. This product operation between two GITE-polynomials is defined as follows.

Denote by $\pi_i \cdot \pi_j$ the intersection (product) of partitions π_i and π_j, and by $\pi_i + \pi_j$ the sum of two partitions. Let $Y_i \circ Y_j$ be a predefined operation between Y_i and Y_j.

Definition 3 (Product of GITE-polynomials): Let

$$D_1 = \sum_{i=1}^{m_1} B_i^1 Y_i + B_0^1 Y_0, \text{ and } D_2 = \sum_{i=1}^{m_2} B_i^2 Y_i + B_0^2 Y_0.$$

The product of D_1 and D_2, denoted as $D_1 \circ D_2$ is defined by:

$$D_1 \circ D_2 = \sum (B_i^1 \cdot B_j^2)\{Y_i \circ Y_j\} + B_0^1 \cdot B_0^2 Y_0,$$

over each pair of products from D_1 and D_2, including the implicit terms $B_0^1 Y_0$ and $B_0^2 Y_0$. ■

Here $B_i^1 \cdot B_j^2$ is a logic product (AND) of the corresponding functions. In other words, when $B_i^1 \cdot B_j^2$ evaluates to 1, a specific predefined operation between Y_i and Y_j is performed.

According to the definition, the $Apply$ operation corresponds to the *product of the partition portions* of GITE-polynomials. The GITE-polynomial operation that corresponds to the *sum of the partition portions* is the *factorization* on the set of GITE-polynomials.

Definition 4 (Factorization of GITE-polynomials): Let D_i and D_j be two GITE-polynomials with partitions π_i and π_j. Denote by D the product $D = D_i \circ D_j$. The factorization of D is its representation in the following form: $D = GITE(\pi_i + \pi_j; D_1, \dots, D_l)$ where l is a number of blocks in $(\pi_i + \pi_j)$; $D_1, \dots, D_l$ stand for GITE-polynomials corresponding to remaining functions. ■

Example 1: Consider two partitions on the Boolean cube,

$$\begin{aligned} \pi_1 &= \{B_1^1, B_2^1, B_3^1\} = \{x_1, \bar{x}_1\bar{x}_2, \bar{x}_1 x_2\}, \\ \pi_2 &= \{B_1^2, B_2^2, B_3^2\} = \{\bar{x}_1, x_1\bar{x}_3, x_1 x_3\}, \end{aligned}$$

and two GITE-polynomials defined by these partitions

$$\begin{aligned} D_1 &= GITE(\pi_1, Y_1, Y_2, Y_3) \\ &= x_1 Y_1 + \bar{x}_1\bar{x}_2 Y_2 + \bar{x}_1 x_2 Y_3, \\ D_2 &= GITE(\pi_2, Y_4, Y_5, Y_6) \\ &= \bar{x}_1 Y_4 + x_1\bar{x}_3 Y_5 + x_1 x_3 Y_6. \end{aligned}$$

The product of the GITE-polynomials corresponding to the partition $\pi_3 = \pi_1 \cdot \pi_2$ is calculated as follows:

$$\begin{aligned} \pi_3 &= \pi_1 \cdot \pi_2 = \{\bar{x}_1\bar{x}_2, \bar{x}_1 x_2, x_1\bar{x}_3, x_1 x_3\}, \\ Y &= [Y_2 \circ Y_4, Y_3 \circ Y_4, Y_1 \circ Y_5, Y_1 \circ Y_6], \\ D_1 \circ D_2 &= GITE(\pi_3; Y) \\ &= \bar{x}_1\bar{x}_2\{Y_2 \circ Y_4\} + \bar{x}_1 x_2\{Y_3 \circ Y_4\} + \\ &\quad x_1\bar{x}_3\{Y_1 \circ Y_5\} + x_1 x_3\{Y_1 \circ Y_6\} \end{aligned}$$

The factorization corresponds to the sum of partitions $\pi_4 = \pi_1 + \pi_2$ and is calculated as follows:

$$\begin{aligned} \pi_4 &= \pi_1 + \pi_2 = \{x_1, \bar{x}_1\}, \\ D_1 \circ D_2 &= GITE(\pi_4; D_{234}, D_{156}) \\ &= x_1 D_{234} + \bar{x}_1 D_{156}, \end{aligned}$$

where $D_{156} = Y_1 \circ (\bar{x}_3 Y_5 + x_3 Y_6)$ and $D_{234} = (\bar{x}_2 Y_2 + x_2 Y_3) \circ Y_4$.

