AnSER: A Lightweight Reliability Evaluator for use in Logic Synthesis

Smita Krishnaswamy, Stephen M. Plaza, Igor L. Markov, and John P. Hayes
{smita, splaza, imarkov, jhayes}@eecs.umich.edu
Advanced Computer Architecture Lab, University Of Michigan
2260 Hayward St., Ann Arbor 48109-2121

ABSTRACT

As device features continue to scale, traditional circuit optimizations targeting area and delay have to be closely monitored for their effect on reliability. Yet, existing reliability analysis algorithms suffer from poor scalability and technology-dependent approaches which are unable to efficiently guide synthesis optimizations. Therefore, we present a new reliability analysis framework known as *Analysis of Soft Error Rate (AnSER)* that runs in linear time and outperforms state-of-the-art by two to three orders of magnitude. Moreover, our algorithms are amenable to uses in logic synthesis due to their technology independence and scalability.

1. INTRODUCTION

Reliability is becoming an increasingly important concern in logic synthesis due to unstable behavior resulting from soft errors and trends in device technology. Therefore it is important for designers to assess the impact of their decisions on reliability. Since logic masking is a key mechanism by which errors can be mitigated, the synthesis phase is especially important in maintaining reliability. However, the integration of reliability into design flows has been difficult due to the computational complexity of reliability evaluators and their reliance on detailed technology-dependent information. Past reliability evaluators rely on laborious calculations using several types of decision diagrams and electrical simulation [8, 7].

To address these problems, we develop a technology-independent reliability evaluator in OAGear known as *AnSER* that scales linearly in the size of the circuit. AnSER can be used before or after technology mapping. Additionally, we embed AnSER into OAGear — an emerging software platform for circuit analysis and logic synthesis tools. The key contributions of this work include:

- A fast *soft-error rate (SER)* evaluation algorithm that uses the *SimEqi* package in OAGear for logic simulations.
- An implemenation of the reliability analysis framework AnSER native to OAGear, to facilitate the integration of reliability concerns into logic synthesis and technology mapping.
- Extensible software design to facilitate the integration of new error models and future algorithms for reliability analysis.

The remainder of this paper is organized as follows: Section 2 presents our SER evaluation algorithm, Section 3 describes our software design, Section 4 gives validation and runtime results. Section 5 concludes the paper. Finally, the Appendix gives algorithms for SER under a multiple-error assumption and a node impact measure that identifies critical areas of the circuit. We added these algorithms to extend our AnSER framework.

2. SER EVALUATION

To evaluate the soft-error rate (SER), we utilize signal signatures computed by bit-parallel logic simulation [2]. A given node f in a logic circuit can be characterized by its signature, S_f, i.e., logic values observed in response to K input vectors $X_1 \cdots X_K$, $S_f = \{f(X_1), \dots, f(X_K)\}$ where $f(X_i) = \{0,1\}$ indicates the output of f for a given input vector. In addition to signatures, we also compute *don't-care masks* (denoted S_f*) using simulation. *Observability don't-cares (ODCs)* occur when the value of an internal node does not affect primary outputs for a certain input pattern. We use methods from [6] compute the ODC mask for nodes in the circuit. A similar algorithm was given in [9]. In an ODC mask S_f*, a 1 indicates a care-bit and 0 indicates a don't-care.

Our SER evaluation algorithm uses a logic-level fault model that extends the standard stuck-at (SA) fault model. For every clock cycle, we assume that each node g in the circuit has a *temporary* single SA-1 (TSA-1) fault, with probability $Perr1(g)$ if g is controlled to 0 and a temporary SA-0 (TSA-0) with $Perr0(g)$ otherwise. We derive our *single-error SER* using the TSA fault model. Due to the similarity to SA faults, TSA faults inherit the wide applicability of SA faults. For instance, in a cycle where a node n is controlled to 1, $Perr0(g)$ can be the probability of a bit flip due to a particle strike.

