Combinational Circuit ATPG Using Binary Decision Diagrams

Sanjay Srinivasan Gnanasekaran Swaminathan James H. Aylor
Dept. of Electrical Engineering, University of Virginia, Charlottesville

M. Ray Mercer, University of Texas at Austin

Abstract

The increasing size and complexity of current VLSI circuits has brought testing and design for testability into the mainstream of the design process. A significant amount of research has been done in the area of gate-level combinational ATPG using the stuck-at-fault model. The problem has been shown to be NP-complete and most of the current research attempts to find efficient ways to generate tests for hard faults in a reasonable amount of time on the average. Hence, there remains a lot of interest in the testing world for efficient techniques to do combinational ATPG. Use of Ordered Binary Decision Diagrams (OBDDs) for function representation has provided significant impetus to algebraic CAD techniques. This paper presents techniques for gate-level ATPG using OBDDs [1][2].

1. Introduction

The increasing size and complexity of current VLSI circuits has brought testing and design for testability into the mainstream of the design process. The high cost of Automatic Test Pattern Generation (ATPG) for sequential circuits has forced the incorporation of design for testability techniques like scan design. This technique converts the problem of ATPG for sequential circuits into one of ATPG for purely combinational circuits. A significant amount of research has been done in the area of gate-level combinational ATPG using the stuck-at-fault model. The problem has been shown to be NP-complete and most of the current research attempts to find efficient ways to generate tests for hard faults in a reasonable amount of time. However, VLSI designers have been reluctant to incorporate ATPG techniques into their environments simply because current ATPG systems either take inordinate amounts of time in generating tests for faults or do not achieve significant fault coverage in the allotted time. Hence, there remains a lot of interest in the testing community for efficient techniques to do combinational ATPG. This paper presents techniques for gate-level ATPG using Ordered Binary Decision Diagrams (OBDDs) [1][2]. With current OBDD technology, a stand-alone OBDD based ATPG system is not practical (in terms of memory usage) for large circuits. Hence a combination of path sensitization based and OBDD based ATPG systems is preferred. Such a system uses the path sensitization based TPG to generate tests for the easy faults and the OBDD based generator for the hard and redundant faults.

The authors would like to acknowledge the support provided by the Defense Advanced Research Projects Agency under contract number J-FBI-89-094.

2. Related Work

The gate-level ATPG problem has been approached in two major ways: algebraic techniques based on Boolean Differences, literal proposition, etc., and path sensitization techniques that operate on the circuit topology like the D-algorithm [3] and PODEM [4]. In the first set of approaches, the circuit under test is typically represented by some form of a switching function like a truth table, sum-of-products expressions and Karnaugh maps. Test pattern generation is carried out by the manipulation of these representations. Until recently, algebraic techniques have not found favor because these techniques did not scale well with the size of the circuits under test. The main problems were those of generating the representations from circuit netlists and their manipulation when there were a large number of input variables and internal nodes. In addition, the work done in attempting to generate a test for a fault may not be used for the next fault and may be thrown away if there was no test for that fault. These problems have been alleviated to a great extent with the renewed interest in the use of OBDDs for the representation and manipulation of combinational logic. A significant amount of research is being carried out to study efficient (both time and space) implementations of OBDDs [5] and their use in a variety of applications like gate-level/ sequential ATPG [6][7][8], and function verification [9].

Path sensitization based ATPG algorithms operate on the circuit topology. ATPG is the problem of assigning values to inputs such that the circuit output(s) contain different values for the fault-free and faulty cases. The path sensitization algorithms implicitly or explicitly search the entire space of input vectors to find a test. This problem is made difficult because the algorithms in essence are searching 'blindly' for a result (the problem is alleviated to some extent by the use of controllability and observability measures). In addition, faults that are hard-to-detect for one

0-8186-3830-3/93 $03.00 © 1993 IEEE

algorithm may not so difficult for another. This situation occurs because it is the specific decisions an algorithm makes and not the function the circuit implements that make it difficult for the algorithm to find a test for some fault. In addition, detection of redundant faults depends largely on the backtrack limits used, i.e. there is no deterministic method for proving faults to be redundant.

