SAND2020-7327C

Multiscale System Modeling of Single Event Induced Faults in Advanced Node Processors

Matthew Cannon*, Arun Rodrigues*, Dolores Black*, Jeff Black*, Luis Bustamante‡, Ben Feinberg*, Lawrence T. Clark†, John Brunhaver†, Hugh Barnaby†, Michael McLain*, Sapan Agarwal‡ and Matthew J. Marinella*
*Sandia National Laboratories, Albuquerque, NM
†Arizona State University, Tempe AZ
‡Sandia National Laboratories, Livermore, CA

Abstract:
We propose a new simulation environment which allows the ability to track and incorporate experimental effects of single event-induced faults from transients on individual transistors to complex systems with multiple processors, memories and other devices.

Index Terms:
Single event effects (SEE), single event transient (SET), single event upset (SEU), fault modeling, structural simulation toolkit (SST)

Presenting Author:
Matthew Cannon, Sandia National Laboratories, 1515 Eubank Blvd SE, MS-0986, Albuquerque, NM 87123, USA, phone: 505-844-5866, email: mcannon@sandia.gov

Presentation Preference: Oral Presentation

Session Preference: Hardness Assurance

Acknowledgements:
This work is supported by Sandia National Laboratories, a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

I. Introduction

Single event effects (SEEs) are a key radiation susceptibility for processors fabricated with modern highly scaled process technologies [1]. This can be exacerbated by systems with multiple processors, each with varying sensitivity to radiation. It has become increasingly important to develop a holistic approach to model the effects of these faults in key pipeline stages of a processor along with the capability to integrate radiation-induced charge generation at the transistor-level.

To address this need, a multi-scale simulation framework to model the effects of single event upsets (SEUs) and single event transients (SETs) has been developed as illustrated in Fig 1. The model utilizes Sandia's Structural Simulation Toolkit (SST), a discrete event simulator (DES) originally developed for high performance computing [2]. The framework considers several levels of abstraction to model the impact of single event effects on a processor, which can be coupled to experimental data at the transistor level:

1) A DES of software running on a faulty processor is used to simulate and track the faults and their impact on benchmark algorithms to probabilistically characterize sensitivity (Section II).
2) A modeling process takes transistor level single event data and translates it to probabilities of SEUs on the output registers of each pipeline stage in the processor, using the following steps (Section III):
   a) The probabilities and physics of a single event on a transistor are modeled to create a probability distribution of charge collected from the radiation event (Section III-A);
   b) The charge collected is used to fit a dual, double exponential current source and circuit simulations are used to abstract radiation events as binary SETs on a gate with varying pulse widths [3] (Section III-B);
   c) The probabilities and durations of SETs on each gate are abstracted to model the probability of a SEU being latched at the register at the end of the pipeline stage (Section III-C).

By using four levels of abstraction, we enable a more comprehensive coverage of possible faults than if Monte Carlo injection were directly used on a full processor model. Using SST also allows us to model large scale heterogeneous systems with multiple discrete components and to track fault propagation through the system. This modeling framework can be used to both characterize how an existing system will behave in a radiation environment and enable the design of both hardware and software level fault mitigation strategies.

II. Pipeline Stage to System Software Fault Simulation

We developed a fault injection tool based on SST and the clspim MIPS model [4]. Starting with an algorithm in C or C++, it is compiled to individual assembly instructions that can be run on a DES model of a faulty processor. SEUs are injected into the registers of a simulated system. The probability of an SEU on each register is calculated in the next sections

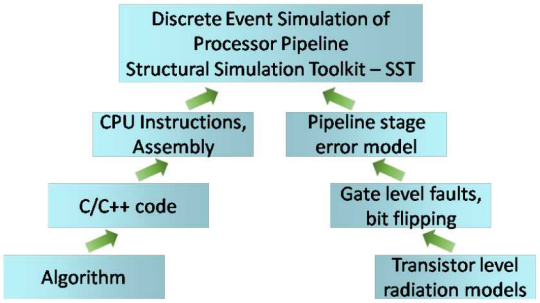


Fig. 1: The impact of transistor level radiation effects on a running algorithm are modeled through a discrete event simulation of the processor pipeline.

and is based on both the register itself and the combinational logic feeding into the register. Faults can be injected either as temporary SEUs or as permanent stuck at faults.

