

MEMESTAR: A Simulation Framework for Reliability Evaluation over Multiple Environments

Christian J. Hescott, Drew C. Ness and David J. Lilja
ECE Department, University of Minnesota, Minneapolis, MN 55455
research@hescott.com, dneess@ece.umn.edu, lilja@umn.edu

Abstract

We present a methodology for the simulation of soft errors targeting future nano-technological devices. This approach efficiently scales the failure rate of individual devices according to cell area and considers the effect of multiple faults within a circuit. Furthermore this methodology measures circuit operation over a range of environments and consequently provides a means of targeting designs to the expected operating environment rather than worst case. We demonstrate the effect area has on circuit reliability and fault tolerance.

1. Introduction

The projected decrease in circuit reliability of future technologies is driving a need for better simulation tools that can help predict a circuit's reliability early in the design cycle. With the increasing complexity in computer architectures, verification has become a very large problem. With predicted increase in environmental impacts leading to single event effects (SEE), ensuring reliable behavior becomes next to impossible.

To date, SEE models mostly use fault injection to characterize circuits and find what portions are most susceptible to faults. Examples include calculating error probabilities for each circuit node [7] and injecting a single fault (usually in the form of a bit flip) to observe its effects [4],[11]. These techniques are costly in terms of simulation time and consequently limit the size of circuits and input sets that can be used.

As fault tolerance becomes more integrated into circuit and architectural design, there will be an increasing need to quickly evaluate alternative design choices over a range of operating environments. In such circumstances, it will be more beneficial for designers to observe a circuit's overall reaction to faults rather than classifying each node's susceptibility.

Single fault models are good for identifying circuit behavior at a fixed operating environment. They provide a method to observe fault propagations leading to classification of fault types and estimates of latching probabilities of current pulses injected into combinational logic. They do however have their shortcomings.

The single fault model assumes that no more than one fault can enter the system at any given evaluation window. Fault rates in current technology validate this assumption. Furthermore, limiting analysis to combinational logic circuits either by only choosing such or by stopping the analysis once a fault has been latched eliminates the possibility of multiple latched faults interacting.

If two independent single event upsets (SEU) occur close enough in time or a fault from a single event remains latent long enough, the possibility of interaction increases and cannot be ignored. This is possible as soft error rate (SER) increases in future

technologies or the device operates over a long enough time frame that a second SEU occurs and interacts with a latent fault.

Additionally multiple faults can occur concurrently even in today's technology when an SEU particle strike occurs at an angle of incidence large enough to cross more than a single node. Such "double-bit-flips" can cause multiple concurrent faults in neighboring nodes leading to interactions not covered in a single fault injection model [12].

SEUs in combinational logic can propagate to more than one latch or flip-flop due to the fan-out of the logic cone. Fault injection frameworks that ignore combinational logic (for simplification and efficiency) must consider this possibility. However, determining which memory elements should receive concurrent faults without including the combinational logic cone remains an important problem. There is obvious correlation that should be considered.

The last two examples of multiple faults can occur in current technologies and operating environments and has been addressed in previous work. However, temporally close faults has yet to be considered and without the proper fault injection framework becomes computationally expensive. It is this reason that we develop the Multiple Environment and Multiple Error Simulation Tool for Analysis of Reliability (MEMESTAR). The goals of which are:

1. Develop a framework that naturally scales today's fault models to future technologies.
2. Utilize a common metric that allows cross-circuit and cross-operating environment comparisons.
3. Provide a simple yet extensible framework for current and future fault models applied to different technologies.

We assume that future devices will continue to use traditional boolean logic. Alternatives would require an entire paradigm shift in circuit design. Secondly, we assume faults manifest as bit flips in boolean logic. It is not obvious how faults will physically manifest in future technologies. Today's current pulses may not be present in the future. However this assumption is only for base comparisons of unknown fault models. Because of our third goal, the fault injection model can easily be extended to include the behavior of more complex models.

