Heuristics to Compute Variable Orderings for Efficient Manipulation of Ordered Binary Decision Diagrams *

Kenneth M. Butler[†] Don E. Ross[‡] Rohit Kapur[§] M. Ray Mercer[§]

[†]Texas Instruments
P.O. Box 655303 MS 3683
Dallas, TX 75265

[‡]Dept. of Electrical Eng.
Texas A&M University
College Station, TX 77843

[§]The University of Texas
Dept. of ECE ENS 143
Austin, TX 78712

Abstract

Variable ordering is critical in the efficient representation of functions as Ordered Binary Decision Diagrams (OBDDs). In this paper we present new heuristics to determine "good" variable orderings. We use a new representation form, Ordered Partial Decision Diagrams (OPDDs), to evaluate heuristically generated orders on large circuits. Several improved orders are located for benchmark circuits, and an overall ordering strategy is introduced which includes partial functional calculations and which requires insignificant computation resources.

1 Introduction

Ordered Binary Decision Diagrams (OBDDs) provide a way of compactly representing symbolic information. To facilitate these functional manipulations, an ordering must be imposed on the variables in which the functions are expressed. Because OBDD computational complexity is closely related to its space requirements [BRYA86], it is important to find an ordering that attempts to minimize the memory used during all phases of OBDD manipulation. However, the complexity of finding the optimum ordering has demonstrated the need for heuristic ordering methods [FRIE90].

Although the number of possible variable orderings is large, if a significant fraction of them result in acceptable levels of resource consumption during symbolic calculations, then simple heuristics (asymptotically approaching random ordering) may be all that are necessary. In order to investigate this statement more closely, we calculated "resource consumption distributions" for several small circuits. We computed the exhaustive set of variable orderings for the smaller ISCAS circuits and an example function given in [BRYA86]. The total node count of the post-*Reduce* functions was measured and a distribution was made of the number of orderings versus the total node count. The results indicated that no distribution pattern exists for even two of these circuits, much less a general distribution pattern. We interpret this to mean that it is unlikely that a general distribution pattern exists for larger circuits, which can be even more diverse. However, in all cases there existed multiple orderings that are optimal with respect to the number of nodes required to represent all the functions. We expect this property to hold for larger circuits as well, so the empirical problem becomes that of finding one of a potentially "large" number of acceptable orderings scattered sparsely in a vast sea of existing orderings.

To enable us to compare ordering heuristics on large circuits, we resort to a form of OBDDs in which we can transfer the effects of an inefficient representation from intractability in space to potential intractability in time. Where complete information would be prohibitive, we can limit memory requirements by limiting the results to partial information about functions. The partial OBDDs are called Ordered Partial Decision Diagrams (OPDDs). OPDDs are also sensitive to ordering, and the efficiency of the OBDD orderings produced can be determined by observing the percentage of functional information achieved by computing OPDDs [ROSS91].

2 Variable Ordering Heuristics

Previous heuristics for variable ordering are quite similar [FUJI88], [MALI88], [BERM88], [BERM89], [MINA90]. They all use topological analyses of the circuit to arrive at a variable ordering. Most of the previous algorithms perform some sort of variation of the depth or breadth first search in combination with a circuit partitioning scheme. In this section we develop similar heuristics but provide the comparative information missing in previous work.

2.1 Depth and Breadth First Topological Heuristics

For depth first and breadth first traversals to be well defined, the order in which the gates are visited must

*This work was supported by the National Science Foundation under Grant MIP-8552537, by the Semiconductor Research Corporation under Contract #90-DP-142, and by the Innovative Science and Technology Office of Strategic Defense Initiative Organization, administered through the Office of Naval Research under contract N-00014-86-K-00554.

be well defined. Furthermore, multiple iterations may be required to order all circuit inputs. Selecting the circuit output(s) from which to traverse selects the set of functions which will influence the ordering the most. Clearly, the larger a function, the larger its OBDD can become, and the more imperative it is to achieve a good ordering for that function. This leads to a desire to order outputs from the one with the most inputs reaching it to the one with the least, and select outputs from which to traverse in descending order from this list, as the iterations proceed. Surprisingly, implementing this as part of the heuristic has more effect upon the results than any perturbation of the traversal.

