published in *Proceedings of the International Workshop on Logic Synthesis (IWLS' 98)*, 7-10 June 1988, Tahoe City, CA, USA, pp. 19–24

# ⊕-OBDDs - a BDD Structure for Probabilistic Verification

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 usefull in the verification of combinational and sequential circuits. Due to limitaions 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 [Bry92, MT97].

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 [Bry91]. Therefore, more general BDD modells 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. ⊕-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, ⊕-OBDDs do not provide a canonical representation of Boolean functions. For canonical representations like OBDDs, testing the equivalence 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 exponential operation 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 ⊕-OBDDs based on a minimization algorithm introduced in [Waa97] requires time cubic in the number of nodes. Hence, it seems not to be suitable for practical purposes. In [GM96], a fast probabilistic equivalence test for ⊕-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 ⊕-OBDDs in symbolic simulation of digital circuits. In symbolic simulation of digital circuits the potential of the ⊕-OBDDs often can not be exploited, because no EXOR(EQU) gates appear. To avoid this problem, we propose methods on how to integrate ⊕-nodes into the data structure.

The paper is structured as follows: In Section 2, we remind some basic definitions concerning ⊕-OBDDs. In Section 3, we present the probabilistic equivalence test based on Boolean signatures. Section 4 deals with ⊕-OBDD synthesis. In Section 5 we show how to integrate new ⊕-nodes into the BDD datastructure and Section 6 concludes the paper with experimental results.

## 2 $\oplus$-OBDDs and Some Basic Facts

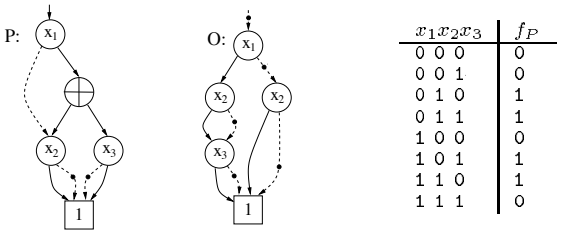
A $\oplus$-OBDD $P$ over a set $X_n = \{x_1, \ldots, x_n\}$ of Boolean variables is a directed acyclic connected graph $P = (V, E)$. $V$ is the set of nodes, consisting of nonterminal nodes with out-degree 2 and of terminal nodes with out-degree 0. There is a distinguished nonterminal node, the *root*, which, as only node, has the in-degree 0. (To deal with Boolean functions $f : \mathbb{B}^n \to \mathbb{B}^m$, we consider *multi rooted shared* $\oplus$-OBDDs by introducing multiple roots into a single $\oplus$-OBDD, each root representing a subfunction of $f = (f_1, \ldots, f_m)$, $f_i : \mathbb{B}^n \to \mathbb{B}$.) The two terminal nodes with no outgoing arcs are labelled by 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 $\sigma$ 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 $\sigma$.

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 $\delta_{a_i}$.
- 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 root 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 ‖MB88‖ to achieve an even more compact representation.
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 $\pi$ 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 by 0. A dot on an edge denotes that it is complemented.




| $x_1x_2x_3$ | $f_P$ |
|---|---|
| 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 $f_P$*

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, there exist Boolean functions with small (low polynomial degree) $\oplus$-OBDD representation whose OBDDs are of exponential size, like the *hidden weighted bit function*.

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 ‖GM93‖ the result of the application of a generic boolean operator $\otimes$ on two $\oplus$-OBDDs $R$ and $Q$ of the variable ordering $\pi$ 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 $\Omega$-OBDDs for a basis $\Omega$ of binary Boolean functions by allowing all so-called functional nodes labelled by an element of $\Omega$ ‖Mei89‖. 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 $\Omega$-OBDDs, $\Omega \in \{\{\vee\}, \{\wedge\}, \{\vee, \wedge\}\}$ is **co-NP**-complete. For this reason the approach proposed by ‖SDG93‖ containing AND-nodes as well as $\oplus$-nodes fails. The situation is different for $\Omega = \{\oplus\}$, where the determination of equivalence is within **co-R** (see ‖GM93‖).

Recently, a deterministic equivalence test for a $\oplus$-OBDD-like data structure was introduced ‖Waa97‖ that can be adapted to our model. This equivalence test is based on a minimization 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 too time expensive for practical applications.

The probabilistic equivalence test for $\oplus$-OBDDs proposed in [GM93] 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 [BCW80]. 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 $p_{F_1 \oplus F_2} = p_{F_1}+p_{F_2}-2p_{F_1}p_{F_2}$ for the binary Boolean operations OR and EXOR. For a more detailed description of the application of this algebraization technique see [JBFA92].