The paper is organized as follows. Section 2 derives a modeling parameter for controlling a circuit's fault susceptibility. Our multiple fault injection model and analysis framework is presented in sections 3 and 4. Experimental details are provided in section 5. Simulator validation is presented in 6. Two case studies are presented in section 7. A brief related work section is included in section 8 and conclusions in section 9.

2. Multiple Operating Environments and Circuit Area

Ideally we'd like to measure a circuit's soft error failure rate over a range of operating environments. Such a model can be used for measuring the dynamic circuit reliability as a device's environment changes (e.g. laptop on a airplane). This analysis can also provide designers with a method for dynamic fault optimizations alleviating the need for worst-case scenario designs. We derive a control parameter used in fault injection analysis that provides such comparisons. The SER of circuit is derived in [8] as

$$SER = R_{PH} \cdot \alpha \cdot P(SE) \quad (1)$$

where α is the fraction of effective particle hits, $P(SE)$ is the probability of soft error given an effective particle hit. The R_{PH} parameter is the particle hit rate given by

$$R_{PH} = \int_{E_{n,min}}^{E_{n,max}} F_n(E_n) dE_n \cdot A_t \quad (2)$$

where $(F_n)(E_n)$ is the location dependent neutron flux between neutron energies $E_{n,min}$ and $E_{n,max}$ and A_t is total silicon area.

Note that particle hit rate can change if either location or total area of the circuit changes. Normalizing R_{PH} by total circuit area gives

$$R_{PHA} = \frac{R_{PH}}{A_t} \quad (3)$$

the average particle hit rate per unit area. The parameter R_{PHA} provides a method of comparison across multiple circuit topographies while taking into account any area differences. Furthermore, varying R_{PHA} for a fixed total circuit area predicts a circuit's reaction to changing environments of particle flux.

We use R_{PHA} interchangeably with Gate Failure Rate throughout the remainder of the paper and assume that all particle hits result in enough charge to cause a fault. Alternatively α could be wrapped into R_{PHA} . The term $P(SE)$ is determined through simulation.

3. Multiple Fault-injection Model

We wish to model the failure rate of each gate over the life of the test bench. Given a gate's reliability $R_n = 1 - R_{PHA} \cdot A_n$ where A_n is the gate's total sensitive area, we need to decide at any given instant whether or not a fault should be injected into gate n . Using the Monte Carlo approach we can evaluate statistically for every gate if a fault should be injected. However for larger circuits this can be prohibitively expensive due to the cost of generating N pseudo-random number samples. Instead we assume that particle strikes are independent and consequently the number of failures in the circuit at any given time can be modeled as a binomial distribution

$$P_p(n|N) = \binom{N}{n} p^n (1-p)^{N-n} \quad (4)$$

for $0 \leq n \leq N$. That is the probability of n gates having faults given N total gates and p (probability of gate failure) is provided by equation 4. This is implemented in a Monte Carlo simulation using the inverse cumulative binomial distribution function (F_p^{-1}) where

$$F_p(n|N) = \sum_{i=0}^n \binom{N}{i} p^i (1-p)^{N-i} \quad (5)$$

We treat R_{PHA} as a discretized control parameter and substitute total sensitive area A_t for N in equation 5 and R_{PHA} for p . We attach a relative weight based on sensitive cell area to each gate instance which determines the likelihood of that gate failing (relative to the other gates in the circuit). We assume faults are distributed uniformly according to this weight. Note we follow the common practice of estimating a cell's sensitive area using its total area.

Our fault injection algorithm consists of four steps.

1. Calculate total number of faults given A_t and R_{PHA} .
2. Determine location of faults using a weight vector.
3. Determine fault type for each selected location.
4. Inject each fault.

These four steps are repeated throughout the simulation of the test bench at each injection point in time. Fault types can include bit flips or pulse injections depending on the desired experiment and gate types. The time factor is explained further in the next section.

