On Accelerating Path Delay Fault Simulation of Long Test Sequences

I-De Huang[1*], Yi-Shing Chang[1], Suriyaprakash Natarajan[1], Ramesh Sharma[1], and Sandeep K. Gupta[2]

Intel Corp., USA[1]
{i-de.huang, yi-shing.chang,
suriyaprakash.natarajan,
ramesh.sharma}@intel.com

Electrical Engineering – Systems[2]
University of Southern California,
Los Angeles, CA, USA
sandeep@poisson.usc.edu

Abstract

In this paper, we propose an approach to accelerate path delay fault simulation of long test sequences. Several key ideas, namely judicious selection of path delay faults to be simulated, extraction of a compact set of necessary conditions to detect selected faults at primary inputs, and an on-demand selective simulation of input vectors based on their satisfaction of these necessary conditions, are proposed. We demonstrate the benefits of our methodology via experiments on benchmark circuits, with one large test case (S9234) showing a 114X speed-up over a traditional approach.

1. Introduction

Delay testing is necessary to ensure that each fabricated chip that is shipped to a customer operates correctly at the rated clock frequency. During silicon debug, the goal of delay testing is to identify and rectify design-related timing failures that prevent many fabricated chips from operating at rated frequency. During high-volume manufacturing (HVM), delay testing ensures that each chip is classified based on the clock frequency at which it works correctly (commonly called *speed binning*) or rejected. Speed binning is integral to commercial success of most high volume microprocessor designs.

Speed paths, i.e., circuit paths that limit a chip's clock frequency, are behind most timing-related failures. Since these paths are deemed timing critical, it is assumed that appropriate excitation of each such path by test vectors will expose the worse-case circuit delay. Hence, practical delay testing can be viewed in terms of two tasks, namely (i) identification of a set of paths for delay testing, and (ii) development of test vectors that excite these paths.

Path identification is often carried out in an ad hoc manner after first-silicon is obtained for a chip design. Increasingly, this process is being accelerated by identifying paths a priori, i.e., before first-silicon. This is achieved by using static timing analysis (STA) to identify paths that can have delays above a predetermined threshold. The objective is to identify a minimum set of paths that are guaranteed to include all speed paths.

The second task, namely generation of delay test vectors for paths identified in the above step, must be carried out in a manner that minimizes test application cost while maximizing test quality. While the majority of chips use scan-based methods for delay testing, for most high volume microprocessor designs, tradition and customer requirements necessitate the use of a set of vectors in an existing test suite, usually a large set of long functional test sequences accumulated over generations of a processor's design. To minimize delay test application cost, targeted delay test generation must be undertaken only for the paths that are not tested by the existing functional tests.

One way to identify speed paths that are covered/not-covered by the functional test suite, is to perform path delay fault simulation. However, these designs are extremely large and the numbers of selected paths can be large – even when only the paths within a small window of the rated cycle time are selected for simulation. Also, functional test sequences typically contain astronomical numbers (billions) of vectors. These make simulation times impractically high and no existing approach can be used for path delay fault simulation of functional sequences. This paper addresses this problem and proposes techniques that can significantly accelerate path delay fault simulation for long test sequences -- functional as well as random.

Testing for path delay faults has been dealt with extensively in the literature [1][2]. Various types of tests have been defined for a path delay fault [1]-[5]. Path delay fault simulation has been investigated extensively in terms of enumerative techniques [5][7][9][11][13] and non-enumerative techniques [6][8][10][12][14] to accelerate computation of path delay fault coverage. Paths that are false, i.e., those that do not affect the timing of a circuit

* The work was performed when the author was with USC. The work was supported in part by a grant from Intel Corporation.

and hence need not be simulated thereby improving overall simulation performance, have been investigated in [15]-[19]. The method proposed in this paper is orthogonal to techniques proposed earlier and can be applied in conjunction with most of those techniques.

The rest of paper is organized as follows. We first provide an overview of functional simulation in Section 2. In Section 3, necessary definitions are introduced and certain characteristics of functional vector fault simulation are identified. In Section 4, the proposed techniques for accelerating fault simulation are presented. Section 5 demonstrates the benefits of our approach with results on benchmark circuits, and Section 6 concludes the paper.

2. Focus of paper

Path delay fault simulation of a large circuit, such as a processor, with long sequences of functional vectors has certain unique characteristics and sub-tasks. The circuit targeted is a large sequential circuit with a large number of combinational blocks, and the test sequences are applied at the primary inputs of this large sequential circuit and responses are observed at it primary outputs. The path delay faults that are targeted by our approach are within the combinational regions of such a circuit, and can be several sequential stages away from the primary inputs and primary outputs. Let us now consider path delay fault simulation for a delay fault in path P_i in a combinational block B_j. Fault simulation of a single vector v_k in the functional sequence entails three main subtasks.

1. **Fault-free simulation:** Simulation of vector v_k to determine the values it implies at the flip-flops at the inputs of block B_j. This entails fault-free simulation for every logic block between the primary inputs and the inputs of block B_j.
2. **Path delay fault simulation** for the delay fault on path P_i within block B_j. This task is identical to path delay fault simulation for a combinational logic block or a full-scan circuit.
3. **Propagation of fault effect** captured at the flip-flops at the outputs of block B_j to some primary outputs of the circuit under simulation.

