High Quality ATPG for Delay Defects*

Puneet Gupta and Michael S. Hsiao
The Bradley Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg, 24060, VA

Abstract: ***The paper presents a novel technique for generating effective vectors for delay defects. The test set achieves high path delay fault coverage to capture small-distributed delay defects and high transition fault coverage to capture gross delay defects. Furthermore, non-robust paths for ATPG are filtered (selected) carefully so that there is a minimum overlap with the already tested robust paths. A relationship between path delay fault model and transition fault model has been observed which helps us reduce the number of non-robust paths considered for test generation. To generate tests for robust and non-robust paths, a deterministic ATPG engine is developed. Clustering of paths has been done in order to improve the test set quality. Implications were used to identify the untestable paths. Finally an incremental propagation based ATPG is used for transition faults. Results for ISCAS'85 and full-scan ISCAS'89 benchmark circuits show that the filtered non-robust path set can be reduced to 40% smaller than the conventional path set without losing delay defect coverage. Clustering reduces vector size in average by about 40%.***

1. INTRODUCTION

Increasing performance requirements motivated testing for the correct temporal behavior, commonly known as delay testing [1]. Delay defects can be modeled in a number of different ways, among which the most common are the path delay fault (PDF) model [2] and the transition fault model [3,4]. Test patterns for transition faults and PDFs consist of a pair of vectors $\{V_1, V_2\}$ where V_1 is required to initialize the target node and V_2 is required to launch the appropriate transition at the target node and propagate it to an observation point, such as a primary output (PO). In PDF, cumulative effect of gate delays along the path is considered whereas in the transition fault, every transition (both $1\rightarrow 0$ and $0\rightarrow 1$) models excessive delay on a single node in the circuit. In addition, a transition fault can be modeled as 2 stuck-at faults, making test generation simpler.

The test that delivers a rising (falling) transition to a node and sensitizes a path from that node to an observation point will detect a slow-to-rise (slow-to-fall) transition fault at that node. That same test may detect a path delay fault associated with the particular route into and out of the node in question. Conversely, a PDF test may also detect some transition faults. Nevertheless, a complete transition test set may not detect all critical paths; likewise, a test set that exercises longest paths may not detect all transition faults. Because the transition fault model is for capturing gross defects, while the PDF model is for detecting small defects, in order to achieve high delay defect coverage we require both high path and transition fault coverage.

Since there can potentially be a large number of paths in a circuit, we need to have ways to classify them and reduce the effective number of paths to be tested. PDFs can be broadly classified in two ways [5]: robust PDF and non-robust (NR) PDF. A nine-value-based ATPG for both robust and NR tests is presented in [6]. DYNAMITE [7] and RESIST [8] give a more in depth analysis of PDFs and uses deterministic approaches for ATPG. FSIMGEO [9] is a simulation based ATPG engine for PDFs, but this misses the delay faults on the less critical paths. To overcome this, segment delay faults were considered and studied in [10]. In this the authors study the technique of covering delay defects on untestable critical paths by robustly testing their longest possible segments that are not covered by any of the testable critical path. The disadvantage of this scheme is that there are a large number of untestable critical paths and generating NR tests for all can be futile. A different approach for the selection of critical paths has been presented in [11]. Here the authors try to generate a longest path passing through each gate. But since the longest path passing through a gate may actually be one of the shortest paths in the whole circuit, this technique does not guarantee a proper coverage of the critical paths if only these paths are considered. A statistical based approach is presented in [12-13] where critical paths are selected based on the statistical properties of the already detected paths.

Obtaining high PDF coverage may require testing of a large number of paths, many of which overlap with one another. To improve the delay test sets, in this paper we make an attempt to generate tests such that they not only have high robust path coverage for the critical paths, but the test set is also capable of detecting other delay fault models (such as transition faults) that were missed by the critical path analysis.

Since the number of paths can be very large for practical circuits, we try to generate tests for a filtered path set. The idea behind this filtered set is to reduce the number of NR paths to be considered for test generation without losing PDF coverage. Hence, instead of selecting NR paths based on their lengths, we discard all NR paths that significantly overlap with previously tested robust paths. This concept can be understood by considering Figure 1. It shows a circuit model with 2 paths originating from 2 PIs and ending at 2 different PO. Let path P_1 (PI1-PO1) be longer than P_2 (PI2-PO2) and let us assume that P_1 is robustly testable whereas P_2 is robustly untestable. The overlap of P_1 and P_2 is L_{over}. We know that since we can test P_1 robustly, the region of overlap is also tested robustly. If L_{over} is greater than some preset threshold, then the delay

* This research is supported in part by NSF under contract 0196470 and a grant from NJ Commission on Science and Technology.

due to the non-overlapping portion of P_2 alone will not likely make P_2 faulty (if the defects on the non-overlapping segment are small distributed delay defects). Hence, the likely fault that can make P_2 faulty is a *large* delay present in the non-overlapping section. By making sure that the test set covers the transition faults associated with these gates, we can discard many paths like P_2, reducing the total number of paths needed to be considered for test generation. Due to this observation, a high-quality delay test set should achieve a high Transition Fault Coverage (TFC) to cover those NR paths that the test strategy did not specifically target.

