published in
Proceedings of 17th NORCHIP '99 Conference,
Oslo, Norway 1999, pp. 52-60.

Overcoming Ordering Restrictions for Synthesizing Binary Decision Diagrams

Christoph Meinel, Harald Sack, Christan Stangier

FB IV - Informatik, Universität Trier
D-54286 Trier, Germany
phone: ++49 (651) 201-2093
fax: ++49 (651) 201-3954
email: {meinel,sack,stangier}@uni-trier.de

Abstract

Many problems in computer-aided design of highly integrated circuits (CAD for VLSI) can be transformed to the task of manipulating objects over finite domains. The efficiency of these operations depends substantially on the chosen data structures. In the last years, ordered Binary Decision Diagrams (OBDDs) have proved to be very efficient in this context. One restriction for the efficient synthesis of two or more OBDDs is that they must share a common variable order. We address the problem of efficiently synthesizing OBDDs that have different variable orders by adapting a reordering based synthesis technique. In particular, we apply this technique to the basic operation of the irredundant cover problem, i.e. testing functional containment of two OBDDs.

1 Introduction

A central problem in the design of CAD (Computer-Aided Design) systems for VLSI (Very Large Scale Integration) circuits is to represent the functional behavior of a circuit. For the solution of problems like combinational circuit verification, where it is considered whether a combinational circuit C satisfies a given specification S, computer-internal representations of C and S have to be determined that can be used to test the relevant properties. During the last decade *Reduced Ordered Binary Decision Diagrams* (ROBDDs or, for short OBDDs) introduced by Bryant [Bry86] have become a successful data structure in this context (for a complete survey see [MT98]).

The switching behavior of digital circuits can be described in terms of Boolean functions $f : \{0,1\}^n \to \{0,1\}$ and by introducing a suitable 0-1-encoding almost all finite problems can be modeled. The great importance of Boolean functions stems from the possibility to obtain substantially simplified and optimized circuits by applying optimization techniques during the design process. But, before applying optimization techniques, Boolean functions themselves have to be represented as efficient as possible in computer memory. Most conventional representations of Boolean functions like circuit netlists or truth tables have serious disadvantages, like not being compact, or insurmountable problems regarding the algorithmic handling. In contrast to descriptive conventional representations based on computation rules, OBDDs are based on a decision process. Their success is due to the fact that OBDDs are a canonical representation of Boolean functions (thus, making the test of functional properties as satisfiability or equivalence straightforward) and that they are compact for most Boolean functions of practical importance.

The variable order chosen for the representation of a Boolean function as an OBDD plays a very important role, since the size of the OBDD representation for a function may vary exponentially with different orderings. However, the applicability of OBDDs depends heavily on the size of the representation and therefore on the chosen variable order.

The problem of finding an optimal variable order is known to be NP-hard [BW96]. Hence, much effort is spent on developing heuristics that compute acceptable orders. We distinguish two different types of heuristics: *static heuristics*, i.e. heuristics that choose a variable order for a given Boolean function represented as a circuit by analysis of the circuit topology [MWB88, FOH93], and *dynamic heuristics*, i.e. heuristics that change the variable order dynamically during the construction of the OBDD, based on the exchange of adjacent variables [Rud93].

In this paper we address the problem how to work with OBDDs of different variable orders, especially w.r.t. the application of a binary Boolean operator to two OBDDs.

Even though in [FHS78] an efficient algorithm for the equivalence test of Free BDDs, i.e. BDDs without the ordering constraint is given, this algorithm cannot be used for computing arbitrary Boolean operations. We address the similar problem by working with OBDDs of different variable orders, but we extend it by providing a constructive algorithm for the application of an arbitrary Boolean operation.

In a naive approach one may first reorder the involved OBDDs to a common variable order and then carry out the operation to be applied. The major problem with this method is the fact that for some OBDDs it is not possible to transform them to a common variable order within the given resource limitations. Instead of doing the computation in separate steps we incorporate the reordering procedure completely into the synthesis process.

