TWO HIERARCHIES OF GENERALIZED KRONECKER TREES, FORMS, DECISION DIAGRAMS, AND REGULAR LAYOUTS

Marek Perkowski, Lech Jozwiak †, Rolf Drechsler +,

Portland State University, Dept. of Electr. Engn., Portland, Oregon 97207,
Tel: 503-725-5411, Fax: 503-725-4882, *mperkows@ee.pdx.edu*
† Faculty of Electrical Engineering, Eindhoven University of Technology,
P.O.Box 513, 5600 MB Eindhoven, The Netherlands, *lech@eb.ele.tue.nl*
+ Institute of Computer Science, Albert-Ludwigs-University,
79110 Freiburg in Breisgau, Germany, *drechsle@informatik.uni-freiburg.de*

Abstract— The paper presents two hierarchies of canonical AND/EXOR trees, forms and decision diagrams. The first hierarchy generalizes the Kronecker and Generalized Kronecker representations by introducing new canonical AND/EXOR forms. We propose to call all these new forms and future AND/EXOR forms including KRO and GRM, the Zhegalkin forms [42] to honor the Russian scientist who in 1927 discovered the forms now attributed to Reed and Muller and invented by them in 1954. The new Zhegalkin representations and forms can be used for synthesis of quasi-minimum ESOP circuits and the new diagrams can represent large functions and can be used for optimal synthesis of highly testable multilevel circuits in several technologies, especially in Fine Grain Field Programmable Gate Arrays. The second hierarchy generalizes and extends the Universal Akers Array to expansions other than Shannon and neighborhoods other then 2-inputs, 2-outputs. These diagrams are called Zhegalkin Lattice Diagrams.

I. INTRODUCTION

In recent years Decision Diagrams (DDs) have revolutionised the representation and processing of binary logic functions. Because of the processing speed and relatively small memory requirements, the DDs are widely used in logic synthesis, verification and simulation. Many modern design automation systems use Binary Decision Diagrams (BDDs) as the main internal representation of functions, on which all meaningful operations are executed. DDs originate from binary decision trees (binary expansion trees, Shannon trees), which in turn are based on the fundamental expansion theorem of Shannon that is applied in every node of a tree. Every node is related to one input variable of the function. The well-known diagrams are "reduced", and they are "ordered", which means that the variables of nodes in the tree are ordered in the same way in all branches.

BDDs [2] have an important disadvantage - some very large functions of practical interest cannot be represented by them, because the number of nodes of the tree becomes too large. Therefore, very early, the research on the representation of Boolean functions consisted in generalising the concept of a binary tree, and we will follow this approach below.

There exists a hierarchy of AND/EXOR representations investigated previously in [10, 16, 33]. We will call it the *Green/Sasao hierarchy*, because while the original paper of Green investigated only two-level forms, the paper of Sasao interlinked forms with corresponding trees and diagrams. All these representations 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" [34, 12, 40, 37].

The hierarchy is based on three expansions:

$f(x_1, x_2, ..., x_n) \;=\; \overline{x}_1 f_0(x_2, ..., x_n) \oplus x_1 f_1(x_2, ..., x_n)$
in short $f = \overline{x}_1 f_0 \oplus x_1 f_1$, called Shannon, (1.1)

$f(x_1, x_2, ..., x_n) \;=\; 1 \cdot f_0(x_2, ..., x_n) \oplus x_1 f_2(x_2, ..., x_n)$
in short $f = f_0 \oplus x_1 f_2$, called Positive Davio, (1.2)

$f(x_1, x_2, ..., x_n) \;=\; 1 \cdot f_1(x_2, ..., x_n) \oplus \overline{x}_1 f_2(x_2, ..., x_n)$
in short $f = f_1 \oplus \overline{x}_1 f_2$, called Negative Davio, (1.3)

where f_0 is f with x_1 replaced by 0 (negative cofactor of variable x_1), f_1 is f with x_1 replaced by 1 (positive cofactor of variable x_1), and $f_2 \;=\; f_0 \;\oplus\; f_1$.

By applying recursively expansions (1.1) - (1.3) (or any subset of them) to the function, next to its cofactors with respect to the first level variable of the function, then to second level variable cofactors, to third level cofactors, and so on, various types of binary decision trees can be created [33]. Sasao characterized those representations in an uniform way that we will use here in our generalizations. The concepts of **Shannon Trees, Positive Davio Trees, Negative Davio Trees, Kronecker Trees, Reed-Muller Trees, Pseudo-Kronecker Trees,** and **Pseudo Reed-Muller Trees**, as well as of the corresponding decision diagrams and flattened (two-level) canonical forms are discussed in [33, 7, 14, 15, 32, 38]. In addition, the **Free Kronecker Trees** that use S, pD and nD nodes disregarding any order of variables and expansions are discussed in [11]. At every tree level, different variables and expansions can occur. Thus, the order of variables in every branch can be different, and such diagrams are also called non-ordered. Similarly, one can also define Free Binary Decision Trees (leading to Free BDDs) and Free Positive Davio Trees (leading to Free FDDs). Free Kronecker Trees lead to Free KFDDs (FKFDDs) defined and investigated in [11].

The Kronecker DDs have been further generalised to more general data structures, K*BMDs [9]. These diagrams use *arithmetic* operations of addition and multiplication instead of binary logic operations. K*BMD diagrams find applications in verification of digital systems, and for solving general problems in discrete mathematics.

Below, we will propose another approach to the generalization of Kronecker diagrams: our generalization, however, still uses binary operators, so it is useful for logic synthesis.

In [24] the Generalized Kronecker Trees, Forms and Decision Diagrams that unify Kronecker and Generalized Reed Muller representations have been proposed. These representations are better then the previously proposed because the previous representations did not allow to create GRM forms after flattening. Here, further generalizations of the representations from [24] are proposed, creating thus an enhanced Green/Sasao hierarchy. The flattened forms are used to design the minimized Exclusive Sum of Products (ESOP) circuits, and the multi-level representations are used to design multi-level AND/EXOR circuits, that next through the "EXOR-related technology mapping" are adjusted to AND/OR/EXOR custom VLSI, standard cell, or FPGA technologies. Because of the superioriority of new representations, the circuits obtained as above are also never worse than the AND/EXOR circuits obtained from the previously known representations, (including the GKTs [24]).

In section II various kinds of trees with multi-variable nodes are introduced. We review the concept of the Generalized Kronecker Trees and introduce the Generalized Kronecker Diagrams and their "pseudo"-like generalizations along the same lines, as the Pseudo Kronecker representations were created to extend the Kronecker representations. In section III two extended Green/Sasao hierarchies of representations are presented. Section IV presents the Zhegalkin Lattice Diagrams hierarchy. These diagrams map logic functions to regular grid on two-dimensional space and allow to efficiently realize some kinds of functions, especially totally symmetrical, partially symmetrical or nearly symmetrical multi-output functions, functions with many don't cares, and multi-output Boolean relations. Section V concludes the paper.