The techniques presented in this paper attempt to tackle the above problems by using OBDDs generated from the circuit topology as the base circuit representation. These algorithms do not use the "set line to value and test" philosophy of path sensitization algorithms and hence there is no 'blind' searching of the solution space. The problem of 'backtracking' is non-existent because these techniques are deterministic and no computation is wasted (there may be significant computation in generating and manipulating OBDDs, however). The duplication of work in generating tests is avoided because all information needed to generate tests is computed and stored in one or two passes through the circuit and then used to generate tests by simple manipulation of the OBDDs. Certain other advantages of the algorithms will be presented in a later section.

Some of the more significant work done in the area of ATPG using OBDDs includes CATAPULT [6], WAVE [7], [8] and TSUNAMI [15]. The algorithm presented in [6] and [7] is based on the theory of Boolean Differences and the calculation of controllability and observability of nodes in the circuit through the use of OBDDs. The authors of these systems present a specific method for handling reconvergent fanout where the Boolean difference chain rule is not applicable. Their representation of circuit outputs in terms of inputs and internal fanout stems enforces extra computation in order to express tests in terms of primary inputs only. Their results compare favorably against the TOPS path sensitization based ATPG algorithm.

The system presented in [8] is based on the theory of difference propagation where only the difference between the 'good' and 'bad' machines is propagated. It is not clear that this approach is optimum in terms of memory usage and has the restriction that faults are considered serially. TSUNAMI also requires a serial consideration of faults. In addition, significant re-evaluation of circuits OBDDs are required when the faults under consideration lie close to circuit primary inputs.

Techniques like EST [13] are added to path sensitization test generators in an effort to reduce the amount of work repeated across faults. Though the technique provides significant improvement in terms of runtimes, it does not provide an improvement of the basic path sensitization algorithms. It is conceivable that the EST system, when presented with a set of completely unrelated hard or redundant faults, will perform only as well as the basic path sensitization algorithm.The algorithms to be presented provide a very convenient method for enumerating all tests for a fault without any duplication of work and for generating tests for multiple faults using the same data structures and algorithms.

The paper is organized as follows: Section 3 provides a brief introduction to OBDDs and the particular implementation being used, Sections 4 and 5 explain the pre-processing step used by A1, Section 6 presents algorithm A1 while Section 7 presents its results on the ISCAS'85 benchmarks. Section 8 discusses enhancements to A1 and presents results from that algorithm. Section 9 discusses the use of testability measures in the context of algebraic ATPG techniques while Section 10 concludes the paper.

3. Binary Decision Diagrams

Much of the interest in algebraic techniques for ATPG has re-surfaced with the renewed interest in the use of OBDDs for Boolean function representation and manipulation. OBDDs are directed acyclic graphs (binary trees) with two terminal nodes, 0 and 1, representing the value of the function. Other forms of functional representation had the disadvantage of growing exponentially in size with the number of input variables. OBDDs, on the other hand, are known to show this property only in rare cases. Another advantage of the OBDDs is that the time complexity of any single operation is bounded by the product of the graph sizes for the functions being operated on. Also, OBDDs provide a canonical representation for switching functions. Bryant [2] has provided very efficient and detailed data structures and algorithms for representing and manipulating switching functions in the form of OBDDs. For a detailed explanation and analysis of the structures and algorithms, refer to [2]. Also, Ross et al [10] report very encouraging results on the use of OBDDs to represent and evaluate logic functions.

An example OBDD for the expression (A+B)C is shown in Figure 1. The nodes represent the switching variables

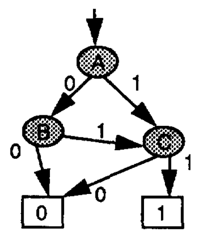

Figure 1: Example OBDD for (A+B)&C

while the arcs carrying values 0 and 1 represent the path through the OBDD when the corresponding (the node the arcs emanate from) variable is assigned the value 0 and 1 respectively. The terminal nodes 0 and 1 represent the value of the function for a specific assignment to the input variables. For example, the path traced when A=0, B=1 and C=1 is shown by the bold arrows.

