An Efficient Test Data Reduction Technique Through Dynamic Pattern Mixing Across Multiple Fault Models

S. Alampally[1], R. T. Venkatesh[2], P. Shanmugasundaram[2], R. A. Parekhji[1] and V. D. Agrawal[2]
[1]Texas Instruments, Bangalore (India) and [2]Auburn University (USA).

Abstract: *ATPG tool generated patterns are a major component of test data for large SOCs. With increasing sizes of chips, higher integration involving IP cores and the need for patterns targeting multiple fault models for better defect coverage in newer technologies, the issues of adequate coverage and reasonable test data volume and application time dominate the economics of test. We address the problem of generating compact set of test patterns across multiple fault models. Traditional approaches use separate ATPG for each fault models and minimize patterns either during pattern generation through static or dynamic compaction, or after pattern generation by simulating all patterns over all fault models for static compaction. We propose a novel ATPG technique where all fault models of interest are concurrently targeted in a single ATPG run. Patterns are generated in small intervals, each consisting of 16, 32 or 64 patterns. In each interval fault model specific ATPG setups generate separate pattern sets for their respective fault model. An effectiveness criterion then selects exactly one of those pattern sets. The selected set covers untargeted faults that would have required the most additional patterns. Pattern generation intervals are repeated until required coverage for faults of all models of interest is achieved. The sum total of all selected interval pattern sets is the overall test set for the DUT. Experiments on industrial circuits show pattern count reductions of 21% to 68%. The technique is independent of any special ATPG tool or scan compression technique and requires no change or additional support in an existing ATPG system.*

Keywords: ATPG optimizations, pattern merge, test data volume reduction, composite fault models.

1. INTRODUCTION

Manufacturing test contributes a significant portion to the overall cost of an IC and the tester time required for test application is a critical entity for cost minimization. Due to the larger designs and process variations seen in the advanced technology nodes, the number of fault models to be considered for minimizing test escapes has gone up. Consequently there is an increase in test data volume (TDV) and test application time (TAT). An account of the costs of testing on an ATE is given in [1] and a cost model proposed in [2] gives a good explanation of the cost metrics. TDV and TAT are popular test cost metrics and are proportional to the number of patterns required to get the desired coverage, the maximum scan chain length and the number of inputs and outputs. An insight on TDV for SOCs is provided in [3].

Test compression techniques listed by [4] have been used to contain TDV and TAT by reducing the scan chain lengths and increasing the number of chains by providing extra hardware for test data distribution among these chains. They basically exploit the fact that only 2-5% of the bits in the pattern are 'care bits' and the rest can be compressed [4]. The work in [5] explores the solution space for maximizing compression on some large industrial designs. However, there are limits as explained in [6]. Procedures such as test point insertion, mixing combinational and sequential compression approaches and more complex test compression logic can potentially improve the compression, but at the expense of additional hardware complexity. In order to achieve further reduction in test data volume, we have to look beyond the on-chip compression techniques. Reducing the number of patterns through 'pattern reuse' can be a good solution.

In this work, we target test pattern reduction with the help of a simple ATPG flow improvement across multiple fault models. Effective and efficient pattern sets are chosen based on the proposed metric. We intend to exploit both 'care' and 'don't care' bits in a pattern to efficiently combine and compact the pattern sets of different models into one single set. The technique has been evaluated across complex models such as path delay, dynamic bridging and small delay defect driven faults in addition to the stuck-at and transition delay fault models. The results show a comparison with existing pattern optimization approaches. The rest of the paper is organized into six sections. Prior work in the area of multiple fault model pattern optimizations is reviewed in Section 2. The motivation for the work in this paper is explained in Section 3. A detailed description of the methodology is provided in Section 4. Sections 5 and 6 provide the results on large SOC designs and observations are made based on these results. Section 7 concludes the paper.
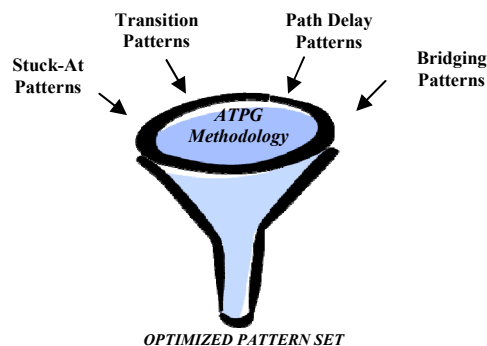
2. TEST PATTERN REUSABILITY

