On Variable Ordering of Ordered Functional Decision Diagrams

Bernd Becker* Rolf Drechsler†

Michael Theobald†

TR-94-006

Abstract

In this paper methods for finding good variable orderings for ordered functional decision diagrams (OFDDs) are investigated. We present an algorithm for exact minimization of OFDDs that is applicable for functions up to $n = 14$ variables. We present an upper bound for the size of OFDDs representing tree-like circuits. Various methods for dynamic variable ordering based on the exchange of variables are presented. Experimental results are given to show the efficiency of our approaches.

*Fachbereich 20 - Informatik, J.W.Goethe-Universität, D-60054 Frankfurt, and International Computer Science Institute, Berkeley, CA 94707; email: becker@informatik.uni-frankfurt.de

†Fachbereich 20 - Informatik, J.W.Goethe-Universität, D-60054 Frankfurt; email: <name>@informatik.uni-frankfurt.de

This work was supported by DFG grant Be 1176/4-1.

1 Introduction

Graph based data structures are often used in CAD systems for efficient representation and manipulation of Boolean functions. The most popular data structures are ordered binary decision diagrams (OBDDs), that were introduced by Bryant [Bry86] and in the meantime are used for the solution of numerous tasks in design automation [Bry92].
In recent years synthesis based on AND/EXOR realizations gains more and more interst [Ber68, Sau92, BDT93], since AND/EXOR based graphs often allow more succinct representation for classes of Boolean functions than OBDDs [BDW93]. Thus, in many applications Reed-Muller expansions (RMEs) are used.
In this paper we focus on a special class of restricted RMEs, called ordered functional decision diagrams (OFDDs). OFDDs are also applied to problems in logic synthesis, e.g. technology mapping [SKR92] and design for testability [BD93b].
In [BD93a] it has been proven that the size of an OFDD representing a Boolean function largely depends on the chosen variable ordering, i.e. the size of an OFDD varies from linear to exponential.
In this paper we present an algorithm for exact minimization of OFDDs based on the ideas presented in [FS87] and [ISY91], where similar algorithms were applied to OBDDs. We prove that tree-like circuits can be represented efficiently using OFDDs. A method for exchanging adjacent variables (including complemented edges) is introduced that is the basic operation for dynamic variable ordering. Various methods for dynamic variable ordering are presented. We performed experimental results that show that OFDDs are often more succinct than OBDDs. We compare our heuristics with the heuristics presented in [SKR93]. We constructed OFDDs for some of the ISCAS89 [BBK89] and some of the MCNC91 benchmarks. Thus, we show for the first time that OFDDs are also applicable for representation of large circuits.

The paper is structured as follows: In Section 2 OFDDs are defined. The method for exact minimization is presented in Section 3 and heuristical methods based on exchange of adjacent variables are introduced. Experimental results are given in Section 4. We finish with a resume of the results in Section 5.

2 Ordered Functional Decision Diagrams

In this section we define OFDDs. The core of the data structures is a decision diagram (DD), which is a directed acyclic graph with some additional properties. We start with general definitions. Then we introduce further restrictions. Based on these definitions it is possible to generalize the data structure to more powerful ones (see [BD93a]).

Definition 1 A *decision diagram (DD)* over $X_n := \{x_1, x_2, \ldots, x_n\}$ is a rooted directed acyclic graph $G = (V, E)$ with vertex set V containing two types of vertices, *non-terminal* and *terminal* vertices. A non-terminal vertex v is labeled with a variable from X_n, called the *decision variable* for v, and has exactly two successors denoted by $low(v), high(v) \in V$. A terminal vertex v is labeled with 0 or 1 and has no successors.

The size of a DD, denoted with $|DD|$, is given by the number of internal nodes. If DDs are used as a data structure in design automation, further restrictions on the structure of DDs turn out to be necessary.

1. 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.

2. 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.

Since we want to work with small representations of Boolean functions, we define methods to reduce decision diagrams. For this we introduce two reduction types, that can be combined [BDT93].

Type 1: Identify two nodes v, v' in the DD, where the sub-DDs rooted by v, v' are isomorphic.

Type 2: Delete all nodes v whose successor $high(v)$ points to the terminal 0 and connect the incoming edges of the deleted node to the corresponding sucessor.

