Ordered Kronecker Functional Decision Diagrams—A Data Structure for Representation and Manipulation of Boolean Functions

Rolf Drechsler, *Member, IEEE*, and Bernd Becker, *Member, IEEE*

***Abstract*— Ordered Kronecker functional decision diagrams (OKFDD's) are a data structure for efficient representation and manipulation of Boolean functions. OKFDD's are a generalization of ordered binary decision diagrams (OBDD's) and ordered functional decision diagrams and thus combine the advantages of both. In this paper, basic properties of OKFDD's and their efficient representation and manipulation are given.**

Starting with elementary manipulation algorithms, we present methods for the construction of small OKFDD's. Our approach is based on dynamic variable ordering and decomposition-type choice. For changing the decomposition type, we use an efficient reordering-based method. We briefly discuss the implementation of PUMA, an OKFDD package, which was used in all our experiments. These experiments demonstrate the quality of our methods in comparison to sifting and interleaving for OBDD's.

***Index Terms*—Binary decision diagram (BDD), Boolean function, decision diagram, dynamic minimization, logic synthesis, verification.**

I. Introduction

DECISION diagrams (DD's) are often used in CAD systems for efficient representation and manipulation of Boolean functions. The most popular data structure in this context is ordered binary decision diagrams (OBDD's) [10], which are used in many applications [11]. Nevertheless, some relevant classes of Boolean functions cannot be represented efficiently by OBDD's [5], [27]. As one alternative, ordered functional decision diagrams (OFDD's) [16], [21] have been introduced and in the meantime are used in various applications of XOR-based logic synthesis (see, e.g., [13]). If ease of manipulation and canonicity is not a main concern, still other types of decision diagrams, like ternary decision diagrams [28] and (free) Kronecker functional decision diagrams [20], have proven to be useful, especially in the area of technology mapping for multilevel XOR-based circuitry.

Recently, ordered Kronecker functional decision diagrams (OKFDD's) have been introduced as a means for efficient representation and manipulation of Boolean functions [15]. OKFDD's are a generalization of OBDD's and OFDD's as well and try to combine the advantages of both representations by allowing the use of Shannon decompositions and (positive and negative) Davio decompositions in one and the same decision diagram. Thus, the data structure has the potential to dynamically adapt the representation of a Boolean function to a given problem.

From a (more) theoretical point of view, it has been shown that there exist certain classes of Boolean functions whose OFDD size is exponentially smaller than the OBDD representation of the same function and vice versa [5]. Thus, it is useful to consider a representation, like OKFDD's, that integrate both OBDD's and OFDD's. Furthermore, it has been proved in [4] that a "restriction" of the OKFDD concept results in families of functions that lose their efficient representations. It follows that OKFDD's in full generality should be considered. On the other hand, based on [1], OKFDD's are the "most general type" of ordered decision diagram (ODD's). In this sense, it is interesting and important to also devise effective practical algorithms for representing and manipulating Boolean functions with OKFDD's.

As is well known for OBDD's [10] and OFDD's [5], OKFDD's are also very sensitive to the variable ordering [15], and the computation of an optimal variable ordering is NP-hard [7]. In addition to the position of a variable in the ordering, a so-called decomposition type has to be chosen for OKFDD's. Thus, there is a need for heuristics to choose a suitable variable ordering and decomposition type list for OKFDD's.

In [14], first topology-based heuristics have been presented. These heuristics allow the fast construction of OKFDD's from given circuit descriptions. But often these heuristics fail to determine small graphs. In [15], it has been shown that dynamic variable ordering methods for OBDD's [19], [27] can also be applied to OKFDD's. But the methods presented there were rather time consuming and thus are only applicable to small circuits within reasonable time bounds.

In this paper, we briefly discuss the implementation of basic synthesis operations on OKFDD's and then present effective algorithms to perform dynamic variable ordering and decomposition-type choice for OKFDD's. In particular, a new method for exchanging the decomposition type of a variable is introduced. Taken together, we succeeded in the application of dynamic methods also to larger examples without spending too much time.

We briefly describe the features of our OKFDD package, PUMA. Using this package, we present a large set of experi-

Manuscript received February 20, 1996; revised December 17, 1997. This paper was recommended by Associate Editor M. Fujita.

R. Drechsler is with the Institute of Computer Science, Albert-Ludwigs University, Freiburg 79110 Germany (e-mail: drechsle@informatik.uni-freiburg.de).

B. Becker is with the University of Freiburg im Breisgau, Freiburg 79110 Germany (e-mail: becker@informatik.uni-freiburg.de).

Publisher Item Identifier S 0278-0070(98)08489-9.

ments to clarify the relation between OBDD's and OKFDD's with respect to size and run time. We also demonstrate that our approach can be helpful for technology mapping, i.e., the minimized OKFDD's can be mapped on field programmable gate array (FPGA's), as has been proposed in [29]. Using our dynamic methods, we can often reduce the OKFDD size by more than 50%. This simplifies the mapping process tremendously.

This paper is structured as follows. OKFDD's are introduced in Section II. The basic manipulation algorithms are described in Section III. Furthermore, the implementation of PUMA and the main features are discussed. Different methods for the minimization of OKFDD's by dynamic variable ordering and decomposition-type choice are given in Section IV. Experimental results are described in Section V. In Section VI, the results are summarized.

II. Ordered Kronecker Functional Decision Diagrams

In the following, basic definitions of DD's and OKFDD's in particular are presented.

Definition 1: A DD over $X_n := \{x_1, x_2, \cdots, x_n\}$ is a rooted directed acyclic graph $G = (V, E)$ with vertex set V containing two types of vertices: *nonterminal* and *terminal*. A nonterminal vertex v is labeled with a variable from X_n, called the *decision variable*, for v and has exactly two successors, denoted by low(v), high$(v) \in V$. All nodes with label x_i are denoted as level i. A terminal vertex v is labeled with a "0" or "1" and has no successors. The size of a DD, denoted by $|DD|$, is given by its number of nonterminal nodes.

