published in
Proc. of the 1999 IEEE/ACM Int. Workshop on Logic Synthesis (IWLS99), Lake Tahoe, CA, 1999, pp.71-74 and
Proc. of 4th Int. Workshop on Applications of the Reed-Muller Expansion in Circuit Design (Reed-Muller'99), Victoria, B.C., Canada, 1999, pp. 179-184.

Algorithmic Considerations for ⊕-OBDD Reordering

Christoph Meinel, Harald Sack
FB IV - Informatik, Universität Trier
D-54286 Trier, Germany
email: {meinel,sack}@uni-trier.de

Abstract

Ordered Binary Decision Diagrams (OBDDs) have already proved useful in the process of electronic design automation. Due to limitations of the descriptive power of OBDDs more general models of Binary Decision Diagrams have been studied. In this paper, ⊕-OBDDs and algorithms for improving their efficiency are addressed. Furthermore, some problems related to the reordering of variables and functional nodes within ⊕-OBDDs will be discussed.

1 Introduction

A major problem in the computer aided design of digital circuits is to choose a suitable representation of the circuit functionality for the computer's internal use. A concise representation which simultaneously provides fast manipulation is very important for problems in form of Boolean functions. During the last years, Ordered Binary Decision Diagrams (OBDDs) have proved to be well qualified for this purpose (for an overview see [Bry92, MT98]).

Applications based on OBDDs have to respect certain limitations, since the descriptive power of OBDDs is limited. For this reason, not every Boolean function of practical importance can be represented efficiently. For example, the OBDD-representations of the *multiplication* is always of exponential OBDD-size [Bry91]. Therefore, more general BDD models have been studied.

In this paper we address ⊕-OBDDs (also known as Mod2-OBDDs), introduced as an extension of OBDDs [GM96]. ⊕-OBDDs are more, sometimes even exponentially more, space-efficient than OBDDs. They preserve the OBDD property of being an efficient data structure for Boolean function manipulation: Important operations as *apply, quantification*, and *composition* have the same complexity as in the case of OBDDs. Even better, the Boolean functions *exclusive or* (EXOR), *logical equivalence* (EQU), and *negation* can be performed in constant time.

However, ⊕-OBDDs do not provide a canonical representation of Boolean functions and therefore, checking the equivalence of two ⊕-OBDDs becomes an essential operation. A deterministic equivalence test is too time consuming for practical applications. Therefore, all synthesis operations are based on the use of Boolean signatures for identification of the unique Boolean functions represented by the ⊕-OBDD and equivalence is probabilistically determined.

We try to improve ⊕-OBDD efficiency by reducing ⊕-OBDD size. To achieve this, we apply heuristics based on the reordering of variables and functional nodes within the given graph. As an operation of only local effect, reordering of variables or functional nodes can be implemented very efficiently. We show which problems arise during reordering of ⊕-OBDDs in difference to OBDDs and propose some possible solutions. Finally, we give a sketch of the reordering algorithm.

The paper is structured as follows: In Section 2, we recall basic definitions concerning ⊕-OBDDs. Section 3 covers the reordering of variables and functional nodes for ⊕-OBDDs in general. Section 4 discusses problems related to reordering and its implementation and gives an outline of the algorithm. Section 5 concludes with an outlook of work to be done.

2 ⊕-OBDDs

Definition of the Data Structure

A ⊕-OBDD P over a set $X_n = \{x_1, \ldots, x_n\}$ of Boolean variables is a directed acyclic connected graph $P = (V, E)$. V is the set of nodes, consisting of nonterminal nodes with out-degree 2 and of terminal nodes with out-degree 0. There is a distinguished nonterminal node, the *root*, which, as only node, has the in-degree 0. To deal with Boolean functions $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$, we consider *multi rooted shared* $\oplus$-OBDDs by introducing multiple roots into a single $\oplus$-OBDD, each root representing a subfunction of $f=(f_1,\ldots,f_m)$, $f_i : \mathbb{B}^n \rightarrow \mathbb{B}$. The two terminal nodes with no outgoing arcs are labeled by the Boolean constants 0 and 1. The remaining nodes are either labeled with Boolean variables $x_i \in X_n$ (*branching nodes*), or with the binary Boolean function $\oplus$ (EXOR) (*$\oplus$-nodes, functional nodes*). On each path, every variable is permitted to occur at most once. In the following, let $l(v)$ denote the label of the node $v \in V$.

