Forschungsbericht Nr. 97–15

Case Study: Manipulating $\oplus$-OBDDs by Means of Signatures

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

ISSN 0944-0488

Case Study: Manipulating $\oplus$-OBDDs by Means of Signatures

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

Abstract

We present a case study concerning manipulation of $\oplus$-OBDDs based on a probabilistic equivalence test. Efficient Boolean function manipulation requires efficient algorithms for Boolean synthesis as well as for testing the equivalence of two Boolean functions. Due to the fact that $\oplus$-OBDDs do not provide a canonical representation, the equivalence test becomes an extensive operation. A recently introduced deterministic equivalence test performs only in high polynomial degree execution time and, therefor is not qualified for practical purposes. Hence, we tried to work with a probabilistic equivalence test (with one-sided error probability) based on Boolean signatures. From our experimental results we can conclude that the application of the probabilistic equivalence test is well suited for working in the field of Computer Aided Design.

1 Introduction

Many problems in digital system design, combinatorial optimization, or mathematical logic can be formulated in terms of Boolean functions. Thus, a concise representation which simultaneously provides fast manipulation is very important for problems in form of Boolean functions. Ordered Binary Decision Diagrams (OBDDs) provide a well known data structure for efficient Boolean function manipulation and hence, are widely used in many fields. For an overview see [MT97].

One drawback of OBDDs is that not every Boolean function of practical importance can be represented efficiently. For example, the OBDD-representation of the *multiplication* or the *hidden weighted bit function* (HWB) are of exponential size [Bry91]. Therefor, as an extension of OBDDs, $\oplus$-OBDDs (Mod2-OBDDs) were introduced in [GM93b]. $\oplus$-OBDDs are more, sometimes even exponentially more, space-efficient than OBDDs. $\oplus$-OBDDs preserve the OBDD property of efficient manipulation. Apply operation, quantification, and composition have the same complexity as in the case of OBDDs. Even better, the Boolean functions exclusive or (EXOR), the logical equivalence (EQU), and the negation can be performed in constant time.

Any Boolean function with efficient OBDD-representation can be represented efficiently by $\oplus$-OBDDs, but the contrary does not hold. However, $\oplus$-OBDDs do not provide a canonical representation of Boolean functions. For canonical representations like OBDDs, in practical applications a comparison whether two given OBDDs represent the same function is done simply by comparing the value of the associated pointer variables. For noncanonical representations, the test for equivalence is much more difficult. In Boolean synthesis as a task of Boolean function manipulation with OBDD like data structures, the efficiency depends crucially on the equivalence test. That is because, for OBDD synthesis operations

To be published at the 3rd International Workshop on Applications of the Reed-Muller Expansion in Circuit Design (Reed Muller 97) Oxford, UK require exponential time, if they are carried out without a cache which memorizes already computed results. To look up already computed OBDDs in the cache, we always require the execution of the equivalence test. For this reason it is very important to have a fast equivalence test.

The recently discovered deterministic equivalence test based on a minimization algorithm introduced in [Waa97] can easily be adapted to $\oplus$-OBDDs, but it performs only in high polynomial degree execution time. Hence, this equivalence test seems not to be suitable for practical purposes. In [GM93b] a fast probabilistic equivalence test for $\oplus$-OBDDs of linear execution time has been proposed. This equivalence test is based on the probabilistic equivalence check of [BCW80]. If two $\oplus$-OBDDs represent the same Boolean function, the probabilistic equivalence test will always give the correct answer. On the other hand, if two $\oplus$-OBDDs which represent different functions are compared, the equivalence test answers with a certain error probability. Therefor, we decided to execute a case study working with this fast probabilistic equivalence test to decide whether it can be used for practical work.