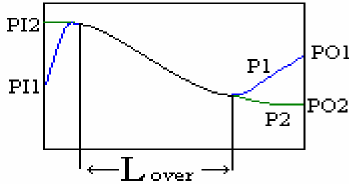

Fig. 1. A circuit model with 2 paths

A general relationship between paths can be deduced from the Venn diagram of Figure 2. Out of the total possible *P* paths in the circuit, region R1 (right rectangle) represents the robust paths in the circuit, while Region R5 (left rectangle) represents the robustly untestable paths. R6 represents the untestable NR paths. R6 is a subset of R5 because untestable NR paths are also robustly untestable. Region R4 represents the conventional *M* longest paths considered for ATPG. Note that this set contains some robust and some non-robust paths. Nevertheless, not all paths from R4 need to be tested, since many of them overlap with already tested robust paths, as explained earlier. Using our proposed filtering technique, we choose paths more intelligently. Let us suppose that the region R2 contains the set of NR paths that do not overlap with the tested robust paths. Then, region R3 (overlapping between R2 and R4) contains paths which are both long and do not overlap with an already tested robust path. Thus, while selecting NR paths for test generation, we want to select

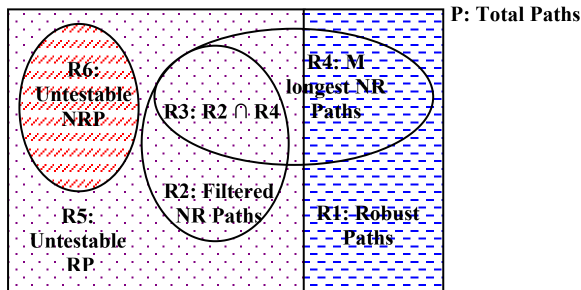

Fig. 2. Relationship between robust and non-robust paths

paths from R3, rather than all of NR paths in R4. If the test set can accommodate more patterns, we can choose additional paths from R2. Results show that there has been a reduction in the NR path set by as much as 40% for some circuits.

In the paper we also propose clustering of paths to reduce the test set size by considering multiple compatible paths together for test generation. Results show that clustering of paths reduces test set data by about 40%. Untestable paths are dropped in the initialization phase and reusable vector storage schemes [14] have been used to further reduce the test set size.

The rest of the paper is divided as follows. Section 2 gives the basic definitions and terminology used in the paper. Section 3 describes the delay testing algorithms along with the observation for NR path dropping. Optimization of vector sets using clustering is also described in Section 3. Section 4 presents the results for combinational and full-scan ISCAS'85 and ISCAS'89 benchmark circuits. Conclusions are given in Section 5.

2. Terminology

A scan based delay test [3, 4, 14], consists of two patterns $<V_1,V_2>$, applied on two successive clock pulses. The first pattern initializes the nodes along the path (and possibly also the off-path inputs) and the second pattern propagates the transition along that path to a PO. Since the patterns must be applied at the rated speed, at-speed testing is needed. For full scan circuits, both the vectors in the scan flip-flops must be ready for consecutive time frames to ensure at-speed testing.

A physical path P is an interconnection of gates from PI to PO. A rising (falling) path P^r (P^f) is defined as the path corresponding to a rising (falling) transition starting at the PI. The polarity of the transition for each gate on the path depends on the inversion parity along that path. *Path Length* is defined, as the number of gates in a given path P. Segment S is a contiguous section of a path P. A segment can start and end at any point in the given path P.

2.1. Delay Fault Models

Given a combinational or a full scan circuit C, a delay defect may manifest as a lumped delay defect on a gate/signal or small-distributed delay over C due to process variations. The *transition fault model* is considered as a logical model and is a good candidate for modeling lumped delay defect. It considers a rising or a falling transition at the inputs and the outputs of logic gates. The vector pair $<V_1,V_2>$ detects a transition fault, if it launches the transition at the fault site and V_2 detects the corresponding stuck-at fault.

A vector pair $<V_1,V_2>$ is said to be a *non-robust path delay* test for a path P, if it launches the transition at the beginning of P, and all off path inputs of P under V_2 have a non-controlling value (NCV).

A vector pair $<V_1,V_2>$ is said to be a *robust path delay* test for a path P if (a) it is a NR test for P and (b) whenever the on-path input of a gate G on P takes a NCV under V_1, then all the side inputs of G should take NCV under V_1 as well.

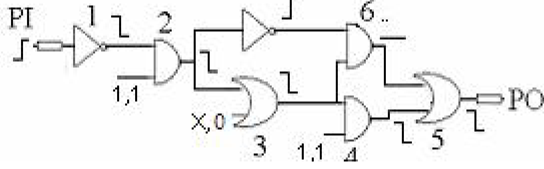

Fig. 3. A sample circuit

Consider the circuit in Figure 3. The rising path P1= PI-1-2-3-4-5-PO has a path length $L_{P1} = 7$. P1 is robustly testable as shown in the figure. Now consider another path P2 = PI-1-2-3-6-5-PO which also has $L_{P2} = 7$. To robustly test P2, the off-path input of gate 6 must be a steady one, which is not possible in this case. Hence P2 is robustly untestable.

