URL: http://www.elsevier.nl/locate/entcs/volume22.html 15 pages

Probabilistic Symbolic Simulation and Verification with ⊕-OBDDs

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 verification of combinational and sequential circuits. Due to limitations of the descriptive power of OBDDs several more general models of Binary Decision Diagrams have been studied. In this paper, ⊕-OBDDs - also known as Mod2OBDDs - in respect to their ability to serve as a tool for combinational verification are considered. Besides the application of ⊕-OBDDs, the more general problem of how to exploit the inherent potential of ⊕-OBDDs more efficiently is addressed.

1 Introduction

A major problem in the computer aided design of digital circuits (VLSI-CAD)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. Although OBDDs were introduced in the context of CAD applications, they are now used in many different fields like e.g. the solution of combinatorial problems or design and verification of communication protocols. For an overview see [11].

Applications based on OBDDs are limited, since the descriptive power of OBDDs is limited. Therefore, not every Boolean function of practical importance can be represented efficiently. For example, the OBDD-representations of the *multiplication* or the *hidden weighted bit function* (HWB) are of exponential size [4]. Hence, more general BDD models have been studied. In this paper we address ⊕-OBDDs (also known as Mod2-OBDDs), introduced as an extension of OBDDs [6]. ⊕-OBDDs are more, sometimes even exponentially more, space-efficient than OBDDs. ⊕-OBDDs 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, $\oplus$-OBDDs do not provide a canonical representation of Boolean functions. For canonical representations like OBDDs, testing the equivalence of two OBDDs reduces to a simple pointer comparison in the computer. For non canonical representations, the equivalence test is much more difficult. Doing symbolic simulation of digital circuits with OBDD-like data structures, the efficiency crucially depends on a fast equivalence test. More precisely, synthesis of OBDDs becomes an operation of exponential time complexity if there is no cache available to look up - rather than to recompute - the result of single synthesis operations that already occurred at a previous step of the computation. Looking up this cache requires that the equivalence of the OBDDs of the current and the cached operation is easy to check.

The fastest known deterministic equivalence test for $\oplus$-OBDDs based on a minimisation algorithm introduced in [12] requires time cubic in the number of nodes. Hence, it is not suitable for practical purposes. In [6], a fast probabilistic equivalence test for $\oplus$-OBDDs has been proposed that requires only a number of arithmetic operations that is linear in the number of variables.

In this paper we show how to work with $\oplus$-OBDDs in symbolic simulation of digital circuits based on a probabilistic equivalence test. Another problem that we address is that the potential of the $\oplus$-OBDDs in symbolic simulation often can not be exploited, because no EXOR(EQU) gates appear in the description of the circuit to be simulated. To avoid this problem, we propose methods on how to integrate additional $\oplus$-nodes into the data structure.

The paper is structured as follows: In Section 2, we remind some basic definitions concerning $\oplus$-OBDDs. In Section 3, we present the probabilistic equivalence test based on Boolean signatures. Section 4 deals with reduction rules and $\oplus$-OBDD synthesis. Section 5 deals with integration of additional $\oplus$-nodes into the BDD data structure and Section 6 concludes the paper with experimental results and an outlook on future work.

2 $\oplus$-OBDDs and Some Basic Facts

A $\oplus$-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.

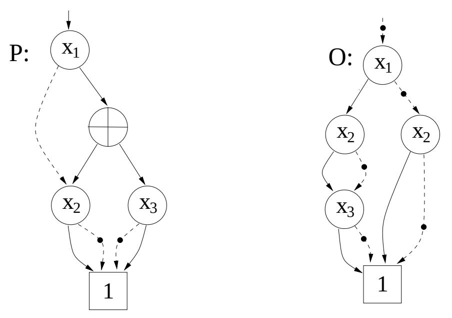
(To deal with Boolean functions $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$, we consider *multi rooted shared* $\oplus$-OBDDs by introducing multiple roots into a single $\oplus$-OBDD, each root representing a subfunction of $f=(f_1,\ldots,f_m)$, $f_i : \mathbb{B}^n \rightarrow \mathbb{B}$.) The two terminal nodes with no outgoing arcs are labelled with the Boolean constants 0 and 1. The remaining nodes are either labelled with Boolean variables $x_i \in X_n$, denoted as *branching nodes*, or with the binary Boolean function $\oplus$ (EXOR), denoted as $\oplus$-*nodes*. On each path, every variable must occur at most once. In the following, let $l(v)$ denote the label of the node $v \in V$ and $size(P)$ the number of nonterminal nodes of P.