In order to optimize the $\oplus$-OBDD representation of a given Boolean function, we tried to perform reductions on the $\oplus$-OBDD based on the probabilistic equivalence test. Synthesis of a $\oplus$-OBDD from a given circuit representation of a Boolean function is done with a variation of the standard ITE algorithm. This modified ITE algorithm, denoted as ITE-$\oplus$, performs an EXOR operation on two $\oplus$-OBDDs by introducing a new $\oplus$-node with the two $\oplus$-OBDDs as successors. But for circuit descriptions not containing EXOR gates, respectively EQU gates, we had to find a way to introduce $\oplus$-nodes into the $\oplus$-OBDD representation. In this case we can use an apply algorithm based on the positive or negative Davio expansion of the Boolean function to be computed.

Our results, however, had to be certified because even if we have only a very small error probability when testing a single equivalence, the repeated application of the probabilistic equivalence test amplifies the overall error. Thus, we checked the correctness of our synthesized $\oplus$-OBDDs by translating them into a circuit description and verifying these circuit descriptions against the circuit specifications with a standard synthesis tool (SIS).

The paper is structured as follows: In Section 2, we recall some basic definitions concerning $\oplus$-OBDDs. In Section 3, we present the probabilistic equivalence test based on Boolean signatures. Section 4 deals with reducing $\oplus$-OBDDs and in Section 5, we introduce the ITE-$\oplus$ algorithm. Section 6 concludes the paper with experimental results.

2 $\oplus$-OBDDs and Some Basic Facts

A $\oplus$-OBDD P over a set $X_n = \{x_1, \dots, x_n\}$ of Boolean variables is a directed acyclic connected graph $P = (V, E)$. V is the set of nodes, consisting of nonterminal nodes of outdegree 2 and of terminal nodes with outdegree 0. There is a distinguished nonterminal node, the *source/root*, which, as only node, has the indegree 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 $f = (f_1, \dots, f_m)$, $f_i : \mathbb{B}^n \to \mathbb{B}$.

The terminal nodes, the *0-sink* and the *1-sink*, are labeled with the Boolean constants 0 and 1, respectively. The remaining nodes are either labeled with Boolean variables $x_i \in X_n$, denoted as *branching nodes*, or with the binary Boolean function $\oplus$ (EXOR), denoted as $\oplus$-*nodes*. 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.

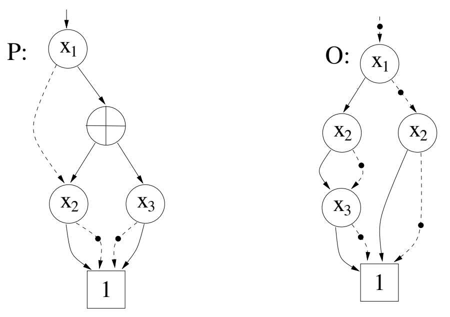
$E \subseteq V \times V$ denotes the set of edges. The two edges starting from a branching node v are labeled with 0 and 1. The *0(1)-successor* of a node v is denoted by $v0(v1)$. There is a permutation σ on the variable indices which defines an order $x_{\sigma(1)} < x_{\sigma(2)} < \ldots < x_{\sigma(n)}$ on the set of input variables. If w is a successor of v in P labeled by $l(v), l(w) \in X_n$, then $l(v) < l(w)$ according to σ. On each path, every variable must occur at most once.

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 sinks 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 labeled with the Boolean variable x_i ($l(v) \in X_n$), then v is associated with δ_{a_i}.
- If v is a $\oplus$-node labeled with the Boolean function $\oplus$ ($l(v) = \oplus$), 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 [MB88] to acquire a 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, both computing f_P*

For illustrating the concept of $\oplus$-OBDDs see Figure 1. Let $\mathbb{B}_3 = \{\{0, 1\}^3 \rightarrow \{0, 1\}\}$ and let the function $f_P \in \mathbb{B}_3$ 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 refers to the edge labeled by 0. A dot on an edge denotes that it is complemented.

Since OBDDs are special cases of $\oplus$-OBDDs (namely $\oplus$-OBDDs without $\oplus$-nodes), for each variable ordering each Boolean function can be represented by means of a $\oplus$-OBDD.

