# EFFICIENT ALGORITHMS FOR CREATION OF LINEARLY-INDEPENDENT DECISION DIAGRAMS AND THEIR MAPPING TO REGULAR LAYOUTS

Marek Perkowski, Bogdan Falkowski &, Malgorzata Chrzanowska-Jeske, and Rolf Drechsler +,

Portland State University, Dept. of Electr. Engn., Portland, Oregon 97207,

Tel: 503-725-5411, Fax: 503-725-4882, *mperkows@ee.pdx.edu*

& Nanyang Technological University,

School of Electrical and Electronic Engineering,

Nanyang Avenue, Singapore 639798.

+ Institute of Computer Science, Albert-Ludwigs-University,

79110 Freiburg in Breisgau, Germany, *drechsle@informatik.uni-freiburg.de*

**Abstract— A new kind of a decision diagrams is presented: its nodes correspond to all types of nonsingular expansions for groups of input variables, in particular pairs. The diagrams are called the Linearly Independent (LI) Decision Diagrams (LI DDs). There are 840 nonsigular expansions for a pair of variables, thus 840 different types of nodes in the tree. Therefore the number of nodes in such (exact) diagrams is usually much smaller than the number of nodes in the well-known Kronecker diagrams (which have only single-variable Shannon, Positive Davio, and Negative Davio expansions in nodes). It is usually much smaller than 1/3 of the number of nodes in Kronecker diagrams. Similarly to Kronecker diagrams, the LI Diagrams are a starting point to a synthesis of multilevel AND/OR/EXOR circuits with regular structures. Other advantages of LI diagrams**

**include: they generalize the well-known Pseudo-Kronecker Functional Decision Diagrams, and can be used to optimize the new type of PLAs, called LI PLAs. Importantly, while the known decision diagrams used AND/EXOR or AND/OR bases, the new diagrams are AND/OR/EXOR-based. Thus, because of a larger design space, multi-level structures of higher regularity can be created with them. The paper presents both new concepts and new efficient synthesis algorithms.**

[1]

## 1. INTRODUCTION

It has been known for few years that the Linearly Independent Logic (LI) [23, 24, 15, 26, 7, 8, 9] can potentially create circuits that are superior in terms of the number of gates, speed, area and testability to canonical AND/EXOR circuits (both two- and multi-level). The concepts of LI logic have been also used for image processing [16] and encoding [1]. Similarly to the Reed-Muller (RM) logic that is a special case of the Linearly Independent logic, the circuits realized using LI logic are obtained by repetitive expansions of a logic function. Unlikely to RM logic, however, where there are only 3 expansion types for a variable; Shannon (S), Positive Davio (pD), and Negative Davio (nD), the LI logic uses more expansion types for **sets of variables**, and the number of such expansions is very high even for 2 variables.

The reason why the LI-logic-based circuits are most often much smaller and never worse than RM-logic-based circuits is simply because they include all canonical AND/EXOR circuits as their special cases, operating in a much larger design space. The circuits generated using LI logic are obtained by expansions with respect to certain nonsingular matrices representing "basis functions". These expansions are called therefore the **nonsingular expansions**. Unfortunately, no efficient algorithms for the calculation of all nonsingular expansions of LI logic have been so far created. The approach from [26] only outlined efficient approaches for limited types of nonsingular expansions, but no detailed synthesis algorithms were presented. Paper [7] presented a "fast transform" method to find a single expansion for certain polarities of variables, but still the problem of selecting the best polarity (and thus, the best expansion) among all polarities of two-variable nonsingular expansions was not discussed. Therefore, although there exist fast transforms (expansions), still no methods are known to select a good one among a huge number of

[1] This research has been supported in part by the NSF grant MIP-9629419.

such transforms. (Applying "fast" transforms successively for all possible polarities would be extremely inefficient and thus have been not proposed.) In conclusion, LI logic despite having a high potential as a powerful generalization of Reed-Muller logic, has been not proven to be practically useful because there were no efficient circuit minimization techniques developed for it.

In this paper, we will develop a general approach to create various types of LI circuits. However, the algorithms for expansion selections will be restricted here to pairs of variables in expansions, similarly to [7].

For the first time, however, it will be presented how good-quality polarities of expansions can be found quite efficiently. Section 2 will present the very idea of expansions for groups of variables as the generalization to well-known expansions. In section 3 we introduce the nonsingular expansions. We illustrate an example of a LI Universal Logic Module with two control variables (corresponding to a node of a tree) and present a theory how to compute the expansion "data input functions" (DI functions, for short) for such two-variable nodes. These expansions are calculated repeatedly in the process of tree creation. Section 4 introduces the LI Trees, and section 5 the LI Decision Diagrams and the LI Forms. Section 6 presents approximate algorithms for the generation of various types of LI Decision Diagrams for multi-output functions. Section 7 describes general principles of creating algorithms to select, for a given set of expansion variables, good polarities (i.e., basis functions) for nodes at a given level of the diagram. Section 8 presents an example of such an exhaustive exact algorithm for completely specified functions, and section 9 presents an approximate algorithm for incompletely specified functions. This algorithm becomes more efficient when the function is weakly specified. Section 10 concludes the paper.

## 2. NEW TYPES OF DIAGRAMS FOR MULTI-VARIABLE EXPANSION NODES

Fig.1

Figure 1a presents a standard tree obtained for single expansion variables in nodes. Every level corresponds to a single variable, and every node denotes a single-variable (standard) expansion: Shannon, Positive Davio or Negative Davio. Figure 1b presents the new tree obtained for pairs of expansion variables in levels. Every level corresponds to a pair of variables, and the sets of variables in levels are not overlapping. Every node is a two-variable expansion. It can be shown using the methods outlined below that the total number of such expansions for two variables equals 840. Thus, there are 840 types of nodes in the tree. We will call this tree a LI decision tree, because Linearly Independent logic is used to create all possible expansion types.

Obviously, as shown in the trees, the forms obtained from flattening of the LI tree for 4 variables have at most 16 terms (leafs), the same number of leafs as the forms obtained from the standard tree. However, the product terms are no longer products of literals, but of arbitrary functions of two variables; for instance a single term $(a + b)(c \oplus d)$ corresponds to four product terms, $a\bar{c}d$, $ac\bar{d}$, $b\bar{c}d$, $bc\bar{d}$, of the standard tree. Thus, the new forms are three-level and not two-level, and have on average a much smaller number of terms. However, these forms allow to create circuits that are as regular as for the standard forms, allowing to generalize the concept of a PLA.

Fig. 2

A simplified hierarchy of decision diagrams that includes our new diagrams is shown in Figure 2. It shows BDDs with single-variable, one-node-type expansions as the narrowest category, next the Kronecker Decision Diagrams with single-variable, three types of node expansions as a broader concept, next the two-variable LI DDs with 840 node types for pairs of variables, and finally, the Linearly Independent Decision Diagrams with arbitrary sets of variables in groups (arbitrary sizes of sets, overlapping or not) as the broadest type of decision diagrams. In addition, similarly as in standard diagrams, one may allow to have any combination of expansion types in level nodes, thus leading to "Pseudo-Kronecker type" of the Linearly Independent diagrams. "Mixed-variable" LI diagrams can have levels corresponding to groups of variables of various sizes. Detailed families of diagrams can be described similarly as it was done in literature for single variables.

Linearly Independent logic [23, 24, 26] 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 (basis functions, LI 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 $SF$ with $m$ variables we have $2^m$ rows). The columns correspond then to the basis functions. A "1" in the intersection of a column "$i$" and row "$j$" means that function "$i$" covers minterm "$j$". Any subset 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 matrix $M$ is nonsingular and 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 corresponding to the columns will be called the "linearly independent family of Boolean functions" (or a set of LI Boolean functions, or a LI set, called also the basis functions [24, 26]). Here, we will call them *LI functions,* for short. The matrix will be called a *"nonsingular matrix"*, and the data input functions $SF_i$ will be called the *DI functions.*

## 3. THE NONSINGULAR EXPANSION

To introduce the ideas of Linearly Independent (LI) logic in a tutorial approach, we will first present a simple example of a nonsingular expansion.

*Example 2.1.* Given is function $f(A,B,C,D)$ from Figure 3.

Fig. 3.

Let us assume that we want to find a certain expansion of this function with respect to variables $\{A,B\}$. As the first step we create an auxiliary equation based on standard Shannon expansion with respect to cofactors on variables $A, B$. This expansion uses the standard cofactors: $f_{\overline{A}\ \overline{B}}(C,D)$, $f_{\overline{A}\ B}(C,D)$, $f_{A\ \overline{B}}(C,D)$, $f_{AB}(C,D)$. All these cofactors are calculated from the initial function $f(A,B,C,D)$. Thus we can write:

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

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

$$= \overline{A}\ \overline{B}\ (C+D) \oplus\ \overline{A}\ B(C\oplus D) \oplus\ A\ \overline{B}(C\overline{D}) \oplus\ \ AB\ (\overline{C}D)$$ (after inserting the values of cofactors (rows) of the Kmap from Figure 3.)

Next we will show how to find the **nonsingular expansion** of this function for the given **"basis functions"** on variables $A$ and $B$. Here we arbitrarily select basis functions as: $f_{A+B} = A+B$, $f_{\overline{B}} = \overline{B}$, and $f_{\overline{A}} = \overline{A}$, and $f_1 = 1$. Our goal now is to find the unknown **"data functions"** $SF_i(C,D)$:

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

Now, in order to calculate the equations of the unknown functions $SF_i(C,D)$ from equation (2.1) as some functions on variables $C$ and $D$, 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 $f(A,B,C,D)$ from Fig. 3 we have:

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

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

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

$$(C+D) = f_{\overline{A}\ \overline{B}}(C,D) = SF_{\overline{B}} \oplus SF_{\overline{A}} \oplus SF_1. \qquad (2.1.1)$$

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

$$(C \oplus D) = f_{\overline{A}\ B}(C,D) = SF_{A+B} \oplus SF_{\overline{A}} \oplus SF_1 \qquad (2.1.2)$$

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

$$(C\overline{D}) = f_{A\ \overline{B}}(C,D) = SF_{A+B} \oplus SF_{\overline{B}} \oplus SF_1. \qquad (2.1.3)$$

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

$$(\overline{C}D) = f_{AB}(C,D) = SF_{A+B} \oplus SF_1. \qquad (2.1.4)$$

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

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} SF_{A+B}(C,D) \\ SF_{\overline{B}}(C,D) \\ SF_{\overline{A}}(C,D) \\ SF_1(C,D) \end{bmatrix} = \begin{bmatrix} C+D \\ C \oplus D \\ C\ \overline{D} \\ \overline{C}\ D \end{bmatrix} \qquad (2.1.5)$$

where we denoted:

$$\begin{bmatrix} f_{\overline{A}\ \overline{B}}(C,D) \\ f_{\overline{A}\ B}(C,D) \\ f_{A\ \overline{B}}(C,D) \\ f_{A\ B}(C,D) \end{bmatrix} = \begin{bmatrix} C+D \\ C \oplus D \\ C\ \overline{D} \\ \overline{C}\ D \end{bmatrix}$$

Denoting the vector of cofactors by $FV$ and the vector of data functions by $CV$, the matrix equation (2.1.5) can be described in the short form:

$$M \times CV \ = \ FV$$

Therefore $M^{-1} \times FV = CV$, which in full form is:

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

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 3, the function F can be represented by a vector

$FV^T = [(C+D) \quad (C \oplus D) \quad (C\overline{D}) \quad (\overline{C}D)\ ].$

Thus vector $CV$ is:

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

Substituting and simplifying we obtain $CV$:

$$\begin{bmatrix} (C+D)\oplus(C\oplus D)\oplus(C\overline{D})\oplus(\overline{C}D) \\ (C\overline{D})\oplus(\overline{C}D) \\ (C\oplus D)\oplus(\overline{C}D) \\ (C+D)\oplus(C\oplus D)\oplus(C\overline{D}) \end{bmatrix} = \begin{bmatrix} (C+D) \\ (C\oplus D) \\ (C\overline{D}) \\ (C) \end{bmatrix}$$

Then, substituting values from CV to (2.1), the equation (2.1) obtains the form:

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

Concluding, we were able to expand the original function with respect to four basis functions on variables $A$, $B$. We will call these functions (in our case, functions $A+B$, $\overline{B}$, $\overline{A}$, 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 columns.

