Iterative OPDD Based Signal Probability Calculation

Avijit Dutta and Nur A. Touba

Computer Engineering Research Center
University of Texas, Austin, TX 78712

Abstract

This paper presents an improved method to accurately estimate signal probabilities using ordered partial decision diagrams (OPDDs) [Kodavarti 93] for partial representation of the functions at the circuit lines. OPDDs which are limited to a certain maximum number of nodes are built iteratively with different variable orderings to efficiently explore different regions of the function. Signal probability bounds (upper and lower) are computed from the OPDDs. From each OPDD, information is extracted to tighten the signal probability bound and guide the variable ordering for the next OPDD. By restricting the size of each OPDD to a small number of nodes, they can be constructed and processed quickly to obtain a fast and accurate estimate of signal probabilities. Experimental results demonstrate the effectiveness of the approach compared with existing methods.

1. Introduction

The signal probability of a net in a combinational circuit is the probability that a randomly generated input vector will produce a logic value of 1 on this net. There are a number of important applications where calculating signal probabilities in a circuit are necessary. Originally signal probability was studied in the context of pseudo-random testing to determine detection probabilities for faults. Given the detection probabilities for faults in a circuit, it is possible to compute the expected fault coverage for a particular pseudo-random pattern test length and to identify random pattern resistant faults [McCluskey 88]. More recently, as soft errors in logic circuits have become an important issue, signal probability is also needed in this context for determining the probability of a single event upset (SEU) propagating to a latch. Knowing the soft error susceptibility of nodes in a circuit allows better insertion of soft error protection schemes and better selection of error detecting codes.

For circuits that do not have reconvergent fanout, signal probabilities can be computed exactly in linear time. However, in the general case where reconvergent fanout exists, computing signal probabilities is NP-hard [Parker 75]. A wide variety of techniques have been developed for estimating signal probabilities which provide varying degrees of accuracy and runtime. A fast and simple approach, used in COP [Brglez 84], is to assume all signals in the circuit are independent, however, this can lead to large inaccuracies due to correlations between signals from reconvergent fanout. COP can be improved by estimating the impact of correlations using cofactors [Al-Kharji 97] and first order Taylor expansion [Uchino 95]. Partitioning the circuit into "supergates" which totally enclose reconvergent fanout can be used to speedup signal probability calculations [Seth 85, 89], [Chakravarty 90]. Another approach for estimating signal probabilities is to use sampling simulation [Jain 85], [Wunderlich 85], [Rejimon 05].

One class of techniques computes signal probability bounds (upper and lower) which has the nice property of not only estimating signal probability, but also bounding the maximum error in the estimate. One such technique is the "cutting algorithm" [Savir 80] which cuts fanout lines in the circuit to make it fanout-free and then assigns a probability bound of [0,1] to the cut-lines. Techniques for tightening the bounds obtained with the cutting algorithm include [Markowsky 87] which uses a blocking heuristic to reduce the number of cuts, [Savir 90] which combines it with the Parker-McCluskey method [Parker 75], and [Kapur 92] which uses conditional probabilities to tighten the initial bounds on the cut lines. Another technique for computing signal probability bounds is to use ordered partial decision diagrams (OPDDs) as described in [Kodavarti 93]. If the full BDD [Bryant 86] was known, then exact signal probabilities could be computed by counting the paths that go to the terminal 1 node [Zeng 03]. However, constructing a full BDD can be exponential in the number of inputs and thus is not practical in many cases. The idea in [Kodavarti 93] is to construct an OPDD which is limited to a certain maximum number of nodes (thereby limiting both time and memory). A bound on the signal probability is then obtained from the OPDD which contains the efficiently obtainable implicants (which typically includes the largest ones). By using a few different variable orderings and saving the best upper and lower bound seen for any ordering, it was shown in [Kodavarti 93] that the signal probability could be computed accurately in very short time.

