VLSI DESIGN
1995, Vol. 3, Nos. 3-4, pp. 301-313
Reprints available directly from the publisher
Photocopying permitted by license only

Multi-Level Logic Synthesis Based on Kronecker Decision Diagrams and Boolean Ternary Decision Diagrams for Incompletely Specified Functions[1]

MAREK A. PERKOWSKI, MALGORZATA CHRZANOWSKA-JESKE and ANDISHEH SARABI
Department of Electrical Engineering, Portland State University, Portland, OR 97207

INGO SCHÄFER
Arkos Design Inc., 5619 Scotts Valley Drive, Scotts Valley, CA 95066

This paper introduces several new families of decision diagrams for multi-output Boolean functions. The introduced families include several diagrams known from literature (BDDs, FDDs) as subsets. Due to this property, these diagrams can provide a more compact representation of functions than either of the two decision diagrams. Kronecker Decision Diagrams (KDDs) with negated edges are based on three orthogonal expansions (Shannon, Positive Davio, Negative Davio) and are created here for incompletely specified Boolean functions as well. An improved efficient algorithm for the construction of KDD is presented and applied in a mapping program to ATMEL 6000 fine-grain FPGAs. Four other new families of functional decision diagrams are also presented: Pseudo KDDs, Free KDDs, Boolean Ternary DDs, and Boolean Kronecker Ternary DDs. The last two families introduce nodes with three edges and require AND, OR and EXOR gates for circuit realization. There are two variants of each of the last two families: canonical and non-canonical. While the canonical diagrams can be used as efficient general-purpose Boolean function representations, the non-canonical variants are also applicable to incompletely specified functions and create don't cares in the process of the creation of the diagram. They lead to even more compact circuits in logic synthesis and technology mapping.

Key Words: *Decision Diagrams, Kronecker Decision Diagrams, EXOR logic, Incompletely Specified Functions, Fine Grain FPGAs*

1. INTRODUCTION

Recently, the AT 6000 series of FPGAs from Atmel (formerly Concurrent Logic, Inc. [9]) have been brought to market. The AT6000 FPGA series are fine-grain FPGAs with a cellular architecture (local connections) and very limited routing resources. There exist also other similar chips, fabricated or in development, by Algotronix [1] (now part of Xilinx), Toshiba, Plessey, Pilkington, Motorola [22], and National Semiconductor, in addition to other companies and Universities. However, high quality logic synthesis software tools for these families are still missing.

One possible approach is to adopt BDD-based methods. However, all these methods are based on the "unate paradigm" [7]. The "unate paradigm" is the assumption that most of the practical logic functions occurring in logic design are *unate or nearly unate.* The meaning of *unate* and *nearly unate* for logic minimization purposes is that the circuit realization of a *(nearly) unate* function with AND and OR gates is not greater in terms of the number of gates than a circuit using the AND and EXOR gates. On the other hand the meaning of *linear or nearly linear* for logic minimization purposes is that the circuit realization of a *(nearly) linear* function with AND and EXOR gates is not greater in terms of the number of gates than a circuit using the AND and OR gates (it is usually smaller). Although the unate paradigm holds for many control logic circuits, it is not valid for many data path circuits. Also the theoretical studies in EXOR logic

[1] The work presented in this paper was partially supported by NSF grant MIP-9110772.

by Sasao [30, 31] and others [27] confirm the usefulness of EXOR gates. Arithmetic functions like counters, adders, multipliers, signal processing functions and error correcting logic that belong to the class of *(nearly) linear* functions [16] cannot be efficiently minimized for circuit speed and area using BDDs only. To address these deficiencies, the concepts of: Adaptive logic trees [14] and Functional Decision Diagrams (FDDs) [33, 18] have been developed and applied to FPGA mapping. The adaptive logic trees were introduced for multi-level representation of switching functions based on Reed-Muller [10] canonical expansion. The Functional Decision Diagrams were introduced for completely specified functions as a generalization of adaptive logic tree structures into directed acyclic graph (DAG) structures.

Free, multiple-valued decision diagrams for incompletely specified functions were first introduced in [23]. These diagrams, called orthogonal, include binary Kronecker Functional Decision Diagrams and other diagrams presented here. The same paper also presented an efficient algorithm for the construction of diagrams, in which variable order and expansion type selections were based on self-learning. A comprehensive presentation of various tree, DAG and flattened structures for AND/EXOR networks was given in [23, 24, 25]. These papers introduced for the first time several new decision diagrams and synthesis algorithms, both for binary and multiple-valued logic.

Some of these concepts were next implemented in software tools: Permuted (free) Reed-Muller trees [36] use Positive and Negative Davio expansions and were applied for technology mapping to AT6000 for completely specified functions. Kronecker Functional Decision Diagrams (KFDDs) introduced in [29, 32, 12] were used for AT6000, Actel and Xilinx technology mapping for both completely and incompletely specified functions. Pseudo KDDs and Free KDDs were introduced in [15] and were used for ATMEL 6000 technology mapping for completely specified functions.

While the number of families of canonical and non-canonical diagrams introduced in the above papers is very large, and most of them have not been implemented yet, we continue here our more detailed study of some of these families. This paper has two main contributions: an efficient algorithm for generating KDDs, in sections 3 and 4, and the introduction of several efficient new families of decision diagrams, in sections 2, 6, and 7. Section 2 defines the Kronecker Decision Diagrams (KDDs), the Pseudo Kronecker Decision Diagrams (PKDDs), the Free Kronecker Decision Diagrams (FKDDs), and some other types of diagrams. The basic ideas of the algorithm to create KDDs for incompletely specified multi-output functions are presented in section 3. This algorithm uses the new concept of *constrained don't cares,* also explained in section 3. Section 4 discusses one more important aspect of this algorithm: redundancy conditions for the selection of expansion variables. Analysis of benchmark results is given in section 5.

Compared to other algorithms for the generation of decision diagrams (DD), our algorithm proves that particularly compact KDD diagrams can be obtained for **incompletely specified functions**. Encouraged by this result, which points to the usefulness of don't cares and constrained don't cares, we develop in section 6 new families of decision diagrams which take additional advantage of these don't cares. We create families of non-canonical variants that are applicable to both incompletely specified functions and completely specified functions. The algorithms to construct such diagrams create don't cares and constrained don't cares in the process of creating the diagrams. The first class of diagrams are called Boolean Ternary DDs (BTDDs). The word "Boolean" is to denote that they are for Boolean logic, the word "Ternary" denotes that there are three outgoing edges from each node, instead of the usual two edges. The second class are called Boolean Kronecker Ternary DDs (BKTDDs). Those are diagrams which have two kinds of nodes: binary nodes as in KDDs, and ternary nodes as in BTDDs. BKTDDs are better for efficient logic optimization and technology mapping by creating diagrams with even less numbers of nodes. In section 7, we introduce certain diagrams which are canonical counterparts of BTDDs and BKTDDs and are used for efficient general-purpose representation and manipulation of switching functions.

Nodes in KDD diagrams correspond to two-input multiplexers and AND/EXOR gates. Nodes in BTDDs are combinations of multiplexers and AND/EXOR gates with two-input OR, EXOR and inhibition ($A \cdot \bar{B}$) gates. Therefore, all these nodes can be readily realized with cells of the AT6000 series, Motorola's MPA10XX, or other fine-grain FPGAs. This makes the new diagrams a useful tool for fine-grain FPGA synthesis.

2. DEFINITIONS OF VARIOUS TYPES OF KRONECKER FUNCTIONAL DECISION DIAGRAMS

In this section, the representation of switching functions based on various Kronecker Functional Decision Diagrams will be presented. The definitions of decision diagrams are essentially due to Bryant [8] and will be reviewed in the following. All decision diagrams will be

introduced systematically. Because many different names are used in papers, we will refer to the names assigned by the original creators of the diagrams, or to names that are already used by more than one author. Some names used here are the result of discussions at the First International Workshop of Applications of the Reed-Muller Expansion in Circuit Design, Hamburg, September 1993 [27].