Fact 1: [GM93b] *Let π be an ordering on X_n. Each Boolean function over X_n can be represented by means of a $\oplus$-OBDD that tests variables according to π.*

Fact 2: [GM93b] *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.*

Fact 2 is proven by an example: The *hidden weighted bit function* HWB is defined as follows: If $wt(x)$ is the number of ones (the "weight") in the input assignment $x = (x_1, \ldots, x_n) \in X_n$ and if, for simplicity, x_0 denotes 0, then HWB is defined by $\mathrm{HWB}(x) = x_{wt(x)}$. In [Bry91] it has been proven that each OBDD representation of the HWB is of exponential size, but its $\oplus$-OBDD representation is of cubic size [GM96]: The equality $\mathrm{HBW}(x) = \bigoplus_{k=1}^{n} x_k \wedge E_k(X)$ where $E_k(x)$ equals one if x contains exactly k ones can be verified easily. Since, for each variable ordering, $x_k \wedge E_k(x)$ 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, an efficient algorithm is needed which performs the application of a binary Boolean operation on two $\oplus$-OBDDs.

Fact 3: [GM93a] *Let R and Q be two $\oplus$-OBDDs of the variable ordering π and let $*$ be a binary Boolean operation. Then a $\oplus$-OBDD P w.r.t. the variable ordering π for $f_P = f_R * f_Q$ can be constructed in time $O(size(R) \cdot size(Q))$. If $* \in \{\oplus, \equiv\}$, then the resulting $\oplus$-OBDD can be constructed in constant time by creating a new $\oplus$-node and connecting it with R and Q.*

$\oplus$-OBDDs do not provide a canonical representation of Boolean functions. Besides binary synthesis, the equivalence test is one of the basic tasks of Boolean function manipulation. Therefor, a feasible equivalence test for $\oplus$-OBDDs is also necessary if we want to use them for practical purposes.

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 labeled by an element of Ω [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. According to [GM93a], the equivalence test for Ω-OBDDs, $\Omega \in \{\{\vee\}, \{\wedge\}, \{\vee, \wedge\}\}$ is **co-NP**-complete. The situation is different for $\Omega = \{\oplus\}$. There, the determination of equivalence is within **co-R**.

Recently, a deterministic equivalence test for a $\oplus$-OBDD-like data structure was introduced [Waa97] that can easily be adapted for our model. The main idea behind this equivalence test is the following: 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., as a 2^n-dimensional vector space with an additional multiplication operation. The product of two Boolean functions corresponds to element-wise conjunction and the sum to element-wise EXOR. $\oplus$-OBDDs, representing Boolean functions, are corresponding to vectors in this vector space. An algorithm is given which computes a $\oplus$-OBDD-like data structure with a minimum number of nodes for the given Boolean function as a linear combination of bases in the given vector space according to a certain variable order. This minimization can be done by solving linear equation systems using Gaussian elimination, and therefor needs execution time at most cubic in the number of nodes.

We can use this minimization algorithm for an equivalence test: Let P_1 and P_2 be two $\oplus$-OBDDs representing the Boolean functions $f_{P_1}, f_{P_2} \in \mathbb{B}_n$. If we compute $f_{P_1} \oplus f_{P_2}$ and minimize the resulting $\oplus$-OBDD according to the given algorithm, we must obtain the *0-sink* if the two $\oplus$-OBDDs are equal.

During the process of Boolean synthesis, we need the equivalence test to determine whether a computation has already been done or a node to be created does already exist. This task is managed by looking up some cache data structures containing the already computed results. The deterministic equivalence test mentioned above is too time expensive to be executed each time we are doing a cache lookup. Therefor, we are in need of a fast equivalence test.