2.2. Delay Test Set Size

As mentioned earlier, a scan based delay test needs two vectors for testing a given transition. Hence, any given test set V having *n* test patterns can be represented as:
$V=\{(v_{11},v_{12}),(v_{21},v_{22}),\ldots\ldots(v_{i1},v_{i2}),\ldots\ldots(v_{n1},v_{n2})\}$.

Vector Reusable Test Set (VRTS)[14] is a special form of vector storage in which a test set T having *m* elements can be represented as:
$T=\{(v_{11},v_{12}),(v_{12},v_{22}),\ldots\ldots(v_{(i-1)2},v_{i2}),\quad\ldots\ldots\quad(v_{(m-1)2},0v_{m2})\}$ *where* $m<n$. Thus, instead of storing *2n* vectors for test set V, we only need to store *m* vectors. Since this method reuses the vector space, it is called VRTS.

3. Delay Test Method

Since a circuit can potentially have a large number of paths, we want to select a small number of paths for ATPG and still want to have a high quality test set. A robustly tested path detects small-distributed delays along the path, and hence NR path that overlap significantly with this path may become futile to test. By considering the transition fault model along with the PDF model, we can compute a measure for selecting NR paths for ATPG. The two fault models can be related to each other by the following observation.

Observation 1: By making sure that a test set has high transition fault coverage, many of the NR paths that overlap largely with already tested robust paths need not be tested.

We will explain the observation with the following example: Using Figure 1, consider the case when a large portion of a NR path (e.g. P2) overlaps significantly with an already robustly tested path (e.g. P1). One of the following two scenarios can occur:

1) The non-overlapping portion of P2 has a small delay (according to distributed delay model): Since $L_{P1} \geq L_{P2}$, and P1 is already tested robustly, this small delay alone is not likely to make P2 faulty. Hence we don't need to consider P2 separately for ATPG, similar to those less critical or shorter paths that we do not consider (region P-R1-R2-R4-R6 of Figure 2).

2) The non-overlapping portion of P2 has a large (lumped) delay: This large delay can always be tested by using the transition fault model at the nodes of the non-overlapping portion of P2. □

Thus, in order to have high quality delay test, we need to have high robust path coverage, high NR path coverage and high transition fault coverage. To achieve this efficiently, we have designed a 3-phase ATPG. All 3 phases are described in the following sub-sections.

3.1. Robust Test Generation Phase

The first step towards the test generation for robust paths is to enumerate the robustly testable paths for which tests are to be generated. We use an implication-based technique similar to [15] for the removal of all the untestable robust paths. This implication-based technique can be best understood by a simple example. Consider the circuit of Figure 3. To robustly test the path P^r = PI-1-2-3-6-5-PO, there is a transition from a non-controlling (NCV) to a controlling value (CV) at the input of gate 6. Hence, the off-path inputs of gate 6 should have a steady NCV for both V_1 and V_2, which imply a constant '0' at the output of gate 2. This is a conflict and hence P^r is robustly untestable. This implication-based analysis identifies a large number of untestable robust paths for most of the circuits.

After removing paths that are robustly untestable, we want to generate tests for the N longest paths. The algorithm used for doing this is as follows:

robust_ATPG(){
For all paths P not detected {
essential_values(P,val0,val1);
Generate vector V_i *(values in val0 need to be satisfied) //only need to do logic simulation*
If V_i *generated {*
Generate vector V_{i+1} *(values in val1 needs to be satisfied) //only need to do logic simulation*
If V_{i+1} *generated {*
Add V_i *and* V_{i+1} *to the test set T*
Drop all path detected by vector pairs $<V_{i-1},V_i>$ *and* $<V_i,V_{i+1}>$ *}*
}}

The function *essential_values()* analyzes a given path P and finds the values needed by V_1 and V_2 on all the gates of P and stores them in vectors val0 and val1, respectively. It also finds the essential off-path values under V_2. For example, in Figure 3 for path P^r = PI-1-2-3-4-5-PO values in val0 = gate1=1, gate2=1, gate3=1, gate4=1, gate5=1, and val1 = gate1=0, gate2=0, gate3=0, gate4=0, gate5=0. val1 also contains nodes corresponding to *side input* of gate2 to be logic 1 and the *side input* of gate4 to be logic 1 as well. Since the function needs to satisfy the values in val0 and val1, only 3-valued logic simulation is needed. A vector pair is produced for P if V_i and V_{i+1} satisfy all the values in val0 and val1 corresponding to P respectively. All other paths detected by $<V_{i-1},V_i>$ and $<V_i,V_{i+1}>$ are then dropped. The final test set produced after considering all N paths is called T_R.

3.2. NON-ROBUST TEST GENERATION PHASE

An implication-based approach similar to that used to enumerate robust paths is used to first drop all the paths that cannot be tested non-robustly. However, this implication based technique poses restrictions on the values required by V_2 only. After dropping the identified untestable NR paths, we further remove additional NR paths that satisfy the following two conditions according to observation 1:

1) The NR path P_{NR} overlaps with an already detected robust path P_R with an amount greater than a preset threshold Δ_{NR}. The overlapping section should be contiguous.
2) $L(P_R) \geq L(P_{NR})$.

