An Efficient Path Delay Fault Coverage Estimator

Keerthi Heragu
Dept. of Electrical & Computer Eng.
Rutgers University
Piscataway, NJ 08855-0909
heragu@caip.rutgers.edu

Michael L. Bushnell
Dept. of Electrical & Computer Eng.
Rutgers University
Piscataway, NJ 08855-0909
bushnell@caip.rutgers.edu

Vishwani D. Agrawal
AT&T Bell Labs
600 Mountain Avenue
Murray Hill, NJ 07974
va@research.att.com

Abstract—We propose a linear complexity method to estimate robust path delay fault coverage in digital circuits. We adopt a path counting scheme for a true-value simulator that uses flags for each signal line. These flags determine the new path delay faults detected by the simulated vector pair. Experimental results are presented to show the effectiveness of the method in estimating path delay fault coverage.

I. Introduction

Two commonly used fault models are *transition delay faults* [5, 6, 10], which represent delay defects at inputs and outputs of gates, and *path delay faults* [9], which consider cumulative delays along combinational paths. The number of gate delay faults in a circuit is linearly proportional to the number of gates but the number of possible paths can be exponential making it impossible to enumerate all path delay faults in a large circuit. Still, many existing methods for computing fault coverage use some form of path enumeration. Storing of detected path delay faults [8] is also infeasible for the above reason.

Pomeranz and Reddy proposed a method [7] for computing path delay fault coverage with polynomial time complexity in the number of lines in the circuit. As the order of the polynomial is increased, better estimates are obtained at the expense of increased computation time, which is exponential in the limit, due to the addition of cutsets in the circuit. For example, consider the circuit in Figure 1.

We will assume that path faults (R–2, 5, 7) and (R–1, 6, 8) have been detected earlier on separate test patterns. 'R' indicates a rising transition on the primary input (PI) which is at the origin of the corresponding path. In the Pomeranz-Reddy zero-order approximation method [7], the set of lines, 1, 2, 5, 6, 7, and 8, with respect to rising transitions, will be considered as *old-lines* as they have been part of a previously detected path fault. If a test pattern that detects the faults, (R–2, 6, 8) and (R–1, 5, 7), is now applied to the circuit, their method will not count these path faults as detected because both paths do not include a new line (line with the corresponding transition, which is not part of any previously detected path fault) in them. To reduce the error resulting in the fault coverage estimation, they add multiple cutsets to the circuit [7].

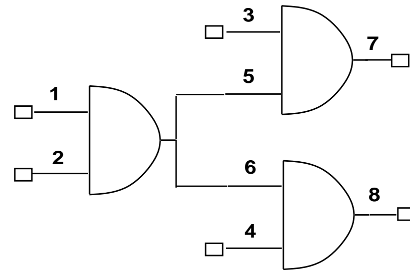

Figure 1: Error in zero-order approximation method

We use an efficient technique to compute the number of new path faults detected by a vector pair, which helps us in estimating the delay fault coverage for a test set. For every gate output in the circuit, we have twice as many flags as the number of inputs of the gate, which indicate whether or not each of those input-output pairs of signal lines have been included in a previously detected path fault. This information helps us to avoid the requirement that every detected path fault should include at least one line [7] that has not been included in any previously detected path fault

thus improving the approximation. We limit the complexity of our method to $O(n)$ by avoiding the use of *cutsets*, where n is the number of lines in the circuit, as opposed to $O(l^{k+1})$ for a kth order approximation involving subcircuits of l lines each [7], which is *exponential* in the worst case. We present experimental results to demonstrate the effectiveness of our method.

II. Estimation of Delay Fault Coverage

In a combinational circuit, a test pattern for a path fault essentially has two vectors to initiate a signal transition at the origin of the path and to propagate it to a primary output. We consider the problem of determining the number of path faults tested independently of gate delays by a given vector pair.

