Preprint from Workshop Notes, International Workshop on Logic Synthesis (IWLS'97),
Tahoe City, California, May 19-21, 1997

Functional Simulation using Binary Decision Diagrams

Christoph Scholl Rolf Drechsler Bernd Becker

Institute of Computer Science, Albert-Ludwigs-University
79110 Freiburg im Breisgau, Germany
email: {scholl,drechsle,becker}@informatik.uni-freiburg.de

Abstract

In many verification techniques fast functional evaluation of a Boolean network is needed. We investigate the idea of using Binary Decision Diagrams (BDDs) for functional simulation. The area-time trade-off that results from different minimization techniques of the BDD is discussed. We propose new minimization methods based on dynamic reordering that allow smaller representations with (nearly) no runtime penalty.

1 Introduction

One of the most important tasks during the construction and design of *Integrated Circuits* (ICs) is the proof of correctness, i.e. the check whether a design fulfills its specification. Simulation is a basic task of many verification tools. Recently, methods based on decision diagrams have been proposed [1, 5] to speed up cycle based functional simulation. Decision diagrams are used to reduce the runtime, which is proportional to the number of logic gates in traditional approaches (like event driven simulation or levelized compiled code simulation), to runtimes which are proportional to the sum of the number of inputs and the number of outputs of the circuit. In [1] a BDD is translated into a C program. The number of operations to evaluate an input vector in the resulting program is very small, but the program has the drawback of being large in size. In practice for such large programs memory bandwidth becomes a problem. An access to memory can take many clock cycles if the requested item resides in a level of the memory hierarchy which is very slow [6, 5]. Therefore in our approach to functional simulation a central problem is to minimize the amount of memory needed and to optimize memory traffic.

In this paper we investigate how the drawback of large simulation programs can be avoided, if we allow the number of operations of the simulator to evaluate an input vector to slightly increase. We apply (restricted) dynamic reordering techniques and study the effect on the trade-off between the average number of operations to evaluate an input vector and the size of the resulting simulator.

2 Preliminaries

In this section we introduce basic notations and definitions that are needed for the understanding of the paper.

2.1 Ordered Binary Decision Diagrams

Each Boolean function $f : \mathbf{B}^n \to \mathbf{B}$ can be represented by a *Binary Decision Diagram* (BDD) [2], i.e. a directed acyclic graph where a Shannon decomposition is carried out in each node.

A BDD is called *ordered* if each variable is encountered at most once on each path from the root to a terminal and if the variables are encountered in the same order on all such paths. A BDD is called *reduced* if it does not contain vertices either with isomorphic sub-graphs or with both edges pointing to the same node.

For functions represented by reduced, ordered BDDs efficient manipulations and evaluations are possible [2]. In the following only reduced, ordered BDDs are considered and for briefness these graphs are called BDDs.

As well-known the choice of the variable ordering largely influences the size of the BDD, i.e. it may vary from linear to exponential.

2.2 Boolean Relations

Each Boolean function $f : \mathbf{B}^n \to \mathbf{B}^m$ can be viewed as a Boolean relation which can be represented by its characteristic function:

Definition 1 *A relation $F \subseteq \{0,1\}^n \times \{0,1\}^m$ is called* Boolean relation *with n inputs and m outputs. Each Boolean function $f : \{0,1\}^n \to \{0,1\}^m$ can be viewed as a Boolean relation $R(f)$ with*

$$(\epsilon, \delta) \in R(f) \Longleftrightarrow f(\epsilon) = \delta (\forall\, \epsilon \in \{0,1\}^n, \delta \in \{0,1\}^m).$$

A Boolean relation F can be represented by its characteristic function, i.e. a Boolean function χ_F with $\chi_F(\epsilon, \delta) = 1$ iff $(\epsilon, \delta) \in F$.

If f has input variables $i_1, \ldots, i_n$ and we introduce additional output variables $o_1, \ldots, o_m$ (for $f_1, \ldots, f_m$ respectively), $\chi_{R(f)}$ can be computed by the following formula [3]:

$$\chi_{R(f)}(i_1, \ldots, i_n, o_1, \ldots, o_m) = \bigwedge_{i=1}^{m} (o_i \equiv f_i(i_1, \ldots, i_n)). \quad (1)$$

3 Functional Simulation

We briefly review previous work and then describe our approach.

3.1 Previous Work

3.1.1 Single-Output Circuits

If we transform a circuit into an (ordered) BDD, we can evaluate the corresponding function for a given input vector in time $\mathcal{O}(\#I)$, if $\#I$ is the number of inputs of the circuit. Since in typical circuits the number of gates $\#G$ is much larger than the number of inputs, this method is (at least asymptotically) much faster than traditional approaches, like event driven simulation or levelized compiled code simulation.

