A Non-Enumerative Path Delay Fault Simulator for Sequential Circuits

Carlos G. Parodi
Rutgers University
Piscataway, NJ 08855
remus@ece.rutgers.edu

Vishwani D. Agrawal
Bell Labs
Murray Hill, NJ 07974
va@research.bell-labs.com

Michael L. Bushnell
Rutgers University
Piscataway, NJ 08855
bushnell@caip.rutgers.edu

Shianling Wu
Bell Labs
Princeton, NJ 08542
shianlingwu@lucent.com

Abstract

We extend the path status graph *(PSG) method of delay fault simulation to sequential circuits. By devising a layered PSG and restricting the number of time-frames over which a fault must be detected, we preserve the non-enumerative nature of the simulation algorithm. The program is capable of simulating a wide variety of circuits (synchronous, asynchronous, multiple-clock and tri-state logic.) Both rated and variable clock modes, as well as robust, non-robust or functional sensitization detection options, are available. The simulation can be stopped and restarted through a checkpointing facility. The program can target any given list of paths. This path list can also be generated by the program based on user-selectable criteria (all paths, longest paths, paths between certain I/O pairs, etc.) User reports include a histogram of path coverage versus path length. Detected and undetected path data remain implicit in the PSG and can be retrieved through post-processing commands. Due to its non-enumerative nature, the program can process most production level digital logic circuits.*

1 Introduction

In order to guarantee that a VLSI circuit does not have fabrication defects, we often test for stuck-at faults. Such testing is based on the assumption that a fabrication defect will cause a permanent logic change in the circuit. As the operational circuit frequency increases, the delay of a signal transition can cause a wrong value to be latched. This fault due to delay behavior is captured by the delay fault model. While the stuck-at fault model has been widely accepted by the industry, use of the delay fault model presents practical difficulties.

The *gate-delay* fault model assumes that the delay of a single gate can be catastrophic (greater than the circuit's clock period). Another fault model, known as the *path-delay* fault, assumes that delays of a chain of gates (a path) cumulatively causes a failure. The gate-delay fault model is often regarded as unrealistic since it assumes a large delay for a single faulty gate. The path-delay fault model, on the other hand, has the shortcoming that the number of paths in a circuit can be exponential in the number of logic gates [1].

Recent work has shown the possibility of non-enumerative simulation techniques for path delay faults [2]. Although exact simulation for combinational circuits has been demonstrated, such methods have no straightforward extensions to sequential circuits. In this paper, we extend the *path status graph* (PSG) technique for non-enumeratively simulating path delay faults in sequential circuits. The testability of all paths is recorded in a graph by tagging the path edges. Although in the worst case the method can be as memory intensive as any enumerative approach, experiments show that this rarely happens. This work now provides a path delay fault simulation capability in Bell Labs' *Gentest* environment [3].

2 Background

In 1985, Smith [4] proposed a path delay fault model. To simulate path delay faults in a circuit, he developed a six-valued algebra. Since the delay of other paths can affect the test for a target path, two detection modes are possible: an optimistic or non-robust mode and a pessimistic or robust mode. Suitable changes in the algebra allow these modes. The non-robust algebra works with the assumption that all other paths aside from the path-under-test have no excessive delay. The robust algebra does not make this assumption and, as a result, yields lower fault coverage [1].

Chakraborty *et al.* [5] analyzed path-delay faults in sequential circuits and developed a 13-valued algebra and algorithms for simulation and automatic test pat-

0-7803-5092-8/98 $10.00 © 1998 IEEE

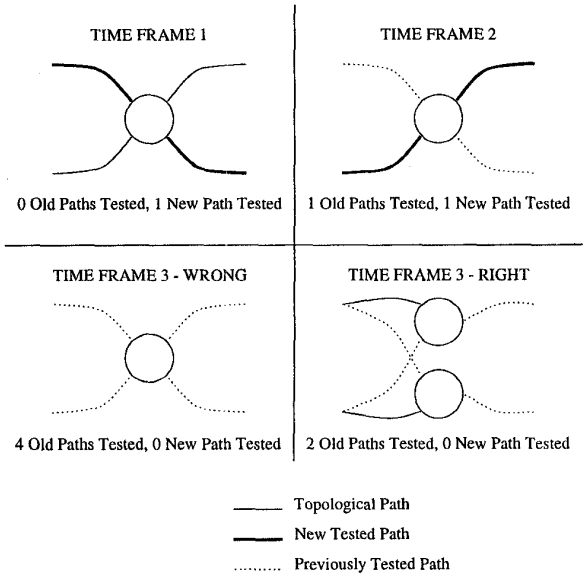

Figure 1: Path status graph node split

tern generation. Bose *et al.* [6] extended the sequential path delay fault simulation analysis and developed a faster simulator based on fault list management and the *optimistic update rule*, which for non-robust simulation would eliminate much of the redundant simulation of captured faults.