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. [2]

Fig. 4.

The nonsingular expansion with functional coefficients from Example 2.1 is realized using an **"universal logic module"** with control variables $A, B$ (illustrated in Figure 4). We call it an universal logic module, because similarly to a multiplexer with control inputs $A$ and $B$, all functions of two variables can be realized with this module using constants on its data inputs. [3]

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 4 is a generalization of the universal modules:

[2] Moreover, our method of solving this example can be generalized to **arbitrary** sets of Linearly Independent functions. Such matrices can be nonsingular, or singular. Expansions for singular matrices produce sets of solutions of data functions, the best of which are selected using some additional criteria. The method does not require calculating matrices M.

[3] Obviously, the solution from Fig. 4 is not minimal, even assuming the use of LI universal modules. This example has been created to clearly illustrate all principles and matrix calculations. In general, we do not claim that our circuits are always best, only that they are not worse than those obtained from Pseudo-Kronecker diagrams for single variables. The advantages of such circuits become clear for large functions, because of their highly regular realizations with predictable timing, [14, 31, 35, 28, 32, 25, 32].

for Shannon Expansion (a multiplexer), for positive Davio Expansion (an AND/EXOR gate), and for negative Davio Expansion (an AND/EXOR/NOT gate with inverted control variable),

Let us observe that formula (2.2) describes only one of the 840 nonsingular expansions for the pair of variables $A, B$ [24, 7] and thus, 840 different universal modules. (All these universal modules are similar and can be realized by a single switchable universal module). In general, any set of one, two, three, or four variables out of set $\{A, B, C, D\}$ can be selected for the first level expansion of a tree. So, there are very many different trees representing successive expansions. Even if the problem of fast calculating of a single particular expansion were solved, the more important problem remains: how to select the **best one of all the nonsingular expansions** (or the best of nonsingular expansions of certain kind). This problem is difficult, because there are very many such expansions [24]. Our approach here will be to modify some methods known the from the Reed-Muller (AND/EXOR) logic, which is a special case of the LI logic. First, we will formally define the representations of Boolean functions that will be next used in functions' optimization.

Now we will adopt the fundamental theorem of LI logic to a special case of binary nonsingular matrix $M$. 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 $\mathrm{F}(x_1, ..., x_m, ..., x_n)$ such that the set of input variables $\{x_1, ..., x_n\}$ properly includes the set $\{x_1, ..., x_m\}$. There exists an unique expansion

$$F(x_1, ..., x_n) = f_0(x_1, ..., x_m) SF_0(x_{m+1}, ..., x_n)$$
$$\oplus f_1(x_1, ..., x_m) SF_1(x_{m+1}, ..., x_n) \oplus ... \oplus f_{2^n-1}(x_1, ..., x_m) SF_{2^n-1}(x_{m+1}, ..., x_n) \qquad (2.4)$$

where functions $f_i$ are the given basis LI functions of $m$ variables, and the coefficient functions $SF_i$ are the "data input functions" of the remaining input variables and 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.* Omitted. The proof is a formalization of the general case of applying the method for solving the

EXOR logic equations, applied in the example. It is space consuming but straightforward. The method as presented in the example works for any basis LI functions as the columns of a nonsingular matrix $M$.

We will call (2.4) 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. Thus, the data functions on variables $x_{m+1}, ..., x_n$ for given basis LI functions of matrix $M$ are uniquely determined by expansion (2.4). This means that this expansion can be used to create canonical trees. These trees are called LI trees and will be introduced in the next section. Of course, separation of input variables to sets $\{x_1, ..., x_m\}$ and $\{x_{m+1}, ..., x_n\}$ influences the final implementation cost. The important problem of finding good sets is not discussed here.

## 4. LINEARLY INDEPENDENT TREES

Now that we understand the basic concept of the nonsingular expansion, we can build the theory around all nonsingular expansions in exactly the same way as the RM logic is created based on Shannon and Davio expansions: we first introduce trees, then the decision diagrams constructed from the trees, and finally the flattened forms obtained from the diagrams. Thus, we use here the plan of RM logic (which is a special case of the LI logic) to build the entire body of LI structures, circuits and the respective synthesis/optimization methods. The creation of diagrams from trees is, however, more complex, so it is deferred to the next section.

Let us recall that 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 [41]. Kronecker Trees are quite useful to obtain high-quality multi-level circuits by replacing their nodes with respective gates (such as a multiplexer realizing the Shannon expansion node). They can be also generalized to *Pseudo-Kronecker Trees* [21, 38, 6] that lead to even better (i.e. smaller, faster) circuits. The decision diagrams are next created from such Kronecker or Pseudo-Kronecker trees by applying reduction transformations to pairs of nodes of such trees.

It can be observed, however, that a powerful generalization is possible when in the trees one may allow to have nodes for **sets** of variables, instead for single variables only. These sets of variables will be called *blocks*. The concept of an "expansion tree" is now generalized, and the tree is no longer a binary tree, but has *multi-variable* nodes (many children of a node rather than just two). 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 level can have S, pD or nD expansion. Thus, the total number of expansions for a pair of variables in the Kronecker tree is $3 \cdot 3 = 9$. In contrast, there are 840 various nonsingular expansions for a pair of variables in a LI tree. (The total number of expansions for a pair of variables on the top of the Pseudo-Kronecker tree is $3^3 = 27$, because the expansions can be mixed in levels.) These simple calculations demonstrate the power of the concept of expansions for pairs of variables.

The new type of a tree introduced here will be called the *Linearly Independent Kronecker Tree* (LIKT). It is a special case of a general LI Tree. A **LI Tree** is a tree that uses any nonsingular expansions in nodes in a tree level, and any orders of variables, possibly repeated.

*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 are called *blocks*. In the case that each block includes just a single variable, the LIKT reduces to its special case of the well-known Kronecker Tree. If there is only one block that includes all variables, the tree reduces to the special case of a **nonsingular form** [23, 24, 26] (called also the LI form or the orthogonal form).

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, the type S, nD, or pD expansion is selected for all its nodes. If the block is multi-variable, one nonsingular expansion polarity is selected for the nodes of the tree at the tree level corresponding to this block. For the block with $n$ variables there are $2^n$ children nodes of a node.

In LIKTs created for paired input variables 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 for only three types: S, pD and nD. However, for a block with two variables there are 840 nonsingular expansions and 840 matrices $M$. Therefore, for the two-variable nodes there are 840 types of nodes, called LI(2) nodes (expansion types). Each of the expansion types has 4 columns in matrix $M$, so that the expansion types 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 by their basis matrices $M$, shown below. Thus, in LI(2)-$[n_{1,i}, n_{2,i}, n_{3,i}, n_{4,i}]$, the number $n_{j,i}$ is a natural number corresponding to the binary vector of the $j-$th column of the $i$-th matrix $M$. This number is read with the 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 vector of 4 natural numbers, each corresponding to one LI function, being a column of M, starting from the left, and denoted by LI(2)-[15,3,10,7]. The name **"polarity"** comes from standard Reed-Muller logic, where it describes a variable or its negation consistently taken in an expansion. It has been generalized there to three polarity matrices, corresponding to all non-singular 0-1 matrices for a single variable: Shannon, Positive Davio and Negative Davio. Here, the situation is much more general, and the polarity for a group of variables is a set of functions on these variables. It will be called the **"basis matrix"**. Because, however, it plays the same role as polarity in Reed-Muller logic, it can be also called the **"polarity matrix"**. This way, there are as many expansion types as basis matrices. All these matrices are non-singular.

*Definition 3.* **Linearly Independent Forms** (LI Forms) are obtained by **flattening the LI Trees.** Flattening has the following stages:

1. find all branches of the tree that lead to constant 1 in terminal nodes.
2. for each such branch make an ordered product term by multiplying the expressions from the edges in the branch.
3. make an EXOR of these terms.

In other words, flattening to LI forms corresponds to using recursively the flattening rule $a(b \oplus c) = ab \oplus ac$ and rules $a \oplus 0 = a$, $a \cdot 0 = 0$, and $a \cdot 1 = a$ of Boolean algebra to the tree expressions. An example of LI Form and how it was obtained by flattenings will be presented in Example 5.3.

Observe, that there exist two combinational problems that have to be solved for such trees: variable pairing and pair ordering. In the more general case of trees with both single-variable and pair-variable nodes, there exists also the problem of partitioning to pairs and single variables. These problems are not a subject of this paper.

## 5. LINEARLY INDEPENDENT DECISION DIAGRAMS AND FORMS

*Definition 4.* (Reduced) LI Decision Diagrams (LI DDs) are created from respective types of LI trees by:

(p1) combining isomorphic nodes of any kind, [5],

(p2) performing standard Ordered Kronecker Functional Decision Diagram (OKFDD) transformations [5] on S, pD and nD nodes,

(p3) performing generalizations of standard Ordered Kronecker Functional Decision Diagram (OKFDD) transformations [5] on multi-variable nodes. These generalized transformations remove any node that evaluates to its single argument.

Let us explain the transformations from (p3). We will say that a node **evaluates to a single argument function** $H_i$, when after the following stages:

1. Substitute in the expression describing the expansion of the node the data input functions being constants by these constants.

2. (Constant propagation and Boolean simplification) Simplify this expression using recursively standard Boolean algebra rules: $A\ 0 = 0,\ A\ 1 = A,\ A + 0 = A,\ A + 1 = 1, A + A = A, AB + \overline{A}B = B$.

the expression becomes $H_i$.

Observe that in this transformation only those multivariable nodes should be evaluated that their formulas include as arguments some logic constants and/or repeated signals $H_i$.

For instance, the following examples illustrate the concept of evaluation:

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 $H_3$.

The above method of creating the reduced LI DD is a generalization of the standard RM logic simplification rules for S, pD and nD nodes that are applied to create the OKFDDs. Two nodes that evaluate to the same arguments become **isomorphic nodes** and as such are combined in a standard way known from the way how DDs are created from trees in RM logic.