3.1.2 Multi-Output Circuits

If a multi-rooted BDD, i.e. a BDD representing a function f with $\#O$ outputs, is evaluated, the straightforward method would require time $\mathcal{O}(\#I \cdot \#O)$. If we represent the functional behavior by using the characteristic function of the relation $R(f)$ of f, the evaluation time can be reduced to $\mathcal{O}(\#I + \#O)$ as follows [1]:

- Compute a BDD representation for the characteristic function of the relation $R(f)$ of f. The characteristic function can be represented by a BDD with $\#I$ 'input variables' and $\#O$ 'output variables'. The BDD is constructed with the restriction that all the input variables occur in the ordering before the output variables.
- If we want to evaluate the characteristic function, we have the problem that we do not know the output vector, rather we need to determine it. But since all input variables occur in the ordering before the output variables, we can make use of the fact that each input vector produces a unique output vector: The evaluation can simply be done by starting from the single root of the BDD for the characteristic function and evaluating this BDD according to the values of the input variables. A unique path to the terminal 1 determines the values of the outputs. It is easy to find this path, because each node which is labeled by an output variable has exactly one outgoing edge to the terminal 0 and one edge to another node.

The major drawback of this method is that the number of nodes at the cut line between the input and the output variables is equal to the number of different combinations that can occur at the outputs. Since this number often is exponential in the number of outputs the restriction can be infeasible for practical applications. For this reason Ashar and Malik [1] proposed two methods.

One is based on an interleaving of input and output variables. An output variable is located in the ordering directly after the last input variable on which this output depends. To determine an ordering of the input and output variables under this restriction Ashar and Malik [1] use a heuristic from [8]. Since there is no output variable before any input variable it depends on, nodes labeled with output variables still have the property that there is exactly one outgoing edge to the terminal 0 and one edge to another node. Therefore it is still possible to evaluate the multi-output function in time $\mathcal{O}(\#I + \#O)$.

The second method to reduce the number of nodes of BDD representations uses a partition of the circuit. Then their method is applied only to the subcircuits of this partition. To evaluate the overall circuit for some input vector BDDs for several subcircuits have to be evaluated. Obviously the partitioning method may increase the runtime, because the same input values are read more than once and/or intermediate variables are introduced.

McGeer et al. [5] also use a variable order with all input variables before the output variables they depend on. To optimize memory traffic they translate the decision diagram into an array, and a special-purpose program is automatically generated to traverse the array. In addition they use MDDs (*multi-valued* decision diagrams) instead of BDDs, where several BDD variables are combined into one MDD variable, such that the number of variables, which have to be evaluated, is reduced.

3.2 Optimization by Reordering

Reordering the variables of a BDD may have a large influence on the size of the representation.

We now study the effect of using dynamic variable ordering methods [4], like sifting [7], to reduce the size of the BDD representing the characteristic function. Here, we make no longer use of the restriction that the input and output variables should not be mixed.

If BDD sizes for these unrestricted reordering methods are smaller, we can choose larger subcircuits of the original circuit to be represented by BDDs for their characteristic functions, such that the number of evaluations of subcircuits is reduced.

However, in general, by dropping the ordering restriction above we can not further guarantee the time of $\mathcal{O}(\#I + \#O)$ to evaluate a multi-output function f which is represented by a BDD for $\chi_{R(f)}$.

3.3 Changed Evaluation Procedure

If we evaluate the BDD for the characteristic function, it is now possible to reach a node labeled by an output variable before all input variables are read, which this output depends on. Therefore we are not able to decide at this point which outgoing edge we have to follow. Both successors of this node can be different from the terminal 0. If we have reached such a node, we have to choose an arbitrary edge which we will follow. Thus, it is possible that we have to backtrack when it turns out that the decision at this output node was wrong. This is the case if we follow an edge starting from a node labeled by an input variable and reach terminal 0.

3.4 Evaluation Sifting

As explained in the previous section the number of steps in the evaluation can be increased if we drop the ordering restriction that all output variables should

appear after the input variables they depend on. On the other hand experimental results (see Section 4) show that the BDD sizes are reduced to a large extent if we apply sifting without this ordering restriction. For this, it would be desirable to combine the two advantages of small BDDs and a few evaluation steps.