Since the first task entails simulation of a fault-free version of the circuit, we assume that a high-level simulation methodology is used. We ignore the computational effort required for such simulation, since such a simulator can use abstract behavioral descriptions for many logic blocks and can be orders of magnitude faster than gate-level logic simulation. The third task entails the propagation of fault effects that take the form of erroneous logic values captured at flip-flops at one or more outputs of block B_j. Such fault effects can be propagated in a manner that is identical to that for stuck-at fault simulation. Hence, the complexity of this step can be reduced to the same level as that for stuck-at fault simulation. Also, recent research [20] has suggested that obtaining a reasonable estimate of path delay fault coverage and untestability may be possible through logic simulation alone, by counting the number of times a path is excited, thus avoiding altogether the overhead of the third sub-task. Development of efficient algorithms, accurate and approximate, for this sub-task is a subject of our ongoing research.

Having set aside the first and the last of the above three sub-tasks, in this paper we concentrate on the second sub-task, namely path delay fault simulation for a combinational logic block. Hence, the objective of this paper is to accelerate such path delay fault simulation for a combinational logic block by an order of magnitude (or more) compared to classical approaches. The results of our research in this paper (on the second sub-task) have a direct and significant impact on our overall goal of accelerating path delay fault simulation of large sequential circuits for long functional test sequences. Since this sub-task involves analysis of only the combinational logic to which the path that is being tested belongs, we demonstrate the merits of our approach on combinational versions of ISCAS89 benchmark circuits in Section 5. Our approach can also be directly applied for path delay fault simulation with built-in self-test and scan-based approaches.

3. Intuition behind our approach

Figure 1 shows a typical coverage graph obtained as a result of path delay simulation of long functional sequences. This fault coverage graph is similar in shape to that of stuck-at faults with an initial sharp ramp and an ensuing relatively flat region. However, the initial ramp for path fault coverage curve would be significantly less steep than that for stuck-at, and the eventual coverage attained would also be much smaller that that for stuck-at faults. The complexity for simulating N_v fully-specified vectors for N_f faults is proportional to the shaded region, and is reduced significantly mainly due to *fault dropping*. As we can observe from the figure, we may view the coverage curve as three phases, P_1, P_2, and P_3.

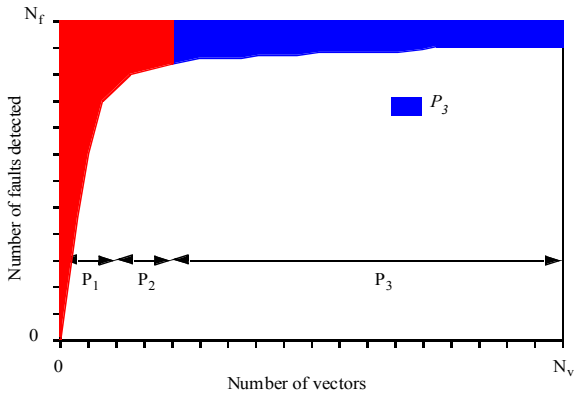

Figure 1. The complexity of fault simulation for functional vectors.

In phase P_1, the curve has a steep slope as each vector applied can easily detect many faults for the first time. As many faults are detected by previous vectors, each vector

in phase P_2 detects fewer faults for the first time. Then our hierarchical approach (described ahead) can be invoked. In this phase, simulation is performed for appropriate faults in appropriate parts of the circuit, *only for vectors that satisfy certain necessary conditions*. After the fault simulation, if the current vector detects a fault, the fault is dropped from the fault list.

By the time the simulation enters phase P_3, only the *hardest-to-detect faults* or *untestable faults* remain in the fault list, and the slope of curve becomes extremely low. The overall complexity of fault simulation for long test sequences is dominated by P_3, especially if many faults are untestable, because many vectors in this phase do not contribute to coverage. (Please note that N_v is not shown to scale in the figure.)

Typically, only path delay faults (PDF's) with high nominal delays are targeted for path delay fault testing. Since such paths are typically long (in terms of the sum of the numbers of inputs of their on-path gates), a test for such a path delay fault must satisfy *many necessary conditions* at primary inputs as well as many internal lines. Consequently, as shown in [21][22], many such path delay faults are *untestable*. Hence, simulation need be performed for a relatively *small number of paths*. Furthermore, in many circuits these path delay faults are confined to a *fraction of the circuit gates and lines*.

The above observations provide the rationale behind the approach presented in this paper.

4. Key ideas

In this section, we present our key ideas, namely pruning of an initial set of path delay faults to obtain a ***reduced set of faults***, ***selective circuit simulation***, and ***selective vector simulation***. These are followed by a description of our proposed multi-phase approach for fault simulation.

4.1. Definitions

We start by defining some necessary terms.

Logic value system: Throughout this paper we deal with sequences of two vectors, *even though for simplicity we often refer to them as vectors or tests*. In this paper, we use eight basic logic values, {CF, CR, S0, S1, T0, T1, 00, 11}, where CF stands for clean falling (no hazard), S0 stands for static 0 (no hazard), T0 stands for transition to value 0 (dynamic hazards possible), and 00 stands for hazardous 0 (static hazards possible). CR, S1, T1, and 11 are defined in a similar manner. We denote logic values at a line by using a subset of the basic value set.