Further structural restrictions turn out to be important.

Definition 2: A DD is *free* if each variable is encountered at most once on each path in the DD from the root to a terminal vertex. A DD is *ordered* if it is free and the variables are encountered in the same order on each path in the DD from the root to a terminal vertex.

In the following, letter O will be used to denote ordered DD's.

DD's can be related to Boolean functions by using the following well-known decomposition types (given here for an arbitrary Boolean function f: $\mathbf{B}^n \rightarrow \mathbf{B}$ over the variable set X_n):

$$f = \overline{x}_i f_i^0 + x_i f_i^1 \quad \text{Shannon (S)} \tag{1}$$

$$f = f_i^0 \oplus x_i f_i^2 \quad \text{positive Davio (pD)} \tag{2}$$

$$f = f_i^1 \oplus \overline{x}_i f_i^2 \quad \text{negative Davio (nD)} \tag{3}$$

where $f_i^0 (f_i^1)$ denotes the *cofactor* of f with respect to $x_i = 0$ $(x_i = 1)$, and f_i^2 is defined as $f_i^2 := f_i^0 \oplus f_i^1$, $\oplus$ being the exclusive OR operation.

Decomposition types are associated to the variables in X_n with the help of a decomposition-type list (DTL) $d := (d_1, \cdots, d_n)$ where $d_i \in \{S, pD, nD\}$.

Now, KFDD's can formally be defined as follows:

Definition 3: A KFDD over X_n is given by a DD over X_n together with a fixed DTL, $d = (d_1, \cdots, d_n)$. The function f_G^d: $\mathbf{B}^n \rightarrow \mathbf{B}$ represented by a KFDD G over X_n with DTL d is defined as follows.

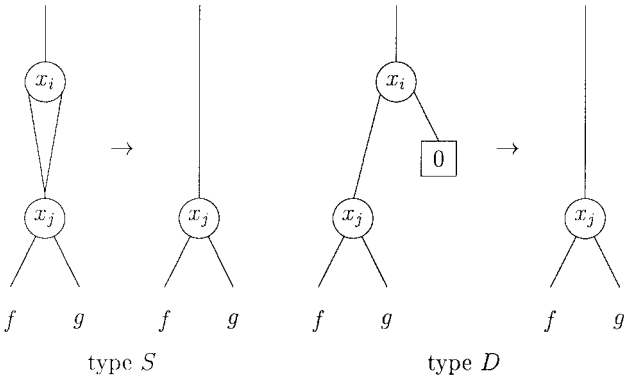

Fig. 1. Reduction types.

1) If G consists of a single node labeled with 0 (1), then G is a KFDD for the constant 0 (1) function.
2) If G has a root v with label x_i, then G is a KFDD for

$$\begin{cases} \overline{x}_i f_{\text{low(v)}} + x_i f_{\text{high(v)}} : & d_i = \text{S} \\ f_{\text{low(v)}} \oplus x_i f_{\text{high(v)}} : & d_i = \text{pD} \\ f_{\text{low(v)}} \oplus \overline{x}_i f_{\text{high(v)}} : & d_i = \text{nD} \end{cases}$$

where $f_{\text{low(v)}}$ $(f_{\text{high(v)}})$ is the function represented by the KFDD rooted at *low*(v)(*high*(v)).

Of course, a KFDD is an OKFDD iff the underlying DD is ordered.

A node in a KFDD is called an S-node if it is expanded by Shannon decomposition (1). It is called a D-node if it is expanded by Davio decompositions [(2) or (3)], the latter being an nD-node and the former a pD-node. According to the DTL, at every node of a fixed level i in the KFDD, the same decomposition of type d_i is applied. This directly infers that KFDD's are a generalization of BDD's and FDD's: if in all levels only Shannon decomposition is applied, the KFDD will be a BDD. If only Davio decompositions are applied, i.e., $d_i \in \{pD, nD\}$ for all i, the KFDD will be an FDD. Analogously, pFDD's and nFDD's are defined.

Reductions of three different types are used to reduce the size of KFDD's.

- Type I) Delete a node v' whose label is identical to the label of a node v and whose successors are identical to the successors of v and redirect the edges pointing to v' to point to v.
- Type S) Delete a node v whose two outgoing edges point to the same node v' and connect the incoming edges of v to v'.
- Type D) Delete a node v whose successor *high*(v) points to the terminal 0 and connect the incoming edges of the deleted node to *low*(v).

In Fig. 1, the graphical representation of reductions of type S and D are shown. While each node in a KFDD is a candidate for the application of the reduction type I, only S-nodes (D-nodes) are candidates for the application of the reduction type S (reduction type D). A KFDD is *reduced* iff no reductions can be applied to the KFDD. Two KFDD's G_1 and G_2 (with the same DTL's) are called *equivalent* iff G_2 results from G_1 by repeated applications of reductions and inverse reductions.

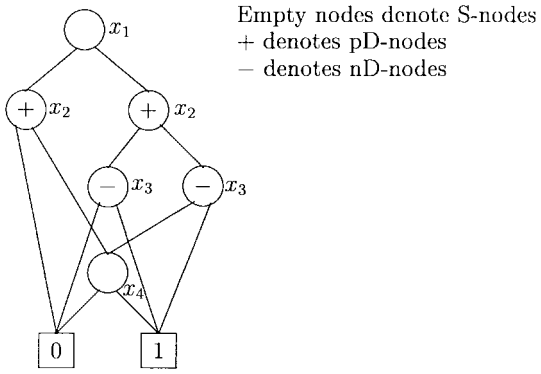

Fig. 2. Example for OKFDD.