Proceedings of the 24th IEEE VLSI Test Symposium (VTS'06)
0-7695-2514-8/06 $20.00 © 2006 IEEE

This paper presents a new method for using OPDDs to estimate signal probability that gives significantly tighter bounds than the method in [Kodavarti 93]. There are two key ideas in the proposed method. The first is to constructively combine information obtained in one OPDD with the next. In [Kodavarti 93], the probability bounds are computed independently for each OPDD and then the best upper bound across all OPDDs is combined with the best lower bound across all OPDDs to form the final bound. In the proposed method, the implicants from each OPDD are extracted and combined together when computing the final upper and lower bound. Double counting is avoided by making the implicants disjoint. The second key idea is to use the information from previous OPDDs to guide the selection of the variable ordering for the next OPDD. Here a heuristic algorithm for selecting the variable orderings to efficiently explore the unknown space is presented. The proposed variable ordering algorithm uses information about the implicants found so far to help in finding new implicants to tighten the bounds further. Experimental results are shown which demonstrate the effectiveness of the proposed method.

Note that there has been some work in the domain of model-checking that uses over-approximating and under-approximating of BDDs [Ravi 98]. However, this is fundamentally different than what is done here because it involves first building the full BDD and then compressing it by discarding nodes in a deterministic manner. What is done here is to avoid building a full BDD, but rather iteratively build small limited-size partial BDDs that can be processed very quickly with no risk of "blowing up."

2. Combining Information Across OPDDs

OPDDs are a variant of ordered binary decision diagrams (OBDD) introduced in [Brayant 86]. Partial information is obtained by restricting the number of nodes when building the graph to a constant k. The missing information is represented by an UNKNOWN (U) terminal node. Figure 1 shows the full OBDD representing the function: *ab'+ac'+b'c*. The 0-arcs are represented with dashed lines and the 1-arcs are represented with solid lines. Figure 2 through 4 shows corresponding OPDDs for various variable orderings. An OPDD can have 3 terminal nodes namely the 0-node, 1-node, and U-node. Variable ordering plays an important role in the amount of information that can be gathered from an OPDD. OPDDs can be built with different variable orderings to explore different regions of the function. From an OPDD, the 0-probability, 1-probability, or U-probability can be computed as described in [Kodavarti 93]:

1) Initialize the sum S_r of the root node n_r to $S_r = 1$.
2) Initialize the sum S_i of all nodes n_i, for all $i \neq r$, to $S_i = 0$.
3) Associate a line probability (p_i) for each circuit input x_i, for all i.
4) At every non-terminal node n_i, representing input variable x_i, perform two operations (during breadth first traversal of the OPDD):
   Add $(1-p_i)*S_i$ into the sum of the 0-arc child
   Add p_i*S_i into sum of the 1-arc child
5) Calculate the lower bound of the signal probability from the final value of the sum, S_1, at the ONE terminus, and the upper bound including the sum, S_U, at the UNKNOWN terminus.
   Lower bound = S_1
   Upper bound = S_1+S_U

One limitation of the approach in [Kodavarti 93] is that the information is not retained across the OPDDs. The proposed algorithm maintains a global disjoint cube cover across different OPDDs to obtain tighter bounds on the signal probability.

Note that for the rest of the paper, 1-path, 0-path, and U-path will denote paths terminating at 1-terminus, 0-terminus, and U-terminus, respectively. The corresponding cubes will be denoted as 1-cube, 0-cube, and U-cube, respectively. The 1-paths in the OPDD encode cubes covering some part of the ON-set of the function represented at the root node. The 0-paths do the same for the OFF-set. By construction of the OPDD these cubes are disjoint. By constructing OPDDs with different variable orderings, different regions of the function can be explored. From each OPDD, disjoint cubes are collected and a global list of such cubes is maintained both for the OFF-set (*g0-cov*) and the ON-set (*g1-cov*). The cube list is maintained as a sorted list in decreasing order of the cube sizes. The larger the number of don't cares (X's), the larger is the cube and the larger is its contribution to the signal probability of the line. While adding a new cube to the current global cube cover, three things are checked:

1) If it is contained in one of the existing cubes then it is not added. This is checked by iterating over all the bits of the cubes. If the smaller cube matches with the larger cube at all the specified bit positions then it is contained in the larger cube.
2) If it is mutually disjoint with each of the existing cubes, then it is simply added to the list and the sorted order is maintained. This can be easily checked by looking for a conflict in any of the specified bit position of the cube to be added with each of the existing cubes in the cover.
3) If the cube to be added overlaps with one or more of the existing cubes in the cover then the cube is made disjoint and added to the list. This is done the following way. If two cubes have overlap then the cubes are traversed bit by bit and for the first bit position where one of the cubes has a specified bit and the other cube is unspecified, the

unspecified bit is specified with the opposite value of the specified bit value of the other cube.