To solve the problem of the large number of paths, Pomeranz and Reddy [7] and Heragu *et al.* [8] have proposed approximate non-enumerative methods based upon counting of tested paths. The precision of these methods depends on the amount of memory used. Exact results can be obtained, but would require the same memory as needed for path enumeration.

Gharaybeh *et al.* [2] calculated path delay fault coverage without enumerating the tested and untested paths. They recorded the information obtained from the delay fault simulation in a graph, known as the *path status graph* (PSG). This method has an advantage over other non-enumerative methods in that it has absolute precision and yet it requires a minimal amount of memory.

The PSG is a graph derived from the circuit connectivity description. Only gates with fanouts greater than one are represented as nodes in the PSG, which contains the complete data on all paths. To preserve the information on detected and undetected path faults, the edges of the PSG contain flags. Initially, all paths are assumed to be in the *untested* state. As simulation proceeds, flags are appropriately changed. At any time in simulation, the number of tested or untested paths can be determined by a linear-time path-counting algorithm. Actual details of all paths remain implicit in the graph, and can be retrieved as needed.

Figure 1 shows how the PSG preserves accuracy by splitting nodes. Here, three vector-pairs (shown as time-frames) are simulated. In time-frame 1, one path (shown in bold) is tested. A second path is tested in time-frame 2. Now, all edges connected to the node have been included in tested paths. Still, the node has two more untested paths. Thus, before time-frame 3 is initiated, the node is split. In the split graph, the numbers of total, tested and untested paths are preserved. We obtain the number of tested paths by counting paths only containing the tested (dotted) edges. Similarly, untested paths are correctly obtained as those with at least one untested edge (thin solid). Actual simulation also uses a recombination procedure to collapse the previously-split nodes when possible. Only in the worst case does the node count in the PSG increase exponentially. In most practical cases, it remains reasonable [2].

It was recently observed that most path-delay fault simulators assumed that all signals are stable before the application of the transition [9]. This assumption is correct for a variable clock testing methodology [5], where the circuit operation speed is slowed down (with the exception of one specific transition-causing vector applied at-speed). When all vectors are applied at the rated clock, the results obtained by simulation assuming a variable-clock are overly optimistic [9, 10, 11].

3 Extension to Sequential Circuits

In a sequential circuit, many paths terminate at flip-flops that are not directly observable. Thus, the effect of an activated path might not appear at a primary output until several clock cycles after the activation. In order to correctly simulate this situation, the effect of activated paths that are not directly observable must be stored and treated separately.

A path enumeration type of simulator commonly stores the effect of an activated path as a pseudo stuck-at fault effect in the destination flip-flop [6, 12, 13]. This technique requires the storage of one fault effect per path and its use in our context will defeat the advantage of the non-enumerative simulator. We, therefore, record the cones of all activated paths that terminate at the same flip-flop as a single fault effect. We also save the cone of paths in the PSG format for the complete fault description. Figure 2 illustrates the difference between the enumerative and non-enumerative techniques. This fault record (one per cone) requires a minimal amount of memory; the record consists of the destination flip-flop, the time frame during which the fault originated, and the true value that would be latched if the circuit

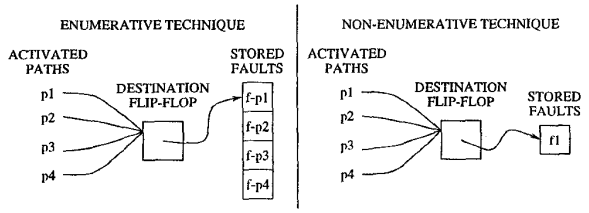

Figure 2: Non-enumerative fault storing

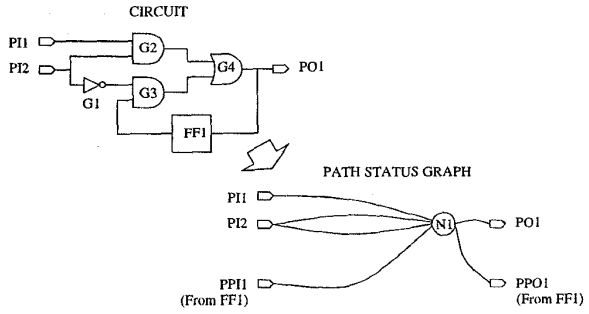

Figure 3: Sequential PSG construction

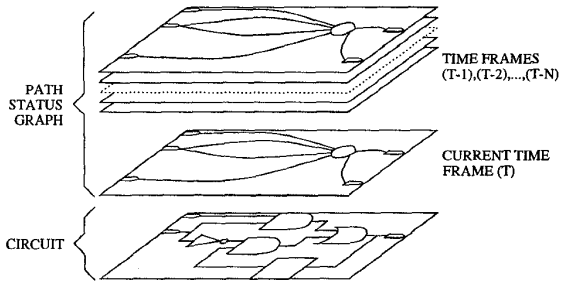

Figure 4: Layered sequential PSG

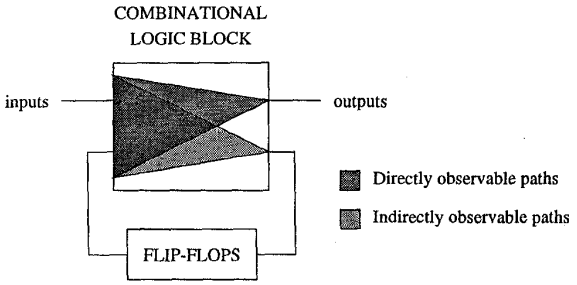

Figure 5: Path types according to destination

were not faulty. No additional memory is required to store the cone of paths that gives rise to a fault because (as will be shown later) that information resides in the edge tags of a specific layer of the PSG (layer that corresponds to the time frame in question).

