Static Variable Ordering in ZBDDs for Path Delay Fault Coverage Calculation*

Fatih Kocan, Mehmet Gunes, Mitchell A. Thornton
Southern Methodist University, Dallas, Texas, U.S.A.

Abstract

Zero-suppressed Binary Decision Diagrams (ZBDDs) are data structures that represent sets efficiently and they have recently been suggested for use in *nonenumerative* path delay fault (PDF) coverage calculations. Many heuristics have been proposed to order variables (representing primary inputs) in ZBDDs to avoid size explosion; however, in ZBDD-based PDF coverage calculations, the variables represent the nets in a circuit, not the circuit primary inputs. This fact motivates us to investigate new ordering strategies since the number of nets in a circuit is relatively large as compared to the number of primary inputs. Several new static ordering heuristics are proposed based on structural properties of the circuit undergoing PDF coverage calculations and are evaluated. The experimental results show that the new heuristics we propose greatly reduce the size of the ZBDDs.

Topic 3. Programmable Logic, VLSI,CAD, and Layout

1 Introduction

Delay testing of VLSI circuits involves a test of the ability of the combinational or combinational part of sequential circuits to propagate signals within a specified amount of time. Usually, if a test fails and there are no DC faults such as stuck-at, then we look for transition, gate, segment, and path delay faults in a circuit [1]. The PDF model represents a delay along a path from a primary input or the output of a flip-flop to a primary output or the input of a flip-flop, and is considered to be the most accurate of these fault models. The number of possible PDFs, twice the number of paths in a circuit, is far higher than the possible number of transition delay or stuck-at faults in a circuit. Therefore, test vector generation and fault coverage calculation for PDFs are very time-consuming tasks in comparison to those of stuck-at or other delay fault models.

Exact and approximate enumerative and nonemumerative PDF coverage calculation algorithms are proposed in the literature [6]. In [2], an exact nonenumerative algorithm with a path-status graph (PSG) data structure is presented. The PSG holds the detection statuses of the PDFs in a circuit and, in the worst case, requires exponential space. Recently, in [6], ZBDDs [4] are used to calculate exact PDF coverage nonenumeratively. Each PDF is modeled as a subset of all nets in a circuit, and all detected PDFs are stored in the ZBDD compactly. Since the toal possible number of PDFs in a circuit is exponential in size, the size of the ZBDD in the worst case may be exponential. Therefore, it is necessary to find a good ordering of variables in ZBDDs to keep the size under control, which will in turn reduce processing time.

In this paper, we describe several new static variable ordering heuristics and measure their impact on the size of ZBDDs. The heuristics are based on structural properties of the circuit undergoing PDF coverage computations since the resulting ZBDD is used to store signal paths rather than function behavior. Using ZBDDs for PDF coverage computations results in the structure of the circuit being represented instead of the functional behavior that is normally represented by ZBDDs in other applications. Past static variable ordering techniques were intended to be applied to ZBDDs that represent function behavior rather than circuit structure and thus may not be as well-suited for minimizing ZBDD size for PDF coverage calculations. We implement our new static variable ordering methods and analyze their behavior in terms of the maximum number of peak nodes created and final ZBDD structure size.

The paper is organized as follows. In Section 2, the ZBDD-based PDF coverage calculations are briefly described. In Section 3, new ordering heuristics are presented. In Section 4, experimental results are given for a limited number of PDFs and all possible PDFs per circuit are given for each ordering heuristic. Finally, we conclude our paper in Section 5.

2 Preliminaries

PDFs are detected by applying a set of pairs of test patterns to the circuit: the first pattern in a pair initializes the circuit while the second one propagates the fault. PDFs can be detected robustly, non-robustly, or functionally; however, the type of detection is an issue in simulation and test generation, not in coverage calcu-

*This work was supported in part by Texas Advanced Technology Program (ATP) grant 003613-0029-2003

	Physical Paths	
$1 \cdot 10 \cdot 22$	$3 \cdot 11 \cdot 16 \cdot 22$	$6 \cdot 11 \cdot 16 \cdot 23$
$2 \cdot 16 \cdot 22$	$3 \cdot 11 \cdot 16 \cdot 23$	$6 \cdot 11 \cdot 19 \cdot 23$
$2 \cdot 16 \cdot 23$	$3 \cdot 11 \cdot 19 \cdot 23$	$7 \cdot 19 \cdot 23$
$3 \cdot 10 \cdot 22$	$6 \cdot 11 \cdot 16 \cdot 22$	

Table 1: Physical Paths

lation. In coverage calculation, we apply test vectors one-by-one and find the ratio of detected PDFs to total number of PDFs. The coverage calculation must be repeated for every type of detection, if needed.