To do this, we use an additional variable, called the *operator variable* representing a binary Boolean operation to be computed. An operator node is introduced at the top level of the resulting OBDD and connected to the two OBDDs to be applied to. By exchanging adjacent variables the operator node can be moved down the OBDD towards the terminal nodes until a final shortcut operation can be applied and the operator node disappears. The noticeable property of this approach is that the two OBDDs to be connected need not to share a common variable order.

In another context, an approach called Reordering Based Synthesis, based on the same principles has been proposed by [HDB97]. The purpose of this method was to substitute the conventional recursive synthesis algorithm for OBDDs by a sequential one. But, this technique can only be used for OBDDs sharing a common variable order.

A possible application of our new algorithm is, e.g. the determination of the irredundant cover, a problem occurring in two level logic minimization. For the programming of PLA elements (*Programmable Logic Arrays*) the Boolean functions to be represented are given in form of two-level logic expressions. To develop an efficient realization as a PLA, the two-level logic expression can be minimized by utilizing an irredundant cover determined by prime implicants [BHMS84]. To find a minimal irredundant cover, the Boolean functions represented by prime implicants are tested for containment. Containment can easily be determined by applying the Boolean implication to the functions involved.

The experiments performed up to now are rather artificial, but allow a good insight into the behavior of the proposed algorithm. We chose pairs of Boolean functions represented by circuit descriptions out of a common benchmark set. We constructed the OBDDs for each circuit and optimized them separately by common reordering strategies. Because the circuits have almost nothing in common, the variable orders obtained by reordering heuristics are rather different. Finally, we tested these OBDDs for containment.

Besides irredundant cover computation, there is a wide range of applications for this algorithm. Starting with sequential analysis of digital circuits where the transition relation and the set of reachable states have different variable ordering restrictions, to the more efficient usage of partitioned OBDDs (POBDDs) [NJF+96], where the single partitions of an POBDD also have different variable orderings.

The paper is structured as follows: In Section 2, we give some basic definitions concerning OBDDs, different reordering, and synthesis procedures. In Section 3, we introduce our new algorithm. Section 4 deals with first experimental results and Section 5 concludes the paper with an outlook on future work.

2 Basic Definitions

2.1 OBDDs

An *Ordered Binary Decision Diagram* (OBDD) P for a Boolean function $f : \mathcal{B}^n \to \mathcal{B}$ over the variables $X_n = \{x_1, \ldots, x_n\}$ is a directed acyclic graph consisting of *inner nodes* labeled with variables from X_n and *sinks* labeled by the Boolean constants 1 and 0. Each inner node has two outgoing edges: the 1-edge and the 0-edge. The OBDD has a starting node called its *root*. The computation of $f(a_1, \ldots, a_n)$ $a_i \in \{0, 1\}$ follows a path from the root to a sink, where on a node labeled with x_i the input bit a_i is tested. If $a_i = 1$, the path follows the 1-edge, otherwise the 0-edge. The value of the reached sink determines $f(a_1, \ldots, a_n)$. Each variable occurs at most once on a path from the root to the sink. The variables on a path respect a given order, which is a permutation π on the variable indices so that $x_{\pi(1)} < x_{\pi(2)} < \ldots < x_{\pi(n)}$. For each edge leading from a node labeled by $x_{\pi(i)}$ to a node labeled with $x_{\pi(j)}$ the proposition $i < j$ holds.

To deal with Boolean functions $f : \mathcal{B}^n \to \mathcal{B}^m$, we consider *multi-rooted shared* OBDDs by introducing multiple roots into a single OBDD, with each root representing a subfunction of $f = (f_1, \ldots, f_m), f_i \in \mathcal{B}^k \to \mathcal{B}, k \leq n$.

All important operations on Boolean functions usually required in synthesis and verification have efficient OBDD algorithms. Due to the fact that there are 2^{2^n} functions on n variables, many functions have an OBDD representation of exponential size, but most functions of practical relevance can be represented with small or moderate OBDD sizes.

The size of an OBDD depends substantially on the variable order. Therefore, one may have an exponential improvement in size by choosing a good variable order.