Of the two basic traversals, neither is dependably superior, although we observed that depth first based heuristics required less total OBDD nodes for more circuits than breadth first. However, breadth first heuristics generated the best orderings for two of the ISCAS circuits, so a breadth first traversal is advisable as an alternative ordering heuristic. For breadth first traversals, more than one output can be traversed, but better orderings resulted from selecting only the single largest output to traverse. Also, we took care to preserve the depth first order as much as possible within the breadth first expansion, by keeping inputs to the same gate together in the traversal queues. For depth first heuristics, selecting the gate input with the largest number of circuit inputs reaching it as the first to traverse backwards from created superior orderings to selecting the gate input with a smaller number first. For ties, using the input with the most (or the least) fanout created only minor differences in ordering results (25% or less). Neither consistently won. We did not attempt to partition the inputs to a gate into disjoint sets, as was suggested in previous work [MALI88]. However, comparison with the orderings produced by those partitioning heuristics [MALI90] shows that the addition of this feature produces up to a 50% decrease in the nodes required for one of the ISCAS circuits. So, this simple feature appears to be effective even for highly reconvergent circuits, even though its theoretical motivation comes from tree interconnections of gates, which almost never occur in practice.

In the results section, specific results for various traversal options are presented. A brief summary of the variations presented there follows:

B: Breadth first from single largest PO. Order all variables reaching that PO. Pick next smaller PO, etc., until all PIs ordered.

D: Depth first from single largest PO. At a gate, follow input with most circuit inputs reaching it first. If tie, follow input with least fanouts first.

2.2 Simulation Based Heuristics

In this section we describe a heuristic for ordering variables based on the results of fault simulation. Decisively controlling variables are sought by simulating their incremental effects on decreasing residues of the static good machine state.

The algorithm implementing all of the fault simulation based variable ordering heuristics is given in Figure 1. After an initialization of both the "good" and "faulted" machines to the all X state, each unordered PI (initially all PIs) is individually set to both binary values while holding the remaining PIs at X. Simulation occurs for each of these copies of the "faulted" machine. Thus, we are able to observe each unordered PI's effect on the current good machine state when it is set to a 0 by itself and 1 by itself.

```
function order-variables-by-simulating()
{
  initialize-good-machines-to-all-X(),
  initialize-faulted-machines-to-all-X();
  while(some PIs are still unordered)
  {
    for(all PIs ordered on previous iteration)
    {
      set-PI-to-opposite-of-implication-value(),
    } /* end for all PIs */
    simulate-good-machines(),
    for(i=0, i≤1, i++)
    {
      init-faulted-machines-to-current-good-machines-state();
      for(ceiling(num-unordered-PIs/simulator-bits) repetitions)
      {
        singly-set-unordered-PIs-to-value(i),
        simulate-faulted-machines();
        count-values-differing-from-good-machines(),
      }
    } /* end for i */
    order-next-PIs-with-most-implications();
  } /* end while some PIs */
  return;
} /* end order-variables-by-simulating() */
```

Figure 1: A pseudo-code representation of the algorithm implementing the heuristic for variable ordering via fault simulation (SIM).

Each PI has a "0 implication count" and a "1 implication count". For PI i, these counts simply express the number of non-X implications that occurred on non-single-input gates as a result of fixing PI i to each of 0 and 1, respectively, which did not exist in the previous state of the machine (the current good machine state). Once these counts have been compiled, those PIs with the highest "implication counts" are appended to the current variable ordering. Also recorded for each of these newly ordered PIs is the value which caused these large numbers of implications. If a tie occurs, this value is arbitrarily set to 0, but could just as easily be chosen randomly or in some other fashion.

On subsequent passes through the outermost loop, each newly ordered PI is fixed to the *opposite* of its "many implications" value (in an attempt to point the search in the direction of the larger remaining residue function). An incremental good machine simulation is performed, thus updating the current good machine state. This process continues until all variables have been ordered. Many different variations of the implication counting methods and variable selection criteria are possible, each with varying behaviors. We will limit our results to the basic heuristic (SIM).

The fact that SIM did reasonably well suggests that perhaps implication counting might best be used as a tie-breaking mechanism for other heuristics. This is particularly true in circuits containing many XOR and/or XNOR functions, because they tend to cause

many X values in the simulator until the inputs driving those gates have all been ordered.

2.3 Testability Measure Based Heuristics

The heuristics developed in this section are motivated by applications that require functional information at the internal nodes of the circuit, such as test generation. Because all OBDDs need to be minimized and not merely output OBDDs, each line of the circuit at which functional information is to be calculated should contribute some information about the variable ordering that would minimize its functional representation. Then, depending on the relative size of the function being computed, a line's vote for an ordering is given a weight. A weighted average of all the votes leads us to an ordering that would attempt to minimize the functional representations at all the lines in the circuit.