To estimate the cost to evaluate an arbitrary input vector we determine for a BDD the average number of read accesses to BDD nodes in the evaluation procedure, which are needed to evaluate a random input vector. Read accesses to BDD nodes constitute critical operations when the BDDs are large and memory management effects (e.g. page faults and cache misses) are of importance. The expected value for this number of read accesses to BDD nodes (for a random input vector) is denoted by E_{r_a} in the following.

Now we have two optimization goals in the computation of a variable order for the BDD of $\chi_{R(f)}$: First we would like to minimize the number of BDD nodes to represent $\chi_{R(f)}$ and secondly we would like to minimize the expected value E_{r_a}.

To take account of these two goals we change the cost function of sifting: Until now the cost function is only the size of the resulting BDD. To determine the optimal position of a variable in the variable order it is sifted to all possible positions and then, the position, where the resulting BDD size is minimized, is selected. Now the cost function is changed to some combination of BDD size and the expected value E_{r_a}. For each position of the variable we determine the new size $size^{new}$ of the resulting BDD and the new expected value $E^{new}_{r_a}$. Then we choose the position for the variable where the expression

$$\alpha \cdot \frac{size^{new}}{size^{old}} + (1-\alpha) \cdot \frac{E^{new}_{r_a}}{E^{old}_{r_a}} \qquad (2)$$

is minimized. ($size^{old}$ and $E^{old}_{r_a}$, respectively, mean the BDD size and expected value before moving the variable, α is a number between 0 and 1 to influence the trade off between BDD sizes and read accesses in the evaluation.)

The resulting procedure is called *evaluation sifting* in the following.

3.4.1 Estimation of E_{r_a}

To estimate E_{r_a} we tried out two alternatives:

- If we assume that for a node v the probability to leave the subtree with root v^0 by a backtracking step of the evaluation procedure is independent from the probability to leave the subtree with root v^1, we can compute E_{r_a} bottom up in the BDD. However, since this assumption is only true for some special cases (e.g. when all output variables occur after the input variables), this will give only an estimation of E_{r_a}.
- To reduce this runtime for larger BDD sizes we also use another method for the estimation: We simulate $c \cdot n$ input vectors (for some small constant c) and count the average number of read accesses to the BDD in this way. (n is the number of variables in the BDD.)

4 Experimental Results

We performed experiments to compare sifting and evaluation sifting to the ordering strategy from [8, 1], where the input variables are located before the output variables they depend on.

Apart from BDD sizes for the different strategies, we determined for 500,000 random input patterns the average number of read accesses to BDD nodes in the evaluation procedure, which were needed to evaluate the input vectors, and finally we determined runtimes for the evaluation of random input vectors.

In columns TSLBS-ord of Table 1 we give results for the variable ordering strategy from [8, 1], in columns orig_sift results for the original sifting algorithm and in columns eval_sift for evaluation sifting (we chose $\alpha = 0.5$ in formula (2) and used simulation to estimate E_{r_a}).

The sizes of the resulting BDDs are given counted in numbers of nodes in Table 1 (columns 'node'), the average numbers of read accesses are given in columns 'r.a.' and the simulation times in CPU seconds for 500,000 random input patterns measured on a SUN *Sparc 20* workstation (256 MB physical memory) in columns 'time'.

As easily can be seen there are some examples where the partitioning techniques proposed in [1] and [5] are needed for their method, since the BDDs for the characteristic function of the relation can not be constructed[1] for the variable ordering method from [8, 1].

In contrast, sifting can build the BDDs in all considered cases. Beside the cases where the other methods fail for some examples the results of unrestricted sifting are up to a factor of 9 better than the results of the ordering heuristic from [8, 1].

For this reason we can expect that we will need substantially less subcircuits in the partition for larger circuits when we apply sifting (and this will lead to reduced evaluation times as explained in Section 3.1).

We now study the effect of the size reduction with respect to evaluation time for the circuits which could be represented in one partition. On a first view we can observe the "surprising" result that the average number of read accesses to BDD nodes increases for the original sifting algorithm in the worst case only by a factor of 1.5 compared to the variable ordering from [8, 1]. On the other hand already for original sifting there are examples where the number of read accesses is reduced.