Figure 1: Pattern optimization across multiple fault models.

Test pattern reuse is a general term associated with the ability of patterns of one fault model to detect faults belonging to another fault model, thereby enabling reusability. This characteristic can be exploited to reduce the final pattern count when multiple fault models are considered. As fault models under consideration increase in number there is more room for pattern reuse as pattern-sets associated with different models have varied care-bit densities. Figure 1 illustrates a general pattern optimization approach across fault models by exploiting pattern reusability. One fine example is that of transition fault patterns for the 'Launch off Shift' (LOS) scan approach (as against 'Launch off Capture' (LOC)) and stuck-at patterns. Experimental results in [7] show the effectiveness of stuck-at ATPG patterns for transition fault detection.

A pattern selection approach for the combination of stuck-at, transition and I_{DDQ} models is discussed in [8]. Hybrid LP-ILP technique is used to optimize patterns after their generation. The solution to the LP formulation gives a pattern set that ensures the desired coverage across all the models under consideration. Though this technique optimizes the pattern count, the exponential complexity of linear programming is not suitable for large designs. Another technique proposed in [9] considers path delay, transition and stuck-at models on large SoC designs. The algorithm starts with ATPG on the path delay faults and later uses the generated patterns to detect transition faults. The remaining transition faults are targeted through ATPG and the process is continued till stuck-at faults are also targeted. Though both the works provide good pattern optimization, the optimization is limited due to the non-dynamic nature of fault model selection. The methodology proposed in this paper makes this selection variable during the ATPG process and enables further pattern count reduction for a reasonable run time overhead.

3. MOTIVATION

In a deterministic ATPG process, patterns are generated to target a set of faults that are specific to a fault model. At the end of the process, we get pattern sets and the detected faults for each fault model. It has also been found that reuse of test patterns is possible amongst the fault models. The proposed concurrent ATPG algorithm exploits these facts to minimize the final test set:

- Since patterns have a large number of unspecified or 'don't care' bits in them, opportunities exist for augmenting the random pattern detectability across fault models.
- Transition test patterns have been found to provide good coverage for stuck-at faults. The converse is true as well. The usage of the stuck-at patterns for detecting delay faults depends on whether a 'Launch off Shift' (LOS) or a 'Launch off Capture' (LOC) scheme is being used [10]. In the former case, this is straightforward due to similar clocking used. In a concurrent ATPG flow, patterns can be generated for all other models and then the resulting optimized pattern set can be used to fault simulate on the stuck-at model. The remaining stuck-at faults can be detected with patterns.

- Since path-delay patterns target cumulative delay faults on an entire path, transition faults on nodes on the path under test are detected by the same pattern. If the number of detectable faults with the path-delay fault pattern set is high, the transition fault coverage with those patterns will also be proportionately high. This can also significantly reduce the transition fault pattern-set. On the other hand, transition patterns typically do not give a good coverage on path delay faults since they typically propagate the transition through the shortest (easiest) paths. If small delay defect (SDD) model is considered instead of transition fault model, the probability of getting path delay coverage improves since the timing slack information forces SDD ATPG to propagate transition through longer paths.

- Dynamic bridging fault model requires a transition on the victim node. Hence there exists a high probability of detecting transition faults while targeting the dynamic bridging fault model.