For the experiments and algorithms presented in this paper, the OBDD representation and manipulation software written by the authors of [11] were used. The software provides functions for generating OBDDs from switching expressions and functions like *Apply* and *Compose* [2] to manipulate the OBDDs. The routines to generate all the needed OBDDs from the circuit netlist were developed as a part of the work performed in this effort.

4. Test Generation Algorithms

Since the use of OBDDs can be memory intensive, a trade-off between memory use and speed becomes important. Memory usage can be minimized by repeating a large part of the computation across all faults but this approach would significantly increase the time required for complete test generation. The techniques presented in this paper attempt to minimize the repetition of work by generating all data structures for all faults in one pass through the circuit.

The timing behavior of the algorithms is highly dependent on the size of the OBDDs used. The OBDD size in turn, depends on the variable ordering [2]. In the worst case (a 'bad' variable ordering), a function with 'n' input variables may require 2^n nodes in the OBDD representation. The results of these experiments (and elsewhere in the literature) have shown that the generation of OBDD representations of large combinational networks can become very expensive (in terms of time) if improper variable orderings are chosen. For the results presented, the circuit variable orderings reported in [12] were used. The use of these orderings resulted in significant improvement of runtimes over the 'natural' ordering case.

The results obtained indicate that, with current OBDD technology, a stand-alone OBDD based ATPG system is not practical for large circuits because of the dependence on variable ordering. Hence, a TPG system that uses a path sensitization based test generator for the easy faults and a OBDD approach for the 'hard' and 'redundant' faults is preferred.

The basic test generation algorithms are based on the difference in representation of good and faulty circuits. However, the systems reported in the literature have not completely solved the problem in terms of achieving a good balance between memory usage and run times. This balance is of great importance when considering large circuits. The work presented here attempts to focus on this problem of a good balance.

5. Data Structure Generation

This section describes the preprocessing step that is common to the algorithms presented in this paper. This preprocessing step basically involves the building of the OBDDs and other data structures needed by the algorithms. All explanations are with respect to the example circuit shown in Figure 2.

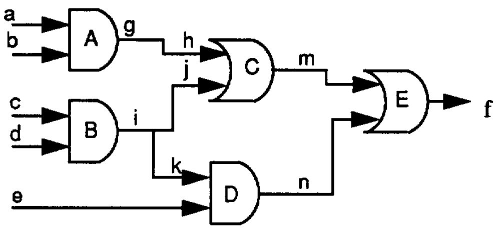

Figure 2: Example Circuit

The circuit has five primary inputs (PIs) 'a' through 'e' and one primary output (PO) 'f.' The remaining lower case symbols represent the internal lines in the circuit. The output of gate B is connected to line 'i' which fans out to lines 'j' and 'k' connected to gates C and D respectively.

The building of the OBDDs occurs using the forward pass. During the forward pass, every gate output is expressed in terms of the PIs and any of the fault variables representing the faults for which test generation is invoked (only if that fault is visible at that gate and is the closest to it). The test generator is supplied with a list of faults to be tested. Each fault is represented by a fault variable and prepended to a global fault variable list. For example, an OBDD is generated for the output of gate A with 'a' and 'b' as the switching variables and stored in the data structure associated with A. Similarly, the output of gate B is expressed in terms of 'c' and 'd.' Following this step, a OBDD for output of gate C is generated by ORing the OBDDs associated with A and B, while a OBDD for gate D is generated by ANDing the input variable 'e' with the OBDD of gate B. Finally, the OBDD of the entire function 'f' is obtained by ORing the OBDDs of gates C and D. No special consideration is needed for the fanout stem during the forward pass. If 'j' was a fault site, a new variable is created at that site and the tree for gate 'C' will be expressed in terms of this new variable and any other PIs visible at that gate's input.