$E \subseteq V \times V$ denotes the set of edges. The two edges starting in a branching node v are labeled with 0 and 1. The *0(1)-successor* of node v is denoted by $v_0(v_1)$. There is a permutation σ on the variable indices which defines an order $x_{\sigma(1)} < x_{\sigma(2)} < \ldots < x_{\sigma(n)}$ on the set of input variables. If w is a successor of v in P with $l(v), l(w) \in X_n$, then $l(v) < l(w)$ according to σ.

The function f_P associated with the $\oplus$-OBDD P is determined in the following way: For a given input assignment $a=(a_1,\ldots,a_n) \in \{0,1\}^n$, the function f_P associated with the $\oplus$-OBDD P is determined by extending the Boolean values assigned to the leaves to all other nodes of P as follows:

- Let v_0 and v_1 be the successors of v, carrying the Boolean values $\delta_0, \delta_1 \in \{0,1\}$.
- If v is a branching node, $l(v)=x_i \in X_n$, then v is associated with δ_{a_i}.
- If v is a $\oplus$-node, then v is associated with $\oplus(\delta_0,\delta_1)=(\delta_0+\delta_1) \bmod 2$.

The function $f_P(a)$ takes the value associated with the source of P. To achieve a more compact representation, we may furthermore consider the use of complemented edges as introduced in [MB88].

But, $\oplus$-OBDDs do not provide a canonical representation of Boolean functions, because for a unique Boolean function f there are different $\oplus$-OBDDs that are representing f but having different $\oplus$-node placement.

Probabilistic Equivalence Test

Since a deterministic equivalence test for $\oplus$-OBDDs requires runtime at least cubic in the number of nodes [Waa97], for practical applications we have to choose a faster method. The probabilistic equivalence test for $\oplus$-OBDDs proposed in [GM93] needs only linear many arithmetic operations in the number of variables. It is based on a probabilistic equivalence test for *read-once branching programs* (BP1), originally introduced in [BCW80]. Equivalence of two $\oplus$-OBDDs is determined by an algebraic transformation of the $\oplus$-OBDDs in terms of polynomials over a finite field. So, the problem of deciding the equivalence of two $\oplus$-OBDDs reduces to the decision of the equivalence of two multivariate polynomials that is solved probabilistically. For a more detailed description see [MS98].

Reduction and Synthesis of $\oplus$-OBDDs

The reduction rules that, if exhaustively applied, guarantee a canonical representation for OBDDs can be extended for $\oplus$-OBDDs. In this case, these rules serve only for a reduction in size and do not provide any canonicity. Additionally to the OBDD reduction rules, rules for the treatment of $\oplus$-nodes have to be considered. For a detailed description of reduction rules for $\oplus$-OBDDs see also [MS98].

For describing the $\oplus$-OBDD synthesis algorithm we will assume that the reader is familiar with standard OBDD synthesis algorithms. Like for convenient OBDD applications we use the *ite*-algorithm [BRB90] and extend it for our purposes. As a first extension, we directly create a $\oplus$-node if a Boolean EXOR has to be computed and as a second difference, the creation of the cofactor $f|_{x_i=d}, d \in \{0,1\}$ (x_i being source node) has to be adapted for $\oplus$-OBDDs. *Regular cofactors* $f|_{x_i}$, i.e., cofactors of a function associated with a branching node v labeled by the variable x_i, $(l(v)=x_i)$, are computed by simply returning the *d-successor* of node v. Creating the cofactors of a function associated with a $\oplus$-node v according to a variable x_i sometimes requires the allocation of a new $\oplus$-node connected to the cofactors of the left and right successor of v. In the worst case we have to create new $\oplus$-nodes for every $\oplus$-node on a path between v and the branching node v_B labeled by the variable x_i (see Figure 1).

To speed up the performance of the *ite* operation the usual memory efficiency techniques like caches and hash tables can be adapted. In the case that the circuit under consideration does not contain any EXOR(EQU) gate, we have to introduce $\oplus$-nodes in a different way. To do this, we may use alternative functional decompositions applied to the

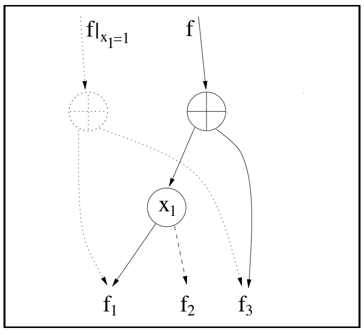


Figure 1: *Cofactor creation $f|_{x_1=1}$ in ⊕-OBDD P with $l(source_P) = \oplus$.*

Boolean function to be represented. Normally, functional decomposition for OBDDs is done by using the *Boole/Shannon* decomposition (BS):

$$\text{BS: } f = xf|_{x=1} + \overline{x}f|_{x=0}$$

By using positive or negative Davio expansion (pDE/nDE) we can directly make use of the EXOR operator by translating it to a ⊕-node.

$$\begin{aligned}\text{pDE: } f &= f|_{x=0} \oplus x(f|_{x=1} \oplus f|_{x=0})\\ \text{nDE: } f &= f|_{x=1} \oplus \overline{x}(f|_{x=1} \oplus f|_{x=0}).\end{aligned}$$

Thus, we are able to introduce additional ⊕-nodes for taking advantage of the potential of the ⊕-OBDD data structure.

