Fault Simulation and Emulation Tools to Augment Radiation-Hardness Assurance Testing

Heather M. Quinn, *Senior Member, IEEE*, Dolores A. Black, *Member, IEEE*, William H. Robinson, *Senior Member, IEEE*, and Stephen P. Buchner, *Member, IEEE*

***Abstract*—As of 2013, the gold standard for assessing radiation-hardness assurance (RHA) for a system, subsystem, or a component is accelerated radiation testing and/or pulsed laser testing. Fault injection tools, which include both fault simulation and emulation tools, have become more common in the last 15 years. Fault simulation tools use analytical methods for assessing RHA, whereas fault emulation uses hardware methods. Both fault simulation and emulation allow designers to augment traditional RHA techniques to determine whether circuit designs, microarchitectures, components, and application-specific integrated circuits (ASICs) meet the requirements for a particular mission. Fault simulation and emulation can provide the designers the luxury of testing on the benchtop without the time and financial constraints of accelerated radiation testing. This paper explores how to design, implement, and validate a fault simulation or emulation system. The paper ends with several case studies of currently used fault simulation and emulation systems.**

***Index Terms*—Emulation, fault diagnosis, radiation hardening, simulation.**

I. Introduction

MOST modern electronic components are affected by radiation that naturally occurs in space and the Earth's atmosphere. For systems deployed into space or the atmosphere, the effect of radiation could cause system failure. Protons and electrons can cause degradation of the transistors' switching capabilities due to accumulated ionizing dose. Protons, neutrons, and heavy ions can cause single-event effects. Unlike accumulated ionizing dose effects, every single proton, neutron, and heavy ion has the potential to trigger single-event effects in a component. There are a number of single-event effects (SEE) that can affect modern electronics, including the following:

- single-event transients (SETs) that cause temporary changes in a gate's output voltage;
- single-event upsets (SEUs) that cause memory values to invert;
- single-event functional interrupts (SEFIs) that cause components to malfunction until reset;
- single-event latchups (SELs) that cause parasitic thyristor in complementary metal—oxide—semiconductor (CMOS) to turn on and draw a high current;
- single-event gate ruptures (SEGRs) that cause power metal–oxide–semiconductor field-effect transistors (MOSFETs) to have a transient electric field across the gate oxide;
- single-event burnouts (SEBs) that cause a forward bias in the parasitic transistor power MOSFETs.

While the first three types of SEEs are not permanently destructive, the last three can lead to the permanent failure of a component.

As modern components continue to shrink in feature size, SETs and SEUs have made it harder to operate fault-free in high-radiation environments [1]–[8]. Despite not causing permanent damage to the component, both SETs and SEUs could affect the computational system such that incorrect output data are generated [9]–[13]. Analog SETs can be especially harmful as they can cause a corruption of input data in the analog-to-digital converter or a loss of lock for the clock signal in the phase-locked loop [14]–[18]. Similarly, analog SETs in voltage regulators and dc/dc converters can harm and/or reset the components they support [19]–[21]. As many deployed systems process their sensor data in real time, data reliability problems could cause the loss of critical sensor data that cannot be recollected. For national security and science missions, the loss or corruption of data is a concern, which has led to an increase in the use of component-level and system-level mitigation techniques for SEUs and SETs to suppress the effect of the fault in the system. For many mission-critical systems, it has become necessary to determine whether the systems are hardened to radiation-induced faults to minimize either unanticipated failures or computationally incorrect output data while deployed. For systems that rely on mitigation techniques to suppress radiation-induced faults, determining whether the system has been properly hardened is essential.

Radiation-Hardness Assurance (RHA) is an experimental evaluation of whether a system, subsystem, component, or design, which we generically call a "design" for the rest of the paper, will operate correctly for the given mission in the

Manuscript received June 29, 2012; revised November 20, 2012 and March 28, 2013; accepted April 01, 2013. Date of publication May 23, 2013; date of current version June 12, 2013. This work has been authored by an employee of the Los Alamos National Security, LLC (LANS), operator of the Los Alamos National Laboratory under Contract No. DE-AC52-06NA25396 with the U.S. Department of Energy. The U.S. Government has rights to use, reproduce, and distribute this information. The public may copy and use this information without charge, provided that this Notice and any statement of authorship are reproduced on all copies. Neither the Government nor LANS makes any warranty, express or implied, or assumes any liability or responsibility for the use of this information. The Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; therefore, the Laboratory as an institution does not endorse the viewpoint of a publication or guarantee its technical correctness. This paper is published under LAUR- 13-22778.

H. M. Quinn is with Los Alamos National Laboratory, Los Alamos, NM 87545 USA (e-mail: hquinn@lanl.gov).

D. A. Black is with Embry-Riddle Aeronautical University, Albuquerque, NM 87117 USA (e-mail: black6@erau.edu).

W. H. Robinson is with Vanderbilt University, Nashville, TN 37235-1824 USA (e-mail: william.h.robinson@vanderbilt.edu).

S. P. Buchner is with the Naval Research Laboratory, Washington, DC 20375 USA (e-mail: stephen.buchner@nrl.navy.mil).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNS.2013.2259503

given radiation environment. For many missions, RHA testing can significantly reduce the risk of having failures in deployed systems, as the testing provides an understanding of 1) how the design is affected by radiation and 2) whether the design is hardened enough for the given mission. It should be noted that although a system could fail from multiple causes while deployed, RHA focuses on failure mechanisms that are induced by radiation.

Mission requirements can vary greatly from mission to mission and can define many important parameters for RHA testing, including the mission location, the mission length, the availability requirements, and the reliability requirements. These five factors affect RHA testing in these ways:

1) *mission location:* determines the radiation environment, including the fluxes for electrons, protons, neutrons, and heavy ions;
2) *mission length:* determines how much ionizing radiation the system will experience and determines the probability of highly energetic ions for single-event effects;
3) *deployment date:* determines what solar cycle is relevant for the mission;
4) *availability requirement:* determines how much downtime the system is allocated per year and whether there is a ground intervention limit per year;
5) *reliability requirements:* determines how reliable the system or components must be and whether there is a requirement for handling potentially incorrect data sets.

Not only are these five factors necessary guidelines for system design, they also need to be defined before RHA testing is started. With clear guidelines, needless overdesign and overtesting can be minimized.

Once the mission requirements are determined, then it is possible to develop a system design and then construct RHA tests that can determine whether a design meets the mission's needs. While this paper focuses predominantly on SEE hardness, it may also be necessary to determine the effect of accumulated ionizing dose for space missions or prompt dose radiation for nuclear survivable missions. It should be noted that often the results from RHA testing are only relevant for that particular design on that mission. Therefore, designers should not expect to use RHA results from one mission for a different mission or from one design for a different design.

Currently, the gold standard for determining RHA is through accelerated radiation sources and pulsed lasers. In both of these cases, the radiation source and the pulsed laser are used to induce faults in the design under test (DUT) to determine whether the fault translates into an observable error. While accelerated radiation and other radiation sources are the basis of most of the research in the radiation effects community, using pulsed lasers has become more common in recent years. As accelerated and radioactive sources strike the DUT randomly, it is possible that these methods could miss the critical node that induces a particular fault, if the test is not run long enough or the critical node is difficult to reach due to packaging constraints. While these problems can also be true in pulsed laser testing, these problems can be minimized in pulsed laser systems as the granularity of data regarding both spatial and temporal information about the induced fault in finer.

Recently, many organizations have been using *fault emulation* and *fault simulation* techniques to augment traditional RHA techniques. Fault emulation methodologies depend on techniques that can mimic the fault's behavior in the DUT, whereas fault simulation methodologies depend on modeling the fault's behavior in the DUT. As there is a great deal of overlap in the discussion regarding the background, methodology and validation for these two topics, we use the term *fault injection* to indicate where fault emulation and fault simulation can be used interchangeably.

Fault injection techniques have been used widely in the computer design field for many years. While not a replacement for standard RHA methodologies, fault injection can help designers explore how faults affect the DUT in a quick, inexpensive manner. In some cases, it is possible to perform root-cause analysis and identify all potential failure modes in a DUT, whereas the limitations of time and finances can make such an exploration in an accelerated radiation facility infeasible.

The paper is organized as follows. Section II gives background information on fault injection with an example of how fault injection techniques can be used on a microprocessor-based system. Sections III, IV, and V cover fault injection methodologies, techniques, and validation. In lieu of a traditional related work section, we will cover related work throughout the paper. We close the paper with case studies in Section VI that provide an overview of successful fault injection experiments.

II. Background on Faults, Errors, and Scope

An important aspect of fault injection and RHA testing is to determine how faults affect the design. While charge generation induced by ions causes the fault in the design, the effect of the fault might be seen at different levels or abstractions of design based on how faults propagate. For example, a fault in a transistor might be observed at the gate's outputs or even at the design's outputs, if it propagates through the design. For this paper, unless specified otherwise, the abstraction that we will use is shown in VI. The abstraction covers a wide array of scenarios from the transistor level to the entire system.

This abstraction can be useful for categorizing fault injection tools. While it would be ideal to insert faults at the device level in all fault injection tools, it can be more expedient to inject faults at different layers of abstraction. Caution must be taken to make certain that faults inserted into higher layers of abstraction are reasonable analogs for how radiation-induced faults present in the system. As discussed later in the paper, many of the currently available fault injection tools in the radiation effects community focus on injecting faults in the device, circuit, or component layers, but fault injection tools for higher levels of abstraction are both possible and useful.

A. Fault Propagation and Scope

Because not all faults propagate through all of the abstraction layers, it is necessary to differentiate between faults and errors. In fault injection, the term *error* is often used to indicate faults that cause an observable perturbation on the output of the design. Fig. 2 illustrates the difference between faults and errors in this context. In general terms, an error is a fault that is observable from outside the scope boundary. Faults that do not translate into errors and are masked within the scope can often be ignored, as there is no observable effect outside the scope.

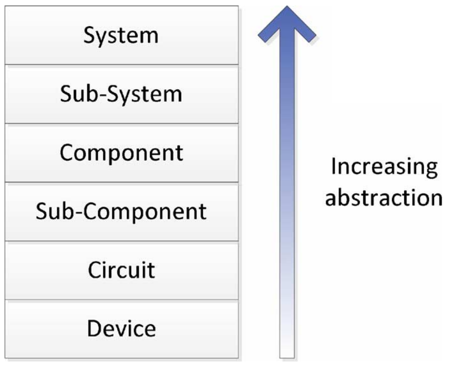

Fig. 1. Layers of abstraction.

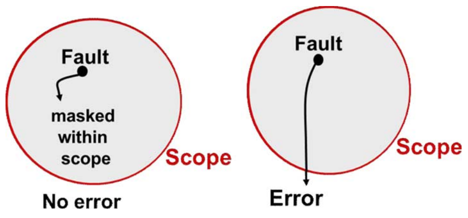

Fig. 2. Illustration showing how the fault must be able to translate outside the scope to determine whether it is an error.

For example, the scope might be a component. Faults will only be categorized as errors if the fault can propagate to the component's outputs.

Unobservable faults can be classified as either masked faults or latent errors. In the case of masked faults, either the design's logic or the timing of the fault cause the fault to not be observed. For example, a fault on the most significant bit of a leftward shifter would not be observable, as the shift operation would shift out the fault. Latent errors are not masked but are rendered unobservable due to the window of operation of the design. There might be specific operational loads—a particular input combination on the circuit, a workload on the component—that could make the latent error observable.[1] As the user of a fault injection system has a greater observability into fault propagation, it is possible using a fault injection tool to categorize faults as errors, masked faults, and latent errors. This type of classification can be useful in determining what percentage of the faults are errors and which faults the design can naturally mask.

The computer architecture community uses different concepts for discussing how masking might affect error rates. In [23], Seifert *et al.* emphasized that the soft error rate (SER) of modern microprocessors with large caches or large memory arrays are usually protected with an error correction code (ECC), and therefore the failure rate of the device is dominated by the contribution of sequential elements. Equation (1) can be used to estimate chip level SER for the nodes within the circuit [23]:

$$\mathrm{SER}^{\mathrm{circuit}} = \sum_{i}^{\mathrm{all\ nodes}} \mathrm{SER}_i^{\mathrm{nominal}}\ x\ \mathrm{TVF}_i\ x\ \mathrm{AVF}_i$$

where the nominal $\mathrm{SER}^{\mathrm{nominal}}$ is the un-derated SER and is independent of the circuit environment, TVF refers to the timing vulnerability factor, and AVF refers to the architectural vulnerability factor. The TVF is defined as the fraction of time a storage element is susceptible to upsets, and AVF is equal to the probability that a fault in the storage element will be observed at the output [24].

[1]For a more in-depth discussion on latent errors, see [22].

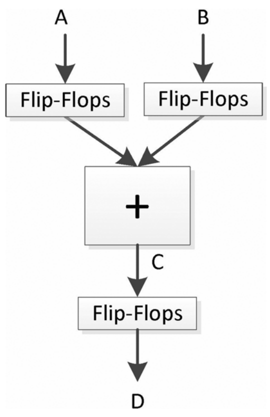

Fig. 3. An adder with flip-flops on the output to register data.

To explain these concepts, we provide an example that shows how fault injection can be used on a microprocessor-based system. This example shows how faults and errors in a device translate into faults and errors in the full system.