These observations indicate how we can reuse patterns targeted for one fault model for detecting faults in another fault model. Significant pattern count reduction can be achieved if ATPG is performed simultaneously on all fault models. Since current ATPG tools target only one fault model at a time, we create a flow to support such a *concurrent ATPG* process in our work. The ATPG run is split into small intervals. In each interval, patterns are generated targeting all fault models under consideration separately. And the pattern set for each fault model is cross fault simulated on the other fault models. The pattern-set that has highest effectiveness across all fault models is selected for that interval. This process is repeated till final coverage levels are achieved for all fault models. In the next section, we will describe the effectiveness algorithm in detail. The size of the final pattern-set P_{total} can be denoted as:

$$P_{total} \leq \sum_i P_i \text{ , where } i \in F$$

where F is the set of fault models considered and P is the set of patterns for each fault model.

4. ALGORITHM FOR CONCURRENT ATPG

The proposed flow is shown in Figure 2. The following abbreviations have been used for convenience:

N:– Number of fault models.

FC_cum$_n$:– Cumulative fault coverage for model 'n'.

FC_curr$_n$:– Fault coverage of model 'n' in the current interval.

FC_curr$_{n/m}$:– Fault coverage of model 'm' after fault simulation with model 'n' patterns.

P$_n$:– Pattern set of model 'n'.

F$_n$:– Fault set for model 'n' at the start of an interval.

F$_n^*$:– Fault set for model 'n' after ATPG.

F$_{n/m}$:– Fault set after fault simulation of model 'n' patterns on model 'm'.

I:– Number of patterns in an interval.

FFC$_n$:– Final fault coverage for model 'n'.

SP$_{n/m}$:– Saved patterns for fault model 'm' after fault simulation of model 'n' patterns on model 'm'.

SP$_n$:– Total saved patterns across all models if model 'n' patterns are chosen in an interval.

Information for concurrent ATPG like the 'list of fault models to be considered for optimization', 'number of patterns per interval' and 'fault coverage limit' for each model is provided to start with. With this initial data for all the fault models in hand, we begin an interval by generating the specified number of patterns (size of interval) for each fault model and then use these pattern sets to fault simulate on each of the other fault models under consideration. The effectiveness of each of the pattern sets is then evaluated using a metric. Pattern sets chosen to be optimal (using this metric) at the end of an interval are saved along with corresponding detected fault sets (for all the models). Undetected faults for all fault models are targeted at the start of the next interval. This process of pattern generation is continued till the desired coverage for all the fault models is obtained.

The metric used for determining the most effective pattern set at the end of an interval is related to the amount of pattern savings that is realized across all the fault models. The number of saved patterns denoted by SP$_n$ for a fault model 'n' is defined as the total number of the patterns saved across all the other fault models when patterns of fault model 'n' are chosen at the end of an interval. In other words, we need not generate SP$_n$ number of patterns if we use fault model 'n' patterns since they detect faults across other fault models, which otherwise requires SP$_n$ additional patterns with targeted ATPG. Depending upon the speed and accuracy requirements, we can adopt either an approximate or an accurate metric for determining the SP$_n$.

a. Accurate metric

To calculate the accurate metric of savings with patterns P$_n$, fault simulation is performed on the undetected faults of fault model 'm' with P$_n$. ATPG is then run for fault model 'm' on a fault set containing only the faults detected by P$_n$. The number of patterns required to get equivalent detection on model 'm' through ATPG is termed as SP$_{n/m}$. The SP$_{n/m}$ denotes number of patterns saved (need not be generated) by selecting patterns of set P$_n$ for the final pattern set in the interval. This process is repeated on all other fault models with pattern set P$_n$. The sum is denoted as SP$_n$.

b. Approximate metric

To calculate the approximate metric of pattern set P$_n$, fault simulation is performed on fault model 'm' with P$_n$ on undetected faults in the current interval to get FC_curr$_{n/m}$. To get the effectiveness of the set P$_n$, a simple ratio-metric calculation is performed. If I patterns from set P$_m$ are required to get FC_curr$_m$ coverage, the number of model 'm' patterns that would be needed to get FC_curr$_{n/m}$ is calculated as:

$$SP_{n/m} = (FC\_curr_{n/m} / FC\_curr_m) * I$$

This number represents saving achieved by selecting P$_n$ over P$_m$ in the interval. The individual savings on each fault model with pattern set P$_n$ are summed up to get SP$_n$.

