RALF: Reliability Analysis for Logic Faults — An Exact Algorithm and Its Applications

Samuel Luckenbill[1], Ju-Yueh Lee[2], Yu Hu[3], Rupak Majumdar[1], and Lei He[2]
1. Computer Science Department, University of California, Los Angeles
2. Electrical Engineering Department, University of California, Los Angeles
3. Electrical Engineering Department, University of Alberta, Edmonton Canada

Abstract—**Reliability analysis for a logic circuit is one of the primary tasks in fault-tolerant logic synthesis. Given a fault model, it quantifies the impact of faults on the full-chip fault rate. We present RALF, an exact algorithm for calculating the reliability of a logic circuit. RALF is based on the compilation of a circuit to deterministic decomposable negation normal form (d-DNNF), a representation for Boolean formulas that can be more succinct than BDDs. Our algorithm can solve a large set of MCNC benchmark circuits within 5 minutes, enabling an optimality study of Monte Carlo simulation, a popular estimation method for reliability analysis, on real benchmark circuits. Our study shows that Monte Carlo simulation with a small set of random vectors generally has a high fidelity for the computation of full-chip fault rates and the criticality of single gates. While we focus on reliability analysis, RALF can also be used to efficiently locate random pattern resistant faults. This can be used to identify where methods other than random simulation should be used for accurate criticality calculations and where to enhance the testability of a circuit.**

I. Introduction

Fault-tolerant techniques are increasingly important for designing robust systems in modern circuit technologies. Conventional semiconductor devices using scaled CMOS are vulnerable to soft errors (e.g., due to single-event upsets), resulting in a reduced mean time to failure (MTTF), and nano devices (e.g., carbon nano-tubes) suffer from excessive manufacturing defects, reducing yield. Thus, a lot of recent research has focused on system optimizations that target fault tolerance. In this paper, we focus on technology-independent logic optimizations for fault tolerance at the circuit level.

Reliability analysis of logic circuits is one of the fundamental algorithmic tasks for many optimization algorithms for fault tolerance. It computes the signal probability of logic gates under an input distribution, and the probability of the propagation of errors among logic gates under given fault models. It also identifies useful metrics for individual circuit components which can be used to guide logic synthesis to optimize a circuit for reliability. For instance, reliability analysis can compute *criticality* of a gate, the number of input vectors for which a fault at this gate is observable at the primary outputs divided by the total number of input patterns. In addition, reliability analysis serves in the verification phase to assess the robustness of the overall circuit after a logic optimization.

Existing reliability analysis approaches can be divided into the following two categories: simulation-based [1] and symbolic [2]–[4]. Simulation-based analyses propagate a set of test vectors through the circuit and observe the output behavior. Due to the huge number of input combinations, one usually cannot afford to obtain an exact solution using a simulation-based analysis. Nevertheless, simulation-based approaches using random input vectors and Monte Carlo sampling are popular because of their flexibility, runtime controllability, and ease of implementation. For example, they have been used in a few recently proposed fault-tolerance techniques [1], [5].

Symbolic analyses use formal methods to propagate probabilities through a symbolic representation of a circuit, e.g., a DAG-based netlist [4], a representation for Boolean functions [6], or a representation for probabilistic networks [3], [7], [8]. However, exact symbolic algorithms are difficult to scale in the presence of path re-convergence [2], [3],[1] and approximations introduced in practical implementations often produce large estimation errors [4] and have poor estimation of reliability compared to simulation-based approaches.

In this paper, we propose an exact symbolic algorithm, *RALF* (Reliability Analysis for Logic Faults), for technology-independent circuit reliability analysis. The key technical innovation in RALF is the compilation of circuits into a representation for Boolean functions called *deterministic decomposable negation normal form* (d-DNNF) [9]. The fault rate at the primary output can be computed in time and space linear in the size of the d-DNNF representation, even in the presence of path re-convergence in the circuit. The d-DNNF representation is generally more succinct than a BDD representation, enabling RALF to scale to practical circuits. Unlike existing symbolic analysis approaches using exhaustive enumeration of satisfying assignments, BDDs, or probabilistic transfer matrices [3], RALF scales to many real-world circuit benchmarks and finds both the exact fault rate for the full circuit and the exact criticality for each individual gate in the circuit. For most of the MCNC big 20 circuits [10], RALF returns the accurate criticality for one gate in under one minute. It can computes the full-chip fault rate for a circuit with 200 inputs and 238 gates in under 5 minutes. In contrast,

[1]For example, the algorithm from [3] took over 10 minutes to compute the fault rate for a MCNC benchmark (9symml) with nine inputs.

an equivalent BDD-based technique does not scale for these circuits.

