ORDERED AND SHARED, LINEARLY-INDEPENDENT, VARIABLE-PAIR DECISION DIAGRAMS FOR INCOMPLETELY SPECIFIED FUNCTIONS

Marek Perkowski, Lech Jozwiak †, Rolf Drechsler +, Bogdan Falkowski &,

Portland State University, Dept. of Electr. Engn., Portland, Oregon 97207,
Tel: 503-725-5411, Fax: 503-725-4882, *mperkows@ee.pdx.edu*

† Faculty of Electronics Engineering, Eindhoven University of Technology,
P.O.Box 513, 5600 MB Eindhoven, The Netherlands, *lech@eb.ele.tue.nl*

+ Institute of Computer Science, Albert-Ludwigs-University,
79110 Freiburg in Breisgau, Germany, *drechsle@informatik.uni-freiburg.de*

& Nanyang Technological University,
School of Electrical and Electronic Engineering,
Nanyang Avenue, Singapure 639798.

Abstract— The paper presents a new kind of decision tree: it is based on nonsingular expansions for pairs of variables. Such trees are next used to create Linearly Independent (LI) Decision Diagrams (LI DDs). There are 840 nonsigular expansions for a pair of variables, so number of nodes in such (exact) diagrams is never larger than that of trees with single-variable Shannon, Positive Davio, and Negative Davio expansions. The LI Diagrams are a starting point in a synthesis of multilevel AND/OR/EXOR circuits and can potentially achieve better results than the well-known Pseudo-Kronecker Functional Decision Diagrams. They introduce also other gates than AND and EXOR to synthesis.

I. INTRODUCTION

It is known that the Linearly Independent Logic (LI) can create circuits that are superior to AND/EXOR circuits, but there have been no efficient algorithms for the calculation of nonsingular expansions of LI logic. The approach from [15] only outlined some efficient approaches, but no detailed examples were presented. Paper [5] presented a "fast transform" method to find a single expansion for some of the polarities, but still the problem of selecting the best polarity among all polarities of two-variable nonsingular expansions was not discussed. Therefore, although there exist fast transforms, there is still no method to select a good one among a huge number of such transforms. Applying "fast" transforms successively for all possible polarities would be too inefficient as well.

In this paper we will develop a new representation that is based on the Linearly Independent logic and that can be used in the first stage of logic synthesis - the "technology independent, EXOR synthesis" phase, which is next followed by the "EXOR-related technology mapping" [25, 33, 22, 29], not discussed here.

We will also present how a good-quality polarity of an expansion can be found. Because, however, our method is computationally expensive, we will restrict the method only to expansions for *pairs of variables.*

In section II we introduce the LI Universal Logic Module (node) and present the theory how to compute the expansion data functions for such two-variable nodes in the process of tree creation. Section III introduces the LI Trees, LI Decision Diagrams, and LI Forms. Section IV presents approximate algorithms for generation of various LI Decision Diagrams for multi-output functions. Section V describes algorithms to select, for a given set of expansion variables, good polarities for nodes at a given level of the diagram. One of them is for incompletely specified functions. This algorithm becomes more efficient when the function is weakly specified. Exhaustive and non-exhaustive algorithms

AB \ CD	00	01	11	10
00	0	1	1	1
01	0	1	0	1
11	0	1	0	0
10	0	0	0	1

Figure 1: *Function $f(A, B, C, D)$ to Example 2.1*

are presented. Section VI concludes the paper and mentions open problems.

II. THE NON-SINGULAR EXPANSION

To introduce the ideas of Linearly Independent (LI) logic and nonsingular expansions, we will solve a simple example first.

Example 2.1. Given is function $f(A, B, C, D)$ from Figure 1. Let us assume that we want to expand this function with respect to variables $\{A, B\}$.

Let us first find the standard expansion with respect to cofactors on variables A, B. The first expansion will use the, computable from function $f(A, B, C, D)$, standard cofactors: $f_{\overline{A}\ \overline{B}}(C, D)$, $f_{\overline{A}\ B}(C, D)$, $f_{A\ \overline{B}}(C, D)$, $f_{AB}(C, D)$.

$f(A, B, C, D) = \overline{A}\ \overline{B}\ f_{\overline{A}\ \overline{B}}(C, D) \oplus\ \overline{A}\ B f_{\overline{A}\ B}(C, D) \oplus\ A\ \overline{B} f_{A\ \overline{B}}(C, D) \oplus\ \ A\ B\ f_{AB}(C, D)$

$= \overline{A}\ \overline{B}\ f(A, B, C, D) \mid\ _{A=0,B=0} \oplus\ \overline{A}\ B\ f(A, B, C, D) \mid\ _{A=0,B=1} \oplus\ A\ \overline{B}\ f(A, B, C, D) \mid\ _{A=1,B=0} \oplus\ \ AB$ $f(A, B, C, D) \mid\ _{A=1,B=1}$

$= \overline{A}\ \overline{B}\ (C + D) \oplus\ \overline{A}\ B(C \oplus D) \oplus\ A\ \overline{B}(C\overline{D}) \oplus\ \ AB\ (\overline{C}D)$

The second expansion, called a *nonsingular expansions*, will use unknown functions $SF_i(C, D)$:

$$f(A, B, C, D) = (A + B)\ SF_{A+B}(C, D) \oplus\ \overline{B}\ SF_{\overline{B}}(C, D) \oplus\ \overline{A}\ SF_{\overline{A}}(C, D) \oplus\ SF_1(C, D) \qquad (2.1)$$

The basis of the functions on variables A and B for which we are expanding is here arbitrarily selected as: $f_{A+B} = A + B$, $f_{\overline{B}} = \overline{B}$, and $f_{\overline{A}} = \overline{A}$, and $f_1 = 1$.

In order to calculate the unknown functions $SF_i(C, D)$ we will compare the expansions for all possible combinations of values of A and B. This will lead to a set of linear logic equations, which after solving will give the values to the unknown functions $SF_i(C, D)$.

Thus comparing the two expansions for $f(A, B, C, D)$ we have

$\overline{A}\ \overline{B}\ (C + D) \oplus\ \overline{A}\ B(D \oplus C) \oplus\ A\ \overline{B}(C\overline{D}) \oplus\ \ AB\ (D\overline{C})$

$= (A + B)\ SF_{A+B}(C, D) \oplus\ \overline{B}\ SF_{\overline{B}}(C, D) \oplus\ \overline{A}\ SF_{\overline{A}}(C, D) \oplus\ SF_1(C, D)$

By substituting in the above equation A = 0, B = 0, we get the following equation 1 for cofactor $f_{\overline{A}\ \overline{B}}(C, D)$:

$(C + D) = f_{\overline{A}\ \overline{B}}(C, D) = SF_{\overline{B}} \oplus SF_{\overline{A}} \oplus SF_1.$

By substituting A = 0, B = 1, we get the following equation 2:

$(C \oplus D) = f_{\overline{A}\ B}(C, D) = SF_{A+B} \oplus\ SF_{\overline{A}} \oplus SF_1$

By substituting A = 1, B = 0, we get the following equation 3:

$(C\overline{D}) = f_{A\ \overline{B}}(C, D) = SF_{A+B} \oplus\ SF_{\overline{B}} \oplus SF_1.$

By substituting A = 1, B = 1, we get the following equation 4:

$(\overline{C}D) = f_{AB}(C, D) = SF_{A+B} \oplus SF_1.$

The last four equations for cofactors $f_{A^i\ B^j}(C, D)$ can be rewritten to the matrix form of equation:

$FV = M \times CV =$

$$= \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} SF_{A+B}(C, D) \\ SF_{\overline{B}}(C, D) \\ SF_{\overline{A}}(C, D) \\ SF_1(C, D) \end{bmatrix} = \begin{bmatrix} f_{\overline{A}\ \overline{B}}(C, D) \\ f_{\overline{A}\ B}(C, D) \\ f_{A\ \overline{B}}(C, D) \\ f_{A\ B}(C, D) \end{bmatrix}$$

Therefore $CV = M^{-1} \times FV =$

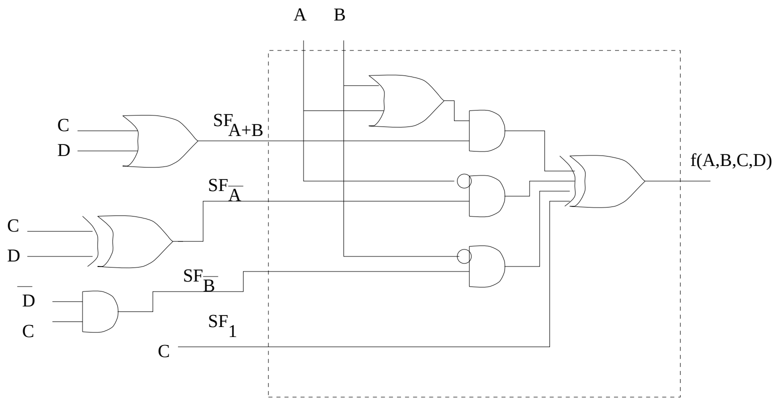

Figure 2: *Universal nonsingular expansion module to Example 2.1*

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} f_{\overline{A}\,\overline{B}}(C,D) \\ f_{\overline{A}\,B}(C,D) \\ f_{A\,\overline{B}}(C,D) \\ f_{A\,B}(C,D) \end{bmatrix} = \begin{bmatrix} SF_{A+B}(C,D) \\ SF_{\overline{B}}(C,D) \\ SF_{\overline{A}}(C,D) \\ SF_{1}(C,D) \end{bmatrix}$$

Now, that the unknown data input functions SF_i have been found, they are substituted into the nonsingular expansion (2.1) to create the expansion formula (2.2). The coefficients $SF_i(C,D)$ are taken from the above vector CV.

