Path Hashing to Accelerate Delay Fault Simulation

Manfred Henftling Hannes C. Wittmann Kurt J. Antreich
Institute of Electronic Design Automation, Department of Electrical Engineering
Technical University of Munich, 80290 Munich, Germany

Abstract

This paper presents an efficient approach to path delay fault simulation. We accelerate fault simulation by more than one order of magnitude with a new speed up technique called path hashing. An intelligent path identification method allows to deal with circuits containing two orders of magnitude more paths than state-of-the-art tools. Using these techniques larger circuits can be handled with a reasonable amount of time and memory.

1 Introduction

To ensure the correct behavior of an integrated circuit, not only its static, but also its dynamic, operation has to be guaranteed. Physical defects, as well as parameter variations during the manufacturing process may affect the dynamic behavior of a circuit. The dynamic characteristics of a module under test are examined with the help of delay testing. It is a key design task to verify correct dynamic operation for given gate delay tolerance ranges.

Two fault models have been proposed to represent delay faults. The *gate delay fault model* [1] models at one gate a delay fault that causes violations of the circuit specification. A weakness of the gate delay fault model is the assumption that process variations affect only individual gates. In practice, fluctuations during the manufacturing process influence entire parts of a chip or wafer. Therefore, the *path delay fault model* [2] has been introduced. It assumes delay faults on entire paths in a circuit. The major worry when using this fault model is the size of its fault dictionary, that may grow exponentially with the circuit depth. In this paper, we consider the path delay fault model and show how to deal with huge path sets.

Fault simulation is an important research topic, because its application is manifold. Smith [2] addressed the problem of path delay fault simulation first. He introduced a six-valued logic tailored for the robust detection of path delay faults. Fink et al. [3] developed an accelerated delay fault simulator that exploits the concept of parallel processing of patterns [4, 5] at all stages of the simulation. Furthermore, they extended the approach to non-robust detection of path delay faults and they proposed a data structure to store the tested paths. Based on these two approaches various fault simulators for combinational [6, 7] and sequential [8, 9, 10, 11] circuits have been proposed.

After the basic definitions in Section 2, the algorithm for path delay fault simulation is explained in Section 3. An improved encoding and the path hashing are presented in Sections 4 and 5. The experimental results shown in Section 6 illustrate the improvements. Section 7 concludes the paper.

2 Basic Definitions

A combinational circuit C can be represented by a directed, acyclic graph $G = (V, E)$ with nodes V and edges E. V denotes the set of signals in C. A directed edge $(x, y) \in E$ of G means that x is an input signal and y is an output signal of a distinct gate. It is useful for further considerations to define the set of primary inputs PI and the set of primary outputs PO.

A structural path P_s in G is defined as an (n+1)-tuple $P_s = (x_0, \ldots, x_n)$, whereby the nodes on the path are the components of the tuple. A path starts at a primary input and ends at a primary output. The set of all structural paths in C is called PS. By choosing the transitions (either rising (r) or falling (f)) at the primary input and at all outputs of XOR/XNOR gates on a structural path P_s, a functional path $P_f = ((x_0, \ldots, x_n), (t_0, \ldots, t_m))$ is defined. t_0 is related to the transition at the primary input and t_i ($1 \leq i \leq m$) is related to the ith of the m XOR/XNOR gates on the path counted from the primary input. PF denotes the set of all functional paths in a circuit.

Smith [2] has introduced the hardware model for delay testing. The combinational circuit under test is embedded between a block of input latches and a block of output latches. All latches are assumed to be glitchless. At time T_1 the first vector V_1 is loaded into the input latches. After all signals in the circuit have taken stable values, the second vector V_2 is applied at time T_2. The logic values of the primary outputs are sampled into the output latches at time $T_S = T_2 + T_C$, where T_C is the desired clock rate.

A path P_f has a nominal delay $d(P_f)$ and may have a delay fault called $\Delta(P_f)$. P_f is said to be faulty if the delay fault causes the wrong value of its primary output at time T_C, e.g. $d(P_f) + \Delta(P_f) > T_C$. A path delay fault is called robust detected, if and only if, its detection is independent of all other delay faults in the circuit. Otherwise it is nonrobust detected. Of course, robust detection implies nonrobust detection.