*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*), where the flattening operation is defined in Definition 3.

The LI Forms are no longer realized in two-level circuits, as is the case of the flattened circuits obtained from AND/EXOR trees and circuits. The LI forms have three levels: the first (from output) level are EXOR gates, the second are AND gates and the third are *arbitrary* Boolean functions defined on blocks of variables. The problem of the best selection of these functions is the subject of this paper. The LI Kronecker Forms can be implemented in a three-level circuit called a LI PLA, with ordered pairs of input variables for third level. An example of such a LI PLA will be given in Example 5.3.

Fig. 5.

*Example 5.1.* Figure 5 illustrates an example of the LI Kronecker Tree. The first level of the tree has Positive Davio expansion for variable $x_1$. It creates an expansion:

$$f(x_1, x_2, x_3, x_4) \;=\; f_0(x_2, x_3, x_4) \;\oplus\; x_1\; f_2(x_2, x_3, x_4),$$

where $f_2(x_2, x_3, x_4) = f_0(x_2, x_3, x_4) \oplus f_1(x_2, x_3, x_4)$, and $f_0(x_2, x_3, x_4)$, $f_1(x_2, x_3, x_4)$ are respectively the negative and positive cofactors of $f$ with respect to input variable $x_1$.

The second level has LI(2)-[15,3,10,7] expansion for the set of LI functions on 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:

$$f_0(x_2, x_3, x_4) = SF(f_0)_1(x_3) \oplus x_2 SF(f_0)_{x_2}(x_3) \oplus \overline{x_4} SF(f_0)_{\overline{x_4}}(x_3) \oplus (x_2 + x_4) SF(f_0)_{(x_2+x_4)}(x_3) \qquad (5.1)$$

where notation $SF(f_i)_{LI_j}(X)$ denotes data function $SF_{LI_j}$, with arguments from the set $X$ of variables, applied to the argument function $f_i$. The function in the subscript is thus a basis function of the expansion. $SF_{LI_j}$ is one of the data functions from the expansion basis matrix M, corresponding to the basis function from its subscript. (In our case, these are functions 1, $x_2$, $\overline{x_4}$, and $x_2 + x_4$. Also, in our case, $X = \{x_3\}$). Formula (5.1) is a specialization of the nonsingular expansion (2.4) applied to cofactor function $f_0(x_2, x_3, x_4)$ as $F(x_1, ...x_n)$, and with expansion variables $x_2$, $x_4$ in the linearly independent functions. Subfunctions $SF_i$ of the remaining variable $x_3$ 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).

Fig. 6.

*Example 5.2.* A LI Kronecker Decision Diagram created from a LIKT corresponding to the expansion from Example 2.1 (and the circuit from Figure 4), is shown in Figure 6. It was obtained by combining isomorphic nodes.

Now we will generalize LI Kronecker trees and LI Kronecker DDs for multi-output functions and "pseudo" data structures. LI "Pseudo" DDs generalize the Pseudo-Kronecker DDs [38, 6].

**LI Kronecker DDs For Multi-Output Functions.**

*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 LI Kronecker DDs for a two-input, three-output function, the *Polarity Vector of a Single-Polarity Nonsingular Expansion* is described by 4 natural numbers (columns from matrix M, as before), and the *Polarity Vector of a Multi-Polarity Nonsingular Expansion* is described by $3 * 4 = 12$ natural numbers. Observe, that in the special case of a multi-output Generalized Reed-Muller (GRM) expansions (realized in multi-variable nodes), Definition 7 is in accordance with the definition of multi-output GRM forms from [47], which we will call the **single polarity GRMs**. Definition 8 is in accordance with the definition of multi-output GRM forms from [4] which we will call the **multi polarity GRMs**. Obviously, the minimal DD (or minimal form) obtained with the expansions for multi polarity GRMs is therefore smaller than the one with expansions for single polarity GRMs. There are, however, some advantages of considering single polarity GRMs. They include: 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 that these invertors practically don't count to the cost of the realization. However, for general LI circuits, these circuits constitute higher fractions of the total circuit costs, so it is reasonable to assume that for some types of gate/layout realizations the polarities (and their corresponding circuits) are the same for each output function.

*Definition 9.* A single-output Kronecker DD 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 DD 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.

**LI Pseudo-Kronecker DDs For Multi-Output Functions.**

*Definition 10.* The *LI Pseudo-Kronecker DD* is defined similarly as the LI Kronecker DD; the only difference is that in every level, any combination of expansions can be used.

For instance, in LIKDD the expansion LI(2)-[15,3,10,7] is used in the entire level of variables' block $\{x_7, x_8\}$, and in LIPKDD expansions LI(2)-[15,3,10,7], LI(2)-[5,8,10,12], and LI(2)-[3,5,8,15] are mixed in a level. An example of LI DDs will be given in Example 5.3. below.

The relation between the LI Pseudo-Kronecker DD and the LI Kronecker DD is exactly the same as the relation between the Pseudo-Kronecker DD and the Kronecker DD. 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* are defined as flattened forms of respective DDs. Because in case of Pseudo-Kronecker DDs every node can have a different polarity, Definition 7 does no longer apply to Pseudo- type representations of multi-output functions. Polarity lists as for Kronecker DDs are no longer created for Pseudo Kronecker DDs because of the total freedom of expansion selection for their levels. The name LI DD will be generic to all kinds of LI DDs (LI Kronecker, LI Pseudo-Kronecker, LI Mixed, LI Ordered, LI Free, etc., [29]).

*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. A *Shared Ordered Linearly Independent Pseudo-Kronecker Decision Diagram* (SOLIPKDD) is an ordered SLIPKDD.

After defining trees, diagrams, and LI PLAs, let us illustrate them by one comprehensive example.

*Example 5.3.* Given is a six-input, two-output function $G,\ H(a,b,c,d,e,f)$ from Figure 7.

Fig. 7.

The respective LI Pseudo-Kronecker DDs for blocks {a,b}, {c,d}, {e,f} is shown in Figure 8.

Fig. 8.

Fig. 9.

A Reduced Shared, Ordered LI Pseudo-Kronecker Decision Diagram created from this diagram is shown in Figure 9. To enable the reader to analyze the final solutions, in this diagram we showed the internal gate-level structure of nodes corresponding to the expansions from Figure 8.

The multi-level circuit obtained from the SOLIPKDD after the propagation of constants is shown in Figure 10. Observe for instance the node with output signal $g$ from Figure 9. It has two data inputs being constants 0, so it is transformed to the gate-level circuit from Figure 10.

Fig. 10.

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 the flattened SOLIPKDD (after constants propagation) is shown in Figure 11. Again, it can be observed that several operators on variable pairs are repeated, thus can be **factorized**, which would lead to a circuit with less gates. But the PLA regularity would be lost. Observe, that the regular array from Figure 11 can be directly mapped to fine grain FPGAs such as those from Concurrent Logic/ATMEL [2], Motorola or Xilinx 6000 series. The circuit can be also easily mapped to any architecture with 6-input lookup tables.

Fig. 11.

From now on, we will assume that **each block has only two variables.** The respective representations will be called *Double-variable LI trees, Double-variable Decision Diagrams,* and *Double-Variable LI Forms*, respectively. Although in this paper we discuss LI diagrams for only two variables in each block, all concepts and algorithms can be expanded to blocks of arbitrary size, but the algorithms would become less efficient. Also, in this paper we will consider the LI circuits designed according to Definition 7 (we developed also similar methods for the circuits realized using Definition 8).

## 6. ALGORITHMS FOR THE GENERATION OF SOLIKDDS AND SOLIPKDDS

Now, that the concept of SOLIKDDs and SOLIPKDDs and the methods for their creation have been explained, we will turn to generating trees with good expansion type selections for their nodes. We will outline the general algorithms and in the next sections we will discuss approaches to the most important step of selecting the expansion polarities for levels or for individual nodes of LI diagrams. In our considerations, for multi-output functions the algorithms will generate 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 DDs, where the order of expansion variables does not influence the cost of Kronecker expressions obtained from their flattening. Therefore, the minimum form can be found by investigating all LIKDDs for **arbitrary** order of blocks. In contrast, it is not so for the pseudo-Kronecker LI representations, for which all possible permutations of blocks should be calculated in order to find the minimum LI Pseudo-Kronecker Form, thus running the algorithm repeatedly for all possible orders of blocks. Which is, however, not practical for large functions.

While creating a multi-output function of LI Kronecker tree, 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) in order to select the best one. Each level of the multi-output diagram corresponds to a block with 2 elements. Thus, for a two-variable block, **the total of 840 nonsingular expansions** are generated in the exhaustive method which will be described in section 7. While calculating the cost function for each expansion, the total cost of nodes of the next level is calculated as the sum of costs of every node in the next level. 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 Boolean function are combined to single nodes.

The local optimization algorithm to create the SOLIKDD for multi-output function is the following.

**Algorithm 1. Generation of the SOLIKDD for multi-output function.**

1. The set of input variables is partitioned to pairs and single elements (several variable pairing algorithms are known from the literature). In examples below 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. Single polarity definition is used for polarity creation.
4. Apply this expansion to all nodes from this level, combine all the isomorphic nodes at the next level (tautology check - see Definition 4).
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 [42]. We do not discuss the variable pairing and variable ordering problems. The algorithm 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 for some of their subfamilies.

The algorithm to create the SOLIPKDDs is very similar to the one presented above. The only difference is in step 3. At each level, the best expansions for each intermediate function (each node of the SOLIPKDD) are chosen separately. So at every level of the diagram, any combination of expansion-node types becomes now possible. These diagrams are no longer canonical, so they cannot be used for function representation in verification processes, but are applicable in circuit synthesis. This way the **Local Optimization SOLIPKDD Algorithm** for Pseudo Diagram is created (not presented here), which finds the best expansion polarity. The difference is that in the Local Optimization SOLIKDD Algorithm the search is among all polarities **for all nodes of the level of the tree together**, and for SOLIPKDD the search is performed among all polarities but **for each node of the level separately**.

# 7. BASIC PRINCIPLES OF EFFICIENT ALGORITHMS FOR THE SELECTION OF THE BEST NONSINGULAR EXPANSION POLARITY

The most difficult step of the LIDD generation algorithms is how to choose a good polarity. In this section we consider that problem and propose a solution. Of course, in general one can select any particular polarity of a nonsingular expansion and next create a diagram for these expansion types. Also, one can create a prespecified **Multi-Output Polarity List** and next expand according to it (Such list describes polarities for variables in a format shown in section 5). The function representations could be thus calculated for these expansion types as it was illustrated in section 3. If there exists a fast transform, it should be applied, instead of inverting matrix $M$ [7]. However, the quality of such an approach can not be very good because of the random nature of selecting these polarities (Observe that this is an analogous problem as the one known by the name "polarity selection problem" in the area of Functional Decision Diagrams and Fixed Polarity Reed-Muller Forms. It is, however, more difficult to solve in LI logic, because there are many more polarities). Therefore, other methods for finding good expansions must be looked for, unless for some reasons the good polarities can be guessed. Let us then look again to the Reed-Muller logic for an analogy.