1) Initial Scope: An Adder With Flip-Flops: Let us start our example with an adder, as shown in Fig. 3. The design in this case is an adder that registers its input and output data into flip-flops (FFs). Right now, the scope of the fault injection is just the adder and the FFs, which means that the fault injection system looks for errors on the D outputs. It is possible with this design to perform fault injection on two types of faults—SET fault injection on the adder gates and SEU fault injection on the FFs.

In the case of SETs, transients are injected using one of three methodologies:

1) instrumenting XORs on the input lines of the design and injecting the transient into the design through the XORs;
2) injecting the transients onto the output of each gate in the design [25];
3) injecting the transients into each transistor of each gate in the design [7], [26].

All three of these techniques show how designs are instrumented so that faults can be inserted into design while it is operating. While all of these techniques leverage SPICE or HSPICE for simulating how the fault affects the design, these three techniques show a progression toward more sophisticated techniques for instrumenting the designs so that faults can be inserted with finer granularity. In the final case, the design no longer needs to be modified to be instrumented, and charge generation using MRED inserts the transients into transistors.

With SET fault injection, it would be possible to explore how faults can propagate through the adder circuit and with the right timing to be latched into the FF. Injecting SETs across the entire clock cycle allows the designers to explore the timing between the transient and the FF's setup and hold time period. Because of this timing aspect, not all SET faults translate into errors. In this case, SET fault injection allows the designers to determine

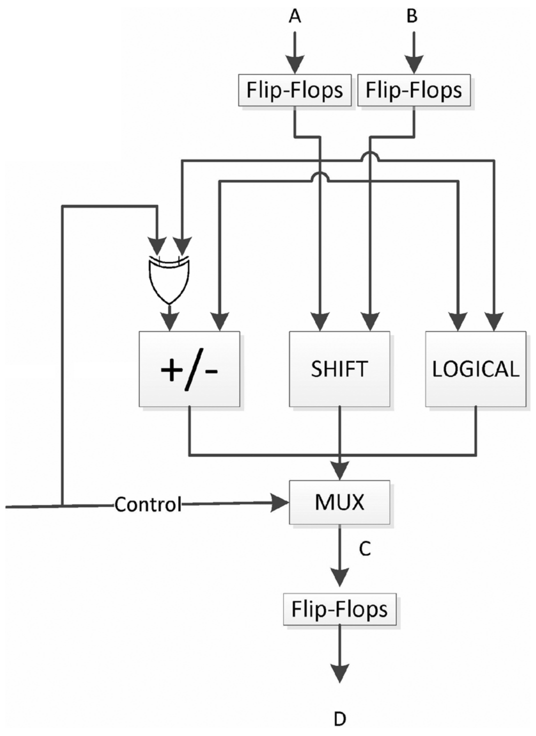

Fig. 4. An ALU with three operations.

which of the transistors and what timing is necessary to create SET faults that propagate to the D outputs.

SEU fault injection for this design is simpler than SET fault injection. There are a number of different techniques for injecting SEUs, including the following:

- injecting SEU into the design's FFs through either a boundary scan or reconfiguration ports [27];
- injection SEUs into the design's FFs by instrumenting the inputs to the FFs [28].

The faults in both cases would be injected directly into one of the FFs, indicating that both methods work at the circuit level of abstraction. For this design, all SEU faults propagate as errors on output D, due to the simplicity of the design. In AVF terms, that means there is no logic derating for this design.

2) Second Scope: The Adder in an Arithmetic/Logic Unit (ALU): In our initial example, it is straightforward to determine the faults that could occur in the design, the scope and errors. In our next example, we widen our scope to a functional unit that includes the adder with FFs from our initial design. In this example, we examine an ALU, which is a critical subcircuit in a microprocessor. The ALU in our example, shown in Fig. 4, includes the ability to add/subtract, shift, or perform logical operations on two inputs.

The ALU scope is similar to the adder's scope, as the scope boundary is still the output D. Furthermore, the same SET and SEU fault injection techniques would be relevant for the ALU design. Unlike the previous example, the ALU design includes two different aspects:

1) control inputs and devices used to determine which arithmetic or logical unit to connect to the output FFs;
2) multiple processing "modes."

While both the control logic and the extra processing modes add more logic to the SET fault injection scenario, the complication from the extra logic comes in the form of fault masking. For example, if the ALU is subtracting A from B, any faults injected into the shifter are logically masked, as the multiplexor is filtering these signals already. If each of the arithmetic and logical units were used equally, then we would expect that the original adder errors we found were at most a third of the potential ALU errors. In AVF terms, this example exhibits logic derating, which can be determined using fault injection tools.

3) Third Scope: Microprocessors: We now widen the scope to a microprocessor. A basic overview of microprocessor design is shown in Fig. 5, where it is inserted in a simple microprocessor-based design that will be used for the next scope. The scope for this design is the outputs of the microprocessor. The additional control structures create more opportunities for logical masking and logic derating. The addition of registers and caches for storing data create more opportunities for SEU-induced faults. Finally, the amount of detail in the design and the number of locations for both SEUs and SETs that can be introduced into the system often lead researchers to use abstractions to help manage the complexity of the design and its interactions with faults. Microprocessor reliability is the motivating reason for AVF and TVF [30]. AVF and TVF analysis has determined the areas of the microprocessor most likely to cause errors. A longer discussion of fault injection for microprocessors is given in the case studies in Section VI.

4) Final Scope: Microprocessor-Based Systems: In our final scope, we are inserting the microprocessor into a full system, as shown in Fig. 5. In this case, there are off-chip memories, clocks, and a bridge chip. Because the memory is used to store the program and the operating system, it is possible that faults in the memory could cause errors in the application. Furthermore, faults on the clock can cause the microprocessor to operate incorrectly. In this example, the scope is the output of the system.

As with the previous scope, abstraction is helpful at managing the complexity of the system. For this design, the most relevant abstraction to use is one from the computer architecture field, called the *system stack*. In the system stack, the computing system is divided into layers, as shown in Fig. 6.

There is a wide diversity of approaches at this stage:

- inject faults into one layer and observe the outputs in the same layer;
- inject faults into one layer and observe the outputs at a higher level of abstraction;
- inject faults into one component and observe the outputs in a different component.

There are two areas we would like to focus on in this scope: mixed-signal fault injection and software/hardware fault injection. At this scope, we can now envision how faults from the analog components or even the sensors enter into the computation system as errors or how faults from the hardware enter into the software layers as errors. The analog/digital and software/hardware boundaries are both important aspects in real systems. An extension to AVF, called program vulnerability factor can be useful for determining the reliability of software [31]. Fault injection systems focusing on these two aspects of the full system are not as mature as fault injection tools that focus on the hardware layers of the system stack but could likely be an important innovation in the next decade. While full-system fault injection

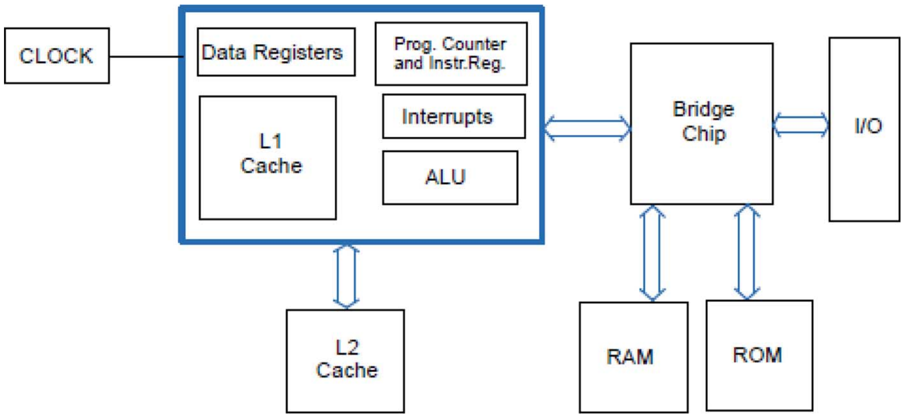

Fig. 5. Simplified view of the organization of a microprocessor in a microprocessor-based system [29].

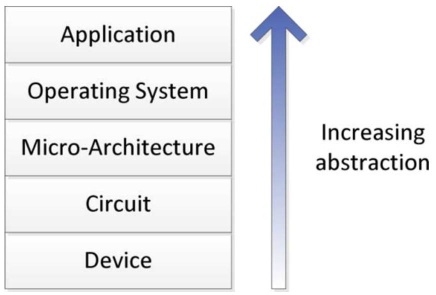

Fig. 6. Traditional computing system stack.

tools are not common at this stage, being able to widely test the system on the bench will provide designers with the opportunity for determining how faults in one component affect the system's reliability.

III. Fault Injection Methodologies

In this section, we discuss how to design fault injection tests or campaigns. While the Section IV discusses the mechanics of inserting faults, this section is specifically concerned with how to develop fault injection campaigns that can be validated with accelerated radiation or pulsed laser testing. Many of these concepts overlap with methodologies for accelerated radiation and pulsed laser testing, which means that there could be a strong overlap between the fault injection methodology and the validation methodology.

An important aspect of a fault injection methodology is *test coverage*, which determines what percentage of the design was tested. There are four aspects that need to be covered in general:

1) the fault model;
2) the fault locations for each fault in the fault model;
3) the input vector set;
4) the operational timing of the fault with respect to the execution of the design.

A figure that illustrates the test space for input vectors, fault locations, and timing is shown in Fig. 7. Most fault injection tools are designed to insert only one type of fault, such as SEUs or SETs, which leaves the need to optimize the other three aspects.

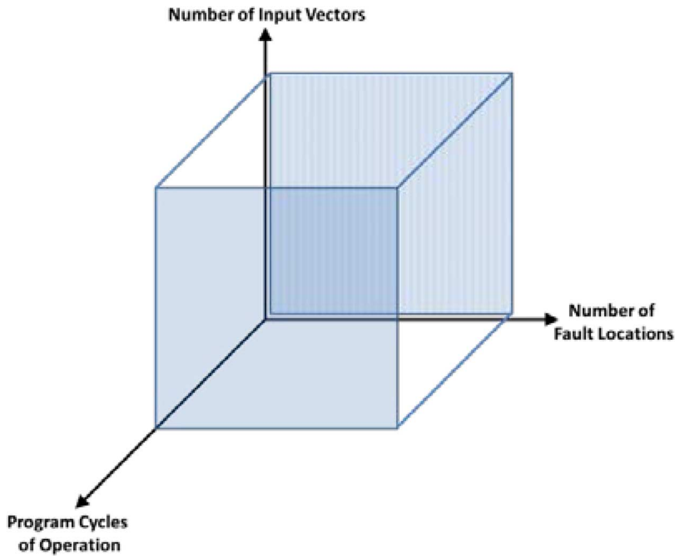

Fig. 7. Test space for fault injection.

It should be noted that it might take the use of several fault injection tools to cover the entire test space for the DUT.

This same test space exists for the accelerated radiation and pulsed laser tests, except that the method of covering the fault locations and fault models axes differs. The concern with fault injection is that designers must cover enough of the test space to ensure that the test space covered by accelerated radiation testing overlaps. Fig. 8 shows the space of all the possible faults for a design. Fault injection, pulsed lasers, and accelerated radiation tests allow the designer to cover some or all of the entire space. Generally speaking, validating a fault injection tool is simpler when the region sampled from the entire fault space is easily covered by both fault injection and either accelerated radiation or pulsed lasers.

Ideally, the fault injection methodology would be able to cover all three axes of the test space. Realistically, with millions of fault locations, millions of program cycles, and trillions (or more) input vectors to cover, complete test coverage is impossible. Even with hardware acceleration in fault emulation, completely covering the test space for most designs is not possible. Fault injection time is often reduced by sampling from the test space, instead of completely covering the test

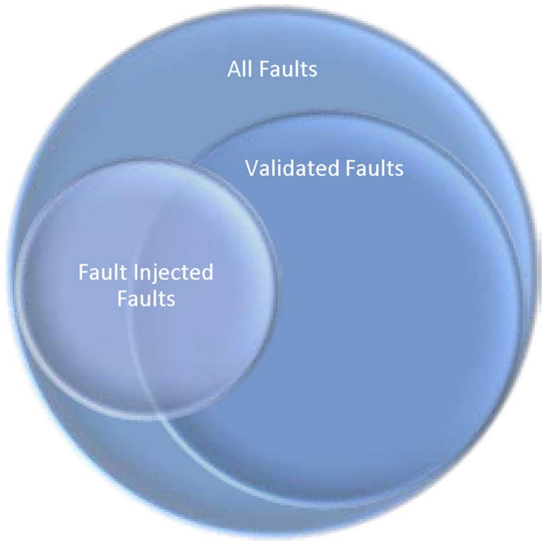

Fig. 8. Space of all faults showing the coverage for fault injected faults and validated faults.

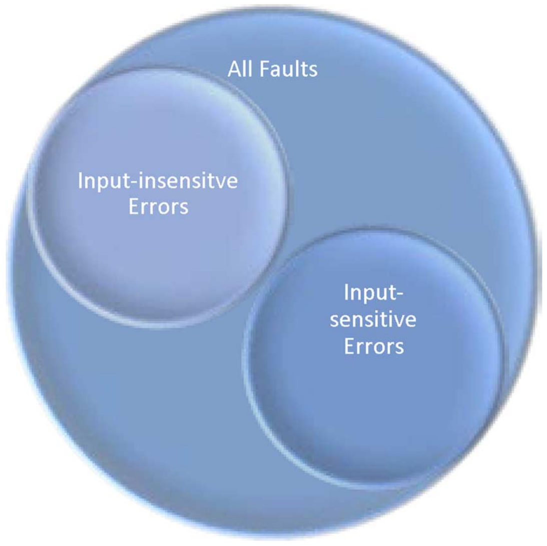

Fig. 9. Space of all faults showing faults that are sensitive to input vectors and faults that are not.

space. One such method is called *statistical fault injection* (SFI) [32] and focuses on covering a subset of fault stimuli chosen at random locations throughout the design. The major drawback of SFI is that the decrease in fault injection time is often directly proportional to a decrease in the accuracy of the results.