At the end of all the iterations, the final lower bound on the 1-probability is computed directly from the global disjoint cube cover (*g1*-cov) by simply adding the probabilities of the individual disjoint cubes. Since they are disjoint, the probabilities are independent and can simply be summed together. The ambiguity *u* is computed as [*u=1 − (g0+g1)*] where *g0* and *g1* are the probabilities computed from the *g0*-cov and *g1*-cov respectively. The final upper bound is equal to the lower bound plus the ambiguity *u*. Accuracy versus runtime can easily be traded off in selecting the OPDD size limit as well as the number of OPDDs that are built. The proposed approach can be used with any set of OPDDs derived with any variable ordering. However, in the next section, a heuristic approach for guiding the selection of the variable ordering to find better cubes for the global cover is described.

The example in Figs. 2-4 illustrates the proposed approach on a very small example. Figure 1 shows the OBDD representation of the function *ab'+ac'+b'c*. The complete ON-set and OFF-set of the function are shown on a Karnaugh map. Figure 2 shows the OPDD with variable ordering *<a,b,c>* where the bound on the maximum number of non-terminal nodes is 3. After the first OPDD, *g0*-cov contains *{01x}* and *g1*-cov contains *{10x}*. The bound on signal probability computed from the OPDD in Fig. 2 using the method in [Kodavarti 93] is *[.25,.75]*. The OPDD in Fig. 3 uses the variable ordering *<b,a,c>*. No new cubes are added to either *g0*-cov or *g1*-cov as the cubes found are already present in the respective sets. The third iteration uses the variable ordering *<a,c,b>*. After the third iteration, one new 1-cube *(1x0)* is found. This is made disjoint with the existing cube in *g1-cov* and is added to *g1-cov*. The contents of *g1*-cov after this iteration is *{10x,110}*. The probability computed from this set is *(.25+.125) = .375*. Similarly the contents of *g0-cov* after the third iteration is *{01x,000}*. The probability computed from *g0-cov* is *(.25+.125) = .375*. The ambiguity after third iteration is computed as *[1 – (.375+.375)] = .250*. So finally the estimated signal probability bound after the third iteration would be *[g1,g1+u] = [.375,.625]*. The actual signal probability in this case is *0.5*. Note that if we don't use the global cube covers to learn across multiple OPDDs then the best bound that can be obtained after third iteration under the same variable orderings is *[.25,.75]*.