At the end of the forward pass, an OBDD is associated with every gate output and every PO of the circuit. The OBDDs associated with the POs will be used in the actual test generation phase of both algorithms. These OBDDs will be in terms of the PI variables and all fault variables visible at that output. The 'ANDing' and 'ORing' operations are carried out using the Apply function defined by Bryant [2].

In summary, the forward pass generates an OBDD for every gate output with the OBDD representing the gate output in terms of the PIs and fault variables. Only the OBDDs for gates that drive a fault site and the OBDDs for circuit outputs are stored.

5.1 Forward Pass Analysis

This section presents a time complexity analysis of the forward pass. Let G be the total number of gates in the circuit, I be the total number of gate inputs, S be the total number of gates with one input ($S \leq G$) and P be the total number of *Apply* operations carried out in the forward pass. Then,

$$P = I - G + S \qquad (1)$$

The above expression results from the fact that gates with more than two inputs require multiple applications of the *Apply* operation (one for every input except the first two). In other words, an 'n' input gate requires (n-1) *Apply* operations to obtain its output OBDD (except for 1 input gates that require one *Apply* operation). Then, from equation (1),

$$P \leq I \qquad (2)$$

.Let C be a constant that represents the maximum number of inputs to any gate. Then, for a worst case analysis, every gate is assumed to have C inputs. As such,

$$I = CG \qquad (3)$$

Hence,

$$P \leq CG \qquad (4)$$

A single *Apply* is an operation of complexity $O(N^2)$ [2] where N is the size of the largest OBDD generated during the forward pass. Then with P *Apply* operations during the forward pass, the overall time complexity of the forward pass is $O(GN^2)$.

6. Algorithm A1

The algorithm A1 generates a test for a fault by considering two OBDD representations of the circuit; one is the normal circuit OBDD and the second is the faulty circuit OBDD. The second OBDD is the OBDD of the circuit with the fault in place and describes the functionality of the faulty circuit. A test for that fault is an input assignment that causes the two circuits to produce different outputs. Such an input assignment can be found by generating a third OBDD that is the XOR of the fault-free and faulty OBDDs and then finding all input assignments that lead to the '1' terminal node of this third OBDD. This third OBDD is referred to as the **Test OBDD.** Each assignment is a test for that fault. The finding of the input assignment is done by using the *satisfy_one* and *satisfy_all* operations [2]. Figure 3 illustrates the steps in A1.

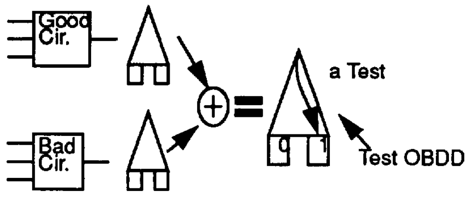

Figure 3: Algorithm A1

As an example, let 'h' s-a-0 and 'i' s-a-1 be the faults under consideration. The forward pass expresses 'f in terms of the PIs and the fault variables 'h' and 'i.' At the end of the forward pass, the output trees will be in terms of the fault variables that are closest to the POs on their respective paths. If there are multiple faults on the same path, only the variable closest to the PO appears in the output tree. The information of the interior faults will be maintained in the entry corresponding to the succeeding fault (going from PI to PO) in the global fault list. As an example, if 'm' and 'h' are fault sites, the output tree will have 'm' as the fault variable and 'm' will be at the head of the fault list. The fault list entry for 'm' stores a tree that expresses 'm' in terms of 'h.' This topological relation requires that the fault list be maintained in the order of levels with the outermost level faults at the head.

To generate a test for 'm' s-a-0, the fault variable 'm' in the 'f' is replaced by the value '0' (using the compose operation) to get f_0, the faulty OBDD. The graphs of 'f' and 'f_0' are XORed to generate the Test OBDD. This OBDD will be in terms of the PIs and possibly the other fault variables. The other fault variables are replaced by their OBDD representations to obtain the Test OBDD in terms of the PIs only. The Test OBDD can then be satisfied to generate a test vector. The idea of partial composition introduced by [5] can be used to speed up the satisfy operation but has not been implemented because the actual benefits, in terms of execution time, of such an approach are not known. With partial composition, many reevaluations of the paths may have to be carried out before a test is found. These extra reevaluations may be more expensive than a single complete composition and a single satisfy operation (the approach used in this paper).