Using RALF, we present an optimality study of Monte Carlo simulation for reliability analysis. Although the accuracy of Monte Carlo simulation has been studied based on between-simulation variability, it is not known how far the criticality estimations obtained by Monte Carlo simulation are from exact values. The scalability of RALF makes such a study feasible for real benchmark circuits. Tested on 93 MCNC benchmark circuits, our study shows the following indications.

- For uniformly distributed faults, 1K randomly generated test vectors are sufficient to obtain a relatively accurate estimation of the full-chip fault rate for most of the MCNC circuits;
- For non-uniformly distributed faults, a large set of random vectors or a set of dedicated test vectors are required to obtain an accurate estimation of the full-chip fault rate;
- 64 random vectors that can be implemented easily in a 64-bit machine are sufficient to obtain a sufficiently good assessment of the relative criticality of individual gates for fault-tolerant optimization.

Furthermore, for random pattern-based circuit testing, RALF can be used to identify random pattern resistant (RPR) faults [11]. RPR faults are faults with very low detection probabilities; specifically those that are hard to detect with random patterns. In fact, the criticality of a logic gate returned by RALF is equal to the fault detection probability. In general, fault-tolerant logic synthesis tends to increase logic masking to prevent the propagation of faults [1], [5], which inevitably lowers the testability of a circuit [12]. By using RALF for post-synthesis RPR fault detection, BIST or other testability enhancement techniques can be specifically applied to gates with RPR faults.

II. Preliminaries

Circuits and Faults. We restrict attention to combinational circuits, which are modeled in the usual way as directed acyclic graphs with a set of primary inputs, a set of primary outputs, and internal gates. For a circuit C with k primary inputs, and $x \in \{0,1\}^k$, we denote by $C(x)$ the value of the primary outputs when the input vector x is applied to the primary inputs. For a circuit C and a node n of C, we denote by $C(\bar{n})$ the circuit identical to C except that the output of node n is negated. We denote by $|C|$ the number of nodes in C. For circuit C and node n of C, we write $C_n(x)$ for the output of node n when the input vector x is applied to the primary inputs.

We use von Neumann faults as our fault model, in which each node of a circuit can be flipped independently with a certain probability. In the following, we assume the probability that a node is faulty is known. In the literature, the stuck-at fault model, where each node of a circuit can be stuck at a constant 0 or 1 with a certain probability, is often studied. Our fault simulation algorithm can be easily modified to handle this fault model.

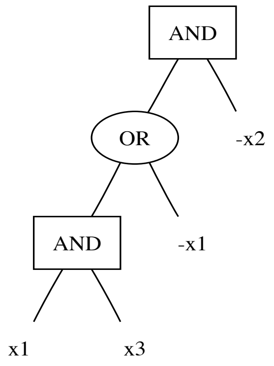

Fig. 1. d-DNNF Representation for Δ

Let C be a circuit with k primary inputs. We define the *criticality* $crit_n$ of a node n of C as

$$crit_n = \frac{1}{2^k}|\{x \in \{0,1\}^k \mid C(x) \neq C(\bar{n})(x)\}|, \qquad (1)$$

Intuitively, the criticality of a node is a measure of the sensitivity of a node with respect to faults. An input x for which $C(x) \neq C(\bar{n})(x)$ is said to cause an observable erroneous output.

The *full-chip fault rate* of a circuit C is the percentage of the primary input vectors which cause observable erroneous outputs due to faults. If the single-fault model is assumed, the full-chip fault rate of a circuit C is calculated by the average criticality of all nodes in C, and it is defined by the following equation:

$$\text{fault\_rate}(C) = \frac{\sum_{n \in C} crit_n \cdot P_F(n)}{|C|}$$

where $P_F(n)$ is the probability that a fault occurs at node n.