This metric is calculated for all fault models and the pattern set having highest SP is selected for the current interval. The approach using the accurate metric consumes more CPU time compared to that using the approximate metric since it involves an extra ATPG step for effectiveness calculation. It was also observed that the basic algorithm (as described in Figure 2) can be speeded up by parallelizing certain independent parts of the flow. Though fault simulation has to always follow pattern generation for each model in the flow without any room for concurrency, individual model runs can be parallelized as all information is available at the start of the interval. This parallelization has also been incorporated into the algorithm to reduce run times.

Table 1: Sample iteration of the flow with approximate metric for an interval limit of 32 patterns.

	Stuck-at (1)	Transition (2)
Step 1	FC_curr$_1$ = 0, F$_1$	FC_curr$_2$ = 0, F$_2$
Step 2	FC_curr$_1$ = 58, P$_1$	FC_curr$_2$ = 48, P$_2$
Step 3	FC_curr$_{2/1}$ = 30	FC_curr$_{1/2}$ = 30
Step 4 (approx)	SP$_{2/1}$ = SP$_2$ = 16.55 (32 * FC_curr$_{2/1}$/ FC_curr$_1$)	SP$_{1/2}$ = SP$_1$ = 20 (32* FC_curr$_{1/2}$/ FC_curr$_2$)
Conclusion: After Step 4, SP$_1$ > SP$_2$, Stuck-at fault pattern set is more effective than transition fault pattern set		

Table 1 provides a snapshot of the pattern generation and fault simulation flow when stuck-at and transition models are used. Step 1 initializes the fault coverage numbers for fault models 1 and 2 to zero. In Step 2, pattern generation is performed on both fault models with pattern count limited to the interval size. In Step 3, fault simulation on the other fault models is performed (e.g. stuck-at fault patterns for current interval are simulated against transition faults and vice-versa). In Step 4, the

metric to evaluate the best pattern set for the current interval is calculated. The metric can either be an approximate one or an accurate one. The pattern set that achieves the highest pattern savings is chosen in that interval.
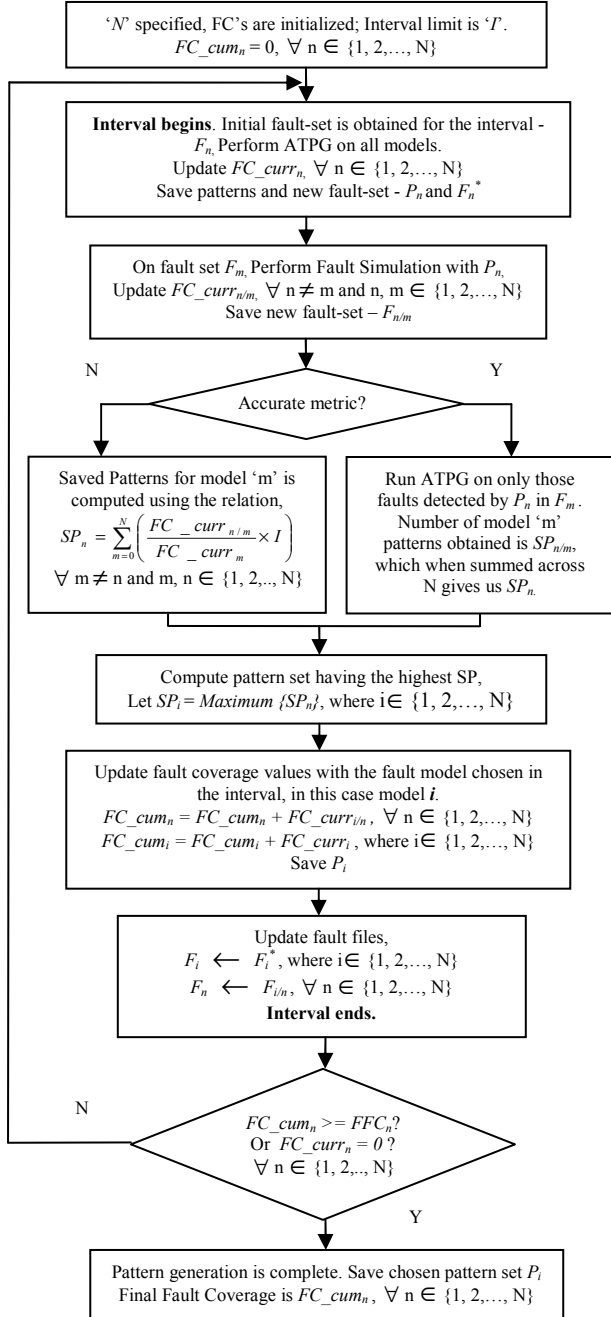
'N' specified, FC's are initialized; Interval limit is 'I'.
$FC\_cum_n = 0, \ \forall \ n \in \{1, 2,..., N\}$