4. Checkpoint Algorithm for Fault Injection

Test benches for sequential circuits usually involve setting the inputs, manipulating the clock to propagate the inputs and checking the outputs for correct behavior. For more complex circuits (e.g. CPU architectures), the test benches may include software benchmarks in which checks are not performed until the end of benchmark execution.

Soft error analysis requires some means to check if faults manifest as measurable errors. In sequential circuits, typically a golden run is used in which the test bench is applied to a circuit under no faults and compared to subsequent simulation under fault injection. For long running test benches it can be computationally expensive to run to completion before performing the check against the golden run.

We propose a new algorithm for measuring fault manifestation over a given test bench that applies to both simple and complex sequential circuit test benches. It improves statistical confidence by breaking up the test bench into TS_{cnt} time slices and performing a golden run and check over each time slice. Each time slice becomes a statistical sample of the entire test bench.

Our checkpoint algorithm consists of creating saved checkpoints of the entire simulation environment as a basis to fall back on once the fault injections and testing are complete. Any catastrophic failures or any dormant faults within the system are eliminated when the previous saved state is loaded. With each checkpoint a golden run is performed in which fault injection is disabled. At the end of the golden run, the desired check values are saved. These values are used as the "correct" values to compare against once faults are injected into the circuit and can include architectural registers, simulated memory or simply circuit outputs.

In figure 1 we show the steps of the check pointing algorithm. We have broken down the algorithm into three phases: *Checkpoint and Golden Run Creation*, *Injection and Checking*, and *Move to Next Check Phase*. The first phase is responsible for creating an initial checkpoint and performing the golden run simulation. The second phase injects a fault and allows it to propagate before stopping the simulation and checking the results against the golden run. The third phase is responsible for removing the effects of the fault injection and continuing the simulation until the next fault injection point.

Checkpoint and Golden Run Creation

1. Create a checkpoint saving entire state of the simulation environment
 2. Disable fault injection
 3. Run for $T_{latency}$ time steps
 4. Create a saved state of all testing criteria (this is the golden run)
Injection and Checking
 5. Reload checkpoint created in step 1
 6. Enable fault injection
 7. Run for $T_{latency}$ time steps
 8. Compare architected states with those created in step 4 (record all visible errors)
Move to Next Checkpoint
 9. Reload checkpoint created in step 1
 10. Disable fault injection
 11. Run for $T_{granularity}$ time steps
 12. Go to step 1
-

Figure 1. Steps of the Checkpointing Algorithm

Two parameters, $T_{granularity}$ and $T_{latency}$ control the total number of checkpoints and the length of each checkpoint respectively. They can be used to fine tune the amount of fault injection a circuit receives over the life of its test bench and how long faults can remain within a circuit before being erased and moving to the next checkpoint.

Several fault injection-check pairs can be made for each saved checkpoint by repeating steps 5 through 8 and supplying a different random seed to the fault selector each time. This amortizes the cost of saving checkpoints as the restore operation is minimal compared to the save operation essentially making additional statistical confidence cheap.

5. Experimental Setup

MEMESTAR was implemented using VPI extensions to Verilog. We demonstrate it using the OpenRISC processor [1] and LGSynth93 benchmark suite. OpenRISC is an embedded scalar processor model that has been successfully implemented on FPGA and ASIC technologies.

We synthesized all designs with Synopsys Design Compiler using Virtual Silicon’s UMC 0.18 μm process library available from IMEC. The worst case parameters were used in the compilation.

Weighting was obtained by normalizing the area of each gate provided in the standard cell library to the largest gate in the library.

All designs were simulated using Cadence’s Logic Design and Verification package version 5.1 running on an AMD Opteron 3.06 GHz processor with SCSI local disks, 4GB of memory and Red Hat Enterprise Linux release 3.

The random fault injector was set to inject faults every clock cycle. Unless noted, a granularity and latency of 1000 cycles was used for OR1200 and 1 cycle for LGSynth93. The test bench for OR1200 used the dhrystone benchmark program running for 10 iterations. LGSynth93 runs used the default test benches.