The signal probability of a node n of circuit C is defined by the probability that the logic value of a node's output signal is 1, that is, $\frac{1}{2^k}|\{x \in \{0,1\}^k \mid C_n(x) = 1\}|$.

d-DNNF Representation. Our algorithm is based on deterministic decomposable negation normal form (d-DNNF) [9], a representation for propositional formulas. A *negation normal form (NNF)* is a rooted DAG whose leaves are either the constants 0 or 1 or a literal (a variable or its negation), and whose internal nodes are either OR nodes or AND nodes. A *deterministic decomposable negation normal form (d-DNNF)* is a NNF which satisfies the following additional conditions: (1) *Determinism*: the children of any OR node are pairwise logically inconsistent and (2) *Decomposability*: the descendants of any AND node do not share any variables.

Consider the circuit:

$$\Delta = (\neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \qquad (2)$$

Figure 1 shows one possible d-DNNF representation of Δ, as produced by the c2d compiler [13]. It is in d-DNNF because it uses only conjunctions, disjunctions and negations, the left and right branches of the disjunctions are logically inconsistent (one contains $\neg x_1$ and the other contains $x_1 \wedge x_3$), and the children of the conjunctions share no common variables.

A *reduced-ordered binary decision diagram* (ROBDD, or BDD in short) [14] can be considered in d-DNNF form, since every internal node of a BDD is of the form $(x \wedge \alpha) \vee (\neg x \wedge \beta)$, i.e., the representation maintains determinism, and the variable x does not appear in α or β, i.e., the representation maintains decomposability. However, it is known that in general, the BDD representation of a formula is more restrictive than the d-DNNF representation [9], and hence a d-DNNF representation of a formula can be more succinct than its BDD representation.

III. The RALF Algorithm

A. Signal Probability Computation

Let C be a circuit with one primary output. For each primary input x, let $p(x)$ be the probability that the input is 1. By compiling a circuit into a d-DNNF representation, we compute the probability, under the input distribution given by p, that the output of the circuit is 1. The algorithm recursively computes a function $val(\alpha)$ that assigns each node of the d-DNNF the probability that the output of the node is 1:

$$val(\alpha) = \begin{cases} p(x) & \text{If } \alpha \text{ is a variable } x \\ 1 - p(x) & \text{If } \alpha \text{ is the literal } \neg x \\ \prod_i val(\alpha_i) & \text{If } \alpha = \bigwedge_i \alpha_i \\ \sum_i val(\alpha_i) & \text{If } \alpha = \bigvee_i \alpha_i \end{cases} \tag{3}$$

The computation of val exploits the properties of determinism and decomposability in the d-DNNF. The root of the d-DNNF represents the primary output of the original circuit, so the value computed for the root node is the probability that the primary output of the original circuit is 1.

Leaf nodes in the d-DNNF represent literals, thus, $val(n) = p(x)$ for a leaf node n corresponding to a variable x and $val(n) = 1 - p(x)$ for a leaf node n corresponding to $\neg x$.

At disjunction nodes in the d-DNNF, we want to compute the probability that one of the children evaluates to 1. If the sub-circuit represented at the disjunction node α has two children β and γ. We want to compute $Pr(\alpha) = Pr(\beta) + Pr(\gamma) - Pr(\beta \wedge \gamma)$. Determinism guarantees that the children of any disjunction are pairwise inconsistent. This means $Pr(\beta \wedge \gamma) = 0$ and $Pr(\alpha) = Pr(\beta) + Pr(\gamma)$. The recursive algorithm gives us $Pr(\beta)$ and $Pr(\gamma)$, and the algorithm sums the values of the children at disjunction nodes.

At conjunction nodes, we want to compute the probability that every child evaluates to 1. The decomposability property guarantees that no children of any conjunction share variables, and hence the probabilities are independent. Thus, the algorithm computes the product of the probabilities that each child evaluates to 1.

The algorithm runs in time and space linear in the d-DNNF representation.