In order to avoid cycles in the path status graph of sequential circuits, flip-flop nodes are considered as pseudo *primary input/output* pairs. Figure 3 shows an example of how a sequential PSG is constructed.

To facilitate the above fault storing technique, we modified the PSG data structure. Instead of recording the activated paths all in the same graph, we added layers to the graph. In the first (bottom) layer, all of the activated paths of the current time-frame are tagged (the ones that are directly observable and others whose effects might be observed in the future time frames). In addition, this layer also keeps a record of *all* paths that have been tested. Higher layers contain flags corresponding to the previously activated and yet unobserved paths. Each layer is a snap-shot of the PSG of one time-frame. As the time-frame advances, the PSGs are moved one layer up (see Section 3.2). Using this new data structure, identification of a stored fault is done by identifying the affected flip-flop and the time frame of fault occurrence (as opposed to enumerating all of the the paths that gave rise to that fault).

The non-enumerative sequential circuit path delay fault simulation has two parts: simulation of combinational logic blocks and stored fault list management.

The first part consists of the event-driven simulation of the active portion of the circuit using the delay fault algebra, and the recording of the circuit activity in the current time frame layer of the PSG. The possible fault effects due to the activated paths that terminate at *primary outputs* (POs) are directly observed, and therefore considered tested. The possible faults due to the activated paths that terminate at flip-flops cause the wrong value to be latched in the destination flip-flop. This is a pseudo stuck-at fault. Figure 5 shows these as *indirectly observable paths.*

The second part of the simulation consists of creating a list of flip-flops affected by faults. The fault effects stored in flip-flops might become observable in subsequent time frames, testing the paths whose activations gave rise to the stored effects. In order to propagate those fault effects, the second step also consists of a pseudo stuck-at fault simulation using D's ($\overline{D}$'s).

3.1 Combinational Logic Block Simulation

The first part of the simulation is an event-driven true-value simulation. It is performed with the user's choice of a robust, non-robust or functional sensitization algebra. One queue is created for each topological level in the circuit. Inputs and memory elements (flip-flops) belong to the first level. The simulation al-

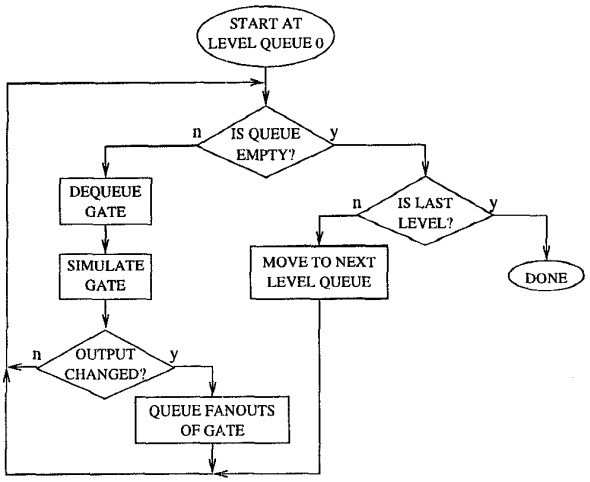

Figure 6: Simulation algorithm

Table 1: Simulation algebra

Value	Description
XX	unknown value
X0	0 final value, uncertain before final value
X1	1 final value, uncertain before final value
S0	stable 0 value
S1	stable 1 value
P0	falling transition
P1	rising transition
XZ	from uncertain to high-impedance final value
SZ	stable high-impedance value

gorithm starts at the lowest level queue, dequeues each gate in the queue and simulates it. The algorithm then moves to the queue for the next topological level. If the simulated gate output changes, its fanouts are queued corresponding to their topological level. Gates are only queued once; they are flagged when queued. If the flag is set, the gate is not requeued, and when it is dequeued the flag is reset. A high-level view of this algorithm is presented in Figure 6.

We adopted a modified Smith's algebra [4] for our simulator. In order to correctly simulate uninitialized sequential circuits, we introduced an X value for representing uninitialized memory. Table 1 shows the simulation algebra. The two extra values, *XZ* and *SZ*, will be discussed in Section 6.