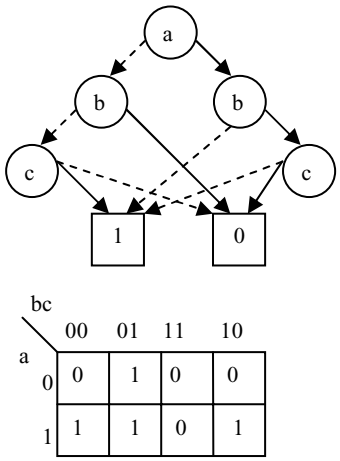

Figure 1. Binary Decision Diagram and Karnaugh Map

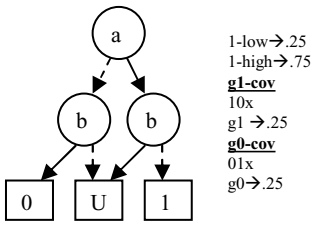

Figure 2. OPDD with variable ordering *<a,b,c>*

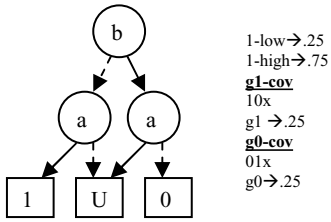

Figure 3. OPDD with variable ordering *<b,a,c>*

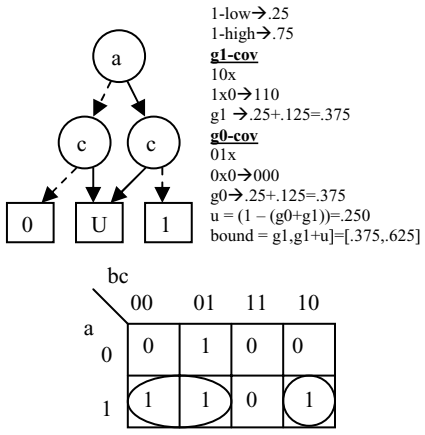

Figure 4. OPDD with variable ordering *<a,c,b>*

3. Unknown Space Exploration

While building an OPDD, whenever the number of nodes exceeds the predefined bound on the number of the nodes, the unknown *U* terminus is created and all paths are directed to it. The number of paths and the length of paths ending at the *U* terminus determine the size of the unknown space and hence the ambiguity in the signal probability. The variable ordering used when building an OPDD has a big impact on the resulting composition of the unknown space. The proposed method involves iteratively building multiple OPDDs and extracting collective information, and thus the goal in selecting the variable ordering for each additional OPDD is to try to explore the unknown space from the previous set of OPDDs. Note that the OPDDs are built for each signal. The gate input orderings corresponding to a variable ordering are computed once for the whole circuit.

One common approach for variable ordering is to perform a depth first search (DFS) of the circuit from primary outputs (POs) to primary inputs (PIs) and append a PI to the ordering as soon as it is traversed. When performing a DFS, a decision has to be made at each gate as to in what order its inputs should be traversed. A number of different heuristics have been developed for making this decision (e.g., [Malik 88] and [Fujita 93]). The conventional heuristics for guiding the DFS are targeting the problem of minimizing the overall size of a full BDD. These conventional heuristics are very useful in forming the first two OPDDs as they are likely to result in identifying the largest implicants (in our experiments the heuristic in [Fujita 93] was used). However, after information has been extracted from the first two OPDDs using the proposed methodology, the usefulness of the conventional heuristics for variable ordering when building subsequent OPDDs diminishes because they are more likely to explore the space of the function that has already been explored in the previous OPDDs. Here we propose a new heuristic to guide the DFS so that it will lead to a variable ordering that more effectively explores the unknown space.

The idea behind the proposed heuristic is to try to measure how much of the unknown space has been explored with respect to each variable across all the OPDDs built so far, and then direct the DFS towards the variables for which the unknown space has been least explored. The *U-effect* of a variable is defined as the sum of all the U-paths in which the variable appears in either complemented or uncomplemented form weighted by the size of the corresponding U-cube for the U-path. Thus, the U-effect of a variable is computed as $\sum(2^{n-k_i})$ for each U-path i in which the variable appears, where k_i is the number of nodes along the path, and n is the number of primary inputs of the circuit. The U-effect of the variables for the OPDD in Fig. 2 is shown in Table 1. There are two U-paths each of which include variables *a* and *b*. The size of the U-cube corresponding to each path is 2, and thus the U-effect for *a* and *b* is 4. Since variable *c* does not appear on any U-paths, its U-effect is 0.

Table 1. U-Effect for OPDD in Fig. 2

Variable	U-Effect
A	2+2=4
B	2+2=4
C	0

As each new OPDD is constructed, the U-effect for each variable is computed and added to the U-effect of the previous OPDDs. Thus, a running total of the U-effect over all the OPDDs is maintained for each variable. The running total of the U-effect over the three OPDDs in Figs. 2-4 is shown in Table 2.

Table 2. Cumulative U-Effect for OPDDs in Figs. 2-4

Variable	U-Effect
A	12
B	8
C	4

The cumulative U-effect for a set of OPDDs gives a rough measure of how well the unknown space has been explored with respect to each variable. This is then used as a heuristic to direct the DFS towards the variables that have the lowest U-effect. This helps to explore the least explored region of the function and hopefully construct an OPDD in which new implicants can be obtained to further reduce ambiguity in the signal probability calculations. This is illustrated in the small example in Fig. 5 where using the U-effects from Table 2 result in a new ordering <*c,b,a*> for which the OPDD built as shown in Fig. 5. For the OPDD in Fig. 5, the U-space vanishes and the exact probabilities are obtained.