A logic value V_i is a fully-specified value if V_i only contains exactly one of the eight basic logic values. V_i is a partially-specified value if it contains more than one but less than eight basic logic values. V_i is a fully-unspecified value (or a don't care value) if it contains all eight basic logic values. For two logic values V_i and V_j, we say V_i is covered by V_j, i.e., $V_i \subseteq V_j$, if V_j contains every basic logic value in V_i.

PV: A vector at primary inputs of a circuit, where the values at some of the inputs can be fully or partially specified. $PV_i = \{V_{i,1}, V_{i,2}, \ldots, V_{i,n}\}$, where n is the total number of primary inputs.

For two vectors $PV_i = \{V_{i,1}, V_{i,2}, \ldots, V_{i,n}\}$ and $PV_j = \{V_{j,1}, V_{j,2}, \ldots, V_{j,n}\}$, we say PV_i is covered by PV_j, i.e., $PV_i \subseteq PV_j$, if $V_{i,k} \subseteq V_{j,k}$ for all $k = 1, 2, \ldots, n$.

FV: A vector at primary inputs of a circuit that is fully specified. $FV_i = \{V_{i,1}, V_{i,2}, \ldots, V_{i,n}\}$, where $V_{i,k}$ is a fully-specified value for $k = 1, 2, \ldots, n$. Input vectors applied at inputs of a chip under test are of this type.

Logical path (LP): A logical path (LP) is a sequence of lines along a physical circuit path L_1 (a primary input), L_2, …, and L_n (a primary output) and a set of signal transitions Tr_1, Tr_2, …, and Tr_n, where $Tr \in \{R, F\}$, such that Tr_i represents the signal transition at L_i and F and R denote falling and rising transitions, respectively. Lines L_1, L_2, …, and L_n are called *on-path lines*. Gates along LP are *on-path gates*. If a line directly connects to an on-path gate but is not an on-path line, it is called a *side-input* of path LP.

4.2. Selection of path delay faults

This paper discusses techniques to accelerate fault simulation that are built on the path-oriented framework in [21][22], which is threshold-driven and has low complexity. A timing threshold (TT) value, say TT_{out}, is specified at the outputs and all paths that potentially have delays greater than TT_{out} are identified. The value of TT_{out} is (Cycle time - Δ), where Δ is a parameter selected by the circuit designer and is typically 10-20% of the cycle time. Then enhanced functional sensitization conditions proposed in [21] are applied at the side-inputs to identify and eliminate from consideration a large subset of the path delay faults that cannot cause any timing failures under the assumption that the accumulated values of additional delays along every circuit path is upper bounded by a specified limit, Δ. This process eliminates untestable path delay faults (PDF's) and provides a near minimum set of target path delay faults.

A vector that satisfies necessary logic conditions (which implicitly capture some necessary timing conditions) to excite a delay value larger than TT_{out} is identified at primary inputs for each target path using the integrated logic-and-timing implication proposed in [22]. (In this paper, we call this vector ***PV*** which was described above.) In addition, partially-specified values may also be associated with some internal circuit lines. The identified PV (and additional conditions, if any) can help reduce the complexity of fault simulation.

4.3. Selective circuit simulation

In many circuits, target path delay faults are confined to a fraction of the circuit. This fraction of the circuit is identified by marking the fan-in cone of each target PDF. Fault simulation is then limited to the circuit gates and lines in the marked region.

4.4. Selective vector simulation

As described above, for each PDF of interest a PV is obtained. This PV represents (a subset of) the necessary conditions that must be satisfied by any vector that detects the PDF.

Consider an FV_i to be simulated. We first compare FV_i with the PV to check whether the PV covers FV_i. If FV_i is not covered by the PV, FV_i cannot detect the target PDF because it does not satisfy all necessary conditions for its detection. Hence, fault simulation need not be performed for FV_i if it is not covered by PV of any PDF of our interests. On the other hand, if FV_i satisfies the conditions at inputs, additional logic values can be checked at internal lines as logic simulation proceeds. This simulation is stopped whenever the pre-computed necessary logic values cannot be satisfied at any line.

We refer to this process of performing simulation only for vectors that satisfy some conditions as ***selective vector simulation***.

4.4.1 Merging of necessary conditions

Since a PV is identified for each PDF, a list of PVs is maintained. If every PV for each PDF is recorded separately in the list, a large number of comparisons are required for each input vector. On the other hand, if all PVs can be merged into a small set of PVs, the resulting vector space is significantly expanded. The number of comparisons is reduced when we combine PVs, but the number of vectors for which fault simulation is performed may be increased significantly. Hence, next we present a heuristic approach to merge PVs without expanding the vector space, while attempting to reduce the number of comparisons.

First, we identify and discard redundant PVs. PV_i is redundant if a PV_j can be found such that $PV_i \subseteq PV_j$. PV_i is discarded as illustrated in Figure 2(a).

After redundant PVs are eliminated, we start the process of merge. PV_k is merged with PV_j if there exists only one input m such that $V_{k,m} \neq V_{j,m}$. This merge process is iteratively performed until no two PVs can be merged. Figure 2 illustrates the merge process.

