ATPG tools for Delay Faults at the Functional Level *

S. Tragoudas M. Michael
Electrical and Computer Engineering Department
The Univerity of Arizona
Tucson, AZ 85721

Abstract We propose and evaluate two frameworks for functional level ATPG for delay faults in combinational circuits. Although functional delay fault models have been recently proposed [9, 13, 10], no systematic methodologies for ATPG have been presented in the literature. The proposed frameworks apply to any proposed fault model, and utilize established techniques such as Reduced Ordered Binary Decision Diagrams (ROBDDs) and Boolean Satisfiability (SAT).

1 Introduction

The objective of delay testing is to detect timing defects, which could degrade the performance of the circuit–under–test. In the case where a gate–level description of the circuit–under–test is not available or does not accurately describe the circuit, as is often in the case in embedded core designs with Intellectual Property (IP) considerations, functional–level test generation must be performed.

Previous work on functional test generation for delay faults can be found in [9, 13, 10], where functional delay fault models are proposed. [13] presents an ATPG tool for gate–level descriptions that may contain functional modules with a small number of inputs. The focus of the work in [9] is on fault modeling and not on ATPG. However, a brute–force ATPG approach is also presented in order to evaluate alternative fault models. Our experiments show that it aborts many faults and results into low fault coverage, especially for large circuits. The absence of well–defined, systematic methodologies for functional test generation for delay faults has led to this work.

We propose ATPG tools that utilize established frameworks such as satisfiability (SAT) and reduced ordered binary decision diagrams (ROBDDs) that have already been applied successfully in other CAD contexts. See [5, 8, 6, 1], among others. Both frameworks provide with *exact* ATPG algorithms that guarantee the generation of a test for each non redundant fault. Exact algorithms will be substituted by *approximation algorithms* when the exact algorithms fail due to space or time limitations. Each generated pair of patterns must guarantee *function–robust propagation* (FRP) [10, 13].

Definition: A two–pattern input combination $< u, v >$ is said to function–robustly propagate a transition tI from input I as a tO transition to an output O, if the value on O changes if and only if the value on I changes.

The list of faults consists of all (I, O, tI, tO) combinations. Transitions TI and TO can be either rising (r) or falling (f). The goal is to generate (one or more) pairs of test patterns per fault that guarantee FRP. When all pairs per faults are generated all possible robust path delay faults will be detected under any possible gate implementation. Since this is too time consuming, [13] proposes a model (we call it M1) in which only one pair of test patterns must be generated per fault. [9] proposes that Δ different test patterns are needed per fault, where Δ is a positive integer that varies from function to function but is always in the low hundreds. We call this fault model M2.

ATPG may allow for single–input transitions (SIT) or multi–input transitions (MIT) in each generated pattern. In the case where logic hazards exist, only SIT pairs of test patterns, referred to as SIT tests, can be considered for function–robust propagation. Results in [9] indicate that SIT tests are more effective in detecting path delay faults than MIT tests. Furthermore, any SIT test guarantees FRP whereas for any MIT generated test, the ATPG tool must explicitly verify that the FRP property is satisfied. Thus, MIT tests are generated only when there are no any SIT tests [9]. Our experimental results show SIT test generation results in very high fault coverage.

The rest of this paper is organized as follows. Section 2 presents the ROBDD–based approach for SIT ATPG and Section 3 the SAT–based approach for SIT ATPG. Both sections give exact as well as approximation algorithms. Both frameworks have been extended to handle MIT ATPG [12] but due to space limitations we omit their description. Section 4 gives experimental results on the ISCAS'85 benchmarks, and compares the two frameworks. Section 5 concludes.