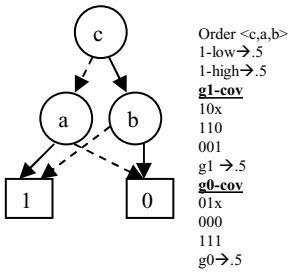

Figure 5. OPDD with variable ordering <*c,b,a*>

When searching the unknown space using the U-effect heuristic, the DFS is modified so that the decision as to which order in which to traverse the inputs of a

gate are made based on the support set for each input. The support set for an input is the set of PIs (variables) that it depends on. The U-effect of each variable in the support set is summed together, and the inputs of the gate are traversed in reverse order of their total U-effect.

So the overall strategy for iteratively building the OPDDs is the following. Conventional heuristics for variable ordering are used to build the first two OPDDs to identify large implicants, and then the proposed heuristic of using the U-effect is used when building subsequent OPDDs to more effectively search the unknown space.

4. Runtime Complexity Analysis

In this section, the runtime complexity for the proposed method is analyzed and compared with [Kodavarti 93]. The complexity is described below in terms of the following: number of gates in the circuit (G), number of primary inputs in the circuit (N), limit on the maximum number of nodes in the OPDD (B), and number of iterations used (I) where one OPDD is built in each iteration. Note that B and I are actually constants that do not scale with circuit size. They are shown in the equations below just for completeness.

Compute Variable Ordering – Initially this is done with conventional heuristics which is the same as for [Kodavarti 93]. The complexity depends on which heuristics are used, but good results for DFS based methods can be obtained in linear time in the number of gates. Thus the overall complexity is $O(GI)$ since it needs to be done for each OPDD iteration. In the proposed method, after the first two OPDDs, then the U-effect heuristic described in Sec. 3 is used. The U-effect for each variable can be computed by traversing each OPDD which is $O(BI)$ and then it needs to be sorted which is $O(Nlog(N)I)$. The support set for each line in the circuit can be obtained in $O(G)$ and needs to be computed only once and stored. So the overall complexity for DFS with the U-effect heuristic is $O(BI+Nlog(N)I+G)$.

Build OPDDs – Building the OPDDs is $O(GB^2I)$ which is the same as for [Kodavarti 93].

Construct Global Disjoint Cover – This is unique to the proposed method. Note that the number of cubes cannot be larger than the number of nodes in the OPDDs. Hence this can be done in $O(NGB^2I^2)$. Note that these are very fast bitwise comparison operations.

Calculate Signal Probability Range – This is just a matter of adding the size of all the disjoint cubes. For both [Kodavarti 93] and the proposed method, this is $O(GBI)$.

As mentioned before, B which is the node limit for the OPDDs and I which is the number of iterations are constants that do not scale with circuit size. So if only the factors that scale with circuit size are considered, then the overall complexity for the proposed method is $O(NG)$ compared with $O(G)$ for [Kodavarti 93]. For realistic industrial circuits, the extra N factor is not very significant for two reasons. One is that while in the worst case the number of inputs in the largest cone of logic could theoretically be equal to N, typically it is relatively small and doesn't scale up much for larger designs (it depends mostly on the function being implemented), and the other reason is that the extra complexity is coming from the bitwise comparison operations for the cubes which can be done very quickly. Consequently, the actual runtimes for the two methods are very similar. Note that when processing a large hierarchical design, the design can be easily partitioned and spread over multiple processors.

5. Experimental Results

The proposed approach is specifically useful for large circuits where building the full BDD is not possible. However, for comparison with previously published results, experiments were performed on the ISCAS-85 benchmark circuits even though it is practical to build a full BDD for most of those circuits with 4000 nodes. In [Kodavarti 93], results were reported for using a limit of 500 nodes in the OPDDs with 4 iterations (i.e., building 4 OPDDs with different variable orderings). These results from [Kodavarti 93] are shown in Table 2 along with the results reported in [Kodavarti 93] for the cutting algorithm which are shown in Table 1. Experiments were performed using the proposed method with the same parameters, namely a 500 node limit for each OPDD and 4 iterations. The results for the proposed method are shown in Table 3. In each of these tables, the number of lines with different ranges of ambiguity between 0% and 100% are shown. As can be seen from the results, the proposed method is able to reduce the amount of ambiguity in the signal probability ranges considerably in all the circuits compared with [Kodavarti 93].