For all results each checkpoint is run a total of 40 times with different random seeds to increase statistical confidence in the results. For OR1200 any measured difference in the 32 architectural registers, program counter or memory state between the injected trial and the golden run is marked as a failure. For LGSynth93

only the outputs are considered. The total probability of failure for a circuit (denoted as *circuit failure rate*) is calculated by taking the total number of failed checkpoints divided by the total number of checkpoints executed. The 95% statistical confidence is calculated and noted where appropriate.

6. Model Validation

This section validates the various components of the MEMESTAR framework.

6.1. Weighted Uniform Distribution Injection Model

Validation of the uniform distribution injection model was achieved using a simple 10-gate circuit with unity weighting. Three different experiments were performed in which the fault location selection algorithm was repeated 100, 1000 and 10000 times recording each fault injection location. The total counts for each gate were statistically analyzed for uniform and normal distributions using a probability plot correlation coefficient (PPCC) and chi-square tests. The PPCC test favored uniform over normal distributions and the chi-square tests indicated no reason to believe the distributions are not uniform. Randomness was tested using an autocorrelation plot over various lags. At both 95% and 99% confidence intervals there were no significant lags (other than the obvious lag 0). Specific results are omitted here due to length. These results validate that our algorithm selects gates uniformly random without introducing any bias.

To determine the contribution of weighting to circuit sensitivity (step two of section 3), a fault injection experiment was conducted on the OR1200 CPU circuit in which the weight of each gate was specified according to its normalized area and compared against a separate run using unity weights. The area was normalized so that the total area was identical to the number of gates in the circuit (i.e. identical to unity’s total weight). This ensures total number of injected faults is identical and only the locations differ. The two weighting types were each applied to two different circuit implementations of the open cores; one using the default circuit and one using a 3MR (N-modular redundancy with $N = 3$) implementation with perfect voter.

The results shown in figure 2 indicate a significant difference in the circuit reliability. The area-based approach yields a decrease in circuit reliability compared to the unity approach for both circuits. This would suggest that this particular circuit has larger gates on sensitive paths. To verify that this change is more than just a shifting effect, the difference in improvement between the base circuit and the 3MR circuit was calculated for both area and unity. The change in reliability is greater for the unity-based weighting scheme compared to area-based one.

The result indicates that in addition to a decrease in circuit reliability, using the area-based weighting results in less reliability improvement when applying 3MR to a circuit (compared to the base circuit). The difference is more apparent with increasing circuit failure rate. This suggests an importance in accurate weighting and consequently fault distribution models when performing simulations as any measured changes in circuit reliability can be affected by the underlying assumptions of fault distribution.

6.2. Binomial Distribution Model

We validate the algorithm that implements the fault injection frequency model (step one in section 3). An experiment was conducted using a simple chain of $N = 10$ buffers with input and output flip flops. This circuit models a simplified pipeline stage

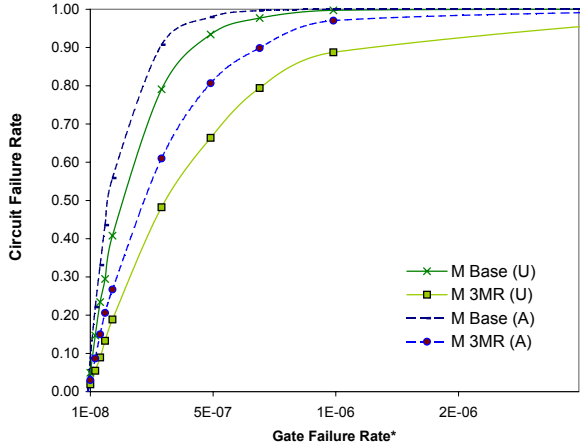


Figure 2. Comparison between area-based weighting (A) and unity based weighting (U) for OR1200 CPU circuit. *Gate failure rate of largest gate for area-based results.

and is easy to compare to analytical models because it contains no fan-in or fan-out. Bit flips were used to ensure that any fault injection into the buffers will result in a latched fault (this simplifies the analysis). Fault injection frequency and time slice width was set to one clock cycle and unity weighting was used.