II. THE FAMILY OF GENERALIZED KRONECKER TREES, CANONICAL FORMS, AND DECISION DIAGRAMS

Definition 1. The **Generalised Kronecker Tree** is a multi-branch tree created as follows:

1) The set of all n input variables is partitioned into disjoint and nonempty subsets S_j such that the union of all these subsets forms the initial set. (If each subset includes just a single variable, the tree reduces to the special case of a KRO tree. If there is only one subset that includes all variables, the tree corresponds to the special case of a GRM.)

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

3) For every level, either S, nD, or pD expansion is selected for its nodes if the set involves a single variable, or the expansion (2.2) below with the same polarity for all variables from S_j is applied to all nodes of the level if the set is multi-variable,

4) The formula for expansion (2.2) introduced for the first time in [24] generalizes the following formula (2.1) for a Generalized Reed-Muller expansion:

$f(x_1, \ldots, x_n) \;=\; a_0 \oplus a_1\hat{x}_1 \oplus a_2\hat{x}_2 \oplus \ldots \oplus a_n\hat{x}_n \oplus a_{12}\hat{x}_1\hat{x}_2 \oplus a_{13}\hat{x}_1\hat{x}_3 \oplus \ldots \oplus a_{n-1,n}\hat{x}_{n-1}\hat{x}_n \oplus \ldots \oplus a_{12\ldots n}\hat{x}_1\hat{x}_2\hat{x}_3 \ldots \hat{x}_n$
(2.1)

where a_i's are either 0 or 1, and $\hat{x}$ denotes variable x or its negation, $\overline{x}$. Thus by assigning a variable or a negation of a variable to each of the $\hat{x}_i$ in (2.1) we create $2^{n2^{n-1}}$ different expansion formulas. Each of them is called *a polarity expansion*, i.e., an expansion of a certain polarity.

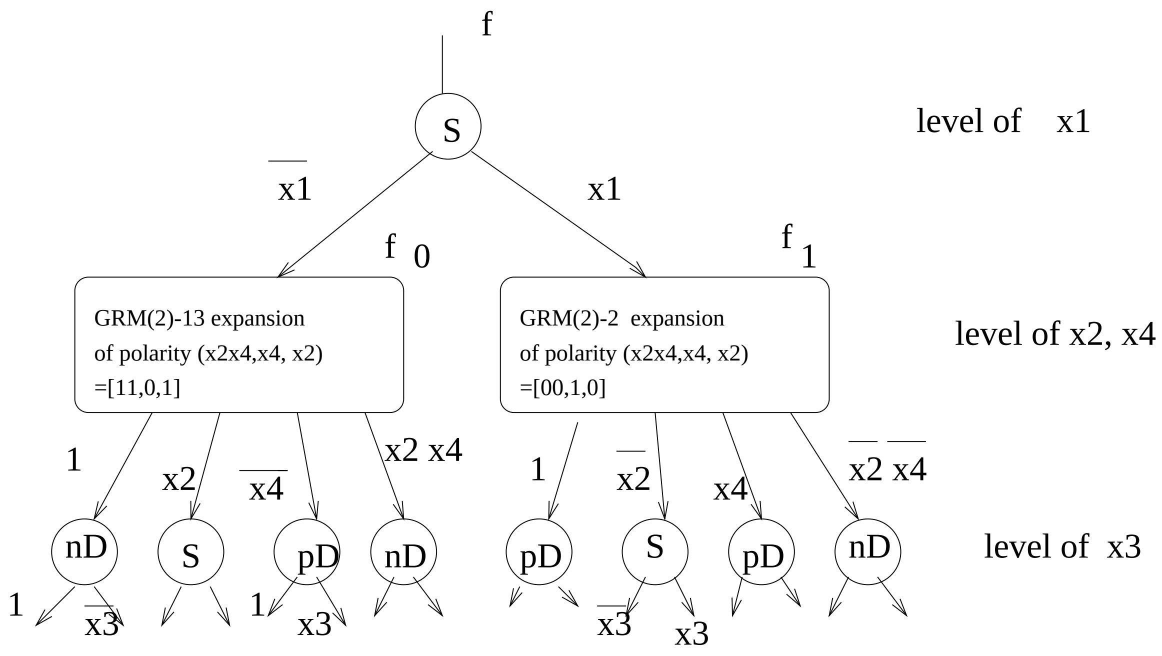

Figure 1: *Example of Pseudo Generalized Kronecker Tree*

The expansion formula (2.2) for function $f(x_1, x_2, ..., x_m, ..., x_n)$ is the same as GRM expansion, (2.1), with respect to variables $x_1, ..., x_m$ and coefficients a_i replaced with subfunctions SF_i of remaining variables $x_{m+1}, ...x_n$:

$$f(x_1, x_2, ..., x_m, ..., x_n) = SF_0(x_{m+1}, ..., x_n) \oplus \hat{x}_1 SF_1(x_{m+1}, ...x_n) \oplus \hat{x}_2 SF_2(x_{m+1}, ..., x_n) \oplus \ldots \oplus \hat{x}_m SF_m(x_{m+1}, ..., x_n)$$
$$\oplus \hat{x}_1\hat{x}_2 SF_{12}(x_{m+1}, ..., x_n) \oplus \hat{x}_1\hat{x}_3 SF_{13}(x_{m+1}, ..., x_n) \oplus \ldots \hat{x}_{m-1}\hat{x}_m SF_{m-1,m}(x_{m+1}, ..., x_n)$$
$$\oplus \ldots \oplus \hat{x}_1\hat{x}_2\hat{x}_3 \ldots \hat{x}_m SF_{12\ldots n}(x_{m+1}, ..., x_n) \qquad (2.2)$$

where the so-called *Data Functions* SF_i are calculated as coordinates of a vector CV $= M^{-1} \times$ FV, in which:

$FV(x_{m+1}, ..., x_n)$ is a vector of cofactors of F with respect to variables from the set $\{x_1, ..., x_m\}$.

A $2^m \times 2^m$ nonsingular matrix M has as its columns all the products of literals $\hat{x_i}$ that have been used in one of $2^{m2^{m-1}}$ particular polarity expansion forms specified by (2.2). The columns of the matrix are linearly independent with respect to the bit-by-bit exoring operation (for an example see [24]).

Definition 2. The **Pseudo Generalised Kronecker Tree** is a tree with multi-variable expansion nodes created as follows

1) The set of all n input variables is partitioned to disjoint and nonempty subsets S_j such that the union of these subsets forms the initial set (If each subset has just a single variable, the special case of PKRM is considered.)

2) For every node of the tree, in a level with variable subset S_j, **any** GRM expansion for all variables from S_j can be performed.

