Exploring Linear Structures of Critical Path Delay Faults to Reduce Test Efforts *

Shun-Yen Lu
Department of of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan 30013
sylu@larc.ee.nthu.edu.tw

Pei-Ying Hsieh
Department of of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan 30013
pyhsieh@larc.ee.nthu.edu.tw

Jing-Jia Liou
Department of of Electrical Engineering
National Tsing Hua University
Hsinchu, Taiwan 30013
jjliou@ee.nthu.edu.tw

ABSTRACT

It has been shown that the delay of a target path can be composed linearly of other path delays. If the later paths are robustly testable (with known delay values), the target path can then be validated through simple calculation. Yet, no decomposition process is available to find paths that satisfy the above property. In this paper, given a set of target critical paths, we propose a two-stage method to find a set of robust-testable paths (with smaller number than the original set). The first stage constructs a necessary subset for critical robust paths, and the second stage identifies remaining functional sensitizable segments and their corresponding composing robust paths. The experiments show that a large percentage (several benchmarks close to 100%, 75% on average) of critical paths can be covered for most circuits. All paths and coverage are verified to match the best possible results. The data also indicate that the remaining hard-to-test (functional sensitizable) paths actually result from only a few tens of segments in the circuit (except for one circuit, s35932). DfT technique can then be applied to these uncovered segments for full testability with small overheads.

1. INTRODUCTION

Due to increasing process variability, path delay fault testing is widely implemented to ensure circuit performance quality. Yet, several key problems in testing path delay faults are lacked for practical solution: (1)Since no complete path delay fault coverage is achievable (the number of path explodes), critical or longest paths are selected for testing. However, the number of "most critical" paths grow dramatically for most highly-optimized circuits, which renders the selection process meaningless. (2)Even we have well-defined critical paths (paths longer than a specified period), ATPG tools cannot find all appropriate patterns to guarantee test quality, especially for functionally sensitized faults [1]. The test generation for these faults is so complex that the resulting patterns often cannot test paths in full confidence.

[2, 3] observed that the delay of a target path can be expressed as a linear combination of other path delays. Hence it is not necessary to test all paths if a subset of paths can be tested with known path delays. Subsequently, [4] introduced the concept of utilizing a small set of testable paths to bound circuit delays. Though conceptually interesting, the proposed method is not used in full scale. In practice, the test method requires more test time and produces limited knowledge about uncovered critical paths (the original goal is on speed binning). Therefore, it is not straightforward to apply to test chip DPM level.

Based on the above findings, we proposed a method to find *robustly-testable basis paths* for a given set of critical paths. The objective is to decompose each critical path into comprising basis paths. We can ameliorate the problem of test time if we keep a reasonable number of final to-be-tested bases paths, say, hundreds to thousands. More importantly, many functional sensitizable paths will be validated without testing them directly. It is important to minimize the number of targeting functional-sensitizable paths. Very often, the ATPG process for these functional sensitizable paths are very time-consuming or even fail to find average-quality test patterns. The treatment of these hard-to-test faults, modeled as primitive faults [5, 6, 6–8], remains an open problem in the domain of delay testing.

In our proposed method, we utilize a similar *path graph* as in [4] to store path information. Basically, the path graph is a duplicate representation of circuit graph with the following property: a path is in the path graph only and only if the path has known path delay after tests. The path delays (of paths in the graph) are obtained either by direct testing or calculation from other path delays. Therefore, we transform our problem to finding paths for constructing a compact path graph with as many critical paths covered as possible. The flow of the overall procedure can be described in the following 3 major steps:

Step 1 Critical path selection
Arbitrary selection tool can be used to specify the target set of paths for testing, such as the one proposed in [9] if statistical models are applied. The critical paths can be sensitized with different criteria: Robust, Non-Robust, Functional Sensitizable (FS), and untestable. We usually first filter out untestable paths after this step (to avoid further processing).

*Sponsored by National Science Council of Taiwan, ROC #NSC95-2220-E-007-013

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICCAD '06, November 5-9, 2006, San Jose, CA
Copyright 2006 ACM 1-59593-389-1/06/0011 ...$5.00.

Step 2 Construct a path graph with critical robust path

Next, we select a few robust paths from the target set of critical paths, and use these robust paths to construct a base path graph. This step processes only robust faults to improve the overall efficiency, since they are easily testable and do not require the following steps. The complete procedure for Step 2 is described in Section 3.