Interval begins. Initial fault-set is obtained for the interval - F_n. Perform ATPG on all models.
Update $FC\_curr_n, \ \forall \ n \in \{1, 2,..., N\}$
Save patterns and new fault-set - P_n and F_n^*

On fault set F_m, Perform Fault Simulation with P_n,
Update $FC\_curr_{n/m}, \ \forall \ n \neq m$ and $n, m \in \{1, 2,..., N\}$
Save new fault-set – $F_{n/m}$

Accurate metric?   N   Y

Saved Patterns for model 'm' is computed using the relation,
$$SP_n = \sum_{m=0}^{N} \left(\frac{FC\_curr_{n/m}}{FC\_curr_m} \times I \right)$$
$\forall \ m \neq n$ and $m, n \in \{1, 2,.., N\}$

Run ATPG on only those faults detected by P_n in F_m. Number of model 'm' patterns obtained is $SP_{n/m}$, which when summed across N gives us SP_n.

Compute pattern set having the highest SP,
Let $SP_i = Maximum \ \{SP_n\}$, where $i \in \{1, 2,..., N\}$

Update fault coverage values with the fault model chosen in the interval, in this case model **i**.
$FC\_cum_n = FC\_cum_n + FC\_curr_{i/n}, \ \forall \ n \in \{1, 2,..., N\}$
$FC\_cum_i = FC\_cum_i + FC\_curr_i$, where $i \in \{1, 2,..., N\}$
Save P_i

Update fault files,
$F_i \ \leftarrow \ F_i^*$, where $i \in \{1, 2,..., N\}$
$F_n \ \leftarrow \ F_{i/n}, \ \forall \ n \in \{1, 2,..., N\}$
Interval ends.

$FC\_cum_n >= FFC_n$?
Or $FC\_curr_n = 0$?
$\forall \ n \in \{1, 2,.., N\}$ N   Y

Pattern generation is complete. Save chosen pattern set P_i
Final Fault Coverage is $FC\_cum_n, \ \forall \ n \in \{1, 2,..., N\}$

Figure 2: Basic concurrent ATPG flow.

5. EXPERIMENTAL RESULTS

Experiments were carried out on some SoCs in Texas Instruments to evaluate the practical benefits explained in the previous sections. The methodology was tried on several different fault model combinations. In addition to uncompressed ATPG, designs with combinational (Synopsys' DFTMAX) [11] and sequential (Synopsys' DBIST) [12] compression techniques were used. LOC and LOS at-speed approaches were used, but since LOS tends to provide more benefits over LOC in terms of pattern count and coverage levels, the former was chosen whenever we had an option to choose between the two approaches. The fault models considered were stuck-at, transition / small delay defect (SDD), bridging and path delay. TetraMAX [13] was used for ATPG and fault simulation. Table 2 lists the size of the designs used in the experiments. Percentage of pattern count reduction was taken as the metric for evaluation of the results. In order to reduce the run times the approximate metric was used.

