Constrained ATPG for Broadside Transition Testing *

Xiao Liu and Michael S. Hsiao
Department of Electrical & Computer Engineering
Virginia Tech, Blacksburg, VA 24061
{*liux, hsiao*}*@vt.edu*

Abstract

In this paper, we propose a new concept of testing only functionally testable transition faults in Broadside Transition testing via a novel constrained ATPG. For each functionally untestable transition fault f, a set of illegal (unreachable) states that enable detection of f is first computed. This set of undesirable illegal states is efficiently represented as a Boolean formula. Our constrained ATPG then incorporates this constraint formula to generate Broadside vectors that avoid those undesirable states. In doing so, our method efficiently generates a test set for functionally testable transition faults and minimizes detection of functionally untestable transition faults. Because we want to avoid launching and propagating transitions in the circuit that are not possible in the functional mode, a direct benefit of our method is the reduction of yield loss due to overtesting of these functionally untestable transitions.

1 Introduction

The stuck-at fault model [1] is insufficient for catching speed-related failures, as more chips are now more vulnerable to such failures due to higher clock rate, shrinking geometries, longer wires, increasing metal density, etc. The three most prevalent fault models for delay testings are: *transition fault* [2], *path delay fault* [3], and *segment delay fault* [4].

A transition fault at a node X assumes a large delay at X such that the transition at X will not propagate to a flip-flop or primary output within the clock period. The path delay fault model assumes a small delay at each gate, and the cumulative effect of gate delays along a specific path from a primary input to a primary output is considered. If the cumulative delay exceeds the clock period, the chip will fail a test that exercises this particular path. Segment delay fault targets path segments instead of complete paths. Of these, transition fault is the most practical, and commercial tools are available for computing such tests. Transition tests have been generated to improve the detection of speed failures in microprocessors [5] as well as ASICs [6]. In this paper we only target at the transition fault model.

At each line in the circuit two transitions are possible: *slow-to-rise* and *slow-to-fall*. A test pattern for a transition fault consists of a pair of vectors, {V1,V2}, where V1 (*initial vector*) is required to set the target node to an initial value and V2 (*test vector*) is required to launch the appropriate transition at the target node and also propagate the fault-effect to a primary output [2, 12]. The second vector is identical to a test vector that detects the corresponding stuck-at fault on the node.

In general, (non-scan) functional testing can be impractical for larger circuits in that large test sets may be required to achieve a desirable fault coverage. As a result, at-speed AC scan testing has

*This research was supported in part by a grant from Intel Corp., and in part by NSF under contracts CCR-0196470 and CCR-0305881

been widely used in the industry to detect delay-induced defects. Compared to functional testing, scan-based testing for delay faults can decrease the overall ATPG complexity and cost, since both controllability and observability on the flip-flops are enhanced. Nevertheless, the drawback of scan-based delay tests lies in two folds: hardware overhead and potential yield loss. In [18], the author reported that scan-based testing may fail a chip due to the delay faults that do not affect the normal operation, and thus it is unnecessary to target those functionally unsensitizable faults. In other words, we want to avoid failing a chip due to a signal transition/propagation that was not intended to occur in the functional mode. Moreover, a scan test pattern, though derived from targeting functionally testable transition faults, can incidentally detect some functionally untestable transition faults if the starting state is an unreachable state.

Several papers [18, 19] have discussed the relationship between functional testing and scan-based testing. However, from our knowledge, currently there is no quantitative analysis on functional untestable transition faults and scan-based testing. In this paper, we describe a novel constrained ATPG algorithm for transition faults. Two main contributions of our work are: (1) the constrained ATPG only targets at the functionally testable transition faults and *minimizes* detection of any identified functionally untestable transition faults; (2) the constrained ATPG can identify more functionally untestable transition faults than the conventional transition ATPG tools. The first contribution (the constrained ATPG) enables us to derive transition vectors that avoid the illegal starting states that can detect any of the identified untestable transition faults, while the second contribution helps us to maximize the state space that we need to avoid. Because we want to avoid launching and propagating transitions via scan that are not possible in the functional mode, a direct benefit of our method is the reduction of yield loss due to overtesting of these functionally untestable transitions. Our experimental results showed that significantly more functionally untestable transition faults can be avoided in the final test set.