Step 3 Add non-critical robust paths for uncovered FS paths

For the remaining FS paths, which is critical but not covered by the above path graph, we propose a method (Section 4) to search additional non-critical robust paths to augment the above path graph. For this purpose, we analyze several generic cases of path graph for the general solution.

2. CIRCUIT MODEL

Before describing details of the proposed method, several terminologies are referenced in this section.

2.1 Circuit Graph and Path Graph

We assume that all paths can be traced in a gate-level circuit model. In order to provide the linear structure for path delays, we represent the circuit in a *circuit graph*. The nodes in the circuit graph represent the converging fanin or diverging fanout points. For example, a two-input gate will become a node, since at least two different paths will converge at the gate. And a gate with fanout branch is also a node, since multiple paths will lead to POs from the node. An example circuit and its equivalent graph are given in Fig. 1 and Fig. 2, respectively.

Path graph is a collection of paths in the circuit. The main purpose of a path graph is to represent a set of paths non-enumeratively [10]. It also holds a property that any path in the path graph has known path delay after tests. In other words, if a path can be traced in a path graph, the path is either robust-testable or the path delay is a linear combination of other robust-testable path delays [4]. An example of path graph is shown in Fig. 3, where we have 4 independent paths from the original circuit graph (Fig. 2). Note that rising and falling transitions for paths are considered but shown in the path graph in order to simplify discussion.

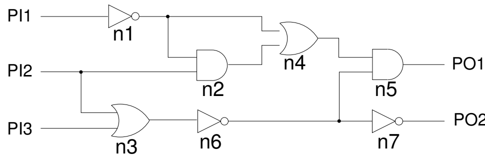

Figure 1: An example circuit

2.2 Simplified Circuit/Path Graph

The circuit graph shown in Fig. 2 implies that each edge represent a pin-pin delay. We can actually simplify the graph significantly with the following transformation rules. The resulting graph contains less nodes and edges, and hence requires less processing in the later procedure. It is important to note that Rule 3 and 4 are vital in the identification of properties of edges, which will be further used in Section 3. Even though all transformed graphs are equivalent, if Rule 3 and 4 are not applied, special treatments of PIs/POs should be considered otherwise. Note that an edge is also called a *segment* in the following discussion, and multiple connected edges constitute a *sub-path*. In order to be compatible with simplified circuit graph, the same segments and nodes should be applied to the path graph (simplified path graph).

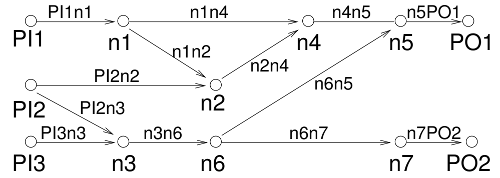

Figure 2: Equivalent circuit graph for the example circuit in Fig. 1

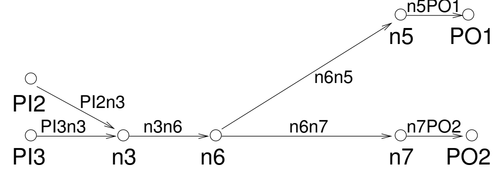

Figure 3: A path graph

Rule 1 Merge a node with single fanout to its successor

As shown in the partial circuit graph (Fig. 4), the node, $n2$, has a single fanout. The delay of the edge, S_1, can be merged into each of S_2, S_3 and S_4. And the resulting graph in the right contains the same information.

Rule 2 Merge a node with single fanin to its predecessor

Same as Rule 1, but applied to a single fanin condition.

Rule 3 Merge all PI nodes

All PI nodes can be identified by their fanout edges, so specifying individual PI nodes is not necessary. For example, in Fig. 2, both fan-outs of $PI2$, $PI2n2$ and $PI2n3$, can be identified unambiguously regardless of label of $PI2$, so we can merge $PI2$ with any other PI nodes.

Rule 4 Merge all PO nodes

Same as Rule 3, but applied to POs.

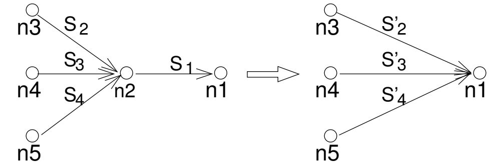

Figure 4: A partial circuit graph

After the transformation of the circuit graph in Fig. 2, we can obtain a simplified version in Fig. 5. Note that no path is missing in the last circuit graph.