The probabilistic equivalence test for $\oplus$-OBDDs proposed in [GM93a] depends on merely linear many arithmetic operations. 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 arithmetics on the $\oplus$-OBDD in terms of a polynomial 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. Further, to represent $\oplus$-OBDDs as polynomials, we need $p_{F_1 \vee F_2} = p_{F_1} + p_{F_2} - p_{F_1} p_{F_2}$ and $p_{F_1 \oplus F_2} = p_{F_1} + p_{F_2} - 2 p_{F_1} p_{F_2}$ to transform the binary Boolean operations EXOR and OR.

Let $GF(2^m)$, $m \in \mathbb{N}$, $m > (\log n) + 1$, denote a Galois field with 2^m elements of characteristic 2. If we consider the elements of $GF(2^m)$ as bit vectors of length m, then addition can be performed by bitwise EXOR.

Let P be a $\oplus$-OBDD. With each node v of P we associate the polynomial p_v : $(GF(2^m))^n \to GF(2^m)$

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

The polynomial associated with the $\oplus$-OBDD P is the polynomial associated with the source v_0 of P. Note that the degree of the evaluated polynomial is always less or equal n and that the polynomial remains unchanged for different representations P of the same Boolean function.

Now let P and Q be two $\oplus$-OBDDs and $a_1, \ldots, a_n \in GF(2^m)$ are generated independently and uniformly random. According to [GM93a], the following can be proven:

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

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 values associated with the function f_P, computed by the $\oplus$-OBDD P, are called Boolean signatures. Instead of $GF(2^m)$, any field $\mathbb{Z}_p$ for a prime $p \geq 2n$ can be chosen. As usual, the error probability can be arbitrarily reduced by increasing m or simply by just using several signatures per node.

According to [BCW80, Bra92], the probability of degeneracy in BDD synthesis based on signatures, i.e., the signatures for two BDDs representing different Boolean functions being equal, is at most $\frac{N^2 \cdot V^S}{2 \cdot P^S}$, where N is the number of nodes, V the number of variables, P the number of elements in the finite field, and S the number of used signatures. Using this formula one gains a maximum probability of degeneracy of $5.05 \cdot 10^{-7}$ when using an array of three signatures over $\mathbb{Z}_p$ for $p = 2^{31} - 1$ with a BDD of 10^8 Nodes depending on 100 Boolean variables.

4 Reductions on $\oplus$-OBDDs

In general, $\oplus$-OBDDs can be reduced in the same manner as OBDDs. In a $\oplus$-OBDD a branching node labeled by $x \in X_n$ is redundant if both of its edges point to the same node. Such nodes can be replaced by reconnecting all incoming edges to their successor. This reduction rule is called simple reduction or *deletion rule*. Identification of isomorphic subgraphs forms the second reduction rule called algebraic reduction or *merging rule* (refer to Figure 2).

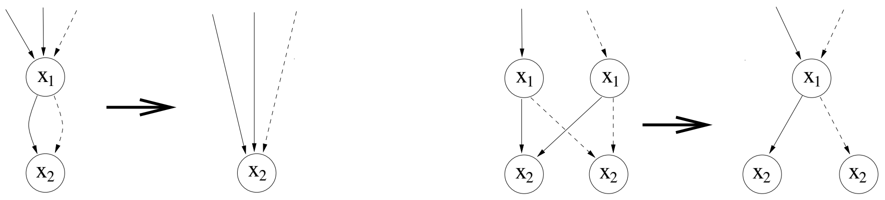


Figure 2: *Deletion rule and merging rule for branching nodes.*

Moreover, in the case of $\oplus$-OBDDs we need reduction rules for $\oplus$-nodes. For a $\oplus$-node v *deletion rule* and *merging rule* are applied in the same way as for branching nodes, except that for the *deletion rule* $\oplus$-nodes with two equal successors are substituted by the *0-sink* (see Figure 3). We also have to consider the case that one successor of a $\oplus$-node is a terminal