Figure 3 shows increasing buffer failure rate R_{PHA} plotted against measured circuit failure rate F_c where

$$F_c = 1 - (1 - R_{PHA})^N \quad (6)$$

Equation 6 is derived from classic probability theory stating that the probability of failure is equal to 1 - probability of succeeding. For this circuit, success occurs when all N buffers do not fail $(1 - R_{PHA})^N$. Also the probability that 0 fault injections occur $(1 - F_c)$ is compared to empirical results. Both indicate a good fit of the data with less than 3% difference between the empirical data and the analytical model. The slight increase in error for the final data point (15%) is due to the fault injection model selecting the same gate more than once. Due to verilog's implementation, multiple even number bit flips cancel each other resulting in lower measured failure rate. This however, is easily accounted for in the algorithm by ensuring unique gate selection.

6.3. Time Slice Size Comparison

Adjusting time slice window sizes is necessary when comparing long running test benches to ones that are orders of magnitude shorter. Decreasing the time slice window by a factor equal to the difference in test bench length ensures the same number of samples are collected for each set.

The results in figure 4 show circuit s563 from the LGSynth93 benchmark suite over a range time slice sizes (1, 2, 5 and 10 cycles). Here the evaluation periodicity was fixed and instead the gate failure rate was scaled by a factor of 1, 2, 5 and 10 respectively matching the time slice sizes. The trend is clearly visible indicating that circuit reliability is not strongly correlated to time slice window size but rather to the number of faults injected per time slice window. This demonstrates using different time slice sizes to compare different circuits is valid as long as the evaluation periodicity is correctly scaled.

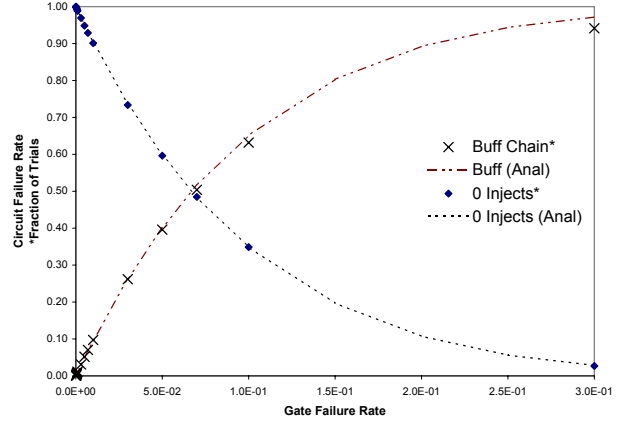


Figure 3. Buffer chain analysis comparing empirical data to an analytical model

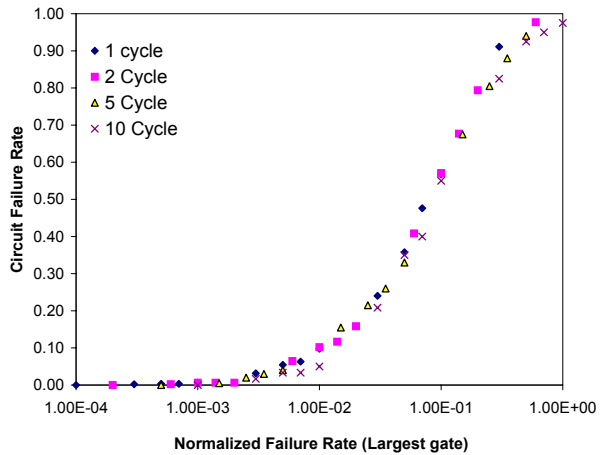


Figure 4. Timeslice width comparison for circuit s563 from LGSynth93

It is important to note that the results of this section do not validate the underlying assumption of multiple faults. Instead it validates the algorithm that implements this assumption. Validating the assumption itself would take detailed 3-dimensional physics-level simulations and is beyond the scope of this paper. A survey of simulation techniques for current technologies and SEU models can be found in [2]. Instead the goal here is to introduce a novel modeling technique and validate its implementation. As detailed physical models of future device failure rates become available this framework can be extended to include such.