In a forward pass through the circuit, the path activation data obtained from the simulation is copied to the current time frame layer of the PSG. Each segment in the PSG is tagged as sensitized for a rising or a falling transition according to the status of the circuit line that the segment represents. In a backward pass, all paths that terminate at a primary output and have been tagged as sensitized, are also tagged as tested in the current time frame of the PSG. Using the method of Gharaybeh *et al.* [2], the PSG nodes are split according to this new testability information in order to preserve exactness.

All paths tagged as sensitized in the current time frame that terminate at flip-flops give rise to a pseudo stuck-at fault. One pass through the circuit flip-flops reveals which of them latched faults, and using the method described previously, a non-enumerative pseudo stuck-at fault is recorded.

3.2 Fault List Management

The second part of the non-enumerative sequential circuit simulation consists of determining which of the captured faults became observable at POs, which got propagated to other memory elements (flip-flops) and which should be eliminated. This stage also consists of event-driven simulation, but in this case, the bad-machine behavior is simulated separately. Each stored fault gives rise to one bad machine behavior (and must be simulated individually). Using machine word parallelism, we simulate 32 bad machines at a time.

Each active fault has a list of flip-flops at which its effect is present. When a fault is simulated, the value opposite to that of the good machine is placed in the flip-flop where the fault effect was. This is done for groups of 32 active faults.

Once a set of 32 faults is simulated, the effect of each fault is checked at all outputs and flip-flops. If the fault became observable at a PO, the cone of paths that gave rise to the fault (obtained from the fault's initial time-frame and the original path destination flip-flop) is tagged as tested in the present layer of the PSG and the fault is eliminated. If the fault only became observable at other flip-flops, the list of flip-flops on which the fault is present is updated with those. If the fault does not become observable at any output or flip-flop, it is eliminated. The algorithm that implements the above described process is shown in Figure 7.

A pictorial view of the layered PSGs during simulation is shown in Figure 8. As the picture shows, the result of the simulation is recorded on upward moving layers of the PSG. The bottom layer contains activated path information of the current time-frame and the higher ones contain activated paths of prior time frames. If an active fault is detected after a few time

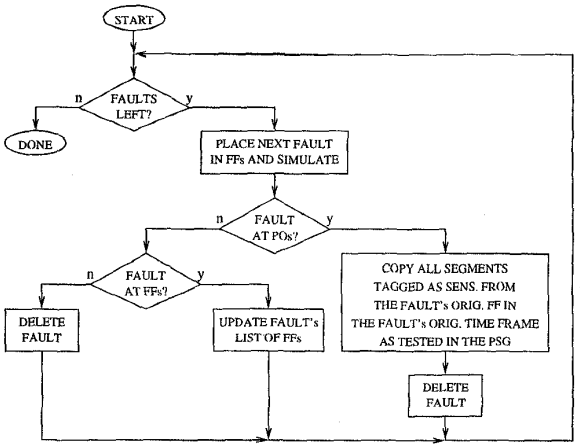

Figure 7: Fault propagation algorithm

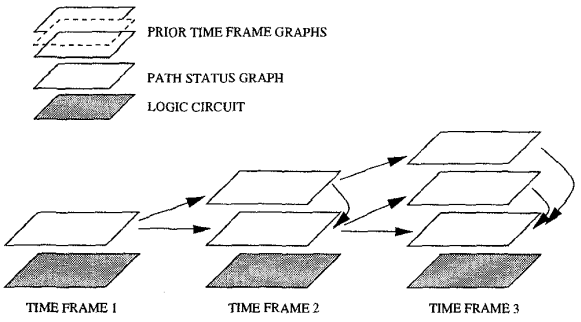

Figure 8: PSG management during simulation

frames, the cone of paths that gave rise to that fault is copied from a higher level to the bottom layer of the PSG (shown with the downward looping arrows on the right side of each time frame). This is because the bottom layer must maintain a record of all tested paths.

Before the simulation of the next time-frame is initiated, the current bottom layer produces a new bottom layer (consisting of all previously tested paths) and the next level layer consisting of paths activated but not detected in the current time-frame. All other layers move one level higher.

Since the number of prior time-frames stored in the graph is fixed, if a fault remains active for those time-frames without being detected, then it drops off the graph. If it were to become observable in a time-frame after being dropped, our simulator would give a lower than the actual fault coverage. The current implementation limits the time-frames to 32, assuming that only rarely will an activated path delay fault require more than 32 time-frames.

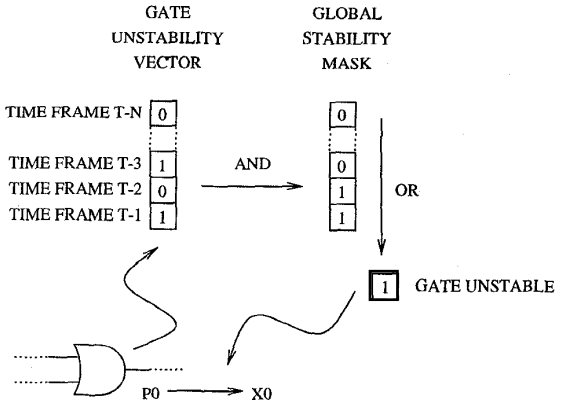

Figure 9: Rated clock instability flags and mechanism

4 Rated Clock Simulation

In the variable clock testing scheme, signals are applied to the circuit allowing enough time for them to stabilize (with the exception of the specific vector-pair test, that is applied at the rated speed) [5]. All signal changes, with the exception of the testing transition, have enough time to settle before the test is applied. For example, an *S0* signal value represents a steady (glitch free) low value on the line.