Example 2.1. An example of the Pseudo Generalised Kronecker Tree (PGKT) can be found in Figure 1. The first level of the tree has Shannon expansion for variable x_1, the second level has GRM(2)-13 and GRM(2)-2 expansions for set of variables $\{ x_2, x_4 \}$ and the third level has S, pD, and nD expansions for variable x_3. The output function of a node is shown by it, the edges on the bottom of the node correspond to the two inputs of a node - these are the (respective to the expansion type) functions from (1.1) - (1.3) and are taken from the set f_0, f_1, f_2. The expansion literals are shown near the input edge(s). Label 1 corresponds to function f_0 in pD and f_1 in nD. Label $\overline{x}_1$ to f_0 in S and f_2 in nD. Label x_1 to f_1 in S and f_2 in pD. The nodes denoted by S denote Shannon Expansions (Equation (1.1)) In them, the Shannon Expansion with inclusive OR operator can be replaced with the expansion with EXOR operator. This can be done because functions $\overline{x}_1 f_0$ and $x_1 f_1$ are disjoint. Nodes pD denote the Positive Davio Expansion (Equation (1.2)), and nodes nD denote the negative Davio Expansion (Equation (1.3)). Observe that S uses both a variable and its negation, pD uses only the positive literal of the variable, and nD uses only the negative literal of the variable. The circuit realisation of S is a multiplexer. Realisation of pD includes a two-input AND and a two-input EXOR gate and is called AND/EXOR gate. Realisation of nD includes a two-input AND gate with inverted input for the control variable x_1, called the inhibition gate, and the EXOR gate. It is called the Inhibition/EXOR gate. All such gates exist in Fine Grain FPGAs and in modern VLSI libraries.

The notation for the multi-variable GRM nodes is described below. As we can observe in Figure 1, the number 13 in expansion name (polarity) "GRM(2)-13" is a natural number corresponding to the binary number 1101, called a *polarity*, in which the rightmost 1 corresponds to the positive polarity of single variable x_2, 0 corresponds to the negative polarity of variable x_4, and the leftmost two ones correspond to the positive polarities of variables x_2 and x_4 in a two variable AND term for variables x_2 and x_4. The expansion of the node GRM(2)-13 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.

The GRM expansion in node GRM(2)-2 (for polarity [00,1,0]) is described by the formula:

$f_1(x_2,x_3,x_4) = SF(f_1)_1(x_3) \oplus \overline{x}_2 SF(f_1)_{\overline{x}_2}(x_3) \oplus x_4 SF(f_1)_{x_4}(x_3) \oplus \overline{x}_2\overline{x}_4 SF(f_0)_{\overline{x}_2\overline{x}_4}(x_3)$

It can be observed that the concept of Pseudo GKTs can be further extended by allowing in the second level of the same tree the mixture of both single variable and two-variable expansion nodes. Such expansions will be called the Mixed Pseudo GKTs (MPGKTs). If, for instance, the GRM(2)-2 node in Figure 1 were replaced with a two-level tree with S node for variable x_2 and pD nodes for variable x_4, an example of a MPGKT would be created.

Definition 3. The **Mixed Pseudo Generalised Kronecker Tree** (MPGKT) is any multi-branch tree created as follows.

1) The set of all n input variables is partitioned to disjoint subsets (blocks) S_j of variables.

2) For every multi-variable set, either apply to a root node in the level **an** arbitrary GRM expansion of *all* its variables or create a subtree of single-variable expansions of variables from S_j. Ordered sets S_j are assumed. For a single-variable level of the tree, any combination of the S, nD and pD nodes can be applied.

The most general is the category of free trees.

Definition 4. The **Free Generalised Kronecker Tree** (FGKT) is any multi-branch tree created as follows. For every node of the tree, **an** arbitrary size subset of input variables can be selected. For single variable sets - an S, pD or nD node can be created, for multi-variable sets - the GRM expansion of its variables of any polarity is calculated. The levels are no more associated with variables or their sets, various local orders and partitions of variables may exist in the branches.

Definition 5. The **Ordered Generalised Kronecker Tree** (OGKT) is any multi-branch tree created as a Free Tree, with the additional constraint that every branch has the same order of variables.

In OGKT the sets of variables in different branches may have different sizes and overlap, but the order must be the same. For instance, in one branch the first set is $\{x_0,x_1,x_2\}$ the second set is $\{x_3,x_4\}$, and the third set is $\{x_5,x_6\}$. In another branch the sets are: $\{x_0\}$, $\{x_1\}$, $\{x_2,x_3\}$, and $\{x_4,x_5,x_6\}$.

The schemata of various types of GKTs are shown in Figure 2 and Figure 3. Figure 2 presents various types of Ordered Generalized Kronecker Trees. Figure 3 has a scheme of a Free GKT. One can appreciate various orders of variables in branches here.

Canonical AND/EXOR Forms are obtained by flattening respective trees. Flattening means finding the AND-terms by following all the paths from the root to all the leafs. This way an AND/EXOR two level expression is created that is equivalent to the tree and that is a canonical form.

Canonical AND/EXOR Decision Diagrams are obtained by combining isomorphic nodes (nodes that correspond to the same subfunctions), and removing nodes that are in some sense redundant. For instance, a node with two inputs originating from the same node is removed [33]. Similarly, a node with four inputs originating from the same node is removed [24]. Transformations for S, pD and nD nodes from [33] are applied to single-variable nodes. Their multi-variable node generalizations, presented in [24] are applied to multi-variable nodes. All these transformations generalize the OKFDD transformations from [33]. This way, the definitions of all flattened forms and decision diagrams can be obtained analogously as in [24].

Example 2.2. This example demonstrates a function fi in which the PGKT is better than the GKT. The PGKT for fi has Shannon node for variable a on top, GRM node for cofactor $fi_{\overline{a}}$ for expansion variables b, c, and GRM node for cofactor fi_a for expansion variables b, c. The expansion diagram is

$fi = \overline{a}(1 \cdot x \oplus b \cdot y \oplus c \cdot z \oplus bc \cdot v) \oplus a(1 \cdot v \oplus \overline{c} \cdot y \oplus \overline{b} \cdot z \oplus \overline{b}\,\overline{c} \cdot x).$

Thus, it has two GRM nodes, and 5 nodes for variables a, x, y, z, v (the variable-nodes for variables x, y, z, v are shared). Observe that in the subdiagram for $fi_{\overline{a}}$ the polarity is [11,1,1] and in the subdiagram for fi_a the polarity is [00,0,0]. It can be shown, that it is not possible to have a smaller diagram for $fi_{\overline{a}}$, because each change to another polarity would require exoring some of variables x, y, z, v and this would lead some extra exor nodes, which is to more nodes than four for only the variable-nodes. Similarly, changing the polarities in the subdiagram for a will always lead to diagrams with more nodes than in the above solution. Since both subdiagrams are worse, the entire diagram is worse by having the same expansion in both subdiagrams. Similarly it can be shown that the variable a must be on top, and, in general, there exists no GKT that is better or equal to fi.