The rest of the paper is organized as follows. Section 2 gives an overview of the three different scan-based transition test application techniques and explains the motivation of this work. Section 3 presents an implication engine we developed to identify a subset of the functionally untestable transition faults. Section 4 proposes a new constrained ATPG algorithm targeting at only functionally testable faults and simultaneously avoiding the functionally untestable transition faults. Section 5 reports our constrained ATPG results, and compares them with a conventional transition ATPG engine. Finally, Section 6 concludes the paper.

2 Background: scan-based transition fault testing

Transition tests can be applied in three different ways: **Enhanced-Scan** [11], **Skewed-load** [7] and **Broadside** [8].

For enhanced-scan transition testing, two vectors (V1, V2) are scanned in and stored in the scan FFs simultaneously. Enhanced scan transition test has two primary advantages: higher fault coverage and lower test data volume. Enhanced-scan gives better fault coverages than both skewed-load transition test and broadside transition test because there is no dependency between the two vectors in enhanced-scan test pattern. In addition, compact transition tests can be achieved to reduce the test data volume and application time for enhanced scan [21]. However, one drawback of the Enhanced-scan testing is that a hold-scan design [11] is needed. Although the hold-scan model has been used in some high-performance circuits, such as microprocessors, the area overhead and the additional routing requirement for control signal still limit the hold-scan cell model from wide use in the ASIC community. Moreover, because any state combination for V1 and V2 is possible, many untestable faults are detected.

For skewed-load transition testing (*also called last shift*) [7, 9, 10], an N-bit vector is loaded by shifting in the first N-1 bits, where N is the scan-chain length. The last shift clock is used to launch

the transition. This is followed by a quick capture. For skewed-load testing, only one vector is stored for each transition pattern in tester scan memory; the second vector is a shifted version of the stored vector. Therefore, skewed-load testing is constrained by the correlation of the bits in the test pattern based on scan chain ordering. Figure 1 shows a simple example where the *slow-to-fall* on line *d* is untestable in Skewed-load testing. To detect the *slow-to-fall* fault on line d, we need

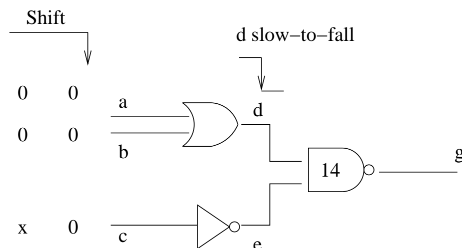

Figure 1: Untestable Fault in Skewed-load

to set the second vector V2=000 to detect the *d s-a-1*, therefore the initial vector must be 00X, the previously shifted version V2. However, this V1, 00X, is unable to initialize the line *d* to logic 1. Since 000 is the only vector that can detect the *d s-a-1* fault, thus *d slow-to-fall* is untestable in skewed-load testing. Based on this observation, skewed-load may miss some functionally testable faults because of the data dependency between the two vectors, resulting in *undertesting* of the functionally *testable* faults.

For broadside transition testing (*also called functional justification*) [8], the first vector is scanned in and the second vector is derived as the circuit response to the first one. For broadside testing, after the first vector is scanned in and applied to the circuit, two clock cycles need to be pulsed: the first to launch the transition and the second to capture the circuit response. PI/PO changes would be made simultaneously with the first clock pulse if necessary [22]. Because it requires neither hold-scan design nor skewed shifting, this has been widely applied for transition testing. In this paper, we will consider the Broadside model only.