From Figure 1, the function F be represented by a vector $FV^T = [(C+D) \quad (C \oplus D) \quad (C\overline{D}) \quad (\overline{C}D)\ ]$.

$$CV = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} (C+D) \\ (C \oplus D) \\ (C\overline{D}) \\ (\overline{C}D) \end{bmatrix} = \begin{bmatrix} SF_{A+B} \\ SF_{\overline{B}} \\ SF_{\overline{A}} \\ SF_{1} \end{bmatrix} = \begin{bmatrix} (C+D) \oplus (C \oplus D) \oplus (C\overline{D}) \oplus (\overline{C}D) \\ (C\overline{D}) \oplus (\overline{C}D) \\ (C \oplus D) \oplus (\overline{C}D) \\ (C+D) \oplus (C \oplus D) \oplus (C\overline{D}) \end{bmatrix} = \begin{bmatrix} (C+D) \\ (C \oplus D) \\ (C\overline{D}) \\ (C) \end{bmatrix}$$

Then $f(A,B,C,D) = (A+B)(C+D) \oplus \overline{B}(C \oplus D) \oplus \overline{A}(C\overline{D}) \oplus 1(C)$ (2.2).

Concluding, we were able to expand the original function with respect to four functions on variables A, B. We will call these functions (in our case, functions $A+B$, $\bar{A}$, $\bar{B}$, and 1), the Linearly Independent Functions, since the columns corresponding to them in matrix M are linearly independent with respect to the operation of EXOR-ing of columns. For comparison, the same function was expanded for classical AND/EXOR logic in [17].

Observe, that a unique expansion was possible because the set of equations had exactly one solution, which is equivalent to matrix M being nonsingular. Hence, the name "nonsingular" used for our expansion. Moreover, our method of solving this example can be generalized to arbitrary sets of Linearly Independent functions.

This nonsingular expansion with functional coefficients is realized using an universal logic module with control variables A, B (illustrated in Figure 2). This way, for the set of LI functions $\{\overline{A}, \overline{B}, (A+B), 1\}$, there exists only one nonsingular expansion specified by its matrix M^{-1}. The module from Figure 2 is a generalization of modules for Shannon Expansion (multiplexer) and Davio Expansion (AND/EXOR gate).

Let us observe that formula (2.2) describes only one of the 840 nonsingular expansions for variables A, B [14, 5]. In addition, any set of two, three, or four variables out of set $\{A, B, C, D\}$ can be selected for the first level expansion. So, there are very many different trees representing successive expansions. Even if the problem of fast calculating of a single expansion were solved, the more important problem remains: how to select the best one of all the nonsingular expansions (or nonsingular expansions of certain kind). This problem is difficult, because there are very many of such expansions [14]. Our approach will be to modify some methods known the from Reed-Muller logic. First, we will formally define the representations of Boolean functions that we will be next used in functions' optimization.

Linearly Independent logic [13, 14, 15] allows to uniquely derive data functions SF_i for universal expansion modules from the original function $f(x_1, x_2, ..., x_n)$, assuming given sets of linearly independent functions of m variables ($m \leq n$). We will review these methods briefly below. We create a $2^m \times 2^m$ matrix M with rows corresponding to minterms (for a subfunction f_i with m variables we have 2^m columns). The columns correspond then to some Boolean functions f_i of m variables. A 1 in the intersection of a column "i" and row "j" means that minterm "j" is in function "i". The set of columns should be linearly independent with respect to EXOR operation (i.e. columns are bit-by-bit exored). If a set of 2^m columns is linearly independent then there exists one and only one matrix M^{-1}, inverse to M

with respect to the exoring operation. In such case, the family of Boolean functions f_i corresponding to columns will be called the "linearly independent family of Boolean functions" (or set of LI (Boolean) functions, or LI set). We will call them *LI functions,* for short. The matrix will be called a *"nonsingular matrix"*.

Let us denote the vector of cofactors with respect to variables $\{x_1, ...x_m\}$ by FV. CV denotes the vector of coefficients for some given canonical form represented by nonsingular M. Given is an arbitrary linearly independent (LI) set of 2^m Boolean functions f_i of m variables. This set can be represented as a $2^m \times 2^m$ nonsingular matrix M with basis functions f_i as columns, $i = 0, ..., 2^m - 1$.

Theorem 1. Given is a function F$(x_1, ..., x_m, ..., x_n)$ such that the set of input variables $\{x_1, ..., x_n\}$ includes properly the set $\{x_1, ..., x_m\}$. There exists an unique expansion

$$F(x_1, ..., x_n) = f_0(x_1, ..., x_m)SF_0(x_{m+1}, ..., x_n)$$
$$\oplus f_1(x_1, ..., x_m)SF_1(x_{m+1}, ..., x_n) \oplus ... \oplus f_{2^n-1}(x_1, ..., x_m)SF_{2^n-1}(x_{m+1}, ..., x_n) \quad (2.3)$$

where functions f_i are the given basis LI functions of m variables, and the coefficient functions (called also the "data input functions") SF_i of the remaining input variables are determined from the coefficient vector $CV = M^{-1} \times FV$, where $FV(x_{m+1}, ..., x_n)$ is a vector of all 2^m cofactors of F with respect to variables from the set $\{x_1, ..., x_m\}$.

Proof. Proof is a generalization of the method of solving EXOR logic equations from the above example. The method as above would work for any basis LI functions as columns of nonsingular matrix M.

We will call (2.3) the nonsingular expansion with functional coefficients. $f_i(x_1, ...x_m)$, $i = 0, ...2^{n-1}$. This is a unique expansion for the set of variables $x_1, ..., x_m$ and the set of functional coefficients. This means, the data functions on variables $x_{m+1}, ..., x_n$ for given basis LI functions of matrix M are uniquely determined by expansion (2.3).

III. LINEARLY INDEPENDENT TREES, DECISION DIAGRAMS AND FORMS.

The (standard) Kronecker Tree has levels that correspond to single (input) variables. Only one of three types of binary expansions (S,pD, and nD) is used in every level of the tree [28]. Kronecker Trees are quite useful to obtain high-quality multi-level circuits. They can be also generalized to Pseudo-Kronecker Trees, [25], that lead to even better circuits. The decision diagrams are created from such trees by applying reductions to nodes of such trees.

It can be observed, however, that one may allow to have nodes in the trees for **sets** of variables, instead for single variables only. These sets will be called *blocks.* The concept of a tree is now generalised, and the tree is no longer a binary tree, but has *multi-variable* nodes. Moreover, **arbitrary nonsingular expansions** are now allowed in the nodes. The number of such expansions is very large, even for small blocks of grouped variables. For instance, let us observe that in the case of two successive levels of a (standard) Kronecker Tree, there are three nodes for a pair of variables, and each node can have S, pD or nD expansion. Thus, the total number of expansions for a pair of variables in the Kronecker tree is $3^3 = 27$. In contrast, there are 840 various nonsingular expansions for a pair of variables in a LI tree.

This new type of a tree will be called the *Linearly Independent Kronecker Tree,* (LIKT). It is a special case of LI Tree, which means, a tree that uses nonsingular expansion in nodes.

Definition 2. The **LI Kronecker Tree** (LIKT) is a tree with multi-variable expansion nodes, created as follows:

1) The set of all n input variables is partitioned into a set of disjoint and nonempty subsets S_j such that the union of all these subsets forms the initial set. (This is a partition of the set of input variables). The subsets will be called *blocks.* If each block includes just a single variable, the tree reduces to the special case of a KRO Tree. If there is only one block that includes all variables, the tree reduces to the special case of a nonsingular form [13, 14, 15].

2) The sets (blocks) are ordered, each of them corresponds to a level of the tree.

3) For every level, if the block involves a single variable, S, nD, or pD is selected for its nodes. If the block is multi-variable, one nonsingular expansion polarity is selected for the nodes of the tree at the level corresponding to this block.