Considering Path Identification an efficient method has been proposed for structural paths [10] and has been extended to functional paths [12]. Its main idea is to avoid storing a set of nodes to represent a path by using a unique number, the Path Identifier (PID), instead. For every gate input a branch identifier is calculated in a preprocessing step. By traversing a path and summing up all branch identifiers on a path, the unique PID is found. Detailed information about this identification method can be found in [12].

3 Path Delay Fault Simulation

The improvements and new techniques described in this paper are based on the robust path delay fault simulation of Smith [2] and its extension to nonrobust simulation introduced by Schulz et al. [6].

Smith proposed a six valued simulation logic with the values $0s, 0p, 0-, 1s, 1p$, and $1-$. Each value consists of the final boolean value $\{0, 1\}$ and the detectability status $\{s, p, -\}$. A detectability status s indicates that a signal remains stable at its final value, p shows that there is at least one path from a primary input that is path delay fault testable, and $-$ indicates that the detectability status of a signal is neither s nor p. The value propagation table for an OR gate and an inverter are shown in Table 1. Analogous propagation tables can be derived for AND and XOR gates [2, 6].

$\vee$	$0s$	$1s$	$0p$	$1p$	$0-$	$1-$
$0s$	$0s$	$1s$	$0p$	$1p$	$0-$	$1-$
$1s$	$1s$	$1s$	$1s$	$1s$	$1s$	$1s$
$0p$	$0p$	$1s$	$0p$	$1-$	$0p$	$1-$
$1p$	$1p$	$1s$	$1-$	$1-$	$1-$	$1-$
$0-$	$0-$	$1s$	$0p$	$1-$	$0-$	$1-$
$1-$	$1-$	$1s$	$1-$	$1-$	$1-$	$1-$

$\neg$	
$0s$	$1s$
$1s$	$0s$
$0p$	$1p$
$1p$	$0p$
$0-$	$1-$
$1-$	$0-$

Table 1: Robust propagation table for OR- and NOT-gates

Schulz et al. [6] showed that it is possible to collapse the six values to four values, if only nonrobust fault detection is required. The detectability stati s and $-$ are merged to the new status $\overline{p}$. The corresponding tables for propagating these values through an OR gate or an inverter are given in Table 2.

Based on the propagation tables, the path delay fault simulation can be performed. It consists of four major steps.

First, the primary inputs must be initialized. The second

$\vee$	$0p$	$1p$	$0\overline{p}$	$1\overline{p}$
$0p$	$0p$	$1p$	$0p$	$1\overline{p}$
$1p$	$1p$	$1\overline{p}$	$1p$	$1\overline{p}$
$0\overline{p}$	$0p$	$1p$	$0\overline{p}$	$1\overline{p}$
$1\overline{p}$	$1\overline{p}$	$1\overline{p}$	$1\overline{p}$	$1\overline{p}$

$\neg$	
$0p$	$1p$
$1p$	$0p$
$0\overline{p}$	$1\overline{p}$
$1\overline{p}$	$0\overline{p}$

Table 2: Nonrobust propagation table for OR- and NOT-gates

test vector V_2 determines the final value; and the first test vector V_1 the detectability status of an input signal. It is either p if initial and final value are different, or s(or $\overline{p}$, respectively) if initial and final value are identical in the robust (or nonrobust, respectively) case.

Second, the true value simulation of the circuit is performed from the primary inputs to the primary outputs. The values of all signals in the circuit are computed according to the propagation tables.

Third, all robust (nonrobust) detected path delay faults are determined. The paths where all nodes on the path show a detectability status p are searched in a depth first manner from the primary outputs to the primary inputs. Of course, the search starts only at primary outputs with detectability status p.

Last, the tested paths have to be stored. The path tree introduced in [6] has turned out to be time and space expensive for large path sets. We use the method of [12] to guarantee efficient path handling.