$E \subseteq V \times V$ denotes the set of edges. The two edges starting in a branching node v are labelled with 0 and 1. The *0(1)-successor* of node v is denoted by $v0(v1)$. 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 Boolean values assigned to the leaves extend to Boolean values associated with all nodes of P as follows:

- Let $v0$ and $v1$ be the successors of v, carrying the Boolean values $\delta_0, \delta_1 \in \{0,1\}$.
- If v is a branching node labelled with $x_i \in X_n$, then v is associated with δ_{a_i}.
- If v is a $\oplus$-node, then v is associated with $\oplus(\delta_0, \delta_1) = (\delta_0 + \delta_1) \bmod 2$.

$f_P(a)$ takes the value associated with the source of P. Thus, the value of a Boolean function f_P represented by the $\oplus$-OBDD P can be computed in time $O(size(P))$. Furthermore, we can also consider the use of complemented edges as introduced in [9] to achieve an even more compact representation.



x_1	x_2	x_3	f_P
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Figure 1. *$\oplus$-OBDD P and OBDD O with complemented edges, both computing f_P*

For an illustration of the concept of $\oplus$-OBDDs see Figure 1. Let $f_P : \{0,1\}^3 \to \{0,1\}$ be defined by the given truth table. Moreover, let π be the natural order on the set of variables, i.e., $\pi(i) = i$. For branching nodes, the dashed line always represents the edge labelled with 0. A dot on an edge denotes that the function represented by the successing $\oplus$-OBDD is complemented. Note that if we are working with complemented edges one of the two leaf nodes can be omitted.

Since OBDDs are special cases of $\oplus$-OBDDs (namely $\oplus$-OBDDs without $\oplus$-nodes), for each variable order every Boolean function can be represented by means of a $\oplus$-OBDD. Therefore, we may conclude that the size of an optimal $\oplus$-OBDD for a given Boolean function f is not greater than the size of an optimal OBDD for f. Moreover, we can show that there exist Boolean functions with small (low polynomial degree) $\oplus$-OBDD representation whose OBDDs are of exponential size. One example is the *hidden weighted bit function* HWB, which is defined as follows: If $wt(x)$ is the number of ones in the input assignment of $x=(x_1,\ldots,x_n)$ (the *weight*) and if, for simplicity, x_0 denotes 0, then HWB is defined by $\text{HWB}(x)=x_{wt(x)}$. In [4] it has been shown that each OBDD representation of the HWB must be of exponential size, but its $\oplus$-OBDD representation is only of cubic size [6]:

The equality $\text{HWB}(x)=\bigoplus_{k=1}^{n}(x_k \wedge E_k(X))$ where $E_k(x)$ equals one if the input assignment of x contains exactly k ones can be verified easily. Since, for each variable order, $x_k \wedge E_k(x)$ can be represented by an OBDD of at most quadratic size, the above equality can be immediately transformed into a cubic size $\oplus$-OBDD for HWB.

For Boolean function manipulation we are in need of an efficient algorithm performing the application of a binary Boolean operator on two $\oplus$-OBDDs. As shown in [5] the result of the application of a generic boolean operator $\otimes$ on two $\oplus$-OBDDs R and Q of the variable ordering π can be constructed in time $O(\text{size}(R)\cdot\text{size}(Q))$. Even better, if $\otimes \in \{\oplus, \equiv\}$, then the resulting $\oplus$-OBDD can be constructed in constant time.

But, $\oplus$-OBDDs do not provide a canonical representation of Boolean functions.