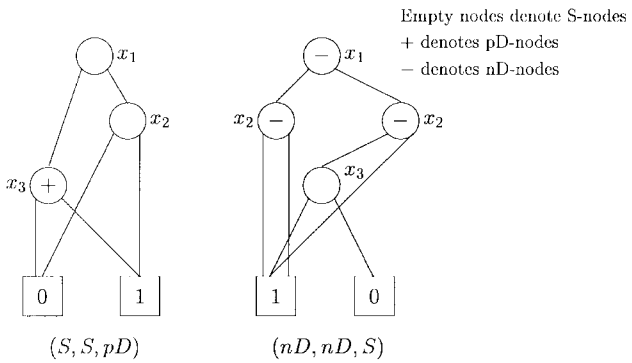

Fig. 3. Example for OKFDD's with different DTL's.

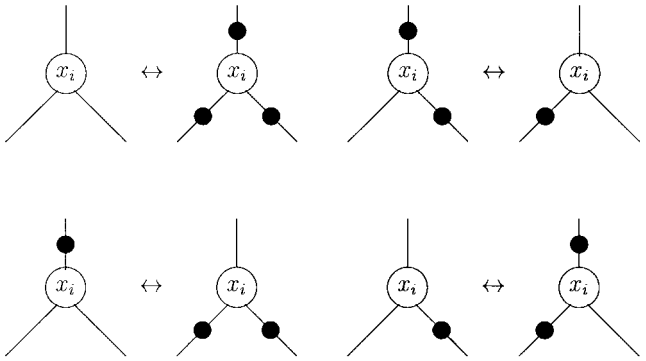

Fig. 4. Identification for canonical form for S-nodes.

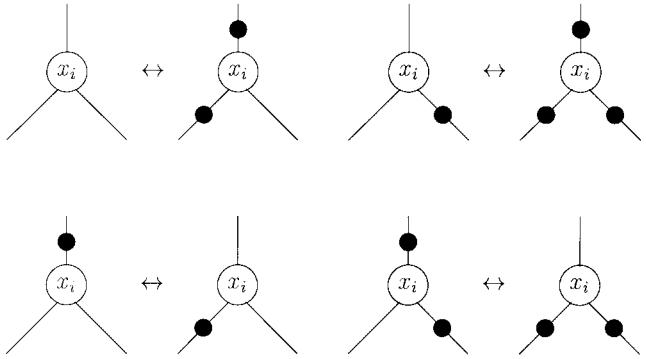

Fig. 5. Identification for canonical form for D-nodes.

A KFDD G_2 is called the *reduction* of a KFDD G_1 iff G_1 and G_2 are equivalent and G_2 itself is reduced.

From [15], it is known that reductions can be used to define canonical representations for not only OBDD's and OFDD's but also for OKFDD's.

Example 1: An OKFDD is shown in Fig. 2, where the left outgoing edge at each node points to $f_{\text{low(v)}}$. The OKFDD represents the function $x_1x_2x_4 \oplus x_1x_2\overline{x}_3 \oplus x_1\overline{x}_3 \oplus \overline{x}_1x_2x_4$. The S-node decomposes the function into x_2x_4 and $x_2x_4 \oplus x_2\overline{x}_3 \oplus \overline{x}_3$, respectively. The latter is in turn decomposed into $\overline{x}_3$ and $\overline{x}_3 \oplus x_4$ through a pD-node labeled with x_2. The nD-node x_3 on the right-hand side results in x_4 and 1.

In contrast to OBDD's, in OKFDD's, the choice of decomposition type also influences the size of the representation.

Example 2: Two OKFDD's of size three and four, respectively, are shown in Fig. 3. [The left outgoing edge at each node denotes $f_{\text{low(v)}}$.] Both OKFDD's represent the same function $f = x_1x_2 + \overline{x}_1x_3$. The OKFDD on the left-hand side has the DTL (S, S, pD), and the OKFDD on the right-hand side has the DTL (nD, nD, S).

Thus, the choice of the DTL influences the size of the resulting graph. In [3] and [5], it has been shown that OKFDD's may have polynomial size if a *good* DTL is chosen, but they may also have exponential size if the DTL is *bad*.

III. Basic Algorithms and Their Integration in an OKFDD Package

In this section, basic OKFDD manipulation algorithms, their implementation, and their computational complexity are discussed. Last, we present the main features of our OKFDD package PUMA.

A. Basic Algorithms

1) Complement Edges: The size of an OKFDD can be further reduced if complement edges (CE's) are used analogously to OBDD's [8]. Then a node represents a function and its complement at the same time. The representation can be chosen in a way that it further remains canonical [15]. For this, different pairs that represent the same function have to be identified. The corresponding pairs for S- and D-nodes are given in Figs. 4 and 5, respectively. A dot on a line symbolizes a CE. Here, always the left representative is used, i.e., the *low* edge (f^0 edge) must be a regular uncomplemented edge.

For the storage of the additional information, 1 bit is needed. As suggested in [8] and [12], the overhead can be saved if the lowest bit of each pointer is used on machines where this is allowed.

2) Boolean Operations on OKFDD's: In the following, algorithms for OKFDD's with fixed variable ordering and a fixed DTL are given.

First, the XOR-operation is presented, as it provides the basis for construction of other operations used in the following. Notice that for two functions f and g, decomposed by positive Davio expansion, one has

$$\begin{aligned} f \oplus g &= (f_0 \oplus x_i f_2) \oplus (g_0 \oplus x_i g_2) \\ &= (f_0 \oplus g_0) \oplus x_i (f_2 \oplus g_2). \end{aligned}$$

```
kfdd_xor_kfdd(F, G) {
  if (terminal case) {
    return result;
  } else if (computed-table has entry (F, G)) {
    return result;
  } else {
    let v be the top node of (F, G);
    low(v) = kfdd_xor_kfdd(F_low(v), G_low(v));
    high(v) = kfdd_xor_kfdd(F_high(v), G_high(v));
    if Shannon(v) {
    if (high(v) == low(v)) return low(v);
    } else {
    if (high(v) == 0) return low(v);
    }
    R = find_or_add_unique_table(v, low(v), high(v));
    insert_computed_table(F, G, R);
    return R;
  }
}
```

Fig. 6. Algorithm for XOR-operation.