In LIKTs, the set of all input variables is thus partitioned to several disjoint blocks, each corresponding to a level of the tree. For every block with a single variable, the corresponding expansions are S, pD and nD. For a block with two variables there are 840 nonsingular expansions. Therefore, for the two-variable nodes there are 840 types of nodes, called LI(2) nodes (expansion types). They will be denoted by LI(2)-$\{n_{1,1}, n_{2,1}, n_{3,1}, n_{4,1}\}$, ..., LI(2)-$\{n_{1,840}, n_{2,840}, n_{3,840}, n_{4,840}\}$, or as their polarity matrices M. Thus, in LI(2)-$\{n_{1,1}, n_{2,1}, n_{3,1}, n_{4,1}\}$, the number $n_{j,i}$ is a natural number corresponding to the binary vector of the $j-$th colum of the i-th matrix M, which is read with bottom row as the least significant bit. In this way, the (expansion polarity) matrix

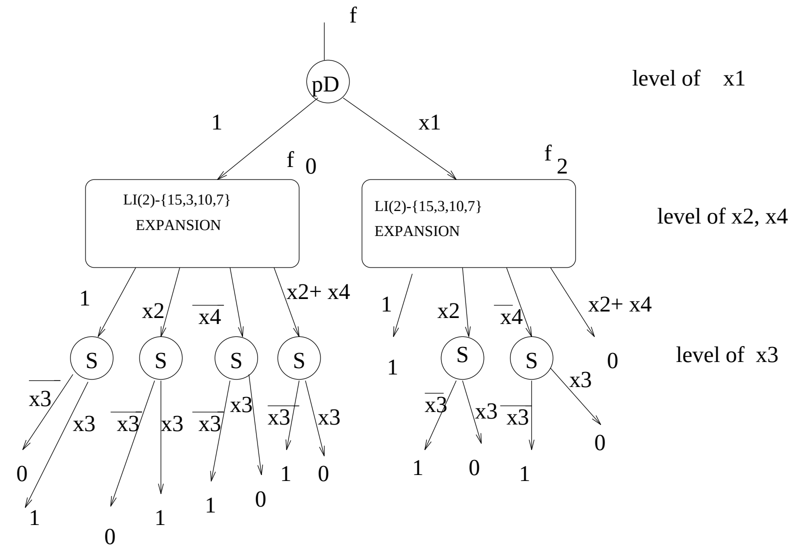

Figure 3: *Example of an LI Kronecker Tree*

$M = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$ is represented as a set of 4 natural numbers, each corresponding to one LI function, and denoted by LI(2)-{15,3,10,7}.

Definition 3. Linearly Independent Forms (LI Forms) are obtained by flattening the LI Trees, i.e. finding all ordered product terms obtained by multiplication of paranthesized expressions corresponding to AND/EXOR trees, using recursively the rule $a(b \oplus c) = ab \oplus ac$.

The LI Forms are no longer two-level forms, as is the case in AND/EXOR flattened forms. They have three levels, the first (from output) level are EXOR gates, the second are AND gates and the third are *arbitrary* Boolean functions defned on blocks of variables. The LI Forms are implemented in a tree-level circuit called a LI PLA.

Definition 4. LI Decision Diagrams (LI DDs) are created by: (1) combining isomorphic nodes of any kind, (2) performing standard Ordered Kronecker Functional Decision Diagram (OKFDD) transformations [3] on S, pD and nD nodes, (3) performing generalizations of standard Ordered Kronecker Functional Decision Diagram (OKFDD) transformations [3] on multi-variable nodes. These generalizations remove any node that evaluates to its single argument.

We will say that the node evaluates to a single argument if after substituting constants and a single variable value H_i to the function realized by this node, after propagation of constants, the function evaluates to H_i. Observe that only multivariable nodes that have only logic constants and the same signal H_i as arguments should be evaluated.

For instance:

Formula $\overline{a}\ \overline{b}H_1 \oplus \overline{a}\ bH_1 \oplus a\ \overline{b}H_1 \oplus a\ bH_1$ evaluates to H_1.

Formula $\overline{a}\ b0 \oplus a\ b0 \oplus \overline{b}H_2 \oplus bH_2$ evaluates to H_2.

Formula $ab0 \oplus a0 \oplus b0 \oplus H_3$ evaluates to H_3.

This method is a generalization of simplification rules for S, pD and nD nodes that are applied to create the OKFDDs.

Definition 5. The *Linearly Independent Kronecker DDs* are created from LI Kronecker Trees as described in Definition 4.

Definition 6. The *Linearly Independent Kronecker Forms* are the forms created by flattening of the LI Kronecker Trees, (or the *Linearly Independent Kronecker DDs*).

Example 3.1. Figure 3 illustrates an example of the LI Kronecker Tree. The first level of the tree has Positive Davio

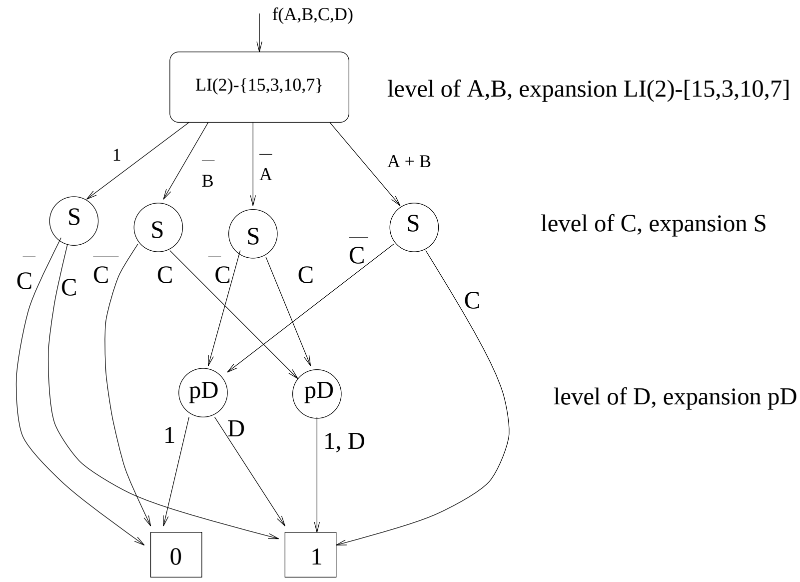

Figure 4: *A LI Kronecker Decision Diagram for function from Example 2.1.*

expansion for variable x_1, the second level has LI(2)-{15,3,10,7} expansion for the set of variables { x_2, x_4 }, and the third level has Shannon expansions for variable x_3.

The expansion of the node LI(2)-{15,3,10,7} is described by the following formula:

$f_0(x_2,x_3,x_4) = SF(f_0)_1(x_3) \oplus x_2SF(f_0)_{x_2}(x_3) \oplus \overline{x_4}SF(f_0)_{\overline{x_4}}(x_3) \oplus (x_2+x_4)SF(f_0)_{x_2x_4}(x_3)$

where notation $SF(f)_i(X)$ denotes function SF_i, with arguments from the set X of variables, applied to argument function f.

This formula is a specialization of nonsingular expansion (2.3) applied to cofactor function $f_0(x_2,x_3,x_4)$ as $F(x_1, ...x_n)$, with expansion variables x_2, x_4 in linearly independent functions. Subfunctions SFi of the remaining variables are calculated for the cofactor function f_0 (so they are denoted as functions $SF(f_0)_1$, $SF(f_0)_{x_2}(x_3)$, $SF(f_0)_{\overline{x_4}}(x_3)$, $SF(f_0)_{(x_2+x_4)}(x_3)$ in this particular LI(2)-{15,3,10,7} expansion).

Example 3.2. A LIK Decision Diagram created from a LIKT corresponding to the expansion from Example 2.1 (and Figure 2), is shown in Figure 4.

Definition 7. A *Single-Polarity Nonsingular Expansion* for a multi-output function is a vector of Nonsingular expansions for its component single-output functions, all of these expansions have the same polarity.

Definition 8. A *Multi-Polarity Nonsingular Expansion* for a multi-output function is a vector of nonsingular expansions for its component single-output functions, each of them can have different polarity.

Thus, for a two-input, three-output function, the *Polarity Vector of a Single-Polarity Nonsingular Expansion* has 4 natural numbers, and the *Polarity Vector of a Multi-Polarity Nonsingular Expansion* has 3 * 4 = 12 natural numbers. Let us observe, that in the special case of multi-output GRM expansions, Definition 7 is in accordance with the definition from [34], while Definition 8 is in accordance with the definition from [2]. Obviously, the minimal DD (or minimal form) obtained from Definition 8 is smaller than the one obtained from Definition 7. There are, however, some advantages of considering representations created according to Definition 7; faster algorithms, and simpler circuits to create the polarity-defining functions. In case of AND/EXOR forms, these circuits are only invertors in the input level so they practically don't count to the cost of realization. However, for general LI circuits, these circuits constitute higher fractions of the total costs, so it is reasonable to assume that they are the same for all the output functions.

Definition 9. The *LI Pseudo-Kronecker Tree* is defined similarly as the LI Kronecker Tree; the only difference is that in every level, any combination of expansions can be used.

The relation between the LI Pseudo-Kronecker Tree and the LI Kronecker Tree is exactly the same as the relation between the Pseudo-Kronecker Tree and the Kronecker Tree. Similarly as the Linearly Independent Kronecker Forms

abc \ def	000	001	011	010	110	111	101	100
000	10	11	10	11	10	10	10	10
001	10	11	10	11	10	10	10	10
011	00	11	00	11	00	10	00	11
010	00	10	10	10	10	01	10	01
110	10	10	00	10	00	00	00	00
111	10	00	10	00	10	00	10	10
101	00	11	00	11	01	10	01	10
100	00	11	10	11	11	00	11	00

f, g

Figure 5: *Function to Example 3.3.*

and the LI Kronecker Decision Diagrams, the *LI Pseudo-Kronecker Forms* and the *LI Pseudo-Kronecker Decision Diagrams* can be defined. Because in case of Pseudo-Kronecker trees every node can have a different polarity, Definition 7 does no longer apply to Pseudo- type representations of multi-output functions.