All three scan-based testing methods can suffer from yield loss due to a chip failing on detection of functionally untestable transition faults. In general, we can relate the functional test and the various scan-based tests in terms of their untestable transition faults as depicted in Figure 2. In

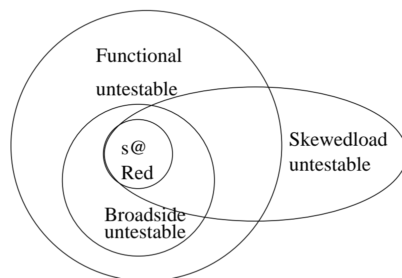

Figure 2: Functional testing vs. Scan-based testing

this figure, the circle at the center depicts the set of redundant stuck-at fault set in the circuit, while each of the outer circle/oval represents the fault set that **cannot** be detected by a particular test method, Based on this relationship, some observations can be made:

1. For every redundant stuck-at fault in the circuit, there must be at least one corresponding functionally untestable transition fault, which is clearly untestable by any test method.

2. All scan-based test methods will incidentally detect some functionally untestable transition faults, because either the states they scan in may be functionally unreachable or the state combination is not functionally possible.
3. If a transition fault is untestable by the Broadside model, it will be definitely untestable in the function mode. Conversely, every functionally testable transition fault will be detectable under broadside testing.
4. Skewed-load can potentially miss some functionally testable faults due to the correlation between the vector and its shifted version.
5. Some of the untestable transition faults in Broadside may be detectable in skewed-load test, and vice versa.

3 Functionally untestable faults identification

In general, functionally untestable transition fault identification in sequential circuits is of the same complexity as sequential ATPG, which is of exponential complexity in terms of the size of the circuit. In this section, we describe a novel untestable transition fault identification method by combining a transition fault implication engine and Broadside ATPG.

In [23], a method for identifying untestable stuck-at faults in sequential circuits by maximizing local conflicting value assignments has been proposed. The technique first quickly computes a large number of logic implications across multiple time-frames and stores them in an implication graph. Then the algorithm identifies impossible combinations of value assignment locally around each gate in the circuit and those redundant stuck-at faults requiring such impossibilities.

For identifying functionally untestable transition faults, in addition to searching for the impossibilities locally around each gate, we also check the excitability of the initial value in the previous time frame. Thus, the implication engine in [23] can be extended to quickly identify a large set of untestable transition faults in the circuit.

Although the transition fault implication engine helps us in identifying a large number of untestable transition faults in the circuit, it may be incomplete (i.e., not all untestable transition faults are identified). To avoid the high cost of calling a functional-mode sequential ATPG to identify the other untestable transition faults, we use a two-time-frame Broadside ATPG instead. As we discussed before, if a transition fault is untestable in Broadside testing, then it is definitely untestable in the function mode as well. So the transition fault implication engine and broadside ATPG can be combined to estimate the total number of functionally untestable transition faults. This saves us from having to invoke a full sequential ATPG. Figure 3 gives us a graphical illustration. In this figure, the outer rectangle represents the total number of functionally untestable

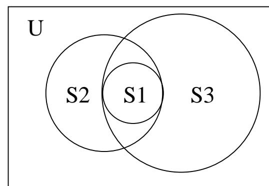

Figure 3: Approximation of Functionally Untestable Transition Faults.

transition faults in the circuit, and region S1 contains the redundant *stuck-at* faults identified by stuck-at fault implication-based method [23]; region S2 contains the untestable transition faults by our new transition fault implication engine and S3 is the set of untestable transition faults identified by the Broadside ATPG, which is typically more complete than S2 for most of the benchmarks tested. Note that the new implication-based method may identify some functionally untestable

faults that ATPG misses, and vice versa. The union of S1, S2, and S3 can give us a close approximation of the functionally untestable transition faults within the circuit.

4 Constrained ATPG For broadside testing

In this section, we describe how we formulate the illegal states as a formula and use it to speed up the ATPG process to generate effective test vectors that avoid functionally untestable transition faults. A side benefit is that it also helps us to identify the functionally untestable faults (region S3 in Figure 3) for broadside testing.

