

- [12] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.
- [13] R. L. Rivest, "RFC 1321: the MD5 message-digest algorithm," Internet Activities Board, Apr. 1992.
- [14] S. Tragoudas and M. Moiz Khan, "Test set preserving watermarking for firm cores," in *Proc. 6th WSEAS Int. Conf. Circuits*, Jun. 2002, pp. 4021–4023.
- [15] VSI Alliance, "Intellectual Property Protection White Paper: Schemes, Alternatives and Discussion Version 1," Intellectual Property Protection Development Working Group, Ver.1.1 Released, Aug. 2000.

On the ZBDD-Based Nonenumerative Path Delay Fault Coverage Calculation

Fatih Kocan and Mehmet H. Gunes

Abstract—We devise one exact and one pessimistic path delay fault (PDF) grading algorithms for combinational circuits. The first algorithm, an extension to the basic grading algorithm of Padmanaban, Michael, and Tragoudas (2003), does not store all of the detected PDFs during the course of grading, and, as a further improvement, it utilizes compressed representation of PDFs. These two techniques yield a space-and-time efficient algorithm. To enable grading of circuits with exponential number of paths, a circuit is first partitioned into a set of subcircuits. The second algorithm efficiently calculates the coverage of partitioned circuits. The former algorithm results in 50%–70% reduction in space and a speedup from 1.6 to 2.48 in ISCAS85 benchmarks. The time complexity of the latter algorithm is $O(N^2)$ subset operations per test vector where N is the number of nets in the circuit.

Index Terms—Fault grading, path delay fault (PDF), simulation.

I. INTRODUCTION

Delay testing of very large scale integrated (VLSI) circuits involves a test of the ability of the combinational circuit or combinational part of sequential circuits to propagate signals in a specified time period [2]. Four fault models were proposed for resistive bridge defects and process problems that change the timing behavior of a circuit. These are *gate delay* and its special case *transition delay* [3], *path delay* [4], and *segment delay* [5] fault models. The path delay fault model is the more realistic one and assumes that gate delays lumped on a path are enough to cause a failure. Also, a circuit may not have any defects to cause delay failure, but it still may have path delay faults due to incorrect cycle time choice. Consequently, circuits are tested for path delay faults to ensure that they meet their timing requirements.

Path delay fault (PDF) testing of a circuit requires the tasks of test vector simulation, test vector generation, and fault coverage calculation. Path delay fault simulation and coverage calculation can be *enumerative* or *nonenumerative* [6]. In enumerative fault simulation and coverage calculation, a test vector is simulated and the number of paths detected by the vector is counted one by one. Since the number of paths in a circuit may be exponential with respect to the circuit size (e.g.,

c6288 has $\approx 10^{20}$ paths), the time and space complexities of the enumerative algorithms are very high. As a result, a line of research is performed to solve the PDF simulation and fault coverage computations nonenumeratively, which is proved to be NP-hard [7]. In [8], path status graph (PSG) data structure is used to store PDFs and count the number of detected PDFs nonenumeratively. In [1], an efficient set data structure, ZBDD proposed in [9], is used to store and count PDFs with the set operations. All these data structures might require exponential size memory at worst case. Usually, to overcome memory problems of PDF coverage calculation algorithms, two complementing techniques are employed. 1) Instead of exact coverage calculation, we settle for estimated result, and 2) a circuit is *virtually cut* (as proposed in [6]) into overlapping subcircuits each of which has a unique set of PDFs, and grading is performed on the virtual subcircuits. For example, the authors of [1] propose a basic nonenumerative algorithm, which gives exact coverage, and a virtual-cut-based one, which results in coverage loss. The run time of the cut-based one grows exponentially with the number of cuts.

In this paper, we present two grading algorithms. The first algorithm, an extension to the ZBDD-based basic algorithm, dynamically prunes ZBDD at runtime. Our version of basic algorithm is suitable for PDF test grading task only, since it does not store all detected PDFs during the course of grading. The second algorithm calculates coverage of partitioned circuits efficiently. The run time of the latter algorithm is $O(N^2)$ subset operations per vector, N is the number of nets. Therefore, the run time does not grow exponentially with the number of cuts, i.e., partitions. Note that the cost of subset operations reduces as we increase the number of cuts.

The remainder of the paper is organized as follows. Section II surveys set-based (ZBDD-based) nonenumerative PDF coverage calculation method. In Section III, we present our test set grading algorithm, and describe the compressed PDF representation along with its advantages. In Section IV, the partition-level, set-based nonenumerative PDF coverage calculation algorithm is presented. In Section V, the experimental results for ISCAS85 benchmarks are included. In Section VI, we conclude our paper.

II. ZBDD-BASED PATH DELAY FAULT COVERAGE PRELIMINARIES

In the PDF model, there can be two PDFs on each physical path in a circuit: 1) slow-to-fall and 2) slow-to-rise. These PDFs are detected by applying a set of pairs of test patterns to the circuit: the first pattern in the pair initializes the circuit while the second one propagates the fault. PDFs can be detected robustly, nonrobustly, or functionally; however, the type of detection is an issue in simulation and test generation, not in coverage calculation. In coverage calculation, we apply test vectors one by one and find the ratio of uniquely detected PDFs over total number of PDFs. If needed, the coverage calculation must be repeated for every type of detection.