In the rated clock testing scheme, signals are applied to the circuit at the circuit's speed of operation; the stability assumption made for variable clock is no longer valid. For example, a signal that under a variable clock is *S0*, might in fact not be so due to a very late glitch from prior time frames.

Bose *et al.* [10] have proposed a method for simulating the rated clock situation by expanding the algebra to contain the information on prior time frames (as opposed to the duplet algebra used so far, which includes information about only two time frames). We instead assign a vector of flags in which we record the stability of each circuit signal for the prior N time frames (N is user-selectable). Figure 9 shows this scheme. If the signal has a non-steady value, the unstability flag is set for that signal in the present time frame. Later simulations of the said signal will require two steps: (1) calculate the variable clock signal value (optimistic), and (2) check if the signal's unstability flag is set in any of the prior N time-frames, and if so, set the signal to Xv, where v is the correct final value.

5 Asynchronous Logic Simulation

Asynchronous logic contains internal memory states that can change at arbitrary times. Structurally, the

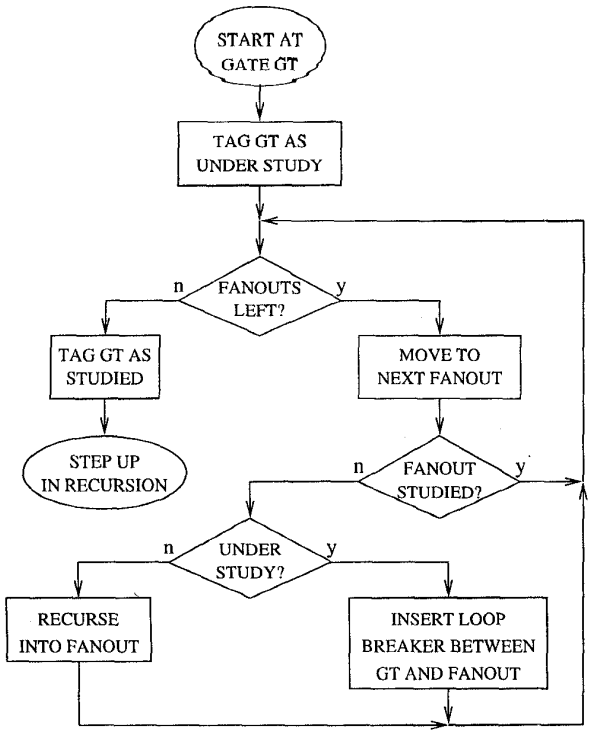

Figure 10: Loop breaking algorithm

circuit consists of combinational logic with feedback. A common method of simulation is to isolate feedbacks and simulate the resulting acyclic logic repeatedly until it becomes stable. The problem of path delay testing for asynchronous logic has not been addressed in the literature.

The main concerns with asynchronous logic are the correct initialization of the memory states and the detection of possible oscillations. To simulate these situations, we (1) break all of the circuit's asynchronous loops by inserting a loop-breaking element, and (2) modify the simulation algorithm to correctly simulate the loop-breaking elements. Delay faults are considered for all paths between primary inputs and primary outputs. A path cannot pass through a gate more than twice. Thus, in a path no loop is traced more than once.

5.1 Loop Breaking Algorithm

We break the combinational logic loops by identifying the back edges in the combinational circuit graph. These edges are replaced with fictitious loop-breaking gates (see Figure 10).

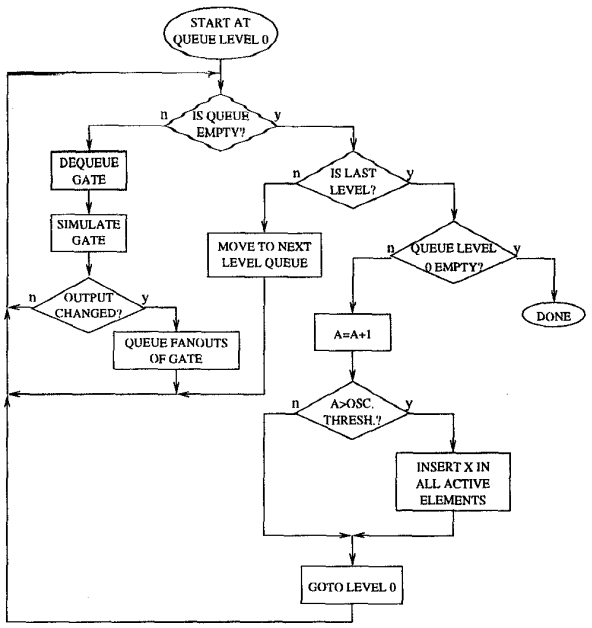

Figure 11: Asynchronous logic simulation algorithm

5.2 Simulation Algorithm

Loop-breaking elements are treated as special wires. The signal on their output will mimic the signal on their input. Nevertheless, in order to detect oscillations, the loop-breaking elements are allowed to change value only a prespecified number of times during the simulation of a single vector-pair. If in the course of simulation of a single stimulus, the loop-breaking gate changes value for a larger number of times, an oscillation is assumed to have occurred and the output of the loop-breaking gate is set to XX. This algorithm is shown in Figure 11.