3 Probabilistic Equivalence Test for $\oplus$-OBDDs

Similarly to $\oplus$-OBDDs, we can consider Ω-OBDDs for a basis Ω of binary Boolean functions by allowing all so-called functional nodes labelled with an element of Ω [10]. By introducing functional nodes into the OBDD representation, we lose canonicity. Hence, it becomes an essential task to decide whether two representations denote the same function. The equivalence test for all Ω-OBDDs, $\Omega \in \{\{\vee\},\{\wedge\},\{\vee,\wedge\}\}$ is **co-NP**-complete. The situation is different for $\Omega=\{\oplus\}$, where the determination of equivalence is within **co-R** (see [5]).

Recently, a deterministic equivalence test for a $\oplus$-OBDD-like data structure was introduced [12] that can be adapted to our model. This equivalence test is based on a minimisation algorithm that requires the solution of a system of linear equations. Doing this by Gaussian elimination, the runtime is cubic in the number of nodes and therefore to time expensive for practical applications.

The probabilistic equivalence test for $\oplus$-OBDDs proposed in [5] needs only linearly 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 [1]. Equivalence of two $\oplus$-OBDDs is determined by an algebraic transformation of the $\oplus$-OBDDs in terms of polynomials over a finite field. More precisely, we assign the polynomial $p_x = x$ to a variable x and for each Boolean function F represented by a $\oplus$-OBDD, we transform the Boolean Functions $\neg F$ and $F_1 \wedge F_2$ into the arithmetic expressions $1-p_F$ and $p_{F_1} \cdot p_{F_2}$, where p_F represents the polynomial assigned to F. By exploiting the law of DeMorgan and idempotence we derive $p_{F_1 \vee F_2} = p_{F_1} + p_{F_2} - p_{F_1} p_{F_2}$ and $p_{F_1 \oplus F_2} = p_{F_1} + p_{F_2} - 2 p_{F_1} p_{F_2}$ for the binary Boolean operations OR and EXOR. For a more detailed description of the application of this algebraisation technique see [8].

Let $GF(2^m), m \in \mathbb{N},\ m > (\log n)+1$, denote a Galois field with 2^m elements of characteristic 2. Note that using $GF(2^m)$ simplifies the polynomial for the EXOR operation. If we consider the elements of $GF(2^m)$ as bit vectors of length m, addition can be performed by bitwise EXOR. Multiplication has to be carried out according to the rules of the polynomial ring over $GF(2^m)$. With each node v of a $\oplus$-OBDD P we associate the following polynomial $p_v : (GF(2^m))^n \to GF(2^m)$:

$$p_v(x_1,\ldots,x_n) = \begin{cases} 0\ (1) & v \text{ is } \textit{0(1)-sink} \\ x \cdot p_{v1}(x_1,\ldots,x_n) + (1-x) \cdot p_{v0}(x_1,\ldots,x_n) & l(v) = x \in X_n \\ p_{v0}(x_1,\ldots,x_n) + p_{v1}(x_1,\ldots,x_n) & l(v) = \oplus \end{cases}$$

The polynomial associated with the $\oplus$-OBDD P is the polynomial associated with the root v_0 of P. Note that the polynomial remains unchanged for different representations P of the same Boolean function.

Let P and Q be two $\oplus$-OBDDs and let $a_1,\ldots,a_n \in GF(2^m)$ be generated independently and uniformly random. The equivalence of two polynomials in symbolic representation can be tested by a random algorithm in the following way [5]:

$$\begin{array}{ll} p_P(a_1,\ldots,a_n) = p_Q(a_1,\ldots,a_n) & \text{if } f_P = f_Q \text{ and} \\ \text{Prob}(p_P(a_1,\ldots,a_n) = p_Q(a_1,\ldots,a_n)) < \frac{1}{2} & \text{if } f_P \neq f_Q. \end{array}$$