This equation makes it possible to split up a positive Davio node recursively into its left and right subgraphs. The algorithm for negative Davio nodes is performed analogously. Altogether, this provides an efficient algorithm for Davio nodes. The basic XOR-operation for Shannon nodes is based on the following equation:

$$f \oplus g = \overline{x}_i(f_0 \oplus g_0) + x_i(f_1 \oplus g_1).$$

The resulting algorithm for XOR-operation on two OKFDD's F and G is sketched in Fig. 6. Analogously to [10], it follows that the algorithm has an asymptotic worst case behavior that is bounded from above by the product of the OKFDD sizes of F and G.

The realization of the AND-operation turns out to be more complicated for Davio nodes in comparison to the XOR-operation. The following recursive equation holds for positive Davio nodes:

$$\begin{aligned} f \cdot g &= (f_0 \oplus x_i f_2) \cdot (g_0 \oplus x_i g_2) \\ &= (f_0 \cdot g_0) \oplus x_i((f_2 \cdot g_2) \oplus (f_0 \cdot g_2) \oplus (g_0 \cdot f_2)). \end{aligned}$$

This equation again defines a recursive algorithm similar to the one from Fig. 6, which has exponential worst case running time [5]. The same result holds for negative Davio nodes. However, for OKFDD's with a constant number of levels, where the Davio expansion is performed, the operation is polynomial since in these cases, the efficient synthesis operations on Shannon nodes can be carried out in the rest of the graph.

The negation of a function f can be computed by observing that $\overline{f} = 1 \oplus f$. Thus, the operation requires an XOR-operation with the constant "1." Since our package uses complemented edges, negation can be performed even more efficiently, simply by setting a complement edge.

Now, using the algorithms for the XOR-, AND-, and NOT-operations, any binary operation can be realized.

3) Change of Decomposition Type by Using the XOR-Operation: As an example, we demonstrate how a Shannon node can be transformed into a positive Davio node. All other transformations can be performed analogously.

Let v be a Shannon node (labeled with x_i) that is to be transformed into a positive Davio node v'. Then the following is done: the successor *low*(v) can be directly used for *low*(v') since it already represents the cofactor with respect to $x_i = 0$. For the successor *high*(v'), an XOR-operation has to be performed on the successors of v to compute the XOR of the two cofactors. This operation can be performed efficiently for OKFDD's as shown before. It follows directly that also the transformation of all nodes of level i from Shannon to positive Davio can be performed in time quadratic in the size of the OKFDD.

4) Restriction of Variables: For an OKFDD G, the restriction $G|_{x_i=c}$ for variable x_i and constant c can be computed by traversing the graph and performing the corresponding substitutions. The case for Shannon nodes is given by [8]. For the case of positive Davio nodes, the following is done. If $x_i = 0$, then at each node v with label x_i, the edge to *high*(v) has to be deleted. All edges ending in v are redirected to point to *low*(v). If nodes within degree zero result, they and their outgoing edges are also deleted. Clearly, all this can be done in linear time. If $x_i = 1$, then at each node v, with label x_i and subfunctions g_0 and g_1, the following has to be done. As before, the *high* edge has to be deleted; at the *low* edge, an OKFDD for $g_0 \oplus g_1$ is rooted, i.e., an XOR-operation is executed; and all edges ending in v are redirected to point to *low*(v). If nodes within degree zero result, they and their outgoing edges are also deleted. Altogether, this can be done in quadratic time.

For negative Davio nodes, a similar procedure is required.

We summarize the results of this subsection in the following theorem.

Theorem 1: Let G_1 and G_2 be two OKFDD's (with the same variable ordering and the same DTL d) for the functions f_1 and f_2. Then the following hold.

1) The negation of an OKFDD can be performed in constant time.
2) An OKFDD for $f_1 \oplus f_2$ can be computed in time and space $O(|G_1| \cdot |G_2|)$.
3) An algorithm for the computation of $f_1 \cdot f_2$ and $f_1 + f_2$, respectively, has exponential worst case run time (independent of the implementation). If the number of Davio levels is constant, the run time of the algorithm becomes polynomial.
4) Changing the decomposition type of one variable in the OKFDD G has time and space complexity $O(|G|^2)$. Changing all decomposition types in an OKFDD needs an algorithm with exponential worst case behavior.
5) The restriction $G|_{x_i=c}$ for a variable x_i and a constant c can be performed efficiently for OKFDD's.

a) For Shannon nodes, the algorithm has complexity $O(|G|)$.

b) For positive (negative) Davio nodes and $c = 0$ ($c = 1$), the algorithm has complexity $O(|G|)$.

c) For positive (negative) Davio nodes and $c = 1$ ($c = 0$), the algorithm has complexity $O(|G|^2)$.

Proof: From the above list, 1), 2), the first part of 4), and 5) follow from the discussion above.

In [5], an example has been constructed where the AND-synthesis of two polynomial OFDD's results in an OFDD of exponential size. Since OKFDD's are a superset of OFDD's, this implies the result for the AND-synthesis. (The result for the OR-synthesis easily follows from 1) and the application of DeMorgan.)

A trivial method to show that the AND-synthesis remains polynomial if the number of Davio levels remains constant is to transform the OKFDD to an OBDD. (Because only a constant number of operations has to be performed and each transformation requires only polynomial time [see 4)], this can be done efficiently.) The AND-operation on OBDD's has polynomial worst case behavior, and after the AND-operation, the OBDD is transformed back in an OKFDD (again in polynomial time).

The second part of 4) again follows from an example given in [5].

This proves the assertions of the theorem. □

B. Implementation of an OKFDD Package

1) Technical Details: First, programming techniques and methods of implementation used to speed up the package are described. The methods are similar to other packages used for representation and manipulation of OBDD's and OFDD's [2], [8], [23]. Hence, these techniques are only briefly reviewed.

For the fast availability of the functions, a *hash-based unique table* is used to store the nodes. A *computed table* is implemented for the optimization of the synthesis algorithms.