The authors of [1] employ a ZBDD set-data structure to calculate the coverage nonenumeratively. In their approach, a physical path is represented with the net labels along the path. The PDFs are modeled from the corresponding physical path by subscripting the first net label of the path with r (for rising fault) and f (for falling fault). For example, in c17 in Fig. 1, $1_r \cdot 10 \cdot 22$, and $1_f \cdot 10 \cdot 22$ are the two fault models on physical path $1 \cdot 10 \cdot 22$. With this modeling, they identify the set of PDFs detected by a vector, and accumulate all of the detected PDFs by a sequence of test vectors. Both of these tasks are carried out nonenumeratively.

Our work extends their basic PDF grading approach. Therefore, we use the basic ZBDD operators, and the *Store_PDF()* module (pp. 308 in [1]) in the subsequent sections. *Store_PDF()* module

Manuscript received March 16, 2004; revised July 14, 2004 and October 11, 2004. This paper was recommended by Associate Editor K. Chakrabarty.

The authors are with the Computer Science and Engineering Department, Southern Methodist University, Dallas, TX 75275 USA (e-mail: kocan@engr.smu.edu; mgunes@engr.smu.edu).

Digital Object Identifier 10.1109/TCAD.2005.850851

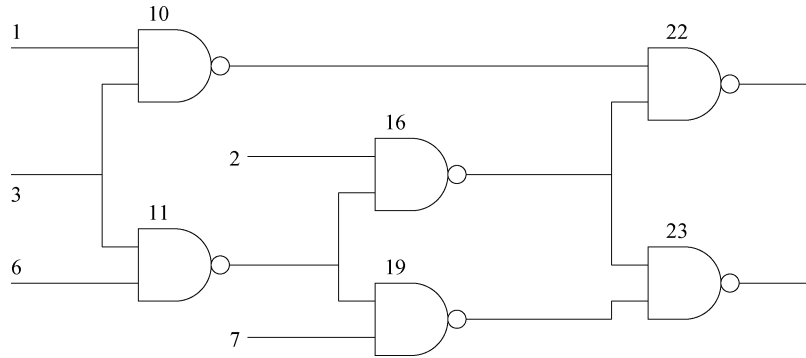
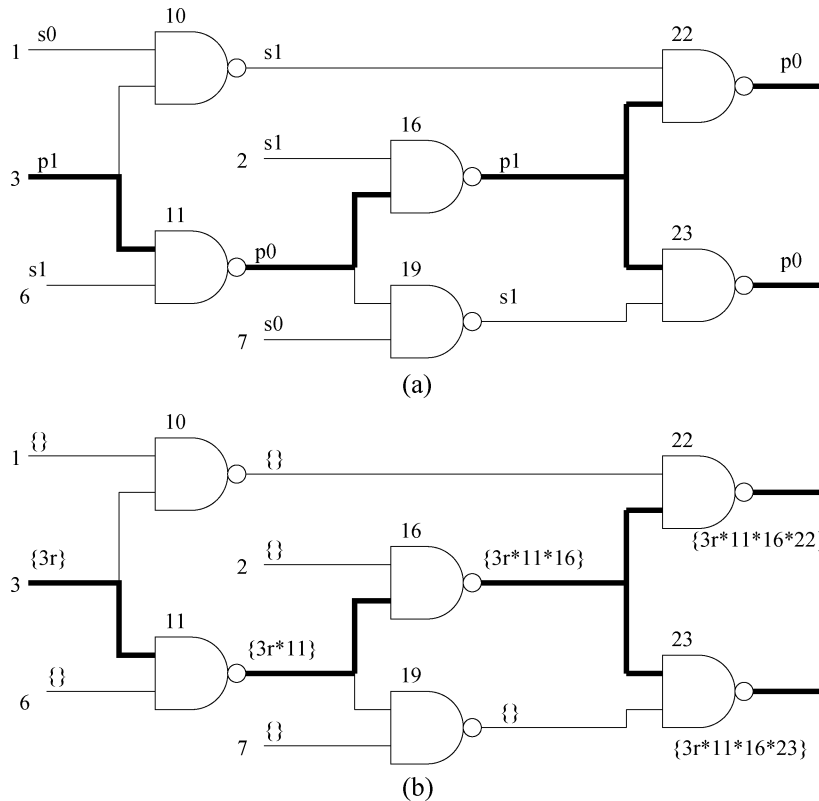


Fig. 1. Circuit c17.