7. Case studies

This section demonstrates the use of MEMESTAR in two different example scenarios.

7.1. Voter Sensitivity (in NMR)

Fault tolerance of future devices will become a very important design criteria as fault rates increase. Evaluating performance, re-

liability, power and cost trade offs early in the design cycle will be critical. Classically, N-modular redundancy (NMR) has been a well-known fault-tolerance technique trading off circuit area (overhead) for increase in reliability and decrease in performance. However it has been previously demonstrated that NMR does not scale well to increasing granularities [3]. In other words applying NMR to devices with reliability that are higher than the NMR voter itself results in worse overall reliability. Using our model we can empirically demonstrate this.

Figure 5 shows the OR1200 circuit with 3MR (NMR with $N = 3$) applied to each gate for varying voter sizes. The 3-input voter is implemented using a 4-input MUX from the cell library with a normalized area of 0.317 (normalized to the largest cell in the library). The label indicates the area of the voter (perfect voter = 0.00). All listed areas are not necessarily physically achievable but rather demonstrate what would be possible if the areas could be scaled freely. Also included is the original circuit without 3MR. Three of the six voter sizes yield a decrease in reliability compared to the original. The voter's area must be decreased by a factor of 3 (compared to the default available from the library) before reliability improvement over the original is observed.

These results demonstrate the importance of voter reliability in 3MR implementation. Choosing designs to maximize 3MR placement has been covered in other work [9] however those models do not account for multiple fault environments and target today's technology. Results that are favorable for a circuit's operating environment may not be when the environment changes. We leave it to future work to do a better in-depth analysis. Here this example demonstrates the ability of our model to quickly evaluate a circuit's reliability over a range of environments and design alternatives.

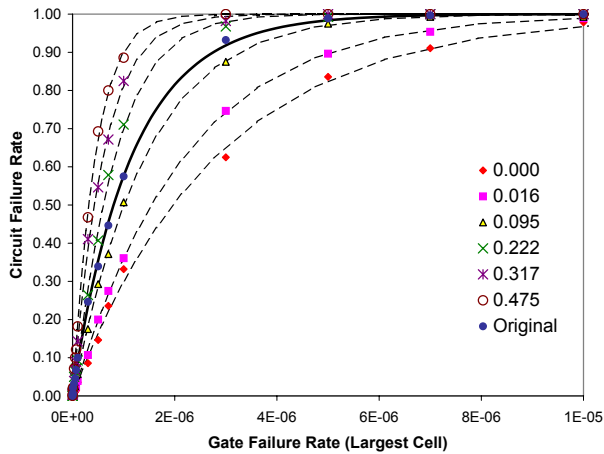


Figure 5. Voter sensitivity in 3MR applied to OR1200 CPU circuit

7.2. LGSynth93 Sequential Benchmarks

We apply our model to a set of sequential circuits from the LGSynth93 benchmark suite. The selected circuits range both in size and test bench length. Figure 6 shows the circuit failure rate for a range of gate failure rates. Some circuits behave as expected such as s5378 which is the largest and also is the least reliable. Two circuits (s641 and s1494) however show poor reliability but are smaller in size compared to the other more-reliable circuits.

This suggests that circuit area is not the only determining factor in reliability.