2.2 OBDD Synthesis

Typically synthesis of OBDDs is based on a recursive algorithm that expands the involved Boolean functions. The ternary *if-then-else*-operator (ITE) [BRB90] forms the core of the recursion based synthesis operation (**if** F is true, **then** G, **else** H).

$$ite(F, G, H) = F \cdot G + \overline{F} \cdot H$$

With ITE it is possible to implement all binary Boolean operations. The implementation of ITE relies on the *Boole-Shannon*-expansion of the function under consideration w.r.t. the leading variable x in the variable order π. Thus ITE can be implemented as

$$ite(F, G, H) = (x, ite(F_x, G_x, H_x), ite(F_{\overline{x}}, G_{\overline{x}}, H_{\overline{x}})),$$

where F_x and $F_{\overline{x}}$ denote F evaluated at $x = 1$ and $x = 0$, respectively. Time and space complexity is bounded by $O(|F| \cdot |G| \cdot |H|)$. Considering binary Boolean operations, only two operands are nonterminals and therefore, the complexity of a binary operation is bounded by $O(|F| \cdot |G|)$.

Alternatively, the Reordering Based Synthesis method depends on the ongoing exchange of variables in adjacent levels of the underlying OBDDs. Since the exchange of neighbored variables is a local operation [Rud93], it can be implemented very efficiently. Suppose we have the two OBDDs F and G with sizes $|F|$ and $|G|$. For computing the OBDD $H = F \otimes G$, $\otimes$ being an arbitrary binary Boolean operation, a new operator variable c is introduced into the OBDD and $H' = \overline{c} \cdot F + c \cdot G$ is constructed. By the level exchange operation the variable c can be moved downwards within the OBDD. If, after a level exchange operation one of the two successors of the operator variable is a terminal node, we replace the operator variable and its two successors by the result obtained in the application of the operator $\otimes$ to the terminal node and the remaining node. Thus, after swapping the operator variable down to the sink, all operator nodes can be replaced.

The advantage of the Reordering Based Synthesis method is its nonrecursive execution, i.e. the synthesis operation can be interrupted at any time and reordering of the other variables can be done. On the other hand the nonrecursive approach result in smaller memory peak sizes, since less intermediate results have to be stored.

3 Including Synthesis into the Reordering Process

Let us assume we have two Boolean functions $f, g : \mathcal{B}^n \to \mathcal{B}^m$ over the Boolean variables $\{x_1, \ldots, x_n\}$. Let us further suppose that f and g are represented by OBDDs P_f and P_g and that P_f has the variable order π_f and P_g the variable order π_g.

Let $\otimes$ be a arbitrary binary operator and h the result of the operation $f \otimes g$, with π_h being the variable order of the OBDD P_h representing h. To perform this computation with a regular synthesis algorithm P_f and P_g have to share a common variable order.

In our approach we introduce an operator variable c, representing the binary Boolean operation $\otimes$ and connect it with P_f and P_g. Since P_f and P_g have different variable orders π_f and π_g, we have to take care that for the level exchange operation the variables in the level directly below the operator variable contain the same variable. To achieve this, we change the order of the two OBDDs separately by placing a common variable below the operator variable while leaving the rest of π_f and π_g unchanged. The variable that is placed next to the operator variable is chosen by a simple greedy heuristic that selects the variable that requires the least number of level exchange operations to be put in the designated place.

Next, we perform a regular level exchange operation of the operator variable and its successor variable (see Figure 1).

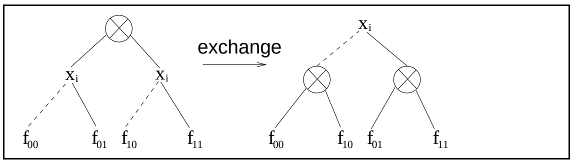

Figure 1: *Exchange of Operator Variable with Regular Variable*

If one of the sub-OBDDs $f00$, $f01$, $f10$, or $f11$ is a terminal node, we can immediately apply the operation represented by the operator node and replace the operator node by the resulting OBDD (see Figure 2). The algorithm continues until the operator variable

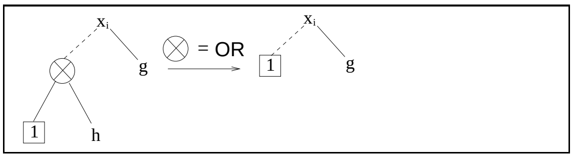

Figure 2: *Possible Terminal Case for logical OR*

reaches the last position in the variable order (see Figure 3). At this point all operator