Once the test for 'm' is generated, the corresponding fault variable in the graph of 'f' is replaced by the OBDD of 'm' in terms of the PIs and any other fault variables that lie behind it (in this case, 'h'). The next fault 'i' is then considered in a similar manner. As the number of faults to be considered decreases, the number of fault variables in the graphs of the POs decreases and this fact reduces the number of compose operations to be performed during test generation. Maintaining the fault list in the order described above, allows the test generator to consider all faults without any special list operations. In essence, all faults are treated by a 'backward' pass through the circuit.

For a circuit with multiple outputs, a faulty OBDD must be generated for every PO at which the fault site is visible and Test OBDDs must be generated sequentially until a test is found. The circuit may have outputs that cannot be affected by the fault site. In such a case, those output OBDDs are not considered during test generation. The information regarding the visibility of a gate output from the circuit outputs is maintained in a special data structure (one for every PO). This data structure consists of a bit string where each bit corresponds to one gate in the circuit. The bit corresponding to a gate is '1' if that gate lies on any path to that PO and is '0' otherwise. The OBDD of a PO is considered in the test generation iff the gate that drives that fault has its output visible at that PO. This scheme allows the test generator to avoid unnecessary OBDD operations with POs that are not affected by a particular fault.

The addition of fault variables to the circuit affects the orderings of the PI variables and hence the time for OBDD generation. One approach to alleviating this problem is to include the fault variables when computing the proper input ordering for the circuit. A second approach would be to consider faults in small groups and continue to use the ordering specified for the PIs.

6.1 A1 Algorithm Analysis

Let F be the number of fault sites being considered and C_o be the number of circuit POs. At the completion of TPG for each fault, the number of fault variables in the output OBDDs is decreased and hence the number of Compose operations is also decreased. For all fault sites,

$$CO = 2 \times (FC_o + (F-1)C_o \ldots + C_o) \qquad (5)$$

$$CO \leq 2 \times F^2 \times C_o \qquad (6)$$

$$SA = 2 \times F \times C_o \qquad (7)$$

$$P = 2 \times F \times C_o \qquad (8)$$

where,

CO = Number of *Compose* Operations

SA = Number of *Satisfy* Operations

P = Number of *Apply* Operations

The factor of two appears in the above equations as there can be two faults (sa0 and sa1) at each site.

Then,

Time for *Compose* = $O(F^2N^3)$

Time for *Apply* = $O(FN^2)$

Time for *Satisfy_one* = $O(FN)$

as the individual complexities of the *compose*, *apply* and *satisfy* operations are $O(N^3)$, $O(N^2)$ and $O(N)$ respectively.

Then total time (forward pass + A1) is $O(GN^2 + N^3F^2)$.

6.2 Additional Features of the Algorithm

As mentioned previously, a test for a fault is obtained by satisfying the Test OBDD of that fault. The satisfying operation is done using Bryant's *satisfy_one* operation which generates a single test for that fault. The Test OBDD however stores *all* possible tests for that fault and these can be enumerated using Bryant's *satisfy_all* operation. This additional result is obtained without any extra computation by the algorithm whereas a path sensitization based algorithm would have to backtrack as many times as there are multiple assignment possibilities at gate inputs. By generating all tests for all faults, it is then possible to choose a minimum sized test set that covers all non-redundant faults.

A1 also provides a very convenient framework for generating tests for multiple stuck-at faults. As an example, consider the faults 'h' s-a-0 and 'k' s-a-1 in the circuit of Figure 2. In this case, the faulty circuit OBDD is obtained by replacing 'h' and 'k' fault variables in the OBDD of 'f' by '0' and '1' respectively. The Test OBDD is obtained by XORing the faulty and fault-free graphs and replacing all the other fault variables by their respective OBDDs. The test is determined by satisfying the Test OBDD. This flexibility is not easily available from a Boolean Difference based ATPG system.