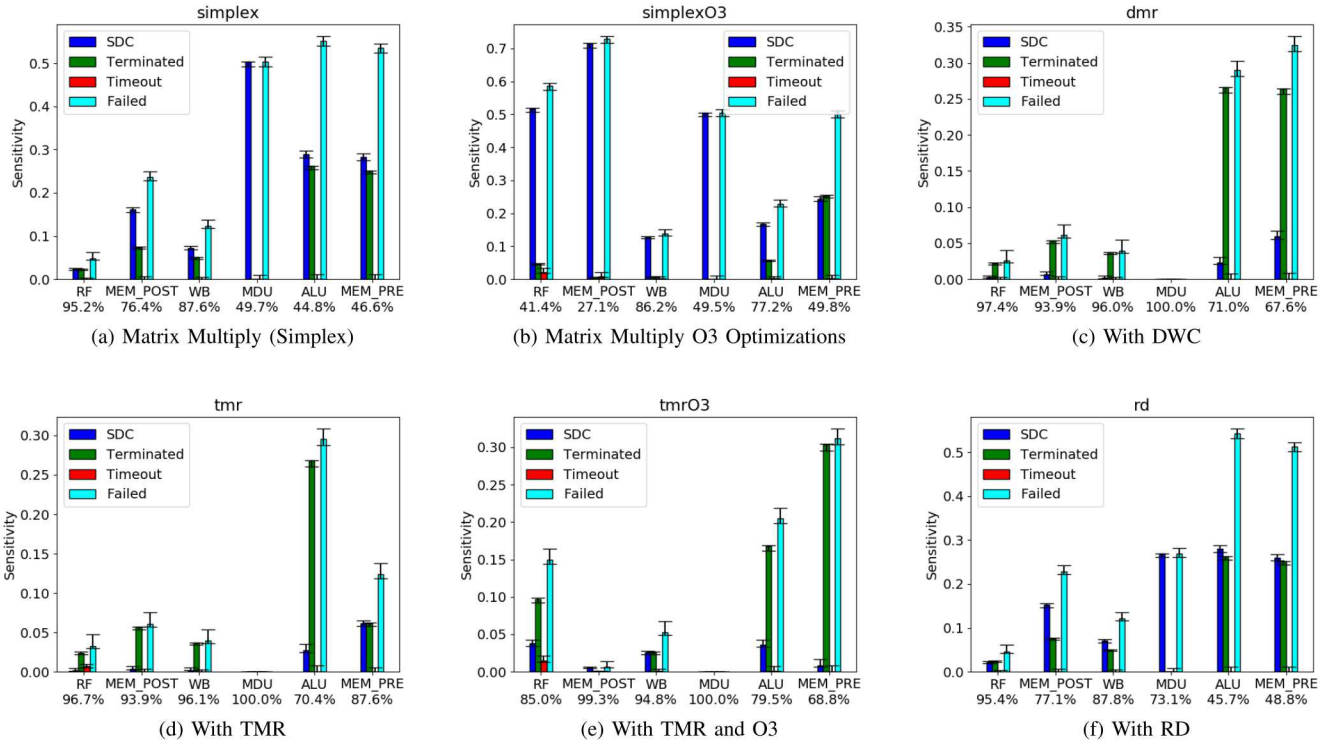
Faults are injected before each cycle of execution either randomly from a probability table, or at specified locations, depending on the fault injection mode. For each fault, we can track when it is injected, how it spreads throughout the system, and if and when it gets masked or quashed (corrected).

Currently, we can inject faults into six locations at a random instruction that results in a random bit flip:

- RF (Register File): random register in the RF;
- MDU (Multiply-divide Unit): Randomly select a high or low output register of the MDU;
- MEM_PRE (Memory, Pre): Randomly select either the address or store value;
- MEM_POST (Memory, Post): Output of the memory stage;
- WB (Write Back): Value written back to the register file;
- ALU (Arithmetic Logic Unit): Output of the ALU. (This does not impact the output of the MDU but rather it impacts the value sent to the MEM stage and operand forwarding).

Previous micro-architectural fault modeling works [5], [6] do not have a method of integrating experimental device data and have investigated the effect of a fault in different parts of the processor. These works use the MARSS and Gem5 simulators, injecting faults on the x86 and ARM architectures. These works can inject faults in any part of the simulator, during any cycle and for any duration. Then the output of the simulator can be compared with a golden model so failures can be detected.