Table 2: Statistics of designs used in the experiments.

Design	Flip-flop count	Gate count (in millions)	ATPG technique
A	219574	4	LOC
B	240000	2.5	LOS
C	33792	0.4	LOS

The results shown in Table 3 were obtained for Designs A and B by bypassing the compression features available for them. The unoptimized results are nothing but a sum of the pattern counts of each model. The results are compared against both the unoptimized runs and the existing optimization technique that is used by a number of teams at Texas Instruments [9]. Pattern count reductions of up to 45% were obtained when compared to the approach of combining patterns from standalone ATPG runs. A comparison with [9] showed improvements of up to 26% on an average. From the table, it can be observed that optimal benefit is achieved when stuck-at and transition faults are considered. This is mainly due to the fact that the two models have large fault-sets and pattern counts as compared to either bridging or path delay fault models. The scope for reuse increases with increase in number of patterns due to matching care bits and increased probability of random fault detection. Design A could not be simulated for the combination of stuck-at and transition models because it uses the LOC approach for at-speed testing. It can be seen that the model which has the largest standalone pattern count forces a limit on the compaction possible. For Design B, it can also be observed that dynamic pattern mixing results in lesser pattern count than the transition fault model. We attribute this anomaly to the new ATPG run that gets fired for every interval in dynamic pattern mixing mode. ATPG tool starts with a new random seed in each interval, increasing the chances of detecting more faults than with that of using single seed for entire pattern generation.

Table 4 provides the results for all the three designs in scan compression mode. Designs A and C uses a combinational compression technique (DFTMAX) whereas B uses a sequential compression technique (DBIST). Design B was evaluated using pattern intervals as opposed to pattern numbers where each pattern interval was made up of 32 patterns. The overall benefits over [9] were reduced when compared to the same design set in the uncompressed mode, but a look at the table shows a 30% average reduction against the unoptimized set. Stuck-at and transition models again seem to dominate for the same reasons mentioned before.

6. OBSERVATIONS

The designs used for experimentation were varied in size and so were the ATPG techniques used on each of them. This variety has helped to arrive at the following observations:

a) As shown in Table 3 and Table 4, the advantage with concurrent ATPG is less in the presence of scan compression. This can be attributed to: (i) The care bit availability with scan compression is lesser, leaving less scope of pattern-reuse. (ii) The don't care bits are also less random, due to increased correlation. The results with combinational and sequential compression differ since don't care bits in the former are more correlated.

b) Figure 3, 4 and 5 give a good account of the pattern mixing across fault models that occurs during the course of a concurrent ATPG run for one of the combinations with Design A. As observed from the graphs, the fault model chosen by the algorithm varies frequently during the ATPG process indicating that it is beneficial to employ this technique against the existing optimization [9].

c) Path delay faults require relatively higher percentage of specified bits compared to other models. Reuse of path delay patterns is not very effective as the fault simulation coverage with these patterns on other models doesn't yield high benefits. It can be clearly observed from Figure 5 that the path delay model gets de-prioritized due to the combined dominance of the transition and dynamic bridging models and requires its own patterns. Situations like this force an approach that combines the method in [9] with concurrent ATPG. In this case, the path delay patterns can be generated with a standalone run and can then be used for fault simulation with transition and dynamic bridging fault models.