For complex ICs, trading accuracy for test time is necessary not only for fault injection testing but also for radiation testing. In this way, the test space cube shown in Fig. 7 can be seen as a volume to optimize. Every fault injection test falls on some surface that is defined by the percentage of the input vectors covered, the percentage of the fault locations covered, and the percentage of the program cycles of operation covered. In that way, the designer can optimize the tradeoff between the time spent on fault injection and the test space coverage.

In studying our own and other's fault injection tools, we have found that many tools attempt to reduce the test space by deliberately reducing the test space to a two-dimensional space by using only one input test vector set or one programming cycle of operation. Even after reducing the test space by one dimension, it might still be necessary to sample from the two remaining axes. It should be noted that sampling is necessary, but the extent of sampling should always be included in discussions on results of the fault injection testing. In the remainder of this section, we discuss the effect of sampling from the three axes of the test space. We will also discuss the use of confidence intervals, standard deviations, and variants for determining the effect of sample size on the quality of the fault injection results.

A. Sampling the Test Space

In this section, we will provide a quick overview of the advantages and disadvantages of sampling from the test space. We break the discussion into input vectors, fault locations and operation timing.

1) Input Vectors: Sampling from input vector sets is very common, as input vector sets are often orders of magnitude larger than the number of fault locations and program cycles. As shown in Fig. 9, the fault space can be separated into 1) errors that can be observed with most input vectors and 2) errors that can only be observed with particular input vector combinations. This second category of error might be largely classified as latent errors, as only specific workloads cause the error. The primary concern for undersampling of the input vectors is that these latent errors might not be explored. Because the population of input-insensitive and input-sensitive errors is design dependent, widely exploring the input vector space will help designers to determine the input-sensitivity of the design.

It should be noted that with input vectors, there is a real use-case problem at hand. Assuming it would be possible to test enough input vectors, then it would be possible to find the complete set of errors, including both input-sensitive errors and input-insensitive errors. In this case, the reliability assessment is likely the lower bound of the reliability calculation. On the other hand, a really narrow input-vector set might not find any input-sensitive errors, providing an upper bound to the reliability. The most realistic bound would use the expected data from the input source, such as from a sensor. It is suggested in [33] that a goodness-of-fit test can be useful to determine whether the input vector set used for fault injection adequately agree with the input data the deployed system expects to use.

2) Fault Locations: The process of injecting faults into a design should provide the designers an understanding of locations in the design that trigger errors. Some papers from the literature report their techniques to sample from the set of all fault locations. For some designs, it might not be possible to sample all fault locations, especially in fault emulation where some fault locations are not accessible in the hardware. The primary concern for not covering all of the fault locations is that the results might not be easy to validate, as radiation can inject faults widely.

Unlike input vectors, sometimes fault locations are not randomly distributed in design. Some errors will cluster based on logical or physical characteristics. For example, with field-programmable gate array (FPGA) designs, errors often cluster physically around and in areas of unmitigated logic [34]. Furthermore, there can be logical characteristics, such all cells of the same function, that are likely to cause errors [35]. As another example, well contacts might also affect how many bits an SEU affects in memory [36]. Therefore, multiple-bit upsets will be more likely found further way from the well-contacts. These types of clustering can help focus fault injection using stratified samples [33]. Once these types of characteristics are determined, it is possible to use this information to create a targeted fault injection test campaign for subsequent passes.

3) Operational Timing: Timing plays an important role in fault injection and can vary widely depending on what level of abstraction is used by the fault injection tool. For tools that inject SETs into devices, it is necessary to distribute faults uniformly over the clock cycle to determine how timing affects the ability to latch an SET. At a higher abstraction, some faults, especially in microprocessors, are sensitive to where the fault occurs in the execution of the software, such as whether the fault is injected before or after a register is read. In these systems, the fault injection system should distribute faults uniformly over the execution of the software or should inject faults into an operational phase most likely to cause an error. The primary validation concern is that the validation process might not be able to insert the fault with the same timing accuracy as the fault injection or not be tested for all operational phases of the design. By statistically sampling widely across all timing considerations—both within a clock cycle and within design execution—it is most likely that fault injection will mimic how faults behave in realistic radiation environments.

B. Sampling Error and Confidence Intervals

In the previous discussion on sampling from the test space, there are two main concerns:

1) sampling error;
2) coverage error.

In sampling error, problems can arise from undersampling and biased sampling, due to either not randomly sampling or disproportionately favoring subpopulations. There is also the risk of coverage error, where a sample is incorrectly excluded from the survey. While sampling error can be addressed through confidence intervals, coverage errors can represent a systematic error in the fault injection system.

Confidence intervals, standard deviations, and variants can be useful in discussing how the sample size affects the fault injection results. An excellent discussion of these topics from a statistical view of simulation can be found in [33]. Kleijnen states, "The sample size effects the 'statistical reliability' of the estimated response of the simulated system [33]." Kleijnen's suggestion is that the "standard deviation of the mean response for the system variant" [33] is a good gauge of the reliability of the measurements made by a simulation system. To this end, fault injection reliability can be increased by increasing sample size. Therefore, as standard deviation is proportional to $1/\sqrt{n}$, where n is the sample size, decreasing standard deviation involves increasing sample size by integral factors of two. Later in the book, Kleijnen suggests using the Student t-statistic to define confidence intervals for n observations (samples) and the set of average observations $\bar{x}_i (i = 1, \ldots, n)$:

$$\text{confidence interval} \equiv \left(\bar{\bar{x}} \pm \frac{t_\alpha s}{\sqrt{n}} \right)$$

where $\bar{\bar{x}}$ is the overall average of the averages from $\bar{x}_i$, s is standard deviation from $\bar{x}_i$, and α is the confidence interval wanted (i.e., 95% or 99%). Finally, the book includes an entire chapter on reducing the variant of the observed data such that the confidence limits can be minimized to the designer's needs and has many suggestions on how to increase fault injection reliability.

There are also a number of methods for defining confidence intervals that are specific to fault injection studies. Leveugle *et al.* [37] develop a method using the margin of error based on the population size, the probability that a member of the population displays the specific characteristic, and the confidence level to determine the sample size. In this method, if margin of error is kept at 95% or lower, then small sample sizes might adequately represent larger populations. Cukier *et al.* [38] builds off of mathematical reasoning similar to [33] and has detailed discussion of many factors that play a role in confidence limits in fault injection systems. As we cannot give [38] the treatment warranted in a short discussion, the authors strongly advise the reader to study this paper in depth. Finally, Constantinescu [39] provides correlation between coverage probability and confidence intervals using sample size that can be useful in determining how large of a sample size is needed to meet a particular coverage probability.

It should be noted that some of the concerns with sampling from the test space could be alleviated by designing the tests to cover one axis completely. In statistics, this situation is the difference between a census and polling. While polling has many of the sampling concerns that we have presented in this section, census testing should be free of both sampling and coverage error. Census testing can avoid these problems by looking at the entire population, as long as representation of subpopulations is not a problem. Because input buses are often quite wide, a census of either the fault locations or the timing is most reasonable. In this manner, the results of validation should be a complete subset of the faults found during fault injection.

While census testing might seem like an unreasonable amount of time to spend testing the design, an automated fault injection tool should be able to test the design with a minimal amount of human interaction, unlike accelerated radiation testing. Multiple fault injection tools can also be run in parallel.

IV. Fault Injection Techniques

In this section, we cover fault injection techniques for both fault emulators and fault simulators. As the injection techniques are dissimilar for fault emulation systems and fault simulation systems, we cover these topics separately.

A. Fault Emulation Techniques

While fault emulation depends on a hardware-in-the-loop technique for inserting faults into the DUT, the fault emulation system is often a hardware-software codesign [40] with varying amounts of hardware and software based on the system design. It is necessary to prepare the design for fault emulation by:

- developing the fault emulation system;
- implementing the design for the hardware aspect of the fault emulation system;
- developing the workload or input vectors for the test.

As we have already discussed issues with input vectors in Section III, we focus on discussing the first two topics in this section.

1) Fault Emulation Hardware: A necessary aspect of fault emulation is the hardware used for fault emulation. Generally, fault emulation is done either on the actual hardware or on proxy hardware. With proxy hardware, the design is ported to a hardware component that is presumably easier on which to emulate faults. Proxy hardware is often used when fault emulation on the implementation hardware is difficult or slow. FPGAs are popular for proxy hardware due to the ability to implement many different types of designs and the ease of emulating SEUs and SETs.

If the actual hardware is being used for fault emulation, then it is necessary to be able to access the internal logic through either boundary scan or write/reconfiguration ports. One of the most powerful tools in fault emulation today is the JTAG boundary scan port [41]–[43]. Depending on how the hardware manufacturer implemented the JTAG specification, it might be possible to access a large percentage of a hardware component's memory locations. Both Freescale and Texas Instruments have JTAG implementations that can read or write to the registers and the caches.

If the faults are injected through hardware mechanisms, then the hardware portion of the fault emulation system needs to expose any necessary ports or pins required to insert the faults, such as a boundary scan port. The input and output pins also need to be exposed so that input vectors can be written into the hardware, and output vectors can be read from the hardware. Finally, it might also be necessary to expose any signals necessary for instrumented designs.

In general, a fault emulation system should be able to handle different types of designs. With many fault emulation systems, the design and placement of input and output pins, especially clocks and resets, are fixed. Due to these restrictions, sometimes the design has to be modified to fit the fault emulation system, which can reduce the usefulness of the tool.

Finally, it might also be necessary to prepare the design for implementation in the fault emulation system, including translating the design for proxy hardware. It might also be necessary to instrument the design for fault insertion [44], [45]. As a general rule of thumb, any design changes run the risk of increasing the uncertainty in the fault emulation results, especially for SETs that are dependent on timing to propagate.

2) Basic Fault Emulation Algorithm: The basic fault emulation algorithm for injecting faults is shown in Fig. 10. The realization of the algorithm is often hardware-software codesigned with the hardware and software aspects working in tandem. While the hardware used to implement the design is the heart of the system, software control is common, especially for logging procedures. The hardware aspect of the system implements the design plus any hardware support for emulation, including instrumentation. It should be noted for electrical pulse injection, as done by [46], the algorithm would need to be modified to allow for the electrical pulse to generate the fault signal.

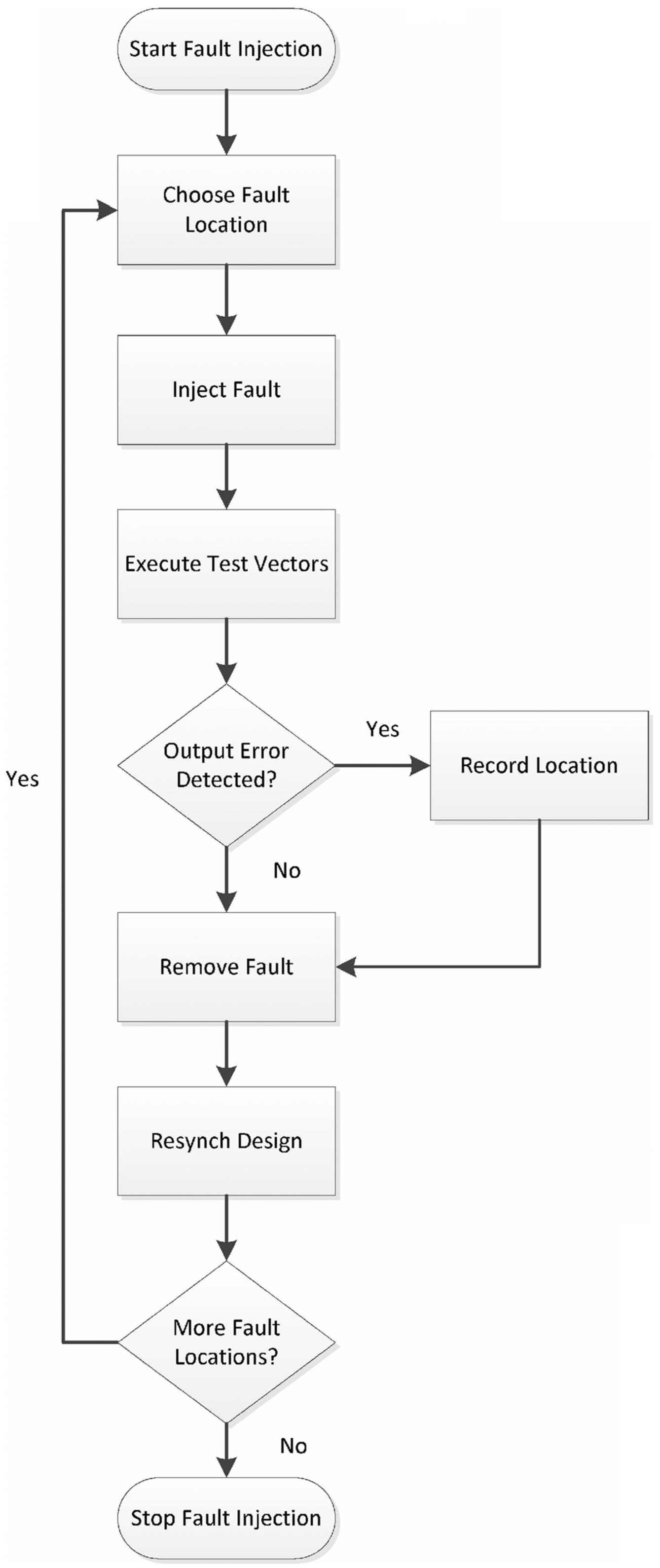

Fig. 10. Basic Fault Injection Algorithm based on [47].