Table 1 lists all physical paths in c17 in Figure 1. There are two PDFs on each physical path: 1) Slow-to-fall and 2) Slow-to-rise. These faults are modeled from the corresponding physical path by subscripting the first variable of the path with f or r. For example, $1_f \cdot 10 \cdot 22$ and $1_r \cdot 10 \cdot 22$ are the two faults on path $1 \cdot 10 \cdot 22$.

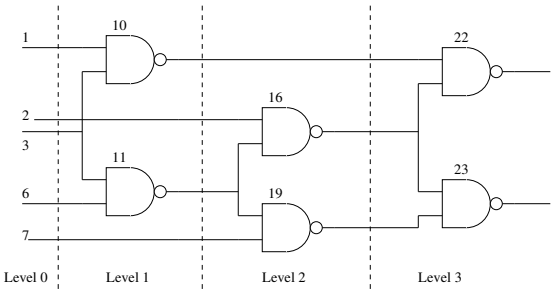
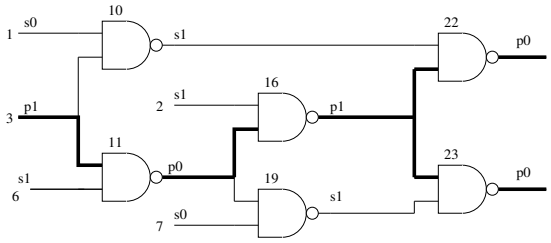


Figure 1: Circuit c17

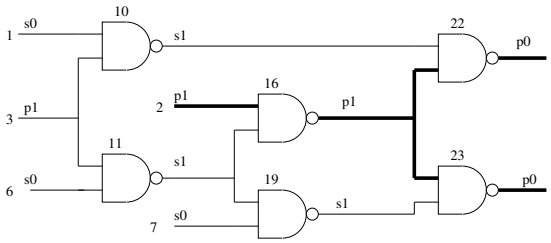
The two vectors $v_1 =< s0, s1, p1, s1, s0 >$ and $v_2 =< s0, p1, p1, s0, s0 >$ detect the highlighted paths in Figures 2(a) and (b), respectively. Here, si means the value of the line in the first pattern and in the second pattern is i; pi means the value of the line in the second pattern is i and in the first pattern is the opposite. v_1 covers two PDFs: $3_r \cdot 11 \cdot 16 \cdot 22$, and $3_r \cdot 11 \cdot 16 \cdot 23$. Similarly, v_2 covers $2_r \cdot 16 \cdot 22$ and $2_r \cdot 16 \cdot 23$. Therefore, the PDF coverage of these two vectors is 4/22.

In set-based nonenumerative coverage calculation, each PDF is modeled as a subset of the nets in the circuit and the PDFs detected by vectors are stored in the set nonenumeratively. At the end of the calculation, the number of stored subsets in the set would give the total number of covered PDFs.

ZBDDs represent sets efficiently [4] and were successfully applied to nonenumerative PDF coverage calculation in [6]. For example, Figure 3 depicts three ZBDDs: the leftmost holds the PDFs detected by v_1, the middle one holds the PDFs detected by v_2, and the rightmost, which is the union of two ZBDDs, holds the PDFs detected by both vectors compactly (that is sharing subpaths $16 \cdot 22$ and $16 \cdot 23$).