3 Reordering ⊕-OBDDs

The size of a Boolean function represented as an OBDD heavily depends on the chosen variable order, since its size may vary exponentially with different orderings. Finding an optimal variable order or even improving an existing one is known to be NP-hard [BW96]. Therefore, heuristical optimization techniques based on local changes in the variable order are used for improvement. Exhaustive search techniques are only feasible for Boolean functions with a small number of variables and do almost have no impact on practical work [DDG98].

The same arguments hold for ⊕-OBDDs, but additionally we have to consider where to place the ⊕-nodes within the graph. By ongoing exchange of a ⊕-node with its adjacent neighbor the number of ⊕-nodes may double in each single exchange. Now, we have two problems to solve: finding a well suited variable order and determining the best positions for ⊕-nodes in the graph.

As in the case of OBDDs the exchange of adjacent nodes in ⊕-OBDDs is a local operation and therefore can be computed very efficiently. But, to exchange two variables that are adjacent in the variable order we have to consider ⊕-nodes that may be placed between the two variables on some paths in the graph.

In this situation there are two possible ways to proceed: We may perform the variable exchange around the ⊕-nodes. In case of a single ⊕-node the operation is simple, but if many cascading ⊕-nodes are involved the situation gets very complicated to handle.

On the other hand, we may first move the ⊕-nodes that are placed between the two variables to positions above or below of them. After that, we are able to perform a variable exchange like in the case of OBDDs.

In the following, we concentrate on the second possibility, because in general it is much easier to implement. The exchange of a ⊕- and a branching node works similar to the node exchange of branching nodes and can be made clear using the formula representation of the Boolean function shown in Figure 2.

$$\begin{aligned}f &= (xg_x + \overline{x}g_{\overline{x}}) \oplus (xh_x + \overline{x}h_{\overline{x}})\\ &= (xg_x \oplus \overline{x}g_{\overline{x}}) \oplus (xh_x \oplus \overline{x}h_{\overline{x}})\\ &= x(g_x \oplus h_x) + \overline{x}(g_{\overline{x}} \oplus h_{\overline{x}})\end{aligned}$$

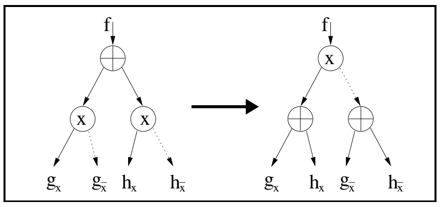


Figure 2: *Exchange of Branching Nodes and Functional Nodes*

By using this strategy we are able to adapt optimization techniques for OBDDs that are based on local exchanges of variables to ⊕-OBDDs.

But, reordering of ⊕-OBDDs can only work efficiently if we take care of some problems related to the implementation of this approach.

4 Algorithmic Considerations for $\oplus$-OBDD Reordering

For $\oplus$-OBDD manipulation and synthesis the computation of the cofactor is an important and frequently performed operation. For $\oplus$-OBDDs the creation of the cofactor according to the branching node at the top is a more complex operation compared to OBDDs. Therefore, this operation should be implemented as efficiently as possible. If the top node of the $\oplus$-OBDD is a branching node we simply return the left or right successor as a cofactor in one step. For a $\oplus$-node at the top we have to determine the cofactor of its left and right successor. To accelerate the decision, whether the function under consideration does depend on the variable the cofactor is computed for, we store the variable index of the successor branching node with the smallest index according to the given variable order σ within the $\oplus$-node.

Storing this index results also in other advantages. Normally, in BDD packages for each variable index a hash table is maintained that is used for a fast node identification. We place the $\oplus$-nodes in the hash table according to the variable index we have computed for them. In this way we can easily address specific $\oplus$-nodes being somewhere between two adjacent variables that we want to exchange.