Please note that the analysis to identify and merge PVs solely depends on the number of target faults. More importantly, this is a one-pass analysis that is performed prior to simulation of vectors and its complexity is independent of N_v, the number of vectors to be simulated.

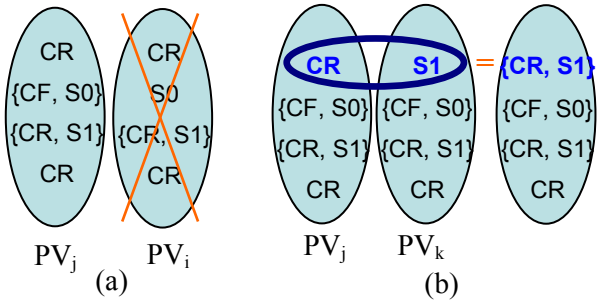

Figure 2. An example to illustrate (a) redundant *PV* identification, and (b) the merge process.

4.4.2 Input prioritization algorithm

Each PV (after merging) is associated with a set of PDFs. A set of such PVs represents necessary conditions for all target PDFs. Prior to performing fault simulation with an input vector, our approach is to compare the input vector with each PV, and only if the input vector is covered by a PV, performing fault simulation for the vector for the PDFs corresponding to that PV. We now discuss a heuristic for efficient comparison of an input vector with the PVs to quickly identify non-satisfaction of PVs by the input vector. This aids in quickly deciding if fault simulation can be avoided for a vector.

For each FV, there is a set of inputs corresponding to the support set (i.e., inputs in the fan-in cone) of the outputs of the target PDFs represented by that PV. The union of these sets of circuit inputs over all the PVs is defined as the set S_{sup}. The inputs in S_{sup} are the only inputs relevant for value comparison during fault simulation with each vector to be simulated. Even among these inputs, only those inputs that do not have don't care value (XX) in at least one PV are relevant for value comparison with each input vector. Let this be defined as the set S'_{sup}. The objective is to prioritize the inputs in S'_{sup} for value comparison. We compute a metric for this prioritization based on a heuristic. The idea is to first compare the value at an input whose likelihood to be satisfied by an input vector is low across most PVs.

Note that each PV has a value at each circuit input in the set S'_{sup} which can be only one of {CR, CF, S0, S1}, since every vector applied to a line must be fully-specified and the values at inputs can have no static or dynamic hazards (as these are typically driven by flip-flops). We assume that input vectors are random and that an input vector can provide any one of these four values at a circuit input with equal probability. In a PV, if a certain input has a don't care value (i.e., the entire set {CR, CF, S0, S1}), then any input vector will satisfy the PV requirement for that input, and the failure probability for that input is 0. On the other hand, if a certain input of a PV has value {CR, S0, S1}, then the probability that an input vector provides a value that is not in this set is ¼ -- which is termed the failure probability. Using this method, a metric for each input in the set S'_{sup} is computed as the average of the fail-

ure probabilities of that input over all PVs. Inputs in S'_{sup} are ranked in decreasing order based on this metric. During fault simulation the inputs are chosen in this order for comparison with input vector values. If no PV is identified to satisfy the necessary values at any input, no fault simulation is carried out for this vector.

Figure 3 describes the pseudo code of the proposed value comparison procedure at inputs before performing fault simulation. For each vector FV_i to be simulated, each input is compared in our prioritized order with each PV. All PVs that cannot satisfy the necessary conditions at that input are removed. If no PV remains, FV_i is discarded *without performing fault simulation*. Otherwise, *fault simulation is performed only for the faults associated with remaining PVs*.

```
for every fully-specified vector FV_i to be simulated{
    for every ordered primary input PI_j {
        for every PV_k {
            // V_k,j is the logic value at PI_j for PV_k
            if(FV_i,j ⊆ V_k,j) {
                if(PI_j is the last) {
                    if(internal lines satisfy conditions)
                        perform fault simulation();
                    next FV; }
                else next PI; }
            else {
                remove PV_m that has V_m,j ⊆ V_k,j;
                if(no PV remains) next FV; } } } }
```

Figure 3. The pseudo code of the proposed input comparison procedure for fault simulation.

Since each functional vector is compared at inputs to determine if fault simulation is required, a link list is used to maintain the data structure of PVs and its corresponding PDF's as illustrated in . If a simulated vector FV_i can only satisfy the conditions for PV_2, the fault simulation is performed only for PDF_4.

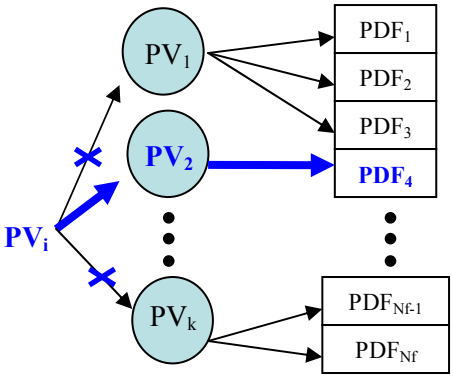

Figure 4. Data structure for maintaining the relationship between PVs and PDF's.

5. Experimental results

We evaluate our path delay fault simulation methodology, which we call the **PFS** strategy, via experiments on combinational parts of ISCAS89 benchmark circuits. We compare our results with an approach without our efficiency improvement techniques, which we call the **GFS** strategy. An Intel Core 2 Duo 2.13 GHz machine is used for these experiments. All gates in the benchmark circuits are assumed to use minimum-size transistors.