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 concerning the polynomial ring over $GF(2^m)$.
With each node $v$ of a $\oplus$-OBDD $P$ we associate the polynomial $p_v : (GF(2^m))^n \rightarrow GF(2^m)$ in the following way:

- if $v$ is a leaf node $0/(1)$, then $p_v = 0$ $(1)$,
- if $v$ is a branching node, $l(v) = x \in X_n$, then $p_v = x \cdot p_{v1} + (1-x) \cdot p_{v0}$,
- and if $v$ is a $\oplus$-node, $l(v) = \oplus$, then $p_v = p_{v0} + p_{v1}$.

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.

Now, 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 [GM93]:

- if $f_P = f_Q$, then $p_P(a_1, \ldots, a_n) = p_Q(a_1, \ldots, a_n)$,
- if $f_P \neq f_Q$, then $\text{Prob}(p_P(a_1, \ldots, a_n) = p_Q(a_1, \ldots, a_n)) < \frac{1}{2}$

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 is depending on the number of elements in $GF(2^m)$. Therefore, we are able to reduce it by enlarging $m$ and by using several signatures per node with different instances of $a_1, \ldots, a_n \in GF(2^m)$. In [BCW80, Bra92] 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. For an example, if we are using $s$ different signatures per node the error probability is 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 Synthesis of $\oplus$-OBDDs

We will assume that the reader is familiar with standard OBDD synthesis algorithms. The conventional apply algorithm for $\oplus$-OBDDs proposed in [GM96] shows that applying a binary Boolean operation on two $\oplus$-OBDDs requires at most quadratic time. But in convenient OBDD implementations, all Boolean operations are implemented by means of the *ite* operator [BRB90] 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-expansion:

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

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 simulating the gate's inputs.

The second step of adaption involves the creation of cofactors $f|_{x_i=d}$, $d \in \{0,1\}$ for $\oplus$-OBDDs. Regular cofactors, i.e., cofactors of a function associated with a branching node $v$, 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 $x_i$ necessitates the allocation of a new $\oplus$-node connected to the cofactors of the left and right successor of $v$. In this 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 2).

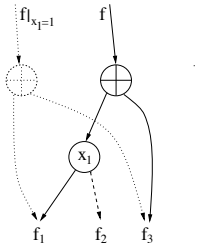



**Figure 2:** *Cofactor creation $f|_{x_1=1}$ in $\oplus$-OBDD $P$ with $l(root_P)=\oplus$.*

To speed up the performance of the *ite* operation, we are using a *computed table*, which is organized 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* organized 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 datastructure

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:

1. Using alternative function decompositions that include $\oplus$-node generation, or
2. substituting linearly dependend functions by $\oplus$-node combinations.

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

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

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:

$$\begin{aligned} f \otimes g \;=\; & (f|_{x=0} \otimes g|_{x=0}) \oplus \\ & x((f|_{x=1} \otimes g|_{x=1}) \oplus (f|_{x=0} \otimes g|_{x=0})). \end{aligned}$$

But, the ongoing creation of $\oplus$-nodes by random 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 diagram and therefore, will be of no benefit.

$\oplus$-nodes should 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 $f, g \in \mathbb{B}_n$ which corresponds to coordinatewise conjunction is denoted by $fg$ and the sum, which correspondes to coordinatewise 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 $\nu$ of the $\oplus$-OBDD $P$ and is defined by induction on $i = n+1, n, \ldots, 0$, where $i$ denotes the level to which $\nu$ belongs:

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

(ii) $\nu$ is node at level $i$, $1 \le i \le n$: let $f_\nu^1$ be the function computed its 1-successor and $f_\nu^0$ 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 practical implementation that proves, whether an already computed $\oplus$-OBDD is part of a linear combination of already computed functions.

Therefore, we decided in a first try only to consider binary linear combinations to investigate their effect on the $\oplus$-OBDDs.

For the implementation we use the symmetry property of the EXOR function. The signature of a new branching node $\nu$ to be created is known in advance, $sig(f_\nu) = x \cdot sig(f_{\nu_1}) + \overline{x} \cdot sig(f_{\nu_0})$. Now the *unique table* has to be tested, whether it contains two nodes $\rho, \sigma$ having the appropriate signatures to construct the linear combination we look for, $sig(f_\nu) = sig(f_\rho) + sig(f_\sigma)$. To find the linear combination we test for every node $\rho$ in the *unique table* whether there exists an appropriate node $\sigma$. Because summation is done by EXOR, we can test $sig(f_\sigma) = sig(f_\nu) + sig(f_\rho)$. The *unique table* is organized as a hashtable and therefore, the existence of $\sigma$ can be testet in time $\mathcal{O}(1)$. The overall time required to find a binary linear combination for a new node to be created computes to $\mathcal{O}(size(P))$.