Thus, if P and Q compute the same function, the algorithm always answers "yes", otherwise it answers "yes" with a probability smaller than 1/2. The bit string representing the polynomial associated with the function f_P computed by the $\oplus$-OBDD P is called *Boolean signature*. The error probability depends on the number of elements in $GF(2^m)$. Therefore, we are able to reduce the error by enlarging m or by using multiple signatures per node with different random instances of $a_1,\ldots,a_n \in GF(2^m)$. In [1,2] an estimation of the probability of degeneracy in BDD synthesis based on signatures is given, i.e., the probability that during the synthesis of a $\oplus$-OBDD P the signatures for two nodes representing different Boolean functions are computed to be equal. By using s different signatures per node the error probability computes to at most

$$error < \frac{size(P)^2 \cdot n^s}{2 \cdot |GF|^s}$$

where $size(P)$ denotes the number of nodes of the $\oplus$-OBDD P, n the number of variables, and $|GF|$ the number of elements in the finite field. For our experiments in section 5 we are working with up to 3 different signatures of 32 bit length per node. Thus, we were sufficient to perform all computations without error. If we have, for example, a $\oplus$-OBDD with 10^7 Nodes depending on 100 Variables and if we are working with 3 different signatures each of 32 bit length, we have to face an error probability of less than $6.31 \cdot 10^{-10}$.

4 Reduction and Synthesis of $\oplus$-OBDDs

In general, reduction rules for OBDDs are also suitable for $\oplus$-OBDDs, but, as a major difference, their application does only lead to a smaller $\oplus$-OBDD-representation and not to a canonical one. We distinguish two types of reductions, the *deletion rule*, also referred to as *simple reduction*, and the *merging rule*, also known as *algebraic reduction.*

In a $\oplus$-OBDD, a node v is redundant if both of its edges point to the same successor. Then we can apply the deletion rule: In a case of a branching node, this node can be replaced by reconnecting all its incoming edges to its successor; a $\oplus$-node with two equal successors has to be replaced by the 0-sink.

The merging rule performs to identification of isomorphic subgraphs and applies for branching nodes in the same way as for $\oplus$-nodes (see Figure 2).

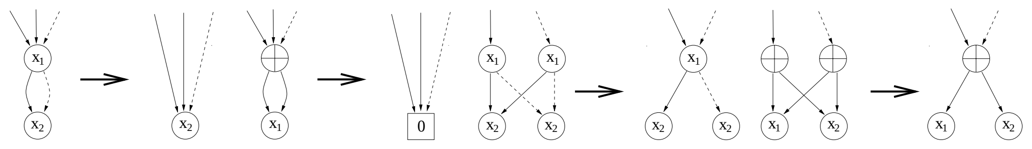


Figure 2. *Deletion rule and merging rule for $\oplus$-OBDDs.*

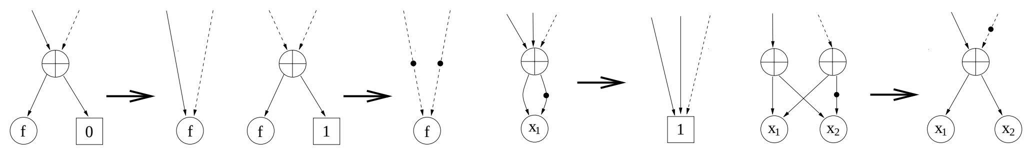


Figure 3. *$\oplus$-OBDD reduction rules for terminal nodes and complemented edges.*

Additionally, we also have to consider the case that one successor of a $\oplus$-node is a terminal node. If the *0-sink* is a successor of a $\oplus$-node, then the $\oplus$-node is replaced by its second successor. On the other hand, if the *1-sink* is a successor of a $\oplus$-node, then the $\oplus$-node is replaced by complementing all its incoming edges and connecting them to its second successor. Note that rules concerning complemented edges must also be taken into account. Hence, the *deletion rule* replaces each $\oplus$-node v having successors which are the complement of each other ($v_l = \overline{v_r}$) by the *1-sink*. The *merging rule* reduces $\oplus$-nodes v and w having isomorphic subgraphs of different complementation parity ($\{v_l, v_r\} = \{w_l, \overline{w_r}\}$ or $\{v_l, v_r\} = \{\overline{w_l}, w_r\}$) to a single node v by using equivalence relations for complemented edges (see Figure 3). Using complemented edges, we can ensure a more efficient usage of the caches required for synthesis operations if we restrict edge complementation to the *0-successor*, respectively to the *left* successor in case of a $\oplus$-node, of the node under consideration. To achieve this, we can choose one of the equivalences shown in Figure 4.