Let us introduce a substitution of certain GITE-polynomials into another GITE-polynomial as follows: Let D_1, D_2, D_3 be defined as,

$$\begin{aligned} D_1 &= GITE(\pi_1; Y_{11}, Y_{12}), \\ D_2 &= GITE(\pi_2; Y_{21}, \ldots, Y_{2m}), \\ D_3 &= GITE(\pi_3; Y_{31}, \ldots, Y_{3k}). \end{aligned}$$

After substitution $Y_{11} \leftarrow D_2, Y_{12} \leftarrow D_3$ we have $D_1 = GITE(\pi_1; D_2, D_3)$ which is a hierarchical structure comprising a number of GITE-polynomials.

A particular case of GITE-polynomials plays an important role in our study - the *D-binomial*.

Definition 5: GITE-binomial is a GITE-polynomial, which comprises exactly one cube, $D = B_1 Y_1 + B_0 Y_0$. ■

The following theorem states that any GITE-polynomial can be represented as the product of all its *GITE*-binomials.

Theorem 1: An arbitrary GITE-polynomial $D = \sum_j B_j Y_j + B_0 Y_0$ can be represented as a product of GITE-binomials $D_j = (B_j Y_j + B_0^j Y_0)$, that is, $D = \prod_j D_j$, where $B_0 = \prod_j B_0^j$.

IV. Decomposition algorithm

The goal of the proposed decomposition is to represent an initial function as a hierarchical network of modular components, each performing a part of the common functionality. We introduce a method for transforming functions into a structured hierarchical network of interconnected components. The MOF is decomposed into components while minimizing the total number of nodes in the MTBDD representing the initial function.

There are two extreme cases of decomposition of GITE-polynomials - the *monolith* and the *binomial-representation*. Recall that, any product of GITE-polynomials $\prod_i D_i$ can be represented as a product of GITE-binomials (Theorem 1). We consider this *binomial representation* as one extreme case. The opposite extreme case can be obtained by using the conventional MTBDD representation of MOF. We refer to this representation as *monolith.*

By applying the factorization (defined in Def. 4) on an initial GITE-polynomial, it is possible to represent a MOF as a monolith with terminals which are GITE-polynomials, $D = GITE(\pi_h, D_1, \ldots, D_j)$. In this case, the partition π_h ($\pi_h \geq \pi$) is referred to as a *header partition*. Similarly, it is possible to use GITE-polynomials as terminals in the binomial representation.

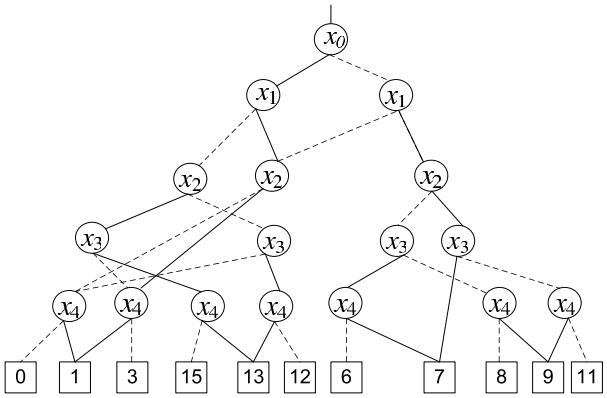

Figure 1. A monolith representation

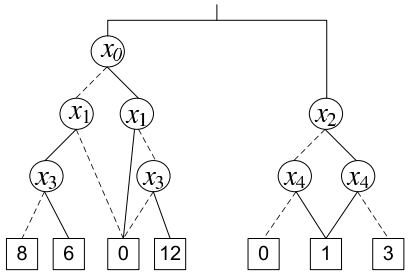

Figure 2. Decomposed MTBDD

There are many ways to group GITE-binomials to form a network of GITE-polynomials. Different grouping of the binomials yields different representations of MOF. Each GITE-polynomial has its own optimal structure, i.e. its own optimal header. This fact forms the basis of our decomposition approach. The decomposition goal is to represent the initial GITE-polynomial in a compact form so to optimize a certain cost function, e.g. the number of nodes in the corresponding decision diagram.