6 Tri-State Logic Simulation

Tri-state signals are generated by buffer-type gates, and resolved by bus-type gates into the standard three values (0, 1 and X) that all other gate types use. The high-impedance state, Z, can therefore only exist between buffers and buses, thus simplifying the simulation truth tables.

6.1 Extended Algebra

It was found necessary to extend our algebra to correctly perform tri-state logic simulation. More specifically, Z-values (high impedance) had to be added. Following the logic of the duplet specified by Smith (path-activation status, final value), three more values seem

necessary: XZ, SZ, and PZ. Nevertheless, the PZ value has no meaning since a line going to a high-impedance state cannot be considered to have a transition (the bus' final value will depend on other signals, yielding no transition at all). As a result, only two values need to be added. The complete algebra is shown in Table 1.

7 Path Selection and User Interface

We now discuss user interaction, namely, how we report the obtained data, and how the user can steer the course of the simulation according to needs.

7.1 Path Histogram Generation Algorithm

Our program should provide the user with detailed information on the path delay fault analysis of the circuit under simulation. In order to meet this need, we devised an algorithm that could produce (in a histogram) the distribution of paths according to their length. This information is calculated for all of the circuit's topological paths and all tested paths. Since the histogram information is obtained from the PSG, and the PSG can be large, the algorithm is designed to be memory efficient.

Our histogram generation algorithm has a linear complexity in the number of graph edges and requires us to maintain an active set of arrays. The number of these arrays is only a fraction of the number of graph nodes. The algorithm traverses the graph in a depth-first manner, creating a pseudo-histogram for each node. Each time a node is reached, all histograms of that node's fanouts are added to the node, offset by the weight of the edge that joins them. Once a histogram has been accessed as many times as the node input edge count, the histogram is eliminated (reducing memory considerably). Figure 12 shows a histogram obtained with this algorithm after the simulation of 2,549 vectors applied to circuit s5378. These vectors were obtained from 25,000 randomly generated vectors using a test generation option in the simulator, which drops vectors that do not activate or detect any new fault. The total PDF fault coverage is 21.24% and the longest paths tested are 24 gates long.

7.2 Path Targeting

Even though the non-enumerative technique can effectively obtain the path delay fault information for the complete circuit, it is sometimes of interest to designers to obtain the fault coverage of a selected subset of paths (the critical timing paths, for example) instead of the complete circuit delay-fault coverage. Our simulator can accept the target path data supplied by the

LNGTH	FCOV	TESTED	TOTAL
2	22.1 %	19	86
3	25.0 %	2	8
4	15.0 %	6	40
5	32.1 %	27	84
6	25.6 %	43	168
7	1.9 %	3	154
8	16.6 %	50	302
9	18.1 %	95	526
10	25.6 %	259	1012
11	22.7 %	373	1646
12	14.6 %	284	1946
13	6.0 %	149	2464
14	11.7 %	313	2678
15	10.9 %	248	2284
16	29.3 %	580	1982
17	48.4 %	1181	2438
18	43.1 %	884	2052
19	36.7 %	941	2564
20	43.8 %	617	1410
21	27.7 %	385	1392
22	31.5 %	214	680
23	36.2 %	145	400
24	18.2 %	70	384
25	0.0 %	0	288
26	0.0 %	0	96

PATH DELAY-FAULT HISTOGRAM PLOT

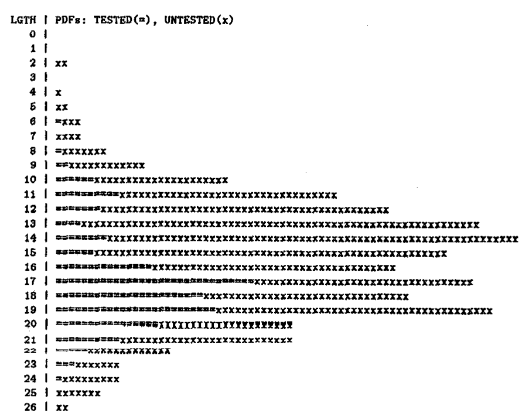

Figure 12: Path-delay fault histogram example (s5378)