Fig. 2. Set of detected PDFs. (a) Simulation of vector $\langle s_0, s_1, p_1, s_1, s_0 \rangle$. (b) Building the set of PDFs detected by the vector.

builds the set of PDFs detected by a vector. To illustrate the function of *Store_PDF()* module, assume that a vector $\langle s_0, s_1, p_1, s_1, s_0 \rangle$ is applied to c17 and the paths detected by this vector is highlighted in Fig. 2(a). The module creates sets on the nets to hold the set of partial PDFs that starts from the primary inputs and ends at the nets. The set of a primary input would be empty if it is not part of any detected path. Otherwise, there would be one set including the label of the primary input, which is subscripted according to the logic value on the input. For example, primary input 3 is p_1 and detected in Fig. 2(a); therefore, the set of input 3 is $\{3_r\}$. The set of PDFs of a gate is computed by taking the union of the sets of its fanins and appending the label of the gate to all PDFs in the union. At the end, the union of the sets of the primary outputs gives the set of the PDFs detected by a vector. Fig. 2(b) illustrates construction of the set of PDFs detected by this vector: The set of detected PDFs is $\{3_r \cdot 11 \cdot 16 \cdot 22, 3_r \cdot 11 \cdot 16 \cdot 22\}$. In addition to this module, we use the following ZBDD operators: *Subset1*(S, v), *Subset0*(S, v), *Union*(S, Q), and *Count*(S). *Count*(S) operation returns the cardinality of set S . *Subset1*(S, v) (*Subset0*(S, v)) returns the members, i.e., minterms, of S with (without) node v .

To enable nonenumerative PDF coverage calculation of ULSI and SoC circuits, whose memory requirements are very high, a circuit is first virtually cut into several subcircuits [6] and then PDF coverage calculation is computed at the subcircuit level. The virtual cut method is originally proposed to improve the estimated coverage value and also employed in the ZBDD-based coverage calculation to reduce memory requirement at the expense of some loss in the coverage. In the ZBDD-based coverage calculation, all PDFs in one subcircuit is stored in one ZBDD and the final coverage is calculated from the collection of ZBDDs. The run time of the cut-based grading algorithm in [1] grows exponentially with the number of cuts.

III. EXTENDED BASIC PDF COVERAGE CALCULATION ALGORITHM

This section modifies the basic coverage algorithm of [1] to enable grading of test sets efficiently. The modified grading algorithm (B^+) does not store all of the detected PDFs during the course of simulation. B^+ requires the set of test vectors (V) *a priori* and calculates the PDF coverage in two phases. The algorithm shows the details of the

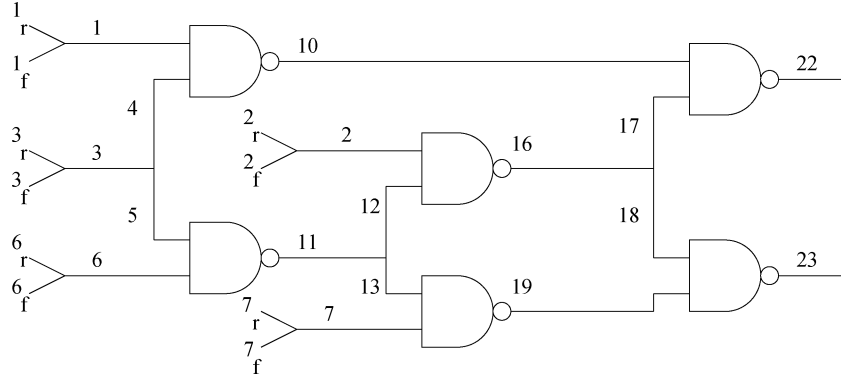


Fig. 3. Plines in modified c17.

two-phase algorithm. In this algorithm, $P(C)$ is a ZBDD and holds all detected PDFs; N_d is the set of nets covered by current vector; $P(V_i)$ is a ZBDD and holds the set of PDFs detected by vector V_i . The algorithm precomputes coverage frequencies of all nets by simulating all vectors in the first phase, and sets the coverage frequency of each net i (F_i). F_i indicates the number of vectors that cover line i . In the second phase, the algorithm resimulates all test vectors one by one, and decrements the frequencies of nets covered by the current vector. At that point, it checks for all zero-frequency nets, removes them along with their PDFs from the set of PDFs, and updates pdf variable with the number of deleted PDFs. At the end, the set of PDFs is empty, and pdf holds the number of covered PDFs.

Note that calculating the frequency of a net in the first phase, instead of marking the last vector in the sequence that detects the net and removing the net after application of this vector in the second phase, gives the flexibility of reordering of test vectors before the second phase. Vector scheduling/reordering helps early removal of nets. For example, let net p be detected by vectors V_{10} and V_{1000} , i.e., $F_p = 2$. Also, let p be the minimum frequency net reported by the first phase of the algorithm. Scheduling of V_{10} and V_{1000} as the first two vectors in the second phase would result in early removal of p and its corresponding PDFs from the set. Therefore, we present two vector reordering methods. After that, we present a compressed representation for PDFs to further reduce the space complexity of the proposed algorithm.

Reordering Based-On PI Frequencies (H1): This method first applies the vectors of the minimum frequency primary input net, next the vectors of the second minimum frequency input net among the remaining vector set, and so on.

Reordering Based-On PI PDF Counts (H2): This method calculates the number of PDFs passing through each primary input net. Then, it schedules the vectors of the minimum PDF count input net first, the vectors of second minimum one from the remaining vectors, and so on.