Example 2: Figures 1 and 2 show a monolith representation and a compact decomposed representation of the same MOF respectively. The decomposed MOF consists of two parts as follows:

$$\begin{aligned} D_{left} &= \bar{x}_0 x_1 \bar{x}_3 Y_8 + \bar{x}_0 x_1 x_3 Y_6 + \\ &\quad x_0 \bar{x}_1 x_3 Y_{12} + B_0^{left} Y_0, \\ D_{right} &= x_2 x_4 Y_1 + x_2 \bar{x}_4 Y_3 + \bar{x}_2 x_4 Y_1 + B_0^{right} Y_0, \\ D &= D_{left} \circ D_{right}. \end{aligned}$$

In this example, the monolith MTBDD has 17 non-terminal nodes, while the decomposition allows to reduce the number of non terminal nodes to 8.

Notice that the terminals in Fig. 1 correspond to a $\circ$ operation between pairs of terminals in Fig. 2, the $\circ$ in this example stands for a bitwise OR. It is important to emphasize that the compactness of the decomposed MTBDD results from a significant reduction of the total number of distinct leaves (from 11 to 7). ■

The proposed decomposition algorithm is based on grouping of the set of cubes representing the function to a set of *blocks*. The algorithm follows the concept of conventional algebraic decomposition methods. In these methods, a single logic function is treated as a polynomial and is being decomposed by a division operation. Namely, a function F is represented as $F = D \cdot Q + R$ where D, Q and R, are the divisor, quotient and remainder, respectively [4]. In this paper, the decomposition is performed simultaneously on the set of functions that are represented as a single GITE-polynomial. Our algebraic decomposition has the following form:

$$D = GITE(\pi_h, D_1, \dots D_j) \circ R.$$

In this equation GITE determines: a divisor π_h referred to as a *block header*, a quotient $(D_1, \dots D_j)$ where each D_i, $i = 1, \dots, j$, is referred to as a *block fragment*, and a *reminder* R which consists of the remaining cubes that were not included in the block. The partition π_h together with the GITE-polynomials D_i form a block consisting of a subset of the initial set of cubes.

The flow of the decomposition algorithm is presented in Fig. 3. The algorithm is based on the consecutive partitions of blocks into a *block header*, set of *block fragments* and a *reminder*. On each step, these three elements are assigned to a stack. The algorithm starts from the given initial binomial representation and finishes when the stack is empty, which means that all cubes of the initial function are distributed between the blocks of the resulting network. As happens in all expansion based methods (e.g. in the Shannon expansion) at a certain point in time the stack becomes empty and the process ends.

The block header is selected in such a way as to provide minimization of the resulting network. We propose to select the header by taking into account the complexity of its component GITE-polynomials. In what follows we briefly describe how to form the block header by gathering a so called set of prefixes.

Let B be a cube in a block. Each hypercube Bp containing the cube B ($Bp \supseteq B$) is called a *prefix*. The set of all prefixes associated with the cubes of the block defines the block *header*.

The block header can be represented as a monolith whose internal nodes are associated with the prefix variables. The terminal nodes of the monolith correspond to the block fragments representing a GITE-polynomial with the remaining input variables. We call such fragments as $tails$.

Each iteration consists of choosing the prefixes for the current block. The prefixes that form the block header are chosen one by one. Each newly added prefix must be orthogonal (disjoint) to all prefixes accumulated so far.

An iteration starts by preparing a list of candidate prefixes. A candidate prefix can be either an initial cube of the initial MOF or its hypercube. Then, the first (basic) prefix

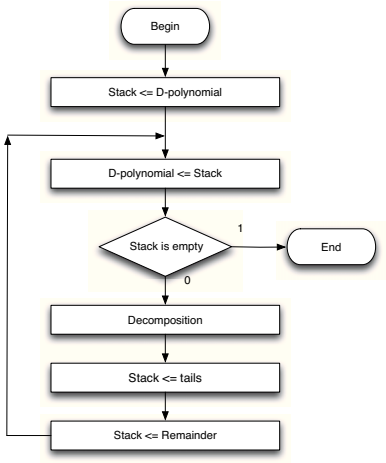

Figure 3. Block diagram of main decomposition procedure

is chosen. The basic prefix defines the set of input variables that will determine the partition π_h. 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, it is imperative to measure the self-orthogonality of the block elements. After choosing the prefix that carries the maximal grade as the basic prefix, the algorithm constructs the block header by adding secondary prefixes. The set of criteria for ranking the candidate secondary prefixes are: a) the number of additional inputs added to the block, b) the number of additional outputs added to the block, and c) the overhead in terms of the percentage of additional literals in the block.