The maximum circuit delay (using nominal delay values for gates) as computed via static timing analysis is described as T_c. We select two sets of paths for each circuit: corresponding to a TT_{out} value of $0.9T_c$ and a TT_{out} value of $0.8T_c$, targeting delay defects whose cumulative value along each path in these sets being upper-bounded by $0.1T_c$ and $0.2T_c$, respectively. The results are shown below for each set of paths for each step of our approach.

5.1. Untestable path elimination

Path pruning techniques were used to reduce the number of paths for both the GFS and PFS strategies.

In the GFS strategy (base-line), untestable paths are identified and removed from the path fault list using classical functional sensitization conditions. The reduction achieved in this manner is presented as R' in Table 1.

Table 1. Path reduction with classical functional sensitization conditions in GFS.

ISCAS 89 circuits	$0.9T_c$			$0.8T_c$		
	Before	After	R'	Before	After	R'
S298	9	9	1.0	50	44	0.88
S444	51	42	0.82	104	88	0.85
S713	7354	412	0.06	12,852	996	0.08
S953	21	21	1.0	80	80	1.0
S1196	151	148	0.98	558	427	0.77
S1494	22	22	1.0	51	51	1.0
S5378	2,983	2,546	0.85	7,568	6,258	0.83
S9234	17,280	8,032	0.46	85,588	26,972	0.32

In the PFS strategy, the original path list for a given TT_{out} is processed with enhanced functional sensitization conditions [21] and integrated logic-timing implications [22] to eliminate paths that need not be tested. The results of path reduction with this approach are presented in Table 2. The fraction of paths that remain after untestability analysis, given by R_1, is shown in Table 2.

For example, for circuit S298, for $TT_{out} = 0.9T_c$, the initial number of paths is 9 (Column 2), and untestability analysis reduces the potential testable paths to 3 (Column 3), signifying a reduction to 33% (fraction of 0.33 shown in Column 4). For circuit S713, there are no paths with delay longer than TT_{out} for $TT_{out} = 0.9T_c$, and only 2% of the paths are testable for $TT_{out} = 0.8T_c$. For another large circuit, S9234, untestability reduces the paths to about 4%

(3%) of the original number of paths for $TT_{out} = 0.9T_c$ ($0.8T_c$).

Table 2. Untestable path elimination in PFS.

ISCAS 89 circuits	$0.9T_c$			$0.8T_c$		
	Before	After	R_1	Before	After	R_1
S298	9	3	0.33	50	6	0.12
S444	51	31	0.61	104	56	0.54
S713	7354	0	0.00	12,852	228	0.02
S953	21	14	0.67	80	49	0.61
S1196	151	3	0.02	558	21	0.04
S1494	22	3	0.14	51	23	0.45
S5378	2,983	686	0.23	7,568	3,033	0.40
S9234	17,280	646	0.04	85,588	2,690	0.03

It can be observed that by comparing corresponding columns for R' in Table 1 with those for R_1 in Table 2, the path reduction achieved with the PFS strategy is significantly higher.

5.2. Circuit reduction

Simulation and coverage analysis need to be carried out only at those circuit lines that are located in the fan-in cone of the target PDF (see Section 4.3). The circuit reduction ratio, R_2, gives the number of the circuit lines in the fan-in cone of the target path delay faults, expressed as a fraction of the total number of circuit lines. Table 3 summarizes the fraction of circuit lines that need to be simulated for different TT_{out} values using our approach. The higher the TT_{out} value, the smaller is the fraction of the circuit lines where simulation and analysis must be performed. For example, for circuit S298, the total number of circuit lines is 345 (Column 2), and for $TT_{out} = 0.9T_c$, simulation is applied to a sub-circuit with only 96 circuit lines (Column 3), i.e., only to 28% (Column 4) of the lines in the original circuit.

Table 3. Reduction due to selective circuit simulation.

ISCAS 89 circuit	Total # of circuit lines	# of circuit lines simulated (#) and R_2			
		$0.9T_c$		$0.8T_c$	
		#	R_2	#	R_2
S298	345	96	0.28	96	0.28
S444	471	119	0.25	119	0.25
S713	824	-	-	399	0.48
S953	1,084	537	0.50	720	0.66
S1196	1,417	595	0.42	739	0.52
S1494	2,052	467	0.23	1,333	0.65
S5378	5,700	2,586	0.45	4,077	0.72
S9234	10,620	2,918	0.27	2,918	0.27

5.3. PV compaction

We discuss the reduction achieved by identifying and eliminating redundant PVs for the path delay faults. Table 4 gives the results of this reduction after we perform redundant PV identification and PV merge process for different values of TT_{out}. For a circuit that has a small number of PDF's, the reduction is not significant. For example, in Table 4, for S298, for $TT_{out} = 0.9T_c$, 3 PVs (Column 2) only decrease to 2 (Column 3). However, for a larger circuit, S9234, the number of PVs is significantly reduced for both path sets (corresponding to $TT_{out} = 0.8$ and $0.9T_c$).

Table 4. Redundant PV identification and merge process.