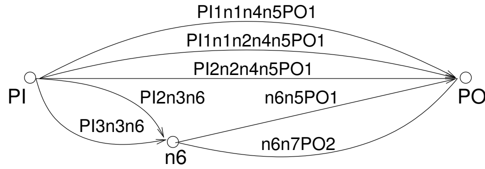

Figure 5: Simplified circuit graph from Fig. 2

3. CONSTRUCT BASE PATH GRAPH

After selecting a set of target critical paths, the next step in our method is to construct a base path graph from a subset of robust paths. It is important that we obtain only the necessary robust paths in this step. Also since our procedure is not influenced by the order of selection, we would be able to handle each robust path one by one.

Fig. 6 is the flow to add one robust path to path graph. We will continue the process until all critical robust paths are covered by the path graph. For a path, each segment on the path will be checked if it's in the path graph or not. We need to handle the following three possible situations:

1. **Covered segments**
   If there is a sub-path of the to-be-added path connecting to PI or PO in the path graph, we will consider any segment on the sub-path as covered. If all segments on the to-be-added path are covered, the path will be processed in Section 3.1. Basically we use the new path to merge nodes in the path graph.

2. **Overlapping segments**
   If a segment is in the path graph, but not a covered segment. The segment is considered as overlapping (not covered) case. We will add the path containing such segments to the path graph and handle the duplication of sub-paths in Section 3.2.

3. **Missing segments**
   If a segment is not found in the path graph (the to-be-added path is a prime with respect to the current path graph), it will also be process in Section 3.2. Though this case will be handled in the same way as overlapping segments, the distinction will be further utilized in Section 4 for adding non-critical paths.

3.1 Covered Segments

If all segments in path P are covered and P is not in path graph, there exists at least one node to split the path P into two sub-paths, and each sub-path is covered but not connected together in the path graph. Otherwise, we will have a covered path (which is already filtered out in the second box of Fig. 6). For example, in Fig. 7, a path $P = (S_1, S_2, S_3)$ is added to path graph. Sub-path $P_1 = (S_1, S_2)$ and $P_2 = (S_3)$ are covered in path graph. Under such condition, we know that node pair $n2'$ and $n2''$ actually split P to P_1 and P_2. ($n2'$ and $n2''$ correspond to the same $n2$ node in the circuit graph, but they are covered by different paths to PI/PO: (S_1, S_2, S_5) and (S_4, S_2, S_3) in our example. We call $n2'$ and $n2''$ node copies of $n2$.) Therefore, after P is added to path graph, according to the following **Merge Property**, $n2'$ and $n2''$ can be merged (Fig. 8). The merging process is important, because more covered critical paths can be found.

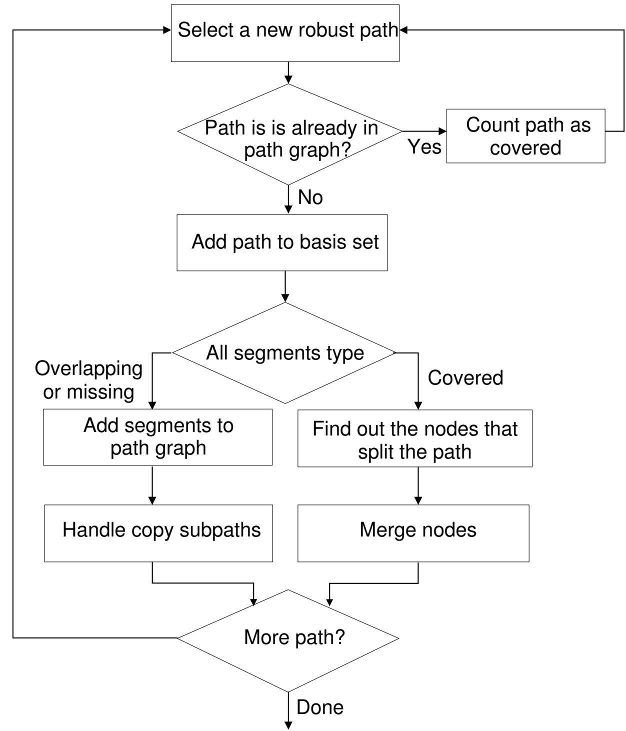

Figure 6: Add path to path graph flowchart

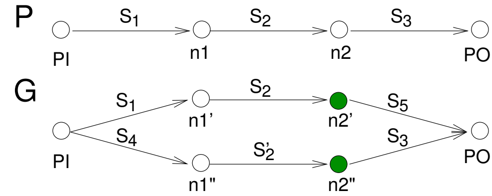

Figure 7: Add a path with covered segments

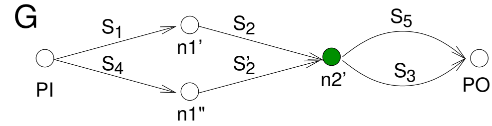

Figure 8: Merge $n2'$ and $n2''$

Otherwise, we will select more robust paths to covered these cases.