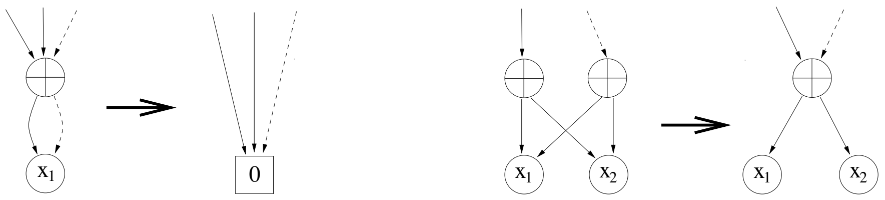


Figure 3: *Deletion rule and merging rule for $\oplus$-nodes.*

node. If the *0-sink* is a successor of the $\oplus$-node, then the $\oplus$-node is replaced by its second successor. On the other hand, if the *1-sink* is a successor of the $\oplus$-node, then the $\oplus$-node is replaced by a complemented edge pointing to its second successor (refer to Figure 4). Considering complemented edges we are able to extend the reduction rules further. Hence,

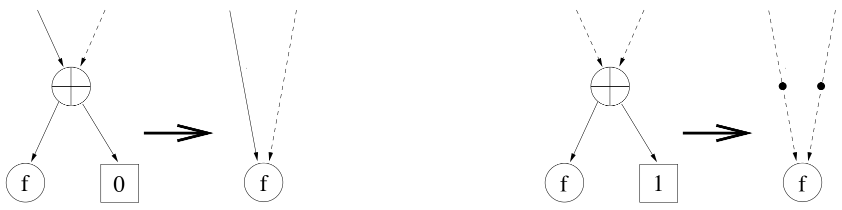


Figure 4: *Reduction rules for $\oplus$-nodes connected with terminal nodes.*

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 5).

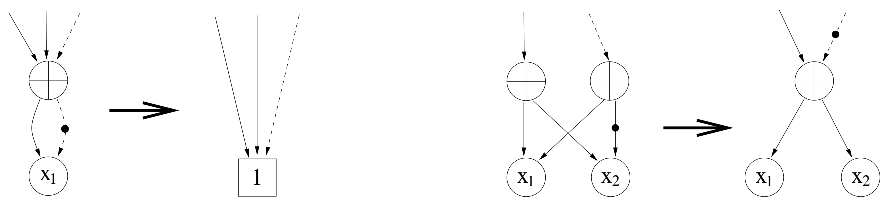


Figure 5: *Deletion rule and merging rule for $\oplus$-nodes with complemented edges.*

If we use complemented edges, we have to extend our considerations concerning the avoidance of additional ambiguity to ensure a more efficient usage of the caches needed in synthesis operations (see Section 5). Like with OBDDs, in $\oplus$-OBDDs an edge leading from node v labeled by a Boolean variable $x_i \in X_n$ to the *1-successor* $v1$ must not be negated. Considering a node w labeled by $\oplus$, we state that only one of the two successors of w, which we denote as w_l, the left successor and w_r, the right successor, may be negated. Hence, we define the left successor w_l always to be not negated. To maintain this rule, we use the equivalences shown in Figure 6.

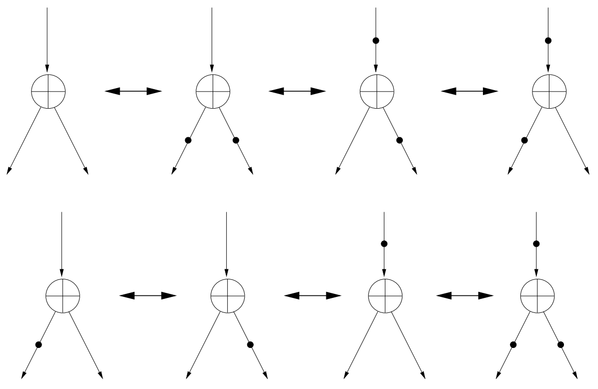


Figure 6: *Equivalences for $\oplus$-nodes and complemented edges.*

5 The ITE-$\oplus$ Algorithm

The synthesis algorithm for $\oplus$-OBDDs proposed in [GM96] shows that applying a binary Boolean operation on two $\oplus$-OBDDs requires at most quadratic time. In convenient OBDD implementations, all Boolean operations are implemented by means of the ITE operator [BRB90].