In testing, an estimation of the number of input vectors that result in a node f being 0 is f's 0-*controllability*. The *observability* of a node measures the likelihood that the node itself controls the output. Together observability and controllability form testability. In other words, the number of test vectors for an error is the number of vectors that simultaneously sensitize and propagate the error. We analyze the testability *probabilistically* by using signatures. For a node f in circuit C, and input distribution i, we denote its 1-controllability as $con_{i1}(f)$ and compute it by counting the number of zeros in the signature:

$$con_1(f) = \text{numOnes}(S_f)/K \tag{1}$$

The *0-controllability* is denoted $con_0(f)$ and can be computed as $1 - con_1(f)$. The observability of a node is the probability that the value at the node is propagated to the primary output. It is computed as follows:

$$obs(f) = \text{numOnes}(S^*_f)/K \tag{2}$$

Together, the observability and controllability information can tell us about the testability of a node. The 1-testability is computed by counting the number of bits where both the ODC mask and signature have a 1, i.e., where the node is both controllable to 1 and observable. The 0-testability is the number of bits where the ODC mask has a 1 and the signature have a 0.

$$test_1(f) = \text{numOnes}(S^*_f \& S^f)/K \tag{3}$$

If we assume that each node g has TSA-0 and TSA-1 probabilities

$Perr0(g)$ and $Perr1(g)$, we can write the SER as a sum of the error contributions from each node g in the circuit C.

$$Perr(C) = \sum_{g \in C} \left(test_0(g) Perr0(g) + test_1(g) Perr1(g) \right) \qquad (4)$$

Since $test_0$ and $test_1$ include error sensitization and propagation conditions, Equation 4 takes into account the possibility of the error being logically masked before reaching the output. We summarize our SER algorithm for TSA faults is given in Figure 1.

After the gates are topologically sorted, bit-parallel simulation is performed. Next the ODCs are computed in reverse topological order. Then testabilities are computed in the *compute_test* function using Equation 3. Finally, the testabilities are weighted by the probability of error and summed to obtain the SER.

3. SOFTWARE DESIGN

We implement AnSER as an extensible framework for the reliability analysis of large gate-level netlists in the OpenAccess database [2]. Figure 2 shows a flow diagram of our implementation. The major features of this framework include the user interface, algorithms manager, and reliability algorithm objects. Algorithm objects are created by the algorithm manager, and computation is done on-demand. The manager can be interfaced in two different ways, as a stand-alone program or as an API. The user interface is minimally coupled with algorithm internals using a *compiler firewall* design pattern [3] to reduce compile-time. The algorithm manager is implemented as a *singleton [3]* data structure to facilitate user access with restricted privileges. The algorithm, in turn, accesses the OpenAccess database and SimEqui.

```
double compute_SER(Circuit C){
  double SER = 0;
  topological_sort(C);
  compute_simulation_sigs(C);
  reverse_topological_sort(C);
  compute_ODCs(C);
  for(gate g ∈ C){
    test1(g) = compute_test1(g);
    test0(g) = compute_test0(g);
    SER+ = C.perr0 * test1(g);
    SER+ = C.perr1 * test0(g);
   }
  }
  return SER;
}
```

Figure 1: The AnSER Single Error algorithm.

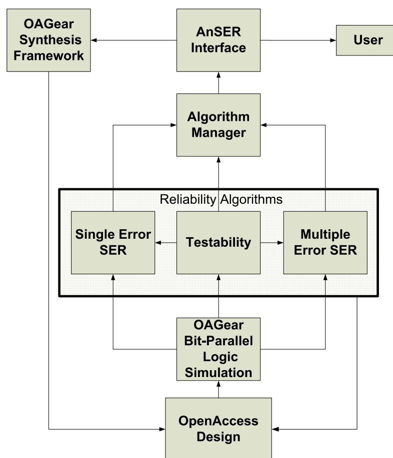

Figure 2: Reliability evaluation framework within OAGear.

In order to add new algorithms to AnSER, the manager is given a reference to a base *algorithm object* and the interface is automatically updated. With this arrangement in mind, we implemented algorithms that evaluate the following important metrics:

- Single-Error SER (see Figure 1)
- Multiple-Error SER (see Appendix)
- Node Testability (see Section 2)
- Node Impact (see Appendix)

When computing SER it is common to assume either a single error per cycle or allow multiple simultaneous errors. Testability and node impact evaluators can both be used to drive synthesis changes that improve the circuit. In addition, our framework is flexible in that precision of computation can be controlled by the user when algorithm objects are instantiated. For instance, the number of simulation vectors used for reliability can be specified by the user.