Note that there might be other candidate nodes to be merged, such as $n1'$ and $n1''$ in path graph. The merged results are shown in Fig. 9.

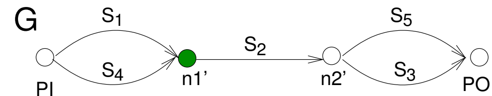

Figure 9: Find out split nodes $n1'$ and $n1''$ and converge again

PROPERTY 1 (**Merge Property**). *If node copies have the same sub-path to PI or PO, these node copies can be*

merged.

PROOF. We will use the path graph in Fig. 10 as an example to proof the above property. Generalization to multiple node copies and multiple segments is trivial. Given the three paths in the path graph, P_1, P_2 and P_3, after tests, we know their respective delays as D_{P_1}, D_{P_2} and D_{P_3}. We know $D_{P_1} = D_{S_1} + D_{S_3}$, $D_{P_2} = D_{S_1} + D_{S_4}$ and $D_{P_3} = D_{S_2} + D_{S_4}$. For the target path $P_4 = (S_2, S_3)$:

$$D_{P_2} = D_{S_2} + D_{S_3} = D_{P_1} + D_{P_3} - D_{P_2}$$

Therefore, P_4 is actually covered by the path graph. We can safely add S_2 from PO to $n1$. Finally, we find that $n1$ and $n1'$ have exactly the same sub-paths to PI, and hence they can be merged (Fig. 11). □

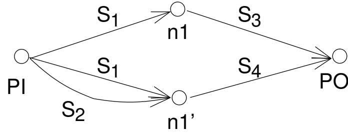

Figure 10: $n1$ and $n1'$ to be merged

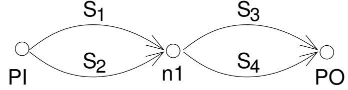

Figure 11: Merge node $n1$ and $n1'$

3.2 Overlapping and Missing Segments

When the robust path has either overlapping or missing segments, these segments will be added to path graph. Usually the addition of new segments involves copying existing nodes (producing node copies). Of course, if there is a covered segment, it will simply be ignored, but the new segments will be appended to these covered segments. (More examples will be provided later).

After the path is added, a special type of linear relations (same sub-paths appearing at different node copies) will emerge in the path graph. They should be handled properly according to **Copy Property** to keep the path graph optimal.

PROPERTY 2 (**Copy**). *If a sub-path appear at different node copies, any other sub-paths share the same end nodes of the sub-path will also appear at all node copies.*

PROOF. A special case in Fig. 12 will be considered. Generalization to other cases is omitted. For the partial path graph in the figure, we can always find two paths containing S_1 and S_2:

$$P_1 = (S_a, S_1, S_b), \text{ and } P_2 = (S_a, S_2, S_b)$$

and

$$D_{P_1} = D_{S_a} + D_{S_1} + D_{S_b} \text{ and } D_{P_2} = D_{S_a} + D_{S_2} + D_{S_3}$$

Then we can get the delay difference of S_1 and S_2 by:

$$D_{P_1} - D_{P_2} = D_{S_1} - D_{S_2}$$

Therefore, after tests, the difference of delays of S_1 and S_2 is obtained. And consider another tested path $P_3 = (S_c, S_1, S_d)$ in the left-bottom path graph (the S_1' between $n1'$ and $n2'$ of Fig. 12). We can also obtain the delay of $P_4 = (S_c, S_2, S_d)$, where $D_{P_4} = D_{P_3} + D_{P_2} - D_{P_1}$. Since the delay of P_4 is known, S_2' can be copied between $n1'$ and $n2'$ (right-hand side of Fig. 12). □

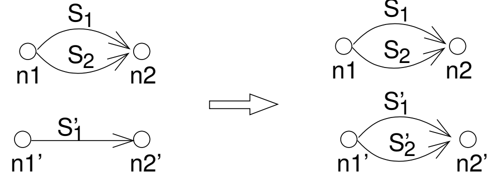

Figure 12: Copy subpath $S2$ to $(n1', n2')$

Note that two possible situations can occur for finding sub-path copies: (1)the newly added sub-path P' copy all shared sub-paths connected to the same end-nodes of the original P. (2)All shared sub-paths connected to the same end-nodes of the newly added sub-path P' are copied to the original P.