A. Detected Path Data Collection

We use a thirteen-valued algebra for our simulation [4], but the method can be adapted to use any other multi-valued logic algebra [2]. Each line in the circuit has a set of flags which indicates whether the line has been included in a previously detected path fault. For example, consider Figure 2, where l and m are the inputs of an AND gate and o is the output. Line o has two sets of flags, *S1* corresponding to rising transitions and *S2* corresponding to falling transitions. The set *S1* has two flags, which correspond to the two inputs of the AND gate. Flag *r1* indicates whether the pair of lines, o and l, were part of a previously detected path fault with a rising transition on o. Flag *r2* represents the same relationship between o and m. Similarly, set *S2* has two flags. Flag *f1* indicates whether the pair of lines, o and l, were part of a previously detected path fault with a falling transition on o. Flag *f2* represents the same relationship between o and m. A 0 on a flag indicates inclusion in

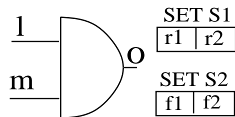

Figure 2: Maintaining new line information

a previously detected path fault. Thus, a 0 on flag *f2* indicates that the signal lines m and o have been part of some previously detected path fault with a falling transition on o. A 1 indicates otherwise. We denote the flags corresponding to a rising transitions as *R-flags* and those corresponding to a falling transition as *F-flags*. Thus *F-flag*(o, m) will mean the same as *f2*.

We need to compute two parameters for each line. Let *new-lines*(i) denote the number of newly detected path segments on which transitions propagate robustly between line i and a primary output (PO), for a given test pattern. Let *old-lines*(i) denote the number of path segments, between line i and a PO, which have been part of previously detected path faults. For every input of a gate which has a transition robustly propagated to a PO, the number of new path segments originating from it and ending at a PO is at least equal to the sum of the new path segments originating from the outputs of the gate and ending at POs. Note that new path segments represent those that have not been part of a previously detected path fault. In addition, if the input-output pair has not been part of a previously detected path fault for the corresponding transition, the old path segments originating from the gate output also add to the number of new path segments of the line under consideration. This information about the input-output pairs is maintained in the flags on each line. The flags corresponding to each input-output pair are updated when the outputs of a gate add to the number of new-lines of the corresponding input. The algorithm shown in Figure 3 computes the number of newly detected path faults after simulating the circuit for a test pattern. We only consider transitions that robustly propagate to the outputs.

Figures 4, 5, and 6 illustrate the working of the algorithm on the ISCAS-85 benchmark circuit c17. Each line has a set of parameters as indicated in Figure 4. The simulated signal values are determined by a forward pass and the other parameters (*new-lines*, *old-lines*, *R-flag*, and *F-flag*) are determined by a backward pass over the circuit as explained in the algorithm. We omit the flags on the PIs as they are unused. The test patterns chosen for illustration do not occur in sequence but it is assumed that only these patterns detect new path faults. The detected path faults in the figures are shown in dotted lines. The number of path faults detected on the first test pattern is three, the sum of the *new-lines* values of all PIs. The faults are (F–1, 8, 16), (F–6, 11, 15, 17), and (R–7, 15, 17). R (F) indicates that the PI at which the path originated had a rising (falling) transition. For example, consider line 15 which has a falling transition. The number of *new-lines* is one (same as the number of *new-lines* of line 17) because *R-flag*(17, 15) was initially one, meaning that the pair of lines,

Algorithm for computing number of newly detected path faults by a vector pair:

1. for all lines i connected to a PO
   - if i has a rising transition,
     - if all its *R-flags* are 1
       - $new\text{-}lines(i) = 1$ and $old\text{-}lines(i) = 0$
     - else
       - $new\text{-}lines(i) = 0$ and $old\text{-}lines(i) = 1$
   - else if i has a falling transition,
     - if all its *F-flags* are 1
       - $new\text{-}lines(i) = 1$ and $old\text{-}lines(i) = 0$
     - else
       - $new\text{-}lines(i) = 0$ and $old\text{-}lines(i) = 1$
   - else
     - $new\text{-}lines(i) = 0$ and $old\text{-}lines(i) = 0$