One of the interesting and popular concepts of Reed-Muller logic are the **butterfly diagrams** that allow to create **all** Fixed Polarity RM expansions by transforming iteratively the function under optimization from one polarity form to another polarity form, 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 $M^{-1}$. Gray-code ordering of polarities is usually used. 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 [47]. Below we will demonstrate that such methods are also applicable to LI logic.

*Property 2.* In matrix $M$, as well as in matrix $M^{-1}$, any column can be replaced by a linear combination of itself with other columns and the matrix remains nonsingular, thus it is a base of a new expansion.

Based on Property 2, a new *polarity expansion* can be obtained using the Basic Rule ($BR$), given below, to certain selected columns (i.e., basis functions).

*Property 3.* Given is the following rule.

RULE BR:

$$f_1(x_1,x_2)SF_1(x_3,...,x_n) \oplus f_2(x_1,x_2)SF_2(x_3,...,x_n) =$$

$$[f_1(x_1,x_2) \oplus f_2(x_1,x_2)]SF_1(x_3,...,x_n) \oplus f_2(x_1,x_2)[SF_1(x_3,...,x_n) \oplus SF_2(x_3,...,x_n)]$$

where $f_1(x_1,x_2)$ and $f_2(x_1,x_2)$ are arbitrary LI functions, and $SF_1(x_3,...,x_n)$ and $SF_2(x_3,...,x_n)$ are the corresponding to them data input functions (DI functions). Any nonsingular expansion formula can be obtained by a repeated application of Rule BR to pairs of LI and DI functions:

$$[\ f_1(x_1,x_2),\ f_2(x_3,...,x_n)\ ]\ ,\ [\ f_3(x_1,x_2),\ f_4(x_3,...,x_n)\ ].$$

This way, rule $BR$ describes simultaneous EXOR-ing of columns in matrix $M$ and corresponding columns in $M^{-1}$. (It is easy to verify that this rule is true, using simple Boolean manipulations and comparing its left and right sides). Observe, that although rule BR assumes pairs of variables in functions $f_1$ and $f_2$, it is a general synthesis rule for functions with **arbitrary** numbers of variables.

By repeated applications of rule BR all possible expansions can be found starting from one initial expansion, the canonical SOP/ESOP expansion with disjoint cofactors as basis functions. The methods based on rule $BR$ will be therefore a fundament of several very general optimization methods of LI logic. But a question remains: "Which pairs of columns to select and how long to continue the application of rule $BR$?"

These are the possible approaches in LI logic to find good (best) expansions based on rule BR.

A1. To find **all** nonsingular expansions for a function. This problem is important theoretically because it is a generalization of problems such as "find the best FPRM form", or "find the best KRM form", which are a classic in RM logic. It should be solved in order to create exact algorithms and enumerate all best solutions. However, practically this approach is of less importance, since the number of all nonsingular expansions is very high.

A2. To find not all but **as many as possible** sequences of nonsingular expansions, but such sequences that can be created in a very efficient way. This would allow to create efficient multi-level minimizers for those functional bases that correspond to some sets of nonsingular expansions for which such sequences can be found.

A3. To find all nonsingular expansions for some limited **special** families of expansions (such families are defined by matrices $M$ having some special properties). This is exactly the approach developed by many researchers for past 40 years in Reed-Muller logic (AND/EXOR logic), which is a proper subset of the general LI Logic. For instance, this was done for Fixed Polarity RM forms, or Kronecker RM forms, which restrict basis functions to ANDs of certain sets of literals. Also, we proposed different families of special functions for the general LI logic as well [26, 24, 23, 35, 7]. They are of interest to create highly regular PLA-like structures with short connections for submicron technologies. There exist classes of expansions which are practical for synthesis using Fine Grain FPGAs, but for which fast transforms do not exist. Many fast transforms were identified for various LI classes [7], unfortunately for some of them the applications are not yet known to us. For all these special families we would be able now to build butterfly diagrams based on rule $BR$. Our future research goal is to find such families with some interesting and useful properties.

From the point of view of A2 and A3 above, another important observation is that until now, the following classes of LI subfamilies have been of interest in general LI Logic:

C1. Those for which both fast forward and fast inverse recursive transforms exist [7]. This is the easiest class of families to create efficient algorithms based on butterfly concepts. The open problem however remains; "is this a practically useful class for current technologies and gate libraries?" We plan to investigate the properties of circuits created using the approach from [7].

C2. Those for which only the fast forward recursive transforms exist. This is a wider class than class C1, so there exists a better chance that such transforms exist for the interesting families of expansions. Similar research plans as in C1 can be formulated.

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 and other FPGAs. Even if general solutions will be not found, it is worthy to find specific solutions for limited number of input variables in a block, for instance for 2 or 3 variables.

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

Points A1 - A3 and C1 - C4 above delineate a large body of theory and algorithms that can be developed for various special applications of LI logic. The next two sections will illustrate two approaches based on the butterfly diagram concepts.

## 8. EXHAUSTIVE ALGORITHM BASED ON THE PRE-COMPUTED BUTTERFLY DIAGRAM FOR COMPLETELY SPECIFIED FUNCTIONS

Even if in general there exists no recursive method to define a universal Butterfly-like diagram for an *arbitrary* LI matrix or if such a method is not known, a generator of a specific diagram can be created **once for all** for a set of variables with *certain selected* cardinality and for any given set of expansion polarities. This pre-computed diagram can be stored in the computer memory, and next, it can be applied to a given function to calculate all respective expansions together with their costs. We will call this a **"pre-computed Butterfly diagram".** It can be used to find the best polarity for a block in the Algorithm 1. The algorithm goes through all polarities, calculates the cost of each circuit corresponding to the polarity, and returns the polarity with the minimum cost. As an example, let us discuss the method applied to a single-output function $F(x_1, x_2, x_3, ...x_n) = F(a, b, c, ...)$. The set of all polarities is created as levels (rows) in a butterfly-like diagram from Figure 12 (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 functions $x(x_1, x_2, ..., x_n)$, $y(x_1, x_2, ..., x_n)$, $z(x_1, x_2, ..., x_n)$, $v(x_1, x_2, ..., x_n)$ correspond to the original cofactors $x = F_{a=0,b=0} = SF_{00}(x_3, ..., x_n)$, $y = SF_{01}(x_3, ..., x_n)$, $z = SF_{10}(x_3, ..., x_n)$, $v = SF_{11}(x_3, ..., x_n)$, (see the top row of the diagram). EXOR-ing the 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 to illustrate clearly the principle of this method.

Fig. 12.

Thus, the first row corresponds to expansion on standard cofactors:

$$\overline{a}\,\overline{b}\,x \oplus \overline{a}\,b\,y \oplus a\,\overline{b}\,z \oplus a\,b\,v$$

with **cost** $= 8 + cost(x) + cost(y) + cost(z) + cost(v)$, where $cost(x)$ is a literal cost (complexity, total number of literals) of function $x$ in the expansion, etc.

The second row is:

$$\overline{a}\,\overline{b}\,x \oplus \overline{a}\,b\,y \oplus a\,z \oplus a\,b\,(z \oplus v)$$

with **cost** $= 7 + cost(x) + cost(y) + cost(z) + cost(z \oplus v)$, where $cost(z \oplus v)$ is a cost of function $z \oplus v$ and is calculated only once, together with function $z \oplus v$.

Comparison of expansion formulas from the first and second rows shows clearly how the second expansion formula is created from the first one by applying rule BR to the last two columns:

$$a\overline{b}z \oplus abv \Longrightarrow az \oplus abz \oplus abv = az \oplus ab(z \oplus v)$$

where $z \oplus v = z(x_1, ...x_n) \oplus v(x_1, ..., x_n)$ is a new function calculated by EXORing functions $z$ and $v$.

While applying the BR rule, the simplification rule $X \oplus X = 0$ is used as well. For instance, it can be observed that, between rows 4 and 5 in Figure 12, the law $(x \oplus y) \oplus (y \oplus z) = (x \oplus z)$ is applied. In addition, the DI functions $SF_i(x_3, ..., x_n)$ are repeating in the EXOR formulas in the levels of the diagram and do not have to be computed repeatedly in the diagram, whenever their EXOR formulas such as $x \oplus z \oplus v$ are created. Thus, the generator of the diagram for all nonsingular expansions for pairs of variables can be created once and stored in the memory. The generator for the part shown in Figure 12 is:

REPEAT 2 times { BR(col3,col4), BR(col2,col3), BR(col1,col2) }

Next, the values of EXOR-sums of subsets of functions $f_i(x_3, ..., x_n)$ which are calculated once for the given function $F(x_1, ..., x_n)$, can be just inserted for any particular initial cofactors also calculated only once. Thus the number of executions of EXOR operation on subsets of cofactors

$x(x_1, x_2, ..., x_n)$, $y(x_1, x_2, ..., x_n)$, $z(x_1, x_2, ..., x_n)$, $v(x_1, x_2, ..., x_n)$

is essentially decreased, because all repeated EXOR-ings have been now supressed. To obtain high speed of the EXORing operation, the cofactors can be represented as BDDs, KFDDS, cube arrays, bit sets, or in any other efficient representation. Observe that because of generality of BR rule, this method can be applied with no change to functions of arbitrary size and only functions $x$, $y$, $z$ and $v$, represented by BDDs, will have more variables.

## 9. NON-EXHAUSTIVE ALGORITHM BASED ON DYNAMIC CREATION OF BUTTERFLY DIAGRAM FOR INCOMPLETELY SPECIFIED FUNCTIONS

While the algorithm from section 8 can be applied to find the minimum cost expansion for small completely specified functions, below we will explain a faster approximate algorithm, similar to the Exhaustive Algorithm, for finding a good expansion for multi-output functions. It works especially efficiently 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 $x \oplus y$ become 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 any of these two possibilities is arbitrary, but it is not possible to just write don't care symbols as $x$ and $x \oplus y$, because this would mean the possibility of selecting 0 for one of those don't cares, and 1 for another don't care in next stages. 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 $x \oplus y$. 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. These rules are used for EXORing any functions, otherwise the application of BR rule is the same as in section 8.

To illustrate the operation of this Approximate Algorithm, assume function $f(a, b, c, d)$ from Figure 13. The calculation of the first few levels of the butterfly diagram is shown in Figure 14. 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 the rectangular Kmaps correspond to the cofactors with respect to variables $a, b$ of the map from Figure 13 (i.e., to the rows of the Kmap).

Fig. 13.

Fig. 14.

In contrast to the algorithm from section 8, in this algorithm, the levels are not pre-computed, but the rule BR is applied to the dynamically selected pairs of functions (i.e. pairs of columns). At every level, the selection is done in such a way that, using the above EXOR-ing rules for incomplete functions 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. Observe that this algorithm can be used without any modification assuming pairs of variables in nodes, but **arbitrary total number of variables**.

Concluding, the above non-exhaustive algorithm to find a good expansion does not visit all possible LI polarities to select the best one but terminates when the curve of best solution cost until now versus the solution number becomes flat for a prespecified number of generated solutions. This value is selected experimentally. The quality of the final solution may suffer, but the algorithm becomes much faster. Using this algorithm in step 3 would allow the SOLIKDD generating Algorithm 1 from section 6 to be applied to larger functions.