A PDF representation can be compressed using the concept of primary lines in a circuit to further reduce the size of ZBDD during coverage calculation. A line is a primary line (*pline*) if it is a branch line or a primary input. A path constructed only with branch labels maintains the path *id*.

Fact 1: Let $x_1 \cdot x_2 \cdot x_3 \cdots x_n$ be a path and there exists only one path from x_1 to x_n . Then, $x_1 \cdot x_n$ can model the entire path without losing any information. Since we have a unique path from x_1 to x_n , we can reconstruct the complete path. If there are no other paths passing through x_1 , then $x_1 \cdot x_n$ can be simplified into x_1 .

The advantages of using plines over nets in PDF modeling in the context of grading are as follows.

- 1) Eliminate redundant nodes from the path/PDF representation. For example, in Fig. 3, there exists only one PDF passing through $1_r \cdot 10 \cdot 22$, and 1_r can model this entire PDF. Compressed PDFs require less number of nodes in the ZBDD than full PDFs. As we add the detected PDFs to the ZBDD, nodes are created. In some cases, for example, to distinguish the detected overlapping subpaths in ZBDD, multiple replicas of the node at the joint of the overlapping subpaths might be created. We use Fig. 4 to illustrate this case. In this figure, net 10 is a joint point in the net-level PDF modeling. Assume that vector v_1 detects PDF $1_f \cdot 7 \cdot 8 \cdot 10 \cdot 11 \cdot 13$ and v_2 detects PDF $1_f \cdot 7 \cdot 9 \cdot 10 \cdot 12 \cdot 13$. Fig. 5(a) and (b) show the ZBDD structures when these two PDFs are stored with nets and plines, respectively. Fig. 5(a) has two replicas of node 10 to distinguish these detected PDFs. Note that whether PDFs are modeled with plines or nets, creation of the replicas of the nodes in the joint points to distinguish the overlapping subpaths is inevitable. However, pline modeling does not cause replication of redundant nodes.
- 2) Enable early removal of nodes from ZBDD when B^+ grading algorithm is used. Fig. 6 illustrates this case. In this figure, assume that there are 100 PDFs passing through net a : 70 of them follow branch b while 30 PDFs pass through c . When net representation is used, 100 PDF must be covered before node a along with its PDFs can be removed. When pline representation is used, node $b(c)$ along with its PDFs can be removed after the coverage of 70(30) PDFs.

Algorithm 1 Grading with frequencies

```

( $B^+$ )
pdf ← 0
for every test  $V_i = (V_{i_1}, V_{i_2}) \in V$  do
   $N_d \leftarrow \text{Simulate}(C, V_i)$ 
  Increment  $F_j$  of each net  $j \in N_d$ 
end for
ReorderVectors()
 $P(C) \leftarrow \text{Empty}()$ 
for every test  $V_i = (V_{i_1}, V_{i_2}) \in V$  do
   $N_d \leftarrow \text{Simulate}(C, V_i)$ 
   $P(V_i) \leftarrow \text{STORE\_PDF}(N_d)$ 
  Decrement  $F_i$  of each net  $i \in N_d$ 
   $P(C) \leftarrow \text{Union}(P(C), P(V_i))$ 
  while  $\exists m F_m == 0$  do
    pdf ← pdf + Count(subset1( $P(C), m$ ))
     $P(C) \leftarrow \text{subset0}(P(C), m)$ 
  end while
end for
Ensure:  $P(C)$  is empty
    
```

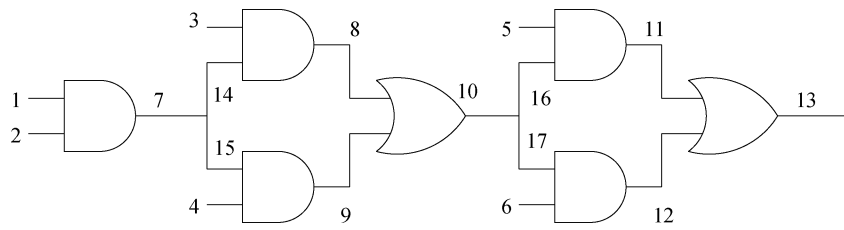


Fig. 4. Pomeranz-Reddy example.