2. for every non-PO line i which is the input of gate G (having outputs o_1 to o_n with transitions on them)
   - if i has a transition
     - $new\text{-}lines(i) = 0$
     - for $k = 1$ to n
       - $new\text{-}lines(i) = new\text{-}lines(i) + new\text{-}lines(o_k)$
       - if o_k has a rising transition
         - $new\text{-}lines(i) = new\text{-}lines(i) + old\text{-}lines(o_k) \times R\text{-}flag(o_k, i))$
       - if o_k has a falling transition
         - $new\text{-}lines(i) = new\text{-}lines(i) + old\text{-}lines(o_k) \times F\text{-}flag(o_k, i))$
     - $total\text{-}lines = \sum_{k=1}^{k=n} (new\text{-}lines(o_k) + old\text{-}lines(o_k))$
     - $old\text{-}lines(i) = total\text{-}lines - new\text{-}lines(i)$
   - else
     - $new\text{-}lines(i) = 0$ and $old\text{-}lines(i) = 0$

   The outputs of gate G which do not have transitions do not contribute to the number of paths.
   - if $new\text{-}lines(i)$!= 0
     - for $k = 1$ to n
       - if ($new\text{-}lines(o_k)$!= 0) OR ($old\text{-}lines(o_k)$!= 0)
         - if o_k has a rising transition
           - $R\text{-}flag(o_k, i) = 0$
         - if o_k has a falling transition
           - $F\text{-}flag(o_k, i) = 0$

3. $i = i - 1$
   - if $i > 0$
     - go to step 2

4. Total number of new paths detected $= \sum new\text{-}lines(i)$ where i is a primary input

Figure 3: Computation of number of newly detected paths by a test pattern (Vector pair)

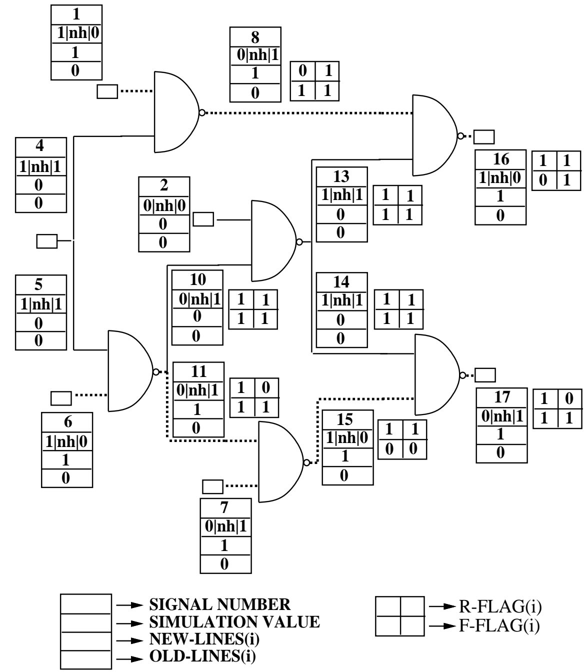

Figure 4: First test pattern for c17

15-17, has not been part of any previously detected path fault. After this computation, *R-flag*(17, 15) is set to 0 as shown. During the backward pass over the circuit, flags along signal lines which constitute a path eliminate the counting of path faults that have been detected before. The second test pattern detects three new faults: (F–5, 10, 14, 17), (F–5, 11, 15, 17), and (F–5, 10, 13, 16). It also detects one previously detected fault (see Figure 5). The third test pattern detects (F–6, 10, 14, 17), a new fault, along with two previously detected faults as shown in Figure 6. The example clearly illustrates the use of the flags on every line in distinguishing the usage of a line with respect to all of the lines immediately preceding it. The zero-order approximation method [7] would not have detected the fault (F–6, 10, 14, 17) during the third test because each of the lines, 6, 10, 14, and 17, have been included in previously detected path faults. In practice, this error tends to be large and hence our method improves the approximation without adding any cutsets, limiting the complexity to $O(n)$.

