International Conference on
Information, Communications and Signal Processing
ICICS '97
Singapore, 9-12 September 1997

ORDERED AND SHARED, LINEARLY-INDEPENDENT, VARIABLE-PAIR DECISION DIAGRAMS

Marek Perkowski, Lech Jozwiak †, Rolf Drechsler +, Bogdan Falkowski &,
Portland St. Univ. Dept. Electr. Engn., Portland, OR 97207, Tel: 503-725-5411, *mperkows@ee.pdx.edu*,
† Faculty of Electr. Engng., Eindhoven Univ. of Techn., P.O.Box 513, 5600 MB Eindhoven, The Netherlands,
+ Inst. of Comp. Sci., Albert-Ludwigs Univ., 79110 Freiburg in Breisgau, Germany,
& Nanyang Techn. Univ., Sch. Electr. and Electrn. Engng., 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 the synthesis process.

1. 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 [7] only outlined some efficient approaches, but no detailed examples were presented. Paper [3] 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 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" [14, 17, 13, 16], not discussed here. In section 2 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 3 introduces the LI Trees, LI Decision Diagrams, and LI Forms.

2. THE NON-SINGULAR EXPANSION

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

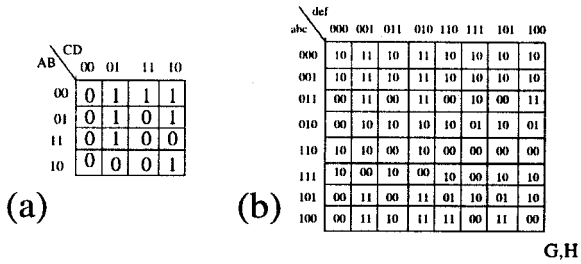

Figure 1: *(a) Function $f(A,B,C,D)$ to Example 2.1, (b) Function to Example 3.3*

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: $f_{\overline{A}\,\overline{B}}(C,D)$, $f_{\overline{A}\,B}(C,D)$, $f_{A\,\overline{B}}(C,D)$, $f_{AB}(C,D)$. We calculate: $f(A,B,C,D) = \overline{A}\;\overline{B}\; f_{\overline{A}\,\overline{B}}(C,D) \oplus \overline{A}\;B f_{\overline{A}\,B}(C,D) \oplus A\;\overline{B} f_{A\,\overline{B}}(C,D) \oplus A\;B\; f_{AB}(C,D) = \overline{A}\;\overline{B}\; f(A,B,C,D)|_{A=0,B=0} \oplus \overline{A}\;B\; f(A,B,C,D)|_{A=0,B=1} \oplus A\;\overline{B}\; f(A,B,C,D)|_{A=1,B=0} \oplus AB\; f(A,B,C,D)|_{A=1,B=1} = \overline{A}\;\overline{B}\;(C+D) \oplus \overline{A}\;B(C \oplus D) \oplus A\;\overline{B}(C\overline{D}) \oplus AB\;(\overline{C}D)$

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

$f(A,B,C,D) = (A+B)\; SF_{A+B}(C,D) \oplus \overline{B}\; SF_{\overline{B}}(C,D) \oplus \overline{A}\; SF_{\overline{A}}(C,D) \oplus SF_1(C,D)$ (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 obtain 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 obtain 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 obtain 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 $CV = M^{-1} \times FV =$

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

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 [9]. 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

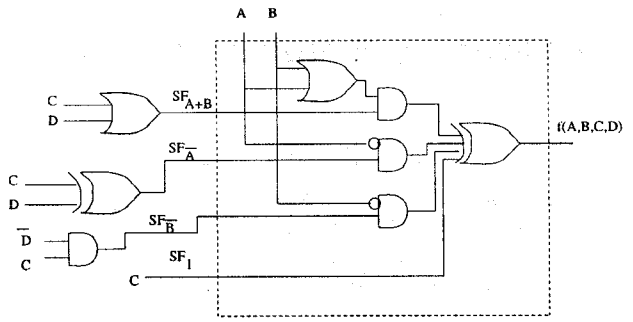

Figure 2: *Nonsingular expansion module to Example 2.1*

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 [6, 3]. 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 [6]. 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 [5, 6, 7] 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

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

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).

3. 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 [15]. Kronecker Trees are quite useful to obtain high-quality multi-level circuits. They can be also generalized to Pseudo-Kronecker Trees, [14], 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 expansions.

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 [5, 6, 7].

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

$$M = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$ 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 [2] on S, pD and nD nodes, (3) performing generalizations of standard Ordered Kronecker Functional Decision Dia-

gram (OKFDD) transformations [2] 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 presents an example of the LI Kronecker Tree. The first level of the tree has Positive Davio 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:

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

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).