After dropping paths based on above criteria, we can drop a large number of NR paths. But the number of paths dropped depends on the number of robust paths detected. Higher robust path coverage generally translates to more NR paths dropped. We also drop additional paths that are incidentally detected by the robust test set T_R generated in Section 3.1.

Once we have the set of filtered NR paths, we generate test for the longest M paths (if the number of paths is still large). The algorithm for NR path ATPG is similar to that of robust path ATPG used in Section 3.1, except that NR condition is enforced. Hence for V_1, the ATPG needs to satisfy only the conditions at the PI. The final test set produced after the end of this function is called T_{R+NR}.

3.3. TRANSITION FAULT ATPG

The test set produced so far may not have high transition fault coverage (TFC) since we did not target some NR paths (by observation 1) that overlap with an already detected robust path. The dropped NR paths can still cause a delay fault if a large delay defect is present on the nodes of the path that can be captured by using the transition fault model.

Genetic Algorithm (GA) is used for the transition fault ATPG. The advantages of GA over conventional deterministic approach are: (1) multiple transition faults can be easily targeted simultaneously, and (2) without backtracking, vectors can be produced in a reasonably shorter time. GA has been used before for stuck-at faults [16-18]. Calculation of fitness function through multiple fault simulation is a bottleneck in the efficiency of GA's. We developed an ATPG called *Incremental Propagation Based ATPG,* which circumvents the problem of fault simulation required for the calculation of fitness function. The algorithm is divided into three phases. Instead of generating tests that will guarantee the detection of some faults, we generate tests incrementally.

All the transition faults detected by the test set T_{R+NR} are dropped initially and the TFC achieved by T_{R+NR} is defined as $TFC_{phase\ 0}$.The three phases are described as follows.

Phase I: In this phase of the ATPG, we generate test patterns that will only launch the targeted transitions. We try to maximize the launch coverage L_1 in this phase and add all the vectors produced to the test set. Hence at the end of the first phase we have a set $T_0=\{v_1, v_2, v_3\ldots\ldots v_N\}$, where vectors v_1 to v_N are stored in the VRTS fashion. Now a Transition Fault Simulation is performed using this vector set and all the detected faults are dropped. Therefore now we have N vectors, which have launch coverage of L_1 and transition fault coverage of $TFC_{phase\ I}$. For most of the circuits L_1 is near to 100%. For every transition fault *f* that is launched, we keep track of the vector number in a *vec_num* database, which launched *f*. This information is later used in phase III. The importance of phase I come from the fact that a large number of transition faults are easy to detect and we want to drop all the easy faults as soon as possible so as to save execution time. Moreover the database *vec_num* produced in this phase helps reduce the time to regenerate V_1 for faults that are hard to detect in the later phases.

Phase II: In the second phase, we generate VRTS such that the first vector excites as many undetected faults F as possible and the corresponding next vector excites as many opposite of F as possible and also propagates them to *k* levels ahead from the fault site; where *k* is the iteration number within phase II. Note here that the second vector need not propagate the fault to a primary output. Hence after the end of first iteration within phase II, we have another test set $T_1=\{v_{N+1}, v_{N+2}, v_{N+3}\ldots v_{N+m}\}$. Transition fault simulation is again performed on T_1 and the detected faults are dropped.

After the end of *k* iterations (*k* in worst case can be equal to maximum number of levels in the circuit) we have *k* test sets. These can be appended together to get the test set $T = \{(T_1, T_2, T_3,\ldots, \ldots, T_k)\}$.Thus the TFC of phase II is :

$$TFC_{phase\ II} = \sum(TFC(T_i)) + \Phi$$

And $\Phi = TFC(V_{T1}^{\alpha 1}, V_{T2}^{1}) + TFC(V_{T2}^{\alpha 2}, V_{T3}^{1}) + \ldots.. + TFC(V_{T(k-1)}^{\alpha(k-1)}, V_{Tk}^{1})$

Where αi is the number of vector in test set T_i and the V_{Ti}^{x} represents vector number X of test set T_i. The term Φ accounts for the TFC for the vectors that are at the boundary of the two VRTS, T_i and T_{i+1}.

Phase III: This phase targets the remaining hard-to-detect transition faults and is a fault dependent phase. Unlike the other two phases it adds a vector pair for each detected fault. In this phase every undetected transition fault is considered separately and a test is generated for it. Fitness of an individual is defined as the number of fault events produced. Once a vector pair $<V_1,V_2>$ is generated for a transition fault *f* , we drop all the other undetected faults that might be detected by $<V_1,V_2>$. From the *vec_num* database generated in phase I, it is easy to find the vectors that launch the transition. We don't have to waste effort in regenerating vector V_1. Hence, if a transition fault F has a database entry in *vec_num,* then we only need to generate V_2. The TFC at the end of this phase is given by $TFC_{phase\ III.}$ Hence, after the end of all three phases the final TFC is:

$TFC = TFC_{phase\ 0} + TFC_{phase\ I} + TFC_{phase\ II} + TFC_{phase\ III}$.
And the final test set is called $T_{R+NR+TF}$.