B. Computing Fault Coverage

The number of path faults detected by a test set is computed as the sum of the number of new path faults detected by every test. The total number of path faults can be determined in $O(n)$ time, where n

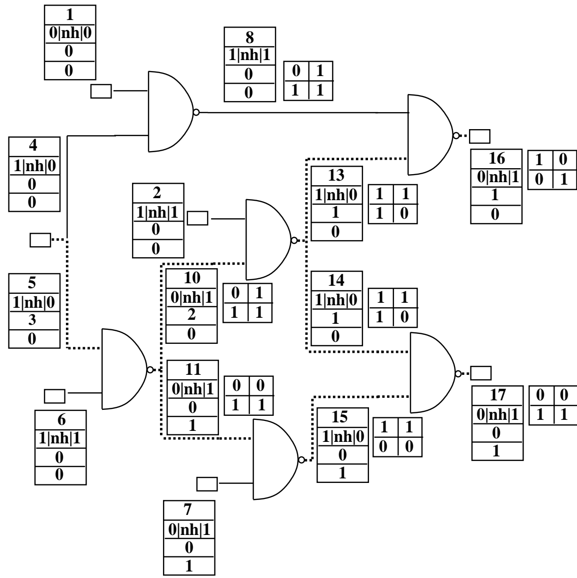

Figure 5: Second test pattern for c17

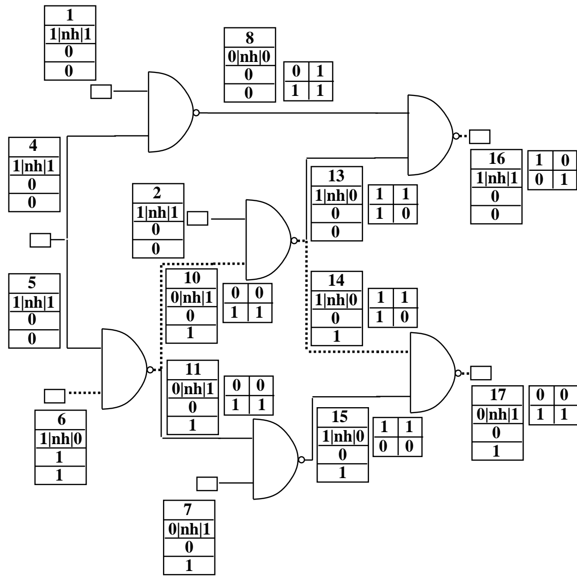

Figure 6: Third test pattern for c17

is the number of lines in the circuit. The number of path segments originating from each input of a gate to a PO is equal to the sum of the number of path segments originating from each gate output to a PO. For lines connected to POs, this number is set to one. In practice, the path fault counting can be done during one of the backward passes over the circuit. It should be noted that new path faults can be detected by tests even when they do not include any new lines. This is because the new line information is maintained only locally across each gate. For example, a gate input which is considered old with respect the gate output can be new with respect to a different line along the path being considered. Hence this path, which should actually be counted as detected, is not. Thus, the fault coverage estimate is always pessimistic, i.e., it is always equal to or lower than the actual fault coverage.

III. Results

We consider the full-scan circuits of the ISCAS-89 benchmarks. To demonstrate the effectiveness of our method, we compared the results with those from an existing delay fault simulator [3] using the same set of vectors [1]. We also compared our coverages with the zero-order approximation method [7]. The execution times in Table 1 are for a SUN 4/280 workstation. The table shows robust coverages for path delay faults. Our coverages correlate well with the actual fault coverages reported for these vectors by Bose *et al* [3]. The speedup obtained will be even higher for larger circuits because the estimator has *linear* complexity of $O(n)$, whereas due to the possibility of exponential number of paths, the fault simulator can have *exponential* complexity of $O(n \times 2^n)$, where n is the number of gates in the circuit.