As we described in Section 2, broadside vectors consists of initial state S1, primary input vectors PI1 and PI2. The intermediate state in the second time-frame is derived directly from S1 and PI1. PI2 in the second time-frame is independently applied. In our broadside ATPG, we unroll the sequential circuit to two time-frames and attempt to generate a vector, which consists of state inputs (S), primary inputs in the 1st time frame (PI1) and primary inputs in the 2nd time frame (PI2). We denote a test vector V^j in the unrolled circuit as (S^j,$PI1^j$,$PI2^j$).

4.1 Problem formulation

Given the set of functionally untestable transition faults, F_u, we want to make sure that the vectors generated will not incidentally detect any fault in F_u. A naive approach is to fault simulate the faults in F_u whenever a vector is obtained, and the ATPG engine would backtrack if some faults in F_u are incidentally detected. However, this naive approach can be computationally expensive. To reduce the expense, instead of focusing on F_u, we project each fault in F_u onto the state space to identify subspaces that will detect them. Subsequently, the ATPG only needs to avoid searching in the identified subspaces. We note that any state s that can detect any fault $f \in F_u$ would be an unreachable state, since fault f would otherwise be functionally detectable.

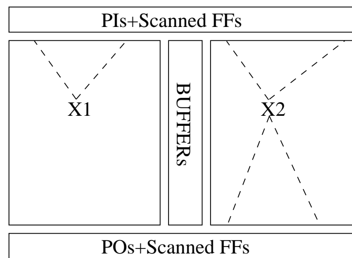

Figure 4: Broadside ATPG

Figure 4 illustrates our broadside testing model. In this model, let us consider the detection of the *slow-to-rise* fault on line X, (We use $X1$ and $X2$ to represent line X in the first and second time-frame respectively.) We need to satisfy the following two objectives simultaneously:

1. Excite $X1$ s-a-1 fault in the first time-frame and detect $X2$ s-a-0 in the second time frame.
2. Avoid detection of any transition faults in F_u by making sure the search space does not overlap with the subspaces that can detect faults in F_u.

The second objective of avoiding detection of functionally untestable transition faults is key to the constrained ATPG. We first identify the state subspace that can detect the functionally untestable transition faults, and this subspace is represented as a Boolean formula. Suppose that the circuit has n flip-flops s_1,...,s_n, a formula in conjunctive normal form (CNF) is used to represent the subspace. A CNF formula over the n binary variables is the conjunction (AND) of m clauses

ω_1,...,ω_m, each of which is the disjunction (OR) of one or more literals, where a literal is the occurrence of a variable or its complement. Each clause in the formula represents a subspace that the ATPG must avoid. Therefore, at any given time during the ATPG search, no clause in the formula should evaluate to false (where each literal in the clause is valued to 0).

The subset of states that can detect any functionally untestable transition fault is obtained as follows.

1. Run the Implication Engine (TRANIMP) to identify the set of the functionally untestable transition faults (F_{u1}).
2. Fault simulate with a 5,000 random vector set, V_{5000}. Remove the easy functionally testable faults and record any vector in V_{5000} that detects one or more of the functionally transition faults in F_{u1}. Place such vectors in $V_{illegal}$.
3. For each vector in $V_{illegal}$, deduce the illegal state (IS) and negate it to construct the corresponding clause ω as a constraint.
4. Combine all the constraint clauses to form the CNF formula ϕ.

Because obtaining the complete set of states that can detect any functionally untestable transition fault is computationally expensive, we obtain only a subset of states via random simulation, although this can be compensated by dynamically adding the newly identified illegal states in ATPG process to the CNF formula ϕ. As a result, there may still exist states outside the subset that may still detect a functionally untestable transition fault. However, our experiments showed that the subset is sufficient to significantly reduce the incidental detection of functionally untestable faults.