Definition 1: A **decision diagram** is a rooted, directed acyclic graph with vertex set V containing two types of vertices. A **terminal** vertex v has as attribute a value $value(v) \in \{0, 1\}$. A **non-terminal** vertex v has as attributes an argument index $index(v) \in \{1, \ldots, n\}$ and two successors $low(v)$, $high(v) \in V$.

Definition 2: A **Binary Decision Diagram (BDD)** is a decision diagram having root vertex v denoting a function f, defined recursively as:

1. If v is a terminal vertex:
   (a) If $value(v) = 1$, then $f_v = 1$.
   (b) If $value(v) = 0$, then $f_v = 0$.
2. If v is a non-terminal vertex with $index(v) = i$, then $f_v(x_1,\ldots, x_n)$ is the function:
   $\bar{x}_i \cdot f_{low(v)}(x_1,\ldots, x_n) \vee x_i \cdot f_{high(v)}(x_1,\ldots, x_n)$.

where the **cofactors** are defined as

$f_{low(v)}(x_1,\ldots, x_n) = (x_1,\ldots, x_{i-1}, 0,\ldots, x_n)$ and $f_{high(v)}(x_1, \ldots, x_n) = (x_1,\ldots, x_{i-1}, 1,\ldots, 1\ x_n)$.

Definition 3: An **Adaptive Logic Tree (ALT)** is a decision diagram having root vertex v denoting a function f_v denoted recursively as:

1. If v is a terminal vertex:
   (a) If $value(v) = 1$, then $f_v = 1$.
   (b) If $value(v) = 0$, then $f_v = 0$.
2. If v is a non-terminal vertex with $index(v) = i$, then $f_v(x_1,\ldots, x_n)$ is the function:
   $f_v(x_1,\ldots, x_n) = f_{low(v)}(x_1,\ldots, x_n) \oplus x_i \cdot [f_{high(v)}(x_1, \ldots, x_n) \oplus f_{low(v)}(x_1,\ldots, x_n)]$.

where $\oplus$ is the Exclusive-OR operator.

ALT is also known as Permuted Reed-Muller Tree [36].

Definition 4: A **Kronecker Functional Decision Diagram (KFDD)** is a decision diagram having root vertex v denoting a function f_v denoted recursively as:

1. If v is a terminal vertex:
   (a) If $value(v) = 1$, then $f_v = 1$.
   (b) If $value(v) = 0$, then $f_v = 0$.
2. If v is a non-terminal vertex with $index(v) = i$, then f_v is one and only one of the functions:
   (a) $f_v(x_1,\ldots,x_n) = f_{low(v)}(x_1,\ldots,x_n) \oplus x_i \cdot [f_{high(v)}(x_1,\ldots,x_n) \oplus f_{low(v)}(x_1,\ldots,x_n)]$.
   (b) $f_v(x_1,\ldots,x_n) = f_{high(v)}(x_1,\ldots,x_n) \oplus x_i \cdot [f_{high(v)}(x_1,\ldots,x_n) \oplus f_{low(v)}(x_1,\ldots,x_n)] \cdot$
   (c) $f_v(x_1,\ldots,x_n) = x_i \cdot f_{low(v)}(x_1,\ldots,x_n) \oplus x_i \cdot f_{high(v)}(x_1, \ldots, x_n)$.

It is possible to set restrictions on the decision diagrams based on the order and the number of times the variables are encountered. The first of these classifies the DDs according to the number of times the variables are encountered.

Definition 5: A **free decision diagram** is a decision diagram such that every variable occurs at most once in any path from the root to the terminal vertices. A **repeated decision diagram** is one in which a variable x_i occurs more than once in a path from the root to the terminal vertices.

Letter F will stand for free in various DDs.

A different restriction is based on the order in which the variables are encountered.

Definition 6: An **ordered decision diagram** is a decision diagram such that for any non-terminal vertex v, if $low(v)$ is also non-terminal, then $index(v) < index(low(v))$. Similarly, if $high(v)$ is non-terminal, then $index(v) < index(high(v))$.

Definition 7: An ordered BDD is called an **Ordered Binary Decision Diagram (OBDD).** An ordered KFDD is called an **Ordered Kronecker Functional Decision Diagram (OKFDD).**

An ordered PRMT is called a *Reed-Muller Tree (RMT)* [36]. PRMT with Negative Davio vertices was also described in [36].

Furthermore, an ordered decision diagram can be reduced. This reduction, as given by Bryant, is obtained by removing isomorphic subgraphs and redundant vertices.

Definition 8: An ordered decision diagram is reduced if it contains no vertex v with $low(v) = high(v)$, nor does it contain distinct vertices v and v' such that the subgraphs rooted by v and v' are isomorphic.

Definition 9: A reduced OBDD is called a **Reduced Ordered Binary Decision Diagram (ROBDD).** A reduced RMT is called a **Functional Decision Diagram (FDD).** A reduced OKFDD is called a **Reduced Ordered Kronecker Functional Decision Diagram (ROKFDD).**

Further restrictions are still possible for decision diagrams based on the type of decompositions.

Definition 10: A decision diagram is **homogeneous (HDD)** if for every variable x_i in the diagram, there exists only one type of relation for f_v; described by one of the forms 2a, 2b, 2c.

Letter H will stand for *Homogeneous* in names of DDs.

Similar to the concept of Shared Binary Decision Diagrams [5, 6]. Shared *KFDDs (SKFDD)* can be defined for multi-output switching functions.

Definition 11: A **Kronecker Decision Diagram (KDD)** is a Shared Reduced Ordered Homogeneous Non-repeated Kronecker Decision Diagram (SROHNKFDD).

This decision diagram has been also called *SROKFDD, Shared Reduced Ordered Kronecker Decision Diagram* [32].

Having thus defined syntactically several families of decision diagrams, we will now relate them to Boolean functions. It is known [10, 25] that there exist three single-variable decompositions over Galois Field (2):

Let x_i be an input variable of function f.

(1) $f = x_i f_{x_i} \oplus \bar{x}_i f_{\bar{x}_i}$ Shannon
(2) $f = f_{\bar{x}_i} \oplus x_i[f_{x_i} \oplus f_{\bar{x}_i}] = f_{\bar{x}_i} \oplus x_i g_i$ Positive Davio
(3) $f = f_{x_i} \oplus \bar{x}_i [f_{x_i} \oplus f_{\bar{x}_i}] = f_{x_i} \oplus \bar{x}_i g_i$ Negative Davio

where f_{x_i} and $f_{\bar{x}_i}$ are the positive and negative cofactors.

One important property of these three expansions is that the functions f_{x_i} and $f_{\bar{x}_i}$, obtained by applying any of the three expansions for x_i, will be independent of the variable x_i. The circuit realization of Equation (1) is given by a multiplexer gate (Fig. 1c), while Equations (2) and (3) describe an AND-EXOR gate structure, shown in Figure 1a, b.

The repeated applications of the Shannon expansion, Equation (1), to all variables of a function leads to the construction of a BDD. The application of the positive Davio expansion to each variable generates an adaptive logic tree [18]. The FDD is created from an order free adaptive logic tree by using reduction operations. (The concept of FDD has also been expanded to Negative Davio nodes). If all three expansions are applied to all variables, the Kronecker Reed-Muller tree [10, 24, 25] is obtained, and next reduced with standard DD reductions to a KDD.

There are two interesting binary families of KFDDs that include KDDs as a special case: Pseudo-KDDs, and Free KDDs.

Definition 12: A **Pseudo-Kronecker Decision Diagram (PKDD)** is a shared reduced ordered non-repeated Kronecker Functional Decision Diagram.

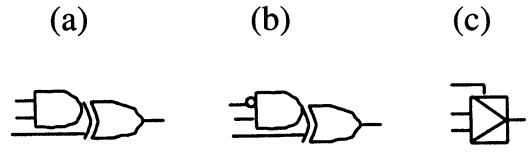

FIGURE 1 Circuit realization of Positive Davio, Negative Davio, and Shannon Expansions.

This decision diagram is obtained by recursively applying expansions (1), (2) and (3), and preserving the same order of variables in all branches of the diagram. In addition, for any variable (in an ordered DD a variable corresponds to a level of the diagram) any combination of expansion types can be applied to the nodes on this level (this will be defined as a non-homogeneous DD).