$\oplus$-nodes may occur in cascades or trees between two adjacent variables. To exchange these variables, we have to move all $\oplus$-nodes between them up- or downwards out of the way. This case is much more easier to handle if we take so called *meta-nodes* under consideration, i.e., $\oplus$-nodes with more than two successors. We do not have to change the underlying data structure, because we may store a pointer to an array of successors in the memory that is normally used for the two successor pointers. See Figure 3 for the node exchange using $\oplus$-meta-nodes.

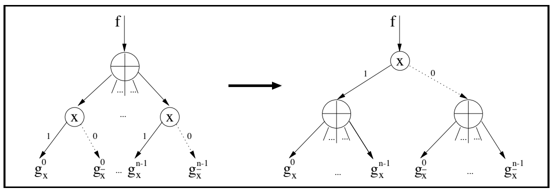
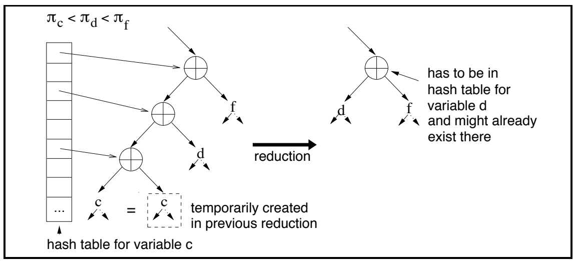


Figure 3: *Exchange of Branching Nodes and Functional $\oplus$-Meta-Nodes*

The use of $\oplus$-meta-nodes provides also other advantages: The identification of possible reductions becomes much easier. If we look at a $\oplus$-meta-node with n successors, this corresponds to a tree of (n–1) binary $\oplus$-nodes. In a structure like this it takes too much time to compare all direct successors of the lowest level of $\oplus$-nodes for equivalences. During the construction of the successor list of the $\oplus$-meta-node we directly identify equivalences while adding a new edge and perform the appropriate reduction.

Another but in our opinion the most important problem that can be solved by using meta-nodes is the occurrence of possible non local operations during the node exchange. Suppose the following situation with binary $\oplus$-nodes: After performing an exchange of $\oplus$- and branching nodes in one level before all reduction steps have been computed we might meet the graph as shown in Figure 4.



$\pi_c < \pi_d < \pi_f$
f
d
c
=
c
...
temporarily created in previous reduction
hash table for variable c
reduction
has to be in hash table for variable d and might already exist there

Figure 4: *Possible Non Local Operations during Node Exchange with Binary $\oplus$-nodes*

At the beginning, each $\oplus$-node is located in the hash table of variable c. After all reduction steps have been performed we end up in a $\oplus$-node connected to two $\oplus$-OBDDs with the top variables d and f. According to our model the $\oplus$-node has to be transfered to hash table d (if $\pi_d < \pi_f$). What, if this node does already exist in hash table d? Then, all fan-ins of the original node have to be redirected and thus, the operation becomes non local, i.e., we have to look at the successors of all nodes in every layer above the current node.

If we are using $\oplus$-meta-nodes, we can avoid this situation, because the reduction already takes place during creation of the $\oplus$-meta-node. Before the node is put into the hash table we can check for its existence. If it is already in the hash table, we connect this node to the predecessor node that is now, after the exchange, labeled by the variable (see Figure 3). This predecessor node is the only node that is affected by this operation. For an upward exchange of a $\oplus$-meta-node this situation cannot occur, since a reduction where all variables c will be eliminated is not possible. Otherwise, the original function would have been independent of variable c, i.e., there would have been no branching node labeled with variable c.

Finally, we want to adapt the well known sifting heuristic proposed by [Rud93] to $\oplus$-OBDDs. To perform reordering with meta-nodes we have two possibilities: First, we can use the regular synthesis algorithm for binary $\oplus$-OBDDs and before reordering we sum up all trees of $\oplus$-nodes to $\oplus$-meta-nodes. Otherwise, as a second choice, we can directly perform synthesis with $\oplus$-meta-nodes. For this approach all synthesis and reduction algorithms have to be adapted, but in our opinion this is the more efficient approach, esp. because it will be much easier to identify reduction conditions during the synthesis.