B. Single-Node Criticality Analysis

Let C be a circuit and n a node of C. To measure $crit_n$, we create a miter from the original circuit. The miter is constructed from two copies of the original circuit that differ only by an inverter at the output of n. The primary outputs of the two copies are pairwise EXOR'd. For a given input, if any EXOR evaluates to 1, we know the the inverted node affects that output. By taking the OR of all of the EXORs, we get a single output that evaluates to 1 when any primary output is affected (Figure 2). The probability that the output of the miter is 1 under a uniform distribution of inputs (i.e., for each variable x, $p(x) = 0.5$) is exactly the criticality of the inverted gate on the primary outputs of the original circuit.

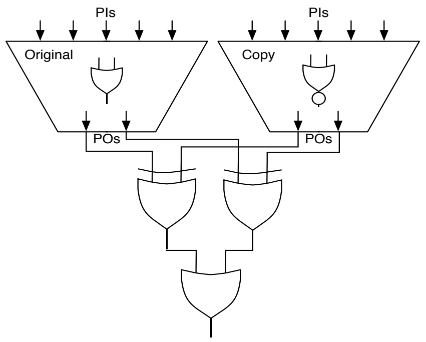

Fig. 2. Miter

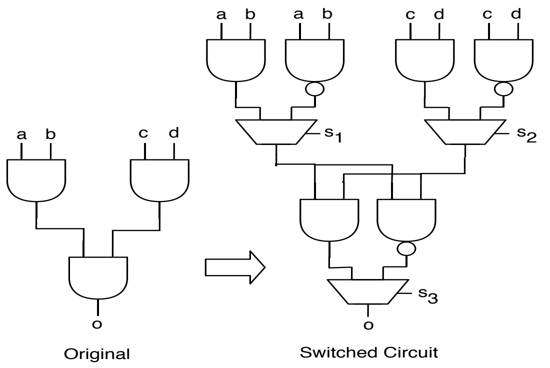

Fig. 3. Programmable-Miter

In our implementation, we first construct the miter and convert it to conjunctive normal form (CNF) in time linear in the size of the circuit. We then use the c2d compiler [13] to compile the CNF to d-DNNF. Compilation can be expensive: compiling to d-DNNF is as hard as counting the number of satisfying assignments, but c2d uses extensive caching to reduce compilation time. Specifically, it divides the problem into subproblems by conditioning the CNF on subsets of variables. For each subproblem, it enumerates satisfying solutions to build d-DNNF, then caches the d-DNNF. If the compiler revisits the same subproblem (a frequent occurrence), it uses the cached d-DNNF version rather than re-building it.

If the miter can be compiled in a reasonable amount of time, the exact analytic solution for criticality is obtained by computing val using Equation (3) on the final d-DNNF. Since the runtime of the algorithm is linear in the size of the d-DNNF, the runtime is dominated by compilation.

C. Multi-Node Criticality Analysis

One important application for criticality analysis is to identify nodes with high criticality, e.g., to harden or re-synthesize

Circuit Characteristics				Miter size				Compilation time (s)	
Name	Gate#	Input#	Output#	CNF		d-DNNF	BDD	RALF	BDD
				Var#	Clause#	Node#	Node#		
i7	581	199	67	4168	9901	79637	-	138.24	-
mult32a	535	34	1	3243	8035	166976	-	118.85	-
i6	455	138	67	3363	8038	67295	895599	58.31	35639.97
i5	402	133	66	2662	6391	507965	-	59.99	-
b9	296	41	21	1560	3667	725729		67.97	
i4	292	192	6	1877	4177	20231	-	3.78	-
my_adder	259	33	17	1618	3976	29865	116621	10.03	75.78
cht	244	47	36	1782	4471	175055	2084948	8.55	5859.04
i2	238	201	1	1622	3547	103434	398017	4.84	91.88
lal	234	26	19	1348	3262	523283	-	67.77	-

TABLE I
10 LARGEST SOLVABLE CIRCUITS USING RALF WITH A 5-MINUTE TIMEOUT

them. Finding the highest criticality nodes requires calculating the criticality of every node in the circuit. Using the single-node criticality analysis described in Section III-B, we have to compile a different miter for every node. In this section, we introduce a more complex circuit called a *programmable miter*. Combined with a slight modification to Equation (3) and the introduction of some additional CNF constraints, the programmable miter allows us to compile a single circuit from which we can compute the criticality of all nodes.