Hence, we extend the standard ITE algorithm in the following way to the ITE-$\oplus$ algorithm. The ITE-$\oplus$ algorithm computes every Boolean operation except EXOR and EQU in the same way as the standard ITE algorithm. To perform logical EXOR of two $\oplus$-OBDDs, i.e. ITE$(f, g, \overline{g})$, we simply introduce a new $\oplus$-node and connect it with the two $\oplus$-OBDDs. For the computation of the EQU of two $\oplus$-OBDDs, we take the complement computed by the EXOR operation.

As usual, 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 recall the results of ITE-$\oplus(f, g, h)$. Before a new node is created, we always refer to 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 to a standard form by reordering it and checking the constraints for complemented edges. Refer to Figure 7 for a brief description of the ITE-$\oplus$ algorithm in pseudocode.

```
function  ite-⊕(f,g,h,result)
begin
  transform_to_standard_triple(f,g,h);
  if  terminal_case(f,g,h,result)
  then
    return result;
  reorder_triple_acc_to_variable_order(f,g,h);
  check_rules_for_complemented_edges(f,g,h);
  if  in_computed_table(f,g,h,result)
  then
    return  result;
  if f=⊕
  then
    result=new_node(label=⊕,then=g,else=h);
  else
    begin
      $r_1$=ite-⊕($f|_{\nu=1}$, $g|_{\nu=1}$, $h|_{\nu=1}$);
      $r_0$=ite-⊕($f|_{\nu=0}$, $g|_{\nu=0}$, $h|_{\nu=0}$);
      if signature($r_1$)=signature($r_2$)
      then
        result = $r_1$;
      else
        result=new_node(label=$\nu$,then=$r_1$,else=$r_0$);
      insert_in_computed_table(f,g,h,result);
    end,
  find_or_add_in_unique_table(result);
  return result;
end.
```

Figure 7: *Modified ITE algorithm for $\oplus$-OBDD synthesis.*

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 labeled by the variable x_i (see Figure 8).

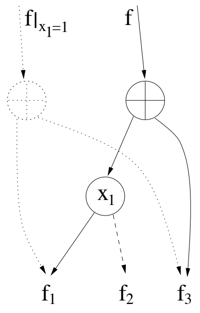


Figure 8: *Cofactor creation $f|_{x=1}$ in $\oplus$-OBDD P with $l(source_P)=\oplus$.*

6 Experimental Results

We applied our $\oplus$-OBDD synthesis tool on some ISCAS85 [BF85] benchmark circuits. Because of the possible error in the execution of the equivalence test we had to determine that the obtained $\oplus$-OBDDs are correct. In order to do this, we transformed the synthesized $\oplus$-OBDD into a BLIF (Berkeley Logic Interchange Format) circuit description file, simply by substituting each node labeled by $x \in X_n$ with an according multiplexer subcircuit and all $\oplus$-nodes with a description of the EXOR function. Then this circuit description file was verified against the BLIF version of the circuit's specification file. This check was done by the SIS standard synthesis tool [SSM$^+$92].

However, for the circuits c880 and c6288, the constructed $\oplus$-OBDD could not be verified with SIS due to memory limitations. The $\oplus$-OBDD for the 16th bit of the c6288 multiplier consisting of more than 700.000 nodes was tested for correctness by comparing expected result and actual result for all input assignments and was proven correct.

We started our experiments with appropriate variable orders obtained by heuristics. Actually, we use fanin heuristics according to [MWBSV88], weight propagation heuristics introduced by [MIY90], and, for comparison, also the original order in which the variables occur in the circuit description.

At first, circuits containing EXOR gates were under consideration, namely c432 and c499. With weight propagation heuristic the ratio between $\oplus$-OBDD-size and OBDD-size was about 0.3 for c432 and c499 (see Table 1).