Definition 13: A **Free Kronecker Decision Diagram (FKDD)** is a SRNKFDD.

This decision diagram is obtained by recursively applying expansions (1), (2) and (3) and allowing no fixed order of expansion variables in the diagram (allowing free order of variables in the branches). For any node of the tree, any expansion variable and expansion type can be applied, so the orders of variables can be different in various branches. (The FKDD is thus neither homogeneous nor ordered).

As we see, the FKDD is the most general kind of the above-introduced KFDDs. It includes all the FDDs [18, 33], the OKFDDs from [12, 32], and the Permuted RMTs from [36] as its special cases. The PKDDs and FKDDs are reduced. Similar to BDDs with negated edges [21, 5], one can define the **PKDDs with negated edges** and the **KFDDs with negated edges.** (In DDs with negated edges, additional inverters may exist on inputs and outputs of the DD nodes).

The concept of a **free diagram** can be applied to a decision diagram with any type of expansion nodes. Similarly, the concept of a Non-repeated DD can be applied to any type of a diagram. In free diagrams, every variable occurs at most once along a branch. In **repeated-variable decision diagrams,** a variable can occur more than once along a branch. Repeated variables are necessary when new types of DDs are created by combining non-isomorphic subgraphs of the initial decision diagram.[1]

Such concepts of free diagrams and repeated-variable decision diagrams are applicable to all kinds of diagrams, hence also to BDDs. However, the concept of Kronecker Functional Decision Diagrams is a different kind of generalization, since it involves three expansion types. The non-homogeneous type of diagrams can be thus defined only for diagrams that include more than one type of expansion. Non-homogeneous KDDs allow for mixing Shannon, Positive Davio and Negative Davio expansion on a same level. Similarly other non-homogeneous types can be defined that mix Shannon with Positive Davio, Shannon with Negative Davio, and

[1] In certain regular architectures, it is possible to combine non-isomorphic nodes. For integrity of the function, however, it would be necessary to re-introduce some of the used variables again.

Positive Davio with Negative Davio on a same level [24, 25, 31].

The three types of generalization:

- free order,
- repetition of variables,
- non-homogeneity,

can be used for any of the decision diagrams introduced in sections 6, and 7 since each of these decision diagrams has more than one expansion type.

From now on, it will be assumed that all the decision diagrams are reduced, shared, free and can have negated edges.

3. BASIC ALGORITHM FOR KDD GENERATION

The crucial part of the KDD synthesis is the determination of the expansion variable ordering and the expansion type selection. In our algorithm for the KDD generation, the expansion variable order selection and the expansion type selection are performed concurrently. However, in the variable selection process, the same expansion variable is tested with each of the three expansions. Therefore, we first discuss the expansion variable selection.

There have been already formidable efforts to determine a good variable ordering [5, 20, 13] for BDDs. In contrast to these methods, we investigated the synthesis methods developed for the multiplexer circuits [34, 2, 3, 19, 35]. Inspired by the multiplexer synthesis algorithms developed by Almaini and Lloyd [2, 19] we adopted a breadth-first, top-down algorithm for the KDD generation.

The basic algorithm to obtain a good variable ordering is based on finding the expansion variable and the selection of the expansion type of the current level nodes such that the number of necessary next level nodes is minimal. To limit the search space for obtaining a good variable and expansion combination, we restrict the selection to being the same for a level. Thus, the three functions f_{x_j}, $f_{\bar{x}_j}$, and g are computed for each of the output functions $f[k]$ of the current level and from this, the expansion that leads to the least number of next level modules for variable x_j can be easily determined. Finally, the variable x_i and its corresponding expansion having the least number of next level modules is selected.

In order to determine the minimal number of the next-level nodes, it is necessary to find the redundant nodes on the next level. A node is redundant if it is trivial (it is either 0, 1, or equal to x_i) or if it can be merged with the existing node on this level (this node can belong to any output function $f[k]$). Therefore, Section 4 investigates the conditions for a next level node to be redundant.

When calculating Davio expansions, the function $g = f_{x_i} \oplus f_{\bar{x}_i}$ is calculated. Let us observe that when function f has don't cares, the operations

$$f_{x_i} \oplus f_{\bar{x}_i} = - \oplus 1,$$
$$f_{x_i} \oplus f_{\bar{x}_i} = 1 \oplus -,$$
$$f_{x_i} \oplus f_{\bar{x}_i} = - \oplus 0,$$

and

$$f_{x_i} \oplus f_{\bar{x}_i} = 0 \oplus -$$

have to be calculated. At the same time, the value of the respective parent function, f_{x_i} or $f_{\bar{x}_i}$ will be used for the other branch of the Davio expansion selected.

While the values of $f_{x_i} \oplus f_{\bar{x}_i}$ for all other combinations of 0, 1, and $-$ arguments are determined independently of the assignment of logic values 0, 1 to don't cares in f_{x_i} and $f_{\bar{x}_i}$, the value of $f_{x_i} \oplus f_{\bar{x}_i}$ for each of the above four cases depends on the choice of the possible initial value of a don't care in f_{x_i} or $f_{\bar{x}_i}$. For instance, in the case of the Positive Davio expansion, when values $f_{\bar{x}_i}$ and $f_{x_i} \oplus f_{\bar{x}_i}$ are concurrently calculated; assuming $f_{\bar{x}_i} = -$, and $f_{x_i} = 1$, the following cases are possible:

$$f_{\bar{x}_i} = 0, \text{ then } f_{x_i} \oplus f_{\bar{x}_i} = 1,$$
$$f_{\bar{x}_i} = 1, \text{ then } f_{x_i} \oplus f_{\bar{x}_i} = 0,$$

Denoting $f_{\bar{x}_i}$ by U, the value of $f_{x_i} \oplus f_{\bar{x}_i}$ will become $\overline{U}$ (see Fig. 2a).

The values of $f_{\bar{x}_i}$ and $f_{x_i} \oplus f_{\bar{x}_i}$ are, therefore, mutually constrained. This fact must be used in any synthesis program that attempts at optimal results for incompletely specified functions using Davio expansions.

Similarly, when $f_{\bar{x}_i}$ and $f_{x_i} \oplus f_{\bar{x}_i}$ are calculated for the Positive Davio Expansion; $f_{\bar{x}_i} = -$ and $f_{x_i} = 0$, the following cases are possible:

$$\text{if } f_{\bar{x}_i} = 0, \text{ then } f_{x_i} \oplus f_{\bar{x}_i} = 0 \text{ and}$$
$$\text{if } f_{\bar{x}_i} = 1, \text{ then } f_{x_i} \oplus f_{\bar{x}_i} = 1.$$

Denoting $f_{\bar{x}_i}$ by U, the value of $f_{x_i} \oplus F_{\bar{x}_i}$ becomes U.

This is also illustrated in Fig. 2a. Value U serves thus as a "constrained don't care". Every symbol U can take any arbitrary value in $f_{\bar{x}_i}$. But, when, as a result of logic minimization, for certain cube, C (minterm C) in $f_{\bar{x}_i}$, a value of zero or one is assigned to U, this cube in function $f_{x_i} \oplus f_{\bar{x}_i}$ must take the respective constrained value (U or $\bar{U}$), as in one of the above two cases. This value is next used in the minimization of $f_{x_i} \oplus f_{\bar{x}_i}$. The order of minimizing the functions f_{x_i}, $f_{\bar{x}_i}$, and $f_{x_i} \oplus f_{\bar{x}_i}$ is arbitrary, but the constraint U's must be always propagated amongst the functions.