3.4. Clustering of Paths to Reduce Test Set Size

It follows from Section 3.1 and 3.2 that tests are generated for each path separately and 2 vectors are added for each path detected. Although additional paths detected by an added vector pair are dropped, using an optimization called *clustering,* we can further reduce the vector space. All the paths are clustered based on their compatibility with each other. Then, instead of considering one path at a time, we consider a whole cluster at a time. Two paths are clustered if none of the values in val0 and val1 of both the paths contradict each other. It is to be noted that 2 paths need not overlap each other for being compatible. In order to limit the compatible path space, we only combine a path P_i with P_j such that $j \leq i$, where the initial ordering of paths can be arbitrary. In our case the initial ordering of paths was the same as the order in which paths are generated. Moreover, cluster size was limited to 50 for each path due to memory limitations.

Once clustering is done based on path compatibility, we generate test for a whole group. The algorithm targets the first path in the cluster. Once it is detected, we try to fill the remaining don't care values of the produced vector such that another paths in the cluster also gets detected. It is a form of compaction with the exception that vectors are modified dynamically based on the clustered paths. The concept of clustering can be best understood by the following example.

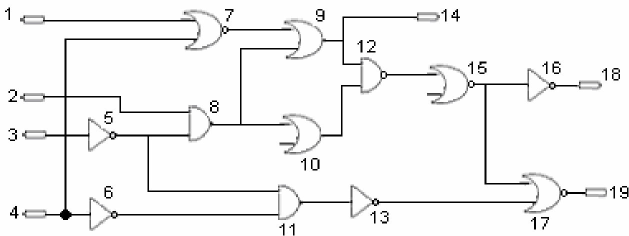

Fig. 4. A Sample Circuit

Consider the circuit of Figure 4. Let paths P_1, P_2, P_3 and P_4 be defined as: P_1^r=1-7-9-14; P_2^f=4-6-11-13-17-19; P_3^f=2-8-9-12-15-16-18; P_4^f=2-8-10-12-15-16-18.
Without clustering, we will require 8 vectors to detect all the 4 paths. But with clustering the compatibility relations (**C**) are as follows:
P_1**C** (P_1, P_3); P_2**C** (P_2, P_3, P_4); P_3**C** (P_1, P_2, P_3); P_4**C** (P_2, P_4).
Hence the cluster of paths will be as follows:
Group 1: P_1, P_3; Group 2: P_2, P_3; Group 3: P_3; Group 4: P_4;
Note that group 2 does not contain P_4 since P_3 is not compatible with P_4. Suppose a test <V_1, V_2> is generated for group 1, which detected both path P_1 and P_3. Hence the final test set will be reduced to only 6 vectors. Thus clustering can help reduce number of vectors.
Since each vector pair using clustering detects more paths, the filtered path set may be further reduced. Consider a vector pair detecting 2 robust paths P_1 and P_2 that share at-least a small common segment and a NR path overlaps them as shown in Figure 5. Let segment S_{L1} of length L_1 be the overlap of NR path with P_1 and segment S_{L2} with length L_2 be the overlap of NR path with P_2. Further assume that L_1 and L_2 are both less than Δ_{NR} but $L_1+L_2 > \Delta_{NR}$. If the segments S_{L1} and S_{L2} are contiguous, we can drop the NR path and further enhance the definition of observation 1 made in Section 3. Thus clustering not only reduced the test set volume, but can also improve the process of filtering NR paths.

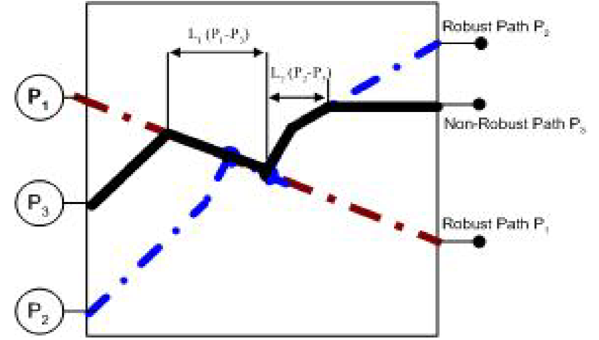

Fig. 5. Clustering helps in NR path filtering

4. Results

This section presents the results for combinational and full scan sequential ISCAS'85 and ISCAS'89 benchmark circuits. The programs were written using C++ and experiments were conducted on a 1.7GHz, Pentium 4, running the Linux operating system. For the calculation of static implications, an implication engine presented in [19] was used. Table 1 presents the analysis on robust and NR paths. The second column shows the total number of paths present in each circuit. The paths include both rising and falling paths. Since the implication engine is not complete, we cannot conclude the detectability about the paths that were not detected as untestable. Column 3 gives the untestable NR paths (P_{UNR}) and column 4 reports the number of untestable robust paths (P_{UR}).