There are two implicit wait cycles in the fault injection algorithm. One is when the test vectors are being executed. While output errors will likely be detected in real time, there could still be a time lag as each test vector is input into the system. Furthermore, there is a multiplicative effect, as the same number of input vectors will be run for each fault location. Therefore, the runtime for the fault emulation system is proportional to $\mathrm{fault\ locations} * \mathrm{input\ vectors} * \mathrm{timing}$. The other wait cycle is during the resynchronization of the design, which we will discuss in more detail in the following section. While the first wait

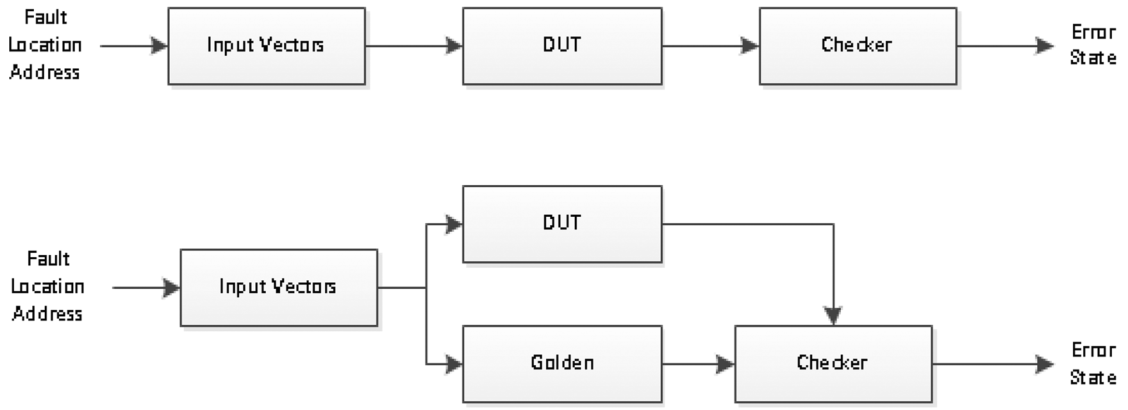

Fig. 11. Two methods for doing error detection.

cycle can be controlled by decreasing the number of the input vector set or the number of fault locations, the second one is likely a set delay in the system.

3) Performing a Measurement: At this stage, performing a fault emulation test is often as simple as uploading the design to the hardware fault emulation system and allowing the system to systematically follow the algorithm shown in Fig. 10. There are three aspects that need to be kept in mind:

1) error detection;
2) fault correction;
3) fault isolation.

In this section, we discuss these three topics.

Once the fault has been injected into the design, the design needs to be monitored to determine whether an error transmits to the output of the design or to the scope boundary. Proper error detection depends on waiting long enough for faults to propagate through the design, and determining—often in real time—whether the output data is correct or incorrect. The fault location causing the error and the error (when possible) should be logged for later analysis.

Fig. 11 shows two methods for error detection using a checker. In the bottom case, the checker includes a golden copy of the design. The fault emulation system designed in [48] used a hardware checker that depends on real-time error detection that must be designed for each DUT. The LANL fault emulation systems use lockstep hardware with error detection handled by the golden hardware component that contains a comparator [9], [49]–[53]. In the Flipper system [54], inputs and golden outputs from simulation are stored in memory on the computer and compared with output from the DUT. It should be noted that lockstep-based fault injection is more useful in simpler circuits or FPGAs, but likely less helpful for circuits with complex dynamic behavior, such as microprocessors.

Once the fault has been injected and the outputs monitored for error conditions, the fault needs to be removed. The design needs to be returned to normal operation before the next fault is injected. Removing the fault from the fault location might be as simple as using the fault injection process on the same fault location to correct the fault. In other cases, it might be necessary to restart or reinitialize the system.

If restarting the system is unnecessary, it is important to make certain the fault from the design's state is removed. Practically, it makes more sense to remove the fault from the design's state manually. This process often involves resetting the design and resynchronizing the fault injection system, as these steps are more likely to make fault injection trials independent and reduces the risk of a latent error in the system. Due to the complexity of detecting latent errors, it is often simpler to restart or reinitialize the design after removing the fault. Kleijnen [33] provides a useful discussion on biases caused by initial and transitory states in simulation system.

Isolating faults is an important aspect of design resynchronization. While there are times where injecting multiple faults at one time are the intention of the fault emulation system, that scenario should be treated as a special case. With multiple faults in the system at one time, determining the effect of individual faults in the system is difficult. Independent trials ensure that errors are properly attributed to the correct fault location and that a latent faulty state from one fault injection iteration does not affect the next one.

4) Current Fault Emulation Systems: FPGA-based fault emulation systems have been popular in recent years, both as a method to conduct fault injection studies on FPGA reliability and as proxy hardware. FPGA-based fault emulation systems often use either the JTAG boundary scan port or the wider SelectMAP reconfiguration port to insert faults into the FPGA by using partial reconfiguration commands. Newer systems leverage the internal configuration access port (ICAP) for reconfiguration. LANL and BYU designed one of the first fault emulation systems for FPGAs with the SLAAC1-V SEU Emulator [47]. Since then many other organizations have created FPGA fault emulation systems [48], [54]–[58]. LANL has also gone on to make other versions of their fault injection tool to support newer hardware devices [41], MBU testing [9], and multiple-independent upset testing [53].

There have also been a handful of fault emulators that use instrumented designs for SET injection on an FPGA [59]–[61], especially for Flash-based FPGAs. In [46], electronic pulses are inserted in the circuit using gate-level instrumentation of the circuit. The instrumentation circuit is connected both to the input pads and the circuit so that an external signal generator can be used to drive an input into the instrumentation circuit, which can translate the signal into a pulse that is driven into the user circuit.

There are also a number of microprocessor fault emulation systems that use interrupts, boundary scan, or software fault injection. Software-based fault emulation uses code modification to insert SEUs into the microprocessor [62]–[65]. As a variation of software-based fault injection tools, interrupt-based methodologies attempt to inject faults into registers or the cache using interrupt controller routines [66], [67]. Boundary scan ports, including the JTAG interface, have also been used to insert faults directly into microprocessors [68]–[70].

One of the most popular FPGA-based proxy systems in the radiation effects community is the FT-UNSHADES emulator that allows for emulation of radiation-hardened by design ASICs on a Xilinx FPGA [44], [71]–[77]. FPGA-based fault emulators have also been used as proxy hardware for "soft" microprocessors that can be implemented in either ASICs or FPGAs [43], [45], [71], [78]–[81]. The use of FPGAs as proxy hardware is less mature than many other fault emulation systems. While FPGAs provide a flexibility that is capable of implementing many different types of circuits, it is possible that an FPGA implementation of a circuit may alter the way faults are propagated. In particular, FFs could be optimized differently, and logic could be changed. While the two implementations could be functionally similar, these changes to the circuit could alter the SEE response in the circuit. As this area of research is especially dynamic at this time, it is likely to improve over the next decade.

B. Fault Simulation

Many fault simulation systems leverage already existing circuit modeling tools. In this sense, any design that can be simulated using a circuit modeling tool can be simulated for faults. In other fault simulation systems, circuit simulation tools, which are used for determining whether a circuit is operating as intended, can be modified to inject faults. Both of these methods are based on the following concepts:

— describing the design as a model or a model;
— leveraging a circuit simulation or modeling tool;
— developing testbenches and/or fault models to be used by the circuit simulation or modeling tool to inject faults into the DUT.

This method of fault injection can be quite powerful and flexible. In this case, both the design and the faults are abstracted, which makes it possible to perform simulation before the design is ready to be implemented in hardware. It is also possible to perform fault simulation of more exotic faults, such as single-event multiple transients (SEMTs) [82]–[84], that are difficult to reproduce in the accelerated radiation environment. While some of these fault simulators may not be as mainstream as other types of fault injection systems or traditional RHA methodologies, their very nature allows them to explore the vanguard of radiation effects. One of the challenges in the coming decade will focus on helping fault simulation tool designers validate their tools so that these tools can be used more widely within the radiation effects community. In this section, we will discuss circuit modeling tools that can be used as fault injection platforms, models of designs, and methods to perform a fault injection.

1) Options for Fault Simulation Systems: Currently, most fault injection tools are built on top of model checking tools, specifically designed analytical tools or circuit simulation tools. Prior to 1990, research on SEE predictions was conducted on static memory elements only [85], static analysis of logic elements, and FFs. Currently, there are many fault simulation techniques that have included the effects of clock-dependent (i.e., dynamic) SEE predications for logic elements and FFs [86], as well as various electrical, logical, and latch-window masking, that affect whether a fault propagates. Once a transient is captured in a memory element, the effects can be analyzed as a SEU. In this section, we will discuss different fault simulation tools that have been designed for radiation effects and fault tolerance. In this discussion, we will include both historical methods for fault simulation and current fault simulation tools.

a) Model-Checking-Based Tools: Commonly, model-checking tools represent the model as a graph so that graph transformations and operations can be applied to the model. Some of these models use graphs of any form, but some use specific graph representations, such as binary decision diagrams (BDD) or algebraic decision diagrams (ADDs) [87], [88].

In [89], Miskov-Zivanov and Marculescu discuss an analytical model using BDDs and ADDs for a unified symbolic analysis for circuit reliability. This technique accounts for logical, electrical, and latch-window masking. Follow-on work by Miskov-Zivanov and Marculescu takes into account the sequential elements [25]. Both of these methods are symbolic approaches for an efficient estimation of the SEE susceptibility. It was shown that the Markov chain approach could only provide steady-state behavior information, but the BDD/ADD could be done on both transient and steady-state effects. The second paper was verified to a Markov Chain and HSPICE™ circuit simulation.

The soft-error-rate-analysis (SERA) methodology [90] by Zhang *et al.* takes into account various approaches for circuit and fault simulation and analysis for probability and graph theory [90]. The authors assert they achieve an order of a magnitude speed-up over Monte Carlo-based simulation approaches.

In recent years, model-checking tools for formal verification have been used to identify areas of the design most vulnerable to faults [91]. In this work, a register transfer level (RTL) description of the design, a formal SEU model, and a formal specification are given to a formal verification tool. The formal specification provides assertions and properties that can be checked during the verification process. This work was used to verify a SpaceWire circuit, where it was found that only the registers storing state variables needed to be protected from SEUs [91].

b) Analytical-Based Tools: Soft Error Analysis Tool—Logic Analyzer tool was developed for a quick and accurate prediction of SER in combinational circuits with the ability to capture the three masking effects concurrently [92]. The methodology used for the development of this tool used logic cell characterization and flip-flop characterization. Their modeling of the voltage glitch propagation was done analytically by the use of mathematical equations assuming a triangular or trapezoidal pulse.

Rao *et al.* describe a method to characterize the vulnerability of an entire standard cell library by using parameters that represent the significant contributions to the SER [93]. Their algorithm prunes the number of transient waveforms to be considered within the circuit, which reduces the simulation time required for SEE analysis.

Sterpone *et al.* wrote a number of papers on analytical models for SEUs and SETs in recent years [46], [94], [95]. In [94], Sterpone *et al.* provides an analytical model of propagation-induced pulse broadening in SETs for a Flash-based FPGA. The work was based on results from electrically injected SETs and included models for the configuration memory. In [95], Sterpone *et al.* develops an analytical model for SEUs in the routing network of SRAM-based FPGAs. This work includes the Static Analyzer for REConfigurable Systems (STARECS) tool that provides static analysis of routing failures caused by SEUs. In [96], Sterpone *et al.* use laser testing to investigate how MBUs affect circuits. This testing was used to inform an analytical model that could determine if circuits had potential routing problems and how to fix the circuit.

Probably the most popular analytical model is based off the AVF and TVF reasoning described earlier in the paper. Mukherjee *et al.* originally developed the methodology that is well understood and accepted for calculating AVF and TVF, independently [30]. AVF and TVF have been extended by many other researchers. Walcott *et al.* [97] provided statistical analysis of AVF, indicating runtime variations in AVF. Li *et al.* [98] provided a method for estimating AVF without the use of circuit simulation and statistical analysis of the AVF of several key aspects of microprocessors. Sridharan and Kaeli [31] extend the AVF concepts to determine a program vulnerability factor that determines the reliability of software independent of specific microprocessor architectural vulnerabilities.

c) Circuit-Simulation-Based Tools: There are many options and examples for fault simulation using circuit-simulation tools. These tools can be very useful if the design either is a circuit or can be easily described as a circuit. Industry tools, such as Berkeley SPICE, Synopsys HSPICE, and Cadence Spectre, have been used to perform deterministic circuit simulation of DUTs to examine the effects of transients [99]. In mixed-mode simulation, device information is coupled to the circuit's representation to analyze the radiation-induced transient. The device simulation provides the physics-based representation of the transient [100]. As an alternative, the transient can be modeled using a double-exponential, time-dependent current pulse [101]. This current pulse can then be used as a source within the circuit-level simulation tool. In either case, circuit-level simulations have been limited to library cells and small circuits because of the simulation time required to characterize the DUT.