4 Encoding of Logic Values

In order to guarantee efficient propagation of logic values introduced in Section 3, the calculation of the output value of a gate with given input values has to be executed as fast as possible. Each input value can be encoded in 3 bits as there are six logic values. We considered two ways of managing the calculation:

- All three bits are integrated in a single word. To determine the output value of a gate, a table-look-up method can be used. This method has the advantage that the implementation is simple.

- The three bits are in three words. To determine the output value of a gate, the words have to be combined by logic equations. This approach enables treatment of several patterns in parallel, because only one bit of a word is needed per pattern and the equations depend on the gate and not on the pattern.

We decided to use the second approach as it turned out to be faster [3, 5]. By using an intelligent encoding, it minimizes the number of boolean operations needed to compute the output of a gate. Table 3 shows the encoding that we have chosen to represent the logic values of a signal x; and Table 4 displays the resulting minimized logic equations for an arbitrary gate with the output y and two inputs a and b.

	x_0	x_1	x_2
$0s$	0	0	1
$1s$	1	0	1
$0p$	0	1	0
$1p$	1	1	0
$0-$	0	1	1
$1-$	1	1	1

Table 3: Encoding of the logic values

AND	$y_0 = a_0 \wedge b_0$
	$y_1 = ((a_0 \vee a_1) \wedge b_1) \vee (a_1 \wedge b_0)$
	$y_2 = (\neg a_0 \wedge b_1) \vee (a_1 \wedge \neg b_0) \vee (a_2 \wedge b_2)$
OR	$y_0 = a_0 \vee b_0$
	$y_1 = ((\neg a_0 \vee a_1) \wedge b_1) \vee (a_1 \wedge \neg b_0)$
	$y_2 = (a_0 \wedge b_1) \vee (a_1 \wedge b_0) \vee (a_2 \wedge b_2)$
XOR	$y_0 = a_0 \oplus b_0$
	$y_1 = a_1 \vee b_1$
	$y_2 = (a_1 \wedge b_1) \vee (a_2 \wedge b_2)$
NOT	$y_0 = \neg a_0$
	$y_1 = a_1$
	$y_2 = a_2$

Table 4: Logic equations

To get an impression of the quality of our logic equations we compare them with those of the fault simulator [3] included in DYNAMITE [13]. Table 5 displays the frequency

gate type	fequency in benchmarks	operations per gate	
		DYNAMITE	TIP
AND	22.9 %	23	12
NAND	14.1 %	24	13
OR	9.0 %	25	12
NOR	6.3 %	26	13
BUF	0.9 %	0	0
NOT	46.7 %	1	1
XOR	0.1 %	5	5
XNOR	0.0 %	6	6
av. operations		13.01	6.95

Table 5: Number of boolean operations

of all gates in the benchmark circuits [14, 15, 16], the number of boolean operations needed by DYNAMITE, and the number of boolean operations needed by our test preparation tool TIP. The bottom line compares the weighted average number of operations required by both tools. It shows that we can save about 50 % of the boolean operations.

5 Path Hashing

Practice shows that in most cases a single pattern is able to detect faults on multiple paths and that many detected paths have common parts. By exploiting this fact it is possible to save computation time during path delay fault detection.

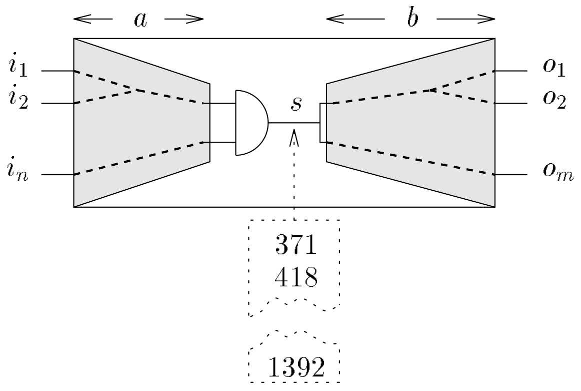

Figure 1: Idea of path hashing

To demonstrate the algorithm, Figure 1 shows a circuit after the simulation of a six-(four-)-valued test pattern. Any path represented by a dashed line from an input i_x via the signal s to the outputs o_y is detected by the pattern. Recalling the algorithm explained in Section 3, it is necessary to traverse every path from s to the inputs, if a path from the outputs to s is found. Assuming that there are a paths from the inputs to s and there are b paths from s to the outputs, it is necessary to handle all $a * b$ paths. Remember that every path is identified by a number that results from the sum of the branch identifiers at every signal. Hence, the sums for the a paths segments are calculated b times. To avoid this repetition we propose to modify the path detection algorithm as follows:

1. After performing the simulation for a pattern, an empty list is set up at all fanouts.
2. Every sum of branch identifiers of a path segment from a fanout to an input whose corresponding path delay fault is detected by the pattern is inserted into the list of the fanout.
3. A path segment from an output to a fanout with a nonempty list is identified. The identifiers of all paths containing this segment can be determined by adding the sum of branch identifiers of the segment to all items of the list of the fanout.

Using this extension PIDs are computed only $a+b$ times, instead of $(a+1)*b$ times. If we assume reasonable values of $a = 30$ and $b = 20$, the computation is $(30+1)*20/(30+20) = 12.4$ times as efficient.

Example

To illustrate the algorithm we have prepared the example presented in Figure 2 and Figure 3. To perform robust fault simulation, the four major steps of Section 3 have to be executed.

In the first two steps we propagate the test pattern $T = (a, b, c, d) = (1p, 0p, 0p, 1p)$ and we get the logic values

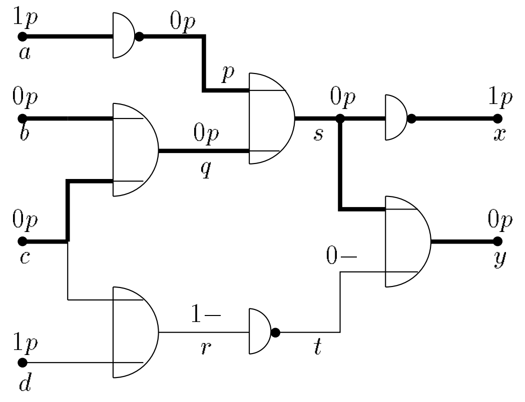

Figure 2: True value simulation

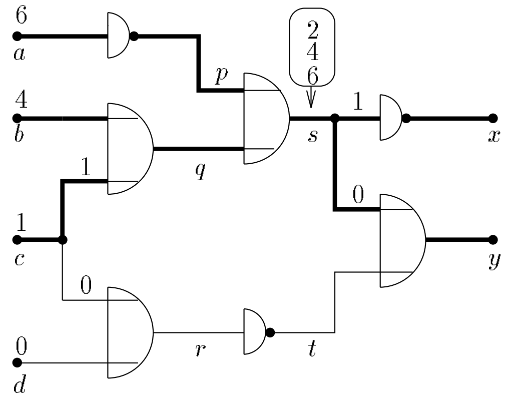

Figure 3: Path identification and path hashing

that are presented in Figure 2. Path delay faults that occur on paths that are drawn in bold are detected by the test pattern. Figure 3 shows the branch identifiers. In the third step all faults are identified. Starting at the output x with the sum 0 we branch to signal s and add the branch identifier 1 to the sum. Since s is a fanout we store the sum. Then we branch to a via p and add the branch identifier 6 to the sum. Since a is an input the fault number 7 can be detected by this pattern. Now we mark this fault as detected and get the first element of the list at signal s. The value of the element is the sum of all branch identifiers on the path between the fanout and the input. In the example we subtract the stored value 1 from the total sum of the path 7. Hence, we get 6 as the first element. Continuing with the signal q we get the entry 4 at the input b and the entry 2 at c. Now all faults that can be detected on paths starting at output x are identified and we have to derive the numbers of the paths beginning at the other output y. The sum is cleared; we step from y to s and add the branch identifier 0 to the sum. Here we find a not empty list. Hence we get all path identifiers without further analysis. The faults with the identifiers $0+2$, $0+4$ and $0+6$ are detected, too. We can return to signal y.