The memory management is done by *garbage collection*. The nodes are only deleted if the storage place is needed for other nodes. Thus, the results need not be recomputed each time if they were used earlier on. By the unique table, different OKFDD's can *share* the same sub-OKFDD's. Therefore, several functions can efficiently be represented at the same time.

2) Features of the Package PUMA: The methods described above have been implemented as the OKFDD package PUMA.[1] The most important features of the package are the following.

1) The package supports Berkeley Logic Interchange Format (BLIF) as standard input format.
2) Several methods for finding good variable orderings and decomposition types can be used.
   a) *Exact minimization:* The algorithm for exact minimization of OKFDD's computes an OKFDD with a minimal number of internal nodes (including CE's). The algorithm can be restricted to a single fixed DTL. (For only Shannon nodes, this results in the exact algorithm for OBDD minimization presented in [17].)
   b) *Heuristic minimization:* For the construction of OKFDD's from standard BLIF-files, the package uses *variable interleaving* [18]. This method has been developed for OBDD's but has proven also to work well for OKFDD's [14].

      Several methods for dynamic variable ordering are supported, like *sifting* and *window permutation* (see also Sections IV and V). Additionally, operations to set an arbitrary variable ordering and/or decomposition type can be used. This allows one to integrate problem-specific ordering methods.

      The package allows dynamic variable ordering not only with an upper node limit in the package but also with respect to a growing factor.
3) Zero-suppressed OBDD's [22] are integrated.
4) The package supports an interactive interface (see Fig. 7). A noncomplete list of features is given in the following.
   a) It can print a profile of the considered graph. This profile distinguishes between different node types.
   b) It can also easily be run on alphanumerical terminals.
   c) Several heuristic methods for dynamic variable ordering with various parameters can be chosen.
   d) The exact algorithm can be applied to a subset of all variables. Thus, large graphs can locally be optimized.

IV. Dynamic Variable Ordering Methods

While the variable ordering plays a dominant role in the identification of the minimal OBDD representation of the functions, in OKFDD's, both the ordering and the decomposition type are important. Depending on the order of the variables and the particular decomposition among the possible three, the size of the OKFDD can vary from linear to exponential [3], [5].

A. Exchange of Neighboring Variables

It is well known that in the case of OFDD's and OBDD's, the size of the decision diagram can be minimized by exchange of adjacent variables. In [15], it has been proven that this idea can be extended to OKFDD's. Therefore, it is also possible to use all techniques based on exchanging of adjacent variables for OKFDD's. The general case for the exchange of a variable i and an adjacent variable j is shown in Fig. 8. Notice that the exchange pattern is independent of the decomposition type of the nodes. The exchange is performed very quickly since only edges must be redirected. In this approach, complemented edges are also used.

[1] PUMA is available by ftp. For more details, contact the authors at ⟨name⟩@informatik.uni-freiburg.de.

```
Command > S
DTL-Sifting
 DD-SIZE : 308 =>
6{N}  12{N}  11{N}  1{S}  7{S}  2{S}  8{S}  3{S}  9{S}  4{S}  10{N}  5{N}
 DD-SIZE : 52
 SHELL-time : 0.89 sec
-------------------------------------------------------------------------------
Command > 0
 All:  PROFILE_LIST:
 <06_I> with label 6  has 2 nodes                    NNNNNNNNNN
 <12_I> with label 12 has 2 nodes                    NNNNNNNNNN
 <11_I> with label 11 has 2 nodes                    NNNNNNNNNN
 <01_I> with label 1  has 6 nodes          SSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
 <07_I> with label 7  has 6 nodes          SSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
 <02_I> with label 2  has 9 nodes SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
 <08_I> with label 8  has 5 nodes             SSSSSSSSSSSSSSSSSSSSSSSSSS
 <03_I> with label 3  has 7 nodes       SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
 <09_I> with label 9  has 4 nodes                SSSSSSSSSSSSSSSSSSSS
 <04_I> with label 4  has 5 nodes             SSSSSSSSSSSSSSSSSSSSSSSSSS
 <10_I> with label 10 has 3 nodes                  NNNNNNNNNNNNNNNNN
 <05_I> with label 5  has 1 nodes                       NNNNNN

 Number of supported vars: 12

  Vars: 12
  Outs: 7
  Size: 52
  SHELL-time : 0.09 sec
-------------------------------------------------------------------------------
Command >
```

Fig. 7. Interactive interface.

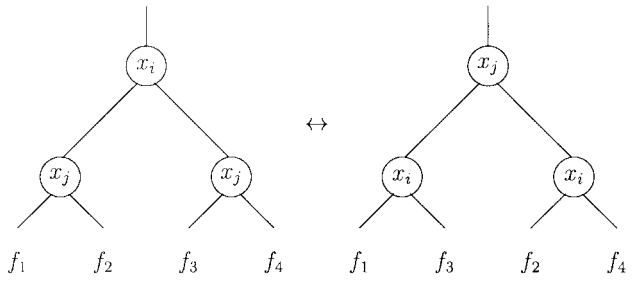

Fig. 8. Exchange of ith and adjacent variable.

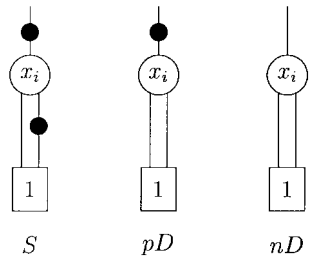

Fig. 9. Different cases for bottom level.

B. Change of Decomposition Type by Reordering

In the following, we discuss a method that is based on reordering: variable x_i is moved to the bottom level of the OKFDD by exchange of neighboring variables. In the bottom level, there exists exactly one node (due to CE's). The different cases for Shannon and positive and negative Davio nodes are shown in Fig. 9 for the function x_i. Then the decomposition type of this single node can easily be changed as follows: the type of the node is changed and the corresponding modifications on the CE's have to be performed. In some cases, an additional depth first search (DFS) run must be used to restore the canonicity of the OKFDD, i.e., the labels at the edges have to be changed. This DFS is needed if a transformation from (or to) negative Davio is performed, since in this case, the incoming edges are affected. The transformation between Shannon and positive Davio, in contrast, is a local operation.