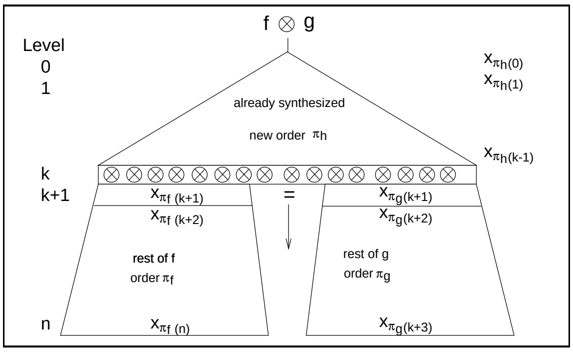

Figure 3: *The Operator Level k Proceeds Downwards to the Terminal Nodes*

nodes can be replaced. Note, that the OBDD that is created by this algorithm is not necessarily reduced. Therefore, a reduction algorithm has to be applied that restores the canonical form of the synthesized OBDD.

4 Experimental Results

For testing our new algorithm we have chosen a simple containment test as it is included e.g. in the determination of the irredundant cover of a Boolean function. We chose an arbitrary number of circuits out of the LGSynth91 [LGS] benchmark suite. From this set of circuits we took pairs that are of similar OBDD size and have a comparable number of inputs, since none of the two circuits should dominate the other.

The OBDDs representing these circuits were constructed and optimized separately. Since the two circuits to be combined usually have not very much in common, they have quite different variable ordering requirements. Now, we computed the logical implication of the outputs of the participating circuits.

This was done with our new algorithm using the implication as the operator and also, for comparison, tested against the naive approach, i.e. representing both the OBDDs to be combined with the same variable order and then computing the implication by calling a standard recursive synthesis routine. For the naive approach we tried both possible variable orders given by the two OBDDs to be combined.

All experiments were performed on an Intel PentiumPro 200 Workstation running Linux. The computation time was limited to one hour and the memory consumption was also limited to 100 MB.

In Figure 5 we have provided a table showing our experimental results. In the table all numbers indicate final OBDD sizes measured in number of nodes. The table is structured as follows: The first column gives the names of the circuits to be combined. We have performed the implication in both directions, i.e. *circuit1* $\rightarrow$ *circuit2* and *circuit2* $\rightarrow$ *circuit1*. For each direction we have provided three columns. The first column shows the result obtained by the application of our algorithm, the second and the third column show the result obtained by the application of the naive algorithm as described above. In the second column we have transformed the order of the second OBDD to the order of the first one, in the third column we have transformed the order in the other direction.

We have evaluated 9 pairs of circuits selected by the criteria given above. In 12 of the computed 18 implications, our algorithm computed a smaller result. Noticeable is the circuit pair *sbc, alu32*, where our algorithm results in an OBDD with only 659 nodes, while the best result of the naive method gives 12223 nodes. There are also two cases where the result of the naive method could not be completely computed, while our algorithm succeeded.

When evaluating the obtained results we have to keep the following facts in mind: Our algorithm is by far not optimal, i.e. we have not used a proper reordering strategy for the remaining parts of the OBDD during the synthesis operation. We also performed a reduction only at the end of the synthesis operation. By using the reduction algorithm more often we could avoid large memory peak sizes and therefore could obtain a gain in runtime.

The runtime of our algorithm is not given in the table by now, because the implementation of the algorithm is in a rather preliminary state and is not optimized for speed. At the moment, in some cases our algorithm requires up to 3 times the runtime of the native approach.

Despite the fact that the implementation of the algorithm is in a rather early state, we have obtained some interesting results, causing further development of our method to be very promising.

5 Conclusion and Future Work

One of the major advantages of the described algorithm is that it can be interrupted at any time to minimize the remaining parts of the OBDDs involved or the newly created synthesized part. To do this, we have to apply heuristics based on dynamic reordering. Current research is going on towards developing a proper reordering strategy.