Table 1 illustrates an example of how the constrained CNF formula ϕ is constructed from the set of illegal states. During the random vector simulation, if a vector v^j detects one or more functionally untestable faults identified by the implication engine (TRANIMP), we record the state variables of v^j, deduce the corresponding constrained clause ω_j from the specified state variables. For example, in Table 1, three illegal states are reported. The first illegal state is $s_1 s_2 s_3 s_4 s_5 s_6 =$ 111XXX. The illegal state subspace can be represented simply as $s_1 \cdot s_2 \cdot s_3$. Negating this conjunction gives us the clause $\omega_1 = s_1' + s_2' + s_3'$. This clause essentially restricts any solution must fall within the subspace expressed by ω_1. For example, when $s_1 s_2 s_3 = 111$, ω_1 would evaluate to 0, indicating that an illegal state space has been entered. The clauses for the other two illegal states can be obtained in a similar manner. Finally, the constrained CNF is formed by the conjunction of all the constrained clauses. Thus, at any time, none of the constraint clauses must evaluate to 0 to ensure that the search remains *outside* of any of the illegal state spaces.

Table 1: CNF Formula Construction

Illegal States						*Constrained Clauses*
s_1	s_2	s_3	s_4	s_5	s_6	
1	1	1	X	X	X	$\omega_1 = s_1' + s_2' + s_3'$
X	1	0	X	1	X	$\omega_2 = s_2' + s_3 + s_5'$
X	0	0	X	X	0	$\omega_3 = s_2 + s_3 + s_6$

ϕ=$\omega_1\omega_2\omega_3$=$(s_1'+s_2'+s_3')(s_2'+s_3+s_5')(s_2+s_3+s_6)$

4.2 Constrained ATPG algorithm

Next, we will discuss how the constrained CNF formula ϕ helps us to speedup the ATPG process (and identify extra functionally untestable faults also). During the ATPG process, we must make sure that no clause in ϕ ever evaluates to false (ie., all literals in a clause evaluates to false).

Whenever we make a decision on a state variable s_k, we apply this decision assignment to all the constrained clauses in ϕ that contain s_k. Application of this assignment may result in some unit clauses (a unit clause is an unsatisfied clause with exactly one remaining unassigned literal left). This remaining literal is called an *implication*. The implied variable automatically becomes the next decision variable. We also check whether there is conflict (where one clause evaluates to false). If there is a conflict, backtrack immediately. A test vector is said to be generated for a transition fault X if it excites the fault X1 s-a-1 and detects faults X2 s-a-0, also it satisfies the constrained CNF ϕ.

Table 2: Implication on Decision Assignment

$CurrentDecision$	$ImpliedAssignment$
s_1=1	$None$
s_6=0	$None$
s_5=1	$None$
s_3=0	$s_2 = 0, s_6 = 1$
Backtrack s_3=1	$s_2 = 0$

Using the constrained CNF formula ϕ shown in Table 1, we explain the implication process on state variables in Table 2. After assigning s_1=1, we apply this assignment to the unsatisfied clauses containing s_1, no unit clause results, thus no implication can be made on other state variables. Next, suppose the ATPG makes the subsequent decisions s_6=0 and s_5=1. Applying these to ϕ still results in no implication. The next decision made by the ATPG is $s_3 = 0$. For clause ω_2, we can directly imply s_2=0 (because to satisfy clause (s_2'+s_3+s_5')=1, s_2 has to be 0). Consequently, after the direct implication s_2=0, clause ω_3 evaluates to 0 because all literals in ω_3 has evaluated to 0. Therefore, we backtrack to the previous decision and assign s_3=1 and continue the ATPG process.