To construct a programmable miter, we create two copies of each gate in the original circuit: one exact and one whose output is inverted. The outputs of the two copies are fed to a MUX allowing us to switch between the exact and inverted gates (Figure 3). This switched circuit allows us to choose which node will be faulty at runtime instead of hard-coding the faulty gate into the miter. Using this switched circuit as the second copy, we construct a miter as described in Section III-B, where the resulting programmable-miter has one additional primary inputs for each MUX, to determine if the "correct" or "inverted" copy of each gate is chosen.

Were we to compile the programmable-miter directly, the result would be capable of calculating the criticality of any number of simultaneous nodes. While this is desirable, compilation becomes significantly more expensive because of the number of possible states that must be considered. If there are n nodes in the circuit, we now have n additional unconstrained variables in the CNF. Without any additional constraints, the compiler must consider all 2^n possible assignments to these variables and build a d-DNNF that captures the behavior of programmable MUX in all possible configurations. To make compilation more tractable, we introduce at-most-one constraints into the CNF for these variables, which models the condition that there is at most one fault at any time.

In order to use Equation (3) for criticality analysis on a programmable miter, we need to constrain its MUX inputs. For the faulty node's MUX, we set the probability that its control input is 1 to 1. For all other MUX inputs, we set the probability that their control inputs are 1 to 0.

For a circuit with n nodes and a corresponding d-DNNF with m nodes, we can compute the criticality of all n nodes in $O(nm)$ time, assuming at most one fault.

IV. Applications and Results

A. Tractability of RALF

In this section, we evaluate the performance of our proposed RALF using the MCNC benchmark set. We chose 183 benchmarks from the third set in the MCNC suite. These benchmarks, when expressed as and-inverter graphs, range from 1 to 12126 AND gates with 820 gates on average. The number of inputs to each circuit ranges from 2 to 257, with an average of 29. The majority of circuits have less than 2000 gates and less than 50 inputs. Assuming the von Neumann fault model with a single-fault constraint (only one fault occurs during each cycle), we ran RALF on 183 benchmarks. 93 (51%) could be solved in under 5 minutes and 89 (49%) could be solved under one minute. Compilation to d-DNNF is performed by c2d [13]. All experiments were run on a 2.83GHz Intel Xeon with 6MB cache and 11GB of RAM.

With a 5-minute timeout, the 10 largest solvable benchmarks and their runtime are summarized in Table I. Note that Table I only shows the compilation time (the time required to convert a circuit from CNF to a functionally equivalent circuit in d-DNNF) since compilation time dominates the total runtime.

As mentioned, BDD is a restricted form of d-DNNF, and therefore we can convert a circuit to its BDD-based representation and then propagate the probability along the BDD using the similar method as RALF. In Table I, we compare RALF with the BDD-based exact solution, where BDD is constructed using Berkeley ABC tool [15] with a 10-hour timeout. The comparison result is also summarized in Table I in terms of number of gate and runtime, where '-' indicates BDD compilation cannot finish within 10 hours. The result shows that the d-DNNF representation is significantly smaller than the BDD representation, approximately 14% of the size on average. Furthermore, only 4 of the 10 circuits could be compiled to BDDs, and RALF overwhelmingly outperforms the BDD-based method in terms of runtime.

For circuits on which RALF did not finish within 5 minutes, we can often calculate the criticality of each gate individually. The full chip fault rate can be obtained after solving criticality gate by gate, and the total runtime is the sum of the runtime of the single-gate calculations. Table II summarizes the runtime for solving the exact criticality of one gate in each of the 10 biggest MCNC combinational circuits. Most of the computations for a single gate completed within 2 seconds while "apex2" and "des" did not finish within a 1-hour timeout.