Definition 10. A single-output Kronecker Tree is specified by a *Single-Output Polarity List*
{ [*variable_block_1*, *expansion polarity_1*] ..... [*variable_block_r*, *expansion polarity$_r$* }, that associates polarities with blocks.

A multi-output Kronecker Tree for a function with k outputs is specified by a *Multi-Output Polarity List*
{ [*variable_block_1*, $expansion\ polarity_{1,1}, ..., expansion\ polarity_{1,v}, ..., expansion\ polarity_{1,k}$],
......
[*variable_block_r*, $expansion\ polarity\_r, 1, ..., expansion\ polarity_{r,v}, ..., expansion\ polarity_{r,k}$], }
that associates polarities with blocks, for each output function separately.

Such lists do not exist for Pseudo Kronecker Trees because of the total freedom of expansion selection for levels.

The name LI Trees will be generic to all kinds of LI trees (LI Kronecker, LI Pseudo-Kronecker, LI Mixed, LI Ordered, LI Free, etc., [18]).

Definition 11. By a *Shared Ordered Linearly Independent Decision Diagram* (SOLIDD) we will understand an LI Decision Diagram that is Shared and Ordered in the same sense as BDDs are shared and ordered. A *Shared Linearly Independent Kronecker Decision Diagram* (SLIKDD) is a Shared LI Kronecker DD. A *Shared Linearly Independent Pseudo-Kronecker Decision Diagram* (SLIPKDD) is a Shared LI Pseudo-Kronecker DD.

Example 3.3. Given is a two-output function from Figure 5. The LI Pseudo-Kronecker tree for blocks {a,b}, {c,d}, {e,f} is shown in Figure 6. A Shared, Ordered LI Pseudo-Kronecker Decision Diagram created from this tree is shown in Figure 7. To enable the reader to analyze the final solutions, in this diagram we showed the internal structure of nodes corresponding to the expansions from Figure 6. The multi-level circuit obtained from the SOLIPKDD after the propagation of constants is shown in Figure 8. The circuit is drawn in a way that enables the reader to observe the effect of propagation of constants. Let us note, that the two EXOR gates that have c and d as inputs can be factored out, and also the two OR gates with c and d as inputs can be factored, thus saving two gates. The three-level LI PLA for flattened SOLIPKDD (after constants propagation) is shown in Figure 9.

From now on, we will assume that each block has only two variables. The respective representations will be called *LI Variable-Pair Trees, LI Variable-Pair Decision Diagrams,* and *LI Variable-Pair Forms*, respectively. Although in this paper we discuss LI trees for only two variables in each block, all concepts and algorithms can be easily expanded to blocks of arbitrary size.