V. Experimental results

The efficiency of the suggested method was evaluated by comparing the compactness of a monolith MTBDD, which corresponds to the initial MOF with the compactness of the proposed decomposed network. In the experiments, the PLA-like representations of the standard combinatorial-circuit benchmarks (LGSYNTH93) were used.

To analyze the experimental results, we define a *block density* - a specific parameter of a block. This parameter corresponds to a number of literals in the block's cubes normalized by the maximal possible number of literal in this block. The success of the decomposition strongly depends on the such defined density. Consequently, the effectiveness of the decomposition can be predicted quite reliably by making some preliminary study of the initial MOF representation.

The experimental results are shown in Tables I and II. Table I lists the benchmarks, for which the decomposition network is simpler than the monolith MTBDD of the initial

Table I
EXPERIMENTAL RESULTS - LOW DENSITY

Title	$\|X\|$	D %	N_{mon}	N_{net}	$ratio$
ALU1	12	18	982	25	0.02
B12	15	29	155	145	0.93
DK48	15	31	3428	58	0.02
DK27	9	34	79	22	0.28
CON1	7	37	16	15	0.94
ALU2	10	39	264	150	0.57
DUKE2	22	40	1435	326	0.23
ALU3	10	42	278	151	0.54
MISEX3C	14	43	10875	705	0.06
WIM	4	50	15	10	0.67
F51M	8	53	255	155	0.61
DK17	10	57	160	55	0.34
APLA	10	64	128	85	0.66
INC	7	79	39	35	0.90

Table II
EXPERIMENTAL RESULTS - HIGH DENSITY

Title	$\|X\|$	D %	N_{mon}	N_{net}	$ratio$
ADD6	12	52	504	731	1.45
RADD	8	57	90	143	1.59
CLIP	9	59	189	376	1.99
Z4	7	61	52	101	1.94
ROOT	8	65	72	134	1.86
SQR6	6	67	63	85	1.35
SQN	7	69	81	116	1.43
MLP4	8	73	240	345	1.44
SAO2	10	73	95	157	1.65
DIST	8	73	125	326	2.61
BW	5	80	25	58	2.32
RD53	5	90	15	53	3.53

MOF. Table II shows the opposite cases. The columns in the tables are as follows: $|X|$ is the number of inputs, D is the benchmark's density D, N_{mon} and N_{net} are the number of nodes in the monolith MTBDD and in the decomposition network. The last column shows the ratio N_{net}/N_{mon}.

The results show that the density 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.

The proposed decomposition, on the other hand, relies upon extracting dense blocks from the given MOF, and treating the sparse remainders and tails separately. Therefore, a sparse MOF can be easily dealt with by splitting them into a network of component MTBDDs. With dense MOFs, choosing suitable blocks is difficult, and arbitrary choices lead to ineffective resulting network.

VI. CONCLUSIONS

A new analytical approach for representation and manipulation of MOFs was presented. The main results can be summarized as follows.

- A GITE operator was introduced. The GITE operator is a generalization of ITE operator on the Boolean domain.
- The concept of GITE-polynomials as a compact analytical representation of GITE-formula was presented.
- The problem of compact representation of multi-output functions was then formulated as a problem of decomposition of GITE-polynomials. A solution to this problem, based on algebra of GITE-polynomials and its properties, was presented.

Experimental results obtained on a number of benchmarks are promising. We believe that the present work will initiate future research of the GITE based representation and its possible applications in logic design. We plan to develop the present concept into directions, theoretical and practical. In the theoretical direction intend to develop fundamental basics of GITE algebra and corresponding decompositions based on this algebra. On the practical direction we intend to apply the developed concept to solving problem of design verification and design for testability of large systems.

REFERENCES

[1] Bahar I., Frohm E., Gaona C., Hachtel G., Macii E., Pardo A., Somenzi F., "Algebraic decision diagrams and their applications," *Proc. of the 1993 IEEE/ACM international conference on Computer-aided design*, 1993, pp. 188 191.