*Partially supported by NSF grant CCR-9815229

2 ROBDD–based ATPG

We propose a solution to the functional test generation problem, in which Boolean functions are represented by Reduced Ordered Binary Decision Diagrams (ROBDDs) [3, 2]. Let u and v be ROBDD nodes at levels i and $i+1$, respectively. Let h be also a ROBDD node at level $j, j > i+1$. The $value(u) \in \{0, 1, X\}$ of an ROBDD is 0 if $low(u) = v$, 1 if $high(u) = v$, and X if $low(u) = h$ or $high(u) = h$. If $value(u) = X$ we say that node u bypasses node v in the ROBDD. Section 2.1 presents an exact algorithm. For some faults the algorithm fails to generate any tests due to memory limitations. Section 2.2 presents an approximation algorithm that handles these faults.

2.1 Exact ROBDD–based ATPG

The problem of finding the number of SIT tests for a given (I, O, tI, tO) fault amounts to constructing an ROBDD for an appropriate function and counting the number of paths in the ROBDD that terminate at the terminal node 1.

Given an (I, O, tI, tO) fault, we use f to denote the function of the output O. Let also f_I and $f_{\bar{I}}$ denote the cofactors of f with respect to I and $\bar{I}$, respectively. Let $F = f_I \cdot \bar{f}_{\bar{I}}$, if $(I, O, tI, tO) \in \{(I, O, r, r), (I, O, f, f)\}$ or $F = \bar{f}_I \cdot f_{\bar{I}}$, if $(I, O, tI, tO) \in \{(I, O, r, f), (I, O, f, r)\}$. The ATPG problem (for both models M1 and M2) amounts to finding an assignment of values on the variables that satisfies F.

The general structure of the algorithm is given below. T denotes the number of (I, O, tI, tO) tests found for all faults. M denotes the number of faults for which the ROBDD of F cannot be constructed (i.e., the number of faults missed). Both values are initialized to 0.

FOR each (I, O, tI, tO) fault DO
  IF $(tI, tO) \in \{(r, r), (f, f)\}$ THEN
    construct ROBDD for $F = f_I \cdot \bar{f}_{\bar{I}}$
  ELSE IF $(tI, tO) \in \{(r, f), (f, r)\}$ THEN
    construct ROBDD for $F = \bar{f}_I \cdot f_{\bar{I}}$
  IF ROBDD of F is constructed
    count number of paths to terminal 1
    IF no paths exist fault is redundant
    ELSE add their cardinality in T
  ELSE increment M

Theorem 2.1 *All (I, O, tI, tO) SIT tests can be generated by examining the paths from the root to the terminal node 1 of the ROBDD of F.*

The proof of the theorem is omitted here. The theorem indicates that that the presented algorithm applies to both M1 and M2 models.

2.2 Approximation ROBDD–based ATPG

Given an (I, O, tI, tO) fault, we use f to denote the function of the output O. The algorithm constructs an ROBDD for function f instead of the function F of the previous section. Our intent is to operate on an ROBDD of reduced complexity.

In the M1 model, the algorithm finds a pair of paths per fault. The two paths connect the ROBDD root to the two terminals so that the constraints imposed by the (tI, tO) combination are satisfied. Under the M2 model, Δ different pairs of paths will be constructed for the same fault.

For each fault (I, O, tI, tO), the approach distinguishes between two steps. The first step finds path p from the root to a terminal node, such that $value(I) = 0(1)$ if tI is a r (f) transition. If tO is r (f), the path terminates at the 0(1) terminal node of the ROBDD. The second step finds path p' such that $value(I) = 1(0)$ if tI is r (f), the terminating constant node has a value 0(1) if tO is r (f), and and each remaining variable has a value that satisfies one of the following three conditions: (i) If $value(i) = X, i \in p$ then $value(i) \in \{0, 1, X\}, i \in p'$, (ii) If $value(i) = 0, i \in p$ then $value(i) \in \{0, X\}, i \in p'$, (iii) If $value(i) = 1, i \in p$ then $value(i) \in \{1, X\}, i \in p'$.