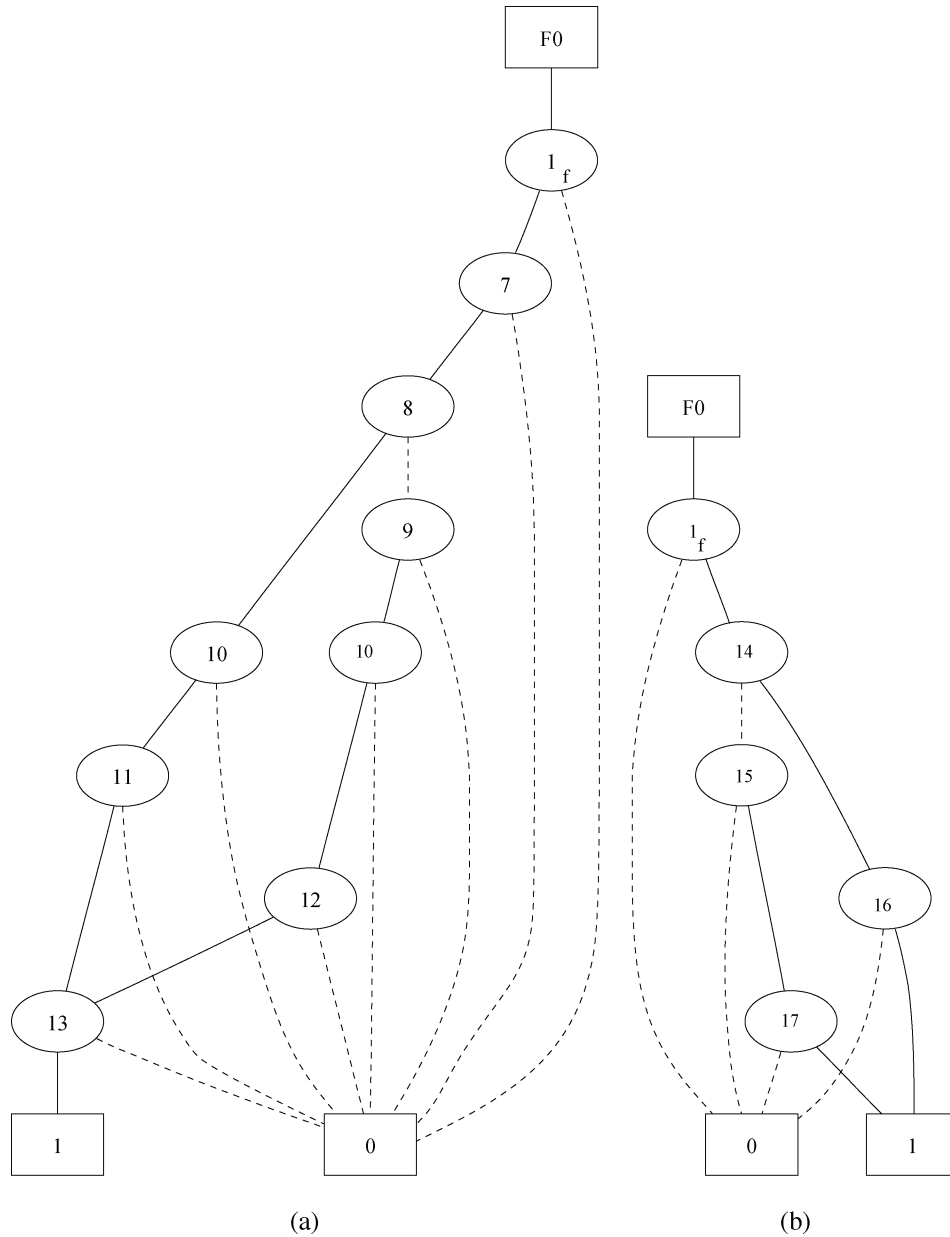


Fig. 5. Spatial relationship. (a) Nets. (b) Plines.

Although compressed PDF representation is correct for fault grading, it may be problematic to PDF ATPG (considering coupling effects), diagnosis and timing analysis efforts. However, some lines or nets of interests can be included in the PDF representation in addition to the primary lines to alleviate the disadvantage of compression.

IV. PARTITION-LEVEL COVERAGE CALCULATION

Exponential number of PDFs in ULSI circuits and SoC, with respect to circuit size, induces space issues during PDF simulation and coverage calculation. This problem is overcome by partitioning or virtu-

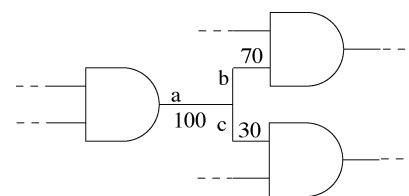


Fig. 6. Distinguishing the PDFs at nets.

ally cutting such a circuit into subcircuits and by performing grading at the partition level. Partitioning of circuits creates overlapping sub-

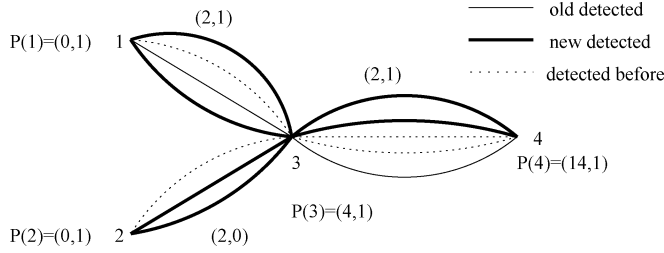


Fig. 7. Counting new PDFs detected by a vector at partition level.

path problem in which some of the newly detected PDFs would be missed due to partitioning. In [1], an algorithm is given for coverage calculation of circuits that are cut one or more times. In their algorithm, they utilize *principle of inclusion and exclusion* to pessimistically report the coverage. Their algorithm's run time grows exponentially with the number of cuts. In this section, we devise an efficient partition-level ZBDD-based nonnumerative coverage calculation algorithm. Our algorithm also gives the coverage pessimistically: in fact our algorithm and theirs report the same estimated value. The partial paths in a partition are nonnumeratively stored in ZBDD using *Store_PDF()* module. We maintain two sets per each partition k : P_k for storing all partial paths detected by vectors $v_1 \dots v_{n-1}$ and C_k for storing partial paths detected by vector v_n . Thus, we store all paths covered by a set of test vectors. To simplify the explanation of the algorithm, we define three fundamental operations on tuples. The operations are \bowtie , $+$, and *First*. The definitions of the operations and identities are as follows.