The regular sifting algorithm works in the following way: For each single variable consecutively the optimal position in the variable order is chosen by an ongoing exchange of adjacent variables. In this process each variable is moved up and down in the variable order and the position that produces the smallest OBDD size is maintained. The only difference for $\oplus$-OBDDs is that we move the $\oplus$-nodes between these variables out of the way before we perform a variable exchange. See Figure 5 for an outline of the variable exchange algorithm in pseudo code.

```
swap_varlable $(i, i+1)$ {
    step through hash table $(i+1)$ {
        If node ls $\oplus$-meta-node
            move $\oplus$-meta-node to level $(i+2)$
    }
    step through hash table $(i)$
        perform regular varlable exchange $(i, i+1)$
}
```

Figure 5: *Algorithm for Variable Exchange*

For the variable exchange we have to scan both variable levels i and $(i{+}1)$. First, we have to remove all the $\oplus$-meta-nodes of level $(i{+}1)$. W.l.o.g. we move the $\oplus$-meta-nodes downwards. After that, we perform the regular variable exchange of level i and $(i{+}1)$. Like for the variable exchange, the $\oplus$-OBDD-size for moving a binary $\oplus$-node above or below a branching node is bounded by $|G|/2 \leq |G'| \leq 2|G|$, for G denoting a $\oplus$-OBDD before and G' after the exchange [BLW96]. If we consider the space complexity of moving $\oplus$-meta-nodes for denoting the size of the $\oplus$-OBDD we do not only have to take the number of nodes into account, but also have to consider the number of edges. All in all, the space complexity will be the same because of the equivalence of the two models.

5 Conclusions

An implementation of the reordering algorithm in connection with an adapted version of the $\oplus$-OBDD synthesis is subject of currently ongoing research. Besides reordering the variables, the development of heuristics for a good placement of $\oplus$-nodes is very promising. Suppose, the $\oplus$-nodes have a quasi optimal position in the $\oplus$-OBDD. Then, we could limit the variable reordering to sections of the order were no $\oplus$-node is in between the variables. In this way, the $\oplus$-nodes are serving as logical bounds for a block restricted sifting heuristic [MS97].

References

[BCW80] M. Blum, A. K. Chandra, and M. N. Wegman. Equivalence of Free Boolean Graphs Can be Decided Probabilistically in Polynomial Time. *Information Processing Letters*, 10(2):80–82, 1980.

[BLW96] B. Bollig, M. Löbbing, and I. Wegener. On the Effect of Local Changes in the Variable Ordering of Ordered Decision Diagrams. *Information Processing Letters*, (59):233–239, 1996.

[BRB90] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient Implementation of a BDD Package. In *27th ACM/IEEE Design Automation Conference*, pages 40–45, 1990.

[Bry91] R. E. Bryant. On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication. *IEEE Transactions on Computers*, 40(2):205–213, 1991.

[Bry92] R. E. Bryant. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[BW96] B. Bollig and I. Wegener. Improving the Variable Ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, (45):993–1002, 1996.

[DDG98] R. Drechsler, N. Drechsler, and W. Günther. Fast Exact Minimization of BDDs. In *Proc. of the 35th ACM/IEEE Design Automation Conference*, pages 200–205, 1998.

[GM93] J. Gergov and Ch. Meinel. Frontiers of Feasible and Probabilistic Feasible Boolean Manipulation with Branching Programs. In *Proc. of the 10th annual Symposium on Theoretical Aspects of Computer Science*, volume 665 of *LNCS*, pages 576–585. Springer, 1993.

[GM96] J. Gergov and Ch. Meinel. Mod2-OBDDs: A Data Structure that Generalizes EXOR-Sum-of-Products and Ordered Binary Decision Diagrams. In *Formal Methods in System Design*, volume 8, pages 273–282. Kluwer Academic Publishers, 1996.

[MB88] J.-Ch. Madre and J.-P. Billon. Proving Circuit Correctness Using Formal Comparison Between Expected and Extracted Behaviour. In *Proc. of the 25th ACM/IEEE Design Automation Conference*, pages 308–313, 1988.

[MS97] Ch. Meinel and A. Slobodova. Speeding Up Variable Reordering of OBDDs. In *Proc. of the Int. Conf. on Computer Design*, pages 338–343, 1997.

[MS98] Ch. Meinel and H. Sack. $\oplus$-OBDDs - A BDD Structure for Probabilistic Verification. In *Proc. of the Pre-LICS Workshop on Probabilistic Methods in Verification (PROBMIV'98), Indianapolis, IN, USA*, pages 141–151, 1998.

[MT98] Ch. Meinel and T. Theobald. *Algorithms and Data Structures in VLSI Desing.* Springer, 1998.

[Rud93] R. Rudell. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In *Proc. of the IEEE Int. Conference on Computer Aided Design*, pages 42–47, 1993.

[Waa97] S. Waack. On the Descriptive and Algorithmic Power of Parity Ordered Binary Decision Diagrams. In *Proc. of the 14th Symposium on Theoretical Aspects of Computer Science*, volume 1200 of *LNCS*. Springer, 1997.