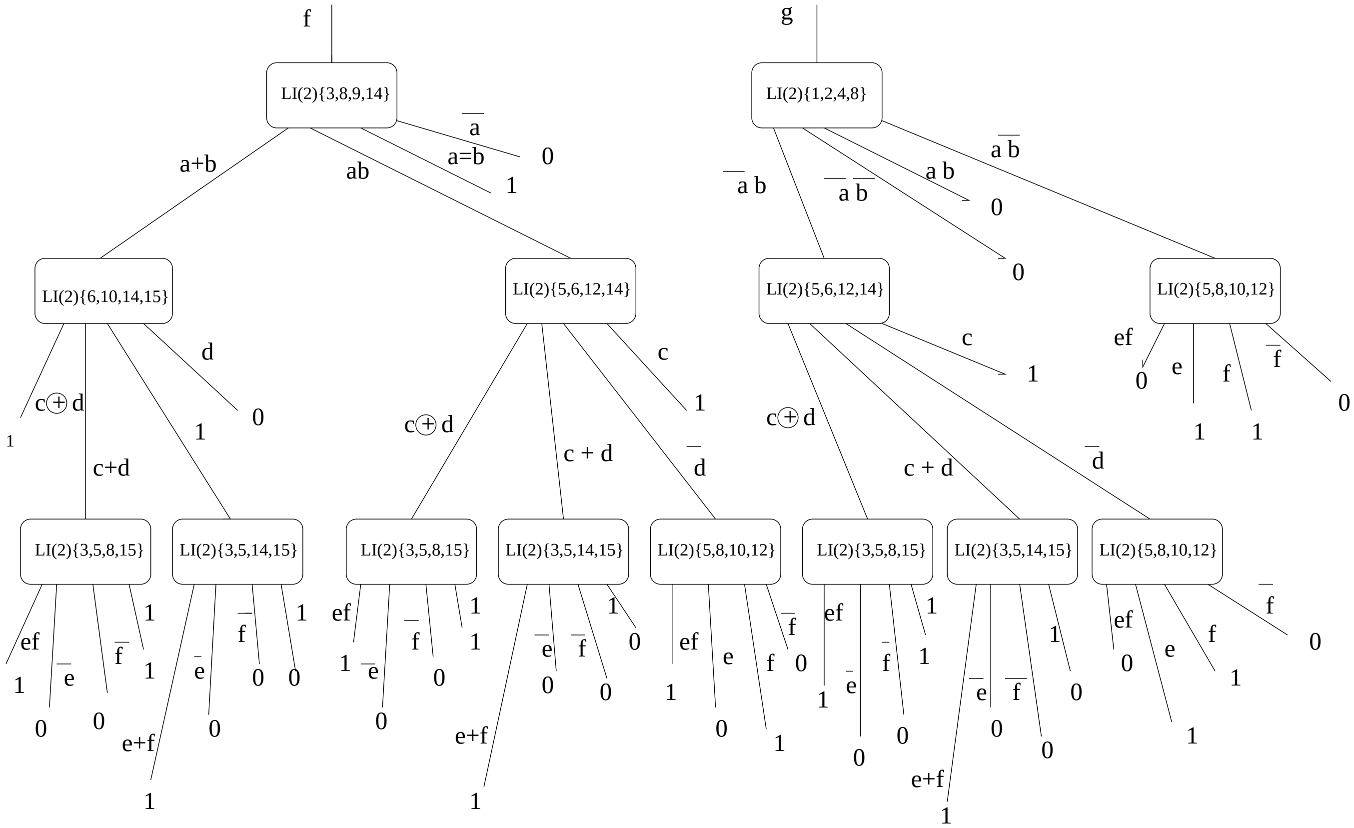

Figure 6: *A LIPKT for blocks* $\{a,b\}$,$\{c,d\}$,$\{e,f\}$ *to Example 3.3.*

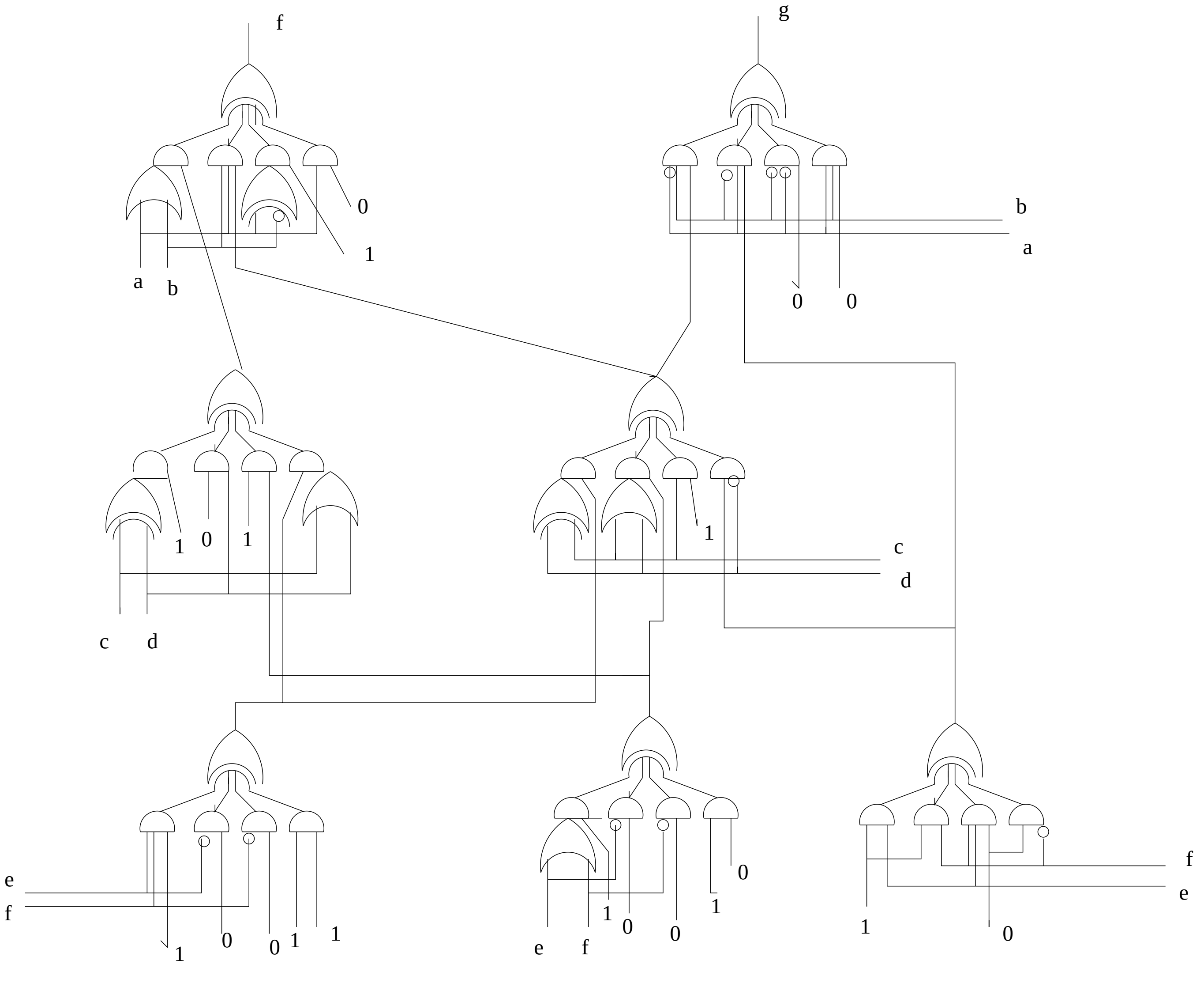

Figure 7: *A LI Pseudo-Kronecker Decision Diagram (SOLIPKDD) for blocks* $\{a,b\}$,$\{c,d\}$,$\{e,f\}$ *to Example 3.3, drawn, with expansion nodes substituted by respective universal module circuits, in order to explain how the final circuit from Fig. 8 is obtained from the diagram.*

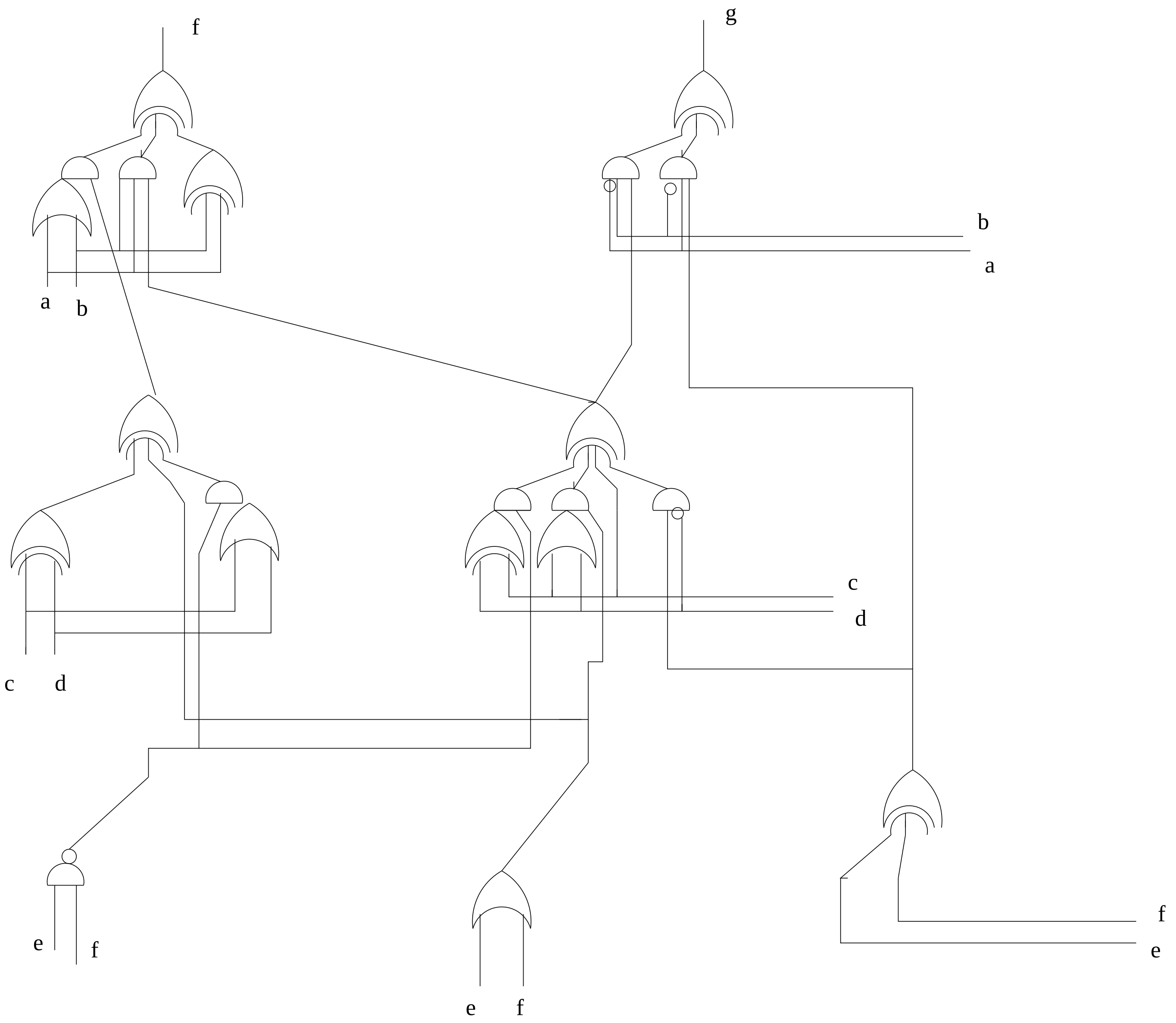

Figure 8: *A multi-level circuit to Example 3.3 obtained from the Shared LI Pseudo-Kronecker DD after substituting circuits of universal modules to nodes of the diagram and propagation of constants.*

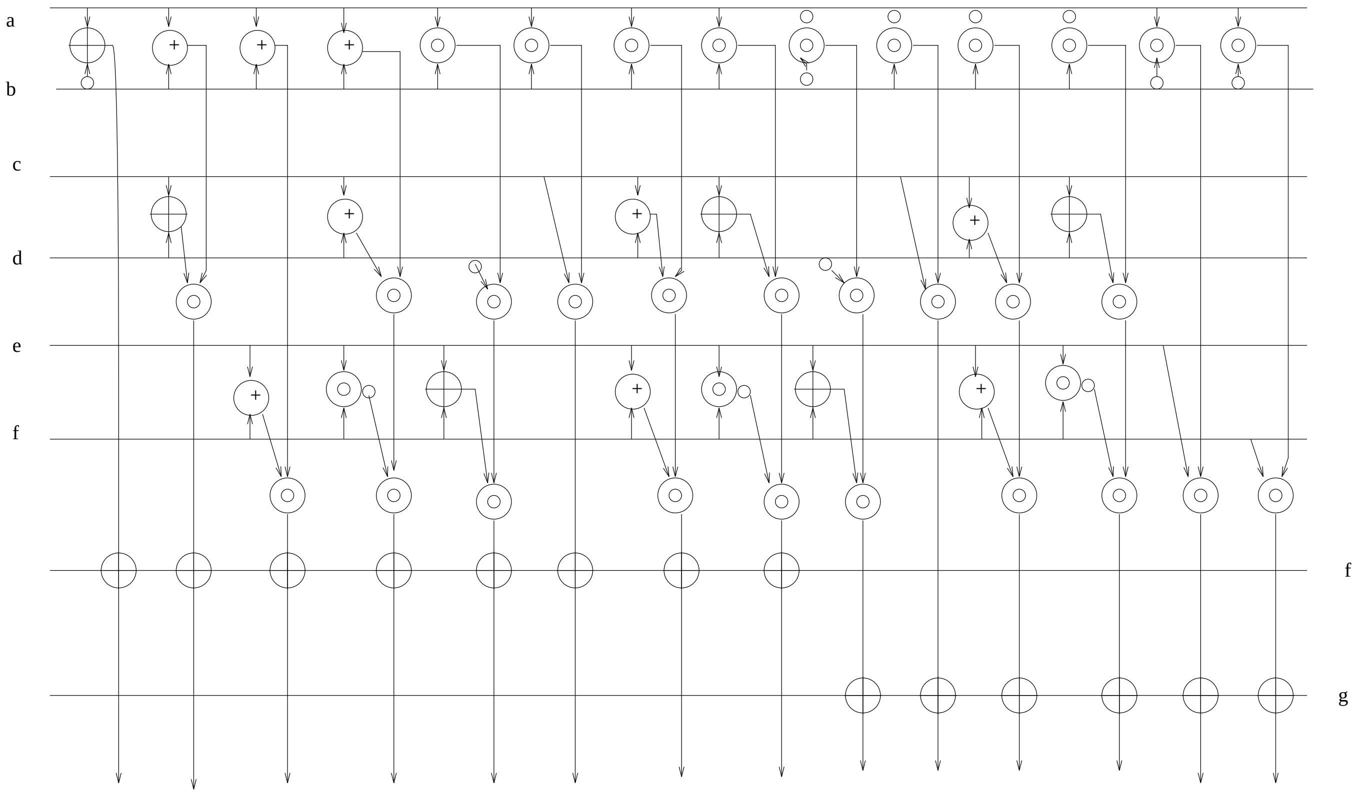

Figure 9: *A three-level LI PLA to Example 3.3 obtained from the SOLIPKDD from Fig. 7.*

IV. ALGORITHM FOR THE GENERATION OF SLIKDDS.

In this paper we will consider the LI circuits designed according to Definition 7, but future considerations will be also given to circuits realized from Definition 8 as well. For multi-output functions the algorithm generates a shared diagram (a Directed Acyclic Graph or DAG) which can be in particular case a forest of trees.