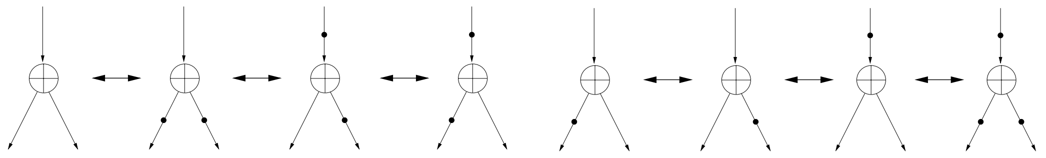


Figure 4. *Equivalences for $\oplus$-nodes and complemented edges.*

Furthermore, we have to take under consideration reduction possibilities concerning groups of $\oplus$-nodes. If, e.g. we have a complete subtree of $\oplus$-nodes within an $\oplus$-OBDD, each leaf has to be compared with every other leaf of that subtree. If two leafs are representing the same function, they can be replaced by a 0-sink which may also affect the node in the level above. Two leafs representing the complement of each others function can be replaced by a 1-sink. Also we have to consider chains of $\oplus$-nodes where we have to compare all of its leaf nodes to apply all possible reductions in the way described above.
Next, we are going to describe a synthesis algorithm for $\oplus$-OBDDs. We will assume that the reader is familiar with standard OBDD synthesis algorithms. The conventional apply algorithm for $\oplus$-OBDDs proposed in [6] shows that applying a binary Boolean operation to two $\oplus$-OBDDs requires at most quadratic time. But in convenient OBDD implementations, all Boolean operations are implemented by means of the *ite* operator [3] which is defined as a ternary Boolean function for the inputs F, G, H by *"If F then G else H"* or $ite(F, G, H) = F \cdot G + \overline{F} \cdot H$ and can be evaluated by recursive application of the Boole-/Shannon function decomposition:

$$f = xf|_{x=1} + \overline{x}f|_{x=0}.$$

We decided to adapt the *ite*-algorithm for $\oplus$-OBDDs: In combinational synthesis, the description of a digital circuit is read and each gate of the circuit will be simulated by a $\oplus$-OBDD. Thus, for all gates except for those which perform an EXOR(EQU)-operation we apply the regular *ite*-algorithm. Gates implementing an EXOR(EQU)-operation are simply simulated by $\oplus$-nodes connected to the $\oplus$-OBDDs representing the gate's inputs.

The second step of adaption involves the creation of $\oplus$-OBDDs for cofactors $f|_{x_i=d}$, $d \in \{0, 1\}$ of a Boolean function f (a cofactor $f|_{x=d}$ of a Boolean function

f is the function that results if variable x is set to a fixed binary value d): Regular cofactors $f|_{x_i}$, i.e., cofactors of a function associated with a branching node v labelled by the variable x_i, $l(v) = x_i$, are derived by simply returning the *0-successor*, respectively the *1-successor*, of node v. Creating the cofactors of a function associated with a $\oplus$-node v according to a variable x_i may necessitate 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 labelled by the variable x_i (see Figure 5).

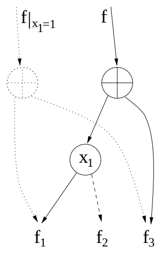


Figure 5. *Cofactor creation* $f|_{x_1=1}$ *in* $\oplus$*-OBDD* P *with* $l(source_P) = \oplus$.

To speed up the performance of the *ite* operation, we are using a *computed table*, which is organised as a hash based cache, to store and reuse the results of *ite*. Before a new node is created, we always look up a *unique table* organised as a hash table to prevent the creation of already allocated nodes. In both, *computed table* and *unique table*, every reference is made by application of the probabilistic equivalence test to identify the underlying $\oplus$-OBDDs. To avoid redundant entries in the *computed table*, we transform the triple (F, G, H) to a standard form by reordering it and checking the constraints for complemented edges.

However, with the modified *ite* algorithm the $\oplus$-OBDD that we create for a circuit description which contains neither EXOR gates nor EQU gates is isomorphic to an OBDD created by conventional *ite*-algorithm.