It has to be decided carefully, when and how often to apply the reduction algorithm. Each application of this algorithms is time consuming. But if we use the reduction algorithm only very rarely, the size of the synthesized OBDD may grow exponentially.

Further, we do not have to limit ourself to the application of binary Boolean operators. We can use more complex operators like e.g. AND-EXIST that is required for sequential analysis. With the help of our algorithm, reachable state set and transition relation in sequential analysis can be used both with their own variable order when computing the next iteration in the computation of the reachable state set.

Furthermore, we don't have to limit the application of our algorithm only to a pair of OBDDs. We can combine several OBDDs at the same time by connecting them with a tree of operator nodes.

Acknowledgements
We would like to thank Arno Wagner for fruitful discussions and providing us help with the evaluation and preparation of our experimental results.

References

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

[BRB90] K. S. Brace, R. L. Rudell, R. E. Bryant. Efficient implementation of a BDD package, in *Proc of 27th Design Automation Conference*, pp.40–45, 1990

[Bry86] R. E. Bryant, Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Transactions on Computers*, C-35:677–691, 1986.

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

[FHS78] S. Fortune, J. Hopcroft, and E. M. Schmidt. The Complexity of Equivalence and Containment for Free Single Variable Program Schemes, in *ICALP'78 - Automata Languages and Programming*, LNCS 62, pp.227–240, Springer, 1978

[FOH93] H. Fujii, G. Ootomo, and C. Hori. Interleaving based variable ordering methods for ordered binary decision diagrams, *Proc. IEEE/ACM ICCAD '93*, pp. 38-41, 1993

[HDB97] A. Hett, R. Drechsler, and B. Becker. Fast and Efficient Contruction of BDDs by Reordering Based Synthesis, in *Proc. of IEEE ED&TC'97*,pp. 168-175, Paris, 1997

[MB88] 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*,pp.308-313,1988

[LGS] LGSynth91 Benchmarks:
http://www.cbl.ncsu.edu/CBL_Docs/lgs91.html.

[MT98] Ch. Meinel, T. Theobald, Algorithms and Data Structures in VLSI Design: OBDD - Foundations and Applications, Springer, Heidelberg, 1998.

[MWB88] S. Malik, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli, Logic Verification Using Binary Desicion Diagrams in a Logic Synthesis Environment, *Proc. of the 25th ACM/IEEE Design Automation Conference*, pp.268–271, 1988.

[NJF+96] A. Narajan, J. Jain, M. Fujita and A. Sangiovanni–Vincentelli, Partitioned ROBDDs – A Compact, Canonical and Efficient Manipulable Representation of Boolean Functions, in *Proc. IEEE/ACM ICCAD'96*, San Jose, CA, USA 547–554, 1996.

[NIJ+97] A. Narajan, A. Isles, J. Jain, R. K. Brayton and A. Sangiovanni–Vincentelli, Reachability Analysis Using Partitioned–ROBDDs, in *Proc. IEEE/ACM ICCAD'97*, San Jose, CA, USA, 547–554, 1997.

[Rud93] R. Rudell, Dynamic Variable Ordering for Ordered Binary Decision Diagrams, in *Proc. IEEE/ACM ICCAD'93*, 42–47, 1993.

[Som] F. Somenzi, CUDD CU Decision Diagram Package.

	circuit1 $\to$ *circuit2*			*circuit2* $\to$ *circuit1*		
Circuits	rbs	$\pi_1 \to \pi_0$	$\pi_0 \to \pi_1$	rbs	$\pi_1 \to \pi_0$	$\pi_0 \to \pi_1$
C1355, C3540	377310	650477	650477	1376675	650483	650483
C1355, C499	61941	51753	51828	61969	52266	52266
adsb32, alu32	16600	16021	16021	15362	15650	15364
i8, k2	25432	32702	32702	26921	37814	32715
s1269, mm9b	465933	-	896657	493487	898628	899118
sbc, alu32	659	41585	12223	41586	12230	12225
sbcc, alu32r	37214	189502	-	-	189293	-
vda, alu4	1916	5571	5571	2975	5696	5696
too_large, vda	4511	11440	11440	15626	19614	18418

Figure 4: *OBDD-Sizes of Functional Containment*