Experiments have shown that this method is superior to the simple method based on XOR-operations of the successors. On average, the new method is two times faster.

C. DTL Sifting

In the last few years, several methods for dynamic variable ordering for OBDD's have been presented and intensively studied [19], [27]. The most promising approach is the *sifting algorithm* [27]. By the sifting algorithm, the variables are sorted into decreasing order based on the number of nodes at each level, and then each variable is traversed through the directed acyclic graph in order to locate its locally optimal position while all other variables remain fixed.

In [15], a first dynamic method has been proposed, but it is infeasible for practical applications since it is too time consuming.

TABLE I
COMPARISON OF SIFTED OKFDD'S AND OBDD'S

name	OKFDD-sifting			OBDD-sifting			interleaving		
	time	size	max	time	size	max	time	size	max
C432	8	1852	10000	4	1331	10000	8	31177	210000
C880	37	8907	20000	23	9118	25000	1	8654	35000
C1355	668	19217	110000	341	37960	155000	2370	40657	180000
C1908	40	5523	20000	49	9411	30000	8	17121	90000
C2670	82	5200	25000	58	6749	25000	1	26303	40000
C3540	647	57300	270000	827	59911	310000	63	153388	1100000
C5315	22	2284	10000	16	2924	20000	5	51777	145000

Based on the methods described in the last two subsections, we use a new method for dynamic variable ordering. (Its practicability will be shown later by experiments in Section V.)

Our method works similar to the sifting algorithm on OBDD's, but for each variable, we try each of the three different decompositions. (The variable is tested at all positions with a fixed decomposition type; then the decomposition type is changed and the variable again is tested at all positions, and so on.) Thus, our method takes approximately three times longer than "normal" sifting.

V. EXPERIMENTAL RESULTS

In this section, we present experimental results for benchmark functions. All experiments were carried out on the package PUMA. All run times are given in CPU seconds on an HP 9000/710 workstation.

First, we compare construction algorithms based on our OKFDD sifting (presented in the last section) with construction using sifting for OBDD's [27] and variable interleaving for OBDD's [18]. The results for some of the benchmarks from [9] are given in Table I. (We list only the circuits for which interleaving could also construct the OBDD.) The names of the corresponding benchmarks are given in the first column. Column *OKFDD sifting* (*OBDD sifting, interleaving*) gives the results for graph construction based on OKFDD sifting (OBDD sifting, variable interleaving). For OKFDD's, the initial DTL consisted of type S only, i.e., we started with a pure OBDD. Davio nodes were introduced by OKFDD sifting. In column *max*, the maximum number of nodes needed during the construction is given. Sifting was performed if the number of nodes became larger than 10 000, and again after each doubling of the graph size. Column *size* denotes the number of nodes needed for the representation of the outputs of the benchmark after the construction. (Notice that we did *not* try to minimize the size by dynamic variable ordering.) Column *time* denotes the overall run time needed for the construction. As can easily be seen, the construction algorithm that makes use of OKFDD sifting never needs more than twice the time of the one using OBDD sifting. For benchmarks C3540 and C1908, the construction based on OKFDD sifting is even faster.

For all considered circuits, the number *max* is smaller or equal for OKFDD sifting, i.e., the graphs remain small during the construction. The increase in size during construction is one major drawback of variable interleaving. Although very fast on average, the resulting graphs are very large, and the method also needs a large number of nodes during the construction.

TABLE II
COMPARISON OF SIFTED OKFDD'S AND OBDD'S

name	in	out	OBDD	OKFDD
C432	36	7	1.2	1.1
C499	41	32	44.8	13.4
C880	60	26	9.1	4.0
C1355	41	32	36.2	13.4
C1908	33	25	12.4	3.8
C2670	233	140	6.6	1.4
C3540	50	22	27.2	22.4
C5315	178	123	3.1	1.3
C7552	207	108	8.2	3.0
s1423	91	79	5.7	1.3
pair	173	137	4.5	1.9
rot	137	107	5.0	3.2
total			164.0	70.2

In a second series of experiments, we applied DTL sifting until no further improvement could be obtained. In this experiment, we only consider the graph sizes to give an impression of the savings that can be obtained by OKFDD's. Our results in comparison to the OBDD results obtained in [27] are given in Table II. (The sizes are given in units of thousand nodes.) Column *in (out)* denotes the number of inputs (outputs) of circuit *name*.

Notice that we used a very simple strategy for our sifting algorithm. In the meantime, more clever heuristics have been developed that, e.g., make use of symmetry [24]–[26]. These ideas can be applied directly to OKFDD's. But already, the simple strategies turn out to be very helpful: The integration of only a few D-nodes in an OBDD can tremendously reduce the size of the representation. For example, the best OBDD size for C1355 known so far is 25 866 nodes [6], [25]. Starting from a nonoptimized OBDD and applying DTL sifting, only one pD- and one nD-level is integrated, and the size is reduced to only 17 577 nodes (see Fig. 10).

Last, we considered technology mapping for FPGA's. In [29], a method for FPGA design using OKFDD's has been presented. We compare our results for larger circuits with the most powerful heuristic from [29] (see Table III). The results from [29] measured in number of nodes are given in column *SPW*. The results obtained by dynamic variable ordering and the execution times are given in the last two columns. As can easily be seen, our approach obtains much better results in all considered cases in negligible time. For some benchmarks, more than 50% improvement is obtained. Obviously, this simplifies the mapping process tremendously.