At step 1, the algorithms insists that a shortest length path is calculated. (This only requires linear time.) This heuristic measure ensures that many variables are assigned a don't care (X) value, and it is therefore easier to find the second path.

If no such pair of paths is found, the algorithm chooses the next shortest path to be the p path, and attempts to find a matching p' path. A (I, O, tI, tO) test is found *only if a pair of paths is generated.*

It is known that functions such as arithmetic multiplication have ROBDDs with exponential size regardless of the variable order [3]. In such cases, f cannot be represented by an ROBDD, and the approximation algorithm needs to be modified.

An ROBDD is constructed for a simplified function which obtained by preassigning a small cardinality subset $\mathcal{I}'$ of the input variable set $\mathcal{I}$ to either 0 or 1. Once the ROBDD for the simplified function is constructed, a pairs of paths (p, p') is found as described earlier. That way we can only target faults (I, O, tI, tO) with $I \notin \mathcal{I}'$. For the remaining faults, the algorithm selects a small cardinality subset $\mathcal{I}'' \in \mathcal{I} \backslash \mathcal{I}'$, presets all inputs in $\mathcal{I}''$ to either 0 or 1, proceeds as before. This completes the description of the approximation for the M1 model.

The approximation ROBDD–based algorithm has been modified to accommodate the M2 model, where Δ SIT (I, O, tI, tO) tests per fault are needed. For each fault, Δ pairs of paths that result to SIT tests are generated. Once a pair of paths (p, p') has been found,

don't care assignments can be utilized effectively, and the maximum number of SIT tests based on the pair is generated. This is a very powerful optimization step in the ATPG process.

3 SAT–based Functional Test Generation

The problem of finding a set of test patterns for a given (I, O, tI, tO) combination is reduced to solving a CNF satisfiability formula. A different formula is constructed for each (I, O, tI, tO). We use f to denote the function of the output O of the given (I, O, tI, tO) combination, and functions f_I and $f_{\bar{I}}$ to denote the cofactors of f with respect to I and $\bar{I}$, respectively. The problem amounts to finding an assignment that satisfies the formula $f_I \cdot \bar{f_{\bar{I}}}$ (or $\bar{f_I} \cdot f_{\bar{I}}$). The algorithm uses procedure Generate_CNF(F) to transform both cofactors or their compliments to a CNF satisfiability format. We omit here the details of Generate_CNF(F). It follows arguments of predicate logic.

The general structure of the SAT–based functional test generation algorithm for SIT ATPG and model M1 is given below. T denotes the set of (I, O, tI, tO) tests found.

Set $T = \emptyset$.
FOR each (I, O, tI, tO) fault DO
IF $(tI, tO) \in \{(r, r), (f, f)\}$ THEN
  $F = f_I \cdot \bar{f_{\bar{I}}}$.
ELSE IF $(tI, tO) \in \{(r, f), (f, r)\}$ THEN
  $F = \bar{f_I} \cdot f_{\bar{I}}$.
Generate_CNF(F); Solve the SAT formula using [4].
IF CNF–SAT is satisfied add the test vector to T.

Theorem 3.1 *An (I, O, tI, tO) test is found if and only if the respective CNF formula is satisfied.*

The theorem indicates that the algorithm is exact for the M1 model. It is known that for circuits with many fan–out reconvergences (such as the c6288 ISCAS'85 benchmark) the size of the SAT formula is prohibitive. In such cases, the exact algorithm is transformed to an approximation algorithm that consists of two phases: The first phase simplifies the function by assigning a small subset $\mathcal{I}'$ of the input variable set $\mathcal{I}$ to either 0 or 1. Then all faults (I, O, tI, tO), $I \notin \mathcal{I}'$ are targeted. The second phase works similarly, by initially simplifying the function after assigning a small subset $\mathcal{I}'' \subseteq \mathcal{I} \setminus \mathcal{I}'$ to either 0 or 1, and then targeting all the remaining faults.