If we compare original sifting and evaluation sifting, we can see that it is possible to achieve a considerable reduction of read accesses and runtimes with only a small overhead in terms of node counts. There are circuits (e.g. c1908, rd73, rd84), for which the numbers of read accesses for evaluation sifting are reduced by about one half. Surprisingly, even the numbers for the ordering from [8, 1] (where no backtracking steps

[1] We aborted the construction of the BDD for the relation, when more than 2,000,000 nodes were needed.

circuit	TSLBS-ord			orig_sift			eval_sift		
	nodes	r.a.	time	nodes	r.a.	time	nodes	r.a.	time
5xp1	93	22.50	2.94	71	25.49	3.81	74	23.77	3.54
alu2	289	18.68	2.67	279	19.09	2.52	295	13.49	1.99
apex7	4,665	86.65	12.70	2,323	107.67	15.25	2,479	84.44	11.96
b9	1,582	53.29	8.07	663	74.99	10.28	735	53.94	7.22
c432	9,530	28.64	7.99	1,584	30.06	4.49	1,730	26.26	4.03
c499	>2,000,000	-	-	14,235	259.44	71.00	17,017	225.06	51.55
c880	467,346	86.11	20.08	367,863	85.00	34.75	393,633	70.41	29.37
c1355	>2,000,000	-	-	15,957	274.50	67.19	16,926	242.76	53.20
c1908	>2,000,000	-	-	109,042	323.81	137.55	117,231	159.90	71.73
clip	171	16.50	2.25	103	17.84	2.52	102	16.98	2.61
count	145	30.75	4.82	142	29.75	4.57	184	15.73	3.03
e64	258	131.00	12.27	258	131.00	12.20	258	131.00	12.21
f51m	75	20.00	2.85	63	17.71	2.76	65	17.55	2.73
misex1	66	15.89	1.98	58	16.77	2.16	59	15.64	1.95
misex2	333	39.98	4.07	221	46.21	4.95	230	40.43	4.30
rd73	42	11.50	1.61	36	12.87	1.85	42	4.98	1.16
rd84	55	14.40	1.95	49	15.64	2.16	52	8.99	1.36
sao2	106	13.59	1.70	85	10.18	1.23	86	10.18	1.26
term1	5,541	30.01	3.99	578	31.18	4.02	612	30.13	3.94
vg2	171	22.98	3.05	140	21.63	2.60	140	21.54	2.58
z4ml	30	13.00	1.82	29	13.25	1.86	30	12.01	1.98

Table 1: Node counts, average number of read accesses and evaluation times for 500,000 random patterns

in the evaluation procedure can occur) are improved in most cases (while keeping the advantage of much smaller BDD sizes).

These unexpected results can be explained by the following observation:
In an optimized BDD it often occurs that less variables are tested until a terminal node is reached. This effect leads to the fact that the additional read accesses due to 'wrong decisions' (see Section 3.3) of the evaluation procedure are more than compensated.

5 Conclusions and Future Work

We discussed the use of dynamic variable ordering for functional simulation. It turned out that the use of dynamic reordering has a large potential in this area.

We developed a modified version of the sifting algorithm (called evaluation sifting), which is able to combine advantages of sifting and variable orders with the restriction that all output variables are located after the input variables they depend on [8, 1]. The small BDD sizes of sifting are almost maintained, whereas evaluation times in comparison to the variable order from [8, 1] are even improved.

Since the numbers of BDD nodes in our approach are much smaller we can expect that we will need substantially less subcircuits in the partition for larger circuits (and this will lead to reduced evaluation times as explained in Section 3.1).

At the moment we are integrating the concept of [5] to reduce the number of evaluations by combining several BDD variables into one MDD variable. Note that we thereby adjust evaluation sifting by changing the computation of E_{r_a} from Section 3.4 to estimate the number of operations in the evaluation procedure from [5] immediately on the basis of the BDD representing the characteristic function $\chi_{R(f)}$.

References

[1] P. Ashar and S. Malik. Fast functional simulation using branching programs. In *Int'l Conf. on CAD*, pages 408–412, 1995.

[2] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[3] E. Cerny and M.A. Marin. An approach to unified methodology of combinational switching circuits. *IEEE Trans. on Comp.*, 26:745–756, 1977.

[4] M. Fujita, Y. Matsunaga, and T. Kakuda. On variable ordering of binary decision diagrams for the application of multi-level synthesis. In *European Conf. on Design Automation*, pages 50–54, 1991.

[5] P.C. McGeer, K.L. McMillan, A. Saldanha, A.L. Sangiovanni-Vincentelli, and P. Scaglia. Fast discrete function evaluation using decision diagrams. In *Int'l Conf. on CAD*, pages 402–407, 1995.

[6] D.A. Patterson and J.L. Hennessy. *Computer Organization and Design. The Hardware/Software Interface.* Morgan Kaufman Publishers - CA, 1994.

[7] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.

[8] H. Touati, H. Savoj, B. Lin, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Implicit enumeration of finite state machines using BDDs. In *Int'l Conf. on CAD*, pages 130–133, 1990.