d) Run times were large mainly due to the frequent ATPG and fault simulation processes at each interval. The ATPG run times can be reduced by increasing the interval size. In most fault model combinations with concurrent ATPG, there would be only one model left for coverage improvement towards the end of the process. Once this state is reached the pattern limitation can be taken off to

allow the ATPG to run till the desired coverage is reached for that model. This will reduce run times.
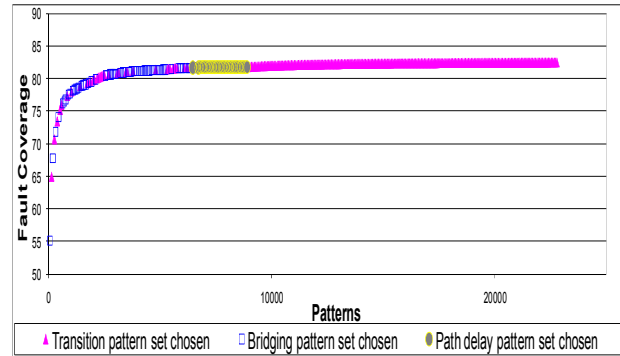

Figure 3: Transition fault coverage when run along with dynamic bridging and path delay fault models.
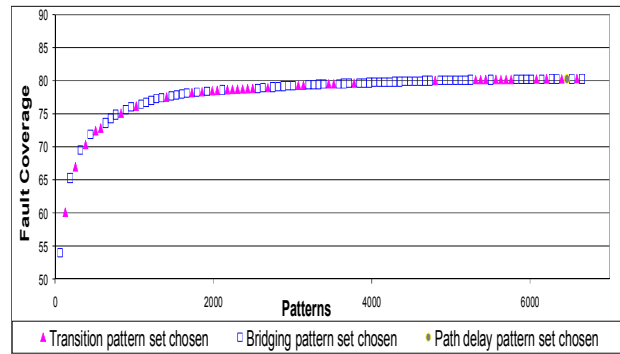

Figure 4: Bridging fault coverage when run along with transition and path delay fault models.
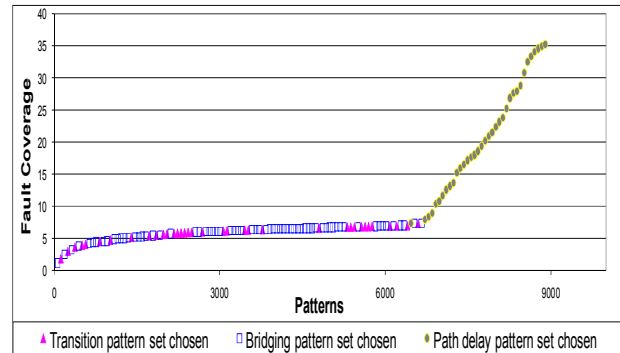

Figure 5: Path delay fault coverage when run along with transition and bridging fault models.

7. CONCLUSION

For large SOCs, structural test patterns obtained using ATPG tools continue to dominate the test time. As the number of fault models being targeted increases, the number of such patterns increases too. This paper presents a methodology for reducing the pattern count across multiple fault models. ATPG is performed in steps and various fault models are concurrently targeted. Pattern sets for a given fault model are incrementally generated

and simulated across the other fault models. Stuck-at, transition, path delay and bridging fault patterns have been considered for experiments, with and without scan compression. This approach has been applied to several industrial designs in Texas Instruments and benefits from 21% to 68% are seen with and without compression, as compared to the conventional approach of just adding the patterns across all fault models. When compared to existing optimization technique, benefits with scan compression are in the range of 3% to 14%, whereas they are 17% to 36% without scan compression. This method does not require any modification to the ATPG flow. Other ideas for further optimizations have also been identified in the paper and will be explored in future work.

References

[1] J. Bedsole, R. Raina, A. Crouch and M. S. Abadir, "Very Low Cost Testers: Opportunities and Challenges," *IEEE Design and Test of Computers*, vol. 18, no. 5, pp. 60-69, 2001.

[2] S. Wei, P. K. Nag, R. D. Blanton, A. Gattiker and W. Maly, "To DFT or not to DFT?" *Proc. Intl. Test Conf.*, pp. 557-566, 1997.