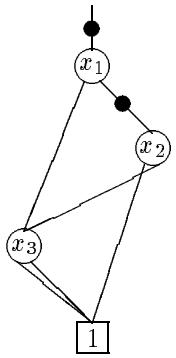

Figure 1: OFDD for $f = x_1x_2 + x_3$

Definition 2 A DD is *reduced,* if no reductions of type 1 and type 2 are applicable to the DD.

A careful analysis of the proofs in [Bry86, SW92, GM92] shows that the following lemma is valid for DDs:

Lemma 1 The reduction of a free DD G is uniquely determined and can be computed in linear time in the size of G.

Until now we have not defined how DDs can be related to Boolean functions. To do this, the following notions are helpful. Let $f : \mathbf{B}^n \rightarrow \mathbf{B}$ be a Boolean function over the variable set X_n. Then f_i^0 denotes the *cofactor* of f with respect to $x_i = 0$, defined by $f_i^0(a) := f(a_1, .., a_{i-1}, 0, a_{i+1}, .., a_n)$ for $a = (a_1, a_2, \ldots, a_n) \in \mathbf{B}^n$. Analogously, f_i^1 denotes the cofactor for $x_i = 1$. Finally, we define f_i^2 by $f_i^2 := f_i^0 \oplus f_i^1$. (Notice, that the three functions f_i^0, f_i^1, f_i^2 can naturally be interpreted as Boolean functions from $\mathbf{B}^{n-1}$ to $\mathbf{B}$ defined over the variables $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$.) Using the definitions above it is an easy exercise to prove the following decomposition equation for an arbitrary Boolean function f:

$$f \quad = \quad f_i^0 \oplus x_i f_i^2 \tag{1}$$

OFDDs are constructed by application of equation (1). More formally we obtain:

Definition 3 Each ordered DD over X_n is an *OFDD over* X_n. Nothing else is an OFDD. The function $f_G : \mathbf{B}^n \rightarrow \mathbf{B}$ represented by an OFDD G over X_n is defined as follows:

1. If G consists of a single node labeled with 0 (1), then G is an OFDD for $f = 0$ ($f = 1$).
2. If G has a root v with label x_i, then G is an OFDD for $f_{low(v)} \oplus x_i f_{high(v)}$, where $f_{low(v)}$ ($f_{high(v)}$) are the functions represented by the OFDD rooted at $low(v)$ ($high(v)$).

It follows that an arbitrary Boolean function can be represented by an OFDD and vice versa that each ordered DD defines an OFDD computing a Boolean function. From Lemma 1 and Definition 3 we conclude:

Lemma 2 Reduced OFDDs are a canonical representation for Boolean functions.

The size of an OFDD can be further reduced, if complemented edges (CEs) are used [Ake78, BRB90, BDT93]. Then a node represents a function and the complement of the function at the same time. The representation is further canonical. Experiments show that CEs reduce the OFDD-size by 10 % on average.

Example 1 In Figure 1 an OFDD for the function $f = x_1x_2 + x_3$ is shown. A dot on an edge symbolizes a CE.

3 Ordering Methods

The size of an OFDD largely depends on the chosen variable ordering [BD93a], i.e. the size of an OFDD varies from linear to exponential. In this section ordering methods are presented.

3.1 Exact Minimization

We start with an algorithm for determining the ordering minimizing the overall size of an OFDD. The algorithms presented in [FS87] and [ISY91] can directly be applied, since reduced OFDDs fulfill all needed prerequisites, i.e. the number of nodes at level i are constant if the ordering of the variables in the upper and the lower part are changed.

But the corresponding algorithm has exponential worst case behavior. Our experiments have shown (see Section 4) that it is only applicable for OFDDs with up to $n = 14$ variables. For OFDDs representing Boolean functions depending on more variables heuristic methods are needed.

3.2 OFDDs for Tree-like Circuits

Most often OFDDs are constructed from the circuit description of a Boolean function. It is well-known that tree-like circuits have OBDD size $O(n)$, when n denotes the number of inputs and the ordering from the description is used. Some heuristics use this fact for finding a good ordering for benchmark circuits. In the following we show that tree-like circuits can also be represented efficiently by OFDDs.

In this subsection we use OFDDs without CEs for simplicity of the proof. Obviously, OFDDs with and without CEs differ in size at most by a factor of 2.