VI. CONCLUSIONS

In this paper, the efficiency of Kronecker functional decision diagrams as a more compact decision diagram than BDD's or

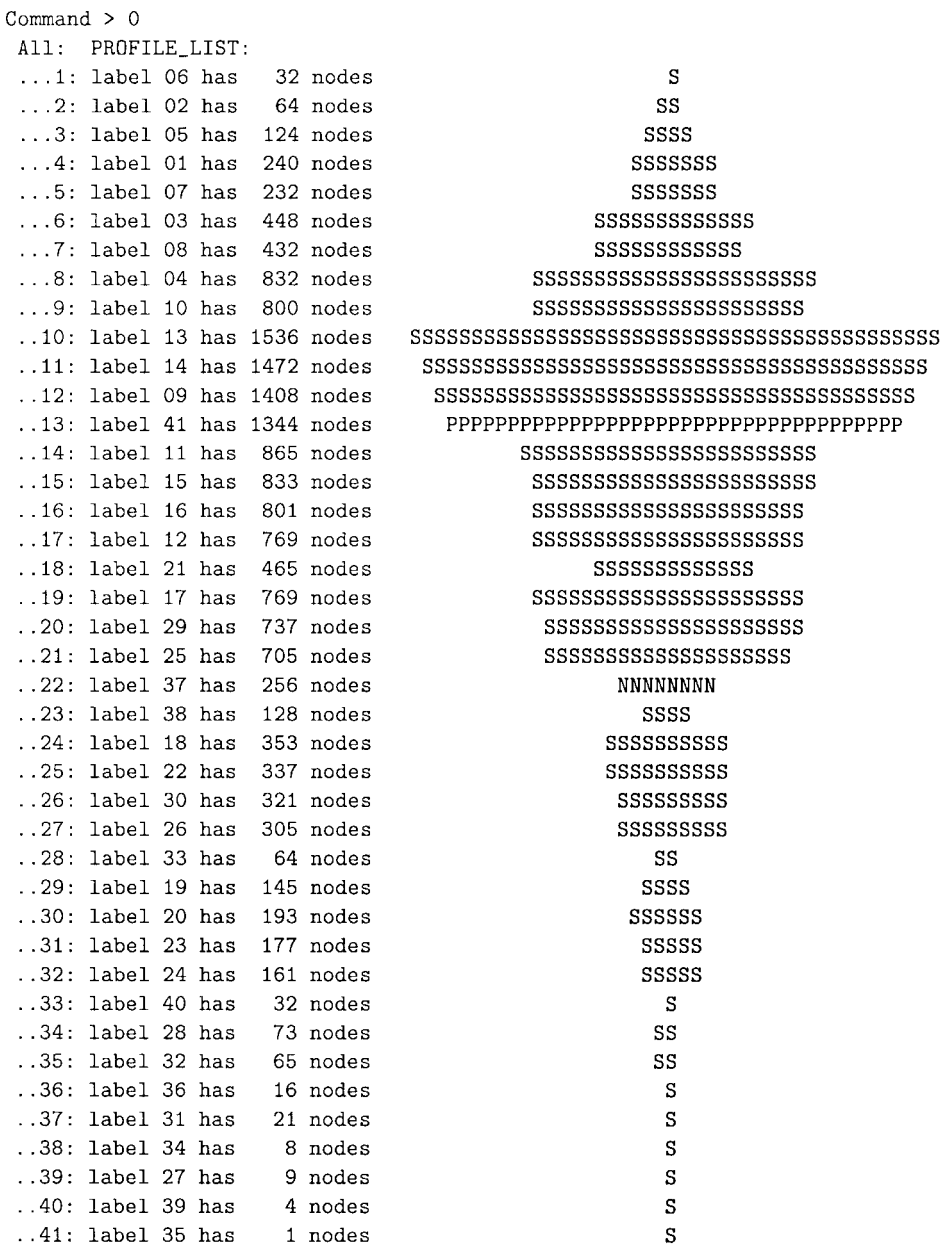

Fig. 10. OKFDD representation for C1355.

TABLE III
RESULTS FOR FPGA DESIGN

name	in	out	SPW	dynamic	time
cc	21	20	117	38	0.1
c8	28	18	237	62	0.2
frg1	28	3	186	77	1.2
pdc	16	40	1118	564	3.5
spla	16	46	724	550	3.2
vg2	25	8	204	76	1.0
total			2586	1367	9.2

FDD's was shown. KFDD's as a generalization of BDD's and FDD's always will be more compact than the two, and this was confirmed through experimental results.

Furthermore, the package PUMA for efficient representation and manipulation of Boolean functions by this data structure was introduced. Efficient algorithms were presented that make fast construction and manipulation of OKFDD's possible. Additionally, the complexity of the algorithms has been analyzed. We investigated dynamic methods for finding good variable orderings and decomposition-type lists of OKFDD's.

The canonicity of the OKFDD's and efficient construction and manipulation techniques presented here make OKFDD's a prime candidate for utilization in applications where OBDD's as a data structure for Boolean functions have been the main construct. Applications in synthesis and verification as well as technology mapping to various FPGA architectures are among those where OKFDD's can be utilized.

ACKNOWLEDGMENT

The authors would like to thank A. Hett and K. Nowak for their help with the realization of the methods discussed in this paper. They also acknowledge the helpful discussions with M. Theobald, A. Sarabi, and M. A. Perkowski.