## 10. CONCLUSIONS

We introduced new concepts of LI Decision Diagrams and shown that they include previously known BDDs, KFDDs and other diagrams. We expect that the LI Decision Diagrams, LI PLAs and LI factorized 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 Free LI, Forms and DDs are also possible along the line of the approach to free diagrams presented in [13]. Generalizations to Mixed, Ordered, Free and other LI representations following the methods presented in [29] are also possible.

This paper opens several new interesting research questions in Linearly Independent Logic. A particularly important open problem is to define generic recursive butterfly diagrams to create *all* expansion polarities of certain practical types and *for arbitrary numbers of variables.* A very practical application of the methods presented and outlined here is for the optimization of various types of lattice diagrams [25, 14, 31, 32, 19, 17], such as regular lattices with 2, 3 and especially 4 inputs/outputs from a node. All methods can be applied to new Xilinx FPGAs, as well as similar lookup-table architectures. The new representation and algorithms developed in this paper 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" [38, 46, 35, 42], not discussed here.

We have programmed the algorithms to create SOLIPKDDs and related lattice diagrams and we obtained circuits of small sizes, regular layouts, predictable timing and high testability for functions with tens of inputs and outputs, [18]. Figure 15 shows a regular layout generated.

Fig. 15.

The advantage of our multi-level synthesis method versus standard approaches for fine grain FPGAs is that our method produces netlists with short and regular connections, which simplifies the phases of technology mapping, placement and especially routing. None of the known approaches has these desirable attributes. The algorithm complexity issues and the discussion of experimental results for FPGA mapping and regular layouts will be discussed in a separate paper.

## References


[1] K. Adams, J. Campbell, L. Maguire, J. Webb, "State Assignment Techniques in Multiple-Valued Logic," *Proc. ISMVL'99*, pp. 220 - 224.

[2] Concurrent Logic Inc., "CLI 6000 Series Field Programmable Gate Arrays," *Preliminary Information*, Dec. 1, 1991, Rev. 1.3.

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

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

[5] 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*, 1994, pp. 415-419.

[6] R. Drechsler, "Pseudo Kronecker Expressions for Symmetric Functions," *Proc. VLSI Design Conference*, 1997, pp. 511 - 513.

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

[8] B. J. Falkowski, and S. Rahardja, "Fast Transforms for Orthogonal Logic", *Proc. of IEEE Intern. Symp. on Circuits and Systems (28th ISCAS),* Seattle, Washington, USA, May 1995, pp. 2164-2167.

[9] B. J. Falkowski, and S. Rahardja, "Classification and properties of fast linearly independent logic transformations,", *IEEE Transactions on Circuits and Systems - II: Analog and Digital Signal Processing*, Vol. 44, No. 8, Aug. 1997, pp. 646 - 655.

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

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

[12] 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.

[13] 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*, September 19-23, 1994, Grenoble France, pp. 8 - 13.

[14] M. Chrzanowska-Jeske, Z. Wang, and Y. Xu, "Regular Representation for Mapping to Fine-Grain, Locally- Connected FPGAs," *Proc. of the International Symposium on Circuits and Systems, ISCAS'97,* Vol. 4, pp. 2749-2752.

[15] M. Perkowski, P. Dysko, B. Falkowski, "Two learning methods for a tree search combinatorial optimizer," *Proc. of the IEEE Intern. Phoenix Conf. on Computers and Comm.*, Scottsdale, Arizona, March 1990, pp. 606-613.

[16] P. Katiyar, "Study of Using Logical Transforms for Texture Analysis," *http://exodo.upr.clu.edu/ pkumar/proposal.html".*

[17] P. Lindgren, R. Drechsler, B. Becker, "Synthesis of Pseudo-Kronecker Lattice Diagrams," *Proc. ICCD'99*, pp. 307 - 310.

[18] A. Mishchenko and M. Perkowski, "Highly testable Linearly Independent Lattice Diagrams and their application to regular layout generation", to be submitted.

[19] A. Mukherjee, R. Sudhakar, M. Marek-Sadowska and S.I. Long, "Wave Steering in YADDs : A Novel Non-Iterative Synthesis and Layout Technique," *Proc. Design Automation Conference (36th DAC),* New Orleans, LA, June 1999, pp. 466 - 471.

[20] 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.

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

[22] 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.

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

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

[25] M. Perkowski, E. Pierzchala, *"New Canonical Forms for Four-Valued logic,"* Report, EE Dept, PSU, 1993.

[26] 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.

[27] 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.

[28] 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. Reed-Muller'97*, pp. 219 - 233.

[29] M.A. Perkowski, L. Jozwiak, R. Drechsler, "New Hierarchies of AND/EXOR Trees, Decision Diagrams, Lattice Diagrams, Canonical Forms, and Regular Layouts" *Proc. Reed-Muller'97 Symposium,* Oxford University, U.K., September 1997, pp. 115 - 132.

[30] M.A. Perkowski, L. Jozwiak, R. Drechsler, B. Falkowski, "Ordered and Shared, Linearly Independent, Variable-Pair Decision Diagrams, *Proc. First International Conference on Information, Communications and Signal Processing, ICICS'97,* Singapore, 9-12 Sept. 1997, Session 1C1: Spectral Techniques and Decision Diagrams, pp. 261 - 265.

[31] M.A. Perkowski, M. Chrzanowska-Jeske, and Y. Xu, "Lattice Diagrams using Reed-Muller logic," *Proc. RM'97*, pp. 61 - 72.

[32] M.A. Perkowski, E. Pierzchala and R. Drechsler, "Ternary and Quaternary Lattice Diagrams for Linearly-Independent Logic, Multiple-Valued Logic and Analog Synthesis," *Proc. ICICS'97*, 10-12 Sept. 1997, Singapore, pp. 269 - 273.

[33] 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.

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

[35] 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.

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

[37] T. Sasao, "Optimization of Multiple-Valued AND-EXOR Expressions using Multiple-Place Decision Diagrams," *Proc. IEEE 22nd ISMVL*, 1992, pp. 451 - 458.

[38] 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.

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

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

[41] 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,* August 27-29, 1995, Makuhari, Chiba, Japan, pp. 11-20.

[42] 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.

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

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

[45] 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.

[46] Ch. Tsai and M. Marek-Sadowska, "Logic Synthesis for Testability," private information.

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

[48] 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.

LIST OF FIGURES.

of a,b.

Figure 13. The Kmap to explain the operation of the Non-Exhaustive Polarity Selection Algorithm.

Figure 14. The initial segment of the calculation of the levels of the Butterfly for an incompletely specified function in the Non-Exhaustive Polarity Selection Algorithm. Arrows show the selection of columns for BR rule.

Figure 15. Part of a diagram showing setting of individual basis functions in universal modules.

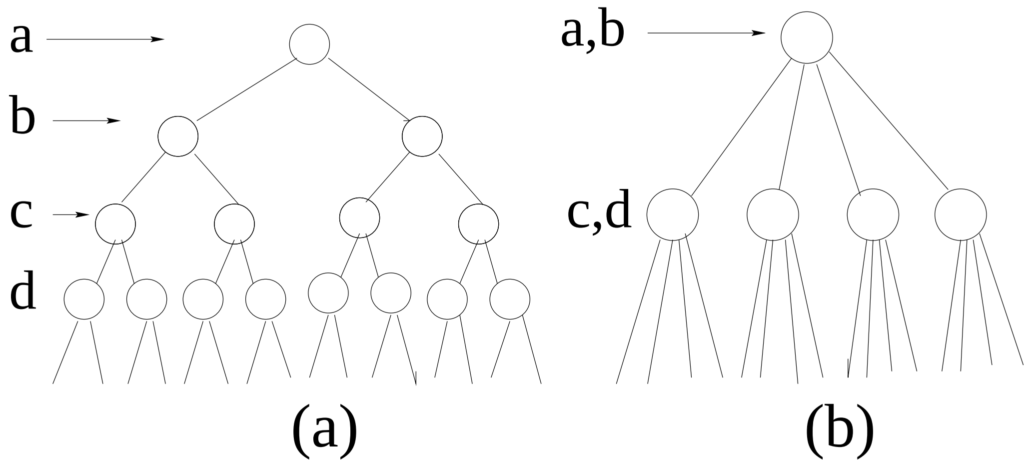


Figure 1: *Standard and multi-variable expansion trees*

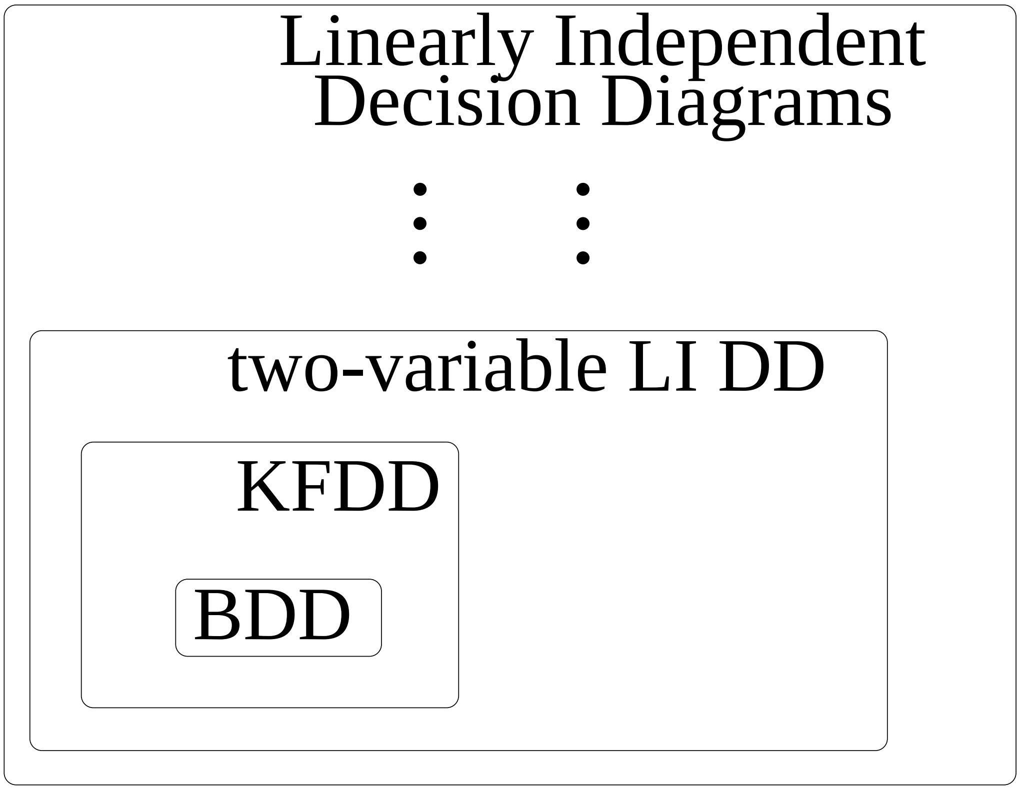


Figure 2: *Simplified hierarchy of decision diagrams*

| 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 3: *Function $f(A, B, C, D)$ to Example 2.1*