In the following, we will use an example to illustrate the addition of overlapping or missing paths, and the handling of sub-path copies. In Fig. 13, $P = (S_1, S_2, S_3, S_4)$ is the path to be added to path graph G. S_1 and S_4 are covered segments, $S2$ is a overlapping segment and S_3 is a missing segment.

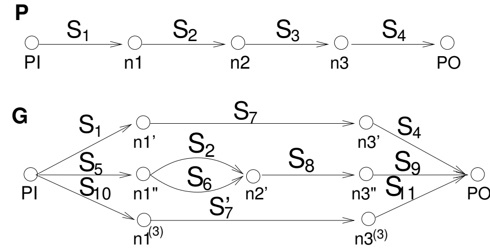

Figure 13: Add path P to path graph G

S_1 and S_4 are ignored and, both S_2 and S_3 are connected to G via S_1 and S_4, as shown in Fig. 14.

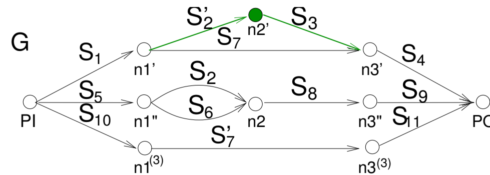

Figure 14: Add sub-path copy S_2 and sub-path S_3 to G

Since S_2 has two copies in G, S_6 should be shared by both copies. Therefore, we copy S_6 to S_2' in Fig. 15.

There is another segment with two copies: S_7 and S_7'. All shared sub-paths are copied (Fig. 16). From this example, we can see that the copying of shared sub-paths requires several level of processing until no segment copies are left out.

4. HANDLE UNCOVERED FS PATH

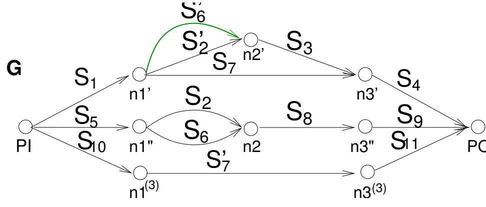

Figure 15: Add sub-path copy S_6' to path graph

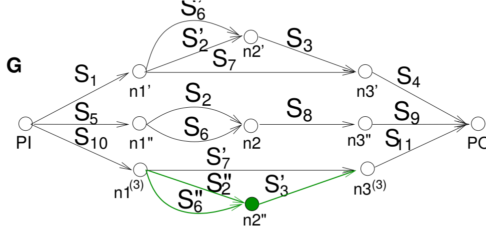

Figure 16: Add sub-path copy S_2'', S_3' and S_6'' to G

After we construct a base path graph from critical robust paths, we can guarantee that all critical robust paths will be covered. Otherwise, the uncovered robust paths will be chosen as another "prime" path for the graph. In this section, we propose a method to handle the remaining critical FS (functional sensitizable) paths. It is also important that the decomposition of these FS paths will require non-critical robust paths, or it should have been covered. Our method consists of two major phases (Fig. 17). The objective of the first phase is to make missing segments of FS paths to become overlapping ones (Section 3) in the path graph. Next, in the second phase, we will try to select robust paths to transform all segments as covered in the path graph. Respective description of each phase is as follows.

4.1 Phase 1: Include Non-existent Segments

In this phase, as long as we find one robust path containing any missing segments, the segments will be included in the path graph. Hence the path containing the segments will be eligible as covered candidates (for next phase). It is found that many FS paths share common missing segments. To avoid processing individual segments multiple times, we first scan remaining paths and collect their corresponding missing segments. Then robust paths will be selected to cover these segments.

Yet another way to improve efficiency is to store functional sensitizable segments that are not contained in any robust paths. If any paths have these FS segments (usually short segments), the paths will never be covered by the path graph. Techniques in [11, 12] can be adapted for FS segments.

The search of robust paths has to observe the following rule: once it is added to the path graph, the less additional (overlapping) segments, the less complexity will be for the next phase. Since in the next phase we need to process all added segments, we would like to keep this number as low as possible. An analogy to solving linear equations: added segments correspond to additional unknown variables. For each additional unknown variable, we need at least include one new equation (robust paths in our discussion).