Example: Let the function f be as shown in Fig. 2b. Its cofactor $f_{\bar{a}}$ is shown in Fig. 2c. The method to calculate

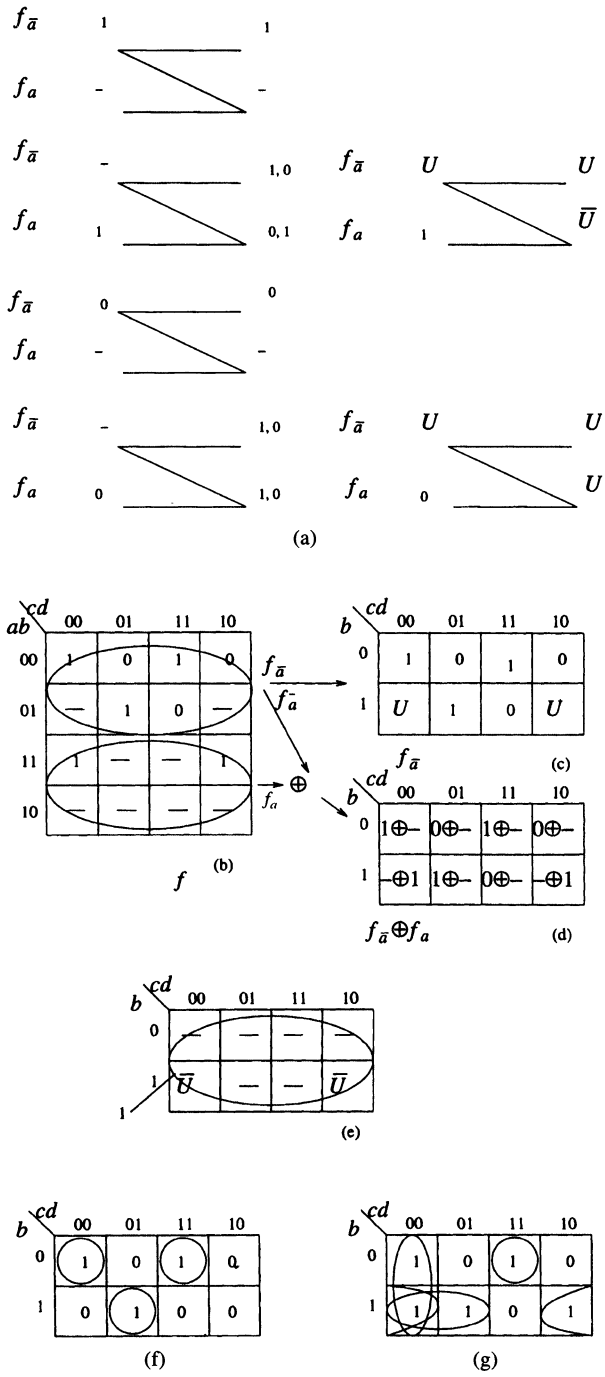

FIGURE 2 The constrained don't cares.

$f_{\bar{a}} \oplus f_a$ is illustrated in Fig. 2d. Applying rules in Fig. 2a to the map in Fig. 2d produces the equivalent map shown in Fig. 2e. Value U is treated in all minterms as a don't care, and the function in Fig. 2e is minimized to constant value 1. This requires treating $\bar{U}$'s for both minterms as 1, so $U = 0$ in these minterms. These values of $U = 0$ are now transmitted to the map for $f_{\bar{a}}$ of Fig. 2c which leads to an equivalent map shown in Fig. 2f. By minimizing the map of Fig. 2f, one gets $f = a \cdot 1 \oplus (\bar{b}\bar{c}\bar{d} + b\bar{c}d + \bar{b}cd) = a \oplus (\bar{b}\bar{c}\bar{d} + b\bar{c}d + \bar{b}cd)$.

In another variant, the function from Fig. 2e is minimized to 0, so symbols $\bar{U}$ for both minterms are assigned values of 0. Respectively, U values in both "constrained don't care" minterms in function from Fig. 2c get a value of 1. This leads to a new map for f_a from Fig. 2g. Thus $f = a \cdot 0 \oplus (\bar{c}\bar{d} + \bar{b}cd + b\bar{c} + b\bar{d}) = \bar{c}\bar{d} + \bar{b}cd + b\bar{c} + b\bar{d}$. This variant is selected as the better choice, and its corresponding specifications of constrained don't cares to values of 0 and 1 are preserved in functions f_{x_i}, $f_{\bar{x}_i}$ and $f_{x_i} \oplus f_{\bar{x}_i}$. Other don't care cubes remain as standard don't cares to be used at lower levels.

4. REDUNDANCY CONDITIONS FOR THE SPLITTING VARIABLE SELECTION

A sub-function at a node will be referred to as its "input function". Every KDD node has two such functions. Every BTDD node has three such functions. By # we denote the sharp operation [11]. We assume that the initial function is represented by the ON (true cubes) and DC (don't care cubes) sets. While creating a KDD from a root vertex, in some cases it is not necessary to realize the input functions (f_x, $f_{\bar{x}}$ or g) of a node with extra nodes. Whether it is necessary or not, depends on the selected expansion type and the selected expansion variable. For brevity, f_i and f_j stand for any of the three functions f_x, $f_{\bar{x}}$, and g. We will also define: $f'_{\bar{x}} = f \cdot x$, $f'_{\bar{x}} = f \cdot \bar{x}$, $g' = (f \cdot x) \oplus (f \cdot \bar{x})$. Analogously, f'_i and f'_j stand for any of the three functions f'_x, $f'_{\bar{x}}$, and g'. This section investigates how to specify the don't care part of the incompletely specified Boolean function in order to select the expansion variable to obtain the minimal KDD.

There exist three basic conditions for which a next-level node is redundant.

Condition 1: An input function f_i is a trivial function:

$$f_i = 0 \tag{7}$$

$$f_i = 1 \tag{8}$$

$$f_i = x_j \tag{9}$$

$$f_i = \bar{x}_j \tag{10}$$

Condition 2: Two input functions, f_i and f_j, at the same level of the KDD are identical:

$$f_i = f_j \quad i \neq j \tag{11}$$

Condition 3: Two input function f_i and f_j at the same level of the KDD are the complements of each other.

$$f_i = \bar{f}_j \quad i \neq j \tag{12}$$

The above conditions can be verified by the following cube calculus operations.

Verification of Condition 1: The case that an incompletely specified input function, f_i, can be specified such that $f_i = 0$ can be checked by:

$$f'_i = f_{dc} \tag{13}$$

where f_{dc} is a function that consists of only don't care terms.

The case that an input function can be specified such that $f_i = 1$ can be verified by:

$$ON(f_j) \cup DC(f_j) = 1 \tag{14}$$

To improve the efficiency of the program, all verifications of conditions are based on the formula: $f'_{x_i} = (f \cdot x_1) \mid_{x_{i=1}} = f \mid_{x_{i=1}}$, and a fast operation of intersection is calculated for all the attempts (most of which fail). A slower operation of substitution is calculated much more rarely, only when the expansion type and its variable have been finally selected.

Thus for verification of (14) we use the formula

$$f'_j = x_i \tag{14a}$$

where x_i is the corresponding expansion variable.

An incompletely specified function, f, can be specified to be dependent on only one variable, x_j, when Equations (15) and (16) are both satisfied:

$$f'_{x_j} = x_j \tag{15}$$

and

$$f'_{\bar{x}_j} = f_{dc} \tag{16}$$

Here f_{dc} consists only of the "don't-care" terms. A similar formula is used for the complement, $\bar{x}_j$.

Verification of Condition 2: For incompletely specified Boolean functions, it needs to be investigated whether the don't care parts of the functions f'_i and f'_j can be specified in such a way that $f'_i = f'_j$. This can be verified by the complement $\bar{f}'_c$ of the intersection of the two incompletely specified functions:

$$f'_c = f'_i \cdot f'_j \tag{17}$$

having no common cubes in the ON part of function f'_i.

$$f'_i \cdot \bar{f}'_c = f'_i \# f'_c = f_{dc_1} \tag{18}$$

and having no common cubes in the ON part of function f'_j.

$$f'_j \cdot \bar{f}'_c = f'_j \# f'_c = f_{dc_2} \tag{19}$$

where f_{dc_1} and f_{dc_2} are certain functions of only don't care cubes and # stands for the sharp operation. If the Equations (18) and (19) are true, the don't care parts of f'_i and f'_j can be specified in such a way that $f'_i = f'_j$.