Table 2: Non-robust variable clock simulation results

Circuit Name →	s5378	adder[1]	s35932	s38584	s38417	bfa0012af[2]	mult[1]	bsm[3]	multadder[2]
PIs	35	66	36	13	29	61	64	34	65
POs	49	32	320	278	106	64	32	34	33
Gates	2779	734	16065	19253	22179	5480	7538	18823	5134
FFs	179	147	1728	1452	1636	544	240	1368	0
Vectors	2549	3863	2124	3335	2000	3000	3481	453195	190
Total PDFs	27084	53208	394282	2.2×10^6	2.8×10^6	1.5×10^8	1.6×10^8	4.0×10^8	2.4×10^{12}
Tested PDFs	5753	4888	26228	5266	1106	116494	544931	164578	1330876
All PDF Cov. (%)	21.24	9.19	6.65	0.24	0.04	0.08	0.35	0.04	0.00
Longest PDF (gates)	26	29	29	56	47	52	50	74	107
Long. Tested PDF	24	24	27	26	23	51	48	58	72
Line Coverage (%)	53.22	81.44	67.35	19.80	2.02	90.48	58.35	76.34	99.86
Time[4]	1m33s	2m32s	1h46m45s	1h5m28s	21m26s	3h37m27s	17h52m45s	67h49m1s	32m49s
Memory (MB)	1.4	1.0	5.5	5.7	4.4	28.1	141.3	13.8	208.5
β (PSG size factor)	2.0	3.6	2.2	1.7	1.5	49	177	4.5	337

user. Alternatively, the user can invoke any of the automatic options in the program to select, (a) all paths, (b) longest paths between each combinational input-output pair, (c) a set containing the longest path through each line, (d) a given number (or percentage) of the longest paths in the circuit, or (e) all paths within a given range of lengths.

We devised two methods for simulating a selected set of paths. In the first method, we simulate all paths non-enumeratively. Then we examine the final PSG for each targeted path. If a path is tagged in the PSG, it is considered as tested.

The second technique consists of restricting the simulation to the given subset of paths. The target set of paths is tagged onto the PSG prior to the simulation. During the simulation run, only the testability of these tagged edges is recorded. This approach has the advantage that the simulation time can be reduced considerably when a small set of paths is targeted. This is because the dynamic size of the PSG will remain small. However, the disadvantage is that the information about any untargeted path, if needed, will require the simulation to be repeated.

[1]Circuits obtained from Rutgers University Center for Computer Aids for Industrial Productivity

[2]Circuits obtained from Stanford University Center for Reliable Computing

[3]A circuit designed at Lucent Technologies

[4]Dual-Processor Sun UltraSparc

8 Results

We present the results of the simulation of large circuits in Table 2. The circuits are arranged in the increasing order of the total PDFs. For some circuits (bfa0012af, mult, bsm and multadder), it is the first time that we were able to simulate completely because other enumerative techniques available to us could not track those large number of *activated* paths (order of 10^5 to 10^6). That is the reason why we are unable to give a comparison with another program. Notice that an enumerative simulator tracks only the combinationally activated paths. However, these paths cannot be dropped from consideration until the fault effect is propagated to a primary output. Unlike a combinational circuit, the activated paths in a sequential circuit tend to accumulate over many time-frames.

The vectors used for the bsm circuit were scan sequences. The simulator contains a special efficiency feature that recognizes the fault detection via scan-out. Vectors for all other circuits were randomly generated. Except for s38417, all other vector sets were compacted by removing some vectors that did not detect any additional fault.

All paths were simulated in the non-robust mode. The PDF coverage is low in all cases, due to the fact that the vectors did not specifically target delay faults. However, for circuits, s5378, s35932, bfa0012af and mult,

the longest tested path is within two gates of the longest physical path. For s5378, this situation is also depicted in the histogram of Figure 12.

Line Coverage in Table 2 is the percentage of lines that belong to at least one tested path. Interestingly, for the multadder circuit, we see a 99.9% line coverage in spite of a very low path coverage. Questions, why the path coverage is low or why the longest tested path (72 gates) is much too short compared to the longest physical path (107 gates), are beyond the scope of a fault simulator. One would require a delay test generator or possibly a false path identifier program to answer these questions.

Given the large number of paths, the CPU time and memory usage are quite reasonable. The time for the bsm circuit is long due to the long scan sequence that was simulated. We also notice that the multadder circuit, that is combinational, has poorer simulation performance than was reported by Gharaybeh *et al.* for other combinational circuits [2]. There are several reasons. First, the present implementation is for sequential circuits and is not optimized for the simpler combinational case. Second, the previous results were for single input change vectors and the simulator was specifically programmed for that type of vectors. The present simulator does not impose any such restriction on vectors. We are looking into optimizations of the program. We have also considered the possibility of using multiprocessing. Those investigations are not yet complete.