Circuit	Gate#	Input#	Output#	Runtime(s)
alu4	2403	14	8	43.81
apex2	3191	39	3	-
apex4	1422	9	19	1.17
des	2508	256	245	-
ex1010	5366	10	10	184.54
ex5p	1470	8	63	0.12
misex3	3652	14	14	0.99
pdc	7553	16	40	2.14
seq	2951	41	35	0.27
spla	7472	16	46	0.88

TABLE II
RUNTIME FOR EXACT COMPUTATION OF ONE GATE IN 10 BIGGEST MCNC COMBINATIONAL CIRCUITS WITH A ONE-HOUR TIMEOUT.

B. Application 1: Fidelity Study of Monte Carlo Simulation

The exact solutions provided by RALF allow us to evaluate the accuracy of Monte Carlo simulation. We use both RALF and Monte Carlo simulation to compute the criticality of each gate in a circuit. We assume a uniform distribution of the primary input vectors and that each input has a signal probability of 0.1, i.e., the probability that an input is 1 is 10%[2].

Under the set of solvable MCNC circuits solved by RALF, we first study the accuracy of Monte Carlo simulation for full-chip fault rate computations by comparing the Monte Carlo Error (MCE) from simulations with 1K and with 128K random vectors. MCE is the absolute value of the difference between the value produced by the Monte Carlo simulation and RALF, our exact algorithm. The relative MCE is the MCE divided by the exact solution. As shown in Figure 4, for most circuits, Monte Carlo simulation using 1K vectors gives a close estimation to the exact solution (with a mean error of 0.5% and a mean relative error of 1%). Note that the improvement reported by most of the existing fault-tolerant synthesis algorithms [1], [5] is above 10% (relative improvement). Therefore, an estimation error under 1% given by the Monte Carlo simulation with 1K vectors is sufficient to assess the quality of these algorithms. On the other hand, we note that increasing the length of random vectors to 128K can reduce the relative MCE from 1% to 0.3%, but that such a small precision improvement is not justified by over 100x runtime increase.

Figure 5 shows the maximal MCE and relative MCE of the criticality values for all gates in a circuit. Monte Carlo simulation using 1K vectors has a large maximal relative MCE, i.e., for certain gates, the estimation is far from the exact value. The relative MCE can be significantly reduced using 128K random vectors. This observation indicates that for non-uniformly distributed faults, a large set of random vectors are needed to estimate the full-chip fault rate. In this case, it is beneficial to design a set of dedicated test vectors to capture corner cases that purely random vectors would not cover.

To evaluate the criticality of single gates in a circuit, Figure 6 compares the accuracy of Monte Carlo simulation with

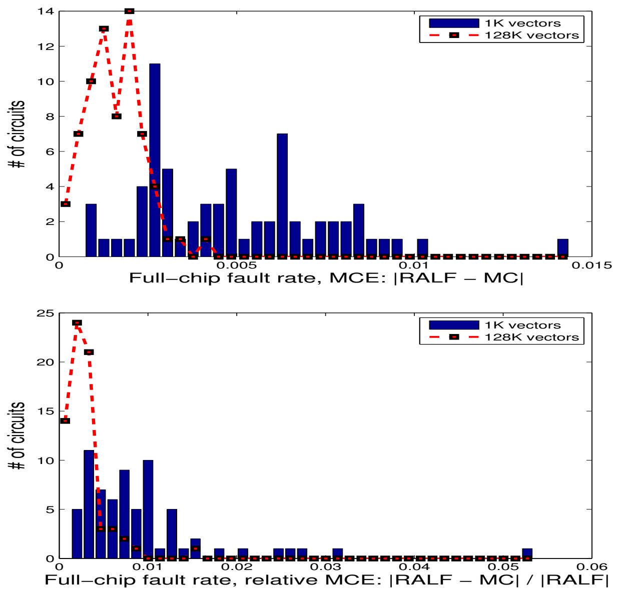

Fig. 4. **Monte Carlo error (MCE) for full-chip fault rate over 93 MCNC circuits with signal probability 0.1 for primary inputs**