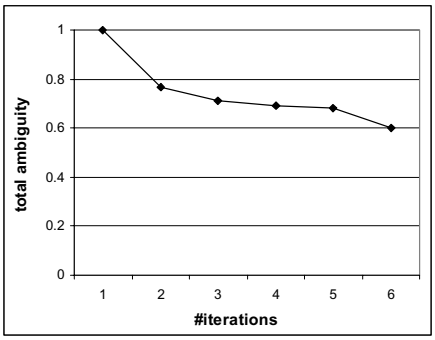

Figure 6. Total Ambiguity (Normalized) vs. Number of Iterations for *C880* (200 Node Limit)

Further experiments were performed to see how the results varied with the number of iterations. Fig. 6 shows results for *C880* where the total ambiguity normalized with respect to the first iteration is shown on the y-axis and the number of iterations is shown on the x-axis. A 200 node limit was used for the OPDDs. As the number of iterations increases, the ambiguity decreases. There tends to be a diminishing marginal return, however, and the amount of improvement from one iteration to the next can vary due to the fact that heuristics are used.

Table 1. Ambiguity for Cutting Algorithm

Circuit Name	Ambiguity Ranges (in %)							
	Total Lines	0	> 0 ≤ 30	>30 ≤ 50	> 30 ≤ 50	> 30 ≤ 50	> 30 ≤ 50	100
C432	160	27	27	27	0	0	6	73
C499	202	13	0	7	20	5	18	139
C880	383	19	9	45	23	0	0	287
C1355	546	9	1	30	8	0	0	498
C1908	880	52	0	37	4	0	0	787
C2670	1269	154	6	136	38	5	55	878
C3540	1669	53	0	53	14	1	0	1548
C5315	2307	137	5	154	28	0	0	1893
C6288	2416	3	0	58	1	0	0	2354
C7552	3513	120	5	190	91	0	0	3107

Table 2. Ambiguity for [Kodavarti 93] with 500 Nodes and 4 Iterations

Circuit Name	Ambiguity Ranges (in %)							
	Total Lines	0	> 0 ≤ 30	>30 ≤ 50	> 30 ≤ 50	> 30 ≤ 50	> 30 ≤ 50	100
C432	160	94	31	14	16	5	0	0
C499	202	137	33	32	0	0	0	0
C880	383	354	29	0	0	0	0	0
C1355	546	338	143	64	0	1	0	0
C1908	880	731	149	0	0	0	0	0
C2670	1269	1237	24	2	6	0	0	0
C3540	1669	1261	119	80	179	28	2	0
C5315	2307	2227	66	6	6	0	0	0
C6288	2416	1281	394	36	231	100	310	64
C7552	3513	3387	112	8	6	0	0	0

Table 3. Ambiguity for Proposed Approach with 500 Nodes and 4 Iterations

Circuit Name	Ambiguity Ranges (in %)							
	Total Lines	0	> 0 ≤ 30	>30 ≤ 50	> 30 ≤ 50	> 30 ≤ 50	> 30 ≤ 50	100
C432	160	158	2	0	0	0	0	0
C499	202	193	7	2	0	0	0	0
C880	383	360	23	0	0	0	0	0
C1355	546	436	102	8	0	0	0	0
C1908	880	816	64	0	0	0	0	0
C2670	1269	1240	21	8	0	0	0	0
C3540	1669	1484	87	98	0	0	0	0
C5315	2307	2297	10	0	0	0	0	0
C6288	2416	1292	383	68	224	114	272	63
C7552	3513	3408	99	2	4	0	0	0

5. Conclusions

The proposed approach provides a means to obtain tighter signal probability ranges by extracting more information from OPDDs. It can be used in estimating soft error susceptibility and random pattern testability.

Acknowledgements

This research was supported in part by the National Science Foundation under Grant No. CCR-0426608.

References

[Al-Kharji 97] Al-Kharji, M.A., S.A. Al-Arian, "A New Heuristic Algorithm for Estimating Signal and Detection Probabilities," *Proc. of Great Lakes Symposium on VLSI*, pp. 26-31, 1997.

