Random Pattern Testability of Circuits Derived from BDDs

Junhao Shi
Institute of Computer Science
University of Bremen
28359 Bremen, Germany
junhao@tzi.de

Görschwin Fey
Institute of Computer Science
University of Bremen
28359 Bremen, Germany
fey@tzi.de

Rolf Drechsler
Institute of Computer Science
University of Bremen
28359 Bremen, Germany
drechsler@tzi.de

Abstract

Design styles based on multiplexors have become very popular due to their good testability. Starting from a function description as a Binary Decision Diagram *(BDD) the circuit is generated by a linear time mapping algorithm. Only one additional input and one inverter are needed to achieve 100% testable circuits under the* Stuck-At Fault Model (SAFM). *Even though free of redundancies, the resulting circuits may contain hard to test faults, since the fault detection probability is very low.*

In this paper we present optimization techniques for circuits derived from BDDs such that the final circuit has good random pattern testability. The BDD ordering is optimized and the input probabilities are varied. The algorithms operate at a high level of abstraction and make use of the compact function representation as BDDs. Experiments are given to demonstrate the efficiency of the approach.

1. Introduction

Binary Decision Diagrams (BDDs) [9] have originally been proposed as a data structure for efficient Boolean function representation and manipulation. But due to their compactness they have also been frequently used in synthesis approaches. BDDs are well suited in this field since they can be used on a high level of abstraction as a pure function manipulation tool as it has been done in [20], but they can also be used in logic synthesis approaches (see e.g. [11]). BDD approaches are especially suitable for multiplexor based design styles, since the circuits can be directly derived from BDDs if each internal node is substituted by a multiplexor cell. Often these cells can be realized at very low cost, e.g. in *Pass Transistor Logic* (PTL). Furthermore, in BDD based design flows technology dependent information, like the layout, can be considered in an early stage and by this high quality results can be guaranteed (see e.g. [15,14]). Thus, they are a good data structure for a unified design flow that demands for efficient manipulation on the register transfer level and above.

One further important argument for the use of BDD circuits are testability aspects. Due to the structural restrictions of BDDs, i.e. the ordering of the variables, testability can be ensured by construction. Multiplexor circuits derived from BDDs have been studied intensively under various fault models [2,1,4,3]. For an overview see [5]. Recently in [10] a technique has been proposed that ensures 100% testability of circuits derived from BDDs under the SAFM and even the Path Delay Fault Model at the cost of one additional input. But even though the resulting circuits are free of redundancies, the resulting *Random Pattern Testability* (RPT) is often very poor, i.e. there are faults with a low detection probability. But a high detection probability is very important in today's system-on-chip test, since BIST patterns have to be applied and storing separate (deterministic) patterns for each module is too expensive [16].

In this paper we study the RPT of circuits derived from BDDs. Two approaches to improve the RPT of the circuits are presented. The first technique is to modify the variable ordering, such that the testability is improved. The second approach determines input probabilities from a set of discrete values. As a measure for the testability we use the minimal fault detection probability and the expected fault detection probability. The algorithms operate at a high level of abstraction and make use of the compact function representation as BDDs. Experimental results show that the testability can be significantly improved. In some cases the minimal fault detection probability could be improved from 10^{-7} to 10^{-3} by choosing the ordering and the input probability accordingly.

The paper is structured as follows: In Section 2 BDDs and BDD circuits are defined and the stuck-at fault model is briefly reviewed. The two approaches to improve the RPT of BDD circuits are presented in Section 3. Experimental results are given in Section 4, followed by the conclusions in Section 5.

2. Preliminaries

2.1. Binary Decision Diagrams

As is well-known a Boolean function $f : B^n \rightarrow B$ can be represented by a BDD which is a directed acyclic graph $G = (V, E)$ where a Shannon decomposition

$$f = \overline{x_i} f_{low(v)} + x_i f_{high(v)} \quad (1 \le i \le n)$$

is carried out in each node. A BDD is called *ordered* if each variable is encountered at most once on each path from the root to a terminal node and if the variables are encountered in the same order on all such paths. A BDD is called *reduced* if it does not contain isomorphic subgraphs nor does it have redundant nodes. Reduced and ordered BDDs are a canonical representation, i.e. for each Boolean function the BDD is uniquely specified [9]. In the following, we refer to reduced and ordered BDDs for brevity as BDDs.