Verification of Condition 3: For the assignment of the don't care part of the incompletely specified Boolean function such that the two input functions are complements of one another, Equations (20) and (21) have to be true.

$$f'_i \cdot f'_j = f_{dc} \tag{20}$$

to verify that the ON parts of both functions have no common cubes, and

$$f'_{i \to 1} + f'_{j \to 1} = x_i \tag{21}$$

in order to verify that the sum of both functions is dependent only on the chosen expansion variable. Here $\to 1$ means the assignment of the don't care cubes of the function to true cubes. If both formulas are satisfied, the don't care part of the input functions f'_i and f'_j can be chosen in such a way that $f'_i = \bar{f}'_j$.

5. EVALUATION OF BENCHMARK RESULTS

The results obtained by our implementation of the algorithm to obtain a KDD for several MCNC benchmark functions are given in Table 1. The column ASYL gives the number of nodes in the ASYL SROBDD, [5]. The columns $d1$, $d2$, and $d3$ give the number of nodes in the KDD if only Shannon, Positive Davio, or Negative Davio expansions are applied, respectively. Thus, column $d1$ gives the number of nodes in a SROBDD with negated edges and the columns $d2$ and $d3$ give two special cases of the FDDs [25, 33, 36]. The sub-column *neg* in each of the columns $d1$, $d2$, $d3$ gives the respective number of negated edges. The results in the column *mix* are obtained by selecting the expansion at each level that leads to the least number of next level nodes.

It can be seen that out of the functions that were also minimized in [5], our program gave a better solution in 8 cases, with ASYL giving a better solution in 7 and one function being of the same cost. As it can be seen, in one case, *apex7*, the solution is 30 percent better than in ASYL. This kind of improvement may be very important for large functions.

Since ASYL uses SROBDDs and our program uses KDDs that are more general; the only reason that our program gives sometimes worse result is the non-optimal selection of variables and expansions for them.

Table 2 gives the corresponding timing results. The time in seconds was obtained on a Sun Sparc 10/40 with memory limit set to 16 Mbytes. The results for ASYL (in

TABLE 1
Benchmark Results for the SROKDD

example	d1		d2		d3		mix		ASYL
		neg		neg		neg		neg	
5xpl	40	11	46	3	36	10	28	8	42
9sym	23	4	26	0	26	0	23	4	25
Z5xpl	45	15	46	3	38	11	28	13	
Z9sym	23	4	26	0	26	0	23		
alu2	184	19	143	7	146	9	163	13	159
apex7	234	9	303	7	253	9	191	7	301
b12	65	5	74	3	69	6	68	6	
b9	140	18	199	10	163	7	126	20	70
bw	100	15	114	7	107	9	93	11	95
clip	76	15	103	7	99	11	78	9	75
cm82a	10	3	9	0	8	0	8	3	
con1	15	1	14	1	19	2	14	1	
duke2	430	8	752	46	625	26	387	9	377
e64	1296	0	1601	0	1656	0	1296	0	
ex5	254	26	416	11	393	12	254	37	
f51m	36	16	36	1	36	6	25	6	39
frg1	119	5	255	9	218	11	120	4	105
inc	71	11	88	9	87	6	71	11	
majority	6	0	7	0	6	1	6	0	
misex1	35	4	43	3	39	1	39	2	36
misex2	104	3	153	0	156	1	98	2	88
rd53	15	5	15	0	12	2	12	3	17
rd73	29	9	23	0	21	1	20	4	31
rd84	40	13	34	0	28	4	28	4	42
sao2	79	11	101	3	120	3	79	13	81
squar5	35	3	26	2	35	3	24	3	
t481	20	3	20	2	14	1	17	4	
vg2	96	2	144	0	182	2	95	1	94
xor5	4	3	4	0	4	0	4	3	

seconds) were obtained on a Sun 4/65 with 25 Mbytes [5].

6. NON-CANONICAL BOOLEAN TERNARY DECISION DIAGRAMS AND BOOLEAN KRONECKER TERNARY DECISION DIAGRAMS

Through the introduction of additional expansions, it is possible to generate more general types of decision diagrams for both completely and incompletely specified Boolean functions. These decision diagrams will be referred to as Boolean Ternary Decision Diagrams and they are non-canonical. In order to construct a BTDD, first a variable x_i is selected. Then one of the possible three expansions of OR type, INHIBITION type, or EXOR type are applied to this variable. Each of these expansions is realized with a corresponding gate with two inputs f_{ind} and f_{dep}. f_{ind} denotes the part of the function which is not dependent on variable x_i, and f_{dep} denotes the part of the function which is dependent on variable x_i. While the OR and EXOR type expansions are realized by an OR or EXOR of f_{ind} and f_{dep}, respectively, the INHIBITION type expansion is realized as an INHIBITION of the form $f_{ind} \cdot \overline{f_{dep}}$. These expansions are called *dependency expansions* since they separate the output function f into functions f_{ind} that does not depend on variable x_i, and f_{dep} that depends on variable x_i.

Next, any of the three expansions used in KDDs; Shannon, Positive Davio, and Negative Davio, is applied to f_{dep}. This adds up to 3 * 3 = 9 different possible combined expansions. The 9 possibilities come from one out of three dependency expansions and one out of three KDD-type expansions. A BTDD node corresponding to this equation has three input functions, f_{ind}, f_{low}, and f_{high}. All nine types of nodes, represented by a circuit with one expansion variable and three input functions, are presented in Fig. 3.

The main idea here is that in all previous decision diagrams, the f_{ind} part of the function is carried through all levels. Here, it is proposed that this part of the function be decomposed separately.

TABLE 2
Timing Results for the SROKDD

example	in	out	d1	d2	d3	mix	ASYL
5xpl	7	10	2.3	3.6	3.2	4.7	3.3
9sym	9	1	1.8	4.4	4.3	6.5	13.0
Z5xpl	7	10	2.7	4.1	3.9	5.1	
Z9sym	9	1	1.6	4.1	4.2	5.7	
alu2	14	8	78.2	53.3	59.1	173.4	18.6
apex7	48	37	282.5	375.6	484.2	588.3	1586.0
b12	15	9	4.8	6.9	4.2	11.2	
b9	49	29	21.8	51.0	63.9	65.3	15.3
bw	5	28	17.7	41.5	20.2	63.9	5.7
clip	9	5	18.2	37.5	32.4	58.6	22.0
cm82a	5	3	0.3	0.4	0.4	0.4	
con1	7	2	0.3	0.5	0.7	0.5	
duke2	22	29	134.7	439.8	164.2	225.2	189.9
e64	65	65	131.5	182.2	55.0	140.2	
ex5	8	63	169.1	364.2	376.2	407.6	
f51m	8	8	2.0	2.6	3.2	3.4	4.2
frg1	28	3	12.5	62.6	63.1	39.7	390.5
inc	7	9	3.4	12.3	7.1	7.8	
majority	5	1	0.1	0.2	0.2	0.2	
misex1	8	3	1.0	4.0	1.8	6.0	1.6
misex2	25	18	7.4	5.4	4.5	14.3	47.4
rd53	5	3	0.3	0.5	0.6	0.6	1.2
rd73	7	3	1.4	2.1	2.2	3.0	8.0
rd84	8	4	4.2	4.7	5.3	7.4	24.0
sao2	10	4	9.5	20.9	53.6	36.5	13.8
squar5	5	8	0.8	1.0	1.5	1.4	
t481	16	1	23.6	36.7	30.2	54.1	
vg2	25	8	25.4	116.5	115.0	75.5	160.0
xor5	9	1	0.1	0.2	0.2	0.2	

The non-canonical OR type expansion is defined as follows:

$$DC(f_{ind}) = DC(f_x) \cdot DC(f_{\bar{x}}) \quad (22)$$

$$ON(f_{ind}) = [ON(f_x) + DC(f_x)] \cdot [ON(f_{\bar{x}}) + DC(f_{\bar{x}})] \# DC(f_{ind}) \quad (23)$$

$$ON(f_{dep}) = ON(f) \# ON(f_{ind}) \quad (24)$$

$$DC(f_{dep}) = DC(f) + ON(f_{ind}) \quad (25)$$

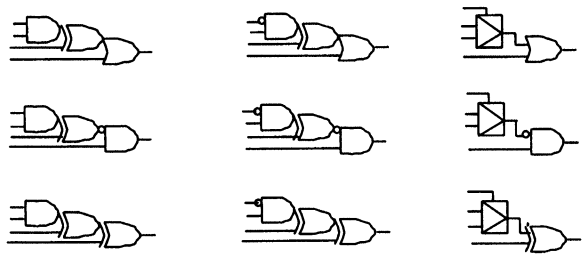

FIGURE 3 BTDD node realizations.