Example 2.3. This example demonstrates a function fk for which the free GKT is better than the ordered GKT. The free GKT for fk has Shannon node for variable a on top, GRM node for cofactor $fk_{\overline{a}}$ for expansion variables b, c, and GRM node for cofactor fk_a for expansion variables x, y. The expansion diagram is

$fk = \overline{a}(1 \cdot x \oplus b \cdot y \oplus c \cdot z \oplus bc \cdot v) \oplus a(1 \cdot c \oplus x \cdot 0 \oplus y \cdot 0 \oplus xy \cdot c)$.

Thus, it has two GRM nodes, and 6 nodes for variables x, y, z, v and c (the node for c is shared). Observe that in the subdiagram for $fk_{\overline{a}}$ the order of variables is $\{b, c\}$ before $\{x, y, z, v\}$, and in the subdiagram for fk_a the order is $\{x, y\}$ before $\{b, c, z, v\}$. It can be shown, that changing the orders in the subdiagram for a to the orders with $\{b, c\}$ before $\{x, y, z, v\}$ will bring trees with higher node costs (we calculate the cost of the 2-variable node as three standard nodes). For instance, there would be one more standard node in the solution:

$fk = \overline{a}(1 \cdot x \oplus b \cdot y \oplus c \cdot z \oplus bc \cdot v) \oplus a(1 \cdot 0 \oplus c \cdot (xy \oplus 1) \oplus b \cdot 0 \oplus bc \cdot 0)$.

Similarly it can be shown that all possible changes of orders in the subtrees to the same order in both subtrees will always result in higher total node costs.

Examples of forms with special properties, like superiority of GKE with respect to PSDKRO, PGKE with respect to GKE, MPGKE with respect to PGKE, and so on, can be created using Theorem 3 in [24]. This way, the hierarchy from Fig. 4 has been also created. The mutual relationships of PSDKROs and GKEs, and FKROs and GKE were for instance investigated: (1) *"are there any PSDKRO's that are not in GKE?"*, (2) *"are there any FKRO's that are not in GKE?"* and positive answers were given to them. Another method to create examples for forms and trees to investigate relations between families is to create an expression for the form for which we want to prove the superiority, and next create from it a multi-output function with as many outputs as the form has product terms, each of its single outputs being a product equal to one product term from the form.

The idea of separating variables to "Kronecker-type" variables and "GRM-type variables" in a canonical AND/EXOR representation, introduced in [24] allows to create very fast ESOP minimizers based on algorithmic ideas introduced in [24, 25, 40, 5, 6], in which function is represented by a flat vector instead of a tree or a DAG.

The concepts presented here can be applied for the synthesis of easily testable two-level AND/EXOR circuits. Obviously, in addition to the expansion forms from KRO and GRM, there are many other new canonical expansion forms resulting from the flattening of the new trees. Furthermore, variants of non-ordered sets S_j can also be considered. ESOP circuits are AND/EXOR two-level circuits with not any constraints imposed on the products of literals - they are thus the best AND/EXOR circuits but they are not canonical. Much research has been devoted to the synthesis of minimal ESOP circuits [39, 35], but so far the exact solution can be obtained only for a very small number of input variables [36]. The quality of AND/EXOR circuits obtained from the expansions proposed here should be significantly better that those corresponding to GRMs or (Pseudo) KROs because the search space of GKTs is much larger than the search space of the GRM expansions. For the case of many variables, it will be, however, not possible to search it exhaustively, even for 8-9 variables, because of the large size of this new space. Therefore the practical usefulness of the proposed ESOP minimization method based on the GKTs and their extensions will depend on the quality of search heuristics to partition and to order the input variables.

III. ENHANCING THE GREEN/SASAO HIERARCHY WITH NEW REPRESENTATIONS

Table 1 shows relationships of the canonical trees, the canonical expressions, and the decision diagrams created from these trees by applying the reduction rules. This table adds new AND/EXOR representations, Zhegalkin Forms [1], to the table from [33]. It should be noticed, that not all known EXOR-based representations are included here, for instance some of those from papers [16, 22, 33, 40] are not included. However, the goal was to compare here the new representations only to those AND/EXOR representations that are popularly known and have been thoroughly investigated in the past. Non-canonical representation have also been not included.

Figure 4 shows the set-theoretical relationships among the known Reed-Muller and the (defined above) new classes of Zhegalkin canonical forms. Observe, that we illustrate here several new families of forms that stand between GRM

[1]Using the name "Zhegalkin" is also useful practically, because otherwise very long composite names such as "Generalized Pseudo Mixed Kronecker Reed-Muller" should be introduced