It is well known that the BDD size depends on the chosen variable ordering. The most promising results regarding the quality of BDD minimization are based on dynamic reordering. In this context, S*ifting* as introduced in [17] is an approach offering a good compromise to trade off run time versus quality.

2.2. Output Probabilities in BDDs

Output probabilities of nodes can be calculated efficiently in BDDs. Let $G = (V, E)$ be the BDD representation with root u of the Boolean function g_u. Then, the output probability $P(g_u = 1)$ can be computed in time $O(|G|)$ using the recursive function $P : V \rightarrow [0,1]$ with $P(v) = value(v)$ if v is a terminal node and otherwise,

$$P(v) = P(x(v) = 0) \cdot (P(v_l) + P(x(v) = 1) \cdot P(v_h) \qquad (1)$$

where $x(v)$ is the variable checked at v and v_l and v_h are the low and high successors of v, respectively. The proof can be found in [6,12]. $P(v)$ is called the output probability of node v.

From Equation (1), it can be seen that $P(v)$ depends on $P(x(v) = 0)$ and $P(x(v) = 1)$. This means the signal probability for the primary inputs can change the output probability. This influence can be seen as follows:

Example 1 *Figure 1 demonstrates the application of the recursive calculation on a BDD representing* $f(x) = \overline{x_3}\,\overline{x_2} + x_1$. *All signal probabilities for the primary inputs are set to 0.5. Figure 2 illustrates the computation for the same function, but the signal probabilities of the inputs* x_1, x_2 *and* x_3 *are 0.8, 0.7 and 0.6, respectively.*

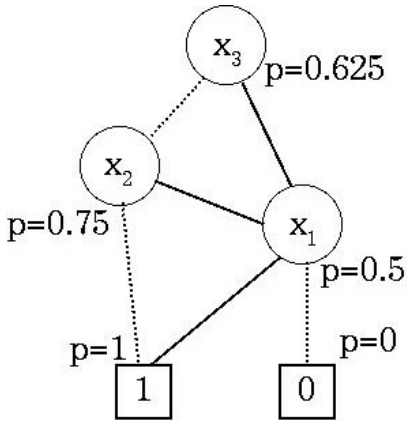

Figure 1. Equal input probabilities

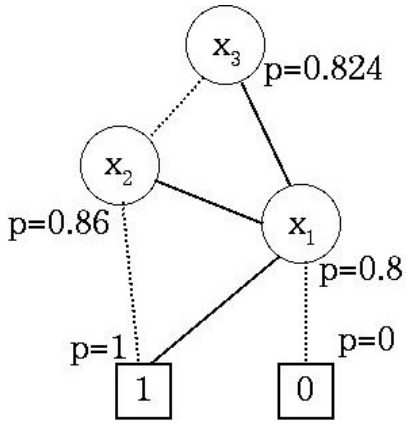

Figure 2. Different Input Probabilities

2.3. BDD Circuits

It is well-known, that BDDs directly correspond to multiplexor based Boolean circuits, called BDD circuits in this paper. More exactly: BDD circuits are combinational logic circuits defined over a fixed library. The typical multiplexor cell is denoted as MUX, and it is defined as given in Figure 3 by its standard *AND-*, *OR-*, *INVERTER-*based realization[1]. The left input is called *control input*, the upper inputs are called *data inputs* (left data input = *0-input*, right data input = *1-input*).

The *BDD circuit* of a BDD is obtained by the following construction: Traverse the BDD in topological order and replace each non-terminal node v in the BDD by a *MUX* cell, connect the control input with the primary input x_i, corresponding to the label of the BDD node. Then, connect the 0-input to *low(v)* and the 1-input to *high(v)*. Finally, connect the output of the multiplexor which substituted the root node with a primary output.