7. Results

A preliminary ATPG system using the above algorithm has been implemented in GNU C++ on a SUN 4 workstation. The OBDD routines written by Brace et al [11] have been used with no attempt to optimize their performance. No fault simulation of any type is used. The idea behind the experiment is to target individual faults in circuits and measure the test generation time for these faults.

The results for algorithm A1 are presented in Table I. The faults considered are faults in c432 and are the same faults considered in [6]. The backtrack figures for the TOPS path sensitization TPG were also obtained from [6]. The TOPS system was also run on a SUN 4 workstation. The time taken to complete the forward pass is the dominant factor in total time to 'generate' tests.

Table I: Results for A1 on circuit c432

A1 Time for Forward Pass = 1.21 sec

Circuit	Fault Source	Fault Destntn	Fault type	#bktrks by TOPS	A1 Time for TG of all faults
c432	360gat	379gat	0	52,656	1.79 sec
	319gat	347gat	0	19,664	
	213gat	259gat	0	27,063	
	115gat	379gat	0	52,656	
	112gat	347gat	0	19,664	
	102gat	259gat	0	27,063	

A1 Total Time (forward pass + test generation) = 3.0 sec
TOPS time for all faults = 311 sec
CATAPULT time for all faults = 11 sec

The large difference in the time taken to completion for the TOPS system versus A1 is the fact that TOPS attempts to find tests by making complete searches in the solution space and makes 'bad' decisions along the way. A1 on the other hand, does not have to make 'bad' decisions in generating tests. It follows a completely deterministic method in searching for a solution. Some results for other ISCAS'85 circuits are presented in Table II.

Table II: Results for A1 on ISCAS circuits

Circuit	# Faults	Time for Frwrd Pass	TG Time
c1355	20	42.3s	113.2s
c1908	39	16.3s	162.2s
c3540	41	65.9s	202.8s
c5315	40	50.4s	124.4s

The faults considered were the faults that were dropped by the TGS path sensitization based test generator [14] with a backtrack limit of 300.

8. Algorithm Enhancement

The algorithm presented in the previous sections are able to perform efficiently because of the use of a single forward pass that generates multiple graphs for all circuit outputs in terms of *all* fault variables. Test generation then involves generating the test OBDDs by composing out all fault variables except the one of interest. However, when a large number of fault variables are considered, this sequential composing to remove the fault variables can be an expensive operation especially when the OBDDs are large (for fault sites close to the POs).

One reason for the high cost of compose operations is the fact that the fault variables are added *after* the PI variables and, therefore, are not part of the variable ordering process. A compose requires the variable to be set to the specified value and the resulting tree to be reduced. Some improvement in speed can be expected if the fault variables are created before the PI variables because composing to remove a variable at the top of the tree simply involves extraction of the residues (if there is a single fault variable at the top, composing involves only moving the pointer to the tree root to the appropriate residue). The effect of the location of the fault variables is shown diagrammatically in Figure 4.

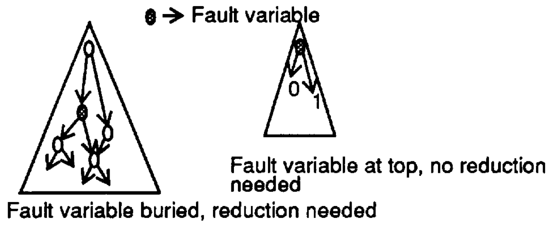
● → Fault variable
0
1
Fault variable at top, no reduction needed
Fault variable buried, reduction needed

Figure 4: Effect of location of Fault Variable

The main problem with this scheme is the fact that creating the fault variables before the PI variables 'disturbs' the optimal ordering of the PI variables. As a result, there is an increase in the time taken to generate the circuit OBDDs. The amount of increase is circuit dependent and may be substantial enough to negate the advantage of reduction in test generation time. The results show that, in general, the time to create the OBDDs of the circuit increases (substantially in case of c3540) but in some cases it actually decreases (c5315). The solution to this problem may be to consider the fault variables when computing a 'good' ordering for the circuit under test.