It is also possible that SEUs can be injected without instrumentation of the circuit description. The SEU Simulation Tool (SST) [102] injects faults into wires using a "force" command available in many simulation tools. With the use of the force command, circuit instrumentation is unnecessary, as the simulation tool can override the existing value.

SEU_Tool uses both analytical reasoning and circuit simulation tools to analyze the contribution of combinational logic in the path to a sequential or memory element [86]. This tool uses parameterized closed-form circuit models for transient pulse generation, a structural VHDL logic-level simulation for pulse attenuation and propagation, a probabilistic model for transient capture, and a second high-level VHDL logic simulation for bit-error observability. In addition to circuit modeling calculations, this method also contains algorithms at various steps to identify the worst-case contributors to soft errors, which reduces computation time. Given the detail of modeling capability in this method, its accuracy is largely a function of the quality and completeness of the parameters used for input [86], [103], [104].

Dhillon *et al.* presented tools for the analysis and optimization of soft-error tolerance of nanometer combinational circuits. The authors asserted the ability of these tools to calculate accurately the "unreliability" of circuits with less computational time than that of SPICE [105]. A multiscale simulation and fault injection approach includes all masking factors demonstrated in the paper.

iRoC Technologies have made a number of tools that can help designers estimate SER of cells and circuits. The TFIT tool can be used to help determine cell SER [106]. TFIT does SPICE simulations of cell libraries using information extracted from the Synopsys TCAD simulator [107]. The idea behind TFIT is that a standard cell library could be characterized in a manner that could be reused in a tool that could determine circuit SER [108], [109], such as SoCFIT [106]. SoCFIT maps RTL circuit descriptions to standard cell mappings and then calculates the SER through probabilistic reasoning.

One area of new research combines these types of fault simulation tools with Monte Carlo radiative energy deposition (MRED). This multiscale simulation approach provides the framework to consider all aspects that contribute to radiation-induced faults, including charge deposition, circuit simulation, and integrated circuit simulation. Integration with MRED enables improved accuracy in the energy deposition calculations and the resulting charge collection [12]. Coupling with SPICE enables the statistical distribution of pulse widths to be considered instead of the fixed pulse width used in previous methods. Finally, integrated circuit simulation provides the framework to study the contributions of individual cells based upon their usage within a synthesized design. We discuss this method of fault simulation in one of the case studies.

2) Preparing a Design for Fault Simulation: Circuit-modeling tools and circuit simulation tools generally need the design to be input in a standard representation that the tool can read. For model-checking tools, the design will likely need to be translated into a model, using the tool's model language. To use circuit simulation tools, it might be necessary to create an HDL or RTL description of the design, if one does not already exist. Most circuit simulation tools take circuit netlists as input. The easiest way to obtain a netlist of a design is to convert an HDL description of the design into a netlist or RTL using a synthesis tool, such as Design Compiler or Synplify from Synopsys.

The circuit description might also need to be instrumented to enable fault insertion. One possible fault injection method inverts the output of a node, such as with an SEU or SET, to mimic an inversion on the node's output. Within the gate-level netlist of the design, an additional structure can be added after the output node of each gate to inject the fault. This structure can model the different types of faults:

- stuck-high fault: OR gate;
- stuck-low fault: INV-AND gate;
- SEU or SET: XOR gate.

An additional input line connected to the structure can be used to inject a fault by asserting a high signal on the input signal. In this way, the extra input signal on the structure overrides the output from the node and effectively inserts a faulty signal into the circuit for simulation. The DUT's netlist can often be modified automatically via scripting to avoid hand-coding mistakes.

3) Circuit Testbench: For circuit simulation tools, it is also necessary to define a *testbench*. The basis for stimulating and

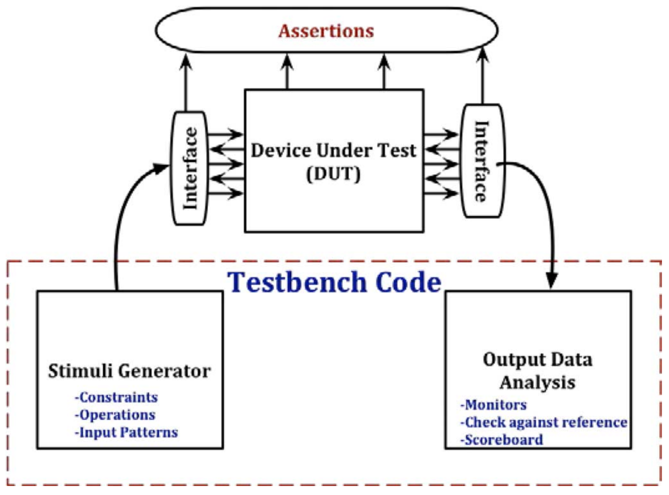

Fig. 12. Basic block diagram for a testbench with a DUT.

verifying the design is a testbench. The term testbench usually refers to code or a circuit description that has predetermined input and output vectors (also known as sequences, patterns or stimuli). The input sequences are applied to the design, and the outputs are observed to determine whether the predetermined output occurs (Fig. 12).

The testbench is a closed system that effectively models the entire design [110]. Modern verification techniques, such as directed and constrained-random verification, coverage-driven verification, and assertion-based verification [111], make use of HDLs, such as VHDL, Verilog, or SystemVerilog [30]. These languages enable development of more efficient functional verification environments and facilitate the reuse of testbench components.

It should be noted that the test vectors can be defined in the testbench a number of ways. One method is to statically define input and output vectors in the testbench. While there is simplicity in statically defining the test vectors in the testbench so that the same test vectors are run every single time the testbench is executed, using the same test vectors is not recommended, unless it is a sample of realistic data. Another method is to input and output test vectors from a file. Both methods provide avenues for exploring the design using a known set of test vectors so that corner cases can be explored. Finally, it is possible to generate test vectors dynamically in the testbench. This method will allow designers the flexibility of sampling randomly from the input vector set without needing multiple testbenches or test vector files, although methods for dynamically detecting an error will be necessary.

All three of these methods have their time and use. Early in the process, when the design might still be changing, having a good set of test vectors that can explore whether the circuit is operating as intended can help designers distinguish between problems with the design and problems with fault tolerance. Later in the process, when the design is known to operate as intended under most scenarios, dynamically generating random test vectors can be useful for exploring input sensitivity.

4) Performing a Measurement: One basic test approach for performing a measurement combines a modified circuit description, a testbench and an IC simulation tool. If the modified circuit description has XOR gates on the output of the gate, then faults can be inserted using a rail-to-rail voltage pulse in the testbench. The IC modeling tool provides full circuit simulation that enables the demonstration of the fault's effect on an operational design. For more advanced approaches, the ability to use charge generation tools, circuit-level simulation tools, and an IC circuit modeling tool provides the means to use fault injection for a multiscale simulation for both static and dynamic RHA testing. A flowchart for this process is shown in Fig. 13.

Both of these test approaches depend on a testbench for functional verification of the design, as shown in Fig. 12. Functional verification is done by comparing simulated design responses from incoming data (stimuli) against expected values. The testbench may include a behavioral model of the design and test vectors. The testbench includes additional constructs, such as stimuli generation, output analysis, or reporting. The DUT responses (or actions) to stimuli are compared to the expected results to validate the behavior. Report statements can be used within the verification process to update the user with the current status of the testbench, as well as the states of the I/O. The report framework can help the users of fault injection tools by logging fault injection information into the simulation log files in a manner that allows these messages to easily separated from the simulation log [26].

The flowchart in Fig. 13 shows how to perform fault simulation with a circuit simulation tool. In this flow, the testbench chooses one of the instrumentation structures for fault injection either at random or systematically and injects the fault through the instrumentation circuitry. The testbench then runs all of the input vectors for the design and monitors the outputs for errors. If an error occurs, then the testbench stops the simulation process and logs information about the injected fault and error state.

V. Validating Fault Injection Results

Validation is helpful for determining whether a tool provides a proper analog for faults that occur in traditional RHA testing. In this manner, the accuracy of the tools can be determined, so that tools with low accuracy can be improved. The validation process by necessity needs two sets of data—one collected from fault injection and one collected by either accelerated radiation or pulsed laser tests. To this end, the use of historical data from previous accelerated radiation or pulse laser tests to validate the fault injection tool can be expedient.

If the data set from a traditional RHA test is taken after the fault injection testing, often times the designers are better prepared. To this end, validation testing should be less exploratory than the initial static accelerated radiation testing of the design. The testers should know:

1) the types of faults that occur in the hardware;
2) how those faults occur in the design;
3) how often these faults occur;
4) how the design operates when in a failure state.

Ideally, validation testing should confirm results already seen in fault injection. Looking back at our adder circuit from Fig. 3, fault injection should tell us that SEUs in flip-flops cause errors on output D. If this adder circuit was tested using an accelerated radiation source, then the test should confirm these results. If the validation testing does not match the fault injection experiments, then the testers must determine what went wrong in

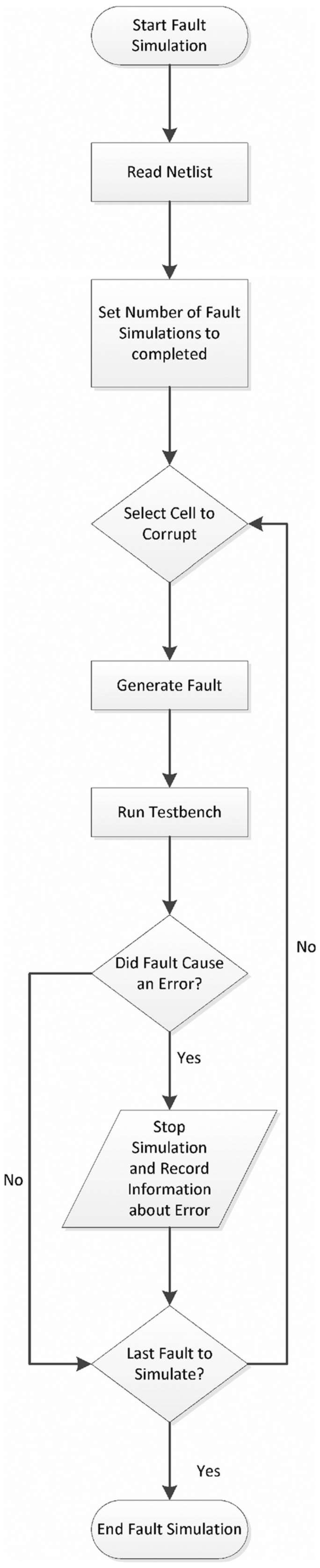

Fig. 13. Flowchart for fault simulation using a circuit simulation tool.

either the fault injection methodology or the accelerated radiation test.

Currently, the validation process is being handled a number of different ways. Some fault injection tool designers will compare error rates, such as 50% of faults cause errors in both fault injection and accelerated radiation testing. Some papers will map out fault locations and will confirm the fault locations with accelerated radiation testing [47]. Because the radiation effects community is still grappling with issues with accuracy and validation, there is no set way to do validation at this stage.

In the remainder of the section, we will discuss traditional methods for RHA and how to compare these results to fault injection results.

A. *Experimental Techniques for RHA*

Several experimental techniques exist for performing accelerated RHA testing but only those relevant to single events are considered here. There are many good resources and introductions to radiation testing using accelerated sources [112], [113]. This part of the paper focuses only on validating fault injection results with accelerated radiation results. This section also includes information on pulsed laser test method, which is more fully explained in a companion article and discussed here only briefly [114]. Fault injection for single-event hardness assurance can be accomplished either with broad ion beams that inject faults randomly in space and time or with focused ion and laser beams that inject faults at known locations and with a known relationship between the injection time and the design clock. Broad ion beams are essential for predicting error rates in space based on the measured SEE cross-section data obtained as a function of ion linear effective transfer (LET). Whereas focused laser-beam and ion-beam data cannot be used to predict error rates in deployed environments, they are well suited for SEU or SET fault injection.

1) Focused Ion Beam Methods: There are two methods used to obtain ion beams with small lateral dimensions that can inject faults at localized spots in the integrated circuits (IC) [115], [116]. One uses a pinhole aperture (having a diameter of a few microns inserted in the beam ahead of the design) that is moved to scan an area larger than the size of the pinhole. The other is to use magnetically focused beams that are electrostatically raster-scanned across the device. In both techniques, if 1) the position of the beam is recorded at which the result of an injected fault is observed and 2) the position is correlated to the circuit design via the layout mask, then the precise location and cause of the injected fault may be determined.

The raster-scanning technique has a requirement on ion energy due to "magnetic rigidity" of the focusing magnets. As higher energy ions require stronger magnetic fields, excessive heating associated with strong magnets limits the energy of the ions that can be focused. As a consequence, the ions have relatively short ranges in silicon which calls into question the results obtained for devices where charge collection occurs via diffusion.