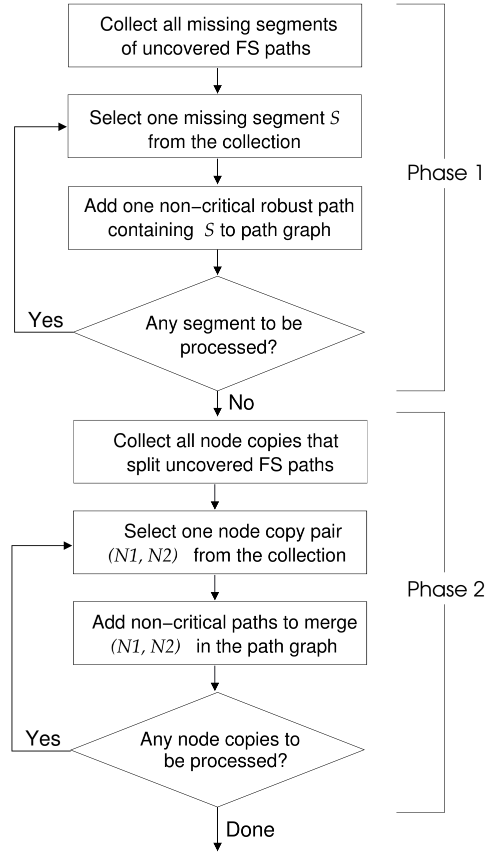

Figure 17: The flow to handle uncovered critical paths

4.2 Phase 2: Cover FS Segments

Since now we only have additional (overlapping) segments to be processed, our objective is to transform these segments into "covered". The key technique is to merge node copies that are induced by separate paths. For example, in Fig. 18, we only have $P_1 = (S_1, S_2)$ and $P_2 = (S_3, S_4)$, so there is a pair of node copies: $n1'$ and $n1''$. As we know that segments in the path graph, (S_1, S_2, S_3, S_4) can be a sub-path containing overlapping segments of a FS critical path. If we are able to merge $n1'$ and $n1''$, these segments will be marked as covered and hence the path containing these segments. In the following, we list all possible ways to merge these two nodes, where the path graph configuration is generic (not just an example) and we can extend it to other cases:

1. One-path selection
   Select P_3 as either $P_3 = (S_1, S_4)$ or $P_3 = (S_3, S_2)$. According to the **Merge Property**, after we add P_3, since from either $n1'$ or $n1''$ we have have the same sub-path (S_4 or S_2) to PO, $n1'$ and $n1''$ are merged.

2. Two-path selection
   Select P_3 and P_4 as either

$$P_3 = (S_1, S_6), P_4 = (S_3, S_6)$$

or

$$P_3 = (S_5, S_2), P_4 = (S_5, S_4)$$

After the addition of these two paths, $n1'$ and $n1''$ are merged. In Fig. 18, PI and PO of path (S_5, S_6) are drawn as separate nodes to denote that it is a new path, but actually there should be just one node for PI and PO.

3. Three-path selection
Select P_3, P_4 and P_5 as either

$$P_1 = (S_1, S_6), P_2 = (S_5, S_4), P_3 = (S_5, S_6)$$

or

$$P_1 = (S_3, S_6), P_2 = (S_5, S_2), P_3 = (S_5, S_6)$$

This is the most complex scenario we need to consider. After adding all the paths, $n1'$ and $n1''$ can be merged

In our findings, the major bottleneck in these path selection processes is the complexity to check robustness of each new path. It is suggested that only partial ATPG is performed initially to choose possible robust candidate paths (only propagate/imply mandatory assignments), and full ATPG can be delayed until it is necessary to confirm the robustness of a path.

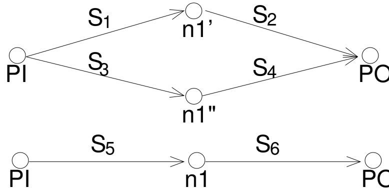

Figure 18: Node pair to be merged

5. EXPERIMENTAL RESULTS

To evaluate our method, experiments have been conducted with combinational parts of ISCAS89 benchmarks (Table 1). A set of critical paths is chosen for each circuit with a specified cut-off period (2nd columns): all paths with delays exceeding the cut-off period are selected (a cell library is used to derive cell delays). For each circuit, two different cut-off periods are chosen to show results based on different critical path sets. The critical paths are sorted into two categories: robust (3rd column) and functional sensitizable (4th column). The total of these two sets of paths is the number of testable paths.

