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.

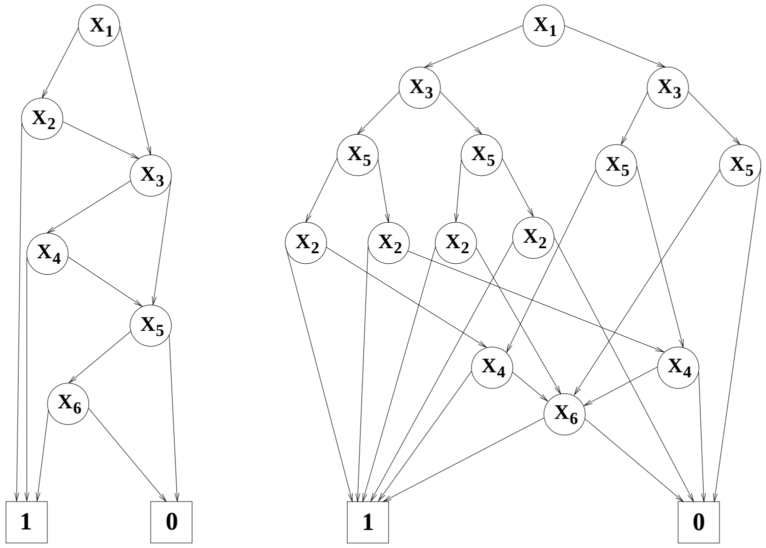

Figure 1: BDDs for function $f_3 = x_1x_2 + x_3x_4 + x_5x_6$

The paper is structured as follows: In Section 2 we review basic notations and definitions. Our approach to functional simulation is presented in Section 3. In Section 4 experimental results are given. Finally, we give conclusions and discuss future work.

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 $f : \mathbf{B}^n \rightarrow \mathbf{B}$ 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.

An example from [2] shows the importance of the variable ordering for BDDs:

Example 1 *Let $f_n = x_1x_2 + \ldots + x_{2n-1}x_{2n}$. If the variable ordering is given by $(x_1, x_2, \ldots, x_{2n})$ the size of the resulting BDD is $2n + 2$. On the other hand if the variable ordering is chosen* *as* $(x_1, x_3, \ldots, x_{2n-1}, x_2, x_4, \ldots, x_{2n})$ *the size of the BDD is* 2^{n+1}*. Thus the number of nodes in the graph varies from linear to exponential depending on the variable ordering. In Fig. 1 the BDDs of the function* $f_3 = x_1x_2 + x_3x_4 + x_5x_6$ *with variable orderings* $(x_1, x_2, x_3, x_4, x_5, x_6)$ *and* $(x_1, x_3, x_5, x_2, x_4, x_6)$ *are illustrated. The left (right) outgoing edge of each node* x_i *denotes the cofactor* $f_{x_i=1}$ *(*$f_{x_i=0}$*). The example proves that the choice of the variable ordering largely influences the size of the BDDs.*

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*

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

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]:

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

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 (see e.g. Example 1).

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. Figure 2 shows the changed evaluation procedure.

4 Experimental Results

We performed experiments to demonstrate the effect of unrestricted sifting compared to other variable ordering strategies: The first was the ordering strategy from [8, 1] and the second was block sifting, where the input variables are located before the output variables and input variables and output variables are sifted separately.

Apart from BDD sizes for the different strategies, we also determined the average number of read accesses to BDD nodes in the evaluation procedure of Figure 2, which were needed to evaluate random input vectors. Read accesses to BDD nodes constitute critical operations when the BDDs are large and memory management effects are of importance. Finally we compared runtimes for the evaluation of random input vectors.

Our goal is to answer the following questions:

- How do the numbers of nodes for the three variable ordering strategies compare? Is it worthwhile accepting the risk of increasing evaluation time when we use unrestricted sifting?
- To what extent do read accesses to BDD nodes increase when unrestricted sifting is applied? Is the number of read accesses for unrestricted sifting much larger than for the other approaches where output variables are located after all input variables they depend on?