The version of A1 presented before was modified to allow the creation of the fault variables before the PI variables. As expected, there was considerable reduction in test generation time across all the circuits tested before. The results of the experiment are shown in Table III.

The encouraging feature is the tremendous decrease in test generation time in all cases. For a small number of fault variables, this scheme provides a very efficient method for test generation. Ideally, such a system would be used in conjunction with a random TPG system and a path sensitization based TPG system (to remove easy faults). Another scheme would involve the consideration of faults

Table III: Results of enhancements

Circuit	# Faults	Frwrd Pass Time (sec)	TG Time (sec)	Total
c432	6	1.2	0.22	1.42
c499	23	49.1	94	153.1
c1908	39	48.6	115.3	173.9
c1355	20	72.6	8.3	80.9
c3540	41	250	102	352
c5315	40	40.2	9.9	50.1

in small groups and multiple executions of the algorithm.

9. Use of Testability Measures

None of the algorithm implementations described before use any testability measures. As a result, for circuits with multiple outputs, POs were chosen for fault effect propagation in no particular order. It is possible that for a particular fault, the 'worst' PO (fault site least observable at that output among all outputs) is chosen. As such, a lot of computation may be wasted. By calculating the observability of each fault site at each of the circuit's outputs, it is possible to order the POs (for every fault) in terms of their testability values and choose the POs in decreasing order of testability. The times reported can thus be further improved. When computing the observabilities, it is important to measure a node observability with respect to a single output. This measurement can be made by partitioning the circuit into output cones and generating the observability measure for every node within each cone.

A system as described above was implemented using the SCOAP observability measures. A initial investigation for the ISCAS'85 circuits and the faults previously considered showed that for all circuits (except c3540) the fault sites are visible only at 1 or 2 outputs on an average (that is also a reason for those faults being hard to detect). Even if the fault sites are visible at multiple outputs (c499 and c1355), the observabilities are the same. As such, it does not matter which PO is chosen for fault effect propagation.

In the case of c3540, 15 of the 41 fault sites considered were visible at multiple outputs with varying observabilities. However, the range of observability values were not spread widely. The test generation time for this implementation was reduced by 15% with the use of testability measures. Therefore, measures will be useful for circuits with large number of outputs and a high degree of fanout (c5315). In addition, it is not clear what the relationship between the observability measures and the OBDD sizes.

10. Conclusions

In this paper we have presented algorithms for generating tests for faults in combinational logic based on OBDDs. Current OBDD technology will make the use of stand-alone OBDD based TPG impractical for large circuit descriptions. However, in conjunction with a path sensitization based TPG, the proposed test generators provide a convenient and efficient method for generating tests for hard-to-detect faults (including redundant faults). More research needs to be done in choosing a proper variable ordering for circuit input variable in order to keep the OBDD sizes small. If a suitable variable ordering is found, the algorithms A1 provides an efficient methodology for generating tests for all faults considered. In addition, A1 provides a deterministic method for generating tests for multiple stuck-at faults and other arbitrary fault models.

Creating the fault variables before the PI variables results in substantial decrease in test generation time. If the fault variables are considered as PI variables in computing a 'good' ordering, the algorithm presented in Section 9 provides the most attractive solution.

In all the experiments reported, no testability measures were used to select PO for fault effect propagation. In some cases then its possible that all (n-1) POs will be considered before the correct one is chosen. Use of such measures will allow a better choice of PO and will result in improvement of results. No fault simulation was used during the experiments. Using fault simulation may further reduce the number of faults under consideration.

The major concern with OBDD based approaches is the difficulty of generating representations of large circuits. This problem can be mapped onto a parallel processor where the entire circuit is partitioned into levels and sets of levels assigned to different processors. Each processor then handles a small circuit and can generate the OBDD representation of its section. With this distributed representation, it is then possible to consider the task of test generation. As future work, a scheme for test generation on a parallel processor is under development.