The above description refers to model M1 and SIT ATPG. The same technique, i.e., selecting subsets of the input variables and assigning them to either 0 or 1 is used to derive an approximation SIT ATPG algorithm for model M2. If the SAT solver returns a partial input variable assignment we take advantage of the don't cares in order to generate additional test patterns for the same fault.

4 Experimental Results

We present here an experimental comparison and evaluation of the proposed methodologies. The ATPG tools were implemented in C, and run on a 270MHz SUNW, Ultra–5 workstation . We executed on the ISCAS'85 benchmarks. Our focus here is on comparing the two frameworks for SIT ATPG under both M1 and M2 models, in terms of fault coverage and time performance. Each fault at the functional level corresponds to an (I, O, tI, tO) tuple.

We do not provide any results on the gate–level path delay fault coverage. [9] determine how many tests per (I, O, tI, tO) tuple are needed in order for the gate–level path delay fault coverage to be satisfactory for some gate–level implementation. Our goal is to evaluate the performance of the proposed tools for SIT ATPG on established models that indicate that one or more SIT tests must be generated per (I, O, tI, tO) tuple.

Table 1 gives results for the exact and the approximation ROBDD–based ATPG algorithms of Section 2. We consider the M1 model where one test per fault is generated. It is known that the ROBDD of circuit c6288 cannot be constructed. We constructed ROBDDs for c6288 by simplifying the function after randomly preselecting inputs (approximately 23%) , as described in Section 2. (This is noted by an asterisk in Table 1.)

Column 2 lists the number of faults for each circuit. The ROBDDs were generated using the CUDD Decision Diagram Package [11]. Column 3 gives the size of the ROBDD (no. of nodes). Columns 4–6 list the results taken for the exact algorithm and Columns 7–9 the results taken for the approximation algorithm. The number of generated tests for each approach are listed in Columns 4 and 7. Columns 5 and 8 give information about the time performance of the ATPG tools. Columns 6 and 9 give the fault coverage obtained by each of the proposed tools.

It is interesting to observe that the fault coverage is significantly high in both cases. In fact, this percentage is even higher when considering that many faults are SIT redundant. The latter information is given later on Table 2 when the SAT–based tool is evaluated.

In both approaches, a fault is aborted when the corresponding ROBDD cannot be constructed due to memory limitations. The approximation algorithm aborts a fault when no pair of paths is found within

5 minutes. The information in Columns 6 and 9 very clearly demonstrate that the both the fault coverage and the time performance are better when the exact ROBDD–based algorithm is used. The only exception is c6288.

An important observation is that the faults covered by the exact algorithm are not a superset to those covered by the approximation. This led us to the following approach, we call it the combined ROBDD–based approach. For every fault, the exact algorithm was performed first. When the exact algorithm fails to generate tests for a non-redundant fault the approximation algorithm was applied. For cases where the approximation algorithm was unable to construct the ROBDD for the given function the variable preassignment heuristic allowed for test generation. Columns 10 and 11 list the total number of tests found and the fault coverage for the combined ROBDD–based approach. Observe the increased fault coverage. The time performance of the combined scheme is very close to the time performance of the exact solution.

Next, we proceed to evaluating the SAT–based framework for SIT ATPG under the M1 model. The results are listed in Table 2. For circuit c6288 (marked with an asterisk), the SAT formula was obtained by function simplification through variable presettings, as described in Section 3. (Approximately 28% of the variables were preset each time.)

Table 2 provides with information similar to that in Table 1. The results were obtained using the SAT solver in [4]. For most of the cases, the solver terminated in microseconds. For circuits with many reconvergencies, such as c3540 and c6288 the solver required an average of 1.3 seconds, ranging from microseconds to almost minutes for a few SAT instances. The results show that the time performance of the SAT–based ATPG is comparable (always slightly worst) to the performance of the ROBDD–based ATPG of Section ??. Note that the SAT–based ATPG does not abort any faults, i.e., the fault coverage listed in Column 5 of Table 2 is the maximum possible for SIT ATPG for model M1. Column 6 lists the percent of redundant faults in each circuit.