- 1) $(A, B) \bowtie (C, D) = (A \times C + A \times D + B \times C, B \times D)$.
- 2) $(A, B) + (C, D) = (A + C, B + D)$.
- 3) $(A, B) \bowtie (C, D) = (C, D) \bowtie (A, B)$.
- 4) $(A, B) + (C, D) = (C, D) + (A, B)$.
- 5) $(A, B) \bowtie (0, 1) = (0, 1) \bowtie (A, B) = (A, B)$.
- 6) *First*(A, B) = A .

Assume that $P(i)$ is a tuple (A, B) and holds the number of incoming new paths in A and redetected paths in B at node i . Further, assume that E_{k-l} is also a tuple and holds the numbers of new and redetected partial paths between input node k and output node l of a partition. As an example, in Fig. 7, the input nodes of the first (second) partition are 1 and 2 (3), and the output node of the first (second) partition is 3 (4). Moreover, there are three types of partial PDFs between an input and output, as indicated by the legend. The unsolid and solid lines represent currently detected partial paths: The solid ones are first time detected while the unsolid ones are redetected by the current vector. The dashed ones just show the partial PDFs that are detected only by the previous vectors. In this example, $E_{1-3} = (2, 1)$, $E_{2-3} = (2, 0)$, and $E_{3-4} = (2, 1)$. The first field of $P(4)$ would give us the number of uniquely detected PDFs by the current vector. To calculate $P(4)$, first we initialize $P(1)$ and $P(2)$ to $(0, 1)$. It follows that $P(3) = P(1) \bowtie E_{1-3} + P(2) \bowtie E_{2-3}$ and $P(4) = P(3) \bowtie E_{3-4}$. Note that the current vector detects total 15 PDFs and only one of them is an old one. Next, we formalize the partition-level new PDF counting in terms of set operations.

Let $C_{k,i-j}$ and $P_{k,i-j}$ be the sets of newly and previously detected partial PDFs between an input i and output j of partition k , respectively. $C_{k,i-j} \setminus (P_{k,i-j})$ is computed with *subset* operations on $C_k \setminus (P_k)$. Then, the number of uniquely detected partial PDFs between input i and output j in partition k is $|C_{k,i-j} \setminus P_{k,i-j}|$ and the number of redetected partial PDFs between them is $|C_{k,i-j} \cap P_{k,i-j}|$. For every valid input-output pair i and j in partition k , we define a pair to hold these numbers, i.e., $E_{k,i-j} = (|C_{k,i-j} \setminus P_{k,i-j}|, |C_{k,i-j} \cap P_{k,i-j}|)$.

TABLE I
DESCRIPTION OF BENCHMARKS

	<i>I/O</i>	<i>Net</i>	<i>Paths</i>
c880	26/60	417	8642
c1355	41/32	514	4173216
c1908	33/25	855	729057
c2670	233/140	1053	679960
c3540	50/22	1647	28676670
c5315	178/123	2184	1341305
c6288	32/32	2384	9.894344x10 ¹⁹
c7552	207/108	3404	726494
cs5378	214/228	2779	27046
cs9234	247/250	5597	489708
cs13207	700/790	7951	2690738
cs15850	611/684	9772	329476092
cs38417	1664/1742	22179	2783158
cs38584	1464/1750	19253	2161446

TABLE II
STORING ALL PDFs IN B^* VERSUS B

	B^*		B^*	B		B
	Final	Peak		Final	Peak	
c880	557	60298	0.05	940	87892	0.09
c1355	850	293314	0.37	1396	554946	0.68
c1908	1061	270830	0.29	1941	595826	0.76
c2670	1710	248346	0.26	2979	334194	0.38
c3540	1921	778764	1.16	3590	1305094	2.65
c5315	3186	295358	0.58	5493	516110	0.82
c6288	3904	2951536	13.05	6320	8352806	19.15
c7552	4247	751170	0.90	7759	1150772	1.83
cs5378	3384	262654	0.27	5539	383250	0.74
cs9234	3884	435372	0.75	9481	1197784	1.84
cs13207	5928	630574	2.38	14186	1591254	2.77
cs15850	6686	1188586	4.48	16461	4863698	10.88
cs38417	17824	3018988	14.84	40339	6036954	20.28
cs38584	20643	1580012	20.22	40048	3828412	13.83