For tree-like circuits defined on basic gates with two inputs (i.e. AND, OR, XOR, NAND, NOR, EQUIV) the following lemma holds:

Lemma 3 Each tree-like circuit SK with n inputs can be represented by an OFDD of size

$$\leq n \cdot depth(SK).$$

Proof: Due to page limitation only a sketch of the proof is given:

Let F and G be two OFDDs with disjoint support for the functions f and g, respectively. We have to consider the basic operations and have to prove that the synthesis can not increase the number of nodes too much. We give the general principle how to do this and apply this principle to AND and XOR.

Assume that $*$ is any of the binary operations given above and that v, labeled with x_i (the first variable of F) is the root of the OFDD for $f * g$. Then we recursively compute the OFDDs rooted at $low(v)$ and $high(v)$. These are OFDDs for the functions $f_0 * g$ and $(f_0 * g) \oplus (f_1 * g)$. (Notice that g is independent of x_i!)

If we do this for the AND operation, we obtain $(f \cdot g)_0 = f_0 \cdot g$ and $(f \cdot g)_2 = (f_0 \cdot g) \oplus (f_1 \cdot g) = f_2 \cdot g$, respectively. We conclude, that the OFDD for $f \cdot g$ can be constructed as follows: We only have to redirect the incoming edges of the terminal 1-node from the OFDD F to the root node of G.

For the XOR operation it holds $(f \oplus g)_0 = f_0 \oplus g$ and $(f \oplus g)_2 = (f_0 \oplus g) \oplus (f_1 \oplus g) = f_2$. Thus, we obtain an OFDD for $f \oplus g$ in the following way: If 1 is an element of the 2-level RME of f (Case 1), then the path in F corresponding to this element (1-path) has to be isolated and an OFDD for $\overline{g}$ has to be rooted at the endpoint of this path. If 1 is not an element of the 2-level RME of f (Case 2), again the 1-path in F has to be isolated and an OFDD for g has to be rooted at the endpoint of this path. It is easy to see that isolation of a path costs at most k additional nodes, if k is the number of different variables. Computation of $\overline{g} = 1 \oplus g$ can be done by adding the 1-path (if non-existent in G) or deleting the 1-path (if existent in G). This again has the same costs as isolation of a path. Similarly OR can be realized.

We obtain the cost for the realization of the negated operations by observing that negation, as pointed out before, can be done by adding or deleting a path. If negation is performed at a node of level i in the circuit, we conclude that at most 2^{i-1} nodes (one for each of $\leq 2^{i-1}$ variables) have to be added. Since there are at most $n/2^i$ nodes in a level, there can be added at most $n/2$ nodes per level because of negation. This finally results in the factor of $depth(SK)$. □

Often the representation is much better, but in the case of a balanced NAND-tree the given bound is asymptotically optimal. An analogous lemma is valid for operations with more than two inputs. In this case we need $\leq 2 \cdot n \cdot depth(SK)$ nodes, but we skip the proof for shortness of the paper.

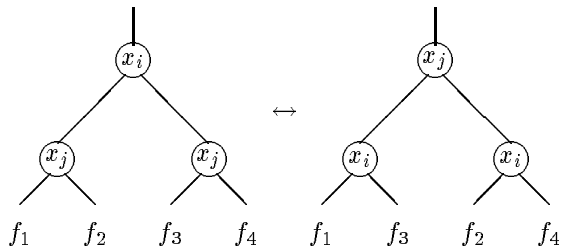

Figure 2: Exchange of i-th and adjacent variable

```
exchange_var_fdd(F, m) {
 do{
    recompute number of saved nodes for needed
      permutations of m variables
    perform best exchange of m variables;
 } while (optimization possible);
}
```

Figure 3: Algorithm for exchange of variables

3.3 Dynamic Variable Ordering

For general circuits such polynomial upper bounds can not be given [BDW93]. In the following we present methods that are based on the exchange of adjacent variables. (Similar methods have been applied to OBDDs in [FMK91, ISY91, Rud93].)

The size is optimized without a complete reconstruction of the OFDD. Only local transformations for the two variables are performed. This is due to the observation that OFDDs are a canonical form. Thus, the exchange of variable i and $i+1$ does not change the sub-OFDDs corresponding to the variables from 1st to $(i-1)$-th and from $(i+2)$-th to the n-th position.