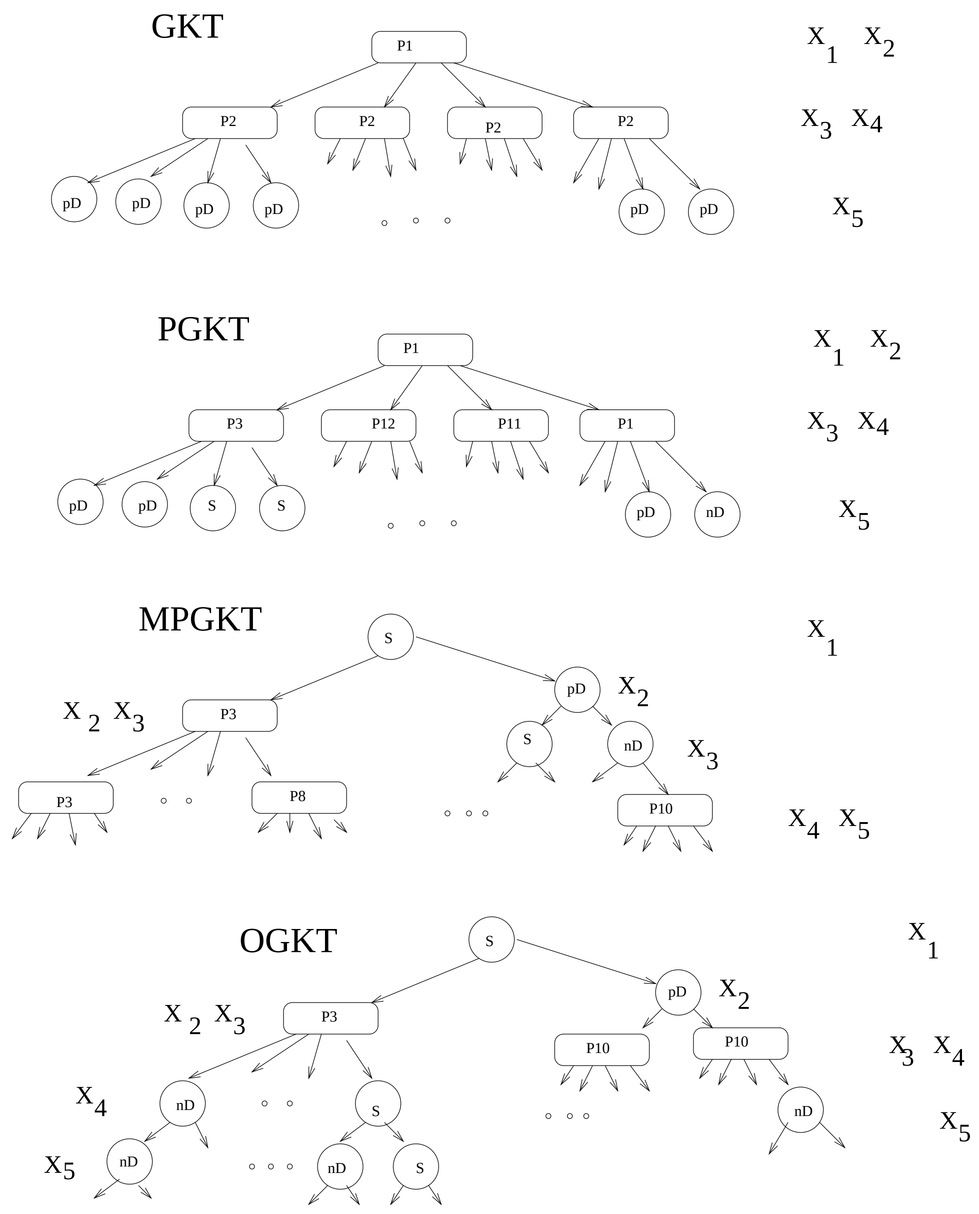

Figure 2: *Schemata of various types of Ordered Generalized Kronecker trees*

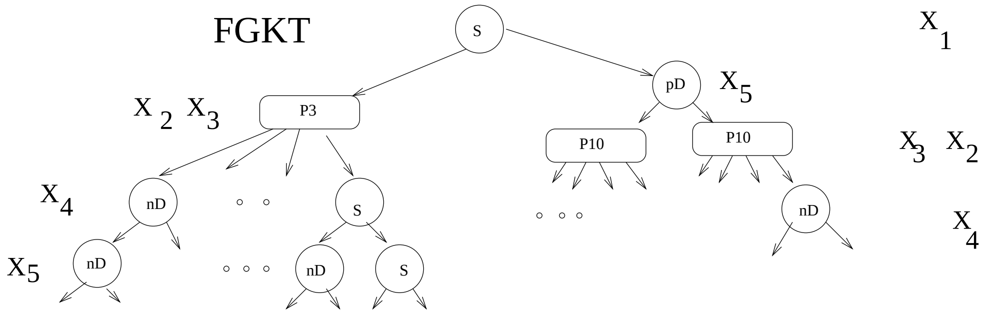

Figure 3: *A Scheme of a Free Generalized Kronecker Tree*

Type of Tree	Expression generated from the tree	Decision Diagram Generated from Tree
Shannon Tree	Minterm Expansion	Binary Decision Diagram (BDD)
Positive Davio Tree	Positive Polarity Reed-Muller (PPRM)	Functional Decision Diagram (FDD) [38]
Reed-Muller Tree	Fixed Polarity Reed-Muller (FPRM)	Reed-Muller Decision Diagram
Kronecker Tree	Kronecker Expansion (KRO)	Ordered Kronecker Functional Decision Diagram (OKFDD) [32, 7]
Pseudo Reed-Muller Tree	Pseudo Reed-Muller Expansion (PSDRM)	Pseudo Reed-Muller Decision Diagram
Pseudo Kronecker Tree	Pseudo Kronecker Expansion (PSDKRO)	Pseudo Kronecker Decision Diagram (PKDD) [34]
Free Kronecker Tree	Free Kronecker Expansion (FKE)	Free Kronecker Decision Diagram (FKFDD) [11]
Generalised Kronecker Tree (GKT)	Generalised Kronecker Expansion (GKE)	Generalised Kronecker Decision Diagram (GKDD)
Pseudo Generalised Kronecker Tree (PGKT)	Pseudo Generalised Kronecker Expansion (PGKE)	Pseudo Generalised Kronecker Decision Diagram (PGKDD)
Mixed Pseudo Generalised Kronecker Tree (MPGKT)	Mixed Pseudo Generalised Kronecker Expansion (MPGKE)	Mixed Pseudo Generalised Kronecker Decision Diagram (MPGKDD)
Ordered Generalised Kronecker Tree (OGKT)	Ordered Generalised Kronecker Expansion (OGKE)	Ordered Generalised Kronecker Decision Diagram (OGKDD)
Free Generalised Kronecker Tree (FGKT)	Free Generalised Kronecker Expansion (FGKE)	Free Generalised Kronecker Decision Diagram (FGKDD)

Table 1: *Relations of Canonical AND/EXOR Trees, Expressions and Decision Diagrams*

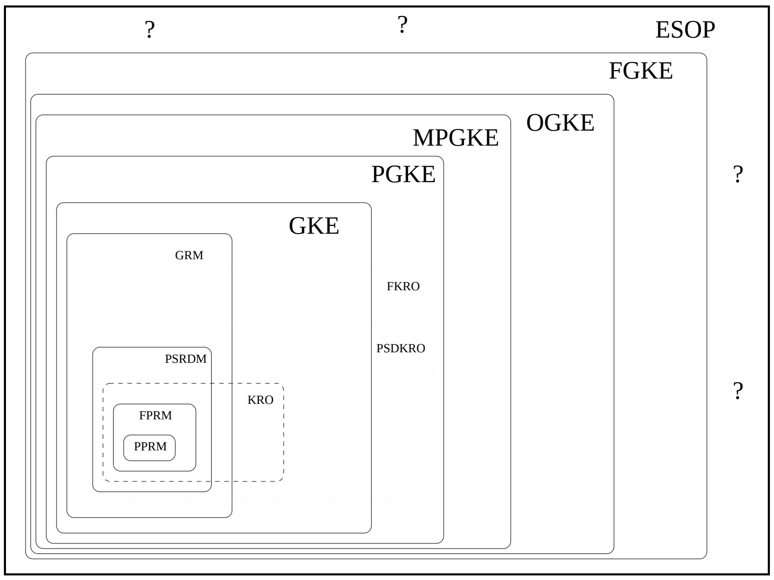

Figure 4: *Set-theoretical relationship among known and new (Zhegalkin) classes of AND/EXOR canonical forms*

Type of Tree	Lattice Diagram Generated from Tree and type of regular layout	For what functions
Shannon Tree	Universal Akers Array	nearly symmetric, all
Positive Davio Tree	Functional Lattice Diagram	nearly symmetric
Reed-Muller Tree	Reed-Muller Lattice Diagram	nearly symmetric
Kronecker Tree	Ordered Kronecker Lattice Diagram	all
Pseudo Reed-Muller Tree	Pseudo Reed-Muller Lattice Diagram	nearly symmetric
Pseudo Kronecker Tree	Pseudo Kronecker Lattice Diagram	all
Free Kronecker Tree	Free Kronecker Lattice Diagram	all
Generalised Kronecker Tree (GKT)	Generalised Kronecker Zhegalkin Lattice Diagram	all
Pseudo Generalised Kronecker Tree (PGKT)	Pseudo Generalised Kronecker Zhegalkin Lattice Diagram	all
Mixed Pseudo Generalised Kronecker Tree (MPGKT)	Mixed Pseudo Generalised Kronecker Zhegalkin Lattice Diagram	all
Ordered Generalised Kronecker Tree (OGKT)	Ordered Generalised Kronecker Zhegalkin Lattice Diagram	all
Free Generalised Kronecker Tree (FGKT)	Free Generalised Kronecker Zhegalkin Lattice Diagram	all
Boolean Ternary Decision Diagram	Boolean Ternary Zhegalkin Lattice Diagram	all

Table 2: *Relations of Canonical AND/EXOR Trees, and Lattice Diagrams*

and ESOP (and between PSDKRO and ESOP) in the Green/Sasao hierarchy of AND/EXOR canonical expansions.