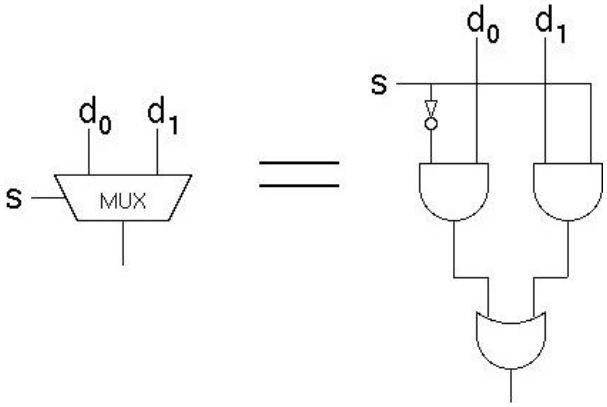

Figure 3 Multiplexor cell MUX

Remark 1 *The handling of nodes that have at least one pointer to a constant has direct implications on the testability of the resulting circuit. For example consider the BDD given in Figure 4 (a) (printed upside-down to see the correspondence to the circuits):*

1. *As has been suggested in [4,3], the MUX cells connected to constant values can be simplified. This can result in redundancies, but the final circuits are smaller counted in the number of literals (see Figure 4 (b)).*
2. *In [10] all terminals are connected to a new test input. By this 100% testability can be ensured by construction (see Figure 4(c)).*

[1] All results in the following also transfer to different realizations.

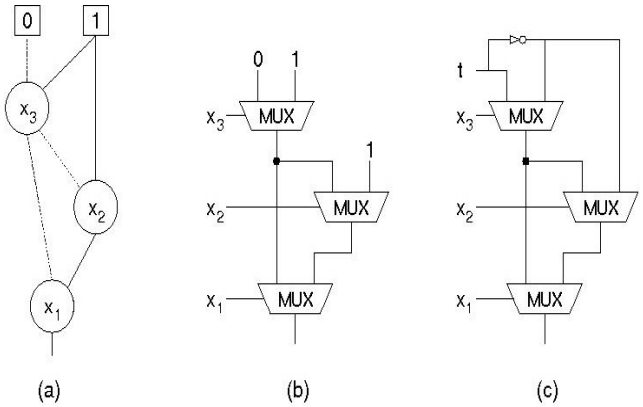

Figure 4. Mapping a BDD to a circuit

In the following only the second method is considered. But the new techniques to increase the RPT can be applied to the first method as well.

2.4. Stuck-At Fault Model

In this paper the *Stuck-At Fault Model* (SAFM) [8] is considered. A fault in the SAFM causes exactly one input or output pin of a node in the circuit to have a fixed constant value (0 or 1) independently of the values applied to the inputs of the circuit.

To measure the random pattern testability of circuits we consider the expected and minimal detection probability for stuck-at faults. We use the definition of the fault detection probability of a single fault as given in [13]. Having all fault detection probabilities the expected probability and minimal probability follow.

3. BDD-Optimization for Random Pattern Testability

In this section two approaches are presented to improve the RPT of BDD circuits. First the RPT is considered during variable reordering. In a second step input probabilities for the variables are determined. We present an iterative - but time consuming - approach and also a fast heuristic that is linear in the size of the original BDD.

3.1 Sifting with Different Objective Function

The variable order of a BDD has a major influence on the number of its nodes. But to decide, if for a given function a BDD with at most some given number of nodes exists, is NP-complete [7]. Therefore heuristics are used to minimize BDDs.

One well-known heuristic is Sifting [17]: A variable is chosen and moved to any position of the variable order based on exchanging adjacent variables. Then it is fixed at the best position (i.e. where the smallest BDD results), afterwards another variable is chosen. No variable is chosen twice during this process. This heuristic can be used to minimize a BDD with respect to other goals than the number of nodes as well. This is done by changing the objective function that measures the size.

A procedure called PLATO (Probabilistic Logic Analyzing Tool) [12,13] calculates the exact detection probability of all faults. After every exchange of variables, this procedure is started and as a result returns the number of redundant faults, the minimum and the expected values of the fault detection probability. These values are used in the BDD minimization process, i.e. included in the objective function of sifting.

Let F_{old} denote the expected fault detection probability before sifting and let F_{act} denote the minimal fault detection probability. The size during sifting is measured by

$$s = -\frac{F_{act}}{F_{old}}$$

Instead of the expected fault detection probability, we also use the number of literals to minimize area in our experiments in an analogous way.

3.2. Computation of Input Probabilities

3.2.1. Iterative Approach

Input probabilities for inputs are determined by a greedy algorithm to improve the RPT. Because the technique does not take advantage of the BDD-like structure of the circuit, it is applicable to any other circuit as well.