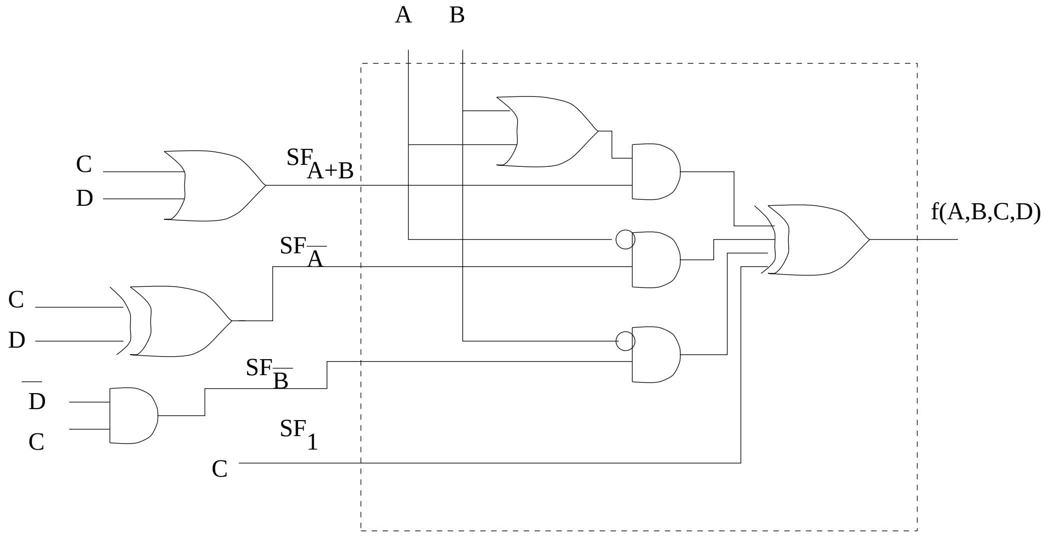


Figure 4: *Universal nonsingular expansion module for basis functions from Example 2.1, applied to the function from Figure 3. The data functions* $SF_{A+B}$, $SF_{\overline{b}}$, $SF_{\overline{A}}$, *and* $SF_1$ *are from the left, and the universal module for basis functions* $A+B$, $\overline{B}$, $\overline{B}$, $\overline{A}$, *and* 1 *is shown in a dotted rectangle.*

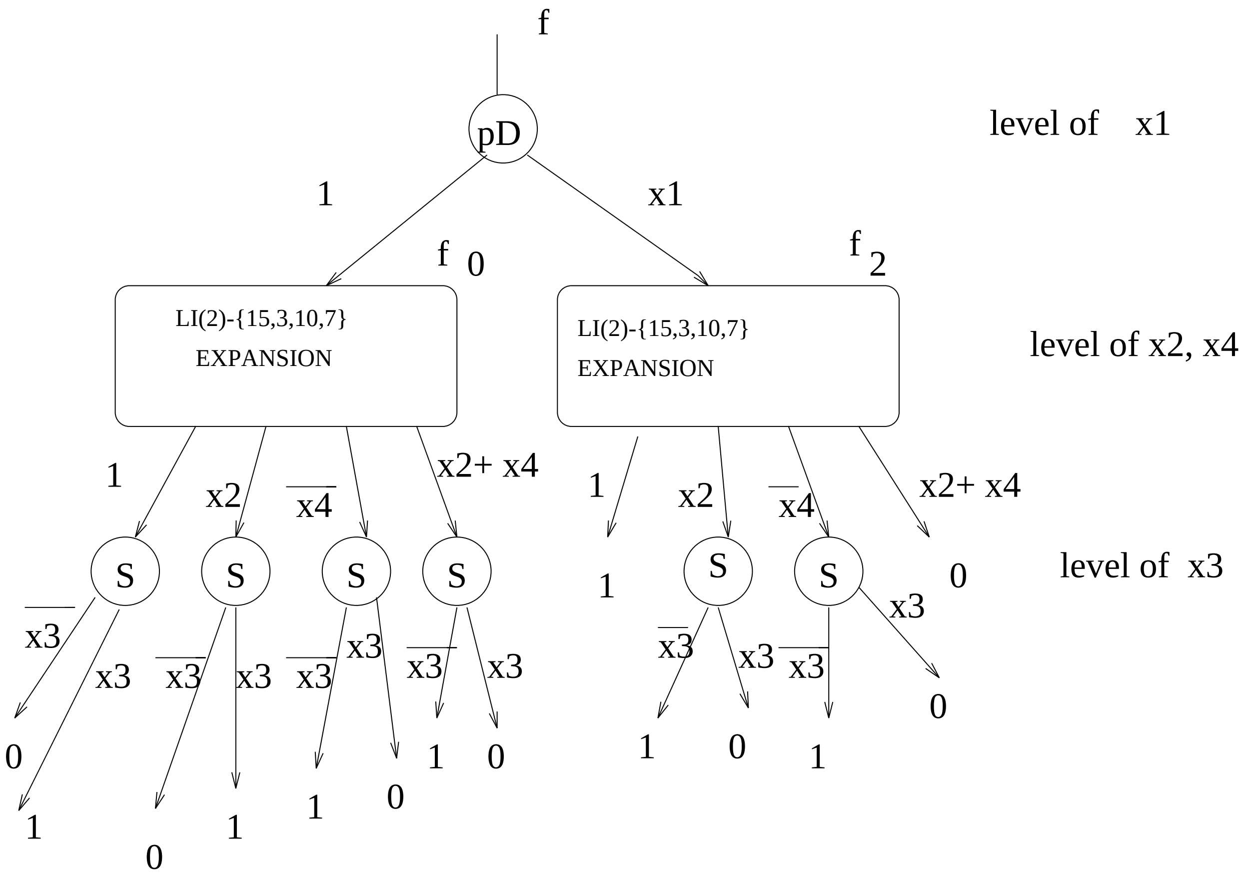


Figure 5: *Example of a LI Kronecker Tree with blocks* $\{x_1\}$, $\{x_2, x_4\}$, *and* $\{x_3\}$

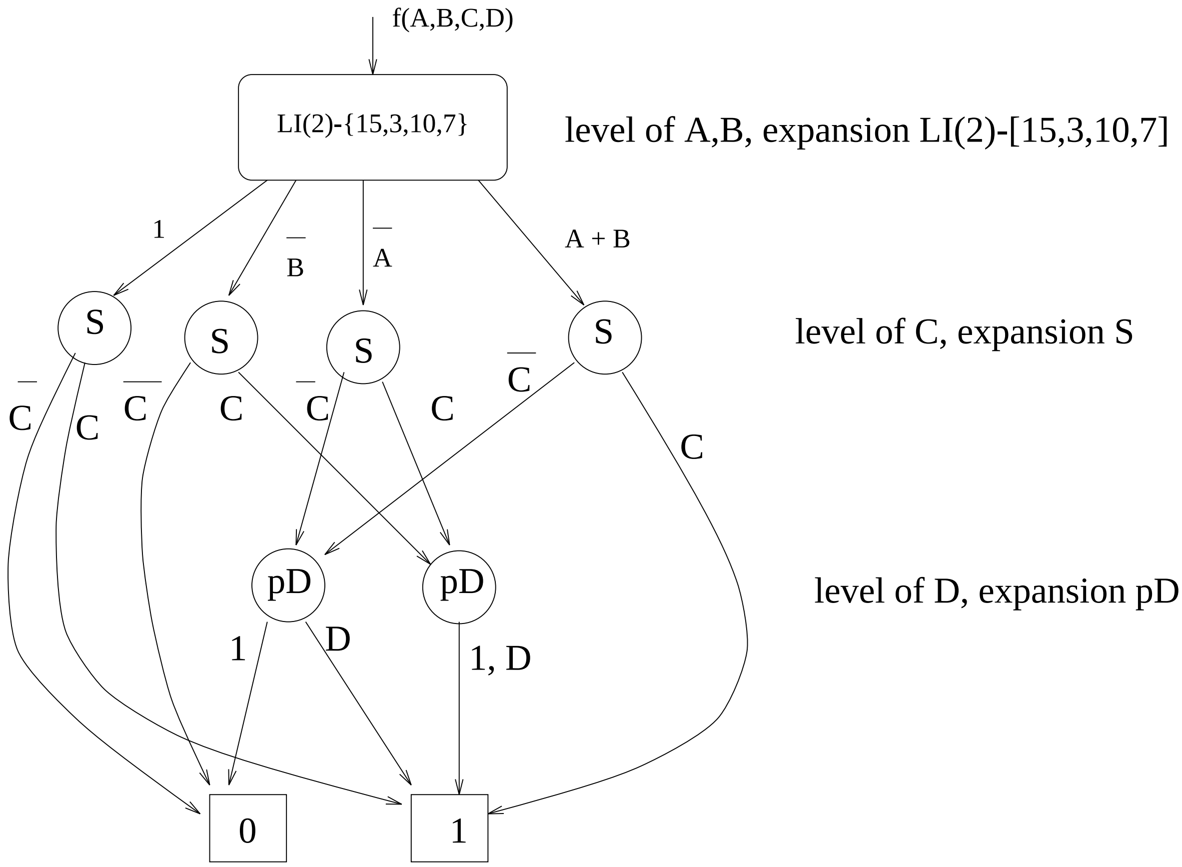


Figure 6: *A "mixed-variable" LI Kronecker Decision Diagram for function from Example 2.1. Level 1 corresponds to a pair of variables $\{A, B\}$, level 2 to a single variable $C$ and level 3 to a single variable $C$. Because in every level expansion types are the same, this is a "Kronecker" type of diagram.*