[3] O. Sinanoglu, E. J. Marinissen, A. Sehgal, J. Fitzgerald and J. Rearick, "Test Data Volume Comparison: Monolithic vs. Modular SoC Testing," *IEEE Design & Test of Computers*, vol. 26, no. 3, pp. 25-37, 2009.

[4] N. A. Touba, "Survey of Test Vector Compression Techniques," *IEEE Design & Test of Computers*, vol. 23, no. 4, pp. 294-303, 2006.

[5] S. Alampally, J. Abraham, R. A. Parekhji, R. Kapur and T. W. Williams, "Evaluation of Entropy Driven Compression Bounds on Industrial Designs," *Proc. Asian Test Symp.*, pp. 13-18, 2008.

[6] T. W. Williams, "The Limits of Compression," *Proc. Intl. Test Conf.*, pp. 1-2, 2008.

[7] W. Kawamura and T. Onodera, "Experimental Results of Transition Fault Simulation with DC Scan Tests," *Proc. Asian Test Symp.*, pp. 212, 2007.

[8] N. Yogi and V. D. Agrawal, "N-Model Tests for VLSI Circuits," *Proc. Southeastern Symposium on System Theory*, pp. 242-246, 2008.

[9] S. Goel and R. A. Parekhji, "Choosing the Right Mix of At-Speed Structural Test Patterns: Comparisons in Pattern Volume Reduction and Fault Detection Efficiency," *Proc. Asian Test Symp.*, pp. 330-336, Dec. 2005.

[10] I. Park and E. J. McCluskey, "Launch-on-Shift-Capture Transition Tests," *Proc. Intl. Test Conf.*, pp. 1-9, 2008.

[11] Synopsys, Inc., *DFTMAX Compression User Guide*, Version E-2010.12.

[12] Synopsys, Inc., *SoCBIST Deterministic Logic BIST User Guide*, version 2005.09.

[13] Synopsys, Inc., *TetraMAX User Guide*, Version V-2010.03, 2010.

Table 3: Pattern statistics for all the designs in non compression mode.

Design	Fault model combinations	Test coverage %	Pattern Count			% Reduction w.r.t	
			Unoptimized	Optimized using [9]	Concurrent ATPG	Unoptimized	[9]
A	Transition	96.91	14590	14059	22784	20.91	17.24
	Dynamic Bridging	90.79	12592	11666			
	Path delay	37.45	1806	1806			
	Final Pattern Count		28808	27531			
A	Transition	96.7	13919	13482	18752	28.78	27.58
	Dynamic Bridging	90.79	12412	12412			
	Final Pattern Count		26331	25894			
A	Small Delay	96.03	12896	2784	8768	67.69	46.06
	Dynamic Bridging	90.79	12412	11666			
	Path delay	37.45	1806	1806			
	Final Pattern Count		27144	16256			
B	Stuck-at	96.41	1535	1535	4448	45.12	36.23
	Transition	91.97	6570	5441			
	Final Pattern Count		8105	6976			

Table 4: Pattern statistics for all the designs in compression mode.

Design	Fault model combinations	Test coverage %	Pattern Count (DFTMAX) / Intervals (DBIST)			% Reduction w.r.t	
			Unoptimized	Optimized using [9]	Concurrent ATPG	Unoptimized	[9]
A (DFTMAX)	Transition	96.91	25056	14048	33250	26.72	3.25
	Dynamic Bridging	89.56	20320	20320			
	Final Pattern Count		45376	34368			
B (DBIST)	Stuck-At	96.49	326	35	1084	31.21	13.9
	Transition	91.60	1214	1214			
	Static Bridging	70.36	17	3			
	Dynamic Bridging	61.84	19	7			
	Final Pattern Count		1576	1259			
C (DFTMAX)	Stuck-At	99.03	4706	498	11392	30.91	7.24
	Transition	95.29	11784	11784			
	Final Pattern Count		16490	12282			