During the simulation, the size of the path status graph (PSG) dynamically changes due the splitting and recombination of nodes [2]. The minimum size occurs when either all paths are untested or all are tested. The last line in Table 2 is the PSG size factor, β. It is the ratio of nodes in the largest PSG during simulation to that in the initial PSG. The original PSG only contains nodes for PIs, POs and gates with fanout greater than one [2]. Thus, the original PSG is usually smaller than the circuit connectivity graph. For example, the number of nodes in the initial PSG of s5378 is 997, a number smaller than PI + PO + gates = 2863. The PSG size factor (β) is an indicator of the amount of memory used by the program. For example, largest memory usage was for multadder (208.5MB) for which $\beta = 337$ also attains the largest value.

9 Conclusion

We have presented a new, non-enumerative path delay fault simulator that accepts circuit descriptions and libraries that are fully compatible with Bell Labs' *Gentest* system [3]. We can now obtain the path delay fault behavior of circuits too large to be analyzed with any prior method. We also include the features necessary for simulating production-level circuits for path-delay fault analysis. Although the simulator has the capability of simulating all paths, it may be more practical to carefully select paths for testing. The simulator provides several options for automatic path selection. Path delay simulation for asynchronous circuits is new and experience is needed to validate the assumptions made here. The program can also benefit from multiprocessing.

In a typical scenario, the non-enumerative fault simulator may be required to simulate the given set of vectors, without a prior path selection. The post-simulation PSG can then provide data on the longest physical path, the longest tested path, and the coverage of lines and paths of any specified characteristics. Future experience will allow us to develop additional scenarios.

Although the program is not intended for the purpose of false path identification, it can be used for it. In some cases, when the complete input space (input vectors) is known, such as BIST, by simulating using functional sensitization algebra, the simulator can be used to find the longest true timing path or set of paths. Once this information is known, BIST can be executed at the speed of the longest path activated, as opposed to a slower, more conservative speed, giving a better possibility of detecting path-delay faults. Also, if the longest BIST-tested path is found to be longer than the critical path, then the BIST test frequency should be suitably adjusted to avoid any yield loss. Functional sensitization must be used for this procedure because the other algebras only report singly-testable paths, and multiply-testable paths affect the timing of the circuit.

Acknowledgment

We thank R. Bencivenga, P. Murphy and R. Srinivasan for their help with *Gentest*, and T.J. Chakraborty and N. Mukherjee for discussions on the previous delay fault simulator. We thank M.A. Gharaybeh for discussions on the PSG and a version of his program.

References

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

[2] M. Gharaybeh, M. L. Bushnell, and V. D. Agrawal, "An Exact Non-Enumerative Fault Simulator for Path-Delay Faults," in *Proc. Int'l. Test Conf.*, pp. 276–285, 1996.

[3] R. Bencivenga, T. J. Chakraborty, and S. Davidson, "Gentest: the Architecture of Sequential Circuit Test Generator," in *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 17.1.1–17.1.4, 1991.

[4] G. L. Smith, "Model for Delay Faults Based Upon Paths," in *Proc. Int'l. Test Conf.*, pp. 342–348, 1985.

[5] T. J. Chakraborty, V. D. Agrawal, and M. L. Bushnell, "On Variable Clock Methods for Path Delay Testing of Sequential Circuits," *IEEE Trans. CAD*, vol. 16, pp. 1237–1249, November 1997.

[6] S. Bose, P. Agrawal, and V. D. Agrawal, "Path Delay Fault Simulation of Sequential Circuits," *IEEE Trans. on VLSI systems*, vol. 1, pp. 453–461, December 1993.

[7] I. Pomeranz and S. M. Reddy, "An Efficient Non-Enumerative Method to Estimate the Path Delay Fault Coverage in Combinational Circuits," *IEEE Trans. on CAD*, vol. 13, pp. 240–250, February 1994.

[8] K. Heragu, V. D. Agrawal, M. L. Bushnell, and J. H. Patel, "Improving a Nonenumerative Method to Estimate Path Delay Fault Coverage," *IEEE Trans. on CAD*, vol. 16, pp. 759–762, July 1997.

[9] Y. Hsu and S. K. Gupta, "A Simulator for At-Speed Robust Testing of Path Delay Faults in Combinational Circuits," *IEEE Trans. on Computers*, vol. 45, pp. 1312–1318, November 1996.

[10] S. Bose, P. Agrawal, and V. D. Agrawal, "A Rated-Clock Test Method for Path Delay Faults," *IEEE Trans. on VLSI Systems*, vol. 6, pp. 323–331, June 1998.

[11] S. Majumder, V. D. Agrawal, and M. L. Bushnell, "Path Delay Testing: Variable-Clock Versus Rated-Clock," in *Proc. 11th Int'l. Conf. on VLSI Design*, pp. 470–475, January 1998.

[12] T. J. Chakraborty, V. D. Agrawal, and M. L. Bushnell, "Path Delay Fault Simulation Algorithms for Sequential Circuits," in *Proc. Asian Test Symp.*, pp. 52–56, November 1992.

[13] I. Pomeranz and S. M. Reddy, "SPADE-ACE: A Simulator for Path Delay Faults in Sequential Circuits with Extensions to Arbitrary Clocking Schemes," *IEEE Trans. on CAD*, vol. 13, pp. 251–263, February 1994.