REFERENCES

[1] B. Becker and R. Drechsler, "How many decomposition types do we need?" in *Proc. Eur. Design & Test Conf.*, Mar. 1995, pp. 438–443.
[2] B. Becker, R. Drechsler, and M. Theobald, "On the implementation of a package for efficient representation and manipulation of functional decision diagrams," in *Proc. IFIP WG 10.5 Workshop Applications of the Reed–Muller Expansion in Circuit Design,*, Sept. 1993, pp. 162–169.
[3] ——, "OKFDD's versus OBDD's and OFDD's," in *Proc. ICALP, LNCS 944*, Apr. 1995, pp. 475–486.
[4] ——, "On the expressive power of OKFDD's," *Formal Methods Syst. Design*, vol. 11, no. 1, pp. 5–21, May 1997.
[5] B. Becker, R. Drechsler, and R. Werchner, "On the relation between BDD's and FDD's," *Inform. Comput.*, vol. 123, no. 2, pp. 185–197, Dec. 1995.
[6] B. Bollig, M. Löbbing, and I. Wegener, "Simulated annealing to improve variable orderings for OBDD's," in *Proc. Int. Workshop Logic Synthesis*, May 1995, pp. 5b:5.1–5.10.
[7] B. Bollig, P. Savicky, and I. Wegener, "On the improvement of variable orderings for OBDD's," in *Proc. IFIP Workshop Logic and Architecture Synthesis*, Grenoble, France, Dec. 1994, pp. 71–80.
[8] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package," in *Proc. Design Automation Conf.*, June 1990, pp. 40–45.
[9] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational circuits and a target translator in Fortran," in *Proc. Int. Symp. Circuits and Systems*, May 1985, pp. 663–698.
[10] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, pp. 677–691, Aug. 1986.
[11] ——, "Symbolic Boolean manipulation with ordered binary decision diagrams," *AC, Comp. Surveys*, vol. 24, pp. 293–318, 1992.
[12] R. Drechsler and B. Becker, "Dynamic minimization of OKFDD's," in *Proc. Int. Conf. Comp. Design*, Oct. 1995, pp. 602–607.
[13] ——, "Sympathy: Fast exact minimization of fixed polarity Reed-Muller expressions for symmetric functions," in *Proc. Eur. Design & Test Conf.*, Mar. 1995, pp. 91–97.
[14] R. Drechsler, B. Becker, and A. Jahnke, "On variable ordering and decomposition type choice in OKFDD's," in *Proc. IFIP Int. Conf. VLSI'95*, Aug. 1995, pp. 805–810.
[15] 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," in *Proc. Design Automation Conf.*, June 1994, pp. 415–419.
[16] R. Drechsler, M. Theobald, and B. Becker, "Fast OFDD based minimization of fixed polarity Reed–Muller expressions," in *Proc. Eur. Design Automation Conf.*, Sept. 1994, pp. 2–7.
[17] S. J. Friedman and K. J. Supowit, "Finding the optimal variable ordering for binary decision diagrams," in *Proc. Design Automation Conf.*, June 1987, pp. 348–356.
[18] H. Fujii, G. Ootomo, and C. Hori, "Interleaving based variable ordering methods for ordered binary decision diagrams," in *Proc. Int. Conf. CAD*, Nov. 1993, pp. 38–41.
[19] M. Fujita, Y. Matsunaga, and T. Kakuda, "On variable ordering of binary decision diagrams for the application of multi-level synthesis," in *Proc. Eur. Conf. Design Automation*, Mar. 1991, pp. 50–54.
[20] P. Ho and M. A. Perkowski, "Free Kronecker decision diagrams and their application to Atmel 6000 FPGA mapping," in *Proc. Eur. Design Automation Conf.*, Sept. 1994, pp. 8–13.
[21] U. Kebschull, E. Schubert, and W. Rosenstiel, "Multilevel logic synthesis based on functional decision diagrams," in *Proc. Eur. Conf. Design Automation*, Mar. 1992, pp. 43–47.
[22] S. Minato, "Zero-suppressed BDD's for set manipulation in combinational problems," in *Proc. Design Automation Conf.*, June 1993, pp. 272–277.
[23] S. Minato, N. Ishiura, and S. Yajima, "Shared binary decision diagrams with attributed edges for efficient Boolean function manipulation," in *Proc. Design Automation Conf.*, June 1990, pp. 52–57.
[24] D. Möller, P. Molitor, and R. Drechsler, "Symmetry based variable ordering for ROBDD's," in *IFIP Workshop Logic and Architecture Synthesis*, Dec. 1994, pp. 47–53.
[25] S. Panda and F. Somenzi, "Who are the variables in your neighborhood," in *Proc. Int. Conf. CAD*, Nov. 1995, pp. 74–77.
[26] S. Panda, F. Somenzi, and B. F. Plessier, "Symmetry detection and dynamic variable ordering of decision diagrams," in *Proc. Int. Conf. CAD*, Nov. 1994, pp. 628–631.
[27] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *Proc. Int. Conf. CAD*, Nov. 1993, pp. 42–47.
[28] T. Sasao, *Logic Synthesis and Optimization.* Norwell, MA: Kluwer Academic, 1993.
[29] I. Schaefer, M. A. Perkowski, and H. Wu, "Multilevel logic synthesis for cellular FPGA's based on orthogonal expansions," in *Proc. IFIP WG 10.5 Workshop Applications of the Reed–Muller Expansion in Circuit Design*, Sept. 1993, pp. 42–51.

Rolf Drechsler (M'94) received the diploma and the Ph.D. degree in computer science from the J. W. Goethe University, Frankfurt am Main, Germany, in 1992 and 1995, respectively.

He currently is with the Institute of Computer Science at the Albert-Ludwigs University, Freiburg im Breisgau, Germany. He recently published two books with Kluwer Academic, one on BDD techniques (coauthored by B. Becker) and one on using evolutionary algorithms for VLSI CAD. His research interests include verification, logic synthesis, and evolutionary algorithms.

Dr. Drechsler is the Symposium's Chair of the IEEE International Symposium on Multiple-Valued Logic 1999 in Freiburg.

Bernd Becker (M'86) received the Dipl.-Math., Dr.rer.nat., and the Dr.habil. degrees from the University of Saarland, Germany, in 1979, 1982, and 1988, respectively.

Between 1979 and 1988, he was with Sonderforschungsbereich "Electronic Speech Recognition" (1979–1981), Institute for Computer Science and Applied Mathematics (1981–1983), and Sonderforschungsbereich "VLSI Design Methods and Parallelism," (1984–1988), all at the University of Saarland. He was an Associate Professor for complexity theory and efficient algorithms at the J. W. Goethe University, Frankfurt am Main, during 1989–1995. Presently, he is with the University of Freiburg im Breisgau as a Full Professor. His research interests include data structures and efficient algorithms (for circuit design), design, test, and verification of VLSI circuits.