Table 1. Untestable Robust and Non-robust paths

Ckt	# of paths P	Untestable NRP(P_{UNR})	Untestable RP(P_{UR})
C880	17284	163	326
C2670	1359920	1190899	1322192
C5315	2682610	2026131	2205279
S641	3488	1079	1280
S1196	6196	1289	1976
S1238	7118	2725	2793
S1423	89452	41102	52923
S1488	1924	0	0
S5378	27084	3645	3645
S9234	489708	419108	446665
S38584	2161446	1646624	1926898
S35932	394282	334713	355494
S38417	2783158	1469251	1795854

Table 2. ATPG results for robust paths

Circuit	# Paths (N)	Without Clustering			Clustering		
		#Det	#Vec	T_C(s)	#Det	#Vec	T_S(s)
C880	5000	4728*	5756	5.83	4728*	3190	29.27
C1355	5000	337	674	101.1	337	674	785.1
C2670	5000	3742	3744	73.99	3742	3484	242.81
C5315	5000	14*	24	10.85	14*	14	77.37
C7552	5000	34*	68	15.67	34*	68	166.67
S641	2208	2096*	1328	1.09	2096*	818	5.67
S1196	4220	3710*	2404	2.02	3710*	1766	11.26
S1423	5000	4822*	3934	10.64	4822*	3136	57.01
S1238	4325	3665*	2392	2.33	3665*	1832	10.1
S5378	5000	4048*	4186	11.24	4048*	2560	54.44
S9234	5000	3085	4458	337.36	3085	2394	5416.7
S35932	5000	4851*	2512	39.6	4851*	1914	130.44
S38417	5000	3638	5266	20895.23	3638	4536	66685.3

* Rest all of the paths were proven to be untestable by the ATPG.

Table 3a. ATPG results for NR paths with and without filtering of NR paths

Circuit	Overlap=100% (No Filtering)				Overlap=90%			
	#NR paths (P_{NR})	#Det/M_{NR}	# Vec (V_{NR})	T_{NR}(s)	# NR Fil. Paths (P_{NRF})	#Det/M_{NRF}	# Vec (V_{NRF})	T_{NRF}(s)
C880	163	129/129	100	0.85	34	34/34	28	0.28
C1355	5504	2752/4672	1896	23.3	5504	2752/4672	1896	23.3
C2670	131293	44760/63448	3452	12675.1	131293	44760/63448	3452	12675.1
C5315	179148	15101/39447	4500	846.95	179148	15101/39447	4500	846.95
C7552	90442	3983/11030	3644	187.9	90442	3983/11030	3644	187.9
S641	201	11/11	0	0.09	176	6/6	0	0.08
S1196	147	6/12	0	0.14	146	6/12	0	0.14
S1423	11821	939/939	512	9.22	9386	392/392	114	4.06
S9234	27557	590/6736	22	6909.16	27557	590/6736	22	6909.16
S5378	0	0	0	0.0	0	0	0	0.0
S35932	20781	11888/12320	0	43.32	20781	11888/12320	0	43.31

Table 3b. ATPG results for NR paths with and without filtering of NR paths

Circuit	Overlap=80%				Overlap=70%			
	# NR Fil. Paths (P_{NRF})	#Det/M_{NRF}	# Vec V_{NRF}	T_{NRF} (s)	# NR Fil. Paths (P_{NRF})	#Det/M_{NRF}	# Vec (V_{NRF})	T_{NRF}(s)
C880	34	34/34	28	0.28	34	34/34	28	0.28
C1355	5504	2752/4672	1896	23.3	5504	2752/4672	1896	23.3
C2670	131247	44760/63448	3452	12675.1	126740	40792/59468	3426	11635.1
C5315	179140	15095/39439	4500	962.11	178986	15033/39285	4482	839.72
C7552	90442	3983/11030	3644	187.9	90379	3956/10967	3684	185.86
S641	18	1 / 2	0	0.07	0	0	0	0.0
S1196	130	3/9	0	0.14	92	0/1	0	0.14
S1423	8544	286/286	90	3.01	7107	1175/1445	750	22.75
S9234	27501	534/6680	22	6445.80	27286	350/6496	22	6869.85
S5378	0	0	0	0.0	0	0	0	0.0
S35932	17693	8944/9232	0	33.82	14763	6158/6302	0	28.88

Since untestable NR paths $\subseteq$ untestable robust paths, the number of robust paths (N_R) needed to be considered for test generation is P-P_{UR} and number of paths considered for NR path ATPG is P_{UR}–P_{UNR}. The results of Table 1 suggest that there are a large number of paths that cannot be tested robustly or non-robustly.

After filtering out untestable paths, we generate tests for longest *N* robust paths. Table 2 reports the results for

robust ATPG. Column 2 of Table 2 gives N for various circuits. The upper limit on N was chosen to be 5000. Fewer robust paths are chosen if there were not 5000 robust paths in the circuit (e.g. S1196). Next, we report the results of the ATPG without and with clustering, respectively. The effect of clustering can be seen by comparing the number of detected paths and the number of vectors generated. For all cases, the number of vectors generated using clustering is less than the number of vectors generated without clustering, without any loss in the path coverage. This is because a group of paths are considered together for ATPG rather than targeting individual paths. Figure 6 shows the percentage decrease in the number of vectors because of clustering. The average reduction of vector size is about 40%. Execution times for big circuits are about 4-5 times more with clustering but are still under limits. The test set produced after robust path ATPG is called V_{RD}.