Using this procedure we are able to obtain all identifiers of paths that are detected by a pattern handling each signal only once.

6 Experimental Results

The path delay fault simulator was implemented in "C", integrated in our test preparation tool TIP, and tested with the help of well-known benchmark circuits [14, 15, 16]. When sequential circuits are processed, only the combinational part is considered. Several experiments were performed to get an impression of the improvements of the new techniques. As a reference the fault simulator [3] and path tree of DYNAMITE were chosen. All experimental results we present were measured on a DECstation 3000/500.

In our main experiment 10,000 randomly generated test patterns have been robust and nonrobust simulated. The results are summarized in Table 6.

Circuit Name	robust tested		nonrobust tested		total time
	paths	time	paths	time	
s1512	1508	0.68 s	2569	1.73 s	2.41 s
s3330	4103	1.87 s	7733	4.29 s	6.16 s
s1269	1303	0.47 s	20414	4.76 s	5.23 s
s9234	2998	4.00 s	23423	15.99 s	19.99 s
c499	12	0.11 s	121264	3.96 s	4.07 s
c1908	704	0.42 s	40703	4.61 s	5.03 s
s38584	30645	30.62 s	87417	70.56 s	101.18 s
s13207.1	4936	8.76 s	49533	21.73 s	30.49 s
s15850.1	8108	11.84 s	277541	64.75 s	76.59 s
s4863	1748	1.98 s	1220832	101.10 s	103.08 s
s6669[1]	6142	34.24 s	—	—	34.25 s
c6288[1]	70	0.98 s	—	—	0.98 s

Table 6: Robust and nonrobust path delay fault simulation

Table 6 shows that we are able to simulate large test pattern sequences in a reasonable amount of time. The second and third columns of Table 6 represent the number of robust tested paths and the CPU-time required. Columns four and five contain the same information for nonrobust simulation. The last column displays the total time of robust and nonrobust fault simulation. The largest circuit we are able to handle is s4863[1] with 2.6 billion paths. It takes about one hundred seconds to simulate the 10,000 patterns. The number of all functional paths is given in Table 8.

Next, we compared our simulation approach to the fault simulation included in DYNAMITE and computed the speed up we got. The results are given in Table 7. The second and third columns of Table 7 contain the simulation times of TIP and DYNAMITE. The last column, the speed up, shows that we are on the average ten times faster than DYNAMITE, whereby the greatest speed up is 49 for circuit s38584. Of course, for the two largest circuits we can give no speed up, because the path tree cannot handle circuits of this size.

With our final experiment we compared the overall simulation time (simulation plus preprocessing time). We have collected these times in Table 8. They show impressively that the overall simulation time of TIP is dominated by the

[1]results for s6669 and c6288 are obtained by storing the PID in a fault list

Circuit	TIP	DYN.	Speed up
s1512	2.41 s	20.33 s	8.4
s3330	6.16 s	43.37 s	7.0
s1269	5.23 s	25.13 s	4.8
s9234	19.99 s	113.52 s	5.7
c499	4.07 s	17.14 s	4.2
c1908	5.03 s	16.09 s	3.2
s38584	101.18 s	4963.97 s	49.1
s13207.1	30.49 s	434.02 s	14.2

Table 7: Simulation times

Circuit	all paths	TIP	DYN.
s1512	6,936	2.42 s	21.24 s
s3330	9,530	6.22 s	44.80 s
s1269	79,138	5.24 s	33.17 s
s9234	489,708	20.06 s	190.01 s
c499	795,776	4.08 s	76.89 s
c1908	1,458,114	5.05 s	107.93 s
s38584	2,161,138	102.23 s	10568.07 s
s13207.1	2,690,738	30.99 s	1395.12 s
s15850.1	329,476,064	77.73 s	—
s4863	2,636,114,122	107.27 s	—
s6669[1]	431,685,738,673,270	34.31 s	—
c6288[1]	$1.97886883477 \cdot 10^{20}$	1.06 s	—

Table 8: Overall simulation time

simulation time, while the preprocessing time of DYNAMITE is significant compared to simulation time. Hence, in view of the user-time, TIP clearly performs better.