| 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 |

G,H

Figure 7: *Six-input, two-output function to Example 5.3.*

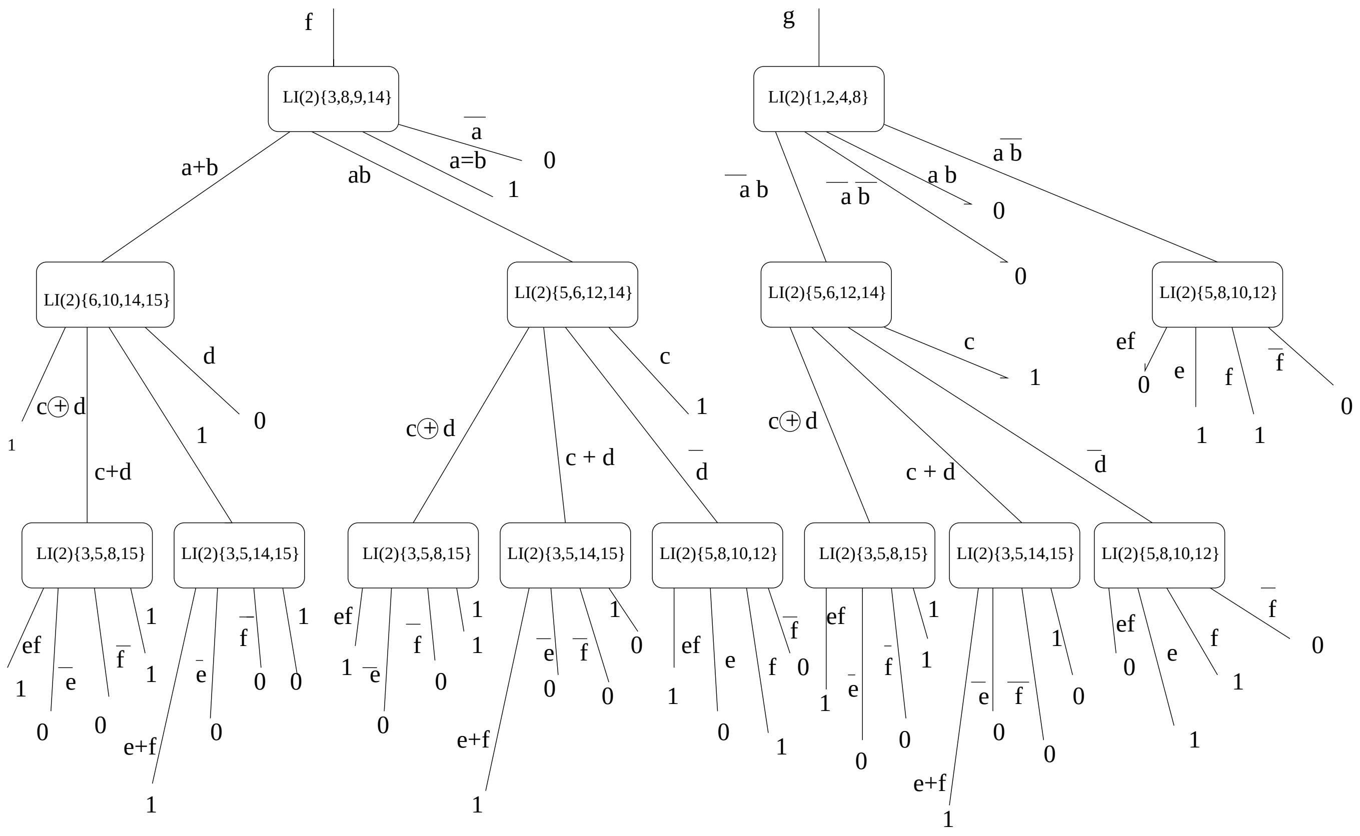


Figure 8: *A LIPKT for blocks* $\{a,b\}$,$\{c,d\}$,$\{e,f\}$ *to Example 5.3.*

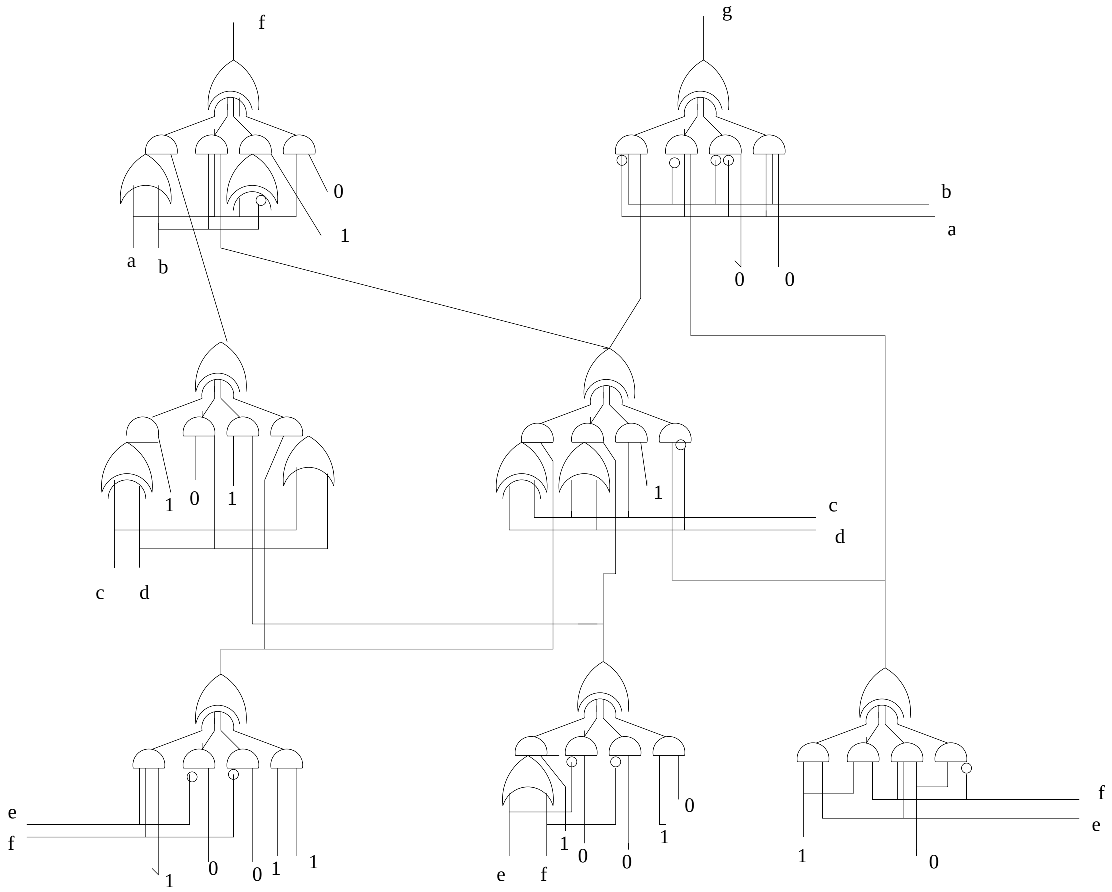


Figure 9: *A LI Shared and Ordered Pseudo-Kronecker Decision Diagram (SOLIPKDD) for blocks {a,b},{c,d},{e,f} to Example 5.3. It is drawn with the expansion nodes substituted by their respective universal module circuits, in order to explain how the final circuit from Figure 10 can be obtained from such a diagram.*

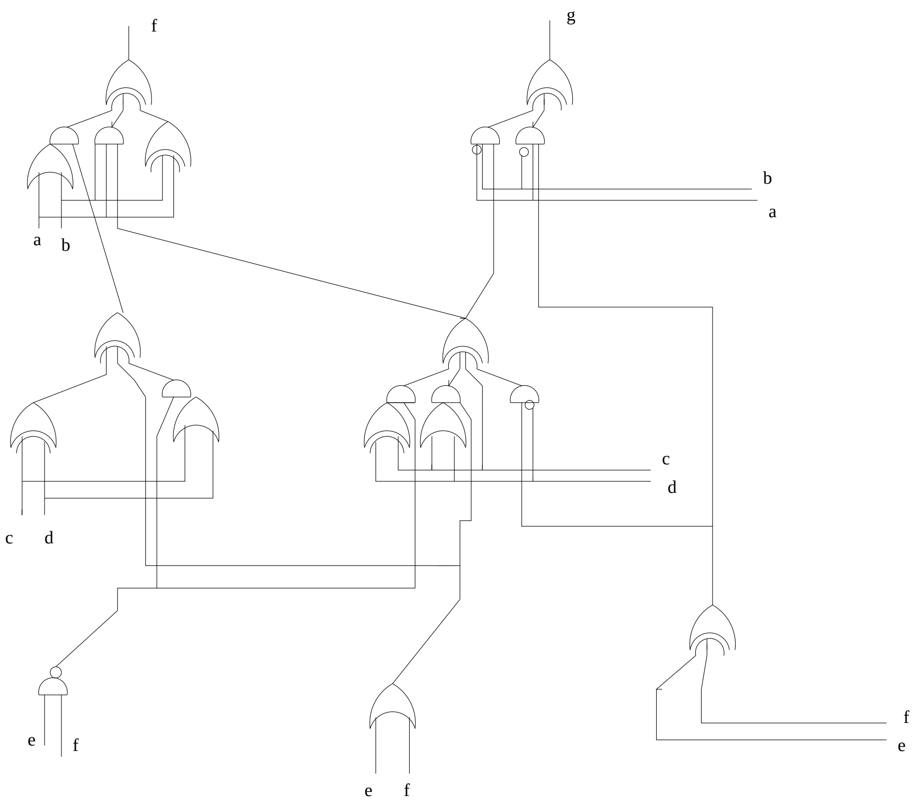


Figure 10: *A multi-level circuit to Example 5.3 obtained from the LI Shared and Ordered Pseudo-Kronecker DD after substituting circuits of universal modules to nodes of the diagram and propagation of constants.*

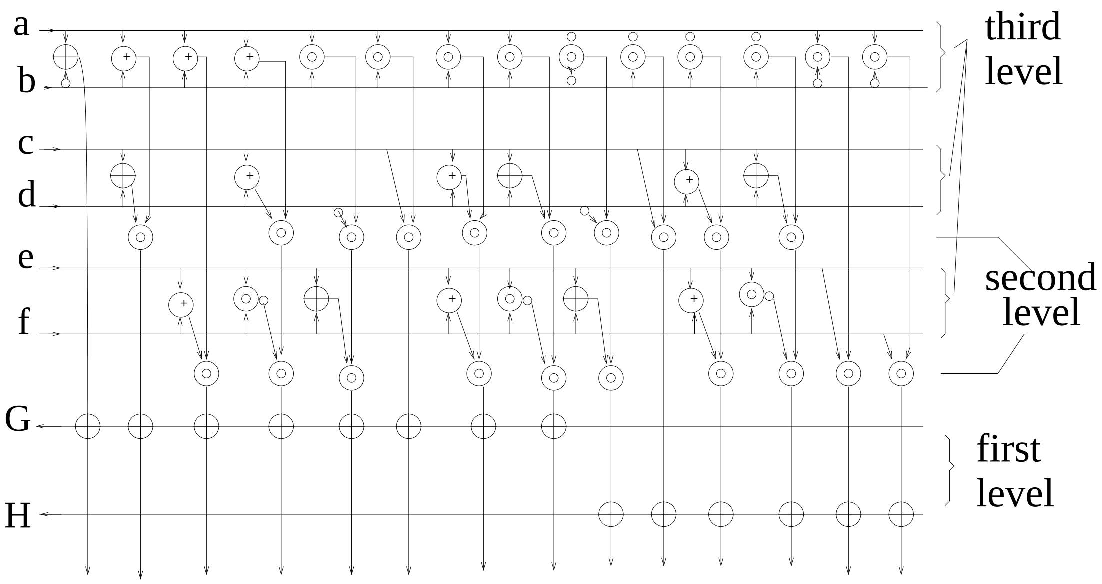


Figure 11: *A three-level LI PLA to Example 5.3 obtained from the SOLIPKDD from Figure 9. Large circles denote two-input gates. Small cross in a circle denotes an OR gate, and large cross an EXOR gate. Small circle in the large circle denotes an AND gate, and small circle outside large circle denotes an invertor.*