We use testability measures (SCOAP [GOLD79]) to give us an estimate on the ordering of variables for each gate. Consider a gate G in the middle of a circuit with fanin lines A and B. Assume A to be fed directly from a primary input of the circuit and B to be fed by the output of a complex block of circuitry. The observability numbers computed by SCOAP for line A will be high and those of line B will be low. From our knowledge of the ordering of decisively controlling variables we require the variables affecting line A to be put higher in the ordering than the variables affecting line B (to reduce the size of the function at the output of the gate G). The heuristic presented focuses on this correlation of observability with ordering to generate weights for all the inputs of the circuit. The weights computed at the inputs of the circuit when considering one gate and its observabilities at its fanin lines are the votes toward the ordering for the OBDD at the output of the gate. This vote is given a weight by considering the number of input variables that fall in the cone of influence of the gate G. A summary of the algorithm is given in Figure 2.

The algorithm presented has a number of parameters which can effect the variable ordering produced. In Step 3, the gates that give a vote towards an ordering can be varied from a single gate to all the gates in the circuit. In the heuristic (SC) we selected the last few gates (four times the number of outputs) in the circuit out of a levelized list of gates. Another parameter is the bias given to the weights to compute a weighted average. For SC we used a bias of 1, as all the functions close to the output have nearly the same size. The conversion of observability to weight at the input of a gate was found to be a critical parameter in the algorithm. If the difference in the weights assigned is not significant, the noise in the backward propagation of the circuit obscures the relevant information needed to generate a good ordering. We used a bias proportional to the square of the observabilities to generate the initial weights at the input of the gate that is considered to give its vote towards an ordering.

3 Results and Conclusions

1. Compute statistical information on the number of primary inputs reaching each gate in the circuit. Let this number be n_g for gate g
2. Compute SCOAP observability numbers for every line in the circuit. Let O_A denote the observability of line A
3. For every gate $g \in Circuit$ (at which the gate outputs OBDD gives its vote towards an ordering)
   (a) Compute a weight at the fanin lines of g such that a high observability gets a large weight and a low observability a small weight
   (b) Propagate these weights backwards to the primary inputs using the following equations
   $$Wt\ at\ input\ of\ gate = \frac{Weight\ at\ the\ output\ of\ gate}{fanin\ of\ gate}$$
   $$Wt\ of\ fanout\ stem = Mean\ weight\ of\ fanout\ branches$$
   (c) Record the set of weights of the primary inputs and associate n_g with that set.
4. Take a weighted average of the sets of weights computed for the primary inputs using a bias that depends on n_g
5. Order the primary inputs (variables) in descending order of their weights

Figure 2: Steps of the heuristic using Testability Measures (SC).

CKT	B	D	SIM	SC	cmu	ucb
C432	**42k**	65%	64%	80k	65%	65%
C499	78%	238k	97%	93%	235k	**217k**
C880	96%	35k	97%	91%	**29k**	33k
C1355	*	483k	*	*	477k	**438k**
C1908	**143k**	180k	228k	342k	147k	147k
C2670	90%	98%	98%	96%	**99+%**	98%
C3540	88%	*	*	**89%**	*	81%
C5315	94%	148k	99+%	99%	99+%	**77k**
C7552	*	90%	91%	91%	**99%**	**99%**

Table 1: Total OBDD nodes or Percent fully specified gate outputs.

CKT	B	D	SIM	SC	cmu	ucb
C432	**52k**	99+%	98%	85k	99+%	99+%
C499	96%	261k	99%	98%	259k	**243k**
C880	99%	35k	99+%	96%	**30k**	33k
C1355	*	550k	*	*	545k	**506k**
C1908	**147k**	183k	232k	359k	149k	149k
C2670	97%	99%	**99+%**	99%	**99+%**	99%
C3540	**96%**	*	*	**96%**	*	89%
C5315	99+%	154k	99+%	99+%	99+%	**83k**
C7552	*	98%	99%	99%	**99+%**	**99+%**

Table 2: Total *Apply* nodes or Mean percentage of minterms known of gate outputs

CKT	B	D	SIM	SC	cmu	ucb
C432	**2108**	97%	86%	3395	97%	97%
C499	76%	5013	93%	85%	4949	**4533**
C880	94%	3257	99%	93%	2790	**2401**
C1355	*	5013	*	*	4949	**4533**
C1908	**2279**	3012	4395	6481	3044	3044
C2670	96%	**98%**	**98%**	97%	**98%**	**98%**
C3540	79%	*	*	**80%**	*	61%
C5315	95%	5662	99%	98%	99%	**1740**
C7552	*	86%	92%	90%	98%	**99%**