The non-canonical INHIBITION type expansion is defined as follows:

$$DC(f_{ind}) = DC(f_x) \cdot DC(f_{\bar{x}}) \quad (26)$$

$$ON(f_{ind}) = [ON(f_x) + DC(f_x) + ON(f_{\bar{x}}) + DC(f_{\bar{x}})] \# DC(f_{ind}) \quad (27)$$

$$ON(f_{dep}) = ON(f_{ind}) \# ON(f) \quad (28)$$

$$DC(f_{dep}) = DC(f) + \overline{ON(f_{ind})} \quad (29)$$

The non-canonical EXOR type expansion is defined as follows:

$$ON(f_{ind}) = (ON(f_x) \# [ON(f_{\bar{x}}) + DC(f_{\bar{x}})]) + ([ON(f_x) + DC(f_x)] \# ON(f_{\bar{x}})) \quad (30)$$

$$DC(f_{ind}) = DC(f_x) + DC(f_{\bar{x}}) \quad (31)$$

$$ON(f_{dep}) = (ON(f) \# [ON(f_{ind}) + DC(f_{ind})]) + (ON(f) + DC(f)] \# ON(f_{ind})) \quad (32)$$

$$DC(f_{ind}) = DC(f_x) + DC(f_{\bar{x}}) \quad (33)$$

Let us observe that the OR expansion above, (22)–(25), results in the ON set of f_{ind} having the largest function included in the ON set of the function f.

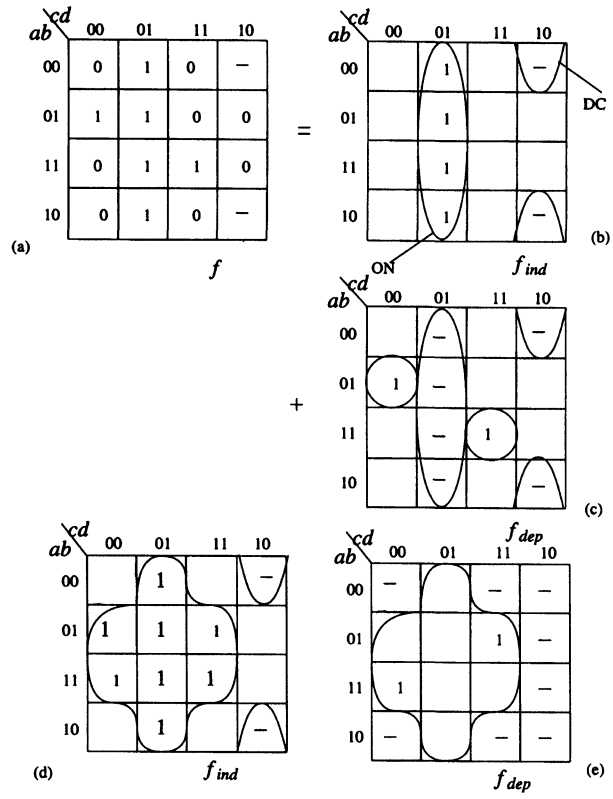

FIGURE 4 Example of a decomposition.

Therefore, assuming OR gate, this expansion is unique (canonical). Function f can then be considered as an OR of f_{ind} and a "remainder" function, called f_{dep}. In a similar way, for the AND expansion, the ON set of f_{ind} will be the smallest function that includes the ON set of function f. Therefore, this expansion is also unique for an INHIBITION gate (a similar expansion can also be defined for an AND gate). Thus, the above two combined expansions are canonical.

However, in the case of the EXOR gate, in principle any two-input function of functions f_x and $f_{\bar{x}}$ can be applied for f_{ind}, since each of these functions is independent of x_i. Hence there are many more possible canonical dependency expansions for EXOR. Our choice of EXOR as the two-input functions shown is, in a sense, arbitrary. It is, however, motivated by EXOR being simple and distinctly different from OR and INHIBITION, so possibly leading to searching larger space.

Example: Let function f be given as shown in Fig. 4a. By applying OR type expansion to variable a, one obtains function f_{ind} shown in Fig. 4b and function f_{dep} in Fig. 4c. (In the figures, empty cells represent the OFF sets.) As shown in Fig. 4, $f = f_{ind} + f_{dep}$. Similarly, applying an INHIBITION type expansion, function f can be represented as a product:

$$f = f_{ind} \cdot \overline{f_{dep}},$$

where functions f_{ind} and f_{dep} are as shown in Figs. 4d, and 4e, respectively.

Fig. 5a presents the realization of f using an OR gate and a MUX. By combining the OR gate and the MUX to an OR/MUX, the realization of Fig. 5a is transformed to that of Fig. 5b. By treating the OR/MUX gate as a node, and rotating Fig. 5b so that OR/MUX node would be on the top, one creates the first node of a BTDD. Next the expansions are applied to circuits described by the AND gates from Fig. 5b, and a complete BTDD is created.

Fig. 5c presents the same function f as in Fig. 5a, realized using a standard MUX (a BDD node). This figure illustrates the advantage of the BTDD nodes over standard MUX nodes used to implement BDDs: the product $\bar{c}d$, which is independent of variable a, is split in the MUX realization into two products, $a\bar{c}d$ and $\bar{a}\bar{c}d$, which are realized by the paths through the "high" and "low" inputs of the MUX, respectively. These two paths have a common part $\bar{c}d$ which would be present at both inputs of the MUX. As it can be seen, this approach makes the MUX circuit both slower and larger. It is slower since signal $\bar{c}d$ has to go through the OR gates and the MUX (which is slower than the OR). It is larger since the OR gates on the inputs of the MUX are now necessary. The advantage of BTDD over BDD can be also illustrated on other types of functions.

Fig. 5d presents a circuit for the same function f, realized according to Figs. 4d and 4e, with INHIBIT/MUX gate for variable a and OR/MUX gate for variable b.

The *Non-canonical BKTDDs* have two types of nodes: binary KDD nodes (Shannon, Positive Davio, Negative Davio), and ternary nodes as in BTDDs. Binary nodes are evaluated as described in sections 2–3.

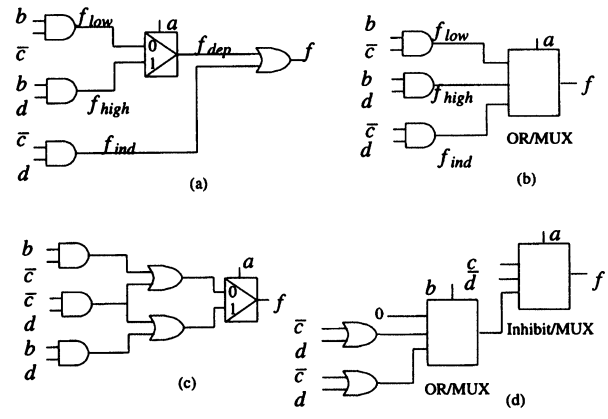

FIGURE 5 The realization of the decomposed function in Figure 4.

In general, for both incompletely specified and completely specified Boolean functions, these expansions are not unique and thus lead to non-canonical diagrams. The expansions for completely specified functions are not canonical since the OR and INHIBITION expansions produce don't care terms in the expansion process. Starting with a completely specified function f, every level of variable expansions adds more don't care terms to f_{ind} and f_{dep} functions of lower levels. This property, not shared by the algorithms to create DDs, is very advantageous in synthesis, since functions with many don't cares can be better minimized. However, the same property causes these diagrams to be non-canonical. Thus, they can not be used for a general-purpose Boolean function representation.