The results after Step 2 (Section 3) are recorded in column 5-7, while the results after Step 3 (Section 4) are in column 8-10. The last column list the CPU time spent on the Step 2 and 3. After each major step, the critical paths are further sorted as Basis (column 5 and 8) or Uncovered FS paths (column 6 and 9). The paths in Basis set will be tested at a ATE for their path delays. The remaining Uncovered FS paths require additional ATPG efforts to generate appropriate test patterns. For column 8 and 9, an related percentage is listed for reference. At column 8 (# New Basis Path), we show the percentage of selected basis path out of total number of critical paths. And the column 9 list the coverage of covered paths, which is defined as the number of covered paths divided by the number of total paths.

Comparing the results of Step 2 and 3, we found that choosing more paths from non-critical robust is essential. Certain circuits can benefit from the selection. For example, in s38584, the FS paths are reduced from 3085 to 2020 after step 2, and further reduce to only 640 at Step 3 with only an increase of 38 paths.

From the table, we found that with a small set of Basis paths (mostly under 10%), a large percentage of FS paths are covered (column 9). Our propose method can effectively reduce both the number of to-be-tested paths (test time) and the number of FS paths for test generation (test development complexity).

The coverage of final FS paths are circuit-dependent. Two exceptions are singled out in the table: s713 and s35932. For s713, we only have around 40% of covered critical paths. Yet, even in this circuit, for all uncovered FS paths, there are actually very few FS segments not covered in our calculation. The number of Uncovered segments after Step 2 and 3 are listed in Column 7 and 10, respectively. Notice that after Step 3, the number is reduced to under a few tens for most circuits. This is significant because we can actually develop DfT (Design-for-Testability) techniques to make these segments robustly-testable. With little extra hardware overheads (the number of uncovered segments), the proposed method in this paper can render the circuits completely path-delay-fault testable. Note that cut-off period is not a important factor in affecting the final results.

There is another anomaly, s35932, in our benchmark. There is no critical robust paths and only 9-15% of critical paths are covered. Even worse, there are 1536 remaining uncovered segments. After manual inspection of the circuit topology, we found that the possible reason for this case is that most FS segments in s35932 are short and connected in series together to form a longer paths. An example of FS segments is shown in Fig.19. There are two short FS segments in the example (marked with bold lines). Further reviews found that similar segments appear repeatively in the circuit. Actually, these FS segments are not difficult cases for ATPGs, since they are short and there is no complex reconverging structures. Further investigation including full primitive fault analysis is required for this circuit.

5.1 Verification of Path Structures

In order to verify that the described method indeed construct a correct path graph, and the delays of covered paths can be calculated by other path delays contained in the path graph, a procedure is proposed below: We denote all segments in circuit graph as S_j, where $j = 1, \dots m$. A path matrix is constructed to represent a path graph. Each row represents a path in the path graph. For example, the ith row is for P_i. The elements at each row denote whether a segment is contained in a path.

$$P_{ij} = \begin{cases} 0, & \text{if } S_j \text{ is not on } P_i \\ 1, & \text{if } S_j \text{ is on } P_i \end{cases}$$

Given a path matrix P (which only contains rows of selected robust paths), we can verify if a FS path is covered by adding an additional row for the FS path. If the rank of new matrix remains unchanged, we know that this path is covered. On the other hand, if we add an uncovered path to the matrix, the rank of the new matrix will increment by one.

With this method, we test all covered and uncovered FS paths (coding with a linear algebra package, Octave [13]). All paths are verified to be correctly categorized.

Circuit name	Cutoff clock	#Robust paths	#FS paths	#Basis paths	#Uncovered FS paths	#Uncovered segments	#New Basis paths	#New uncovered FS paths	#New uncovered segments	CPU Time (second)
s713	200	1352	5944	164	3824	21	178 (0.44%)	3591 (39.59%)	13	0.87
s713	100	1841	6757	258	3871	19	258 (0.60%)	3871 (42.71%)	19	0.71
s1196	50	3336	4986	745	27	6	745 (12.80%)	27 (99.46%)	6	0.09
s1196	20	3659	5309	968	27	6	968 (15.75%)	27 (99.49%)	6	0.10
s5378	130	10143	13790	1156	2473	81	1226 (7.61%)	1936 (85.96%)	20	0.67
s5378	120	11852	15743	1359	2366	51	1400 (7.70%)	1944 (87.65%)	20	0.57
s9234	120	9235	11465	437	1516	80	522 (4.52%)	37 (99.68%)	3	1.13
s9234	115	10373	12832	569	1490	66	643 (4.93%)	45 (99.65%)	3	1.61
s13207	180	7956	19347	182	8159	42	225 (0.96%)	3365 (82.61%)	2	16.62
s13207	175	13476	36081	254	13349	26	280 (0.65%)	6293 (82.56%)	3	15.15
s13207	450	84	2598	34	2514	90	88 (0.45%)	814 (68.67%)	32	826.98
s13207	445	136	3692	60	3556	112	138 (0.49%)	814 (77.95%)	32	794.12
s35932	190	0	7936	0	7936	6496	3153 (8.05%)	7180 (9.53%)	1536	305.55
s35932	185	0	13056	0	13056	7136	3696 (6.66%)	11012 (15.66%)	1536	525.06
s38417	300	0	70	0	70	37	26 (0.55%)	0 (100.00%)	0	2.66
s38417	295	0	154	0	154	58	42 (0.57%)	0 (100.00%)	0	3.40
s38584	400	752	2061	142	1238	25	173 (1.74%)	336 (83.70%)	3	921.43
s38584	398	984	3085	150	2020	29	188 (1.24%)	640 (79.25%)	4	937.12