Synthesis tools can use AnSER to check the impact of design decisions on reliability. This is tracked through notification by the *oaObserver* object. However, recomputation of invalid data is done on the subsequent query. We also extend the OpenAccess database by storing node testability and impact information in *oaAppDef* objects for consumption by other optimization algorithms. In particular, this information can be used to target sensitive circuit areas for re-synthesis and can further the integration of reliability into synthesis algorithms.

In the more detailed companion paper [4] we demonstrate the ability of our reliability evaluation methods to guide synthesis algorithms and selectively duplicate nodes. For instance, we guide the ABC synthesis tool [1] to simultaneously optimize area and reliability. Our results show that we achieve a 13% reliability improvement with a 1% area *decrease*. In OAGear, we provide a demonstration and regression tests of the native SAT-sweeping tool *Ssw*. In this application we analyze changes in the netlist triggered by *oaObserver* during SAT-sweeping in order to reject mergers that decrease circuit reliability.

4. EMPIRICAL RESULTS

We validate our approximate SER algorithm against an exact computation using the ATPG software ATALANTA [5]. We provide ATALANTA with a list of all possible stuck-at faults in the circuit to generate tests in "diagnostic mode" where *all* of the test vectors for each fault are derived. Since TSA faults are SA faults that last only for one cycle, the probability of a TSA fault causing an output error is equal to the number of test vectors for the corresponding SA fault weighted by their frequency of appearance on the input. Assuming uniform input distribution, the fraction of vectors that test a fault is an exact measure of its testability. Then, we compute the exact SER using Equation 4. Table 1 compares AnSER from ATALANTA on small benchmark circuits (since the ATPG-based method does not scale to large benchmarks). The average error is only 3% for 2048 simulation vectors.

Table 2 shows runtime comparisons with other evaluators in the literature. Results show that our evaluator runs orders of magnitude faster than FASER [8] and SERD [7]. These benchmarks were run on 2GHz Pentium Centrino Duo. Table 3 shows SER and runtime results on IWLS benchmarks. The largest circuit only takes 11.7 seconds to complete. SER numbers are given in units of FIT ($failures/10^9 s$) assuming that each gate has a failure rate of $8.0E-5$, a value we obtained for a 100nm technology from the gate characterizations of [7]. Note that AnSER scales linearly in the size of the circuit through these benchmarks.

Circuit	ATALANTA	AnSER	% Error
b1	1.28E-05	1.31E-05	2.81
c17	6.96E-07	6.96E-07	0.01
x2	3.78E-05	3.87E-05	2.20
decod	2.60E-05	2.62E-05	0.83
tcon	5.30E-05	5.39E-05	1.67
z4ml	5.29E-05	5.37E-05	1.50
parity	7.60E-05	7.69E-05	1.24
majority	6.25E-06	6.63E-06	6.05
mux	1.58E-05	1.38E-05	12.54
pm1	2.86E-05	3.00E-05	4.70
pcler8	7.06E-05	7.24E-05	2.52
pcle	5.38E-05	5.34E-05	0.75
average			3.06

Table 1: Comparison of AnSER with exact reliability evaluation using the ATALANTA ATPG software.

Circuit	Time(s)		
	AnSER	SERD	FASER
i1	0.020	20	—
i2	0.000	20	—
i3	0.057	20	—
i4	0.072	20	—
i5	0.046	20	—
i6	0.059	20	—
i7	0.067	20	—
i8	0.20	20	—
i9	0.013	40	—
i10	0.000	60	—
c432	0.006	10	22
c880	0.015	10	—
c1355	0.021	20	40
c1908	0.015	20	66
c3540	0.000	60	149
c6280	1.000	120	278

Table 2: Runtime comparisons with several reliability evaluators from the literature. The dashed lines indicate that there is no published data from the tool in question.