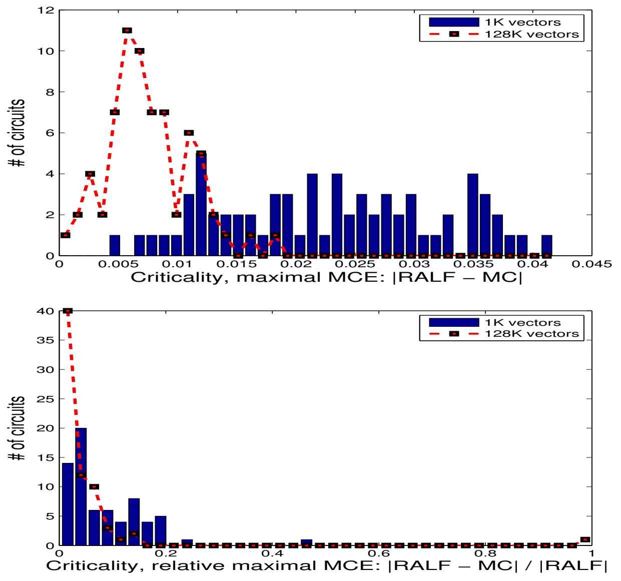

Fig. 5. **Monte Carlo error (MCE) for criticality computation of single gates in over 93 MCNC circuits with signal probability 0.1 for primary inputs**

64 random vectors and 128K random vectors for the "rd73" MCNC circuit. To guide resource-constrained optimizations, we are primarily interested in the ordering of nodes by their criticality values. As shown in Figure 6, assuming uniformly distributed faults[3], the overall ranking of criticality produced by 64 random vectors has a high fidelity to the exact solution. Using fast simulation with 64 vectors, we can accurately cluster gates into a few categories based on their criticality values, thereby excluding the "noise" due to MCE. Note that a logic simulation using 64 random vectors can be implemented

[2]Similar observations are obtained under other signal probability settings, including 0.5 and 0.9.

[3]Similar observation is seen for non-uniformly distributed faults.

very efficiently in bit-parallel fashion on a 64-bit CPU core.

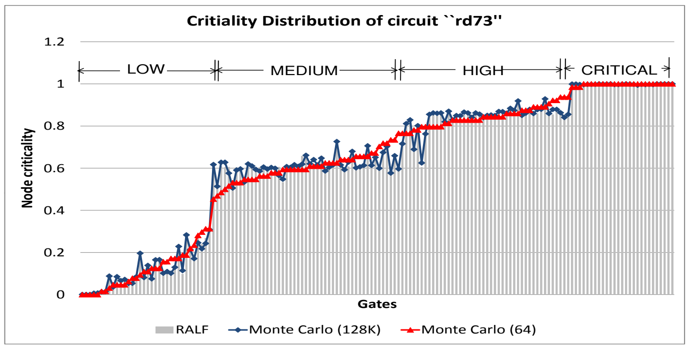

Fig. 6. Criticality distribution of MCNC circuit rd73

C. Application 2: Detection of Random Pattern Resistant Faults

As the criticality of a logic gate returned by RALF is equal to the fault detection probability of the gate, RALF can identify random pattern resistant (RPR) faults that are useful in re-synthesis for testability after a fault-tolerant synthesis. Figure 7 shows the logic schematic of MCNC circuit "o64", which consists of `AND` gates at the primary inputs followed by a network of `OR` gates. Due to logic masking, to propagate a fault through an `OR` gate requires that all the non-faulty inputs to be logic 0. Therefore, in "o64", to propagate a fault at an `OR` gate to the primary output, i.e., to detect a fault, requires that all the other `OR` gates that are not in its transitive fanin network have their inputs to be logic 0. However, under random input vectors, it is rarely satisfied and results in extremely low fault detection probabilities, i.e., criticality values. As a result, it is hard to use random input vectors to cover the RPR faults that occur in these gates. Using RALF, we can plot a distribution curve of the fault detection probability for each gate in "o64" (see Figure 8), and from the plot one can easily identify RPR faults and use BIST or other testability enhancement techniques for those gates with RPR faults.