One main difference in this work is that we can track the fault as it propagates through the system. This is accomplished by changing the simulator's reg_word data structure. This structure is used to represent architectural registers and internal state and is normally a simple 32-bit number. For our simulator, we replace it with a custom data structure that records when the fault occurred, which bit it flipped, and the original fault-free value. Whenever an operation is performed (e.g. adding two registers) we update the faulted and fault-free value and propagate faults to the results register. In this way, we can determine when, where and how a fault is quashed or how it



(a) Matrix Multiply (Simplex) (b) Matrix Multiply O3 Optimizations (c) With DWC

(d) With TMR (e) With TMR and O3 (f) With RD

Fig. 2: Failure sensitivity of different processor registers to faults with correct execution percentages shown beneath.

spreads throughout the system to eventually cause a failure. For example, if a register with the value "1" is upset by an SEU in its least significant bit (making it a "0") and is later OR'd with the value "1" it will end up with the correct value as if the SEU never occurred. This sort of "correction by math" can be tracked and counted in the simulator. This capability is useful when designing fault-resistant data encodings or algorithms.

We can also compare the impact of faults in different parts of the processor pipeline. For example, for a given fault, we can determine how many incorrect writes to the register file or to memory will occur. With this fault tracking ability, it is possible to identify what areas of the processor should respond favorably to fault mitigation strategies and how effective fault mitigation strategies are.

Another difference from prior work is that our fault injection program is built using SST which is designed to model large scale systems and will allow us to observe how faults in different subsystems (such as a processor) affect the operation of a larger system (potentially with many processors and devices). These devices may utilize different technology nodes and could have different upset rates in an environment. Once fully built, this would be able to simulate complex systems and be able to track faults throughout them.

Our SST system modeled a MIPS R2000 processor core. We ran multiple versions of an integer matrix-matrix multiply benchmark on the processor. To demonstrate the versatility of the simulation infrastructure, we explored several different types of software redundancy and fault mitigation:

- Matmat/Simplex: A basic, naïve multiplication of two 12x12 32-bit unsigned integer matrices using a triple-nested loop;
- DMR: Multiplication with duplication with comparison (DWC) over the innermost loop. I.e. The inner loop is performed twice, the results are compared, and if they do not match the computation is performed again;
- TMR: Multiplication with triple modular redundancy (TMR). Each multiplication operation (and its operand load instruction) is performed three times. The results are combined in a bitwise majority function;
- RD: Multiplication with reduced precision. A naïve multiplication, but with 16-bit integers.

Each of these four variants was compiled with and without compiler optimizations. The optimized version passes more intermediate values through registers (rather than memory), unrolls loops (reducing the number of branch instructions), and reuses intermediate results (e.g. address offsets) rather than recomputing them. As a result the optimized code runs in about one quarter of the time of the unoptimized code.

Each of the eight executables was run 105600 times, with one SEU-type fault injected during execution at a random register and during a random cycle of execution. Faults were injected equally to each of the six fault sites. The execution was characterized into one of four states:

- Silent Data Corruption (SDC): The program completed, but the results matrix was incorrect;
- Terminated: The program failed to complete, usually because it tried to perform an illegal memory operation;
- Timeout: The program was still running after 4 times the

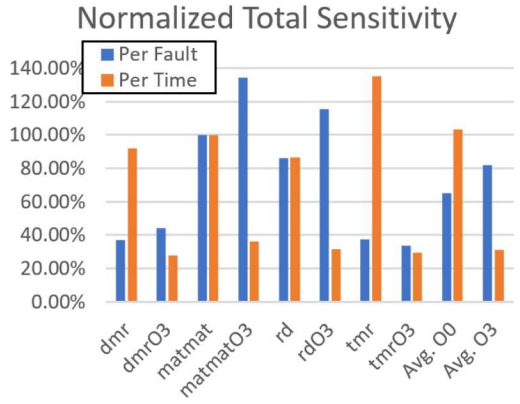


Fig. 3: Summary results of total failure sensitivity, normalized to basic/simplex matmat (with no optimizations).

number of cycles it normally (without faults) completes in. This indicates it is probably stuck in an infinite loop;

- Correct: The program completed and the computed matrix is correct.

The results of our fault injection study are shown in Fig. 2 and Fig. 3.

The different algorithmic variants of the basic benchmark have a substantial effect on failure sensitivity. As expected, the TMR and DMR versions eliminate any risk of MDU faults and the RD version reduces MDU sensitivity by about half. TMR/DMR also substantially reduces the risk of SDC errors from other faults, while having less of an impact on Terminated or Timeout errors.

Compiler optimization also impacts the failure sensitivity. The greater use of registers makes the optimized version 3-11× more susceptible (on a per-fault basis) to Register File (RF) faults and up to 2× more susceptible to MEM_POST faults. Overall, an individual fault is more likely to cause a failure or SDC in the optimized versions. However, because the optimized code runs so much faster, their susceptibility over time is much less. These results show the versatility of our tool and the type of data we can collect.