5 Introduction of $\oplus$-nodes into the BDD data structure

In the case that the circuit under consideration contains no EXOR(EQU) gate, we have to introduce $\oplus$-nodes to take advantage of the potential of the $\oplus$-OBDDs. We have investigated two different approaches:

(i) Using alternative function decompositions that include $\oplus$-node generation or,

(ii) substituting linearly dependent functions by linear combinations of $\oplus$-nodes.

The conventional *ite*-algorithm is based on the Boole/Shannon function decomposition. As an alternative, we may use the positive (pDE) or negative Davio (nDE) expansion:

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

By applying a binary Boolean operator $\otimes$ to two Boolean functions f, g according to the positive Davio expansion, we introduce two new $\oplus$-nodes for each recursive apply step:

$$f \otimes g = (f|_{x=0} \otimes g|_{x=0}) \oplus x((f|_{x=1} \otimes g|_{x=1}) \oplus (f|_{x=0} \otimes g|_{x=0})).$$

But, the ongoing creation of $\oplus$-nodes may increase the total number of nodes beyond the size of conventional OBDDs for the same function. This is the case, if none of the reduction rules can be applied. We can try to avoid this effect by using the alternative function decompositions pDE and nDE not always but only sometimes. But the problem remains that the created $\oplus$-nodes may not be positioned at an appropriate place in the $\oplus$-OBDD and therefore, will be of no benefit.

Maybe $\oplus$-nodes can be introduced in a more sophisticated way: It is convenient to regard the space $\mathbb{B}_n$ of Boolean functions of n variables as an algebra over the two-element field $\mathbb{Z}_2$, i.e. a 2^n-dimensional vector space with an additional multiplication operation. The product of $f, g \in \mathbb{B}$, which corresponds to coordinate wise conjunction is denoted by fg and the sum, which corresponds to coordinate wise EXOR, by $f + g$. In this context, the variable x_i is taken to represent the projection from $\{0,1\}^n$ to the *ith* coordinate and $\overline{x_i}$ as the according complement.

Now let P be a $\oplus$-OBDD representing f_P. The Boolean function f_P can be regarded as the Boolean function assigned to the top node ν of the $\oplus$-OBDD P and is defined by induction on $i = n+1, n, \ldots, 0$, where i denotes the level to which ν belongs:

(i) $i = n+1$: ν is leaf, $f_\nu = 1$.

(ii) ν is node at level i, $1 \leq i \leq n$: let f_ν^1 be the function computed its 1-successor and f_ν^0 the function computed its 0-successor, then $f_\nu = x_{\pi(i)} f_\nu^1 + \overline{x_{\pi(i)}} f_\nu^0$.

If the function under consideration may be expressed as a linear combination of already computed functions, $f = \sum_i f_i$, we are able to represent this function as a tree of $\oplus$-nodes connected to the $\oplus$-OBDDs representing the functions f_i.

Unfortunately it is very expensive to include a test into the implementation that proves, whether an already computed $\oplus$-OBDD is part of a linear combination of the already computed functions.

6 Experimental Results

For our experiments we used an Intel PPro200 Linux workstation, limiting memory-size to at most 200MB. A symbolic simulation procedure based on our $\oplus$-OBDD-package was used together with a subset of the smaller LGSynth91 Benchmark circuits.

Because $\oplus$-OBDD-based symbolic simulation of circuits is probabilistic, we had to check our results for correctness. This was done by translating the constructed $\oplus$-OBDD into a multiplexer circuit containing EXOR gates coded in BLIF (Berkeley Logic Interchange Format). Then the translated $\oplus$-OBDD was verified against the BLIF version of the circuit's specification file from the LGSynth91 Benchmarks. This verification procedure was performed by the standard synthesis and verification tool VIS [7].

The size of OBDDs and $\oplus$-OBDDs depends crucially on the chosen variable order. Because our $\oplus$-OBDD-package is already under construction and dynamic reordering is not yet implemented, we were working with a single static variable order not regarding, whether it is well suited for OBDD or $\oplus$-OBDD representation. Hence, in our experiments we simply used the variable orders given by the circuit descriptions.