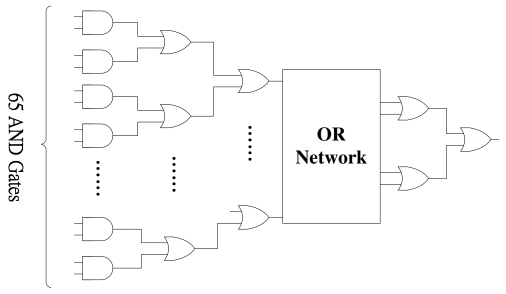

Fig. 7. Logic schematic of "o64"

V. Conclusion

We have proposed RALF, an exact algorithm for reliability analysis for fault-tolerant logic synthesis. We have applied RALF to evaluate the fidelity of Monte Carlo simulation and to identify RPR faults. While the d-DNNF data structure scaled better than related representations such as BDDs, in the worst case, the representation can be exponentially larger than the

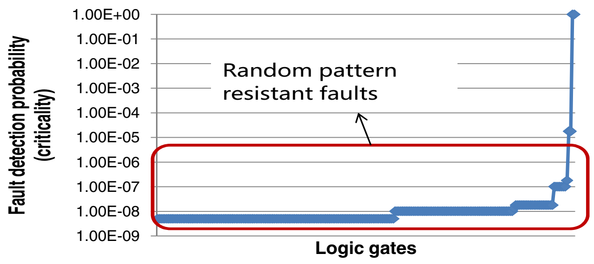

Fig. 8. Fault detection probability of "o64"

circuit. This is unavoidable for any exact representation, since the reliability analysis problem is computationally hard. It will be interesting to explore algorithms which combine exact computations locally with approximations at the circuit level which can scale without sacrificing precision.

RALF currently assumes that the inputs have independent signal probabilities and that there is at most one error per cycle. It will also be interesting to see if d-DNNF-based computations can be used without these assumptions.

Acknowledgements We thank Adnan Darwiche and Knot Pipatsrisawat for many helpful discussions and the anonymous referees for their suggestions. This work was patially supported by State-Key-Lab of ASIC & Systems of Fudan University and Actel.

References

[1] S. Krishnaswamy, S. M. Plaza, I. L. Markov, and J. P. Hayes, "Enhancing design robustness with reliability-aware resynthesis and logic simulation," in *Proc. Int. Conf. on Computer Aided Design*, 2007.
[2] K. Parker and E. McCluskey, "Probabilistic treatment of general combinational networks," *IEEE Transactions on Computers*, vol. C-24, pp. 668–670, 1975.
[3] S. Krishnaswamy, G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Probabilistic transfer matrices in symbolic reliability analysis of logic circuits," *ACM Trans. on Design Automation of Electronic Systems*, vol. 13, 2008.
[4] P. K. Samudrala, J. Ramos, and S. Katkoori, "Selective triple modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs," *IEEE Transactions on Nuclear Science*, vol. 51, pp. 2957–2969, 2004.
[5] Y. Hu, Z. Feng, R. Majumdar, and L. He, "Robust FPGA resynthesis based on fault tolerant boolean matching," in *Proc. Int. Conf. on Computer Aided Design*, 2008.
[6] B. Zhang, W.-S. Wang, and M. Orshansky, "FASER: Fast analysis of soft error susceptibility for cell-based designs," in *ISQED*, pp. 755–760, 2006.
[7] T. Rejimon and S. Bhanja, "An accurate probalistic model for error detection," in *VLSI Design*, pp. 717–722, 2005.
[8] A. Abdollahi, "Probabilistic decision diagrams for exact probabilistic analysis," in *Proc. Int. Conf. on Computer Aided Design*, pp. 266–272, 2007.
[9] A. Darwiche, "Decomposable negation normal form," *Journal of the ACM*, vol. 48, p. 2001, 2001.
[10] S. Yang, "Logic synthesis and optimization benchmarks, version 3.0," tech. rep., Microelectronics Center of North Carolina (MCNC), 1991.
[11] N. A. Touba and E. J. McCluskey, "Automated logic synthesis of random pattern testable circuits," in *International Testing Conference*, 1994.
[12] S. Krishnaswamy, I. Markov, and J. Hayes, "Improving testability and soft-error resilience through retiming," in *Proc. Design Automation Conf*, 2009.
[13] A. Darwiche, "A compiler for deterministic, decomposable negation normal form," in *In Proc. AAAI '02*, 2002.
[14] R. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. on Computers*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.
[15] "ABC: A system for sequential synthesis and verification," in *http://www.eecs.berkeley.edu/ alanmi/abc/.*