For the circuits not containing EXOR gates nor EQU gates, the ITE-$\oplus$ algorithm creates only an OBDD. Therefor, we have to introduce $\oplus$-nodes into the $\oplus$-OBDD synthesis process. One possible solution is to consider alternative function decompositions that introduce new $\oplus$-nodes into the $\oplus$-OBDD:

- positive Davio expansion: $f = f|_{x=0} \oplus x(f|_{x=1} \oplus f|_{x=0})$
- negative Davio expansion: $f = f|_{x=1} \oplus \overline{x}(f|_{x=1} \oplus f|_{x=0})$

If we want to perform a binary Boolean operation $*$ on two $\oplus$-OBDDs according to the positive Davio expansion, we introduce two new $\oplus$-nodes: The first $\oplus$-node is connected with the $\oplus$-OBDDs computed by applying $*$ to the cofactors. The second $\oplus$-node is connected with the $\oplus$-OBDDs computed by applying $*$ to the negative cofactors and the product of the variable x and the other new $\oplus$-node:

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

Up to now, we use either positive Davio expansion, negative Davio expansion, or Boole-Shannon expansion for $\oplus$-OBDD creation. Thus, the $\oplus$-OBDD depends on a single expansion type. We are planning to use a heuristic that decides during the synthesis step what expansion to take for each synthesis step.

In Table 1, in the column *heuristics*, fanin heuristics is denoted as *fanin*, weight propagation as *wp*, and *index* means original order. The column *decomp* denotes the functional decomposition we use. For all tables, Boole-Shannon expansion associated to the *ITE* operator is denoted as *ite*, and positive and negative Davio expansion are denoted as *pDE* and *nDE*. The column Δ-ratio stands for the quotient of $\oplus$-OBDD-size and OBDD-size.

Table 1 presents those results for which we obtained the best ratio between OBDD-size and $\oplus$-OBDD-size. In Table 2 - 4, all obtained results are listed depending on the heuristics used to create the variable order. For Table 2, fanin heuristics is used, for Table 3, weight propagation heuristics, and for Table 4, the original order of the circuit description file. A bar in a cell of a table denotes that the computation exceeded the memory limitations, which were chosen rather low to allow a larger number of experiments within the intended timeframe.

For c6288, we could only synthesize partial circuits up to the 16th bit of the multiplication. The $\oplus$-OBDD-size was about 10% greater than the corresponding OBDD-size. We introduced EQU-gates into the circuit description of c6288 by substituting complexes consisting of four NOR-gates, representing a logical equivalence [Bry88]. But, the synthesized

$\oplus$-OBDDs still were always greater than their OBDD counterparts. A possible explanation of this result is that the used order is not well adapted for $\oplus$-OBDDs.

As a synopsis of the case study we can state that efficient manipulation of $\oplus$-OBDDs by means of signatures is possible, because the error probability according to our verified results seems to be quite minor to effect $\oplus$-OBDDs up to a reasonable number of nodes. Actually, three signatures of 32 bit size were sufficient for all experiments we performed, where up to 1 million intermediate nodes were created for a single $\oplus$-OBDD. Up to now, we are not able to use dynamic reordering for $\oplus$-OBDDs because it is not implemented yet. We are going to implement variable reordering, which can be performed as for OBDDs. The only difference is to avoid the swap of adjacent variables with $\oplus$-nodes in between. We can solve this problem by moving the according $\oplus$-nodes up or down the computation path until the variables to be swapped are directly adjacent. Hence, the next step will be to implement dynamic reordering to draw a direct comparison of the $\oplus$-OBDD representation and the OBDD representation.