TABLE III
ROBUST VECTORS B^* VERSUS B

	B^*		B^*	B		B	PDF covered
	Final	Peak		Final	Peak		
c880	3554	60298	0.93	6833	102200	2.15	7484
c1355	4713	36792	3.05	9337	55188	5.57	17911
c1908	18756	64386	6.79	38866	134904	17.14	27064
c2670	2589	110376	10.68	4978	261632	19.57	10619
c3540	16410	68474	8.33	36075	148190	16.59	25752
c5315	6870	99134	22.41	14515	288204	56.88	37421
c7552	12093	85848	92.26	24460	314776	152.56	52019

TABLE IV
GRADING ALGORITHM

	$H1$		$H2$		$B^*/H2$
	MaxFinal	Time	MaxFinal	Time	
c880	3388	0.96	1737	1.01	2.04
c1355	4384	3.08	134	2.49	35.17
c1908	13871	6.88	3974	6.21	4.71
c2670	2392	11.01	1736	10.45	1.49
c3540	11584	12.44	5037	7.28	3.25
c5315	6853	23.35	5840	23.51	1.17
c7552	10883	92.14	8803	95.43	1.37

Algorithm 2 performs partition-level coverage calculation based on this formalization. Algorithm 2 also updates the sets of the partitions with the partial PDFs detected by the current vector. Note that PO is the set of primary outputs at the partition level. Additionally, the

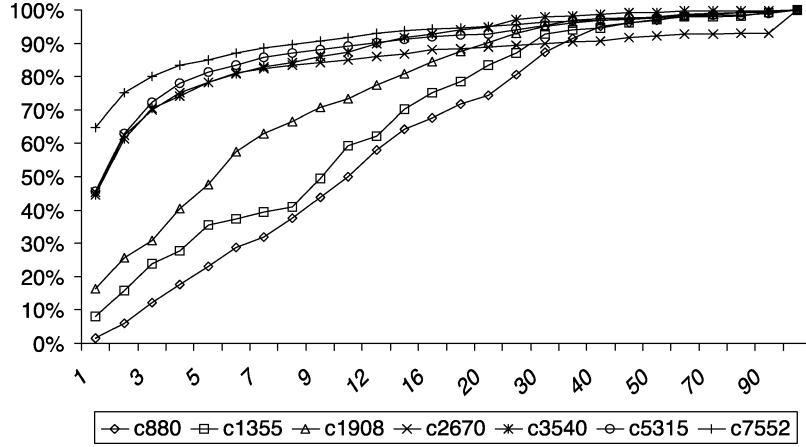


Fig. 8. Frequency distributions of the primary lines.

space complexity of the algorithm is proportional to the sum of memory requirements of the individual partitions.

Algorithm 2 Partition-Level Coverage
 Calculation: Coverage()
Require: $\forall_i C_i$ and $P_i \ i \in [1, K]$, K : the number of partitions
 $\forall_i P(i) \leftarrow (0, 1)$ where i is a PI
for $l \leftarrow 1$; $l \leq \text{maxlevel}$; $l++$ **do**
 for every partition k at level l **do**
 for every logical output j of k **do**
 $P(j) \leftarrow 0$
 for each logical input i that has a path to j **do**
 $E_{k,i \rightarrow j} \leftarrow (|C_{k,i \rightarrow j} \setminus P_{k,i \rightarrow j}|, |C_{k,i \rightarrow j} \cap P_{k,i \rightarrow j}|)$
 $P(j) \leftarrow P(j) + P(i) \bowtie E_{k,i \rightarrow j}$
 end for
 end for
end for
 $\forall_k P_k \leftarrow P_k \cup C_k$ k is a partition
return $\sum_{i \in PO} \text{First}(P(i))$

Number of Subset Operations: The number of subset operations in Algorithm 2 is $O(\sum_{i=1}^K |I_i| \times |O_i|)$ per vector, where K is the number of the partitions and I_i (O_i) is the input (output) count of partition i . $T(N) = O(K \cdot N^2) = O(N^2)$ subset operations per vector, where N is the number of nets. Therefore, the total number of subset operations per T vectors would be $O(T \cdot N^2)$. Note that the cost of subset operation reduces as the number of partitions increases.

V. EXPERIMENTAL RESULTS

We have performed several experiments on the benchmarks in Table I. In the first experiment, we have measured the impact of PDF modeling with primary lines on ZBDD size for two cases. 1) All of PDFs were stored and 2) PDFs detected by a robust test vector set were stored. In the second experiment, we assessed the space and time requirements of dynamic pruning algorithm (B^+) for each vector reordering heuristics. All experiments were performed on Pentium IV, 2.4 GHz, and 1-GB RAM with Linux OS. Additionally, we used ZBDD implementation available in CUDD package [10].

In Table I, we tabulate the number of inputs and outputs (I/O), nets, and paths for each benchmark circuit.