Circuit	No. Gates	SER(FIT)	Runtime(s)
pci_conf_cyc_addr_dec	97	4.89E-3	0.23
steppermotordrive	226	8.006E-3	0.27
ss_pcm	470	1.68E-2	0.3
usb_phy	546	1.53E-2	0.28
sasc	549	2.10E-2	0.26
simple_spi	821	2.50E-2	0.3
i2c	1142	2.7E-2	0.34
pci_spoci_ctrl	1267	0.029	0.342
des_area	3132	0.019	0.782
spi	3227	0.118	0.68
systemcdes	3322	0.127	0.55
tv80	7161	0.104	0.91
systemcaes	7959	0.267	0.97
mem_ctrl	11440	0.494	1.36
ac97_ctrl	11855	0.409	1.38
usb_funct	12808	0.390	1.42
pci_bridge32	16816	0.656	1.78
aes_core	20795	0.550	2.1
wb_conmax	29034	1.030	4.18
ethernet	46771	1.480	5.77
des_perf	98341	3.620	9.34
vga_lcd	124031	4.800	11.7

Table 3: Runtime of AnSER on IWLS 2005 benchmarks, and SER values computed by AnSER.

5. CONCLUSIONS

We presented a lightweight reliability evaluation framework for use in logic synthesis that utilizes the logic simulator within OAGear. Results show that our methods are significantly faster than existing SER evaluators while maintaining high accuracy as compared to exact evaluation. Furthermore, AnSER can be used in early stages of design due to its technology independence. We also demonstrate the extensibility of our method by incorporating an alternative error model and random pattern testability computations.

6. REFERENCES

[1] Berkeley Logic Synthesis and Verification Group, "ABC: a system for sequential synthesis and verification", http://www.eecs.berkeley.edu/~alanmi/abc/.

[2] K-H. Chang, et al., "Fast Simulation and Equivalence Checking using OAGear," *IWLS 2006*.

[3] E. Gamma, R. Helm, R. Johnson, J. Vlissides *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[4] S. Krishnaswamy, S. M. Plaza, I. L. Markov, and J. P. Hayes, "Enhancing Design Robustness using Reliability-aware Resynthesis and Logic Simulation," *IWLS 2007*.,

[5] H. K. Lee and D. S. Ha, "On the Generation of Test Patterns for Combinational Circuits," TR No. 12-93, Dept. of Electrical Eng., Virginia Polytechnic Inst.

[6] S. Plaza, K-H. Chang, I. Markov, and V. Bertacco, "Node Mergers in the Presence of Don't Cares" *ASP-DAC 2007*.

[7] R. Rao, et al., "An Efficient Static Algorithm for Computing the Soft Error Rates of Combinational Circuits," *DATE 2006*, pp. 164-169.

[8] B. Zhang, W. S. Wang, and M. Orshansky, "FASER: Fast Analysis of Soft Error Susceptibility for Cell-Based Designs," *ISQED 2006*, pp. 755-760.

[9] Q. Zhu, N. Kitchen, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "SAT sweeping with Local Observability Don't-cares", *DAC 2006*, pp. 229-234.

Appendix: Multiple Errors and Impact

To further extend the AnSER framework, we implement a reliability calculation based on multiple temporary faults (TMSA). Here each gate has *independent* probabilities $Perr0(g)$ and $Perr1(g)$.

In topological order, for each gate g with inputs x, y, output z, and gate error probabilities $Perr1(g), Perr0(g)$, we compute the cumulative output error probabilities ($Perr1_{in}(z)$, $Perr0_{in}(z)$) considering cumulative error probabilities at the inputs $Perr1_{in}(x)$, $Perr0_{in}(x)$, $Perr1_{in}(y)$, $Perr0_{in}(y)$ and the logical functionality of the gate. As we propagate error probabilities to the POs, we compute the SER of the circuit under the multiple error assumption in linear time. The SER for the output of a circuit C, $o(C)$, is given by

$$Perr(C) = Perr0_{in}(o(C))con_1(o(C)) + Perr1_{in}(o(C))con_0(o(C))$$

We also define the *impact* metric for each gate which estimates the gate's influence on circuit reliability. This metric accounts for the probabilities $Perr0_{in}(n)$,$Perr1_{in}(n)$ that errors propagate to the node and for the probability $obs(n)$ that these errors propagate to the output:

$$impact(n) = Perr0_{in}(n)obs(n) + Perr1_{in}(n)obs(n)$$

Another natural extension of our framework is to directly give testability and testing information for synthesized circuits. The testability of circuits may decrease as synthesis tools optimize for reliability, thereby making circuits resistant to random-pattern testing. However, since our tool maintains testability information, we can output vectors that test selected components of the design.