One input variable x_i is chosen. For this variable x_i the probability $P(x_i = 1)$ is iteratively set to any value from 0.1 to 0.9 with steps of width 0.1. For each value of $P(x_i = 1)$ the fault detection probability is determined. Then, $P(x_i = 1)$ is fixed to the value resulting in the best testability. Afterwards the next variable is chosen and the same process is repeated. By choosing each variable at most once the procedure terminates after assigning a probability to each input variable. Several iterations of the whole process further increase the testability of the circuit. In Section 4 results are given for maximizing the minimal and the expected fault detection probability with this approach.

3.2.2 Linear Time Approach

The previous approach does not exploit the BDD structure and is rather slow. In this section a fast heuristic to calculate input probabilities is introduced. The calculation is done in time linear in the size of the BDD and is based on the efficient calculation of output probabilities (see Equation (1)).

To detect a stuck-at-0 fault on a circuit line, the value 1 has to be assigned to that line by applying an input pattern. Then, the (erroneous) value of the line has to be propagated to an output. The case for stuck-at-1 is equivalent. Therefore both assignments to lines (0 and 1) are equally important for good RPT. A circuit line corresponds to an edge from a node to its parents in the BDD (neglecting the internal lines of multiplexors). Thus, this assignment becomes easy if the output probability of a node is close to 0.5. In the following, this observation is exploited by using the fast calculation of output probabilities as explained in Section 2.2.

For a level i in the BDD, the sum of the square distances to 0.5 of the output probabilities of all nodes in level i is given by:

$$D_i = \sum_{level\ i} (P(v) - 0.5)^2$$

$P(v)$ is given by Equation (1). Provided that the output probabilities of the children are given, $P(x(v) = 1)$ is the only variable in D_i. Let P_i be the solution to the resulting quadratic equation $D_i'(P_i) = 0$, where $P(x(v) = 1)$ is substituted by P_i. Then, the value of $P(x(v) = 1)$ is set as follows:

$$P(x(v) = 1) = \begin{cases} 0.1 & if \quad P_i \le 0 \\ P_i & if \quad 0 < P_i < 1 \\ 0.9 & if \quad P_j \ge 1 \end{cases}$$

This assures, that each primary input has an input probability different from 0 and 1, i.e. all values can occur.

Now, starting from the bottom level going to the top level in the BDD, D_i is minimized for each level.

4. Experimental results

All techniques have been implemented using CUDD as the underlying BDD package [18]. Experiments have been carried out on a SUN Fire 280R at 900MHz with 3 Gbyte

of main memory. Combinational benchmarks from LGSynth91 [19] were investigated.

The technique introduced in [10] leads to BDD circuits without redundancies at the cost of one additional input. Experimental studies in [10] have shown that these circuits are comparable in size to the circuits that result from directly replacing BDD nodes by multiplexors. Also the testability of the circuits without redundancies is much better than that of circuits resulting from other synthesis tools as e.g. SIS. Results for the circuits without redundancies are given, but these directly transfer to other BDD circuits.

In a first series of experiments we compare the minimization of area to the optimization for RPT during synthesis as explained in Section 3.1. Table 1 shows the results. For each benchmark the number of inputs and outputs is given. Also the number of literals as obtained by SIS is reported in column '#lits' for the different optimization goals. Minimal and expected fault detection probability are given in columns 'min.' and 'exp.', respectively. The RPT was optimized with respect to the expected fault detection probability. The optimization for RPT can even reduce the area due to the heuristic approaches (see '5xp1' or 'alu2'). The expected fault detection probability was increased for most benchmarks, while only little or no overhead in area was necessary.

In the second series of experiments the approaches to adjust the input probabilities as explained in Section 3.2 are compared. The circuits previously optimized for RPT were considered. Results for expected and minimal fault detection probability are reported in Figures 5 and 6, respectively. Bars for '0.5' show the RPT with input probabilities $P(x_i = 1) = 0.5$ as a reference.

The fast algorithm only visits each node of the BDD once and then determines the input probabilities. Although the fault detection probabilities are only considered indirectly an improvement was often observed - by 5% in case of 'cm150a'.

The iterative approach always improved the expected fault detection probability when this was used as the objective (bars for 'iterative (exp)'). The improvement was always significant: more than 10% in most cases and up to a factor of two in one case ('cmb').

But the minimal fault detection probability often decreased because it was not considered during optimization. This can be avoided by using the minimal fault detection probability as the objective (bars for 'iterative (min)'). For all but one benchmark the minimal fault detection probability increased - up to a factor of 1000 ('count'). At the same time the expected fault detection probability never decreased but often also increased.