In contrast, the fault coverage obtained by the test generation procedure in [9] is consistently poor for the ISCAS'85 circuits. The fault coverage for SIT ATPG on model M1 is listed in Column 7 of Table 2. We observed that the time performance of the approach in [9] is very poor. The approach terminated in less than a day for very few circuits. Therefore the results of Column 7 for circuits c2670, c3540, c5315, c6288 and c7552 were obtained based on the number of faults targeted and the number of tests found within the first day of the execution of the tool, i.e., we assumed that the ATPG tool would cover the same percentage of faults in the list if it were to execute forever.

We also experimented for SIT ATPG under model M2. Table 3 provides results obtained for the ROBDD–based approach and the SAT–based approach. Note that the exact ROBDD-based solution returns the maximum number of SIT tests for every fault. Therefore, its time performance is the same for both M1 and M2 models.

We have applied the combined ROBDD–based algorithm on the M2 model. The results are listed in Columns 2 and 3 of Table 3. Column 2 lists the average number of SIT tests per fault (Δ) found in each circuit. This number is calculated by dividing the total number of SIT tests by the number of non-redundant faults. Column 3 gives the time performance of the SIT ROBDD–based approach for model M2. We have used a time limit of 5 minutes per fault every time one of the approximation algorithms was invoked. Note that the number of faults that have at least one SIT test is, of course, the same for both M1 and M2 models (see Table 1,Column 10). Thus, the fault coverage of ROBDD–based SIT ATPG for M2 is the same as the one shown in Column 11 of Table 1 for ROBDD–based SIT ATPG for model M1. Columns 4 and 5 list the results obtained for the SIT SAT–based approach for model M2. Note that the SAT–based tool is no longer an exact algorithm for model M2 (see Section 3). (Again, we have a time limit of 5 minutes for every targeted fault). It is clear that the ROBDD–based approach is superior to the SAT–based approach, both in terms of time and number of tests generated. The performance of the SAT–based technique is attributed to the heuristic approach of variable preassignment that we used in order to generate more than one SIT tests.

5 Conclusions

We presented two frameworks for functional ATPG for path delay faults. They are based on ROBDDs and SAT, respectively. Our experimental study shows that most faults have many SIT tests. The SAT–based tool is recommended when one SIT test per fault is required. Otherwise, the ROBDD model is preferable.

References

[1] D. Bhattacharya, P. Agrawal, D. Agrawal, "Test Generation for Path Delay Faults using Binary Decision Diagrams", *IEEE Trans. on Computers*, vol. 44, pp. 434–447.

[2] K. Brace, R. Ruddel, R.Bryant, "Efficient Implementation of a BDD Package", *in Proc. 1990 Design Automation Conference*, pp.40–45.

[3] R. Bryant, "Graph–based Algorithms for Boolean Function Manipulation", *IEEE Transactions on Computers*, Vol.C–35, No.8, August 1986, pp.677–691.

[4] J. Crawfort, *NTAB:Propositional Satisfiability Checker*, CIRL, The University of Oregon, 1996.

[5] S. Devadas, K. Keutzer, S. Malik, A. Wang, "Computation of floating mode delay in combinational circuits: Practice and Implementation", *IEEE Transactions on Computer–Aided–Design*, 12(12):1924–1936, December 1993.

[6] R. Drechsler, "BiTes: A BDD based Test Pattern Generator for Strong Robust Path Delay Faults", *in Proc. 1994 EDTC*, pp. 322–327.

[7] J. P. Marques Silva, K. A. Sakallah, "GRASP–A New Search Algorithm for Satisfiability", *IEEE Transactions on Computers*, 1996, pp.220–227.