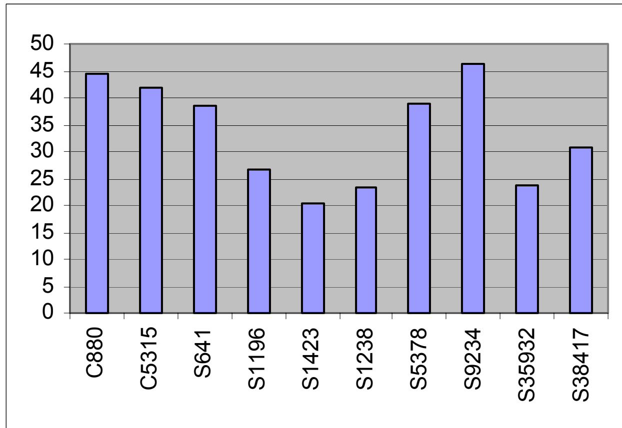

Fig. 6. Percent reduction in # of vectors using clustering

Table 3a and 3b presents the ATPG result for NR paths. First, we filter out the NR paths that overlap with the tested robust paths. Then, we select all paths that are at least 85% longer than the longest path in the filtered set. Δ_{NR} (overlap threshold required to drop the path) was kept to be a path dependent quantity. Clustering was again used for this ATPG. Table 3 (a-b) shows results for the NR paths with and without filtering with varying values of Δ_{NR}. Specifically, we report results for Δ_{NR} of 100%, 90%, 80%, and 70%. The second column under NR path ATPG gives the number of NR paths detected/number of NR paths considered of ATPG (M_{NR}). Note here that Δ_{NR} = 100% means that no filtering has been done.

Paths in the set M_{NR} are covered under region 4 with reference to Figure 2 and also $M_{NR} \subseteq P_{NR}$. In our experiments we choose P_{NR} (# NR Paths) such that $P_{NR} \cap N_R = \Phi$. For most of the circuits the coverage is high at the cost of the addition of few extra vectors (V_{NR}) to the already present vector set V_{RD}.

The column P_{NRF} under 'NR filtered path ATPG' represents the paths after filtering with varying values of Δ_{NR}. As Δ_{NR} decreases from 90% to 70%, the number of paths (M_{NRF}) goes on decreasing for almost all the circuits. This also results in the reduction of the number of vectors. Hence we can infer that filtering at a correct threshold not only decreases the number of vectors but also increases the delay quality of the test set. Using a filtered set of paths enables us to detect a better path set, which is small and hence easy to detect. We can see that for some big circuits the reduction in path size (P_{NR}-P_{NRF}) is about 40%. Set $M_{NRF} \subseteq P_{NRF}$ and is essentially region 3 of Figure 2.

The increase in the number of paths when the threshold is reduced to 70% in circuit S1423 and C7552 can be explained by the definition of M_{NRF}. Since after filtering, the longest path remaining had a small length, set M_{NRF} for Δ_{NR} = 70% is greater than set M_{NRF} for Δ_{NR} = 80%.

The percentage decrease in the number of filtered paths produced with (Δ_{NR}=70%) and without filtering (Δ_{NR}=100%) is plotted in Figure 7 for various circuits. For almost all the circuits there has been a reduction in the number of paths required for testing to achieve high delay coverage. We can also see that the number of vectors required are more with Δ_{NR}=100% (no filtering) than Δ_{NR}=70% in almost all the cases and the execution times are always less with the help of filtering. This proves that the concept of filtering helps us reduce number of vectors with an increase in delay coverage.

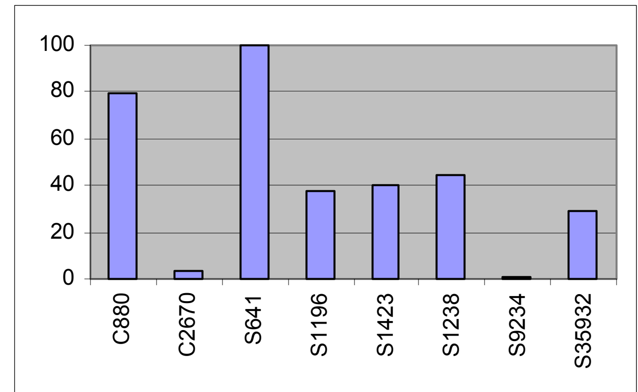

Fig. 7. Percent reduction in the # of paths using filtering

Table 4 presents the results for the transition fault coverage achieved. The second column presents the TFC for the vector set generated so far (V_{RD}+$V_{NRF=70\%}$). We still need to perform transition fault ATPG for some circuits to account for the faults that V_{RD}+$V_{NRF=70\%}$ did not detect. For most of the circuits, the additional number of vectors (V_{TF}) added to the previous test set (V_{RD}+V_{NRF}) are very few since a lot of transition faults are detected while generating tests for the robust paths. For cases such as s35932, we don't need to add any additional vector. The incremental propagation based ATPG produces a high TFC for almost all the circuits in a reasonable amount of time. The last column presents the total number of vectors and total time taken to generate the whole vector set. The time is the sum of T_D + T_{NRF} + T_{TF} and the total vectors produced is the sum of V_{RD}+ V_{NRF} + V_{TF}. These final test sets achieve high robust coverage for the 5000 longest robust paths, high non-robust coverage for the filtered NR paths that do not significantly overlap with tested robust paths, and high transition coverage.