The general case for the exchange of variable i and an adjacent variable j is shown in Figure 2. All other cases can be reduced to this one. The exchange is performed very fast in our OFDD-package [BDT93], since only edges must be redirected. In contrast to the method presented in [FMK91] our approach also uses CEs.

One disadvantage of the method presented above is that it might easily be captured by local minima. But this method can be extended to the exchange of adjacent m variables ($m > 2$). Then, it is more time consuming but leads to better results, analogously to OBDDs [ISY91].

For this we construct a cyclic exchange of variables based on transpositions that visits all permutations of m variables. For $m = 3$ the sequence can be chosen as

$$abc \rightarrow bac \rightarrow bca \rightarrow cba \rightarrow cab \rightarrow acb.$$

This can directly be extented to arbitrary m [RND77]. The corresponding algorithm is shown in Figure 3. The experimental results in Section 4 show the results of this method for increasing m. This method is also called *window permutation* [Rud93].

Finally, we present the *sifting algorithm* [Rud93] for OFDDs. There one variable is exchanged while all others are left stable. The variable is positioned at the location minimizing the overall size of the OFDD. This procedure, that also uses the exchange of adjacent variables, is applied to all variables.

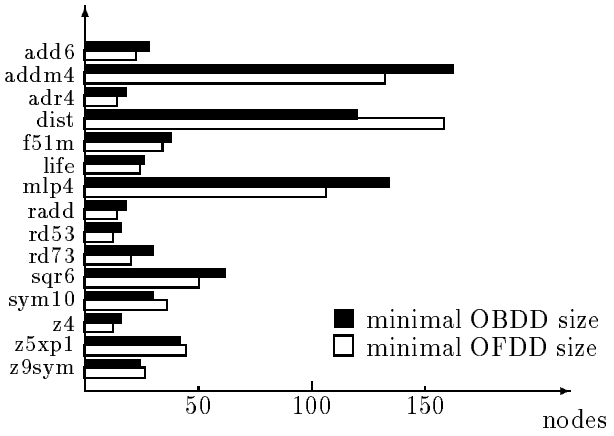

Figure 4: Minimal OFDD size vs. minimal OBDD size

4 Experimental Results

In this section we present experimental results concerning the algorithms presented in Section 3.

4.1 Exact Minimization

We performed exact minimization of arithmetical benchmark circuits [BHMS84], since AND/EXOR based data structures are especially applicable to these types of circuits. The results are presented in Figure 4, where the minimal OFDDs are also compared with the minimal OBDDs. It turns out that for 11 of the considered 15 benchmarks OFDDs allow more succinct representation.

4.2 Dynamic Variable Ordering

In this subsection we present the results obtained by dynamic variable ordering of OFDDs, i.e. for window permutation with increasing m and the sifting algorithm. The results including a comparison with the optimal OFDD size $size_{min}$ for the arithmetical benchmarks are given in Figure 5.
Since sifting guaranteed the best results we also applied it to larger examples, i.e. ISCAS89 benchmarks [BBK89]. We started with the initial ordering and a node limit of 70000 nodes. The dynamic reordering was performed when no more nodes were available. The results in comparison to OBDDs that are also constructed using sifting [Rud93] are presented in Figure 6.
Finally, we compared the results obtained by our sifting method with the heuristics presented in [SKR93] for some of the MCNC91 benchmarks. We compared sifting with the best results obtained in [SKR93], where an other reduction method was used. The corresponding OFDD-sizes are shown in Figure 7.

5 Conclusions

In this paper ordering methods for OFDDs have been presented. We succeeded in minimizing OFDDs for up to $n = 14$ variables. For OFDDs representing tree-like circuits we proved an upper bound for the number of nodes. For general circuits we applied dynamic variable ordering and thus were able to construct OFDDs for large benchmark circuits. We presented experimental results and showed that our approach is much more succinct than previously published methods.
Since OFDDs are more succinct than OBDDs at least for some classes of functions they might partially be an alternative data structure. It is also possible to mix OBDDs and OFDDs level by level. This hybrid approach is currently implemented and first results can be found in [DST+93].