During the ATPG backtrace, an implication stack is dynamically updated to record the implication list of earlier backtrace choices similar to the algorithm described in [17]. We maintain two dynamic implication lists: $Imp_{X1=0}$ for storing the implications that are necessary for setting X1=0, and $Imp_{X2=1}$ for storing implications necessary for setting X2=1. If there is a conflict between $Imp_{X1=0}$ and $Imp_{X2=1}$, then we declare X *slow to rise* untestable. A conflict is observed when $X1 = v$ implies $X2 = v$ (v can be either logic 0 or 1). In other words, a transition is not possible on line X. Otherwise, we try to generate the test vector V for detecting X2 s-a-0. If V can incidentally excite X1 s-a-1 in the first time-frame as well, we mark X *slow to rise* as potentially detected. We simulate the generated test vector *V* to see whether it detects any identified functionally untestable faults in ,if not, we say X *slow-to rise* is detected. Otherwise, we deduct the illegal state from vector *V* and update our CNF formula ϕ, then continue to backtrace to excite X1 s-a-1. If not successful, we declare X *slow to rise* untestable.

5 Experimental results

We implemented a constrained broadside ATPG based on PODEM [16] in C++, as well as the implication-based untestable transition fault identification, also in C++. We further analyzed the effectiveness of our ATPG algorithm by comparing it with a conventional Broadside ATPG. Experimental data was collected for full-scan versions of ISCAS89 benchmark circuits on a 2.8GHz Pentium-4 with 512 MB of memory, running the Linux operation system.

First, Table 3 reports the functionally untestable transition faults identified by using our transition fault implication engine (TRANIMP). In order to see the effectiveness of the implication engine, we list the number of functionally untestable transition faults identified while consider-

Table 3: Functionally Untestable Faults Identified by Implication(TRANIMP)

circuit	$faults$	$1-TF$	$3-TF$	$5-TF$	$7-TF$
s344	1040	0	47	66	66
s1423	4288	33	387	387	387
s5378	15680	351	3673	3695	3695
s9234	29086	1327	6533	7415	7415
s13207	44130	1303	8900	14530	14540
s35932	103842	9536	11255	11255	11255

ing different number of time frames for sequential circuit. The third column presents the number of untestable transition faults identified while only one time-frame is considered. The last three columns show the numbers of functionally untestable transition faults discovered while considering 3-time-frames, 5-time-frames and 7-time-frames, respectively. For example, in circuit s5378, one-time-frame implication found 351 functionally untestable transition faults. When the number of time-frames increases to 7, the number of functionally untestable faults identified increased to 3695.

Several interesting issues to note are listed below:

1. In general, the number of functionally untestable transition faults are much greater than the number of redundant stuck-at faults in the circuit. This is due to the functional dependency between the vectors of each test pattern for broadside-testing.
2. The number of identified untestable transition faults increases with the number of time frames considered in the static implication graph.
3. Except for circuit s13207, the number of identified untestable transition fault saturates when the number of time frames increases to 7. Therefore, we can expect the number of untestable transition faults to not increase too much even if the number of time frames continue to increase.

Table 4: Effectiveness of Random Vectors On Avoiding Functionally Untestable Faults

$circuit$	$Total$ $faults$	$Func\,Unt$ $faults$	5000 RandVec		Pruned RandVec		
			OVT	DET	OVT	DET	UND
s344	1040	66	19	907	0	906	134
s1423	4288	387	160	2843	0	1163	3125
s5378	15680	3695	1415	9080	0	0	15680
s9234	29086	7415	1590	9875	0	0	29086
s13207	44130	14542	5306	12186	0	0	44130
s35932	103842	11255	1275	87328	0	49758	54084

Table 4 shows the (lack of) effectiveness of random vectors in avoiding detection of the functionally untestable transition faults. For each circuit, the total number of faults is first reported in column 2. Column 3 shows the number of functionally untestable faults identified by our implication engine (TRANIMP), column 4 reports the number of detected functionally untestable faults, and column 5 reports the coverage of the remaining faults. Then we remove those vectors which detect at least one functionally untestable faults from the test set and rerun the fault simulation. The results are reported under the *Pruned RandVec* columns. Obviously, for the pruned random vector set, it will not detect any identified functionally untestable faults, as shown in column 6. Columns 7 and 8 list the number of faults detected and missed by the pruned random vector set,

respectively. It is interesting to see that for circuits s5378, s9234, s13207, all random vectors detect at least one functionally untestable fault! Therefore, if we want to reduce the yield loss by avoiding overtesting of functionally untestable faults for circuits such as these, random vectors will not be very effective.