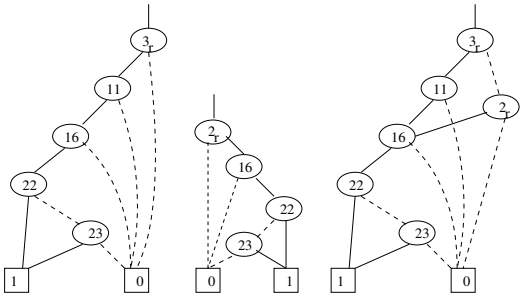


(a) PDFs Detected by v_1



(b) PDFs Detected by v_2

Figure 2: PDFs Detected by v_1 and v_2



(a) ZBBD for v_1 (b) ZBDD for v_2 (c) Union of two ZBDDs

Figure 3: $ZBDD_{v_1} \cup ZBDD_{v_2} = ZBDD_{v_1,v_2}$

3 Variable Ordering Heuristics

In many applications of ZBDDs, determining a good initial variable order is important. Ordering techniques try to deduce *a priori* information from some other representation to get a good variable ordering. It is recommended that (1) the variables with bigger influence on the function, in our case nets, should be placed high in the order and (2) the variables with similar influences, in our case topological closeness, should be grouped together. Using this approach, we propose variable ordering methods using circuit topology and path-count statistics.

The forward- and backward-path counts are illustrated in Figure 4. In forward path counting, the PIs are set to 1, and the path counts at each node are computed from their input path counts. Likewise, the backward path count algorithm sets the POs to 1 and copies the path count at a node to its inputs; a stem's path count is computed by summing the path counts at its branches. A frequency of a node is obtained by multiplying forward and backward path count values on it.

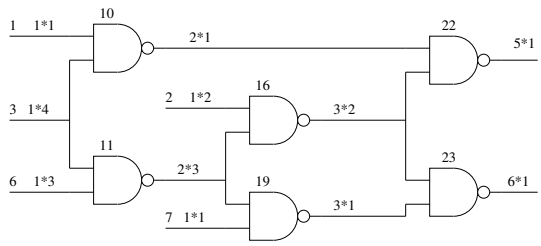


Figure 4: Maximum Frequencies of Variables in c17

The following heuristics are proposed to order variables in ZBDDs for PDF coverage calculation:

- *Variable Frequency* (H_1): Frequency of a variable is defined to be the total number of paths passing through this variable (Figure 4). This heuristic orders variables according to their maximum frequencies and puts the highest frequency variable at the root of ZBDD. When there are variables with equal frequencies, priority is given to the variable at the lower level. In case the variables are at the same level, one of them is picked arbitrarily. For c17, one valid ordering is $< G_{11}, G_{16}, G_{23}, G_{22}, G_3, G_6, G_{19}, G_2, G_{10}, G_1, G_7 >$.

- *Level-and-topology* (H_2) This heuristic orders variables according to their levels starting from the primary inputs (see Figure 1). Given *a well-drawn circuit* [3], variables at the same level are ordered based on circuit topology. For every alternating level, variables are ordered from left-to-right and right-to-left within level. For c17, one valid ordering is $< G_7, G_6, G_3, G_2, G_1, G_{10}, G_{11}, G_{19}, G_{16}, G_{22}, G_{23} >$.

- *Level-and-frequency* (H_3): Similar to H2 with the exception that the variables at the same level are ordered according to their frequencies and for equal frequencies the priority is given to the variable which topologically comes first. For c17, one valid ordering is $< G_3, G_6, G_2, G_1, G_7, G_{11}, G_{10}, G_{16}, G_{19}, G_{23}, G_{22} >$.

- *Backward-Variable Frequency* (H_4): In this heuristic, backward path count algorithm is run on the circuit and the variables are ordered in increasing order according to their path counts. Level and topology information is used to break ties among the equal path counts. For c17, one valid ordering is $< G_3, G_6, G_{11}, G_2, G_{16}, G_1, G_7, G_{10}, G_{19}, G_{23}, G_{22} >$.

- *Forward-Variable Frequency* (H_5): Similar to H_4 with the exception that forward path counts are used in place of backwards and also the variables are ordered from lowest to highest; the lowest one is at the root. Heuristics H_4 and H_5 are derived from the ordering heuristic in BDDs [5]. Our heuristic uses paths counts as weights, which are different from their weights. For c17, one valid ordering is $< G_1, G_2, G_3, G_6, G_7, G_{10}, G_{11}, G_{19}, G_{16}, G_{22}, G_{23} >$.