III. Radiation Events to Pipeline Stage SEUs

The DES is able to incorporate error probabilities calculated from experimental SEE data. The following describes the process starting with experimental transistor level, which is abstracted to a pipeline stage model.

A. Radiation Events to Collected Charge

The total charge collected by individual charge generation due to a particle strikes in a transistor is determined. We follow the methodology described in [7] consisting of two steps:

- TCAD simulations are used to determine the charge collection efficiencies in each sensitive volume of a transistor;
- Monte Carlo Radiative Energy Deposition (MRED) is used to determine the probability distribution and events with maximum collected charge.

Early soft-error, static-upset rate predictions were calculated with a single rectangular parallelepiped (RPP) sensitive volume using tools that assume the same charge collection rate in the entire volume, vastly over-estimating the soft error rate (SER) [8]. To overcome this, we use multiple sensitive volumes each with their own charge collection efficiency.

Next, we combine multiple sensitive volumes with Monte Carlo radiation transport techniques to estimate the total charge collected [9], [10]. This tool is known as Monte Carlo Radiative Energy Deposition (MRED). The power of this approach is that it remains tractable in the absence of simplifying assumptions, and therefore in principle, it is more precise and accurate to predict errors and has been experimentally validated [10]–[12]. In the full paper, this process will be demonstrated using experimental parameters and values for a 14nm FinFET technology.

B. Collected Charge to Single Event Transients

Once the collected charge from a radiation event is determined, we need to convert it to a current source model. Following [3], the collected charge is converted into dual double-exponential current sources. A fast current source provides the charge needed to flip the state and a slow current source models the slow draining of excess charge once the state has been flipped. TCAD models of individual transistors are used to determine the time constants of the current sources. The peak currents of each current source are determined from circuit simulations based on the particular gate type and load and are chosen to flip and hold the output voltage at the opposite rail voltage. Next, the width of the slow current source is determined by setting the integrated charge equal to the collected charge. Finally, the length of rail to rail single event transient is found using a spice simulation of chained logic gates and only transients that are long enough to propagate and be latched are kept. This process allows us to convert the probability distribution of collected charge to a probability distribution of pulse widths. We are currently building a test chip with DFF's and gates to experimentally connect the MRED data to SEU probabilities.

C. Logic Gate to Pipeline Stage Fault Simulation

The next level of abstraction is to create a model of SEU probabilities on the output registers of each pipeline stage in a processor based on probability and pulse width of SETs at individual gates. It is well known that there are logical masking effects that occur within the circuit which naturally quash some SETs before they are latched [13]. This masking effect is usually calculated for the entire circuit as the circuit reliability. We are interested in the probability of a single SET for any given gate within the circuit becoming logically masked before reaching a register.

The probability of error at register r, for instruction i, P_{ri}, is given by:

$$P_{ri} = P_{R-SEU} + \sum_{\substack{t=test\\vectors}} \sum_{g=gates} P_{TV,ti} \times LM_{tigr} \times P_{G-SEU,tigr} \quad (1)$$

$$P_{G-SEU,tigr} = \sum_{\substack{p=pulse\\widths}} P_{SET_{tigp}} \times \frac{T_{SET_{gpr}}}{T_{clock}} \quad (2)$$

- P_{R-SEU} is the probability of an SEU occurring on the register itself in one clock cycle.
- $P_{TV,ti}$ is the probability of each test vector, t, used for instruction i and (for now) is assumed to be 1/(# test vectors). The possible test vectors for each instruction are randomly sampled.
- LM_{tigr} is 1 or 0 and represents whether a SET on gate g is logically masked at register r based on test vector t and instruction i. It is computed using a static fault simulation with the commercial fault simulator ZO1X.
- $P_{G-SEU,tigr}$ is the probability that a SET at gate g that is not logically masked is latched as an SEU at register r, given test vector t and instruction i.
- $P_{SET_{tigp}}$ is the probability of a SET on gate g with pulse-width p, as computed in Section III. As the SET probability depends on the exact data on the gate (rising vs falling edge), it also depends on the test vector t and instruction i.
- $T_{SET_{gpr}}$ is the modified width of a SET as seen at register r. The radiation induced pulse-width r on gate g needs to be modified by the propagation delay to the register if the rising and falling edges have asymmetric delays and by the average latching window of the register. An average value of the propagation delay, independent of the particular test vector, is used to avoid the need for dynamic fault simulations for every test vector. A reasonable approximation is to directly use the radiation induced pulse width, r, and assume symmetric propagation delays and latching exactly at the clock edge.