Numerous fault injection studies have been performed using focused ion beams to provide valuable information on the effects of fault injection [114], [117], [118]. The facility most often used for these studies is at Sandia National Laboratories, Albuquerque, NM, USA, but other facilities, including Gesellschaft fuer Schwerionen Forschung, Darmstadt, Germany, have also reported on fault injection experiments using focused ion beams.

2) Pulsed Laser Methods: The basic mechanisms responsible for SEEs in ICs are the generation and collection of charge at sensitive nodes in the ICs. Both ions and laser light generate charge when they propagate through the semiconducting material that forms the substrate of the IC. Although both methods might generate charge via different interactions—light through absorption and ions via the Coulomb interaction—both methods create electron excitation from the valence to the conduction band of the semiconductor. The requirement on photon energy is that it is greater than the bandgap energy of the semiconductor so that it will be absorbed, creating electron–hole pairs in the process. Ion-generated charge track structures depend on atomic number and energy and are different from those generated by laser light that depend on wavelength, intensity and focusing optics. These differences turn out to be of little consequence in most fault-injection experiments.

To perform fault injection, the IC being investigated is mounted on an x–y–z stage that moves in steps of 0.1 μm so as to allow the laser beam to inject errors at any location in the design. Ideally, this method should be computer-controlled so that the coordinates of the stage are correlated with the result of an injected fault. A powerful microscope objective should be used (e.g., 100X) so as to obtain the smallest spot size for the highest resolution. The beam should be well characterized with a Gaussian radial profile, and the pulse duration should be shorter than the response time of the IC being tested. The energy of each pulse should be recorded by using a beam-splitter to direct a fraction of the beam to a calibrated photodetector. Equipment for obtaining an image of the surface of the device together with the laser spot should be included in the system so that the exact location of the injected fault is known in real time and can be used as well during post processing. The system should offer the option to inject multiple errors at each point to allow for averaging the response, especially in the case of single event transients, by waiting a certain amount of time before moving on to the next point.

Light pulses with widths shorter than the response time of the circuit, i.e., the time it takes the voltage on the node to change by 50% in response to the deposited charge. A microscope objective lens is used to focus the light to a spot with a diameter on the order of a micron or less, depending on the light's wavelength and the magnification of the lens. Threshold SEE energies may be determined by gradually increasing the light intensity until an SEE is recorded. From the results of an automated x–y scan, contour plots of SEE sensitivity can be constructed from which the required spatial information can be obtained with submicron resolution.

For those cases where the surface of the component is covered with multiple metal layers or is flip-chip bonded in a package, the light is blocked, preventing such testing from occurring. An alternate approach is to direct the light from the back side after having removed the back of the package and, if required, thinned the semiconductor substrate. Two approaches have been used successfully for back-side testing, including single-photon absorption requiring photon energy just above that of the bandgap of the semiconductor [119], and two-photon absorption, a nonlinear process requiring high photon intensities with the photon energy less than the bandgap of the semiconductor [120].

Fault injection with pulsed lasers has, over the years, provided a wealth of information [121]. There have been a number of fault injection studies performed using lasers [122]–[124]. Problems associated with packaging and multiple metal layers on the surface have been addressed and are largely solved. The next challenge involves fault injection in highly-scaled components where the transistor gate length of less than 100 nm is significantly smaller than the size of the focused laser beam. Adopting UV lasers with their shorter wavelength and, therefore, higher resolution, should help in this respect.

B. Validation Process

After the accelerated radiation or pulsed laser test is completed, the results need to be correlated to the fault injection data set. Because the faults in accelerator testing do not present themselves in the system uniformly or at specified time intervals, correlating errors to specific fault locations from fault injection can be a challenge. In some cases, the error follows the fault by several seconds, and other times the monitoring software reports the existence of the error before the fault location. On other occasions, some errors need to be dropped from the data set, such as when multiple independent upsets are the source of the error. Often, all of the results around a SEFI event must need to be ignored. This is especially true if removing the SEFI is time consuming, as the system might report errors for several seconds until the hardware recovers.

As errors can be design dependent, differences between the design used in fault injection and the design used in accelerated testing can make validation challenging. For fault injection tools that instrument the design or use proxy hardware, it can be more difficult to compare results from the fault injection tool and accelerated testing. If the same design is used in both cases, the results from fault injection can be used to disambiguate the accelerator test results. Due to the problems described with attributing faults to errors, the most effective approach for analyzing accelerator results is to look at several fault locations before and after the error in the log. This "window" of fault locations can then be compared to fault injection results to determine if any of these fault locations caused an error in fault injection.

As a general rule, comparing fault locations as a validation process is easiest with FPGAs, as getting fault locations is straightforward. In other cases, it might be necessary to compare the data based on percentages or cross sections. For example, the fault injection tool will know both the number of faults injected and the number of errors induced. These values can be used to predict the percentage of faults that cause errors. Similarly, if the per-bit or per-flip-flop cross section is known, it might be possible to predict a error cross-section. It might be possible to extract these same values from the RHA test results so that the values can be compared.

While this method can usually help a designer correlate an error with fault injection results, some errors cannot be completely correlated. In some cases, the errors are due to faults in intermediate data values in flip-flops, and not due to a known fault location. In other cases, the accumulation of faults in the circuit state caused the error, which can be minimized by decreasing the flux. For these cases, sometimes part of an accelerator test can be replayed using the fault injection system, where the tool uses the accelerator log to inject faults into specific locations in a particular order. Through this attribution process, the designer can determine whether the error can be explained and whether further design exploration is needed to address potential design flaws.

If the error rate is much higher than indicated by the fault injection tool, then the flux could be too high, there might be problems with the fault injection tool or the accelerator test fixture. It is always possible that there is a systematic error or a design error that causes results to be skewed in either the fault injection system or the accelerator test fixture. In cases where the fault emulation and the accelerated testing share similar test fixtures, the chance of not detecting either a systematic error or a design error could be a serious problem.

VI. Case Studies

We provide three case studies showing how fault emulation and simulation are being used as of 2013. One case study is the use of fault emulation and simulation to verify FPGA user circuits. The next case study is the use of SPICE and MRED for fault simulation on a circuit. Finally, the last case is the use of fault simulation on CPUs.

A. Fault Emulation and Simulation of FPGA User Circuits

Fault injection on Xilinx's SRAM-based FPGAs is often quite useful and straightforward. These FPGAs experience these types of radiation-induced faults:

- SEUs in the programmable logic and routing (configuration memory);
- SEUs in the user memory (both BlockRAM and user flip-flops);
- SETs in embedded logic (multipliers, processors) of the more modern components;
- SEFIs.

In the older components generally only the configuration memory was used in high-reliability applications, as scrubbing the BlockRAM was challenging. This limited the faults to SEUs in the configurable memory, SEUs in user flip-flops and SEFIs. In this era, over 99% of the fault model could be covered by injecting SEUs into the configuration memory. In the newer components, using the BlockRAM has become more reasonable with built-in BlockRAM scrubbing. Furthermore, SETs in the embedded logic are observable. For these components, covering the SEUs in the configuration memory still covers the majority of faults, but no longer covers most of the faults.

Most SRAM-based FPGA user circuits require the use of SEU mitigation techniques, such as triple-modular redundancy (TMR), to meet mission requirements. User circuits that need TMR mitigation should be tested with and without TMR to verify that TMR mitigation has been applied properly and reduces the number of errors from the unmitigated design. TMR mitigation is often not "perfect" and could be susceptible to placement-related issues and shared logic. In a design that has only had TMR partially applied to the design (*Partially TMR-Protected*), SEUs can affect the portions of the design that is unmitigated. The mitigated portions could also have the same issues as fully TMR-protected designs. While the number of fault locations stemming from unmitigated logic is immutable, the number of fault locations stemming from unmitigated routing can vary based on how the tools layout the user circuit. In the remainder of the section, we present the use of fault injection tools to find TMR-related design problems. For a longer treatment of this topic, see [125] and [126]

1) STARC: A Fault Simulation Tool: Because FPGA user circuits are already described in an HDL, fault simulation tools only need to add testbenches to analyze the circuit, enabling the detection and correction of design flaws in the design phase where fixing flaws is significantly less expensive. LANL designed a tool called the Scalable Tool for the Analysis of Reliable Circuits (STARC) tool to specifically address limitations for traditional circuit reliability tools in model creation, input data sets and computational complexity with these solutions [125]. The input of the tool is a netlist in an industry-standard format called Electronic Design Interchange Format (EDIF). The netlist is a representation of the design description. The tool does combinatorial reliability calculations without input vectors. Without input data sets, the reliability of subcircuits can be determined by type, such as a two-bit adder, and memoized for reuse[2] [127]. Without input vector sets, logic masking cannot be taken into account and STARC estimates the worst case failure rate.

STARC was designed to help designers find problems in the application of TMR, especially portions of the design that might not be properly triplicated. There are cases where the design flow tools, especially synthesis tools, alter the circuit so that the TMR modules are no longer functionally equivalent or independent. In these cases modules might share a subcircuit, which becomes a single point of failure. Feedback loops in TMR-protected systems are also sensitive to *persistent errors* [51], and need to use triplication and voters to break the feedback loops. If the feedback loops are not handled in this manner, the feedback loop's state might not be able to autonomously resynchronize after the SEU is removed. In this scenario, while the first SEU in the feedback loop is masked, a second SEU in the feedback loop is not guaranteed to be masked.

In all of these cases, STARC provides warnings and information about the design to the designer. The output of the tool provides the designer a list of subcircuits that are untriplicated, and warnings about potential single points of failures from functionally nonequivalent modules and logical constants, which create single points of failure [128]. Because EDIF is tightly coupled to the circuit design, the designer should be able to directly use STARC's output to find and fix the design flaws in the user circuit.

2) The SEU Emulator: A Fault Emulation Tool: Once the design is completed and hardware is available, it is possible to move on to fault emulation. Unlike simulation tools, fault emulation allows placement-related issues to be assessed. Fault emulation is possible, because the interfaces that control configuration memory are accessible to the designer. These interfaces can be used by the designer to purposefully corrupt the programming data to mimic SEUs in programming data. Using the fault injection algorithm shown in Fig. 10, faults can be injected systematically throughout the entire FPGA. Resetting and resynchronizing the user circuit after the SEU is removed is done by resetting the user circuit.

[2]Memoization is a combinatorial optimization method from computer science that breaks problems into subproblems. The technique solves the subproblems and then substitutes these solutions in, where possible, into the problem.

The LANL SEU Emulator tool uses two FPGAs with each one hosting the same user design and input vectors. The advantage of this system is that sharing input vectors, detecting output errors, and testing the system for resynchronization is very easy. The disadvantage is the complexity of designing the lockstep system.

Once fault emulation is completed, the SEU locations need to be correlated to the design. While the fault emulation tool returns the locations of sensitive bits, most designers do not know how to translate that location into a physical location on the device. If the logical location is determined, it is possible to use a Xilinx design flow tool, called FPGA Editor, to determine what part of the user circuit is in that location.

LANL has performed validation of on their fault emulation systems several times [9], [35], [47], [51], [53], [125], [129]. Because SEUs in FPGAs are straightforward and most fault locations are reachable through the reconfiguration ports, usually the tools are accurate to 80%–99% of radiation-induced faults with fault emulation of TMR-protected circuits as less accurate as unmitigated circuits. LANL used the "windowing" method described in Section V to correlate accelerator results. LANL also used in-system fault emulation on their Virtex-4 payload before launch and currently has found the analysis provided a worst-case analysis of the error rate on orbit [130]. This testing indicated that the design was immune to 99% of all faults. On-orbit results indicate that the design is currently 99.9% immune to all SEUs.

3) Validation of the LANL Emulation and Simulation Systems: In this section, we compare the use of these methodologies on a circuit. The circuit, an adder tree, is fully triplicated and was designed originally to test for placement-related issues due to both MBUs and logical constants. This design was implemented using a Xilinx Virtex-II FPGA (XC2V1000). All three methodologies (emulation, simulation, and accelerated radiation testing) were used on this design. In the following paragraphs, we describe the amount of time and the quality of the results from fault injection.

In terms of time, STARC is comparatively much faster than the other two methods. Within a minute, the tool returned the result that the design was triplicated properly and with warnings that placement-related issues could exist from logical constants. As STARC cannot currently estimate the placement-related issues, it is unable to estimate how many bits in the design could cause output errors.

In terms of test coverage, the SEU Emulator was much more complete than the other two methods. Fault injection found 285 single-bit SEU locations, 18 733 2-bit MBU locations, 11 264 3-bit MBU locations, and 19 464 4-bit MBU locations that caused errors in the design. Each pass through the fault injection test takes two hours per run and each MBU test is a separate test. The MBU tests were run with specific MBU shapes based on LANL's analysis of how MBUs affect the Virtex-II, which constrained the MBU tests to the six most common shapes. In all, each set of fault injection tests took 14 hours for the seven tests.

As validation for both of these tests, LANL completed a two hour long test at the Indiana University Cyclotron Facility's proton accelerator. During this test, they were able to observe 50 errors, of which 21 were attributed to SEFIs, 13 were attributed to multiple-independent upsets (MIUs), and 16 were attributed to single-bit placement-related issues and MBUs. Of the 16 errors, 88% were correlated to known fault injection error locations. Completely validating the fault-injection results was not possible giving time and financial constraints. At three algorithm iterations a second, it would have taken 16 days to test the single-bit errors, assuming that no single-bit fault location was exercised multiple times. As is, LANL was able to test at a higher flux to be able to shorten the time frame of the test considerably. Because the MBUs for the Virtex-II have only a 2% chance of occurring in proton radiation, as determined in earlier testing [1], completing the 2-bit test would have taken over two years.