ISCAS 89 circuit	# of *PV*s before and after the redundancy identification and merge process			
	$0.9T_c$		$0.8T_c$	
	Before	After	Before	After
S298	3	2	6	4
S444	31	7	56	12
S713	0	0	228	8
S953	14	7	49	15
S1196	3	2	21	13
S1494	3	3	23	17
S5378	686	335	3,033	923
S9234	646	32	2,690	32

The number of inputs in the sets of S_{sup} and S'_{sup} (refer to Section 4.4.2) are given in Table 5.

Table 5. The sizes of S_{sup} and S'_{sup}.

ISCAS 89 circuit	Total # of primary inputs	Size of S_{sup} and S'_{sup}			
		$0.9T_c$		$0.8T_c$	
		S_{sup}	S'_{sup}	S_{sup}	S'_{sup}
S298	17	9	7	9	7
S444	24	14	9	14	9
S713	54	0	0	27	23
S953	45	20	15	20	16
S1196	32	23	13	24	13
S1494	14	14	10	14	14
S5378	214	138	93	171	113
S9234	247	97	29	97	27

5.4. Run time reduction

In order to demonstrate the benefits of using proposed techniques for fault simulation, performance of a baseline timing-aware path delay fault simulator (GFS) is compared with the performance of an optimized timing-aware path delay fault simulator (PFS) that incorporates our above efficiency improvement techniques. The frameworks for these two simulation strategies are illustrated in Figure 5. Identical sequences of random vectors are used for these simulations.

In the GFS strategy, for all paths remaining after removing untestable paths using classical functional sensitization conditions (refer to Table 1), all vectors are simulated with each remaining path delay fault using the timing-aware fault simulation, which comprises of timing

simulation followed by a check of logic and timing conditions to infer detection of each fault (i.e., if each node along a path is sensitized to the preceding on-path node). A fault is dropped from the fault list when it is detected.

In the PFS strategy, the paths remaining after removing untestable paths using enhanced functional sensitization conditions [21] (refer to Table 2) are processed to obtain an initial set of PVs. The path fault list is maintained for later use for fault dropping during timing-aware delay fault simulation. The redundant PV identification and PV merge process are performed to obtain the final set of PVs. These PVs are stored in PV data structure where each PV is associated with its corresponding fault(s) as illustrated in .

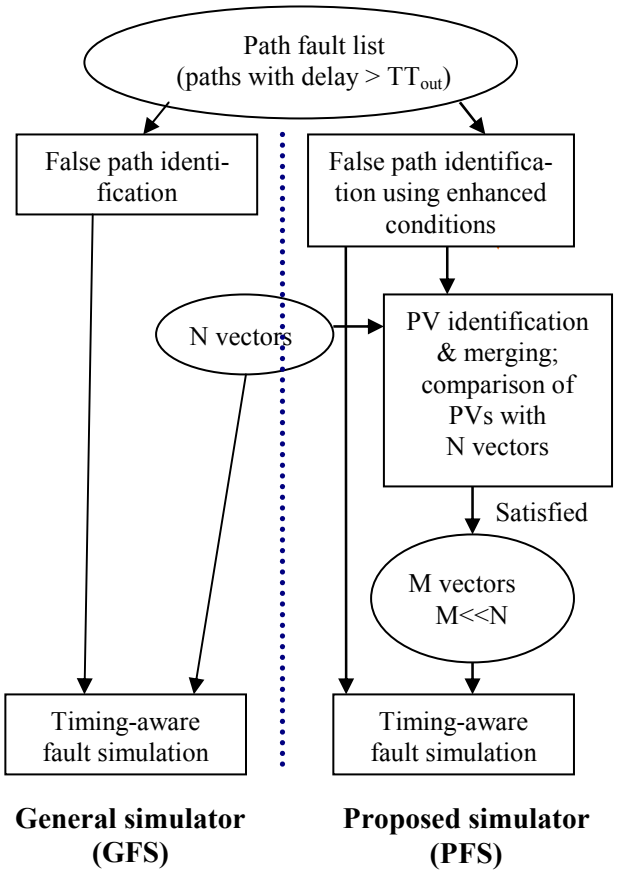

Figure 5. The frameworks of a general PDF simulator and our proposed PDF simulator.

Subsequently, each input vector (FV) is compared with generated necessary conditions for paths (PVs) to verify satisfaction. If the vector satisfies the conditions denoted by any PV, timing-aware fault simulation is performed for the corresponding faults. If not, the vector is bypassed for fault simulation. Please note that only the proposed PFS strategy works with a smaller path list due to the application of false path identification using enhanced functional sensitization conditions.

We now compare the overall performance of the GFS and the PFS strategies. Before that, we provide the break-up of the run-time for different steps in the PFS strategy. The PFS approach has four distinct steps compared to GFS .

1. T_{IS}: Time for identifying and eliminating untestable paths using enhanced functional sensitization conditions and for identifying a PV for each remaining target PDF.
2. T_{me}: Time for identifying redundant PVs and merging PVs.
3. T_{co}: Time for comparing values at inputs for all simulated vectors. Note that this parameter depends on the number of vectors that are simulated.
4. T_{FS}: Time for performing fault simulation.