All standard concepts for DDs: free, ordered, reduced, shared, repeated variable, pseudo, can be applied to the non-canonical diagrams created as above.

BTDDs and BKTDDs can now be formally defined.

Definition 14: A **ternary decision diagram** is a rooted, directed acyclic graph with vertex set V containing two types of vertices. A **terminal** vertex v has as attribute a value $value(v) \in \{0, 1\}$. A **non-terminal** vertex v has as attributes an argument index $index(v) \in \{1,\ldots, n\}$ and three successors $dep_{low(v)}$, $dep_{high(v)}$, $ind(v) \in V$.

Definition 15: A non-canonical **Boolean Ternary Decision Diagram (BTDD)** is a decision diagram having root vertex v denoting a function f_v denoted recursively as:

(1) If v is a terminal vertex:
   (a) If $value(v) = 1$, then $f_v = 1$.
   (b) If $value(v) = 0$, then $f_v = 0$.

(2) If v is a non-terminal vertex with $index(v) = i$, then f_v is one and only one of the functions:
   (a) $f_v(x_1,\ldots, x_n) = f_{ind}\ OPER\ \{f_{dep_{low(v)}}(x_1,\ldots, x_n) \oplus x \cdot [f_{dep_{high(v)}}(x_1,\ldots, x_n) \oplus f_{dep_{low(v)}}(x_1,\ldots, x_n)]\}$.
   (b) $f_v(x_1,\ldots, x_n) = f_{ind}\ OPER\ \{f_{dep_{high(v)}}(x_1,\ldots, x_n) \oplus \bar{x} \cdot [f_{dep_{high(v)}}(x_1,\ldots, x_n) \oplus f_{dep_{low(v)}}(x_1,\ldots, x_n)]\}$.
   (c) $f_v(x_1, \ldots x_n) = f_{ind}\ OPER\ \{\bar{x} \cdot f_{dep_{low(v)}}(x_1,\ldots, x_n) \oplus x \cdot [f_{dep_{high(v)}}(x_1, \ldots, x_n)\}$.

where OPER $\in$ {OR, INHIBITION, EXOR}

f_{dep}, f_{ind} are functions calculated in (22)–(33), corresponding to the selected operator type $OPER$. Expansions for $f_{dep_{high(v)}}(x_1, \ldots, \bar{x}_n)$, $f_{dep_{low(v)}}(x_1, \ldots, x_n)$, and $[f_{dep_{high(v)}}(x_1, \ldots, x_n) \oplus f_{dep_{low(v)}}(x_1, \ldots, x_n)]\}$ are calculated according to "constrained don't care method" from section 3.

Definition 16: A non-canonical **Pseudo-Boolean Ternary Decision Diagram (PBTDD)** is a Shared Reduced Ordered Boolean Ternary Decision Diagram.

Similar definitions of non-canonical: Free Boolean Ternary Decision Diagrams (FBTDD), Pseudo-Boolean Kronecker Ternary Decision Diagrams (PBKTDD), Free Boolean Kronecker Ternary Decision Diagrams (FBKTDD), and other diagrams can be introduced.

7. CANONICAL BOOLEAN TERNARY DECISION DIAGRAMS AND BOOLEAN KRONECKER TERNARY DECISION DIAGRAMS

The concept of canonical BTDDs and BKTDDs are created for completely specified Boolean functions by defining OR, INHIBITION, and EXOR expansions that create no don't care terms and therefore make each of the functions f_{dep} and f_{ind} completely specified, and thus unique.

The canonical OR type expansion for canonical BTDDs is defined as follows:

$$f_{ind} = f_x \cdot f_{\bar{x}} \tag{34}$$

$$f_{dep} = f\,\#\,(f_x \cdot f_{\bar{x}}) \tag{35}$$

The canonical INHIBITION type expansion for canonical BTDDs is defined as follows:

$$f_{ind} = f_x + f_{\bar{x}} \tag{36}$$

$$f_{dep} = (f_x + f_{\bar{x}})\,\#\,f \tag{37}$$

The canonical EXOR type expansion for canonical BTDDs is defined as follows:

$$f_{ind} = f_x \oplus f_{\bar{x}} \tag{38}$$

$$f_{dep} = f \oplus (f_x \oplus f_{\bar{x}}) \tag{39}$$

It can be easily proven that for every completely specified Boolean function, all of these expansions are unique and thus, combined with unique expansions. Shannon, Positive Davio and Negative Davio for f_{high}, lead to canonical DDs.

By defining the nine ternary nodes in BKTDDs with expansions (34)–(39) and the three binary nodes as in KDDs, canonical BKTDDs are created.

Similar to definitions from Section 6, canonical BTDDs, BKTDDs, Free BTDDs, Free BKTDDs, Pseudo BTDDs, and Pseudo BKTDDs can be formulated.

Boolean Ternary DDs are a particular case of *Boolean Multiple Decision Diagrams*—diagrams for Boolean functions that have more than two edges in nodes. Other example of the Boolean Multiple Decision Diagrams are the Orthogonal Decision Diagrams from [26, 25].

8. CONCLUSIONS

In this paper, several families of decision diagrams were introduced which are generalizations or extensions to

both *BDDs* and *FDDs*. Some of these families, such as the KDDs, include all types of expansions from BDDs and FDDs. It is then obvious that KDD diagrams can be more compact. The questions are: How reduced KDDs with heuristics can get for industrial benchmark logic functions? Can one represent some large functions with the KDDs which can not be represented by BDDs? Although our numerical results are very good for some benchmarks, especially the incompletely specified and arithmetic, it results from the comparison with ASYL that there is still much space for improvement in our algorithm, since the exact KDD diagram should have at most as many nodes as a BDD diagram for the same function.

KDD diagrams allow also to represent some distinctly large functions that can not be represented by BDDs and FDDs. An example of a such function is given in [4]. An important question remains: "are there such functions among industrial benchmarks?"

Pseudo-KDDs and Free KDDs make further use of the possibility of mixing types of expansions on a graph level, and changing orders of variables in its sub-graphs. The PKDDs and FKDDs are more difficult to be used as canonical representations than the KDDs. Further work must be thus devoted to create standard DD type algorithms to manipulate such diagrams. However, the PKDDs and FKDDs are already very useful for synthesis [15], since they produce diagrams with clearly less nodes than all the other diagrams.

The Boolean Ternary Diagrams have more powerful nodes than KDDs, so it is very likely that they will have less nodes. Since there are 9 instead of 3 possible expansion types for every expansion variable, the solution space of all diagrams is much larger, and we expect that more compact diagrams will be generated if good heuristics for variable and expansion type selections are found. The canonical BTDDs may be thus good candidates for canonical representations. Similarly, since the circuit realization of a BTDD node is not much more complicated than the realization of a KDD node, and because of decreased diagram size, the canonical BTDDs may be good candidates for synthesis of both completely and incompletely specified Boolean functions.

Finally, the canonical Boolean Ternary Kronecker Decision Diagrams include all the node types of KDDs, and still preserve their canonicity. For selecting the expansion type, one has only one choice in BDDs and original FDDs, two choices in PRMTs and modified FDDs, three choices in KFDDs, nine choices in BTDDs, and twelve choices in BKTDDs. There is no danger of loosing good choices in BKTDDs, since the three standard expansions of the KDD are still available in them. Therefore, the exact BKTDD is never worse than the exact KDD.

Concluding, the introduced classes of decision diagrams open a very wide research area with many possible applications. It should be investigated whether they can drastically improve over BDD-based algorithms in such applications as spectral methods, mapping, decomposition, factorization, transduction, cellular logic synthesis, verification, testing, modeling and simulation, and technology mapping to FPGAs, especially fine-grain FPGAs.

References

[1] Algotronix, "The CHS 2*4, The world's first custom computer", *Algotronix Ltd,* Kings Buildings - TTC, Mayfield Road, Edinburgh EH9 3JL, Scotland, 1992.