Table 1: Experimental Data for the proposed method

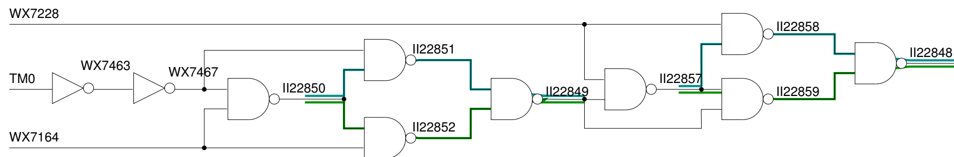

Figure 19: An example of s35932 uncovered FS segment

6. CONCLUSIONS

In this paper, we propose several techniques to analyze the linear structure of critical paths using a path graph. Using these techniques, we select a small set of robust paths to validate a larger set of critical paths. For example, in s9234, only 643 paths are chosen to cover 23106 paths. By testing the selected set of paths, we can calculate delays of other untested critical paths. A large percentage of functional sensitizable paths is thus relieved from difficult test generation efforts. More importantly, we found that only a small set of segments (a few tens at most) are responsible for remaining uncovered FS paths. It is suggested to apply DfT techniques for these segments. With little overheads, critical paths in a circuit can be validated for their path delays completely by our proposed method.

7. REFERENCES

[1] A. Krstic and K.-T. Cheng, *Delay Fault Testing for VLSI Circuits*. Boston, MA: Kluwer Academic Publishers, 1998.

[2] J. D. Lesser and J. J. Shedletsky, "An experimental delay test generator for lsi logic," *IEEE Transactions on Computers*, vol. 29, no. 3, pp. 235–248, Mar. 1980.

[3] W. K. Lam, A. Saldanha, R. K. Brayton, and A. L. Sangiovanni-Vicentelli, "Delay fault coverage, test set size, and performance trade-offs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 1, pp. 32–44, Jan. 1995.

[4] M. Sharma and J. H. Patel, "Bounding circuit delay by testing a very small subset of paths," *Proceedings of IEEE VLSI Test Symposium*, pp. 333–341, Apr. 2000.

[5] A. Krstic, K. T. Cheng, and S. T. Chakradhar, "Identification and Test Generation for Primitive Faults," *Proceedings of IEEE International Test Conference*, pp. 423–432, Oct. 1996.

[6] R. Tekumalla and P. R. Menon, "Test Generation for Primitive Path Delay Faults in Combinational Circuits," *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pp. 636–641, Nov. 1997.

[7] M. Sivaraman and A. J. Strojwas, "Primitive Path Delay Fault Identification," pp. 95–100, Jan. 1997.

[8] ——, *A Unified Approach for Timing Verification and Delay Fault Testing*. Boston, MA: Kluwer Academic Publishers, 1998.

[9] L.-C. Wang, J.-J. Liou, and K.-T. Cheng, "Critical Path Selection for Delay Fault Testing Based Upon a Statistical Timing Model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 11, pp. 1550– 1565, Nov. 2004.

[10] K. Fuchs, F. Fink, and M. H. Schulz, "Dynamite: An efficient automatic test pattern generation system for path delay faults," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, pp. 1323–1335, Oct. 1991.

[11] K. T. Cheng and H. C. Chen, "Classification and identification of nonrobust untestable path delay faults," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 845–853, Aug. 1996.

[12] U. Sparmann, D. Luxenburger, K. T. Cheng, and S. Reddy, "Fast identificationof robust dependent path delay faults," *Proceedings of Design Automation Conference*, pp. 119–125, June 1995.

[13] "Octave: Interactive language for numerical computations," *http://www.gnu.org/software/octave/*.