In summary the joined application of both approaches - during synthesis and with optimized input probabilities - can largely improve the testability of circuits. Fault detection probabilities can increase by several orders of magnitude while the overhead in area is very low.

7. Conclusions

Two new approaches to improve the RPT of BDD circuits during and after synthesis were presented. The synthesis approach using Sifting and the fast algorithm to optimize input probabilities exploit the structure of BDD circuits to increase the testability. The iterative approach can be applied to arbitrary circuits. The minimal or expected fault detection probability can be used as objectives.

Both techniques – BDD optimizations and choosing input probabilities – are orthogonal. Thus, the combination of both further increases the testability of circuits. On average the expected fault detection probability was improved by more than 10%. Even an

improvement by several orders of magnitude of the minimal fault detection probability can be achieved. Since BDDs are used as the underlying data structure the results can be applied in logic synthesis, but also at higher levels of abstraction, like register transfer or system level.

References

1. P. Ashar, S. Devadas, and K. Keutzer. Gate-delay-fault testability properties of multiplexor-based networks. In Int'l Test Conf., pages 887 - 896, 1991.
2. P. Ashar, S. Devadas, and K. Keutzer. Testability properties of multilevel logic networks derived from binary decision diagrams. Advanced Research in VLSI: UC Santa Cruz, pages 33 - 54, 1991.
3. P. Ashar, S. Devadas, and K. Keutzer. Path-delay-fault testability properties of multiplexor-based networks. INTEGRATION, the VLSI Jour., 15(1):1 - 23, 1993.
4. B. Becker. Synthesis for testability: Binary decision diagrams. In Symp. on Theoretical Aspects of Comp. Science, volume 577 of LNCS, pages 501 - 512. Springer Verlag, 1992.
5. B. Becker. Testing with decision diagrams. INTEGRATION, the VLSI Jour., 26:5 - 20, 1998.
6. M. Blum, A. Chandra, and M. Wegman. Equivalence of free Boolean graphs can be decided probabilistically in polynomial time. Information Processing Letters, 10:80 - 82, 1980.
7. B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. IEEE Trans. on Comp., 45(9):993 - 1002, 1996.
8. M. Breuer and A. Friedman. Diagnosis & reliable design of digital systems. Computer Science Press, 1976.
9. R. Bryant. Graph-based algorithms for Boolean function manipulation. IEEE Trans. on Comp., 35(8):677 - 691, 1986.
10. R. Drechsler, J. Shi, and G. Fey. MuTaTe: An effcient design for testability technique for multiplexor based circuits. IEEE Trans. on Computer-Aided Design, 2004.
11. W. Günther and R. Drechsler. ACTion: Combining logic synthesis and technology mapping for MUX based FPGAs. Journal of Systems Architecture, 46(14):1321 - 1334, 2000.
12. R. Krieger. PLATO: A tool for computation of exact signal probabilities. In VLSI Design Conf., pages 65 - 68, 1993.
13. R. Krieger, B. Becker, and R. Sinkovic. A BDD-based algorithm for computation of exact fault detection probabilities. In Int'l Symp. on Fault-Tolerant Comp., pages 186 - 195, 1993.
14. L. Macchiarulo, L. Benini, and E. Macii. On-the-fly layout generation for PTL macrocells. In Design, Automation and Test in Europe, pages 546 - 551, 2001.
15. A. Mukherjee, R. Sudhakar, M. Marek-Sadowska, and S. Long. Wave steering in YADDs: A novel non-iterative synthesis and layout technique. In Design Automation Conf., pages 466 - 471, 1999.
16. I. Parulkar, S. Gupta, and M. Breuer. Estimation of BIST resources during high-level synthesis. Jour. of Electronic Testing: Theory and Applications, 13(3):221 - 237, 1998.
17. R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In Int'l Conf. on CAD, pages 42 - 47, 1993.
18. F. Somenzi. CUDD: CU Decision Diagram Package Release 2.3.1. University of Colorado at Boulder, 2001.
19. S. Yang. Logic synthesis and optimization benchmarks user guide. Technical Report 1/95, Microelectronic Center of North Carolina, 1991.
20. F. Balarin, M. Chiodo, P.Giusto, H.Hsieh, A,Jurecska, L.Lavagno, C.Passerone, A. sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara. Hardware-Software Co-Design of Embedded Systems: The Polis Approach. Kluwer Academic Publishs, 1997.