In Table 1, the column circuit contains the name of the circuit netlist to be simulated. As a reference, column two contains the final size for the OBDD representing the circuit. The remaining columns contain the final sizes of the resulting $\oplus$-OBDDs computed by the *ite*-algorithm and by positive or negative Davio expansion.

Altogether, we required up to 3 distinct signatures of 32 Bit length per node, resulting in a maximum of additional 96 bit of memory per node over the conventional OBDD node size to simulate all circuits correctly. Of course the additional memory required for signatures lessens memory efficiency for $\oplus$-OBDDs compared to OBDDs.

The $\oplus$-OBDD-sizes given for the application of the *ite*-algorithm differ only from OBDD-sizes if EXOR(EQU) gates are included in the circuit description. The exclusive application of nDE/pDE sometimes leads to very good results, e.g. for s420.1, but, in many cases too many $\oplus$-nodes seem to be created

circuit	OBDD	⊕-OBDD-size		
		ite	**nDE**	**pDE**
c432	1733	1733	4745	6266
c499	45922	12800	13703	13698
c880	346660	346660	284187	399071
c1355	45922	45922	14197	14392
c1908	36007	36007	39126	17824
c5315	>200MB	>200MB	66376	73749
cordic	157	45	87	87
count	234	234	543	294
example2	469	469	605	644
frg2	6471	6471	7606	5049
i2	335	335	861	317
k2	28336	28336	7736	6054
pcler8	139	139	122	223
s208.1	1033	1033	186	158
s420.1	262227	262227	732	568
s510	19076	19076	641	767
s1494	1016	1016	1486	1378
s820	2651	2651	563	822
s832	2651	2651	565	822
s635c	656	656	1746	728
s967	1732	1732	1084	1281
s967c	1723	1723	1084	1281
x3	2670	2670	2391	2503
xi30	121	150	240	240

Table 1
Comparison of OBDD-size and ⊕-OBDD-size for different function expansions

in the wrong places so that $\oplus$-OBDD-size is greater than OBDD-size. But, also in many cases $\oplus$-OBDD-size is smaller than OBDD-size and therefore, we think that $\oplus$-OBDDs are a promising approach for combinatorial verification in a probabilistic way. For example, the OBDD for circuit C5315 could not be created because of memory limitations, but the $\oplus$-OBDDs depending on pDE/nDE succeeded.

In Table 2 and Table 3, we have tried to limit the use of pDE/nDE-expansion during the simulation process. In the first column the circuit name is given. The second column denotes the OBDD-size for comparison. Columns 4 to 7 show $\oplus$-OBDD-sizes in relation to the application of pDE/nDE-expansion. The header 50% denotes that pDE/nDE-expansion was applied in 50% of all apply-steps while the ite-algorithm was used in the remaining apply-steps during a single run. Note that the given percentage is closely related to the number of $\oplus$-nodes in the $\oplus$-OBDD. For each circuit under consideration we performed 10 simulation runs with the given percentage of pDE/nDE usage. During a single run the consideration whether to use pDE/nDE or ite as an apply-step was chosen at random with the given percentage.

Thus, we tried to show the influence of the number of $\oplus$-nodes in a $\oplus$-OBDD on its size comparing the values in a given row. On the other hand, the experiment also shows the influence of the positioning of $\oplus$-nodes in the $\oplus$-OBDD on its size, if we compare the column for a given circuit under a fixed percentage. For each circuit in row 1, we placed the minimum number of achieved nodes, row 2 contains the average number of nodes out of 10 random runs, and row 3 contains the maximum number of nodes.

The difference of the maximum and the minimum size shows the impact of the placement of $\oplus$-nodes on the $\oplus$-OBDD-size.

s1196, C1908, and C432 seem to be circuits that do not profit very much from the use of $\oplus$-nodes at all. Therefore, the less $\oplus$-nodes are used, the smaller is the average size of the $\oplus$-OBDDs.

For the other circuits the situation is different. The more $\oplus$-nodes are used, the less becomes the overall size.