The run time for the GFS approach is just the corresponding fault simulation time (as the run-time for identifying and eliminating untestable paths with classical functional sensitization conditions is negligible).

Table 6 shows the individual run-times of the different steps for 10K and 1M (1 million) vectors in PFS. The run times for T_{me} are negligible compared to those of the rest and hence are not reported. As can be seen from Table 6, the run times are dominated by T_{IS} and T_{FS} for almost all circuits except S5378, even though T_{co} is performed for each vector. In fact, T_{IS} the dominant contributor to the run-time when the number of vectors simulated is small. However, as we will see shortly in the comparison with GFS, the increase in T_{IS} for PFS is offset by a more significant reduction in T_{FS} (which scales with number of vectors) thereby reducing the overall run-time when compared to the performance of GFS.

For S9234 with $TT_{out} = 0.8T_c$, PFS takes only 124 seconds to perform fault simulation for 1M vectors as shown in Table 6 (while GFS takes 32,478 seconds). The significant run time reduction comes from (1) the path reduction is 0.03 for PFS compared to 0.32 for GFS (please refer to Table 1 and Table 2), and (2) timing-aware delay fault simulation is performed for only 3,368 out of 1M vectors for PFS.

Table 6. Run-times for different steps of PFS for 10K and 1M vectors.

ISCAS 89 circuit	Run-times for PFS steps (seconds)									
	$0.9T_c$					$0.8T_c$				
	T_{IS}	T_{co}		T_{FS}		T_{IS}	T_{co}		T_{FS}	
	both	10K	1M	10K	1M	both	10K	1M	10K	1M
S298	0.02	0.02	0.16	0.17	16.3	0.02	0.00	0.19	0.24	20.7
S444	0.11	0.01	0.5	0.2	23.5	0.19	0.00	0.87	0.58	58.9
S713	-	-	-	-	-	0.85	0.00	0.25	0.08	10.2
S953	0.09	0.02	0.23	0.19	14.1	0.30	0.00	0.55	0.2	23.6
S1196	0.03	0.00	0.14	0.16	7.63	0.19	0.00	0.71	0.08	9.95
S1494	0.05	0.00	0.12	0.56	56.2	0.37	0.02	1.24	2.95	306
S5378	19.4	2.46	223	90.3	9,186	87.4	11.4	1,030	139	11,888
S9234	43.6	0.03	1.42	0.59	53.2	158	0.03	2.9	1.17	124

* The run times for T_{me} are negligible compared to those of the rest and hence are not reported.

The only exception is S5378, for which the reduction in the number of paths achieved through timing based false

path identification is not as significant as for other circuits (please refer to Table 1 and Table 2). In addition, from Table 4, we note that after merge the number of PVs remains significantly high due to unmergeable PVs for individual paths, and almost all of vectors are simulated. (For $TT_{out} = 0.8T_c$, timing-aware fault simulation is performed for 999,879 out of 1M vectors.) These account for the significant fault simulation time (T_{FS} columns in Table 6) compared to other circuits.

We now explore how this speed-up in performance of PFS scales with the length of test sequence. For this experiment, we simulated random vector sequences of increasing lengths -- 5K, 10K, 20K, and 1M vectors. The factor of speed-up of PFS over GFS is given for the different sequence lengths in Table 7.

Table 7. Cumulative run-time speed-up with PFS over GFS for different number of vectors.

ISCAS 89 circuit	Speedup (X) using PFS for different number of vectors							
	$0.9T_c$				$0.8T_c$			
	5K	10K	20K	1M	5K	10K	20K	1M
S298	26.2	32.4	36.7	41.6	22.7	26.2	29.7	32.6
S444	20.7	31.8	38.1	39.2	9.8	12.5	15.6	15.9
S713	-	-	-	-	15.9	31.9	59.3	288
S953	37.2	54.7	72.3	113	21.1	32.8	44.5	67
S1196	120	157	201	383	60.2	100	103	278
S1494	66.6	76.1	78.9	82.2	12.3	14	14.5	15
S5378	0.89	1.04	1.15	1.23	0.41	0.57	0.71	0.93
S9234	2.99	5.93	11.8	278	1.07	2.13	4.22	114

As expected, since T_{IS} is a constant cost, T_{FS} dominates the total cost for long sequences. In such a situation, PFS outperforms GFS significantly as is illustrated by the results in Table 7. For example, for circuit S9234, the speed-up ratio increases by a factor of 93 (from 2.99 to 278) for an increase in test length by a ratio of 200 (i.e., from 5K vectors to 1M vectors) for $TT_{out} = 0.9T_c$. For all circuits, there is an increase in the speed-up factor though some are modest. Even for S5378 for $TT_{out} = 0.9T_c$, whose speed-up is fractional (meaning GFS performs better) for 5K vectors, PFS is catching up with GFS on performance and eventually exceeds GFS performance after simulating 10K vectors. However, for $TT_{out} = 0.8T_c$, GFS performs better than PFS even for 1M vectors due to the reasons earlier described.

6. Conclusion

In this paper, we propose an efficient approach for path delay fault simulation of long test sequences. Key components of our approach are a new technique for selection of path delay faults, and completely new notions of selective circuit simulation and selective vector simulation.