In the new families, there are included their respective restricted families not shown in the diagram; for instance families with sets S_j limited to 2, 3, 4, ..., $n-1$ variables. These families are important from the practical point of view.

Similarly as in examples 2.2 and 2.3 one can create functions that prove that the proper inclusion relations illustrated in Table 1 and Figure 4 are true; for instance, that the MPGKTs are better than PGKTs, the MPGKE are better than PGKEs, and so on.

An interesting open question is the following: *"Is the FGKE family of expansions equal to the well-known class of ESOP circuits or is it properly included in ESOP, as the Figure 4 would suggest?"*

The guess of the authors is that it is very close to ESOP but not the same and perhaps only experimentation will answer this question. In any case, flattening of the optimal new diagrams will lead to better ESOP circuits than those obtained from flattening of known diagrams, which will lead to high quality ESOP minimization programs.

IV. ZHEGALKIN LATTICE DIAGRAMS AND REGULAR LATTICE LAYOUTS.

Akers introduced the so-called Universal Akers Arrays [1, 4] in 1972. They are regular lattices and look like BDDs for symmetric functions. The nodes are multiplexers (Shannon expansions). Each multiplexer obtains one data input from North and one from East, and directs its output to South and West. Diagonal lines are used for inputs (control variables of mulitplexers). It was proven that every binary function can be realized with such structure, but an exponential number of levels was necessary (which means, the control variables in diagonal buses were repeated very many times). (Sometimes the only way to implement a function is to repeat the same variable **subsequently**, without other variables interspersed. We will cal such subcategory of arrays the *variable interval arrays.* The arrays of Akers were universal and they were unnecessarily large, because they were calculated once for all for the worst case functions. No efficient procedures for finding order of (repeated) variables were given, and some functions were next shown for which this approach is very inefficient. Nevertheless, the idea of the Akers' Array is very captivating from the point of view of submicron technologies, because connections are short, delays are equal and predictable, late arriving variables can be given close to output, and, similarly as in PLAs, logic synthesis can be combined with layout, so that no special stage of placement and routing is necessary. He showed also many interesting properties of his both universal and non-universal arrays that can be utilized to develop new design methods and efficient computer algorithms.

Because of the progress of hardware and software technologies since 1972, our approach is quite different from that

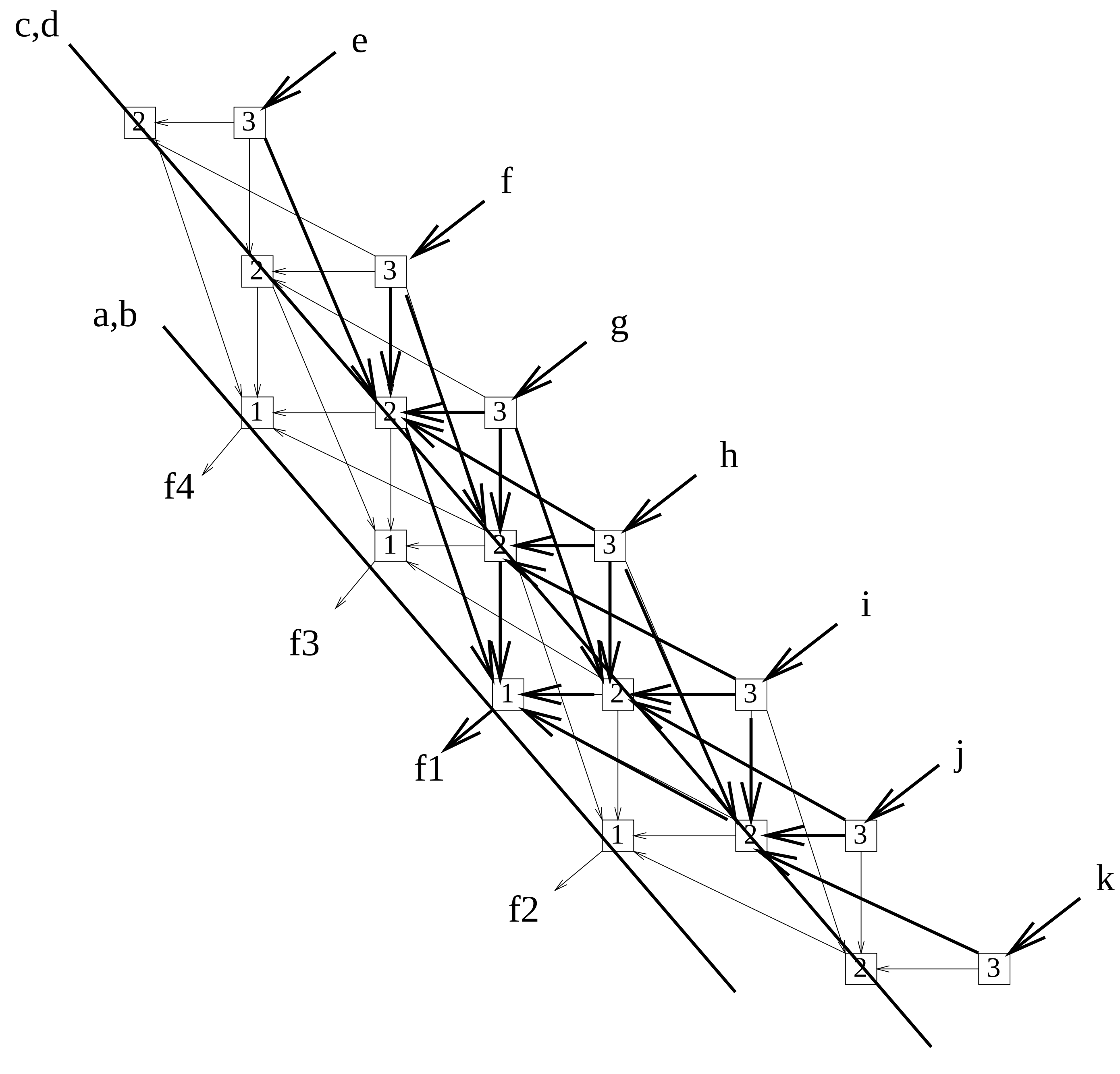

Figure 5: *An Example of Zhegalkin PGKT 4x4 Lattice that realizes a 4-output, 11-input function. Every small rectangle is a Universal GRM Expansion Module with corresponding two control variabls in diagonal buses, and four data inputs. Numbers in squares correspond to levels of the lattice. Predictability and equality of delays should be appreciated*

of Akers. We do not want to design a universal array for **all** functions, because this would be very inefficient for **nearly all** functions. Instead we create a **logic/layout functions' generator** that gives efficient results for **many** real-life functions, not only symmetrical ones. We argue that there is no need to realize other functions, since it was shown in [31] that, in contrast to the randomly generated "worst-case" functions, 98% of functions from real-life are decomposable. Therefore, the "other" functions are either decomposable to the easy realizable functions, or they do not exist in practice [31, 26].