A. Size Impact of Pline-Based Modeling of PDFs

We performed two sets of experiments to show that modeling PDFs with plines reduces the size of ZBDD. In the first set, we stored all PDFs in a ZBDD and measured the final and peak node usages, and the processing times in seconds. This experiment was repeated with both pline-modeled PDFs (B^*) and net-modeled PDFs (B). We used the following static ordering of variables in ZBDD. The variables were ordered according to their levels in the graph of the circuit: starting from the root, first the input plines at the first level, next the plines at the second level, and so on. We also observed that the ordering among the plines at the same level were not important, and the static ordering of variables had great impact on the size of ZBDD [11]. For net-based experiments, we used the same static ordering. Table II tabulates the pline/net modeling results under B^*/B columns. Pline modeling reduced approximately half of the nodes and processing times, compared with net modeling. We observed that with this static ordering, the number of final nodes of B^* is equal to the number of plines in the circuit. For example, when all PDFs of c880 were stored, there were 557 final nodes in ZBDD, which is equal to the number of plines in the circuit.

A similar experiment was performed with a set of robust test vectors. Table III tabulates the pline/net results for the vector set under B^*/B columns. The last column (PDF) indicates the coverage. Pline modeling required approximately half of final and peak nodes of net modeling. Moreover, B^* requires less ZBDD processing time than B .

B. Results for Dynamic Pruning Algorithm (B^+)

We computed the *maximum final* node counts, i.e., the maximum number of final nodes reached during the grading, and processing times of the test set grading algorithm for two vector ordering heuristics when the same set of robust test vectors in Section V-A were used, and also the PDFs modeled with plines. Table IV tabulates the results. In all benchmarks, the ordering of vectors based on the path counts of the inputs ($H2$) outperformed the ordering based on frequencies ($H1$). Compared to the pruningless results in Table III, pruning reduced the node counts further in the ZBDD considerably. The node-count ratios for $B^*/H2$ are given in the last column of the table.

Since the pruning algorithm sensitive to the frequency distribution of the nets or plines (depending on the chosen modeling), we also provided the cumulative coverage frequencies of each benchmark in Fig. 8. We created the graph as follows. For each benchmark, we

calculated the coverage frequencies of the primary lines (F_i) for the set of robust test vectors and normalized each F_i according to $(F_i/\max(F_1, \dots, F_n)) \times 100$, where n is the number of lines in the circuit. The x axis shows the normalized frequencies and the y axis shows the cumulative percentage of plines. For example, in c880, approximately, 20% of the plines has a normalized frequency of 5 or less. Almost in all benchmarks, 90% of plines has a normalized frequency of 30 or less.

VI. CONCLUSION

In this paper, *first*, we modified the basic ZBDD-based algorithm to enable efficient test set grading of circuits. The algorithm collects statistics about the coverage frequencies of the nets/lines and at run time removes the fully covered nodes to reduce the size of the ZBDD. This algorithm is suitable for PDF test set grading since it does not store all of the detected PDFs. This algorithm results in more than 50% reduction in time and space, compared to the pruningless version of the algorithm. *Second*, we devised a run-time efficient partition-level test set grading algorithm. The run time of the algorithm is $O(N^2)$ subset operations per vector, where N is the number of nets/lines in the circuit.

In the future, we plan to investigate better pruning heuristics to avoid ZBDD size explosion during the course of grading.

ACKNOWLEDGMENT

The authors would like to thank Prof. M. K. Michael and Prof. S. Tragoudas for enabling the experimental results by providing the robust test vectors. They would also like to thank the reviewers, whose comments improved this paper.

REFERENCES

- [1] S. Padmanaban, M. K. Michael, and S. Tragoudas, "Exact path delay fault coverage with fundamental ZBDD operations," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 22, no. 3, pp. 305–316, Mar. 2003.
- [2] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*. Norwell, MA: Kluwer, 2000.
- [3] J. A. Waicukauski, E. Lindbloom, B. K. Rosen, and V. S. Iyengar, "Transition fault simulation," *IEEE Des. Test Comput.*, vol. 4, pp. 32–38, Apr. 1987.
- [4] G. L. Smith, "Model for delay faults based upon paths," in *Proc. Int. Test Conf.*, Nov. 1985, pp. 342–351.
- [5] K. Heragu, J. H. Patel, and V. D. Agrawal, "Segment delay faults: a new fault model," in *Proc. IEEE VLSI Test Symp.*, Apr. 1996, pp. 32–39.
- [6] I. Pomeranz and S. M. Reddy, "An efficient nonenumerative method to estimate the path delay fault coverage in combinational circuits," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 13, no. 2, pp. 240–250, Feb. 1994.
- [7] D. Kagaris and S. Tragoudas, "On the nonenumerative path delay fault simulation problem," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 21, no. 9, pp. 1095–1101, Sep. 2002.
- [8] M. A. Gharaybeh, M. L. Bushnell, and V. D. Agrawal, "The path-status graph with application to delay fault simulation," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 17, no. 9, pp. 324–332, Sep. 1998.
- [9] S. Minato, "Zero-suppressed BDDs for set manipulation in combinatorial problems," in *Proc. ACM/IEEE Design Automation Conf.*, 1993, pp. 272–277.
- [10] F. Somenzi, "CUDD: CU decision diagram package," Univ. Colorado, Boulder.
- [11] F. Kocan, M. H. Gunes, and M. Thornton, "Static variable ordering in ZBDD for path delay fault coverage calculation," in *Proc. IEEE Int. Midwest Symp. Circuits and Systems*, Jul. 2004, pp. 1–97.