7 Conclusion

In this paper, we presented a new encoding for the six-valued logic and path hashing. Experimental results show that both techniques impressively accelerate the robust and nonrobust fault simulation for path delay faults. It was demonstrated that it is possible to perform a fault simulation for path delay faults for circuits with up to three billion paths.

Our future work in this area is to further optimize the simulation algorithm. Moreover, we want to extend it so that circuits with single scan path and sequential circuits can be processed.

References

[1] Z. Barzilai and B. K. Rosen. "Comparison of AC Self–Testing Procedures". In *Proceedings IEEE International Test Conference*, pages 89–94, October 1983.

[2] G. L. Smith. "Model for Delay Faults Based Upon Paths". *Proceedings IEEE International Test Conference*, pages 342–349, September 1985.

[3] F. Fink, K. Fuchs, and M. H. Schulz. "Robust and Nonrobust Path Delay Fault Simulation by Parallel Processing of Patterns". *IEEE Transactions on Computers*, pages 1527–1536, December 1992.

[4] J. A. Waicukauski, E. B. Eichelberger, D. O. Forlenza, E. Lindbloom, and T. McCarthy. "Fault Simulation for Structured VLSI". *VLSI Systems Design*, pages 20–32, December 1985.

[5] Kurt J. Antreich and Michael H. Schulz. "Accelerated Fault Simulation and Fault Grading in Combinational Circuits". *IEEE Transactions on Computer–Aided Design*, pages 704–712, September 1987.

[6] M. H. Schulz, F. Fink, and K. Fuchs. "Parallel Pattern Fault Simulation of Path Delay Faults". *Proceedings ACM/IEEE Design Automation Conference*, pages 357–363, June 1989.

[7] S. Koeppe. "Modeling and Simulation of Delay Faults in CMOS Logic Circuits". In *Proceedings IEEE International Test Conference*, pages 530–536, September 1986.

[8] Irith Pomeranz and Sudhakar M. Reddy. "An Efficient Non–Enumerative Method to Estimate Path Delay Fault Coverage". In *Proceedings IEEE/ACM International Conference on Computer–Aided Design*, pages 560–567, November 1992.

[9] Pomeranz Irith and Sudhakar M. Reddy. "An Efficient Non-Enumerative Method to Estimate Path Delay Fault Coverage". In *IEEE/ACM International Conference on CAD*, pages 560–567, 1992.

[10] Irith Pomeranz, Lakshmi N. Reddy, and Sudhakar M. Reddy. "SPADES: A Simulator for Path Delay Faults in Sequential Circuits". In *Proceedings of the European Conference on Design Automation*, pages 428–435, September 1992.

[11] S. Bose, P. Agrawal, and V. D. Agrawal. "A Path Delay Fault Simulator for Sequential Circuits". In *Sixth International Conference on VLSI Design*, pages 269–274, January 1993.

[12] Hannes C. Wittmann and Manfred Henftling. "Efficient Path Identification for Delay Testing — Time and Space Optimization". In *Proceedings European Test Conference*, February 1994. in press.

[13] Karl Fuchs, Franz Fink, and Michael H. Schulz. "DYNAMITE: An Efficient Automatic Test Pattern Generation System for Path Delay Faults". *IEEE Transactions on Computer–Aided Design*, pages 1323–1335, October 1991. volume = Vol. CAD–10, number = No. 10.

[14] Franc Brglez and Hideo Fujiwara. "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran". In *IEEE International Symposium on Circuits and Systems; Special Session S6AB on ATPG and Fault Simulation*, pages 663–698, June 1985.

[15] Franc Brglez, David Bryan, and Krzysztof Kozminski. "Combinational Profiles of Sequential Benchmark Circuits". In *Proceedings IEEE International Symposium on Circuits and Systems*, pages 1929–1934, May 1989.

[16] Franc Brglez. "ACM/SIGDA Benchmark Electronic Newsletter DAC'93 Edition, June 1993.