We omit the CPU times for the Pomeranz-Reddy method [7] as they were similar to our method, but there is a marked improvement in the accuracy of our coverage estimation. The memory requirements for both the methods were similar. The error columns in Table 1 indicate the error in the coverage estimation for both methods. The large error in the Pomeranz-Reddy method is because of the requirement that every detected path fault should include at least one line that has not been included in any previously detected path fault, as explained earlier with the example of Figure 1. Figure 7 gives the error in estimating fault coverages by the Pomeranz-Reddy method using zero-order approximation and our method and illus-

Table 1: Coverage results for path delay faults

Ckt.	# Vect.	*Our Estimator* Cov%	Error	CPU(s)	*Pomeranz-Reddy* [7] Cov%	Error	*Bose et al* [3] Cov%	CPU(s)	Time Ratio
s298	704	66.6	1.8	0.8	20.5	47.9	68.4	10.1	12.6
s420	1428	67.6	6.7	2.7	34.1	40.2	74.3	29.3	10.8
s444	1466	49.9	0.5	3.6	15.9	34.5	50.4	32.1	8.9
s510	1476	80.3	5.5	3.5	27.5	58.3	85.8	25.9	7.4
s526	1416	78.8	1.1	3.9	17.3	62.6	79.9	33.5	8.6
s820	1968	89.0	0.7	9.5	19.0	70.7	89.7	61.1	6.4
s832	1992	86.5	0.2	10.2	18.5	68.2	86.7	63.3	6.2
s1196	4524	34.7	3.9	37.8	6.8	31.8	38.6	238.3	6.3
s1488	3832	86.2	5.5	45.1	11.5	80.2	91.7	242.7	5.4
s1494	3852	88.1	2.7	46.1	11.8	79.0	90.8	258.4	5.6

trates the improvement in approximation gained by our method. Better estimates can be obtained by increasing the order of the complexity polynomial by the Pomeranz-Reddy method, but this increases the complexity of estimation.

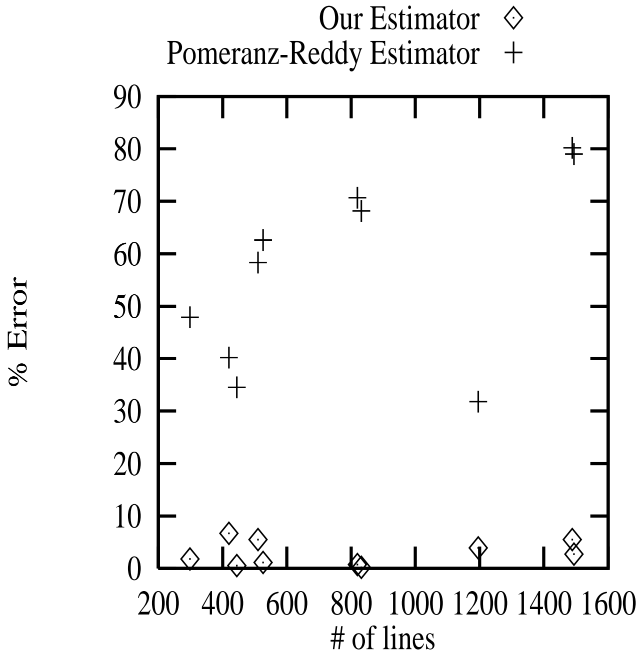

Figure 7: Estimation error in fault coverage

We also conducted experiments on the circuits suggested by Pomeranz and Reddy [7]. The basic block C_1 of the circuits is shown in Figure 8. This block is repeated a different number of times to obtain circuits of different sizes. The number of paths in a

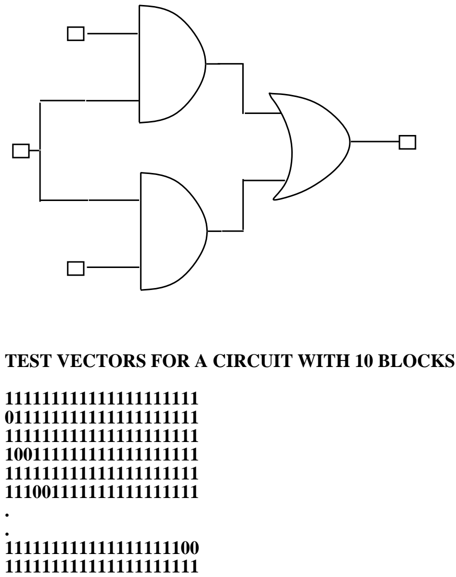

Figure 8: Basic cell of circuit with an exponential number of paths

Table 2: Coverage estimation for circuit with exponential number of paths

Blks	# Vect.	Faults	Detected	CPU(s)
10	23	6,140	6,140	0.5
20	43	6,291,452	6,291,452	0.6