Based on analysis of realizations of arithmetic, symmetric, unate and standard benchmark functions and new technologies [3], we have very substantially generalized the concept of Akers Array in the following ways:

1. Instead of assuming only a Shannon expansion, we use any subset of S,pD and nD expansions. We allow also all Linearly-Independent expansions [16, 17, 25, 5], the Boolean Ternary expansions from [20, 21], as well as all Zhegalkin expansions from [24] and this paper.

2. We consider all Kronecker, Pseudo-Kronecker, Mixed, and other Decision Diagram concepts that are used in Reed-Muller logic, and we generalize these concepts. In addition, we take diagram ideas from Boolean Ternary Decision Diagrams introduced in [20, 21], as well as the expansion types (such as mixed or free) from this paper.

3. We allow more powerful neighborhood geometries. Instead of having only two inputs and two outputs from every node (we call them 2x2 Lattices), we consider also regular lattice diagrams in which node has 3 inputs and 3 outputs, diagrams with 4 inputs and 4 outputs from a node, and diagrams with 8 inputs and 8 outputs from a node, The 3x3 Lattices are for 3-valued logic or for Boolean Ternary Decision Diagrams. The 4x4 Lattices (as in Figure 5) are for 4-valued logic, or pairs of binary control variables. They are for instance used for Lattice realizations of GKTs, PGKTs, etc. The 8x8 Lattices are for 8-valued logic, or triplets of binary control variables. Similarly we can create higher-neighborhood geometries, but we believe it is more practical to keep the number of neighbors small. In essence, the 2x2,3x3,4x4 and 8x8 neighborhoods are used in patents [3] and published works.

4. We allow to mix control variables in diagonal buses. This permits to realize Free diagrams.

Table 2 presents some of new Lattices. They are counterparts of known and new trees. The last column includes type of functions for which this array type is recommended. In theory, **every** Lattice from this table can implement **every** (multi-output) binary function (so, we could write *"all"* in all rows). In practice, however, some of the arrays, those that have "nearly symmetric" in the third column, are good only for nearly symmetric functions and can be done universal only by repeating the same variable in consecutive levels (making them the variable interval arrays). Taking the same variable in consecutive levels is a trick that practically changes the lattice to a tree in upper levels, so it is very area wasteful. Besides, changing a lattice to a tree can be done better using other introduced by us methods.

For symmetric or nearly symmetric functions, the lattices with "nearly symmetric" in the third column can be still better choices than other types of arrays. The universal Akers Array and variable interval arrays are superseeded by our new arrays, because we demonstrate how a better array can be created for each of them.

The choice of the appropriate array type for a given function remains a difficult problem to be solved and at the moment we dispose just examples and heuristics, but we do not know the best solutions. In theory, one would need just the most powerful array type, and assume that the design algorithm will select the best expansions and variable ordering in any case (the same as OKFDD versus BDD or FDD). But creating good a heuristic algorithm for the most general arrays is more difficult than to create such an algorithm for a restricted type of array. Perhaps, some simpler arrays are also better for layout, like assuming only Shannon Expansions (multiplexers) allows to design the function from mostly pass-transistors and very regular small-grid layout (also bidirectional pass-transistors in some technologies).

An example of Zhegalkin Lattice PGKT is shown in Figure 5. The input variables are: a and b variables that control various GRM expansions in level 1 (realized by Universal GRM Expansion Modules introduced in [24]), c and d variables control various GRM expansions in level 2, other input variables are e, f, g, h, i, j, k. Only one of the output functions, $f1$ has all its possible lattice connections and nodes in previous layers. Other output functions can be realized with only a subset of second and third level nodes. (The particular structure type from Figure 5, in which **all** variables are paired, does not allow to realize **arbitrary** functions without resorting to the variable interval array. Thus, this kind of array, with all variables paired and GRM used for them, although theoretically universal, does not give good results for strongly unsymmetrical functions. By adding single variable layers with S, pD and nD expansions the array's power is greatly improved. However, for some symmetric functions, the more restricted array may be better.

Calculation of data input functions to lattice nodes for **any** type of expansions and **any** lattice neighborhoods is performed by the same technique of logic solving equations for a given structure, as one used for Linearly Independent logic in [24, 25]). This technique is very general and can be adopted to many non-binary logics. However, in contrast to the LI logic, where the equations have always one solution resulting from non-singularity of the matrix M [24, 25, 19, 22], the structural equations, in general case, can have one, many, or no solutions. (Also, they are not only equations over Galois Field.) When there are many solutions, the one evaluated as best is taken. When there are no solutions, the backtrack to another structure, another expansions, or another blocks of input variables is executed. Selection of the order of (usually repeated) variables is done using the concept of the best separation of most different-value minterms, using the Repeated Variable Maps from [26]. The problems of variable ordering and variable partitioning, known for long in logic synthesis to be tough ones, here become even more important and at the same time more difficult, because the variables must be repeated. Hopefully, it was found that in contrast to the worst-case randomly generated functions, for real-life benchmark functions only few repetitions of variables are enough. It is especially easy to symmetricize weakly specified functions and Boolean relations.

V. CONCLUSIONS AND FUTURE WORK

The generalisation of the DDs can be done in several ways, which can be combined together to create various representations.

- D1. Replacing the single type of expansion realised in nodes of the Shannon tree, called the Shannon expansion, with other type of expansion,
- D2. Allowing for several types of expansions in every level of the expansion tree,
- D3. Allowing more freedom in ordering variables in branches of the tree (including the case of no ordering at all),
- D4. Generalizing Trees to Directed Acyclic Graphs by combining non-isomorphic tree nodes together, and thus requiring repetitions of variables (as in Zhegalkin Lattices), Real repetitions add much power to the diagrams, consecutive repetitions in levels (variable interval arrays), can be modified to other diagrams without repetitions.
- D5. Creating generalised expansions for sets of variables, instead for single variables.

Any combination of the above generalisation types of a binary tree can be next used to create canonical trees, flattened (two-level) expressions (expansions, forms) based on them, decision diagrams and corresponding regular layouts.