Property 1. In case of SLIKDDs, the order of blocks in the expansions has no influence on the cost of the flattened form that would be found from this diagram.

The same property exists for Kronecker Trees, where the order of expansion variables is irrelevant to the cost of Kronecker expressions obtained from their flattening, and in contrast to the Pseudo-Kronecker Trees. This property does not hold for the SLIPKDDs. The diagrams for all single-output functions are created together, level-by-level from their roots (outputs). In every level, all possible expansions of a block are applied (or some of their subsets). Thus, for a two-variable block, 840 nonsingular expansions are generated in the exhaustive method from section V. The nodes of the next level that correspond to the same function are calculated only once in the cost function. Each level of the multi-output diagram corresponds to a block with 2 elements. The best expansion found by the Polarity Selecting Algorithm for a level is next applied to all nodes from the level of the multi-output diagram. The next-level nodes that correspond to the same function are combined to single nodes.

Local Optimization Algorithm for Generation of the SLIKDD for a completely specified multi-output function.

The local optimization algorithm to create the SLIKDD for multi-output function is the following:

1. The set of input variables is partitioned to pairs (for simplification it is assumed here that all blocks have two elements). Make an ordered list of blocks.
2. Take the first block.
3. Use one of the algorithms from the next section to calculate the best polarity of the expansion for this level of the diagram.
4. Apply this expansion to all nodes from this level, perform reductions from Definition 4 and combine all the isomorphic nodes from the next level to single nodes (tautology check).

5. Repeat steps 3 and 4 for all remaining blocks from the list of blocks.

This algorithm is a straightforward generalization of the algorithm from [29]. It can be also easily further extended to incompletely specified multi-output functions and to diagrams with inverted edges. Because of a variety of applications, we developed several algorithms to be used in step 3. They are exhaustive or not, for complete and incomplete functions, and for either all nonsingular expansions or only some of their subfamilies.

The order of blocks in LI Kronecker representations has no influence on the realization costs of forms made by their flattening. It is not so for the pseudo-Kronecker LI representations, so all possible permutations of blocks should be calculated, thus running the algorithm repeatedly for all possible orders of blocks. Which is, however, not possible practically in case of larger functions.

V. ALGORITHMS FOR THE SELECTION OF THE BEST NONSINGULAR EXPANSION POLARITY

One can select any polarity of nonsingular expansion and create a diagram (or a tree) for this polarity. In case of multi-output functions, a sequence of polarities of blocks is created, represented as a Multi-Output Polarity List. The expansion could be thus calculated, as it was illustrated in section III. If there exists a fast transform, it should be applied, instead of inverting matrix M [5]. However, the quality of such solution can not be very good, because of the random nature of selecting polarities. (the same problem as in Functional Decision Diagrams and Positive Polarity Reed-Muller Forms). Therefore, other methods must be looked for, unless for some reasons the good polarities are given, or can be guessed. We will use the analogy to standard Reed-Muller logic.

One of interesting concepts of Reed-Muller logic are the butterfly diagrams that allow to create all fixed polarity expansions by transforming from polarity to polarity, and doing this just by incremental exoring of some terms from the forms. This way, all forms of certain type are systematically created without even creating their matrices M and without calculating their inverse matrices M^{-1}. In another similar approach, we applied the concept of Gray-code ordering of all Generalized Reed-Muller polarities in an algorithm to find the exact minimum GRM form [34]. We will introduce similar ideas for the new forms.

Property 2. The following rule is true.

RULE BR: $f_1(x_1,x_2)SF_2(x_3,...,x_n) \oplus f_3(x_1,x_2)SF_4(x_3,...,x_n) =$

$$[f_1(x_1,x_2) \oplus f_3(x_1,x_2)]SF_2(x_3,...,x_n) \oplus f_3(x_1,x_2)[SF_2(x_3,...,x_n) \oplus SF_4(x_3,...,x_n)]$$

where $f_1(x_1,x_2)$ and $f_3(x_1,x_2)$ are arbitrary LI functions, and $SF_2(x_3,...,x_n)$ and $SF_4(x_3,...,x_n)$ are the corresponding to them data input functions (DI functions, for short). [1].

Property 3. Any nonsingular expansion can be obtained by a repeated application of Rule BR to pairs of functions $[\ f_1(x_1,x_2),\ SF_2(x_3,...,x_n)\ ]$, $[\ f_3(x_1,x_2),\ SF_4(x_3,...,x_n)\ ]$.

This way, rule BR describes simultaneous EXOR-ing of columns in matrix M and corresponding columns in M^{-1}. But how to select the pairs of functions?

Property 4. In matrix M, as well as in matrix M^{-1}, any column can be replaced by a linear combination of itself with other columns. Thus, any *polarity expansion* can be obtained by a repeated application of the basic rule BR to certain selected columns.

The following approaches are possible in LI logic.

A1. To find all nonsingular expansions for a function. This problem is important theoretically and should be solved in order to create exact algorithms and to enumerate all solutions. However, practically it is of less importance, since the number of all nonsingular expansions is very high.

A2. To find as many as possible of the nonsingular expansions but in a very efficient way. This would allow to create fast multi-level minimizers for arbitrary functional bases corresponding to these subsets.

A3. To find all nonsingular expansions for some selected families of expansions (such families are defined by matrices M having some special interesting properties).

Problem A3 is what researchers have been doing for the past 40 years in Reed-Muller logic (AND/EXOR logic), which is a proper subset of the general LI Logic. We proposed different such families for the general LI logic in [15, 14, 13, 22, 5] There exist classes of expansions which are practical for synthesis to Fine Grain FPGAs, but for which there are no fast transforms. Many fast transforms were identified for various LI classes [5], unfortunately for some of them the applications are not known to us now.

[1] it is easy to verify that this rule is true by simple Boolean manipulations comparing its left and right sides

Another important observation is that until now, the following classes of families have been of interest in general LI Logic:

C1. The class for which both fast forward and fast inverse recursive transforms exist [5]. This is the best class of families to create efficient algorithms. An open problem however remains, is this class practically useful?

C2. The class for which fast forward recursive transforms exist. It is a wider class than class 1, therefore there is a better chance that such transforms exist for some interesting families of expansions.

C3. Families that are important practically, such as those used in Generalized AND/OR/EXOR PLAs and can be mapped to Motorola, ATMEL or XILINX Fine Grain FPGAs. Even if general solutions will be not found for these families, it will be very practical to find their corresponding specific solutions for limited numbers of input variables because the cells have small numbers of inputs.

C4. The family of all nonsingular expansions. Even for two variables, this family includes very many expansions that have no any fast recursive transforms.

Various butterfly diagrams related to the above approaches and classes of families will be considered below.

A. Exhaustive Algorithm based on the Pre-Computed Butterfly Diagram for Completely Specified Functions

Even if in general there is no recursive way to define the universal Butterfly-like diagram for *arbitrary* LI matrix, a specific diagram can be once created for a set of variables with *certain number* of elements and for any set of expansion polarities.

This diagram can be stored in memory, and next used for evaluations for each particular function of the respective number of variables. We will call this a "pre-computed" Butterfly diagram.

Our exhaustive algorithm goes through all polarities. The set of all polarities is created as levels (rows) in a butterfly-like diagram from Figure 10 (for the lack of space, only first few levels are shown). Small K-maps correspond to some LI functions $f(x_1, x_2) = f(a, b)$ and x, y, z, v correspond to the original cofactors $SF_{00}(x_3, ..., x_n)$, $SF_{01}(x_3, ..., x_n)$, $SF_{10}(x_3, ..., x_n)$, $SF_{11}(x_3, ..., x_n)$, respectively (top row of the diagram). EXOR-ing on LI functions according to BR rule is shown here graphically on Karnaugh maps. EXOR-ing of the respective DI functions is shown on formulas that stand on the right sides of the respective Kmaps. However, the simplification rule $X \oplus X = 0$ is used in these formulas, in order to express them all in terms of EXORs on some subset of the initial cofactors.

It can be observed, that by applying the law $(x \oplus y) \oplus (y \oplus z) = (x \oplus z)$, the DI functions $SF_i(x_3, ..., x_n)$ are repeating in the levels of the diagram and do not have to be computed repeatedly in the diagram, for instance,.... Thus, the diagram for all nonsingular expansions for pairs of variables can be created only once, and next the values of EXOR-sums of subsets of functions $SF_i(x_3, ..., x_n)$ can be just inserted for any particular initial cofactors. Thus the number of EXOR operations on subsets of cofactors x, y, z, v is essentially decreased (for efficiency of EXORing, the cofactors can be represented as BDDs or in any other way).

Because our exhaustive algorithm goes through all polarities, it can be used to find the best polarity for a block, and thus it can be also used as a part of some future exact algorithm to create minimum trees or diagrams.

B. Non-exhaustive algorithm based on dynamic creation of butterfly diagram for incompletely specified functions

Below we will explain a faster approximate algorithm, an equivalent of the Exhaustive Algorithm, for finding a good expansion for multi-output functions that works especially well with strongly unspecified functions.