Figure 7 shows identical data but uses *number of faults injected per timeslice* as the horizontal axis. This essentially normalizes the area of the circuits so that only sensitivity to faults is observed. Here circuit s5378 is better than s641 and s1494 indicating s5378 is less sensitive to total number of faults. In other words s5378 can be expected to fail more if operating in the same environment as s641 and s1494, however the total number of faults tolerated in s5378 is higher.

Using this fault analysis tool makes efficient simulations possible over a wide range of operating parameters. It provides a means to compare different circuit topologies and look for changes in reliability which could lead to discovery of better design techniques.

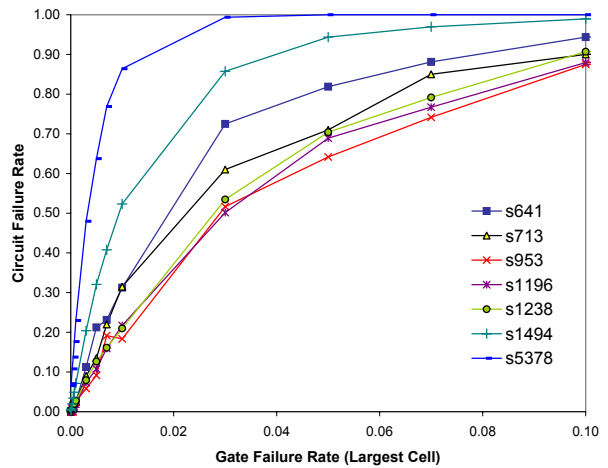


Figure 6. Fault sensitivity for LGSynth93 sequential circuits

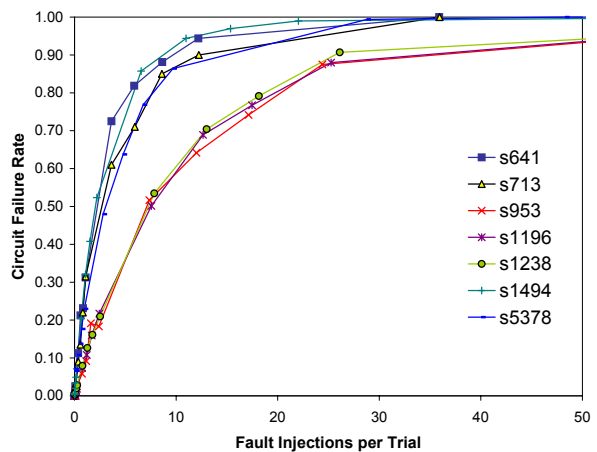


Figure 7. Normalized area fault sensitivity for LGSynth93 sequential circuits

Circuits	Cells	T_{TB} (μ s)	TS_{cnt}	T_{sim}/TS (s)
s641	107	3.5	4	0.22
s713	108	4.2	5	0.22
s953	239	2.7	3	0.16
s1196	280	16.2	16	0.74
s1238	287	17.9	18	0.83
s1494	276	145.3	146	7.14
s5378	667	15.3	16	0.85
OR12K-B	20434	998.0	7	8.88
OR12K-3MR	81736	998.0	7	29.34
s526-1	110	2.4	23	0.85
s526-2	110	2.4	12	0.45
s526-5	110	2.4	5	0.21
s526-10	110	2.4	3	0.14

Table 1. Simulation cost per time slice for each benchmark circuit

7.3. Simulation Performance

To obtain statistical confidence in these results a total of 40 duplicate trials was performed for each data point shown in figures 2 through 7. Table 1 shows the simulation cost (simulation time per time slice) for the LGSynth93 circuits, OR1200 CPU (with and without 3MR) and s526 (with varying time slice sizes). Also shown are the number of cells in the circuit, test bench execution time (T_{TB}) and total time slices per test bench (TS_{cnt}). Interestingly the simulation time is linear with respect to number of cells and total time slice count.