Several of these generalized types of decision diagram representations have already been introduced, investigated theoretically, and implemented in CAD tools, e.g. [6, 37, 39, 40], to mention only those used for Exor Logic. Many more, however, remain still to be analysed and evaluated experimentally. Many types that may result from the above dimensions of generalization have not even been defined yet.

If we consider the well-known Green/Sasao hierarchy of representations of EXOR-based Boolean functions the new representations will be not worse than the known ones in terms of complexity. Based on the previous results in this field (including the Free KRO diagrams from [11]), it is expected that all these representations will find important applications in logic synthesis for ESOP circuits, and fine grain FPGAs, as well as in representation of large functions. It would also be interesting to check if the new Zhegalkin forms will improve on the known canonical AND/EXOR forms for symmetric functions [8]. Definitely, Zhegalkin Lattices proved to be very good and superior to Universal Akers Arrays when realizing all totally symmetric functions, partially symmetric functions, or easily symmetrizable functions for which only few variables require repetitions in the structure. In general, the proposed methods should be combined with Ashenurst/Curtis decompositions, and the problem when to decompose and when to symmetricize remains in general open.

Observe, that all the generalizations proposed in this paper are oriented towards the binary logic synthesis, and therefore still use expansions for the binary logic. However, in line with K*BMDs, all our ideas above can be further extended to multiple-valued logic (where more general multivalued operations extending EXOR and AND operators would be used and the linearly independent logic methods applied [16, 22].) The diagrams and lattices introduced here can also be generalized to integer arithmetic (and also for rational arithmetic realized with continuous logic), where + and * arithmetic operators and more general linearly independent operations would be used. These *"word-level"* expansions can be derived by the careful reader based on papers [14, 16, 22, 9]. The word-level expansions together with the generalization types D1-D5 can be used to create trees, forms, diagrams, lattices, and layouts, along the same lines as presented in [24] and this paper [28]. Lattices have been also proposed for continuous logic [30, 29, 28] and mv logic [17, 28]. All these researches have the same applications to submicron technology and the same basic philosophy as explained in [40] (they can be used for custom logic/layout synthesis, and to develop new types of FPGAs [28, 29, 30].

References

[1] S.B. Akers, "A rectangular logic array," *Trans. IEEE*, Vol. C-21, pp. 848-857, August 1972.

[2] R.E. Bryant, "Graph-based algorithms for boolean function manipulation, *IEEE Transactions on Computers*, C-35, No. 8, pp. 667-691, 1986.

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

[4] M. Chrzanowska-Jeske, Z. Wang and Y. Xu, "A Regular Representation for Mapping to Fine-Grain, Locally-Connected FPGAs," *accepted to ISCAS'97.*

[5] K.M. Dill, K. Ganguly, R. Safranek, and M.A. Perkowski, "A New Linearly Independent, Zhegalkin Galois-Field Reed-Muller Logic," *submitted to Reed-Muller'97.*

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

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

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

[9] R. Drechsler, B. Becker, and S. Ruppertz, "K*BMDs: A New Data Structure for Verification," *Proc. of the European Design and Test Conf.,* pp. 2-8,1996.

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

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

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

[13] S. Panda, F. Somenzi, and B.F. Plessier, "Symmetry Detection and Dynamic Variable Ordering of Decision Diagrams," *Proc. of DAC,* pp. 628-631, 1994.

[14] M. Perkowski, P. Dysko, and 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.

[15] 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.

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

[17] M.A. Perkowski, and E. Pierzchala, "New Canonical Forms for Four-valued Logic", *Internal Report, Department of Electrical Engineering,* Portland State University, 1993.

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

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

[20] M. Perkowski, A. Sarabi, and I. Schaefer, "Multi-Level Logic Synthesis Based on Kronecker and Boolean Ternary Decision Diagrams for Incompletely Specified Functions," *Computer Aided Design and Test,* Dagstuhl-Seminar-Report, 105, 13 Febr - 17 Febr, 1995.

[21] M. Perkowski, M. Chrzanowska-Jeske, A. Sarabi, and I. Schaefer, "Multi-Level Logic Synthesis Based on Kronecker and Boolean Ternary Decision Diagrams for Incompletely Specified Functions," *VLSI Design*, Vol. 3, Nos. 3-4, pp. 301-313, 1995.

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

[23] 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.

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

[25] M. Perkowski, L. Jozwiak, R. Drechsler, and B. Falkowski, "Ordered and Shared, Linearly Independent, Variable-Pair Decision Diagrams for Incompletely Specified Functions," *submitted.*

[26] M. Perkowski, M. Marek-Sadowska, L. Jozwiak, T. Luba, S. Grygiel, M. Nowicka, R. Malvi, Z. Wang, and J. Zhang, "Decomposition of Multi-Valued Relations," *accepted to ISMVL'97*, Nova Scotia, May 1997.

[27] M.A. Perkowski, M. Chrzanowska-Jeske, and Y. Xu, "Lattice Diagrams for Reed-Muller Logic Realization in Regular Layouts with Predictable Timing," *submitted to Reed-Muller'97.*

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

[29] E. Pierzchala, and M.A. Perkowski, "High Speed Field Programmable Analog Array Architecture Design," *FPGA'94*, Berkeley, California, February 1994.

[30] E. Pierzchala, M.A. Perkowski, and S. Grygiel, "A Field Programmable Analog Array for Continuous, Fuzzy, and Multi-Valued Logic Applications," *Proc. 24-th ISMVL*, pp. 148-155, Boston, May 25-27, 1994.

[31] T. D. Ross, M.J. Noviskey, T.N. Taylor, D.A. Gadd, "Pattern Theory: An Engineering Paradigm for Algorithm Design," *Final Technical Report WL-TR-91-1060,* Wright Laboratories, USAF, WL/AART/WPAFB, OH 45433-6543, August 1991.

[32] A. Sarabi, P.F. Ho, K. Iravani, W.R. Daasch, and M. Perkowski, "Minimal Multi-level Realisation of Switching Functions based on Kronecker Functional Decision Diagrams," *Proc. IWLS '93, Intern. Workshop on Logic Synthesis.*

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

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

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

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

[37] I. Schaefer, and 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.

[38] U. Kebschull, E. Schubert, and W. Rosenstiel, "Multilevel Logic Synthesis based on Functional Decision Diagrams," *Proc. IEEE Euro-DAC*, pp. 43-47, 1992.

[39] N. Song, and 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.

[40] A. Sarabi, N. Song, M. Chrzanowska-Jeske, and 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.

[41] N. Song, M. A. Perkowski, M. Chrzanowska-Jeske, A. Sarabi, "A New Design Methodology for Two-Dimensional Logic Arrays," *VLSI Design,* special issue on Decomposition in VLSI Design, (L. Jozwiak ed)., Vol. 3, No. 3-4, 1995.

[42] I.L. Zhegalkin, "Arithmetization of symbolic logic (in Russian), *Mathematiceskij Sbornik*, Vol. 35, pp. 311-373, (1928), Vol. 36, pp. 205-338 (1929).