The sizes of the resulting BDDs are given counted in number of nodes in Table 1. *in* (*out*) denotes the number of inputs (outputs). The number of nodes of the BDDs for the characteristic functions of the relations are given in the last three columns. In column block_sift we give the

Given: Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}^m$,
BDD with node set V and root v_{root} for characteristic function $\chi_{R(f)}$
input variables $i_1, \ldots, i_n$, output variables $o_1, \ldots, o_m$,
label function $l : V \rightarrow \{i_1, \ldots, i_n, o_1, \ldots, o_m\} \cup \{0,1\}$
input vector $(\epsilon_1, \ldots, \epsilon_n)$

Find: Output vector $(\delta_1, \ldots, \delta_m) = f(\epsilon_1, \ldots, \epsilon_n)$

Algorithm:

```
/* Let v^0 be the 0-son of a node v, v^1 the 1-son */
var stack outstack /* Stack with maximum depth m for nodes labeled with output variables */
    v := v_root;
    while (l(v) ≠ 1) do
          if (l(v) input variable)
            then
                 /* Let l(v) be i_j */
                 v := v^{ε_j};
                 if (l(v) = 0)
                   then
                         v := pop(outstack);
                         /* Let l(v) be o_k */
                         δ_k := 0; v := v^0;
                 fi
            else
                 /* l(v) output variable, let l(v) be o_k */
                 if (l(v^1) = 0)
                   then
                         δ_k := 0; v := v^0;
                   else
                         if (l(v^0) ≠ 0) then push(outstack, v); fi
                         δ_k := 1; v := v^1;  /* guess f_k(ε_1, ..., ε_n) = 1 */
                 fi
          fi
    od
```

Figure 2: Evaluation procedure.

number of nodes for block sifting (i.e. restricted sifting), in column TSLBS-ord the number of nodes for the variable ordering strategy from [8, 1] and in column unrestr_sift the number of nodes for unrestricted sifting.

As can easily 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] both for block sifting (input and output variables not mixed) and for the variable ordering method of [8, 1]. In contrast, unrestricted sifting can build the BDDs in all considered cases. Beside the cases where the other methods fail the results of unrestricted sifting are up to a factor of 9 better than the results of the ordering heuristic from [8, 1]. (The size reduction is even more dramatic in comparison to block sifting.)

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

circuit	in	out	block_sift	TSLBS-ord	unrestr_sift
5xp1	7	10	623	93	71
alu2	10	8	414	289	279
apex7	49	37	> 2,000,000	4,665	2,323
b9	41	21	43,102	1,582	663
c432	36	7	2,154	9,530	1,584
c499	41	32	> 2,000,000	> 2,000,000	14,235
c880	60	26	> 2,000,000	467,346	367,863
c1355	41	32	> 2,000,000	> 2,000,000	15,957
c1908	33	25	> 2,000,000	> 2,000,000	109,042
clip	9	5	171	171	103
count	35	16	262,159	145	142
e64	65	65	2,339	258	258
f51m	8	8	765	75	63
misex1	8	7	58	66	58
misex2	25	8	339	333	221
rd73	7	3	42	42	36
rd84	8	4	55	55	49
sao2	10	4	106	106	85
term1	34	10	4,040	5,541	578
vg2	25	8	528	171	140
z4ml	7	4	78	30	29

Table 1: Node counts for characteristic function

For this reason we can expect that we will need substantially less subcircuits in the partition for larger circuits when we apply unrestricted 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. The average number of read accesses (column r.a.) to evaluate an input vector and the simulation times in CPU seconds (column time) for 500,000 random input patterns measured on a SUN *Sparc 20* workstation (256 MB physical memory) are given in Table 2. In the last column the number of input/output violations is given, i.e. the number of times how often an input variable occurred *after* the variable of an output, which the input variable depends on. (Input/output violations can potentially lead to backtracking steps in the evaluation procedure due to wrong decisions at nodes corresponding to output variables.) On a first view we can observe the 'surprising' result that the average number of read accesses to BDD nodes increases in the worst case only by a factor of 1.5 for unrestricted sifting, but on the other hand in many cases there are even less read accesses for unrestricted sifting. This is even more surprising if we consider the large number of violations that result from unrestricted sifting.