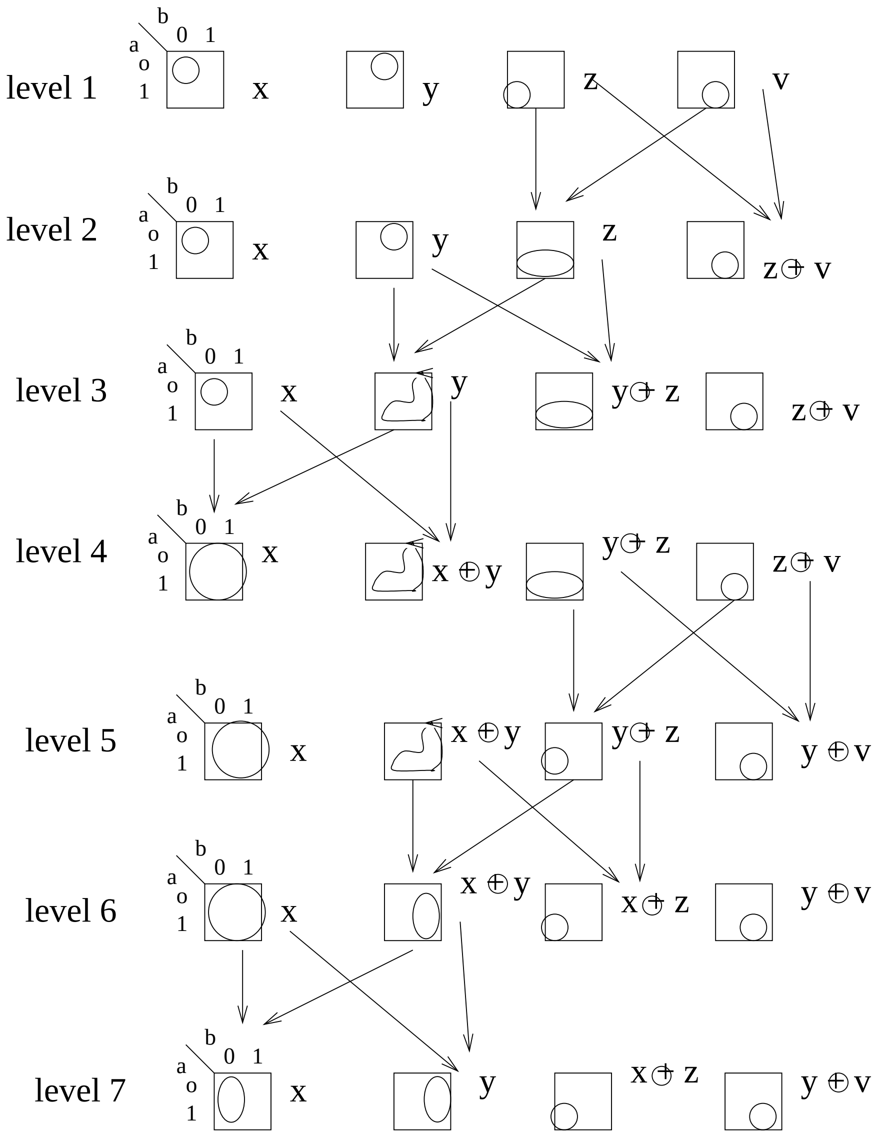


Figure 12: *An initial segment of a Butterfly Diagram to create nonsingular expansions for all LI functions of a,b.*

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

Figure 13: *The Kmap to explain the operation of the Non-Exhaustive Polarity Selection Algorithm.*

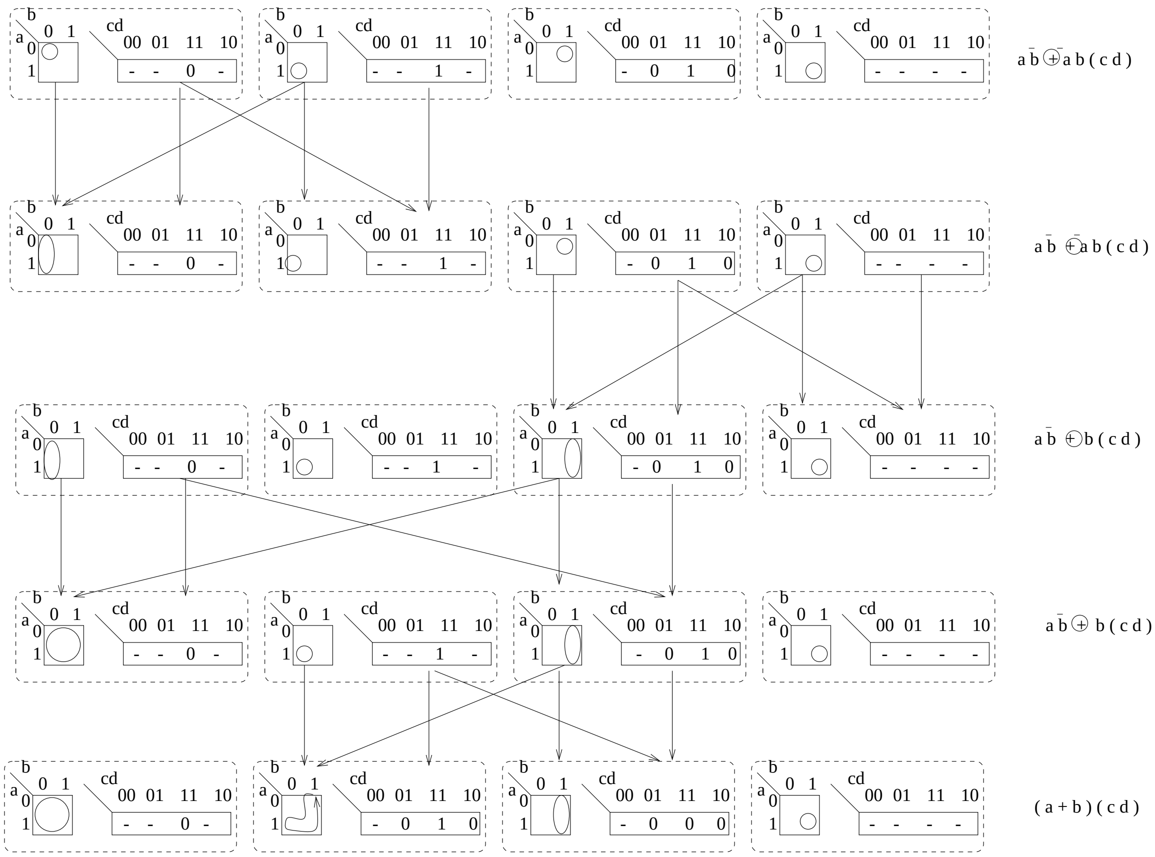

Figure 14: *The initial segment of the calculation of the levels of the Butterfly for an incompletely specified function in the Non-Exhaustive Polarity Selection Algorithm. Arrows show the selection of columns for BR rule.*

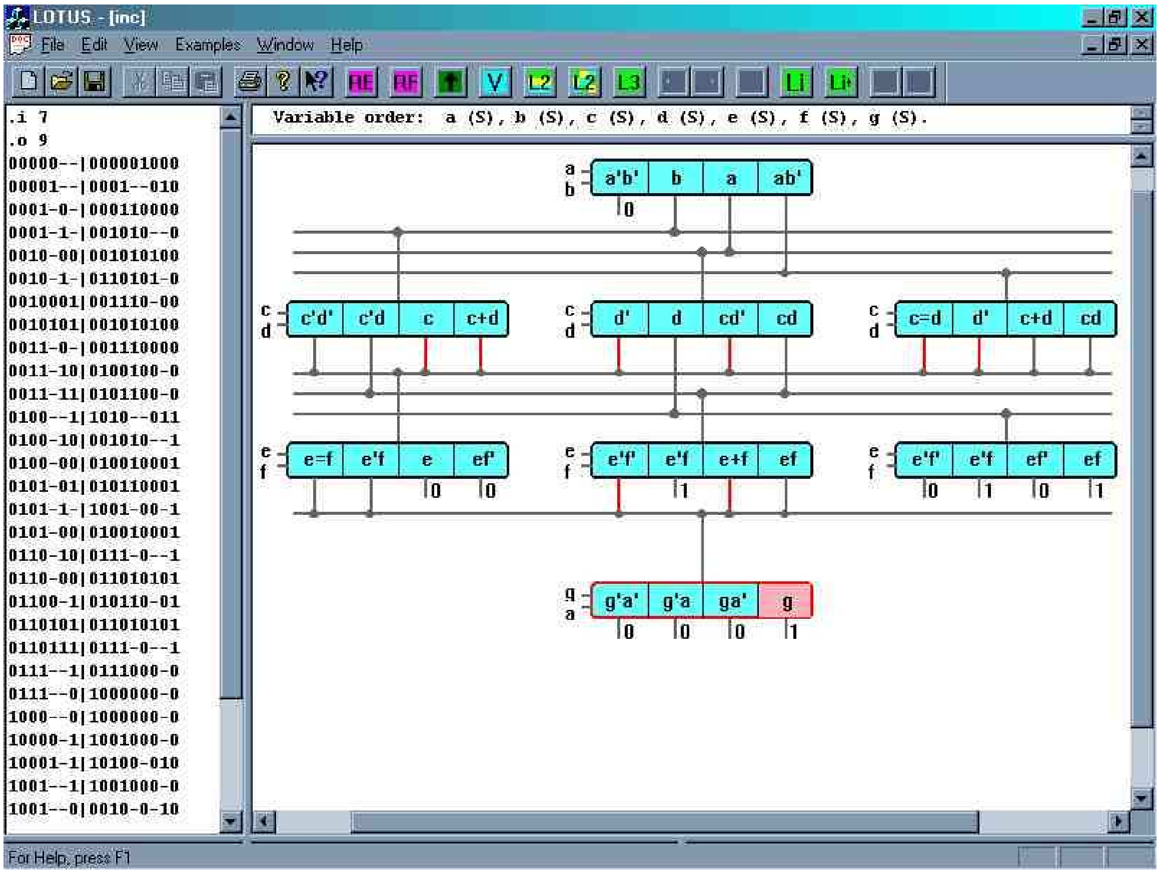


Figure 15: *Part of a diagram showing setting of individual basis functions in universal modules.*

Marek Perkowski has his Ph.D. in Automatic Control from the Technical University of Warsaw, Warsaw, Poland. He served on the faculties of Technical University of Warsaw and University of Minnesota. Currently he is professor of Electrical and Computer Engineering at Portland State University, Portland, Oregon. He has been a visiting professor at the University of Montpellier, France, and Technical University of Eindhoven, The Netherlands. He was also a summer professor and consultant at Wright Laboratories of U.S. Air Force, GTE, Intel, Cypress, Sharp, and other high technology and EDA companies. Dr. Perkowski is the general chair of International Symposium on Multiple-Valued Logic, 2000, the Vice-Chair for Technical Activities of IEEE Technical Committee on MVL, and the Chair of Fourth Oregon Symposium on Logic, Design and Learning, 2001. His recent research interests include spectral decision diagrams, functional decomposition, intelligent robotics, walking robots, robot theatre, axiomatic morality, and all applications of logic synthesis outside circuit design.