If we have to consider linear combinations consisting out of more than two functions, the proposed method is not suitable for practical purposes.

# 6 Experimental Results

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

Because $\oplus$-OBDD- based verification 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 check was done by the standard synthesis and verification tool VIS [Gro96].

Because our $\oplus$-OBDD-package is already under construction and dynamic reordering is not yet implemented, we have to work with a static variable order. In our experiments we simply used the variable order given by the circuit descriptions.

In Table 1, in 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.

Alltogether, 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 *ite*-algorithm differ only from OBDD-sizes when EXOR(EQU) gates are included in the circuit description. The exclusive application of nDE/pDE sometimes leads to good results, but, in many cases too much $\oplus$-nodes seem to be created in the wrong places so that the size is greater than the comparable OBDD-size. But, in many cases $\oplus$-OBDD-size is better than OBDD-size and therefore, we think that $\oplus$-OBDDs are a promising approach for working in the field of probabilistic verification. 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, we have tried to limit the use of pDE/nDE-expansions 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. For each circuit under consideration we performed 10 simulation runs with a given percentage of pDE/nDE usage. During a single run the consideration whether to use pDE/nDE or ite was chosen by random with the given percentage. One purpose of this experiment was to show the influence of the positioning of $\oplus$-nodes in the $\oplus$-OBDD on its size. The other purpose was to show the influence of the number of $\oplus$-nodes in a $\oplus$-OBDD on its size. 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.

Not in every case the average $\oplus$-OBDD-size is less than OBDD-size. The difference of the maximum and the minimum sizes shows the impact of the placement of $\oplus$-nodes on the $\oplus$-OBDD-size. This confirmes us that we should concentrate our work on a better $\oplus$-node placement.

s1196 seems to be a circuit that does not profit very much from the use of $\oplus$-nodes. Therefore, the less $\oplus$-nodes are used, the smaller is the size of the $\oplus$-OBDDs. For s967 and s967c the situation is different. The more $\oplus$-nodes are used in the diagram, the less is the overall size.

We left out a table with experiments concerning additional $\oplus$-nodes created by identifying linear combinations, because we could only achieve minor improvements with our proposed method. Identifying only binary linear combinations is not sufficient. We have to think of other methods where all possible lin-

ear combinations can be recognized.

In both 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.

For a better comparison between OBDD and $\oplus$-OBDD performance we are currently implementing dynamic variable reordering techniques for $\oplus$-OBDDs, because they are commonly used in practical OBDD applications.

| circuit | OBDD | $\oplus$-OBDD-size | | |
|---|---|---|---|---|
| | | ite | nDE | pDE |
| c432 | 1733 | 1733 | 4745 | 6266 |
| c499 | 45922 | 12800 | 13703 | 13698 |
| c499.reencoded | 789 | 789 | 812 | 812 |
| 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 |
| 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 |
| x130 | 121 | 150 | 240 | 240 |

**Table 1:** *Comparison of OBDD-size and $\oplus$-OBDD-size*

| circuit | OBDD | | $\oplus$-OBDD-size | | | |
|---|---|---|---|---|---|---|
| | | | 50% | 20% | 10% | 5% |
| 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 2:** *Influence of placement of $\oplus$-nodes on $\oplus$-OBDD-size*

# References


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

[Bra92] Karl S. Brace. *Ordered Binary Decision Diagrams for Optimization in Symbolic Switch-Level Analysis of MOS Circuits*. PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1992.

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

[Bry91] Randal 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] Randal E. Bryant. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[GM93] Jordan Gergov and Christoph 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] Jordan Gergov and Christoph 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.

[Gro96] 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.

[JBFA92] 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.

[MB88] Jean-Christophe Madre and Jean-Paul 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.

[Mei89] Christoph Meinel. *Modified Branching Programs and Their Computational Power*, volume 370 of *LNCS*. Springer Verlag, Heidelberg, 1989.

[MT97] Christoph Meinel and Thorsten Theobald. Geordnete binäre Entscheidungsgraphen und ihre Bedeutung im rechnergestützten Entwurf hochintegrierter Schaltkreise. In *Informatik '97 - Jahresbericht der Gesellschaft für Informatik*, Aachen, 1997. Springer.

[SDG93] Amelia Shen, Shrinivas Devadas, and Abhijit Gosh. Probabilistic Construction and Manipulation of Free Boolean Diagrams. In *Proc. of the IEEE Int. Conf. on Computer Aided Design*, pages 544–549, 1993.

[Waa97] Stephan 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.