B. Multi-Scale Simulation of ASIC Circuits With MRED to IC Modeling Fault Injection Simulation (MRED2LOGIC)

This case study shows how to do fault simulation of SET-induced soft errors observed at the complex digital IC output. It has been proposed that SET error rates will be the dominant contributor to system error rates for highly scaled technologies [6], [131], [132]. In order to predict SET error rates, a model is needed that generates and propagates SETs from particles in a radiation environment. This work requires coupling of three tools:

1) radiation transport and charge collection estimates;
2) circuit-level fault simulation;
3) high-level IC fault injection simulation.

In this case, MRED was used for charge generation modeling, SPICE was used for the circuit-level simulation and ModelSim was used for timing and logic simulation.

The interaction of these three tools is shown in Fig. 14. The first tool is MRED, which generates charge generation events. The SPICE process determines if the charge-generation event was large enough to generate a sufficient current pulse and converts candidate pulses into inputs that the IC modeling tool can use. Once the signals are in the form that ModelSim uses, the tool analyzes SET propagation through the combinational logic to memory elements to identify if errors occur at the outputs of the IC.

This tool is also able to minimize the workload of the simulation tool by avoiding the simulation of SETs that are not long enough or do not have enough charge to affect the design. There are two decision points in Fig. 14 that show where SETs are removed from the simulation process. In the first one, any charge generation event that is assumed to be negligible is removed from the data array of SETs that are input into the SPICE process. At the second decision point, any SET pulse that does not have sufficient duration to be propagated is removed from the input data array for ModelSim. We provide a brief overview of how we are using MRED, but more information can be found in these: [133]–[137]. MRED can be used to estimate the collected charge,Q_{coll}, from an ionizing radiation event by defining these inputs for simulation to MRED:

1) the ionizing particle's energy, species, and trajectory;
2) the sensitive volume size, locations and charge collection efficiency;
3) the strike location of the ion within the specified combinational cell;
4) the number of events to execute.

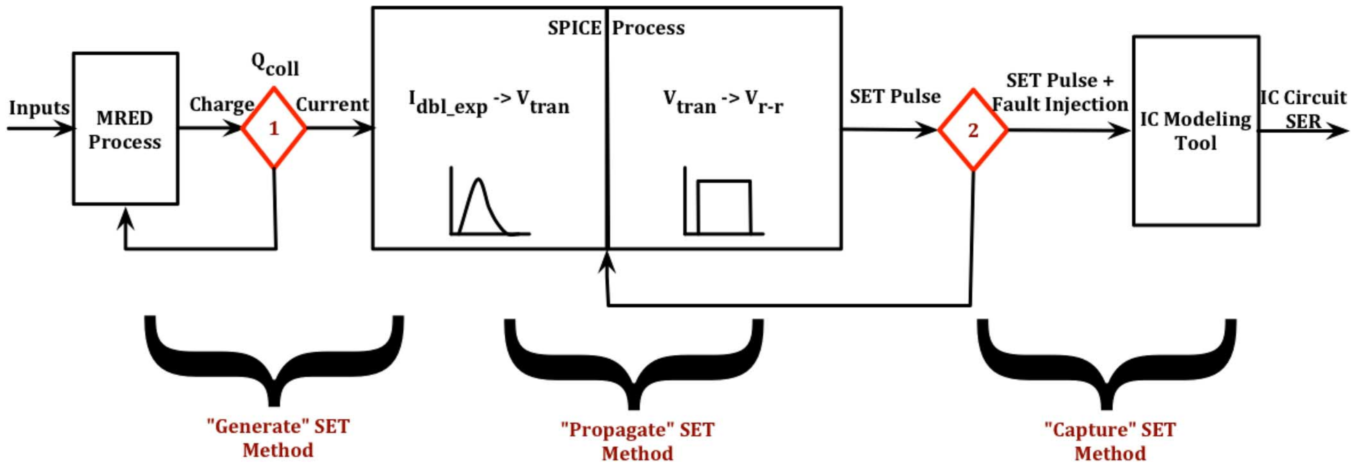

Fig. 14. Multiscale simulation approach black box block diagram.

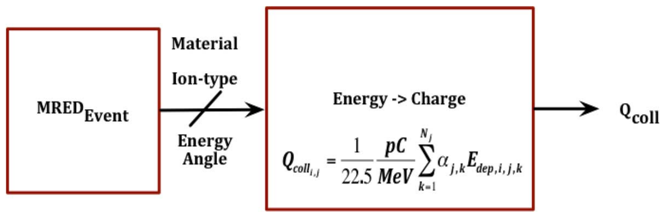
MRED_Event
Material
Ion-type
Energy
Angle
Energy -> Charge
$Q_{coll_{i,j}} = \frac{1}{22.5}\frac{pC}{MeV}\sum_{k=1}^{N_j}\alpha_{j,k}E_{dep,i,j,k}$
Q_coll

Fig. 15. MRED process basic block diagram.

During an MRED simulation run, the beam is randomized over strike location. The output of the run is the collected charge (defined by the energy deposited in the volume and its charge collection efficiency) for each volume for each event. A basic block diagram of this process is seen in Fig. 15, where Q_{coll} is the charge collected, i is the MRED event number, j is the circuit node number, k is the individual sensitive volume, α is the sensitive volume efficiency, and E_{dep} is the energy deposited in the individual sensitive volume.

The SPICE analysis converts the MRED simulated Q_{coll} to a voltage transient (V_{tran}) then to a rail-to-rail voltage $(V_{\mathrm{r-r}})$ so that it can be used as an input to ModelSim. ModelSim expects all input signals to be logic 1 or logic 0. ModelSim simulates the execution of an operational circuit via a testbench as previously described and determines if the generated SET results in a fault to the circuit and eventually a soft error in the DUT. In this case, soft errors are the manifestation of the faults, as described in Section II. If the fault makes it through the first stages of simulation of the combinational elements and migrates through to a sequential or memory element, then the resulting error can be verified by monitoring the output through the testbench.

The inputs for the DUT are the SET pulses generated for the characterized cells. These SET pulses form a distribution of pulse widths to be used in conjunction with a fault injection library [26]. This procedure allows for both randomization of an SET pulse-width and selection of the corresponding combinational library cell to strike. The DUT circuit is monitored through a comparator in the testbench for a soft error, such that the probability of each generated SET is maintained from MRED. The circuit is verified at operational circuit clock speed so that the resulting error contributions from individual cells take into account the dynamic operation of the circuit. This process produces a soft error analysis for each cell that does not differentiate between logical-masked or timing-masked errors but includes them in the analysis for a full IC, mimicking a true experiment. This process is the first method to demonstrate soft error contributions from individual cells with direct traceability to particle strikes from the specified environment.

1) Validation: Validating this tool is a multistage effort and to date is partially complete. To validate such a complex suite of tools, it is necessary to determine whether the generate, propagate and capture methodologies properly mimic how the process works in the radiation environment. On top of it, end-to-end validation is also necessary. In the rest of this section, we discuss the validation that has been completed and will be completed.

The first step of validation was to prove that the generate and propagate methodologies accurately mimic how SETs are generated. The circuits simulated in SPICE duplicated those circuits tested in [138]. The simulation process selected at random one of the combinational cells and the specific ion strike location within the combinational cell to receive an event. The simulation was checked to determine if the latch had changed state and recorded any errors. After simulation, the results were compared to the data in [139]. Fig. 16 contains the NAND2x1 SET heavy-ion cross section results predicted by the model and the experimental data [139]. This figure shows that the predictions are within a factor of two of the experimental data. Further validation of INVx1 and NOR2x1 from the same library showed similar results.

The results from this part of the validation also demonstrated that the simulated cross-sections match the experimental dependence on pulsewidth and LET. The three 90-nm combinational cells were analyzed for SETs at an LET of 2.1 MeV-cm^2/mg for three logic cells using the same radiation environment as [139]. This test determined the SET pulsewidth distributions once the cell characterization had been verified. An example of a resulting distribution is shown for NAND2x1 in Fig. 17.

The next portion of the validation looked at whether the capture methodology accurately predicted how SETs were captured. This part of the validation process used a library [140] that was developed at the Vanderbilt University Institute for Space and Defense Electronics. This library was used with the SET pulsewidth distributions for the INVx1, NAND2x1,

TABLE I
SIMULATION RESULTS FOR PERCENT SUSCEPTIBLE TO SETs FOR LOGICAL-MASKING

Method	INVx1 (cell count = 58) Susceptibility	NAND2x1 (cell count =114) Susceptibility	NOR2x1 (cell count = 51) Susceptibility	Simulation Mutations
Black et al. [142]	9 %	14.3%	8.3%	Cell Count x 131075
MRED2LOGIC	10%	14.3%	8.5%	3000 cell

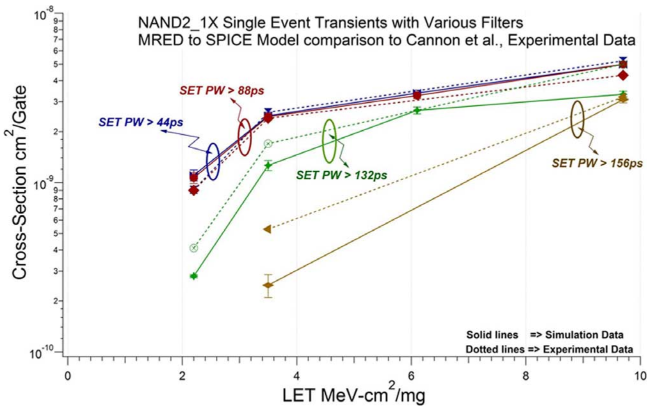

Fig. 16. SET cross-section of 1X drive strength for 2-input NAND gate as a function of LET. MRED to SPICE predictions are shown with solid lines, and experimental data are shown with dashed lines.

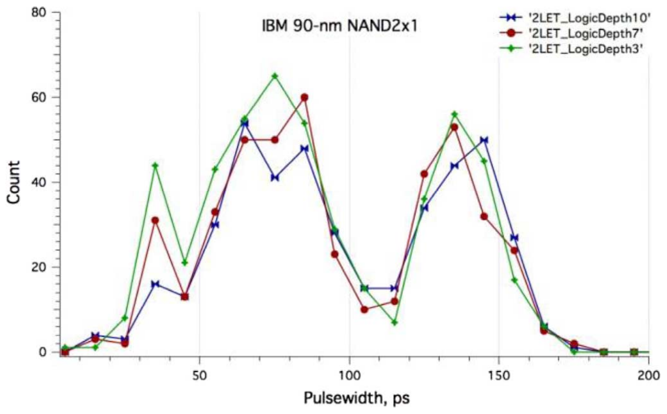

Fig. 17. Distribution of SET pulsewidths for 90-nm NAND2x1 for three different logic depths and particles of 2.1 MeV-cm^2/mg.

and NOR2x1 from [139] to enable SER prediction for those components within an ALU [26].

A random SET is selected based upon the distribution from [139]. Next, the ALU's netlist is parsed to identify the usage of each library cell (e.g., INVx1, NAND2x1, and NOR2x1). The testbench chooses a random cell in the ALU for fault injection and strikes it with a random SET pulsewidth within the distribution associated with the cell. Then, the testbench runs all test vectors for all functions specified by the ALU. In this case, the ALU has a datapath width of 8 bits and uses three control bits; the total number of test vectors is 131 075. The testbench monitors the outputs for errors. If an error occurs, the testbench stops the simulation and logs information about the fault and the error.

Multiple ModelSim simulations were executed for the ALU. The logical-masking error rate was examined by using errors that lasted the full clock width. This method enabled comparison to previous work [141], [142] to verify the accuracy of the fault simulation. The previous method used by Black *et al.* [142] used multiple tools, including Cosmic Ray Effects on Micro-Electronics (CRÈME96) [143], Technology-Computer Aided Design (TCAD), SPICE, and ModelSim. The MRED2LOGIC results for the multiscale simulation for the IBM 90-nm INVx1, NAND2x1, and NOR2x1 agreed with the simulation results from Black *et al.* [142] and Atkinson *et al.* [141]. Table I reports the results of comparing these two sets of results. We compare these results in terms of the method took nearly 30 million mutations in ModelSim. The MRED2LOGIC simulations took three days to complete, which is a bare fraction of the three months of simulation and computing time reported by Black *et al.* amount of time taken for fault injection and the quality of the results. The multiscale simulation required 12 000 total mutations to simulate the entire cell library, whereas the Black *et al.* method took nearly 30 million mutations in ModelSim. The MRED2LOGIC simulations took three days to complete, which is a bare fraction of the three months of simulation and computing time reported by Black *et al.*

While these results are a good beginning to the validation process, there is still work to be done. The most obvious missing piece is the lack of additional radiation validation. The author of this tool is currently working on finishing the validation of the tool, but so far the results are promising.