	TotalPaths	Paths	Variables	Vectors	SimTime
c880	17284	1957	503	17768	1.14
c1355	8346432	1119	628	14143	1.30
c1908	1458114	4303	946	15120	2.22
c2670	1359920	9662	1769	25983	13.77
c3540	57353342	9662	1769	25983	13.77
c5315	2682610	8637	2663	20345	23.45
c6288	1.978×10^{20}	1576	2480	37881	43.09
c7552	1452988	9024	3926	25378	43.70

Table 2: Benchmark Parameters

4 Experimental Results

In this work we performed two sets of experiments on a P4 - 2.4 GHz, 1GB RAM, WindowsXP system for the aforementioned heuristics. In our work, we used the CUDD Package Release 2.3.1 [7] that supports the ZBDD data structure. In the first experiment, called *Limited-PDF*, a test set of 10K vectors that detect at least one path is randomly generated. Then the paths that are detected by test vectors are added to the ZBDD structure one by one. In the second experiment, called *All-PDF*, all PDFs are stored in the ZBDD. This gives a theoretical upper bound on the size of ZBDD when all PDFs are stored in it.

Table 2 presents the number of all paths in the benchmark circuits, the detected paths by a test set, the total number of variables in the ZBDD, the number of random vectors generated to obtain 10K useful vectors, and the simulation time. The simulation time is the time spent to verify the randomly generated vectors.

Table 3 tabulates the performances of the proposed heuristics in terms of final, peak node counts and build times for the *Limited-PDF* case. We observe that $H1$ is the worst in final node counts and at least 1.48 times larger than $H2 - H5$ for all benchmarks. $H2 - H5$ are very close to each other in their final node counts. In terms of peak node counts, $H5$ gives the consistent size for all benchmarks. Ignoring the first two small benchmarks, $H5$ gives the smallest peak node counts for all benchmarks. $H5$ is at least 1.11 times smaller than $H1 - H4$ for all benchmarks. However, $H5$ results in the slowest build time.

Table 4 tabulates the performances of the proposed heuristics in terms of final, peak node counts and build times for the *All-PDF* case. We observe that $H2$ and $H3$ are very close in their final node counts. In terms of peak node counts, $H5$ is at least 6.7 times smaller than $H1 - H4$ for all benchmarks. In this experiment, $H5$ results in the slowest build time, like the Limited-PDF case.

The authors of [6] reported the final node count of the ZBDD when all PDFs were stored in it. Their reported results and its comparison with our *ALL-PDF's*

Name	H1	H2	H3	H4	H5
c880	5755/41902/2.2	2328/74606/1.9	2281/58254/1.7	2180/58254/1.9	2182/30660/2.1
c1355	8908/39858/2.2	4498/94024/1.5	4489/90958/1.5	4489/90958/1.6	4413/35770/2.8
c1908	14400/71540/7.2	7793/98112/5.0	7521/96068/4.9	7475/96098/5.7	8676/49056/6.4
c2670	6591/60298/11.7	3942/205422/8.3	4099/159432/8.2	3946/158410/11.8	4451/38836/12.8
c3540	49789/134904/19.0	15319/156366/10.0	15275/158410/10.1	15113/157388/14.0	15031/62342/19.9
c5315	24026/123662/38.0	12289/300468/20.4	12753/143080/18.4	12185/140014/29.6	13049/74606/40.4
c6288	7709/68474/30.2	3963/45990/14.2	3954/52122/14.5	3961/52122/18.2	3798/34748/38.4
c7552	27940/126728/64.3	14015/352590/30.3	14181/202356/28.6	13905/200312/44.6	15576/67452/80.3