Table 3: Largest OBDD or Mean percentage of minterms known of POs

We generated orderings for the ISCAS combinational circuits [BRGL85] using the ordering heuristics presented in the previous section, the results of which are given in Tables 1, 2 and 3. The C6288 was omitted due to ordering insensitivity (all orderings perform equally). The orderings generated from improvements [MALI90] to the heuristics reported in [MALI88] are labeled "*ucb*" and those generated from heuristics in [BRAC90] are labeled "*cmu*". We ran OPDDs with a memory limit per apply graph of 8000 nodes and a limit on the total number of nodes of 800,000. If during a run the total nodes in all OPDDs exceeded the 800,000 limit, the run was aborted, denoted by an asterisk in the table. Depending on the ordering, the data in the tables correspond to information computed from fully specified functions or partial functions. The best results for each circuit are in bold font, but one must take care to notice that in all cases the second best ordering requires less than two times the number of nodes. Therefore, multiple heuristic strategies were effective for all circuits, relative to the orders of magnitude difference using random orderings. In comparison, random orderings are typically so poor that they abort very early in the circuit, or lead to many orders of magnitude increase in the nodes required to achieve coverages similar to our best known orderings in the rare cases where they do not abort.

These results also indicate that no single heuristic will always provide a variable ordering that causes resource consumption to be within an order of magnitude or less of the best known value(s) for any circuit. This demonstrates the necessity of the creation of a "stable" of heuristics that can be used for circuit variable ordering. Because we cannot presently know a priori which particular heuristic will provide the best ordering for a circuit, a hybrid approach is warranted.

We propose that the rapid functional manipulations enabled by partial functional representation are ideally suited to this problem. In the hybrid approach, several heuristics, based upon different sets of "minimization criteria" are applied, partial results are calculated, and the single best performing ordering is selected as the one to be used in the actual application. Our research has shown that there is a strong correlation between the success (the amount of information extracted) of a variable ordering in partial functional calculations and its "goodness" as an ordering for complete functional calculations. Furthermore, the time and memory resources needed to make this determination can typically be made very small when compared with the time and memory savings realized in full functional calculations, or with the time and memory savings for achieving an equivalent amount of known information from partial calculations in an application.

4 Acknowledgements

The authors would like to thank Sharad Malik of Princeton and Karl Brace of CMU for correspondence and providing us with information on their orderings. Also acknowledged is Dr. Joseph Rahmeh of UT Austin for numerous consultations.

References

[BERM88] C.L. Berman, "Circuit width, register allocation, and reduced function graphs," IBM Res. Rep. RC 14127, Nov. 1988.

[BERM89] C.L. Berman, "Ordered Binary Decision Diagrams and Circuit Structure," Extended Abstract, *Proc. Int. Conf. Comput. Design*, Oct. 1989, pp. 392-395.

[BRAC90] K.S. Brace, R.E. Bryant, R.L. Rudell, "Efficient implementation of a BDD package," *Proc. ACM/IEEE 27th Design Automation Conf.*, June 1990, pp. 40-45.

[BRGL85] F. Brglez, H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in FORTRAN," *Proc. IEEE Int. Symp. on Circ. Syst. (ISCAS)*, June 1985, pp. 695-698.

[BRYA86] R.E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Comput.*, vol C-35, no. 8, Aug. 1986, pp. 677-692.

[FRIE90] S.J. Friedman, K.J. Supowit, "Finding the optimal variable ordering for binary decision diagrams," *IEEE Trans. on Comput.*, vol C-39, no. 5, May 1990, pp. 710-713. see also *Proc. ACM-IEEE 24th Design Automation Conf.*, June 1987, pp. 348-356.

[FUJI88] M. Fujita, H. Fujisawa, N. Kawato, "Evaluation and improvements of Boolean comparison method based on binary decision diagrams," *Dig. Technical Papers, Int. Conf. CAD*, Nov. 1988, pp. 2-5.

[GOLD79] L.H. Goldstein, "Controllability/ Observability analysis of digital circuits," *IEEE Trans. Circ. Syst.*, vol. CAS-26, no. 9, Sept. 1979, pp. 685-693.

[MALI88] S. Malik, A.R. Wang, R.K. Brayton, A. Sangiovanni-Vincentelli, "Logic verification using binary decision diagrams in a logic synthesis environment," *Dig. Technical Papers, Int. Conf. CAD*, Nov. 1988, pp. 6-9.

[MALI90] S. Malik, *personal communication*, May 1990.

[MINA90] S. Minato, N. Ishiura, S. Yajima, "Shared Binary Decision Diagram with attributed edges for efficient Boolean function manipulation," *Proc. 27th ACM/IEEE Design Automation Conf.*, June 1990, pp. 52-57.

[ROSS90] D.E. Ross, "Functional calculations using ordered partial multi decision diagrams," Ph.D. Dissertation, The University of Texas at Austin, Aug. 1990.

[ROSS91] D.E. Ross, K.M. Butler, R. Kapur, M.R. Mercer, "Fast functional evaluation of candidate OBDD variable orderings," *Proc. European Design Automation Conf.*, Feb. 1991, pp. 4-10.