[Brglez 84] Brglez, F., "On Testability Analysis of Combinational Networks," *Proc. Symp Circuits and Sys.*, pp. 221-225, 1984.

[Bryant 86] Bryant, R. E. "Graph Based Algorithms for Boolean Function Manipulation," *IEEE Trans. on Computers*, Vol C-35, No. 8, pp. 677-691, Aug. 1986.

[Chakravarty 90] Chakravarty, S., and H. Hunt III, "On Computing Signal Probability and Detection Probability of Stuck-at Faults," *IEEE Trans. on Comp.*, Vol. 39, pp. 1369-1377, Nov. 1990.

[Fujita 93] Fujita, M., H. Fujisawa, and Y. Matsunaga, "Variable Ordering Algorithms for Ordered Binary Decision Diagrams and Their Evalulation," *IEEE Trans. on CAD*, pp. 6-12, Jan 1993.

[Jain 85] Jain S.K., and V.D. Agrawal, "Statistical Fault Analysis," *IEEE Design & Test of Computers*, pp. 38-45, 1985.

[Kapur 92] Kapur, R., and M.R. Mercer, "Bounding Signal Probabilities for Testability Measurement using Conditional Syndromes", *IEEE Trans. Comp.*, pp. 1580-1588, Dec. 1992.

[Kodavarti 93] Kodavarti, R., and D. Ross," Signal Probability Calculations Using Partial Functional Manipulations," *Proc. of VLSI Test Symposium*, pp. 194-200, 1993.

[Malik 88] Malik, S., A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli, "Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment," *Proc. of Int. Conf. on Computer Aided Design (ICCAD)*, pp. 6-9, 1988.

[Markowsky 87] Markowsky, G., "Bounding Signal probabilities in Combinational Circuits," *IEEE Trans. on Computers*, Vol C-36**,** No. 10, pp. 1247-1251, Oct. 1987.

[McCluskey 88] McCluskey, E.J., S. Makar, S. Mourad, K. Wagner, "Probability Models for Pseudorandom Test Sequences," *IEEE Trans. on CAD*, Vol. 7, No. 1, pp. 68-74, Jan. 1988.

[Parker 75] Parker, K. P., and E. J. McCluskey, "Probabilistic Treatment of General Combinational Networks," *IEEE Trans. on Computers*, pp. 668-670, 1975.

[Ravi 98] Ravi, K., K.L. McMillan, T.R. Shiple and F. Somenzi., "Approximation and Decomposition of Binary Decision Diagrams," *Proc. Design Autom. Conf.*, pp. 445-450, 1998.

[Rejimon 05] Rejimon, T., and S. Bhania, "Time and Space Efficient Method for Accurate Computation of Error Detection Probabilities in VLSI Circuits," *IEE Proc. of Computers*, Vol. 152, Issue 5, pp. 679-685, Sep. 2005.

[Savir 80] Savir, J., G. S. Ditlow, P. H**.** Bardell, "Random Pattern Testability," *IEEE Trans. Comp.*, pp. 79-90, Jan. 1984.

[Savir 90] Savir, J. "Improved Cutting Algorithm," IBM *Jour. of Res. and Develop.,* Vol 34, pp. 40-75, March/May 1990.

[Seth 85] Seth, C., L. Pan and V. D. Agrawal, "PREDICT-Probabilistic Estimation of Digital Circuit Testability," *Proc. of Fault-Tolerant Computing Symposium*, pp.220-225, 1985.

[Seth 89] Seth, C., V. D. Agrawal, "A New Model for Computation of Probabilistic Observability," *Integration, the VLSI Journal*, Vol. 7, pp.49-75, 1989

[Uchino 95] Uchino, T., F. Minami, T. Mitsuhashi and N. Goto," Switching Activity Analysis using Boolean Approximation Method," *Proc. Int. Conf. on CAD,* pp.20-25, 1995.

[Wunderlich 85] Wunderlich, J., "PROTEST: A Tool for Probabilistic Analysis," *Proc. of DAC*, pp. 204-211, 1985.

[Zeng 03] Zeng, Z., Q. Zhang, I. Harris, and M. Ciesielski, "Fast Computation of Data Correlation Using BDDs", *Proc. of Design, Automation and Test in Europe*, pp. 122-127, 2003.