[2] A. E. A. Almaini, and M. E. Woodward, "An Approach to the Control Variable Selection Problem for Universal Logic Modules", *Digital Processes,* Vol. 3, pp. 189-206, 1977.

[3] L. A. M. Bennett, "The Application of Map-Entered Variables to the Use of Multiplexers in the Synthesis of Logic Functions", *Int. J. Electronics,* Vol. 45, No. 4, 1978, pp. 373-379.

[4] B. Becker, R. Drechsler, and R. Wechner, "On the Relation Between BDDs and FDDs", *Technical Report,* University of Frankfurt, 12/93, 1993.

[5] T. Besson, H. Bousouzou, M. Crastes, and G. Saucier, "Synthesis on Multiplexer-based Programmable Devices Using (Ordered) Binary Decision Diagrams", *Proc. EURO-ASIC,* pp. 8-13, June 1992, Paris, France.

[6] T. Besson, H. Bousouzou, M. Crastes, and G. Saucier "Synthesis on Multiplexer-based F.P.G.A. Using Binary Decision Diagrams", *Proc. of IEEE ICCD,* pp. 163-167, 1992.

[7] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel Logic Synthesis", *Proc of the IEEE.* Vol. 78, No. 2, pp. 264-300, February 1990.

[8] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation", *IEEE Trans. on Comput.,* Vol. 35, No. 8, pp. 667-691, August 1986.

[9] Concurrent Logic Inc., "CLi6000 Series Field Programmable Gate Array", *Preliminary Information,* December 1991.

[10] M. Davio, J. P. Deschamps, and A. Thayse, "Discrete and Switching Functions", Mc.Graw-Hill, 1978.

[11] D. L. Dietmayer, "Logic Design of Digital Systems", *Allyn and Bacon,* 1978.

[12] 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", *Proceedings of DAC '94,* pp. 321-326, San Diego, CA, June 1994.

[13] M. Fujita, Y. Matsunga, and T. Kakuda, "On variable ordering of binary decision diagrams for the application of multi-level synthesis", *European Conf. on Design Automation,* pp. 50-54, 1991.

[14] D.H. Green and P.W. Foulk "Adaptive Logic Trees for Use in Multilevel-Circuit Design", *Electr. Letters,* Vol. 5, pp. 83-84, 1969.

[15] Ph. Ho. and M. A. Perkowski, "Minimization of Fine-Grain FPGAs Using Free Kronecker Decision Diagrams", *European Conf. on Design Automation,* 1994.

[16] C. Jay, "XOR PLDs Simplify Design of Counters and Other Devices", *EDN,* May, 1987.

[17] K. Karplus, "ITEM: An If-Then-Else Minimizer for Logic

Synthesis", *Proc. EURO-ASIC,* pp. 2-7, June 1992, Paris, France.

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

[19] A. M. Lloyd, "Design of Multiplexer Universal-Logic-Module Networks Using Spectral Techniques", *IEE Proc. Pt. E.,* Vol. 127, pp. 31-36, January 1980.

[20] S. Malik, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Graph-Based Algorithms for Boolean Function Manipulation", *IEEE Proc. of Int. Conf. on CAD,* pp. 6-9, 1988.

[21] S. Minato, N. Ishiura, and S. Yajima, "Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation", *Proc. 27th ACM/IEEE DAC,* pp. 52-57, 1990.

[22] Motorola MPA10XX Data Sheet, 1994.

[23] M.A. Perkowski, P. Dysko, and B.J. Falkowski, "Two Learning Methods for a Tree-Search Combinatorial Optimizer", *Proc. of IEEE Int. Conf. on Comput. and Comm.,* Scottsdale, Arizona, 1990, pp. 606-613.

[24] M.A. Perkowski, and P. D. Johnson, "Canonical Multi-Valued Input Reed-Muller Trees and Forms", *Proc. 3rd NASA Symposium on VLSI Design,* Moscow, Idaho, October 1991, pp. 11.3.1-11.3.13.

[25] M. A. Perkowski, "The Generalized Orthonormal Expansions of Functions with Multiple-Valued Inputs and Some of its Applications", *Proc. 22nd ISMVL,* pp. 442-450, May, 1992, Sendai, Japan.

[26] M. A. Perkowski, "A Fundamental Theorem for Exor Circuits", *Proc. of the IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design,* Hamburg, Germany, September 1993.

[27] W. Rosenstiel (Ed.), Proceedings of the IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, Hamburg, Germany, September 1993.

[28] R. Rudell, "Dynamic Variable Ordering for Ordered Binary Decision Diagrams", *Proc. Int. Conf. on Computer Aided Design,* pp. 42-47, 1993.

[29] A. Sarabi, P.F. Ho, K. Iravani, W.R. Daasch, and M. Perkowski, "Minimal Multi-level Realization of Switching Functions based on Kronecker Functional Decision Diagrams", *Proc. IWLS '93, International Workshop on Logic Synthesis,* P3a, Taho City, CA, 1993

[30] T. Sasao, and Ph. Besslich, "On the Complexity of MOD-2 Sum PLAs", *IEEE Trans. on Comput.,* Vol. 39, No. 2, pp. 262-266, 1990.

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

[32] I. Schafer, M. A. Perkowski, and H. Wu, "Multilevel Logic Synthesis for Cellular FPGAs Based on Orthogonal Expansions", *Proceedings of the IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design,* Hamburg, Germany, September 1993.

[33] E. Schubert, U. Kebschull, and W. Rosenstiel, "FDD Based Technology Mapping for FPGA", *Proc. EURO-ASIC,* pp. 14-18, June 1992, Paris, France.

[34] Tabloski, T.F., and F. J. Mowle, "A Numerical Expansion Technique and Its Application to Minimal Multiplexer Logic Circuits", *IEEE Trans. on Comput.,* Vol. 25, No. 7, 1976, pp. 684-702.

[35] A. J. Tosser, and D. Aoulad-Syad, "Cascade Networks of Logic Functions Built in Multiplexer Units", *IEE Proc. Pt. E,* Vol. 127, No. 2, pp. 64-68, March 1980.

[36] L.-F. Wu, and M. A. Perkowski "Minimization of Permuted Reed-Muller Trees for Cellular Logic Programmable Gate Arrays", *2nd Int. Workshop on FPGAs,* September 1992, Vienna, Austria.

Biographies

MAREK A. PERKOWSKI received the M. S. degree in electronics in 1970 and the Ph. D. degree3 in automatics (digital systems) in 1980 from Technical University of Warsaw. He was an Assistant Professor at the University of Warsaw from 1980 to 1981, a Visiting Assistant Professor at the University of Minnesota from 1981 to 1983, and since 1983 he has been at Portland State University where he is a Professor of Electrical and Computer Engineering. He is the co-author of three books, seven book chapters, and over 120 technical articles in design automation, computer architecture, artificial intelligence, image processing and robotics.

INGO SCHÄEFER received the B. S. E. E. degree from the University of Stuttgart, Germany, in 1987 and the Ph. D. degree in electrical and computer engineering from Portland State University in 1992. From 1989 to 1992 he was a member of the CAD group at Portland State University. In 1992, he joined the logic synthesis group at Lattice Semiconductor. His research interests include various issues in computer-aided design as well as in digital signal and image processing.

ANDISHEH SARABI received the M. S. degree in Mathematical Sciences in 1991 and the Ph. D. degree in electrical and computer engineering in 1994 both from Portland State University. He has been with the CAD group at Portland State U&niversity since 1989. His research interests are in logic synthesis, cellular FPGAs, XOR logic, and testing.

MALGORZATA CHRZANOWSKA-JESKE received the M. Sc. degree in electronic engineering from the Technical University of Warsaw in 1972, and the Ph. D. degree in electrical engineering from Auburn University in 1988. In 1972, she joined the Electronic Engineering Faculty at the Technical University of Warsaw. In 1977, she became a Research Staff Member of the Research and Production Center of Semiconductor Devices, Warsaw, Poland. Since 1989, she has been an Assistant Professor in the Department of Electrical Engineering at Portland State University. She has published more than 30 technical papers and articles. Her research interests are in computer-aided-design of integrated circuits, device simulation and low-temperature electronics.