Table 5 reports the results from our constrained ATPG for Broadside testing, and we compare it with a conventional non-constrained Broadside ATPG engine. We target only the faults that random vectors could not detect without incidentally detecting at least some functionally untestable transition faults. The sizes of these remaining target faults are first listed for each circuit under the second column. Columns 3 to 6 list the number of detected functionally testable faults, number of proved functionally untestable faults, number of aborted faults and test generation time for our constrained Broadside ATPG. The last four columns show the results when non-constrained ATPG is used. Although the execution time for constrained ATPG is longer than the non-constrained version, we significantly improve the quality of generated test vectors because the new test set detects **only** those functionally testable faults and avoid detecting those functionally untestable ones. In other words, the vectors generated by the non-constrained ATPG may include some illegal states and thus detect **both** functionally testable and functionally untestable faults. For example, in circuit s13207, the 27150 faults that the non-constrained ATPG detected included many of the 22547 functionally untestable faults identified by the constrained ATPG. In addition, the constrained ATPG algorithm identified significantly more functionally untestable faults than the non-constrained ATPG. For instance, in circuit s9234, our constrained ATPG identify 8095 functionally untestable faults out of the 29086 remaining potential functionally testable faults, while non-constrained ATPG only identify 3357 functionally untestable faults. Similarly, the number of aborted faults with our proposed method is also fewer. For instance, only 573 transition faults were aborted as opposed to 1396 transition faults in the non-constrained ATPG.

Table 5: Constrained ATPG Vs.Non-constrained ATPG

Ckt	$Target$ $faults$	Constrained ATPG				Non-constrained ATPG			
		DET	UNT	ABT	$Time(s)$	DET	UNT	ABT	$Time(s)$
s344	134	60	74	0	0.19	83	51	0	0.13
s1423	3125	2406	476	243	1621.65	2485	368	272	463.73
s5378	15680	10871	4270	539	5393.13	12999	2024	657	669.35
s9234	29086	20418	8095	573	8114.77	24333	3357	1396	2915.27
s13207	44130	21526	22549	55	19871.32	27150	16900	80	3148.62
s35932	54084	39952	14123	9	34754.80	39978	14089	17	6179.21

6 Conclusions

We presented a novel constrained broadside transition ATPG algorithm. For each untestable transition fault, f, identified, we first compute the set of illegal (unreachable) states that enable detection of f. Then, by formulating the illegal states as a constrained CNF formula in our ATPG process, we efficiently generated a higher quality test set detecting only those functionally testable faults and avoid overtesting of functionally untestable ones. The cost for the CNF formula construction is extremely low, making our formulation very practical. The constrained ATPG allows for earlier backtrack whenever an illegal state is encountered. In some circuits, significantly more functionally untestable transition faults have been identified. At the same time, more faults could be detected without incidental detection of functionally untestable transition faults. With a test set that reduces launching of transitions that are functionally impossible, we believe our method

offers a practical solution to avoid overtesting of these functionally impossible transitions, thus reducing yield loss.

References

[1] R.D. Eldred "Test Routing Based on Symbolic Logical Statement" *Journal ACM,* Vol.6 pp.33-36 Jan. 1959.

[2] J. A. Waicukauski, E. Lindbloom, B. K. Rosen, and V. S. Iyengar. "Transition Fault Simulation" *IEEE Design & Test of Computers,* 4:32-38, April 1987.