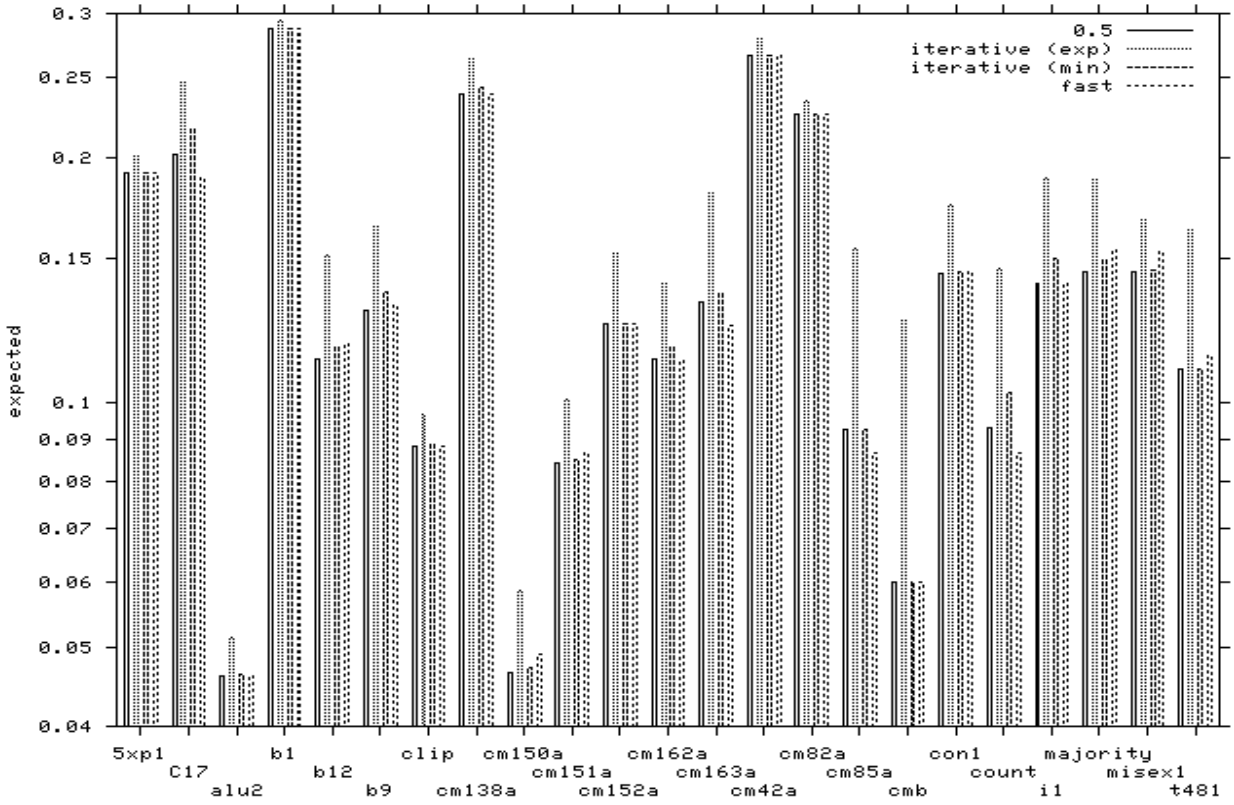

Figure 5. Expected fault detection probability

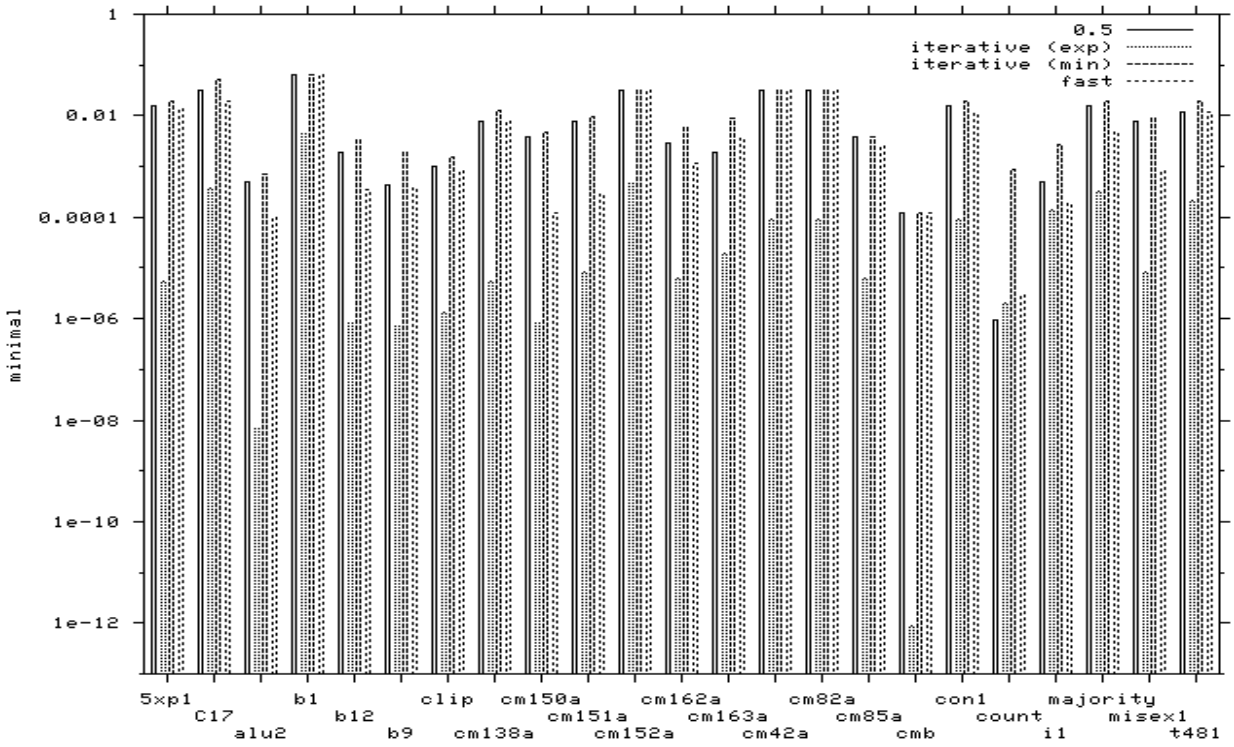

Figure 6. Minimal fault detection probability

Table 1. Circuits Optimized by sifting

			area			test		
Circ.	#in	#out	#lits	min.	exp.	#lits	min.	exp.
5xp1	8	10	192	3.9063e-3	0.165594	160	0.015625	0.191153
C17	6	2	25	0.03125	0.200231	25	0.03125	0.201322
alu2	11	6	670	4.8828e-4	0.040207	626	4.8828e-4	0.46124
b1	4	4	19	0.0625	0.2875	19	0.0625	0.2875
b12	16	9	246	1.9531e-3	0.111737	250	1.9531e-3	0.113064
b9	42	21	440	4.2725e-4	0.119417	492	4.2725e-4	0.129393
clip	10	5	383	9.7656e-4	0.082742	359	9.7656e-4	0.088124
cm138a	7	8	64	7.8125e-3	0.23913	64	7.8125e-3	0.23913
cm150a	22	1	124	7.8125e-3	0.039871	152	3.9063e-3	0.04655
cm151a	13	2	61	0.015625	0.080943	77	7.8125e-3	0.08401
cm152a	11	1	28	0.03125	0.125	28	0.03125	0.125
cm162a	15	5	148	2.9297e-3	0.98868	164	2.9297e-3	0.1128
cm163a	17	5	149	3.90625e-3	0.127631	153	1.9531e-3	0.1328
cm42a	5	10	72	0.03125	0.266447	72	0.03125	0.266447
cm82a	6	3	52	0.03125	0.225694	52	0.03125	0.225694
cm85a	12	3	144	1.95312	0.079843	144	3.90625	0.092544
cmb	17	4	106	1.2207e-4	0.060078	106	1.2207e-4	0.060078
con1	8	2	53	0.015625	0.130899	45	0.015625	0.143973
count	36	16	864	9.5367e-7	0.092149	864	9.5367e-7	0.09289
i1	26	13	170	4.8828e-4	0.134292	178	4.8828e-4	0.140068
majority	6	1	24	0.015625	0.144231	24	0.015625	0.144231
misex1	9	7	147	7.8125e-3	0.136418	139	7.8125e-3	0.144468
t481	17	1	96	0.011719	0.109853	96	0.011719	0.109853