name	$size_{min}$	$m = 2$	$m = 3$	$m = 4$	*sift*
add6	23	58	59	57	45
addm4	132	133	132	132	132
adr4	15	15	15	15	15
dist	159	172	172	171	159
f51m	35	35	35	35	35
life	24	38	25	25	25
mlp4	107	107	107	107	107
radd	15	27	27	26	20
rd53	13	13	13	13	13
rd73	21	21	21	21	21
sqr6	50	50	50	50	50
sym10	36	36	36	36	36
z4	13	13	13	13	13
z5xp1	45	45	45	45	45
z9sym	26	26	26	26	26

Figure 5: OFDD size for dynamic variable ordering

name	*OBDD*	*OFDD*
s27	9	9
s298	86	71
s344	103	115
s349	103	115
s382	121	122
s1196	631	734
s1423	2725	2044
s1488	385	386
s1494	385	386

Figure 6: OFDD-size for ISCAS benchmarks

name	*[SKR93]*	*sift*
b12	292	61
cordic	233	41
cps	4922	1293
ex5	901	272
rd53	34	13
rd73	64	21
rd84	86	29
sao2	154	98
t481	54	20
vg2	1912	245
Z5xp1	54	45
Z9sym	40	20

Figure 7: Comparison with [SKR93]

References

[Ake78] S.B. Akers. Binary decision diagrams. *IEEE Trans. on Comp.*, C-27:509–516, 1978.

[BBK89] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Int'l Symp. Circ. and Systems*, pages 1929–1934, 1989.

[BD93a] B. Becker and R. Drechsler. On the computational power of functional decision diagrams. *20. Workshop über Komplexitätstheorie, Datenstrukturen und effiziente Algorithmen, Berlin*, 1993.

[BD93b] B. Becker and R. Drechsler. Testability of circuits derived from functional decision diagrams. Technical report, Universität Frankfurt, 13/93, Fachbereich Informatik, 1993.

[BDT93] B. Becker, R. Drechsler, and M. Theobald. On the implementation of a package for efficient representation and manipulation of functional decision diagrams. *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, Hamburg*, 1993.

[BDW93] B. Becker, R. Drechsler, and R. Werchner. On the relation between bdds and fdds. Technical report, Universität Frankfurt, 12/93, Fachbereich Informatik, 1993.

[Ber68] E. R. Berlekamp. *Algebraic Coding Theory.* McGraw-Hill Book Company, 1968.

[BHMS84] R.K. Brayton, G.D. Hachtel, C. McMullen, and A.L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis.* Cluwer Academic Publishers, 1984.

[BRB90] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

[Bry86] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 8:677–691, 1986.

[Bry92] R.E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM, Comp. Surveys*, 24:293–318, 1992.

[DST+93] 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. Technical report, J.W.Goethe-University, Frankfurt 14/93, October, 1993.

[FMK91] M. Fujita, Y. Matsunga, and T. Kakuda. On variable ordering of binary decision diagrams for the application of multi-level synthesis. In *European Conf. on Design Automation*, pages 50–54, 1991.

[FS87] Steven J. Friedman and Kenneth J. Supowit. Finding the optimal variable ordering for binary decision diagrams. In *Design Automation Conf.*, pages 348–356, 1987.

[GM92] J. Gergov and C. Meinel. Efficient analysis and manipulation of obdds can be extended to read-once-only branching programs. In *WG'92, LNCS*, 1992.

[ISY91] N. Ishiura, H. Sawada, and S. Yajima. Minimization of binary decision diagrams based on exchange of variables. In *Proceedings of ICCAD*, pages 472–475, 1991.

[RND77] E.M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms - Theory and Practice.* Prentice-Hall, Inc., 1977.

[Rud93] Richard Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Workshop on Logic Synth.*, pages 3a:1–12, 1993.

[Sau92] J. Saul. Logic synthesis for arithmetic circuits using the reed-muller representation. In *Proceedings of EDAC'92*, pages 109–113, 1992.

[SKR92] E. Schubert, U. Kebschull, and W. Rosenstiel. FDD based technology mapping for FPGA. In *Proceedings of EUROASIC*, pages 14–18, 1992.

[SKR93] E. Schubert, U. Kebschull, and W. Rosenstiel. Some optimizations for functional decision diagrams. *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, Hamburg*, 1993.

[SW92] D. Sieling and I. Wegener. Reduction of BDDs in linear time. Technical report, Universität Dortmund, 1992.