Table 4. TFC and the Total Test Set Size

Circuit	TFC (%) V_{RD}+ V_{NRF}	Transition Fault ATPG			Final Test set	
		TFC (%)	#Vec (V_{TF})	T_{TF}(s)	#Vec ($V^{‡}$)	T (s) †
C880	98.69	100.0	22	4.16	3240	33.71
C1355	97.14	99.76	160	22.88	2730	831.28
C2670	82.9	87.83	296	317.0	7206	12194.9
C5315	99.1	99.54	22	45.79	4518	962.88
C7552	93.08	96.14	1069	1200.0	4821	1552.53
S641	100.0	100.0	0	0.0	818	5.67
S1196	99.84	100.0	9	1.64	1775	13.04
S1238	96.77	97.26	95	15.48	3231	72.49
S1423	98.2	99.2	122	29.1	2704	61.95
S9234	71.14	90.89	2705	3978.8	5122	10085.3
S5378	93.4	98.23	434	196.1	2994	250.54
S35932	90.5	90.5	0	0.0	1914	159.32

$T(s)^{†} = T_{RD} + T_{NRF} + T_{TF}$; $V^{‡} = V_{RD} + V_{NRF} + V_{TF}$.

5. CONCLUSION

A high quality Delay Fault ATPG has been presented. Robust paths, non-robust paths and transition faults were considered for ATPG. Since the number of paths in circuits can be huge, measures are taken to select specific paths for ATPG. Selecting non-robust paths based on their lengths can be non-optimal and hence we drop non-robust paths that significantly overlap with an already-tested robust path, and the results show that the final test set is rich in all three aspect of delay testing. In other words, the obtained test sets capture both gross and distributed delay defects in the circuits. For transition fault ATPG, a special incremental propagation algorithm is proposed to reduce the vector space and generate a high TFC test vector. Clustering of paths has been shown to improve the fault coverage and also reduce the number of vectors by 40%.

REFERENCES:

1. A. K. Majhi and V.D. Agrawal, "Delay Fault Model and Coverage", Proceedings of the VLSI Design Conference, 1998.
2. G. L. Smith "Model for delay faults based upon paths," Int. Test Conf. 1985.
3. J. Savir and S. Patil, "On Broad Side Delay Test", Proc. VLSI Test symposium, 1994.
4. J.Savir and S. Patil, "Scan-Based Transition Test," IEEE Trans. on CAD of Integrated Circuits and Systems, 1993.
5. K. T. Cheng, and H. C. Chen, "Classification and identification of non-robust untestable path delay faults", CAD of Integrated Circuits and Systems, 1996.
6. A. K. Majhi, J. Jacob, L.M. Patnaik and V.D. Agrawal, "An efficient automatic generation system for path delay faults in combinational Circuits", VLSI Design Conference, 1995.
7. K. Fuchs, F. Fink and M. Schulz, "DYNAMITE: An efficient automatic test pattern generation system for path delay faults ", CAD of Integrated Circuits and Systems, 1991.
8. K. Fuchs, M. Pabst and T. Rossel, "RESIST: A recursive test pattern generation algorithm for path delay faults considering various test classes", IEEE transactions on Computer-Aided Design of Integrated Circuits and Systems, 1991.
9. S. Yihe and W. Qifa, "FSIMGEO: A Test Generation Method for Path Delay Fault Test Using Fault Simulation and Genetic Optimization", ASIC/SOC Conference, 2001.
10. M.Sharma and J.H. Patel, "Testing of Critical Paths for Delay Faults", ITC, 2001.
11. M. Sharma, J.H. Patel, "Finding a small set of longest testable paths that cover every gate", Test Conference, 2002. Proceedings. International, 2002.
12. J.J. Liou, L.C. Wang, K.T. Cheng, "On theoretical and practical considerations of path selection for delay fault testing", Computer Aided Design, 2002. ICCAD 2002.
13. J.J. Liou, A. Krstic, L.C. Wang, K.T. Cheng "False-path-aware statistical timing analysis and efficient path selection for delay testing and timing validation", Design Automation Conference, 2002.
14. X. liu, M. S. Hsiao, S. Charravarty, P. J. Thadikaran, "Novel ATPG Algorithms for Transition Faults", Proceedings of the IEEE European Test Workshop, May, 2002.
15. K. Heragu, J.H. Patel, V.D. Agrawal, "Fast Identification of Untestable Delay Faults using Implications", CAD 1997.
16. M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Application of genetically-engineered finite-state-machine sequences to sequential circuit ATPG", IEEE Transactions on CAD of Integrated Circuits and Systems, 1998.
17. E. M. Rudnick, J. H. Patel, G.S. Greenstein and T. M. Niermann, "A Genetic Algorithm Framework for Test Generation," IEEE Transactions on CAD of Integrated Circuits and Systems, 1997.
18. D.G. Saab, Y.G. Saab and J.A. Abraham, "CRIS: A test cultivation program for sequential VLSI circuits", CAD 1992.
19. M. S. Hsiao, "Maximizing impossibilities for Untestable fault identification", DATE, 2002.