[2] Baier C. and Clarke E., "The Algebraic Mu-Calculus and MTBDDs," *The 5'th Workshop on Logic, Language, Information and Computation (WoLLIC98)*, 1998, pp. 2738.

[3] V. Bertacco and M. Damiani, The disjunctive decomposition of logic functions, *Proc. ICCAD*, 1997, pp. 78-82.

[4] R. K. Brayton, G. D. Hachtel and A. Sangiovanni-Vincentelli, Multilevel logic synthesis, *IEEE Proc.*, Feb. 1990, pp. 264-300.

[5] Brayton R., "The future of logic synthesis and verification," *Logic Synthesis and Verification*, Kluwer Academic Publishers Norwell, MA, 2002, pp. 403 434.

[6] Randal E.Braynt, Graph-based algorithms for Boolean manipulation, IEEE Trans. on Computer, vol. 35, no. 8, pp. 677- 691, August 1986.

[7] Bryant R.E., "Symbolic boolean manipulation with ordered binary decision diagrams," *ACM Computing Surveys*, 24(3), Sept. 1992, pp. 293-318.

[8] Clarke E.M., Fujita M., McGeer P.C., McMillan K., and Yang J., "Multi-terminal binary decision diagrams: An efficient data structure for matrix representation," *IWLS93: International Workshop on Logic Synthesis*, Lake Tahoe, CA, May, 1993, pp. 115.

[9] Hachtel G. D., Somenzi F., *Logic Synthesis and Verification Algorithms*, Kluwer Academic Publisher, 2005.

[10] Hassoun S. and Sasao T., (editors), *Logic Synthesis and Verification*, The Springer International Series in Engineering and Computer Science, Vol. 654, 2002.

[11] Hinman P., *Fundamentals of Mathematical Logic*, AK Peters edition, 2005.

[12] Y. Iguchi, Sasao T., Matasuura M., "Evaluation of multiple-output logic functions using decision diagrams," *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, 2003.

[13] K. Karplus, Representing Boolean functions with if-then-else DAGs, Board of studies in Computer Engineering, Univ. of Calif.-Santa Cruz, CA, Tech. Rep. USCS-CRL-88-29, Nov. 1988.

[14] Levin I., Keren O., Ostrovsky V., Kolotov G. "Concurrent Decomposition of Multi-terminal BDDs," *Proc. of 7th International Workshop on Boolean Problems*, Freiberg, September 2006, pp. 165-169.

[15] I. Levin, O. Keren., "Split Multi-terminal Binary Decision Diagrams," *The 8'th International Workshop on Boolean Problems*, 2008, pp. 161-167.

[16] S. Minato, Fast Factorization Method for Implicit Cube Set Representation, *IEEE Trans. On CAD*, vol. 15, no. 4, April 1996, pp. 377-384.

[17] Nagayama S,, Sasao T., "Compact representations of logic functions using heterogeneous MDDs," *Proceedings of 33rd Conference of Multiple-Valued Logic*, 2003.

[18] J. Mathew, H. Rahamana, A.K Singhb, A. M. Jabirc and D.K Pradhan, "A Galois Field Based Logic Synthesis Approach with Testability", *21st International Conference on VLSI Design*, 2008 pp. 629-634.

[19] Sasao T., *Switching Theory for Logic Synthesis,* Kluwer Academic Publishers, Feb. 1999.

[20] T. Sasao and M. Matsuura, "A method to decompose multiple- output logic functions," *The 41st Design Automation Conference*, June 2-6, 2004, pp.428-433.

[21] Sasao T., Iguchi Y., Matsuura M., "Comparison of Decision Diagrams for Multiple-Output Functions," *Proceedings of International Workshop on Logic and Synthesis*, 2002.

[22] C. Yang M. Ciesielski, BDS: A BDD-Based logic optimization system IEEE Trans. on CAD, vol. 21, no. 7 July 2002.

[23] C. Yang, V. Singhal and M. Ciesielski, BDD Decomposition for Efficient Logic Synthesis, *Proc. ICCD*, pp. 626-631, 1999.

[24] Yanushkevich S. N., Miller D. M., Shmerko V. P., and Stankovic R. S., *Decision Diagram Techniques for Micro- and Nanoelectronic Design Handbook*, CRC Press, 2005.