[3] G.L. Smith "Model for Delay Faults based upon Paths," *Intl Test Conf.*, pp. 342-349, Sept. 1985.

[4] K. Heragu, J. H. Patel, and V. D. Agrawal, "Segment delay faults: a new fault model," *VLSI Test Symp.,* pp. 32-39, April 1996.

[5] N. Tendulkar, R. Raina, R. Woltenburg, X. Lin, B. Swanson and G. Aldrich, "Novel Techniques for Achieving High At-Speed Transition Fault Coverage for Motorola's Microprocessors Based on PowerPC Instruction Set Architecture," *IEEE VLSI Test Symposium*, 2002, pp. 3-8.

[6] F. F. Hsu, K. M. Butler and J. H. Patel, "A Case Study of the Illinois Scan Architecture," *Intl Test Conf.*, 2001, pp. 538-547.

[7] J. Savir and S. Patil "Scan-Based Transition Test" *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* Vol. 12, No.8 Aug. 1993.

[8] J. Savir and S. Patil "On Broad-Side Delay Test" *VLSI Test Symp.*, pp.284-290 Sept. 1994.

[9] J. Savir "Skewed-Load Transition Test: Part I, Calculus" *Intl Test Conf.*, Oct. 1992, pp. 705-713.

[10] S. Patil, J. Savir "Skewed-Load Transition Test: Part I, Coverage" *Intl Test Conf.*, Oct. 1992, pp. 714-722.

[11] B. Dervisoglu and G. Stong " Design for Testability: Using Scanpath Techniques for Path-Delay Test and Measurement" *Intl Test Conf.*, pp.365-374, 1991.

[12] M. H. Schulz and F. Brglez "Accelerated Transition Fault Simulation," *Design Automation Conf.* pp. 237-243, June 1987.

[13] M. H. Schulz, E. Trischler and T. M. Sarfert, "SOCRATES: A highly efficient automatic test pattern generation system", *IEEE Trans. on Computer-Aided Design*, pp. 126-137, January 1988.

[14] M. H. Schulz and E. Auth, " Improved Deterministic Test Pattern Generation with Applications to Redundancy Identification", *IEEE Trans. on Computer-Aided Design*, pp. 811-816, July 1989.

[15] H. Fujiwara and S. Toida, "On the Acceleration of Test Generation Algorithms", *IEEE Trans. on Computers*, pp. 1137-1144.

[16] P. Goel, " An implicit enumeration algorithm to generate tests for combinational logic circuits", *IEEE Trans. on Computers*, pp. 221-222, March 1981.

[17] I. Hamzaoglu and J. H. Patel, "New Techniques for Deterministic Test Pattern Generation" *IEEE VLSI Test Symposium*, 1998, pp. 446-452.

[18] Jeff Rearick, "Too much Delay Fault Coverage is a Bad Thing," *Intl Test Conf.*, 2001, pp. 624-633.

[19] P. Maxell, I. Hartanto and L. Bentz, "Comparing Functional and Structural Tests" *Intl Test Conf.*, 2000, pp. 400-407.

[20] J. P. Marques-Silva and K. A. Sakallah "GRASP: A search Algorithm for Propositional Satisfiability" *IEEE Trans. on Computers*, May 1999, pp. 506-521.

[21] X. Liu, M. S. Hsiao, S. Chakravarty and P. Thadikaran, "Techniques to Reduce Data Volume and Application Time for Transition Test," *Intl Test Conf.*, 2002, pp. 983-992.

[22] J. Saxena, K. M. Butler, J. Gatt, R. R, S. P. Kumar, S. Basu, D. J. Campbell and J. Berech "Scan-Based Transition Fault Testing- Implementation and Low Cost Test Challenges,"*Intl Test Conf.*, 2002, pp. 1120-1129.

[23] M. S. Hsiao "Maximizing Impossibilities for Untestable Fault Identification" *IEEE Design Automation and Test in Europe Conf.*, 2002, pp. 949-953.