Table 3: Limited-PDF: (Final Nodes/Peak Nodes/Build Time (sec)) in ZBDD Manager

Name	H1	H2	H3	H4	H5
c880	16491/136948/0.9	680/50078/0.1	716/51100/0.1	687/50078/0.3	722/6132/1.0
c1355	99518/2343446/8.6	937/256522/0.5	937/258566/0.5	937/257544/0.6	956/8176/1.6
c1908	84124/757302/4.5	1177/265720/0.4	1245/270830/0.5	1189/261632/1.3	1853/10220/3.4
c2670	25845/158410/7.4	1873/165564/0.5	2801/169652/0.9	1870/162498/4.6	2296/16352/8.8
c3540	2767923/8359960/38.2	2255/620354/1.6	2326/579474/2.0	2291/627508/5.5	3203/20440/12.3
c5315	51928/307622/21.8	3691/255500/0.8	4105/300468/1.6	3749/245280/13.1	4642/36792/26.1
c6288	memout	4143/2464042/12.2	4310/2529450/12.4	4142/2466086/16.0	4425/29638/25.2
c7552	122595/736862/42.3	5030/597870/1.5	5461/528374/3.2	5016/499758/19.2	6867/48034/57.0

Table 4: All-PDF: (Final Nodes/Peak Nodes/Build Time (sec)) in ZBDD Manager

Name	#PDFs	#Nodes	Time	#Nodes Ratio	Time Ratio
c880	17,284	9,708	1.32	14.28	13.2
c1355	8,346,432	122,107	8.91	130.32	17.82
c1908	1,458,112	79,878	8.48	67.87	21.2
c2670	1,359,756	53,529	6.10	28.58	12.2
c3540	56,531,748	155,649	39.44	69.02	24.65
c5315	2,682,610	109,815	8.28	29.75	10.35
c6288	$1{,}978\times10^{20}$	1,917,260	234.56	462.77	19.23
c7552	1,452,986	131,017	22.70	26.05	15.13

Table 5: Final nodes in ZBDD (excerpted from [6])

H2 are tabulated in 5. Note that $H2$ outperforms their static ordering heuristic significantly both in final node counts and build times in all benchmarks. Especially, for the problematic circuit c6288, our $H2$ yields 463 times smaller final node counts than their static heuristic.

5 Conclusion

ZBDDs represent sets efficiently and they have recently been used in *nonenumerative* path delay fault (PDF) coverage calculations. Each PDF is modeled as a subset of all nets in a circuit, and all detected PDFs are stored in a ZBDD compactly. Many heuristics have been proposed to order variables of ZBDDs to avoid size explosion when those structures are used to represent circuit functionality. However, in ZBDD-based PDF coverage calculations, the variables represent the nets in a circuit and are thus structural representations of the circuit. This motivates us to search for new ordering techniques that may take advantage of structural properties. We devised several new ordering heuristics. We found that the new $H2, H3, H4$ and $H5$ heuristics perform well. $H1$ clearly under-performed as compared to the other techniques. We also compared our results with published results. With the problematic benchmark circuit c6288 we found that our $H2$ heuristic resulted in a ZBDD that was 463 times smaller.

References

[1] Bushnell, M. L., and Agrawal, V. D. *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*. Kluwer Academic Publishers, 2000.

[2] Gharaybeh, M. A., Bushnell, M. L., and Agrawal, V. D. The path-status graph with application to delay fault simulation. *IEEE Trans. on CAD of Integrated Circuits and Systems 17*, 4 (September 1998), 324–332.

[3] Jain, J., Adams, W., and Fujita, M. Sampling schemes for computing obbd variable orderings. In *Proc. of IEEE/ACM Int'l Conf. on CAD* (1998), pp. 631–637.

[4] Minato, S. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proc. of ACM/IEEE Design Automation Conference* (1993), pp. 272–277.

[5] Minato, S., Ishiura, N., and Yajima, S. Shared binary decision diagram with attributed edges for efficient boolean function manipulation. In *Proc. of ACM/IEEE Design Automation Conference* (1990), pp. 52–57.

[6] Padmanaban, S., Michael, M. K., and Tragoudas, S. Exact path delay fault coverage with fundamental ZBDD operations. *IEEE Trans. on CAD of Integrated Circuits and Systems 22*, 3 (March 2003), 305–316.

[7] Somenzi, F. CUDD: CU decision diagram package. In *Deparment of Eletrical and Computer Engineering, Univ. of Colorado, Boulder* (2003).