The first observation is that the operations of EXOR-ing on functions $f_i(x_3, ..., x_n)$ can be done on incompletely specified functions as well. It must be, however, taken into account, that when a don't care value "-" is EXOR-ed with a constant, the values of x and $x \oplus y$ are constrained. It means that if $x =$ - and $y = 0$ then the same value should be taken for x and $x \oplus y$; which means, 0 for both, or 1 for both. The choice of these two possibilities is arbitrary, but it is not possible just to write don't care symbols as x and $x \oplus y$. Similarly if $x =$ - and $y = 1$ then the opposite values should be taken for x and $x \oplus y$; which means 0 for x and 1 for $x \oplus y$, or 1 for x and 0 for $x \oplus y$. The choice of these two possibilities is arbitrary, but it is not possible just to write don't care symbols is x and $x \oplus y$. In all other cases the values are not constrained and the standard rules $0 \oplus 0 = 1 \oplus 1 = 0$, $1 \oplus 0 = 1$, $1 \oplus - = -$, $0 \oplus - = -$, $- \oplus - = -$ should be taken.

To illustrate the operation of this algorithm, assume function $f(a, b, c, d)$ from Function 11. The calculation of the levels of the butterfly is shown in Figure 12. The square Kmaps correspond to LI functions and the long-width rectangles to the data functions $SF_i(x_3, ..., x_n)$ (in this particular case, the data functions are $SF_i(c, d)$). In the top row of the diagram these rectangular Kmaps correspond to the cofactors with respect to variables a, b of the map from Fig. 11.

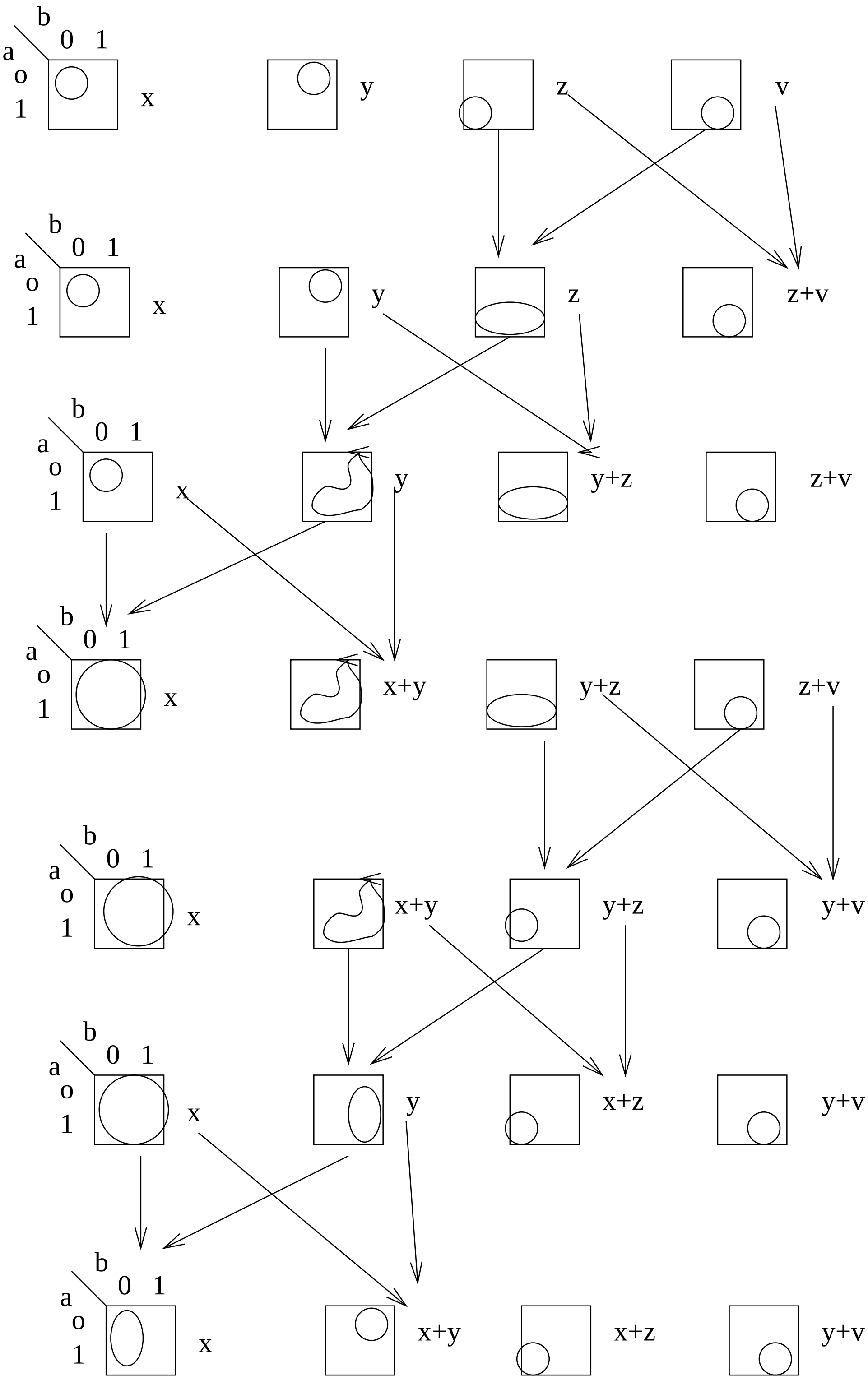

Figure 10: *First part of the Butterfly Diagram to create nonsingular expansions for all LI functions of a,b.*

ab \ cd	00	01	11	10
00	-	-	0	-
01	-	0	1	0
11	-	-	-	-
10	-	-	1	-

Figure 11: *The Kmap to explain the operation of the Non-Exhaustive Polarity Selection Algorithm.*

In contrast to the algorithm from section V.A, in this algorithm, the levels are not pre-computed, but the rule BR is applied to the dynamically selected pairs of functions. At every level, the selection is done in such a way that, using the above EXOR-ing rules for don't cares within Rule BR, as many as possible of the functions $SF_i(x_3, ..., x_n)$ (the long rectangles) will have only symbols 0 and -, which means, as many as possible of these functions will be equivalent to function **0**. In general, when it is not possible to create zero-functions $SF_i(x_3, ..., x_n)$, the choices must be done in such a way that the functions $SF_i(x_3, ..., x_n)$ will have the smallest total cost.

Thus, this non-exhaustive algorithm to find the good expansion does not go through all LI polarities to select the best one. The quality may suffer, but the algorithm becomes much faster. Using this algorithm in step 3 would allow the SLIKDD generating algorithm from section IV to be applied to larger functions.

VI. CONCLUSIONS AND OPEN PROBLEMS.

Based on the previous results in Reed-Muller logic (i.e. the AND/EXOR subset of the general LI Logic), we can expect that the introduced here trees, diagrams, and circuits will find applications in Boolean function representation and multi-level logic synthesis with arbitrary gates (AND/OR/EXOR base). The presented methods can be applied to both completely specified and incompletely specified functions; single-, and multi-output. Both Kronecker-like and Pseudo-Kronecker-like generalizations have been shown. Further generalization to LI Free trees, Forms, DDs, Lattices and layouts is also possible along the lines from [9]. Generalizations to Mixed, Ordered, Free, Lattice and other LI representations along the lines from [18, 19] are also possible. We developed also a very similar approach to multiple-output *Boolean relations*, but before implementing the CAD tool for relations, we have first to practically verify the quality of circuits created from the methods discussed in this paper.

It is also hoped that this paper formulates few new interesting research questions in Linearly Independent Logic. A particularly important open problem is to define generic recursive butterfly patterns to create *all* expansion polarities of certain types and *for arbitrary numbers of variables.*

References

[1] M. Davio, J.P. Deschamps, A. Thayse, "Discrete and Switching Functions," *McGraw Hill*, 1978.

[2] D. Debnath, T. Sasao, "GRMIN: A Heuristic Simplification Algorithm for Generalised Reed-Muller Expressions," *Proc. Reed-Muller'95*, pp. 257-264.

[3] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M.A. Perkowski, "Efficient representation and manipulation of switching functions based on Ordered Kronecker Functional Decision Diagrams," *Proc. Design Automation Conference*, pp. 415-419, 1994.

[4] R. Drechsler, "Pseudo Kronecker Expressions for Symmetric Functions," *Proc. VLSI Design Conference*, 1997.

[5] B. J. Falkowski, S. Rahardja, "Family of fast transforms for GF(2) orthogonal logic," *Proc. Reed-Muller'95*, pp. 273-280.

[6] J. Froessl, B. Eschermann, "Module Generation for AND/XOR-Fields (XPLAs)," *Proc. IEEE ICCD '91*, pp. 26-29, 1991.

[7] D.H. Green, "Families of Reed-Muller Canonical Forms," *Intern. J. of Electr.*, pp. 259-280, February 1991, No 2.