8. Related Work

Probabilistic model checking and its variants utilize binary decision diagrams (BDDs) to encode state space of a particular model. Often used in hardware verification to prove correct functionality, these methods have been adapted to the transient fault regime to provide a substitute to traditional Monte Carlo simulations [6] [10]. The common problem with BDDs is their tendency to require large memory storage as the state space increases. Alternatives to BDDs have been proposed but usually sacrifice some of the original benefit of using such methods. Creating good models that do not abstract away the details of the design under test but still maintain tractability given limited resources remains an important problem.

The model building process is simplified in [5] by avoiding complex property definitions outright. Instead faults are injected into latches of behavioral HDL and a property checker is used to compare the outputs of the DUT to a golden device. This unfortunately does not account for multiple concurrently latched faults due to SEU propagation from combinational logic as discussed earlier and shown to be significant in [4].

Several papers develop fault injection models similar to the one presented here but with only single fault injection [4],[11]. Consequently they do not measure a circuit over a range of operating environments or consider the effects of area on circuit reliability

9. Conclusions

We introduce a multiple fault injection framework for technologies susceptible to increased soft errors. The framework effectively scales soft error rates over a range of gate failure rates according to circuit area and/or operating environment. Each assumption of the algorithm is individually validated.

We use our model to demonstrate NMR sensitivity to voter reliability. For a simple CPU circuit with 3MR applied to every gate, voter reliability must be increased by a factor 3 to obtain an increase in reliability. Additionally we demonstrate our model's ability to measure a group of circuits over a range of operating environments to characterize fault sensitivity with respect to each other.

Future work will apply this framework to do more in-depth evaluation of circuit behaviors and traditional fault-tolerant techniques.

10. Acknowledgements

This work was supported in part by Semiconductor Research Corporation contract no. 2004-HJ-1190, IBM, Intel, the University of Minnesota Digital Technology Center, and the Minnesota Supercomputing Institute.

11. References

- [1] Openrisc 1200 architecture. www.opencores.org.
- [2] P. E. Dodd et al. Basic mechanisms and modeling of single-event upset in digital microelectronics. *Nuclear Science, IEEE Transactions on*, 50(3):583–602, 2003.
- [3] K. J. Gurzi. Estimates for best placement of voters in a triplicated logic network. *Electronic Computers, IEEE Transactions on*, EC-14(5):711–717, 1965.
- [4] Hungse Cha et al. A gate-level simulation environment for alpha-particle-induced transient faults. *Computers, IEEE Transactions on*, 45(11):1248–1256, 1996.
- [5] U. Krautz et al. Evaluating coverage of error detection logic for soft errors using formal methods. In *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, volume 1, pages 1–6, 2006.
- [6] R. Leveugle. A new approach for early dependability evaluation based on formal property checking and controlled mutations. In *On-Line Testing Symposium, 2005. IOLTS 2005. 11th IEEE International*, pages 260–265, 2005.
- [7] L. W. Massengill et al. Analysis of single-event effects in combinational logic-simulation of the am2901 bit-slice processor. *Nuclear Science, IEEE Transactions on*, 47(6):2609–2615, 2000.
- [8] Ming Zhang et al. A soft error rate analysis (SERA) methodology. In *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, pages 111–118, 2004.
- [9] K. Mohanram and N. A. Touba. Cost-effective approach for reducing soft error failure rate in logic circuits. In *Test Conference, 2003. Proceedings. ITC 2003. International*, volume 1, pages 893–901, 2003.
- [10] G. Norman et al. Evaluating the reliability of defect-tolerant architectures for nanotechnology with probabilistic model checking. In *VLSI Design, 2004. Proceedings. 17th International Conference on*, pages 907–912, 2004.
- [11] M. Rimen and J. Ohlsson. A study of the error behavior of a 32-bit risc subjected to simulated transient fault injection. In *Test Conference, 1992. Proceedings., International*, page 696, 1992.
- [12] D. Rossi et al. Multiple transient faults in logic: An issue for next generation ics. In *Defect and Fault Tolerance in VLSI Systems, 2005. DFT 2005. 20th IEEE International Symposium on*, pages 352–360, 2005.