Large circuits present two problems for OBDD based test generation. First, generating the representations of these large circuits can be very expensive unless a 'good' ordering is available. Finding the good ordering is in itself a hard problem. Secondly, large circuits means large number of fault sites. Generating tests for all sites simultaneously requires the use of a large number of extra fault variables and results in significant change in the ordering among the input variables and a consequent degradation in performance. With current OBDD technology, the approach to take would be to use the OBDD test generator on small sets of faults. The 'small' here is circuit dependent.

11. Acknowledgments

The authors would like to acknowledge the help and support provided by Rohit Kapur, CERC, University of Texas at Austin in incorporating the orderings for the various ISCAS'85 benchmarks. Many thanks also go to Ken Butler, Texas Instruments, Dallas, for his very instructive comments on the work done in this area and the use of OBDD based test generation and Bob Klenke for generating the fault lists on the TGS system and his insightful comments on the paper.

12. References

[1] Akers, S. B., *Binary Decision Diagrams*, IEEE Transactions on Computers, Vol. C-27, No. 6, June 1978, pp. 509-516.

[2] Bryant, R. E., *Graph-Based Algorithms for Boolean Function Manipulation*, IEEE Transactions on Computers, Vol. C-35, No. 8, August 1986, pp. 677-691.

[3] Roth, J. P., *Diagnosis of Automata Failures: A Calculus and a Method*, IBM Journal of Research and Development, Vol. 10, July 1966, pp. 278-291.

[4] Goel, P., *An Implicit Enumeration Algorithm to Generate Tests for Combinational Circuits*, IEEE Transactions on Computers, Vol. C-30, No. 3, March 1981, pp. 215-222.

[5] Ross, D. E., K. M. Butler, M. R. Mercer, *Exact Ordered Binary Decision Diagram Size when representing classes of Symmetric Functions*, Journal of Electronic Testing: Theory and Applications (JETTA), Vol. 2., No. 3, 1991, pp. 243-259.

[6] Gaede, R. K., et al, *CATAPULT: Concurrent Automatic Testing Allowing Parallelization and Using Limited Topology*, Proceedings of the 25th Design Automation Conference, June 1986.

[7] Ross, D. E., M. R. Mercer, *WAVE, A Concurrent Approach to Combinational Test Pattern Generation*, Proceedings of the MCC University Research Symposium.

[8] Butler, K. M., M. R. Mercer, *The Influence of Fault Type and Topology on Fault Model Performance and the Implication to Test and Testable Design*, Proceedings of the 27th Design Automation Conference, June 1990, pp. 673-678.

[9] Chang, H. P., W. A. Rogers, J. A. Abraham, *Structured Functional Level Test Generation Using Binary Decision Diagrams*, Proceedings of International Test Conference, September 1986, pp. 97-104,.

[10] Ross D. E., M. R. Mercer, *Calculating Logic Functions Via Ordered Partial Multi-Decision Diagrams*, Personal Communication.

[11] Brace, K. S., R. L. Rudell, R. E. Bryant, *Efficient Implementation of a OBDD Package*, Proceedings of the 27th Design Automation Conference, June 1990, pp. 40-45.

[12] Mercer, M. R., R. Kapur, D. E. Ross, *Functional Approaches to Generating Orderings for Efficient Symbolic Representations*, to appear in the Proceedings of the 29th Design Automation Conference, June 1992.

[13] Giraldi, J., M. L., Bushnell, *EST: The New Frontier in Automatic Test Pattern Generation*, Proceedings of the 27th Design Automation Conference, June 1990, pp. 667-672.

[14] Klenke, R. H., *A Fault Partitioning Method for Automatic Test Pattern Generation in a Novel Distributed Computing Environment*, Center for Semicustom Integrated Systems Technical Report No. 911028.0, October 28, 1991.

[15] Stanion, T., D. Bhattacharya, *TSUNAMI: A Path Oriented Scheme for Algebraic Test Generation*, Proceedings of Fault Tolerant Computing Symposium, June 1991, pp. 36-43.