In many cases we achieve smaller $\oplus$-OBDDs by limiting the application of nDE/pDE expansion compared to the exclusive application of nDE/pDE. It seems that if a circuit profits from the usage of $\oplus$-nodes, a more frequent application of nDE/pDE expansion leads to smaller results. This may be explained by the fact that if we use more $\oplus$-nodes in a $\oplus$-OBDD, we will achieve a higher probability to apply reduction rules.

In all tables we have only listed the sizes of our obtained results and not the runtime, because we are not able to compare our experimental $\oplus$-OBDD package to sophisticated OBDD packages, which are highly optimized for runtime.

The difference of minimum and maximum $\oplus$-OBDD-size in Table 2 and Table 3 confirmes us that we should concentrate our work on a more sophisti-

circuit	**OBDD**		**⊕-OBDD-size**			
			50%	**20%**	**10%**	**5%**
C1355	45922	min	13868	39766	43581	43036
		avg	24700	44877	46497	46714
		max	36783	49877	48780	49418
C1908	36007	min	41876	36485	36080	35600
		avg	45294	40664	40664	37678
		max	50455	46929	46270	43779
C432	1733	min	2811	1861	1748	1737
		avg	3156	2234	2111	1821
		max	3540	2791	2641	2091
C499	45922	min	12890	7113	7095	7244
		avg	13273	7607	8405	10110
		max	13688	8302	11581	13461
s208.1	1033	min	118	146	613	642
		avg	602	796	996	995
		max	1054	1057	1041	1038
s420.1	262227	min	433	131040	131483	254720
		avg	79133	235379	223144	234514
		max	131593	262279	262236	262252
s510	19076	min	596	634	603	588
		avg	8050	17264	17246	17227
		max	19237	19098	19175	19098

Table 2

Influence of placement and number of of ⊕-nodes on ⊕-OBDD-size (1)

cated ⊕-node placement. This could be done by exploiting linear combinations as proposed in section 5, but up to now we were not able to achieve a sufficient implementation with a reasonable runtime.

The chosen variable orders may not be well suited for both OBDDs and ⊕-OBDDs. Hence, for a better comparison of OBDD and ⊕-OBDD performance we are currently implementing dynamic variable reordering techniques

circuit	OBDD		$\oplus$-OBDD-size			
			50%	**20%**	**10%**	**5%**
s820	2651	min	1417	1253	2571	1746
		avg	1920	2283	2725	2605
		max	1689	1842	1868	1804
		max	2332	2898	2979	2820
s832	2651	min	723	1290	1253	2280
		avg	1594	2469	2450	2646
		max	2883	2924	2720	2877
s1196	2294	min	2657	2279	2335	2252
		avg	2795	2543	2420	2373
		max	2830	2772	2528	2545
s967	1732	min	1483	1567	1533	1672
		avg	1637	1704	1731	1748
		max	1689	1842	1868	1804
s967c	1723	min	1483	1622	1546	1360
		avg	1592	1749	1753	1735
		max	1689	1806	1830	1803

Table 3
Influence of placement and number of of $\oplus$-nodes on $\oplus$-OBDD-size (2)

for $\oplus$-OBDDs.

All in all, $\oplus$-OBDDs seem to be well suited for all tasks based on symbolic circuit simulation, working on the base of a probabilistic equivalence test. The achieved results motivate further research with emphasis on $\oplus$-node placement and the adaption and implementation of dynamic reordering techniques for $\oplus$-OBDDs.

References

[1] 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.

[2] K. S. Brace. *Ordered Binary Decision Diagrams for Optimization in*

Symbolic Switch-Level Analysis of MOS Circuits. PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1992.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] The VIS Group. VIS: A system for Verification and Synthesis. In *Proc. of the 8th Int. Conference on Computer Aided Verification*, number 1102 in Lecture Notes in Computer Science, pages 428–432. Springer, 1996.

[8] J. Jain, J. Bitner, D. S. Fussel, and J. Abraham. Probabilistic Verification of Boolean Functions. In *Formal Methods in System Design*, volume 1, pages 63–117, 1992.

[9] J.-C. 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.

[10] Ch. Meinel. *Modified Branching Programs and Their Computational Power*, volume 370 of *LNCS*. Springer Verlag, Heidelberg, 1989.

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

[12] 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.