[8] P. C. McGeer, A. Saldanha, R. K. Brayton, A. Sangiovanni–Vincentlli, "Delay models and exact timing analysis. In T. Sasao, editor *Logic Synthesis and Optimization*, pages 167–189, Kluwer Academic Publishers, 1993.

[9] I. Pomeranz, S. M. Reddy, "Functional Test Generation for Delay Faults in Combinational Circuits", *in Proc. 1995 Intl. Conf. on Computer–Aided–Design*, Nov. 1995, pp.687–694.

[10] I. Pomeranz, S. M. Reddy, "On Testing Delay Faults in Macro–based Combinational Circuits", *in Proc. 1994 Intl. Conf. on Computer–Aided–Design*, Nov. 1994, pp.332–339.

[11] F. Somenzi, *CUDD:CU Decision Diagram Package*, Release 2.2.0, Department of Electrical and Computer Engineering, The University of Colorado at Boulder, 1998.

[12] S. Tragoudas, M. Michael, "ATPG Tools for Delay Faults at the Functional Level", Technical Report CENG–TR–98–118, ECE Dept., The University of Arizona.

[13] B. Underwood, W. O. Law, S. Kang, H. Konuk, "Fastpath: A Path–Delay Test Generator for Standard Scan Designs", *in Proc. 1994 Intl. Test Conf.*, Oct. 1994, pp.154–163.

Table 1: Exact and Approximation ROBDD–based SIT ATPG for model M1.

Circuit	Num Flts	BDD Size	Exact ROBDD			Approx. ROBDD			Comb. ROBDD	
			Num tests found	Time (hours)	% Flt Covrd	Num tests found	Time (hours)	% Flt Covrd	Num tests found	% Flt Covrd
c432	832	1064	801	0.03	96.2	783	0.49	94.1	804	96.6
c499	4329	25866	3915	3.42	90.4	3801	6.79	87.8	3915	90.5
c880	1368	4053	1320	0.12	96.5	1315	1.09	96.1	1321	96.6
c1908	3224	5526	3148	1.65	97.6	3130	2.31	97.1	3153	97.8
c2670	4018	1174	3955	1.98	98.4	3941	2.43	98.1	3956	98.5
c3540	2730	23828	2483	3.10	90.9	2460	5.35	90.1	2493	91.3
c5315	10718	1719	10655	6.52	99.4	10652	9.71	99.4	10662	99.5
c6288*	2337	38511	1655	9.62	70.8	2085	8.22	89.2	2085	89.2
c7552	13932	2112	13811	9.14	99.1	13779	11.74	98.9	13815	99.2

Table 2: SAT–based SIT ATPG for model M1.

Circuit	Num Flts	Exact SAT–based				[9] approach
		Num tests found	Time (hours)	% Flt Covrd	% Flt Red.	% Flt Covrd
c432	832	807	0.06	100	3.1	36.2
c499	4329	3917	5.80	100	9.5	22.1
c880	1368	1325	0.27	100	3.1	41.6
c1908	3224	3159	2.03	100	2.1	31.4
c2670	4018	3958	2.65	100	1.5	27.8
c3540	2730	2510	4.26	100	8.1	20.1
c5315	10718	10675	8.68	100	0.4	27.3
c6288*	2337	2110	7.41	90.3	< 9.7	22.1
c7552	13932	13820	11.76	100	0.8	25.9

Table 3: ROBDD–based Vs. SAT–based SIT ATPG for model M2.

Circuit	ROBDD–based		SAT–based	
	Avg. Δ per fault	Time (hours)	Avg. Δ per fault	Time (hours)
c432	290	0.06	198	0.63
c499	315	4.47	200	7.70
c880	401	0.25	278	1.40
c1908	353	2.24	191	4.33
c2670	489	3.12	275	5.69
c3540	258	4.81	134	7.89
c5315	427	8.29	259	13.11
c6288*	231	18.33	156	> 24
c7552	375	11.10	221	17.45