circuit	**OBDD-size**	**$\oplus$-OBDD-size**	**Δ-ratio**	**heuristics**	**decomp**
c432	89338	**33475**	0.37	wp	ITE
c499	45865	**10240**	0.23	fanin	ITE
c880	30548	**20438**	0.71	wp	pDE
c1355	119201	**13299**	0.11	wp	nDE
c1908	36007	**16333**	0.45	index	nDE
c5315	40306	**7798**	0.19	wp	nDE
c6288.6123gat	686691	**747860**	1.09	fanin	ITE

Table 1: *Comparison of OBDD-size and $\oplus$-OBDD-size for synthesis of some ISCAS85 benchmark circuits (best results).*

circuit	**OBDD-size**	**$\oplus$-OBDD-size**		
		ITE	**pDE**	**nDE**
c432	29864	34011	81923	86515
c499	45865	10240	11609	11609
c880	8025	(*)	73369	64792
c1355	45865	(*)	-	-
c1908	11569	(*)	27183	17734
c5315	32842	(*)	31489	33106
c6288.6123gat	686691	747860	-	-

Table 2: *Comparison of OBDD-size and $\oplus$-OBDD-size for variable orders created by fanin heuristic.*

circuit	**OBDD-size**	**$\oplus$-OBDD-size**		
		ITE	**pDE**	**nDE**
c432	89338	33457	72125	105266
c499	36862	9493	13025	12995
c880	30548	(*)	20438	20635
c1355	119201	(*)	13480	13299
c1908	39373	(*)	20770	28148
c5315	40306	(*)	13244	7798
c6288.6123gat	-	-	-	-

Table 3: *Comparison of OBDD-size and $\oplus$-OBDD-size for variable orders created by weight propagation heuristic.*

*$\oplus$-OBDD-size equals OBDD-size for circuits not containing EXOR(EQU)-gates if Boole-Shannon expansion and ITE-$\oplus$ algorithm is used.

circuit	OBDD-size	⊕-OBDD-size		
		ITE	pDE	nDE
c432	1733	2114	4745	6266
c499	45922	12800	13703	13698
c880	346660	(*)	284187	399071
c1355	45922	(*)	14197	14392
c1908	36007	(*)	17347	16333
c5315	-	(*)	66376	73749
c6288.6123gat	-	-	-	-

Table 4: *Comparison of OBDD-size and ⊕-OBDD-size for variable orders chosen as variables occur in circuit description.*

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.

[BF85] F. Brglez and H. Fujiwara. A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in FORTRAN. In *Proc. IEEE Int. Symp. on Circ. Syst. (ISCAS)*, pages 695–698, 1985.

[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.

[Bry88] David Bryan. The ISCAS '85 Benchmark Circuits and Netlist Format. Technical report, MCNC - Microelectronic Center of North Carolina, 1988.

[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.

[GM93a] 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.

[GM93b] Jordan Gergov and Christoph Meinel. Mod2-OBDDs: A Generalization of OBDDs and EXOR-Sum-of-Products. In *Proc. IFIP WG 10.5 Workshop on the Application of the Reed-Muller Expansion in Circuit Design*, TR WSI-93-2, W. Schickard-Institut für Informatik, Universität Tübingen, pages 170–175, 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.

[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 T heir Computational Power*, volume 370 of *LNCS*. Springer Verlag, Heidelberg, 1989.

[MIY90] Shin-ichi Minato, Nagisa Ishiura, and Shuzo Yajima. Shared Binary Decision Diagrams with Attributed Edges for Efficient Boolean Function Manipulation. In *Proc. of the 27th ACM/IEEE Design Automation Conference*, pages 52–57, 1990.

[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.

[MWBSV88] S. Malik, A.R. Wang, R.K. Brayton, and A. Sangiovanni-Vincentelli. Logic Verification Using Binary Desicion Diagrams in a Logic Synthesis Environment. In *Proc. of the 25th Design Automation Conference*, pages 268–271, 1988.

[SSM$^+$92] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli. Sequential Circuit Design Using Synthesis and Optimization. In *Proc. of the Int. Conf. on Computer Design*, pages 328–333, Cambridge, MA, 1992.

[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.