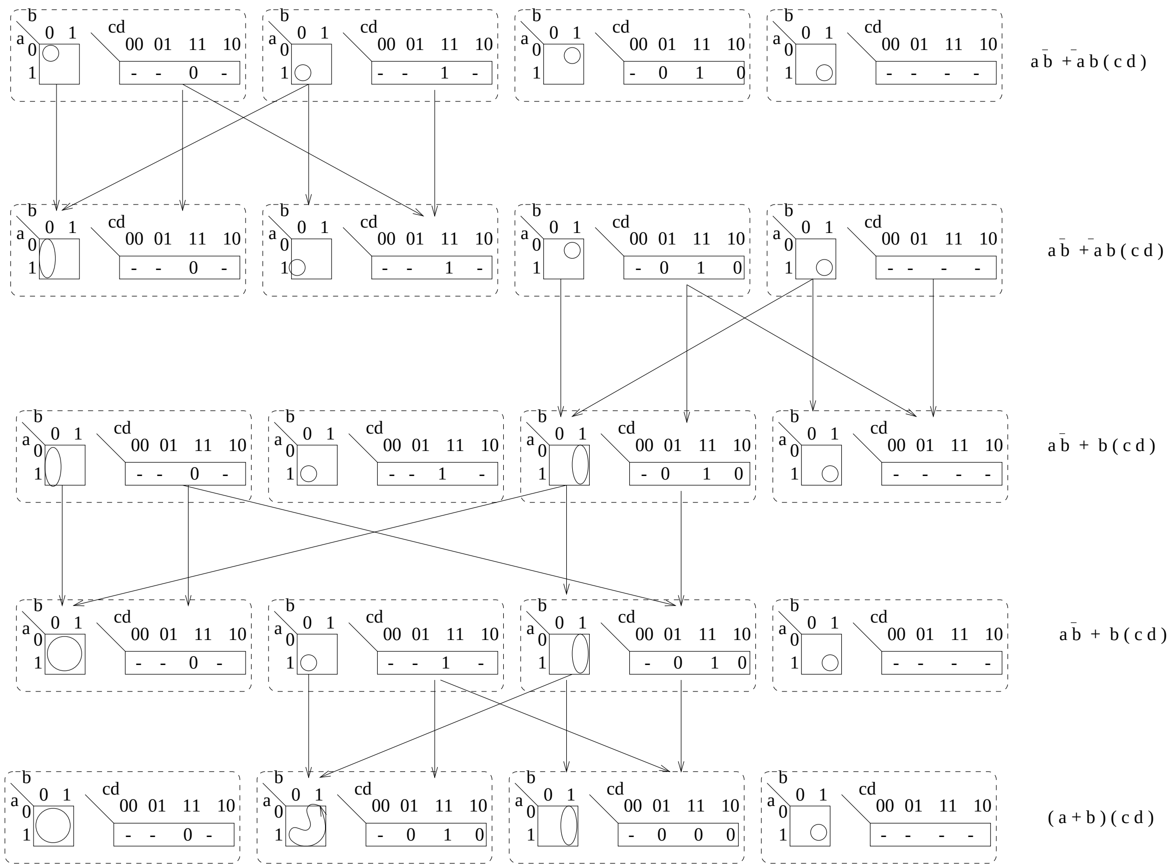

Figure 12: *The calculation of the levels of the Butterfly for an incompletely specified function in the Non-Exhaustive Polarity Selection Algorithm.*

[8] M. Helliwell, and M. Perkowski, "A Fast Algorithm to Minimize Multi-Output Mixed Polarity Generalised Reed-Muller Forms," *Proc. of the IEEE/ACM DAC*, 1988, paper 28.2, pp. 427- 432.

[9] P. Ho, M. A. Perkowski, "Free Kronecker Decision Diagrams and their Application to ATMEL 6000 FPGA Mapping," *Proc. Euro-DAC '94 with Euro-VHDL ' 94*, pp. 8 - 13, September 19-23, 1994, Grenoble France.

[10] M. Perkowski, P. Johnson, "Canonical multi-valued input Reed-Muller Trees and Forms," *Proc. 3rd NASA Symp. on VLSI Design,* Oct. 1991, Moscow, Idaho, pp. 11.3.1-11.3.13.

[11] M.A. Perkowski, "The Generalised Orthonormal Expansion of Functions with Multiple-Valued Inputs and Some of its Applications," *Proc. ISMVL'92*, pp. 442-450.

[12] M. Perkowski, L. Csanky, A. Sarabi, I. Schaefer, "Fast Minimization of Mixed-Polarity AND/XOR Canonical Networks," *Proc. IEEE Intern. Conf. on Computer Design, ICCD'92,* Boston, October 11-13, pp. 32-36.

[13] M. Perkowski, "A Fundamental Theorem for Exor Circuits," *Proc. Reed-Muller'93*, pp. 52-60.

[14] M. Perkowski, A.Sarabi, F. Beyl, "XOR Canonical Forms of Switching Functions," *Proc. of Reed-Muller'93*, pp. 27-32.

[15] M. Perkowski, A.Sarabi, F. Beyl, "Fundamental Theorems and Families of Forms for Binary and Multiple-Valued Linearly Independent Logic," *Proc. Reed-Muller'95*, pp. 288-299.

[16] M. Perkowski, T. Ross, D. Gadd, J.A. Goldman, N. Song, "Application of ESOP Minimisation in Machine Learning and Knowledge Discovery," *Proc. Reed- Muller '95*, pp. 102-109.

[17] M.A. Perkowski, L. Jozwiak, R. Drechsler, "A Canonical AND/EXOR Form that Includes both the Generalized Reed-Muller Forms and Kronecker Reed-Muller Forms," *submitted to Reed-Muller'97.*

[18] M.A. Perkowski, L. Jozwiak, R. Drechsler, "Two Hierarchies of Generalized Kronecker Trees, Forms Decision Diagrams, and Regular Layouts," submitted to the *Reed-Muller'97 Symposium*, Oxford University, U.K., September 1997.

[19] M. Perkowski, E. Pierzchala, and R. Drechsler, "Logic/Layout Synthesis for a Submicron Technology: Mapping Linearly-Independent Expansions to Fat Regular Lattices," *submitted.*

[20] A. Sarabi, M.A. Perkowski, "Fast Exact and Quasi-Minimal Minimisation of Highly Testable Fixed-Polarity AND/XOR Canonical Networks," *Proc. DAC,'92*, pp. 30-35.

[21] A. Sarabi, M. Perkowski, "Design for Testability Properties of AND/EXOR Networks," *Proc. Reed-Muller'93*, pp. 147-153.

[22] A. Sarabi, N. Song, M. Chrzanowska-Jeske, M. A. Perkowski, "A Comprehensive Approach to Logic Synthesis and Physical Design for Two-Dimensional Logic Arrays," *Proc. DAC '94,* San Diego, June 1994, pp. 321 - 326.

[23] T. Sasao, "Representation of Logic Functions using EXOR Operators," *Proc. Reed-Muller '95*, pp. 11- 20.

[24] T. Sasao, "Optimisation of Multiple-Valued AND-EXOR Expressions using Multiple-Place Decision Diagrams," *Proc. IEEE 22nd ISMVL*, 1992.

[25] T. Sasao, H. Hamachi, S. Wada, M. Matsuura, "Multi-Level Logic Synthesis Based on Pseudo-Kronecker Decision Diagrams and Local Transformation," *Proc. Reed-Muller'95*, pp. 152-160.

[26] T. Sasao (editor), "Logic Synthesis and Optimisation," *Kluwer Academic Publishers,* 1993.

[27] T. Sasao, "An Exact Minimisation of AND-EXOR Expressions Using BDDs," *Proc. Reed-Muller'93*, pp. 91-98.

[28] T. Sasao, "Representation of Logic Functions using EXOR Operators," *Proceedings IFIP WG 10.5 Workshop on Applications of the Reed Muller Expansion in Circuit Design, Reed-Muller'95,* pp. 11-20, August 27-29, 1995, Makuhari, Chiba, Japan.

[29] I. Schaefer, M. Perkowski, "Synthesis of Multi-Level Multiplexer Circuits for Incompletely Specified Multi-Output Boolean Functions with Mapping Multiplexer Based FPGAs," *IEEE Trans. on CAD,*, Vol. 12, No. 11, November 1993. pp. 1655-1664.

[30] I. Schaefer, M.A. Perkowski, "Multiple-Valued Input Generalised Reed-Muller Forms," *IEE Proc.,* Pt.E, Vol. 139, No. 6, pp. 519-527, November 1992.

[31] I. Schaefer, M.A. Perkowski, "Multiple-Valued Input Generalised Reed-Muller Forms," *Proc. of the IEEE ISMVL'91*, pp. 40-48.

[32] N. Song, M. Perkowski, "EXORCISM-MV-2: Minimisation of Exclusive Sum of Products Expressions for Multiple-Valued Input Incompletely Specified Functions," *Proc. ISMVL'93*, May 24, 1993, pp. 132-137.

[33] Ch. Tsai and M. Marek-Sadowska, "Multilevel Logic Synthesis for Arithmetic Functions," *Proc. DAC'96*, pp. 242-247.

[34] X. Zeng, M. Perkowski, K. Dill, A. Sarabi, "Approximate Minimization of Generalized Reed-Muller Forms," *Proc. Reed-Muller'95,* pp. 221-230.

[35] X. Zeng, M. Perkowski, H. Wu, A. Sarabi, "A New Exact Algorithm for Highly Testable Generalized Partially-Mixed-Polarity Reed-Muller Forms," *Proc. Reed-Muller'95*, pp. 231-239.