To demonstrate the creation of an abstracted model, we consider a simple 32 bit RISC ALU. To highlight the impact of logical masking we assume that a SET on each gate has the same probability, P_{G-SEU}, of being latched as an upset and that a single upset occurs somewhere in the ALU: $P_{G-SEU,tigr} = 1/\#\ of\ gates$. The probability of an error due to the digital logic at each output register of the ALU for an add and or operations is shown in Fig 4.

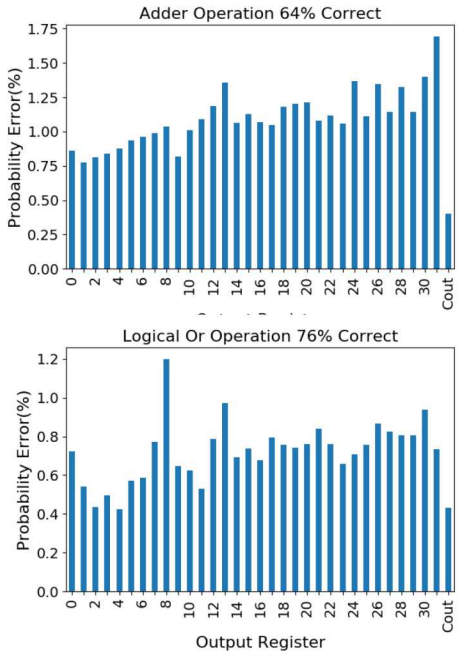


Fig. 4: The probability of an error during an add and or operation at each output register due to the digital logic is shown, average for over one million faults.

IV. Conclusion

Modern computational systems are increasingly susceptible to SEE. It has become increasingly important, but also difficult to model SEE-induced faults in systems with multiple processors. We can address this need by leveraging simulators originally developed for high performance computing to model the effects of SEUs and SETs within the system. By coupling this with physics-based approaches based on experimental data at the transistor level, we can model the effects from the collected charge on a gate to actual errors within the processing system, running real benchmark code. This offers us the potential to study many different fault-tolerant approaches, from radiation hard by design, to applying redundancies in the software. This information can be collected to better inform the design process and allow changes to be made much earlier in the design cycle (and at a much lower cost) to engineer more reliable and complex systems for harsh environments.

The full paper will contain more details about how each of the levels of abstraction interact with each other and demonstrate the integration of experimental data. The pipeline simulator will also include results for other benchmark codes and will show more detailed results such as: how/when faults get quashed within the processor, susceptibility of different registers at different cycles of programming execution (e.g., early, middle and late) and how wide faults can spread to other registers within the processor. Additional future work will include implementation and experimental examination of 14nm logic gate cross sections and the cross section of a complete processor in silicon to validate the processor level experiments and to calibrate the injection probabilities.

References

[1] P. E. Dodd *et al.*, *IEEE TNS*, vol. 57, no. 4, pp. 1747–1763, Aug 2010.
[2] A. F. Rodrigues *et al.*, *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 4, p. 37–42, Mar. 2011.
[3] D. A. Black *et al.*, *IEEE TNS*, vol. 62, no. 4, pp. 1540–1549, Aug 2015.
[4] A. Rogers and S. Rosenberg, "Cycle level SPIM," *Dept. Comput. Sci., Princeton Univ., Princeton, NJ*, July 1993.
[5] K. Parasyris *et al.*, in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2014, pp. 622–629.
[6] M. Kaliorakis *et al.*, in *2015 IEEE International Symposium on Workload Characterization*, Oct 2015, pp. 172–182.
[7] H. M. Quinn *et al.*, *IEEE TNS*, vol. 60, no. 3, pp. 2119–2142, June 2013.
[8] L. W. Massengill *et al.*, *IEEE TNS*, vol. 47, no. 6, pp. 2609–2615, Dec 2000.
[9] A. D. Tipton *et al.*, *IEEE TNS*, vol. 53, no. 6, pp. 3259–3264, Dec 2006.
[10] R. A. Weller *et al.*, *IEEE TNS*, vol. 57, no. 4, pp. 1726–1746, Aug 2010.
[11] K. M. Warren *et al.*, *IEEE TNS*, vol. 56, no. 6, pp. 3130–3137, Dec 2009.
[12] R. A. Weller *et al.*, *IEEE TNS*, vol. 56, no. 6, pp. 3098–3108, Dec 2009.
[13] D. T. Franco *et al.*, *Microelectronics Reliability*, vol. 48, no. 8, pp. 1586 – 1591, 2008, 19th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis (ESREF 2008).