circuit C_n, where the block is repeated n times is $N(C_n) = N(C_{n-1}) \times 2 + 2$. This simplifies to $N(C_n) = 3 \times 2^n - 2$. We applied tests which detect every slow-to-rise and slow-to-fall path faults in the circuit. Table 2 shows the results. The number of blocks is given first, followed by the number of tests applied. The fault coverages obtained by our method and the Pomeranz-Reddy method [7] had no error.

The tests were chosen in such a way that every test pattern detected path faults originating from different PIs. Hence, for every test pattern, the PIs that originated at the paths being tested were *new lines* and hence the Pomeranz-Reddy method worked correctly. But this is not always the case as we saw for the benchmark circuits above and the improvement in our method is very significant.

IV. Conclusion

Our *linear* complexity method of estimating the coverage for path delay faults has good accuracy. We compared our results with a fault simulator [3] to illustrate the difference in their complexities. We also compared our results with a fault estimator [7] to illustrate the vast reduction in the error in estimating the coverage. Only full scan circuits were considered but the method can easily be extended to non-scan synchronous circuits. We are currently investigating methods to compute fault coverage with zero error without increasing the complexity of the proposed method.

Acknowledgment

The research reported here was supported by the Center for Computer Aids for Industrial Productivity (CAIP), an Advanced Technology Center of the New Jersey Commission on Science and Technology at Rutgers University.

References

[1] P. Agrawal, V. D. Agrawal, and S. C. Seth. Generating Tests for Delay Faults in Non-scan Circuits. *IEEE Design & Test of Computers*, 10(1):20–28, March 1993.

[2] S. Bose, P. Agrawal, and V. D. Agrawal. Logic Systems for Path Delay Test Generation. In *Proc. EURO-DAC*, pages 200–205, September 1993.

[3] S. Bose, P. Agrawal, and V. D. Agrawal. Path Delay Fault Simulation of Sequential Circuits. *Trans. VLSI Systems*, 1(4):453–461, December 1993.

[4] T. J. Chakraborty, V. D. Agrawal, and M. L. Bushnell. Delay Fault Models and Test Generation for Random Logic Sequential Circuits. In *Proc. Design Automation Conf.*, pages 165–172, June 1992.

[5] T. Hayashi, K. Hatayama, K. Sato, and T. Natabe. A Delay Test Generator for Logic LSI. In *Proc. IEEE International Conf. on Fault Tolerant Computing (FTCS 14)*, pages 146–149, June 1984.

[6] E. P. Hsieh, R. A. Rasmussen, L. J. Vidunas, and W. T. Davis. Delay Test Generation. In *Proc. Design Automation Conf.*, pages 486–491, June 1977.

[7] I. Pomeranz and S. M. Reddy. An Efficient Non-Enumerative Method to Estimate Path Delay Fault Coverage. In *Proc. International Conf. CAD*, pages 560–566, November 1992.

[8] M. H. Schultz, F. Fink, and K. Fuchs. Parellel Pattern Fault Simulation of Path Delay Faults. In *Proc. Design Automation Conf.*, pages 357–363, June 1989.

[9] G. L. Smith. Model for Delay Faults Based Upon Paths. In *Proc. International Test Conf.*, pages 342–349, November 1985.

[10] J. Waicukauski, E. Lindbloom, B. K. Rosen, and V. S. Iyengar. Transition Fault Simulation. *IEEE Design and Test of Computers*, 4(2):32–38, April 1987.