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

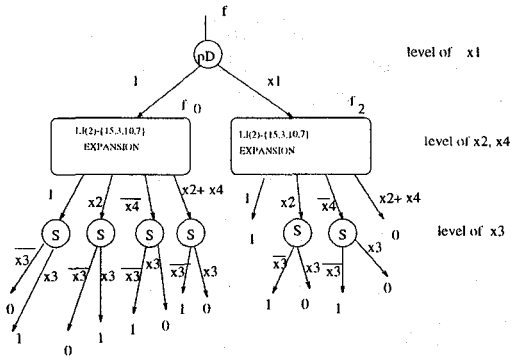

Figure 3: *Example of an LI Kronecker Tree*

case of multi-output GRM expansions, Definition 7 is in accordance with the definition from [18], while Definition 8 is in accordance with the definition from [1]. 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 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.

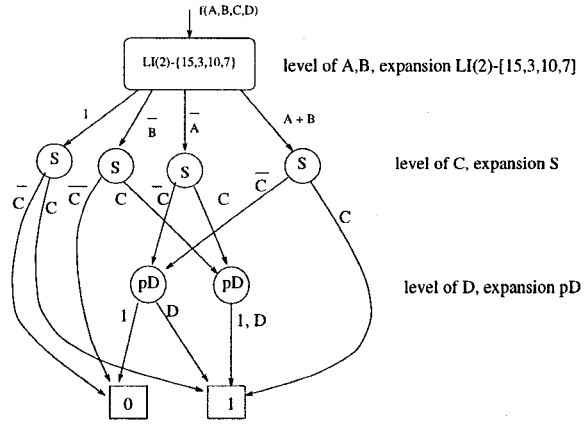

Figure 4: *A LIKDD for function from Example 2.1.*

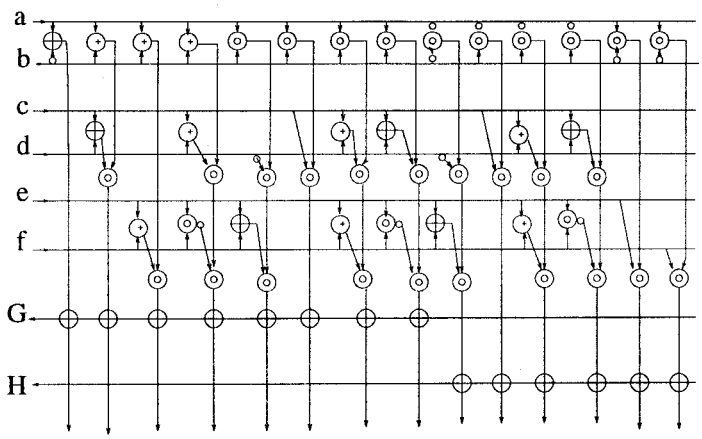

Figure 5: *A three-level LI PLA.*

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., [10]).

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

Example 3.3. A three-level PLA obtained by flattening a Shared LI Pseudo-Kronecker DD for function from Fig. 5 is shown in Fig. 6.

4. CONCLUSIONS

We created efficient heuristic algorithms to create the introduced data structures, but lack of space does not allow us to present them here. 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, [12]. Both Kronecker-like and Pseudo-Kronecker-like diagrams were shown. Further generalization to LI Free trees, Forms, and DDs is also possible along the lines from [4]. LI expansions can be also used in Ordered Lattice diagrams [8], and generalized to various kinds of new types of lattice diagrams, such as: Mixed, and Free Lattice Diagrams and other respective representations [10, 11, 12].

References

[1] D. Debnath, T. Sasao, "GRMIN: A Heuristic Simplification Algorithm for Generalised Reed-Muller Expressions," *Proc. IFIP WG 10.5 Workshop on Applications of the Reed Muller Expansion in Circuit Design, (RM'95)*, pp. 257-264.

[2] 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. DAC*, 1994, pp. 415-419.

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

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

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

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

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

[8] M.A. Perkowski, M. Chrzanowska-Jeske, and Y. Xu, "Lattice Diagrams using Reed-Muller Logic," *Proc RM'97*, Oxford Univ., U.K., Sept. 19 - 20, 1997.

[9] 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," *Proc. RM'97*.

[10] M.A. Perkowski, L. Jozwiak, R. Drechsler, "Two Hierarchies of Generalized Kronecker Trees, Forms, Decision Diagrams, and Regular Layouts," *Proc. RM'97*.

[11] M. Perkowski, E. Pierzchala, and R. Drechsler, "Layout Driven Synthesis for Submicron Technology: Mapping Expansions to Regular Lattices," *Proc. of 7th Intern. Symp. on IC Technology, Systems, and Applications (ISIC-97)*, Sept. 10 - 12, 1997, Singapure.

[12] M. Perkowski, E. Pierzchala, and R. Drechsler, "Ternary and Quaternary Lattice Diagrams for Linearly-Independent Logic, Multiple-Valued Logic, and Analog Synthesis," *Proc. of 1st Intern. Conf. on Information, Communication and Signal Processing (ICICS'97)*, Sept. 9 - 12, 1997, Singapure.

[13] 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.

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

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

[16] 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, Nov. 1993. pp. 1655-1664.

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

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