We first identify path delay faults for a specified TT_{out} value and eliminate untestable paths from that set. Second, the fraction of the circuit lines where simulation is required is identified. Third, necessary conditions for exciting the paths are obtained and then compacted at primary inputs and some internal lines. Finally, input vectors from long test sequences are compared against these compacted necessary conditions, and fault simulation is performed only for input vectors that satisfy necessary conditions.

Experiment results demonstrate that the performance of our approach accelerates path delay fault simulation **by large factors**. Results also show that the performance benefits of our approach are amplified for longer test sequences.

Reference

[1] G. L. Smith, "Model for delay faults based upon paths", Proc. Int'l Test Conference, Nov. 1985, pp: 342-349.

[2] C. J. Lin and S. M. Reddy, "On delay fault testing in logic circuits", IEEE Trans. on CAD, Vol. 6, No. 5, Sept. 1987, pp: 694-703.

[3] K.-T. Cheng and H. C. Chen, "Classification and identification of non-robust untestable path delay faults", IEEE Trans. on CAD of IC and Systems, Vol. 15, No. 8, Aug. 1996, pp: 845-853.

[4] M. H. Schulz, K. Fuchs and F. Fink, "Advanced automatic test pattern generation techniques for path delay faults", Proc. 19th Int'l Symp. on Fault Tolerant Comp., June 1989, pp: 44-51.

[5] F. Fink, K. Fuchs and M. H. Schulz, "Robust and nonrobust path delay fault simulation by parallel processing of patterns", IEEE Transactions on Computers, Vol. 41, No. 12, Dec. 1992, pp: 1527 - 1536 .

[6] K. Heragu, V. D. Agrawal and M. L. Bushnell, J. H. Patel, "Improving a nonenumerative method to estimate path delay fault coverage", IEEE Trans. on CAD of IC and Systems, Vol. 16, No. 7, July 1997 pp: 759 – 762.

[7] T. J. Chakraborty, V. D. Agrawal and M. L. Bushnell, "Path delay fault simulation of sequential circuits", IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 8, No. 2, April 2000, pp: 223 – 228.

[8] D. Kagaris and S. Tragoudas, "On the nonenumerative path delay fault simulation problem", IEEE Trans. on CAD of IC and Systems, Vol. 21, No. 9, Sept. 2002, pp: 1095 – 1101.

[9] S. Bose, P. Agrawal and V. D. Agrawal, "Path delay fault simulation of sequential circuits", IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 1, No. 4, Dec. 1993, pp: 453 – 461.

[10] I. Pomeranz and S. M. Reddy, "On the number of tests to detect all path delay faults in combinational logic circuits", IEEE Trans. on Computers, Vol. 45, No. 1, Jan. 1996, pp: 50 – 62.

[11] Kee Sup Kim, R. Jayabharathi and C. Carstens, "SpeedGrade: an RTL path delay fault simulator", Proc. 10th Asian Test Symposium, 2001. Nov. 2001, pp: 239 – 243.

[12] I. Pomeranz and S. M. Reddy, "An efficient nonenumerative method to estimate the path delay fault coverage in combinational circuits", IEEE Trans. on CAD of IC and Systems, Vol. 13, No. 2, Feb. 1994, pp: 240 – 250.

[13] T. J. Chakraborty, V. D. Agrawal and M. L. Bushnell, "Path delay fault simulation algorithms for sequential circuits", Proc. 1st Asian Test Symposium, Nov. 1992, pp: 52 – 56.

[14] S. Padmanaban, M. K. Michael and S. Tragoudas, "Exact path delay fault coverage with fundamental ZBDD operations",

IEEE Trans. on CAD of IC and Systems, Vol. 22, No 3, March 2003, pp: 305 – 316.
[15] P. Kalla and M. Ciesielski, "Testability of sequential circuits with multi-cycle false paths", Proc. 15th IEEE VLSI Test Symposium, May 1997, pp: 322-328.
[16] Z. Hasan and M. J. Ciesielski, "Elimination of multi-cycle false paths by state encoding", Proc. European Design and Test Conference, Mar. 1995 pp: 155 – 159.
[17] P. Ashar, S. Dey and S. Malik, "Exploiting multi-cycle false paths in the performance optimization of sequential circuits", Proc. IEEE/ACM International Conference on Computer-Aided Design, Nov. 1992, pp: 510 – 517.
[18] W.-C. Lai, A. Krstic and K.-T. Cheng, "Functionally testable path delay faults on a microprocessor", Proc. IEEE Design & Test of Computers, Vol.17, No. 4, Oct.-Dec. 2000 pp: 6 – 14.
[19] S. Kundu, C. Tirumurti, R. Jayabharathi and P. Parvathala, "A Path Delay Fault Simulation System," Proc. 7th IEEE European Test Workshop, May 2002.
[20] S. Natarajan, S. Patil and S. Chakravarty, "Path Delay Fault Simulation of Large Industrial Designs," Proc. IEEE VLSI Test Symposium, 2006.
[21] I. D. Huang and S. K. Gupta, "Selection of Paths for Delay Testing", Proc. Asia Test Symp., 2005, pp: 208 - 215.
[22] I. D. Huang and S. K. Gupta, "On Generating Vectors That Invoke High Circuit Delays – Delay Testing and Dynamic Timing Analysis", Proc. Asia Test Symp., 2007, pp: 479 - 486.