	block_sift		TSLBS-ord		unrestr_sift		
circuit	r.a.	time	r.a.	time	r.a.	time	viol.
5xp1	22.50	3.07	22.50	2.94	25.49	3.81	7
alu2	18.78	2.67	18.68	2.67	19.09	2.52	1
apex7	-	-	86.65	12.70	107.67	15.25	69
b9	50.89	11.96	53.29	8.07	74.99	10.28	39
c432	26.53	4.81	28.64	7.99	30.06	4.49	41
c499	-	-	-	-	259.44	71.00	615
c880	-	-	86.11	20.08	85.00	34.75	27
c1355	-	-	-	-	274.50	67.19	617
c1908	-	-	-	-	323.81	137.55	371
clip	16.50	2.27	16.50	2.25	17.84	2.52	10
count	29.75	10.08	30.75	4.82	29.75	4.57	0
e64	132.00	12.69	131.00	12.27	131.00	12.20	0
f51m	20.00	2.91	20.00	2.85	17.71	2.76	4
misex1	15.64	1.94	15.89	1.98	16.77	2.16	3
misex2	39.80	4.10	39.98	4.07	46.21	4.95	14
rd73	11.50	1.62	11.50	1.61	12.87	1.85	4
rd84	14.40	1.93	14.40	1.95	15.64	2.16	4
sao2	13.59	1.70	13.59	1.70	10.18	1.23	0
term1	28.89	3.81	30.01	3.99	31.18	4.02	12
vg2	25.69	3.15	22.98	3.05	21.63	2.60	0
z4ml	13.00	1.79	13.00	1.82	13.25	1.86	1

Table 2: Average number of read accesses and evaluation times for 500,000 random patterns

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 (see e.g. function from Example 1.) This effect compensates the additional read accesses due to 'wrong decisions' (see Section 3.3).

In addition there are examples (e.g. c432) where the runtime in the case of unrestricted sifting is smaller even though the number of read accesses is larger. This can be explained by memory management effects: If the code to be executed is larger in size the probability of page faults and cache misses during the execution of the program increases.

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.

Finally, we discuss some possible further extensions of our presented approach. If we want to further keep the good runtimes implied by restricted sifting or the ordering method of [8, 1] and allow some more space for the BDD representation compared to unrestricted sifting we can also apply the following basic methods:

All inputs before the corresponding outputs [1]: In many cases not all outputs depend on all inputs. Thus, the sifting algorithm can be modified in such a way that it is allowed to move an output variable before an input variable, if this output does not depend on this input.

Only a constant number of output variables is allowed to occur before their input variables (and of course only if the resulting size reduction of the BDD is large enough): If only a constant number of output variables are allowed to occur before the input variables the asymptotic worst case behavior does not change, i.e. the evaluation time is also given by $\mathcal{O}(\#I + \#O)$. Obviously, for 'real world' applications this number must not be too large.

Apply sifting with a different cost function. Until now the cost function of sifting is the size of the resulting BDD. We can change this cost function to take the expected number of read accesses to BDD nodes (in the evaluation of a random input vector) into consideration. The cost function is now a combination of BDD size and this expected value. Changing the weights for these two components of the cost function gives an immediate influence to the trade-off between BDD sizes and read accesses in the evaluation.
First experiments using this modified version of sifting are promising: We have to allow only a few more BDD nodes compared to the original sifting algorithm and at the same time we are able to *even reduce* the number of read accesses to BDD nodes in the evaluation procedure compared to the approaches where all output variables are located after the input variables they depend on (this is possible because on a path in the BDD not necessarily all input variables are tested, see Example 1). Whereas experimental results of Section 4 show that the increase of the number of read accesses is surprisingly small for unrestricted sifting, it turns out that we can even reduce this number while keeping small BDD sizes.

It is focus of current work to integrate all methods proposed above in the sifting algorithm and to evaluate their quality with respect to size and evaluation time.

Finally, it should be mentioned that the partitioning technique proposed in [1, 5] can also be used in combination with the ideas presented above. 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).

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.