C. Fault Simulation for CPUs

In this final case study, we will discuss fault simulation for microprocessors. For CPUs, the physical manifestation of SEUs and SETs must occur in active computational structures to affect higher abstraction levels, such as the operating system or application. Once a soft error is present, the impact on the software is dependent upon the architectural vulnerability factor (AVF) as determined by the application executing on the IC [30]. AVF is defined as the probability that a fault in that particular structure results in an error. For reliability, designers are concerned with silent data corruption (SDC) as well as detected unrecoverable errors (DUE) [144]. A classification scheme for SDC and DUE has been developed to analyze soft errors at the architectural level, as shown in Fig. 18. Outcomes labeled 1, 2, and 3 indicate non-error conditions. Outcome 4 indicates SDC, where a fault induces the system to generate erroneous outputs. Outcomes labeled 5 and 6 indicate DUE. A soft error can occur in unused data structures or within structures such as a branch predictor that only impact performance and not the correctness of the application [24]. Simple error detection (e.g., parity) does not reduce the overall error rate, but does provide fail-stop behavior and avoids any data corruption. Detected errors are potentially less severe to the operation of a microprocessor than undetected errors because they can be flagged and mitigated with architectural techniques, such as redundant multithreading [145] or other forms of redundancy [146]. There may be some loss in performance to recover but correctness can be maintained. For example, a memory structure can be monitored with parity and

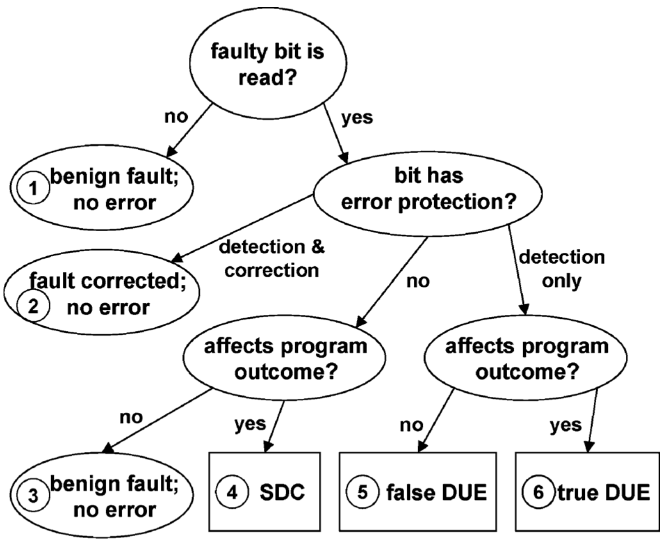

Fig. 18. Classification of the possible outcomes of an SEU in a microprocessor [144].

send a signal to the application when a parity mismatch occurs. Undetected errors do not provide any information about the location or time of the error, leaving the system completely vulnerable.

Previous studies on single processor cores have reported that a majority of the soft errors are masked at the architectural level [147]. For example, one study stated that less than 15% of single-bit corruptions in processor state were actually visible as software errors [148]. However, in nanometer technologies, charge can be collected (i.e., shared) by multiple adjacent nodes for a single incident particle [149], [150]. These adjacent nodes could lie on separate logic paths that could impact the assumption of a single-bit corruption and defeat redundancy techniques that rely on majority voting.

Fault injection has been used to characterize the vulnerability within a computing system [151]–[153]. However, the complexity of covering the test space becomes nearly impossible due to the added dimension of the clock cycles of the program (Fig. 7). The computational time can be reduced by simulating a subset of fault stimuli chosen at random locations throughout the circuit, which is known as statistical fault injection (SFI) [32]. The major drawback of SFI is that the decrease in computational time is directly proportional to a decrease in accuracy. AVF analysis was developed as an alternative to SFI [154], [155]. Using AVF to determine the mean-time-to-failure for an entire processor is valid under most cases [156]. One of the most difficult steps is to compare the fault injection to experimental results since it requires dynamic testing [157]. However, the dynamic testing must include applications that represent the expected workload of the mission. A benchmark suite of applications can be used to represent the workload; however, the component program must be selected carefully to ensure that the dynamic behavior matches the expected workload. Even within the individual programs, there can be dynamic phases of the program that exhibit significantly different behavior, making it difficult to evaluate the SER [158]. Fault injection must consider the different phases of the program to be sure that the test space is covered. Several approaches use simulated faults across benchmark programs to determine the vulnerability of a microprocessor [159]–[162]. The following case study illustrates fault injection for a software benchmark.

1) Fault Injection for Vulnerability of Control Bit Errors: Soft errors can alter the correct execution of code within a microprocessor, particularly if the control logic described below is compromised. Each instruction's opcode/operation is considered control logic, although there are some instructions that have extra control logic. Therefore, every single instruction that is executed has the potential for a control logic error, as instructions can be altered while stored in the microprocessor's cache or in DRAM. While error correcting codes can be a help, changes to instructions have been witnessed in both beam testing [164] and deployed systems [130]. When a fault occurs in the control logic, there is a risk that an incorrect instruction is executed. This can cause incorrect data to be stored into memory or complete failure of the application currently executing on the processor [165].

The observed errors from the fault injection can be categorized as follows [166]:

1) *operation errors*—a change in the operation code used;
2) *operand errors*—a change in or premature use of the register/operand addressed;
3) *execution errors*—a change in the functional units used;
4) *timing errors*—the instruction beginning or ending at an incorrect time;
5) *order errors*—a commitment order violation.

SFI determined that timing errors were the dominant group of control-flow errors.

This case study describes the impact of control-bit errors on the output of a microprocessor. Fault injection was performed by injecting a single-bit fault within the instruction opcode for a VHDL representation of a MIPS R2000 processor [167]. A software-based fault injection method was used to inject one fault at a time into the control bits of every possible instruction during each program cycle. The process was accelerated by using a FPGA for simulation of the Dhrystone software benchmark. The Dhrystone benchmark is composed of eight main procedures and three main functions, with each containing a different frequency of instructions.

The MIPS R2000 processor completes an instruction in five stages: Fetch, Decode, Execute, Memory, and Write-Back [167]. It has a 32-bit word length, and instructions have three formats (Fig. 19):

1) register-type (R-type);
2) immediate-type (I-type);
3) jump-type (J-type).

R-type instructions are typically for instructions that require three operands. I-type instructions are for instructions like load, store, or branch that use immediate constants. J-type instructions are for jump instructions that significantly alter the program counter. The control bits of the 32-bit instruction word are the opcode (instruction word bits 31–26) and the function code (instruction word bits 5-0). However, each instruction does not require all of the control bits. For R-type instructions, the opcode field (bits 31 to 26) and the funct field (bits 5 to 0) are required for correct control flow. For I-format and J-format instructions, only the opcode (bits 31 to 26) are required for correct control flow.

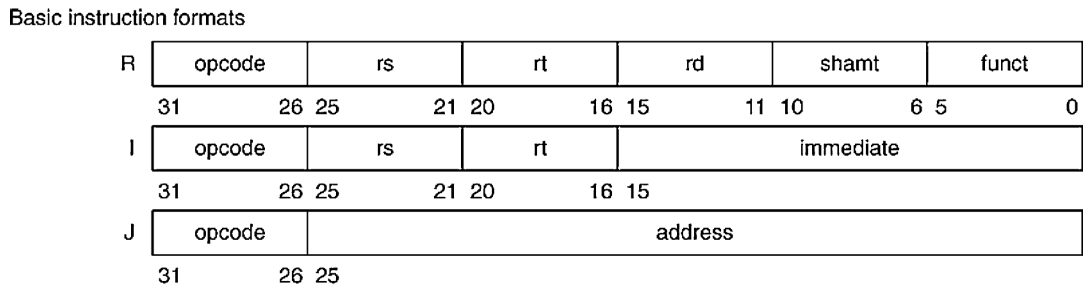

Fig. 19. Instruction formats for a MIPS processor [163].

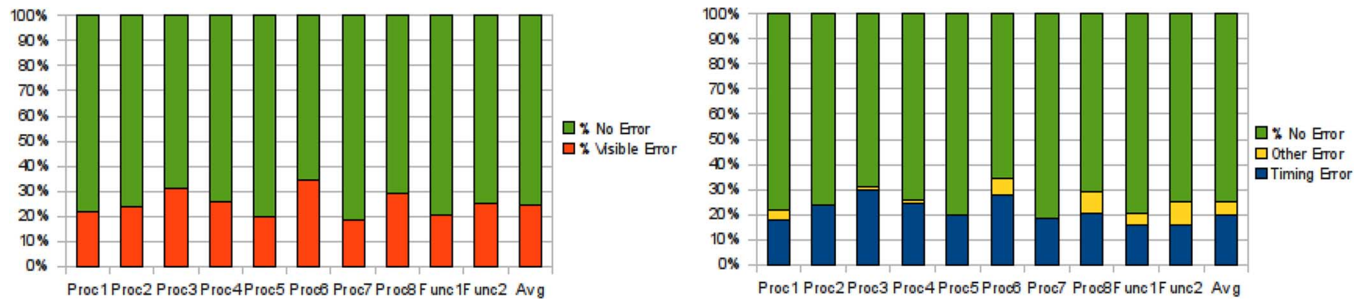

Fig. 20. Error distribution for each section of the benchmark [81].

Fig. 19 shows that software-based fault injection simulations targeting the control-flow circuit showed that over 25% of faults caused errors while running the Dhrystone synthetic computing benchmark. In addition, over 80% of the visible errors affect the commitment time of the current instruction [81]. On some occasions, the instruction was transformed into another instruction that required more cycles; on other occasions, the reverse scenario occurred, and the cycle count was reduced.

2) Validation With Experimental Data: Although this case study has used fault injection to identify the observed errors within the simulation of a benchmark program, these results, like many published results in the computer architecture community, have not been formally validated using experimental techniques. Instead, computer architecture simulations are performed with tools such as gem5 [138] for full system simulation and SimPoint [168] to identify a set of relevant points in the program that represent its complete execution. For the computer architecture research community, these methods of validation are considered well-formed and understood. For the radiation effects community, however, further validation is needed. Future collaborations between the computer architecture and radiation effects community could be fruitful for both communities as the radiation effects community has a deep understanding of experimental validation, and the computer architecture community has a deep understanding of next-generation processing elements.

Validating these results for the MIPS R2000 processor would be very straightforward. As the MIPS R2000 processor exists and the faults are common, the results can be validated using either a radiation source or a pulsed laser with sufficient beam time to enable the dynamic execution of the benchmark program.

VII. Conclusion

In this paper, we have presented methodologies, techniques, and case studies on fault injection systems. Fault injection systems cover a wide array of techniques, including hardware-in-the-loop fault emulation systems that inject faults directly into hardware and circuit-simulation-tool-based fault simulation systems that use testbenches to inject and stimulate the design. These systems can be very useful to designers as a method to augment traditional RHA methodologies for preparing designs for space readiness, after being properly validated with accelerated radiation test or pulsed laser testing.

References

[1] H. Quinn, P. Graham, J. Krone, M. Caffrey, and S. Rezgui, "Radiation-induced multi-bit upsets in SRAM-based FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 52, pp. 2455–2461, 2005.

[2] J. D. Black, R. A. Reed, C. Hafer, E. Peterson, J. Benedetto, and J. Wilkinson, "Soft errors: From the ground up," presented at the IEEE Nucl. Space Radiat. Effects Conf. Short Course, Oahu, HI, USA, 2008.

[3] N. Seifert, D. Moyer, N. Leland, and R. Hokinson, "Historical trends in alpha-particle induced soft error rates of the alpha (TM) microprocessor," in *39th Annu. Proc. Int. Reliab. Phys. Symp.*, 2001, pp. 259–265.

[4] N. Seifert, X. W. Zhu, and L. W. Massengill, "Impact of scaling on soft-error rates in commercial microprocessors," *IEEE Trans. Nucl. Sci.*, vol. 49, pp. 3100–3106, Dec. 2002.

[5] N. Seifert, B. Gill, K. Foley, and P. Relangi, "Multi-cell upset probabilities of 45 nm high-k plus metal gate SRAM devices in terrestrial and space environments," in *Proc. 46th Annu. IEEE Int. Reliab. Phys. Symp.*, 2008, pp. 181–186.

[6] J. Benedetto, P. Eaton, K. Avery, D. Mavis, M. Gadlage, T. Turflinger, P. E. Dodd, and G. Vizkelethyd, "Heavy ion-induced digital single-event transients in deep submicron processes," *IEEE Trans. Nucl. Sci.*, vol. 51, pp. 3480–3485, 2004.

[7] P. E. Dodd, M. R. Shaneyfelt, J. A. Felix, and J. R. Schwank, "Production and propagation of single-event transients in high-speed digital logic ICs," *IEEE Trans. Nucl. Sci.*, vol. 51, pp. 3278–3284, 2004.

[8] J. D. Black, D. R. Ball, W. H. Robinson, D. M. Fleetwood, R. D. Schrimpf, R. A. Reed, D. A. Black, K. M. Warren, A. D. Tipton, P. E. Dodd, N. F. Haddad, M. A. Xapsos, H. S. Kim, and M. Friendlich, "Characterizing SRAM single event upset in terms of single and multiple node charge collection," *IEEE Trans. Nucl. Sci.*, vol. 55, pp. 2943–2947, 2008.

[9] H. Quinn, K. Morgan, P. Graham, J. Krone, M. Caffrey, and K. Lundgreen, "Domain crossing errors: Limitations on single device triple-modular redundancy circuits in xilinx FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 54, pp. 2037–2043, 2007.

[10] H. T. Nguyen, Y. Yagil, N. Seifert, and M. Reitsma, "Chip-level soft error estimation method," *IEEE Trans. Device Mater. Rel.*, vol. 5, pp. 365–381, Sept. 2005.

[11] N. Seifert, "Soft error rates of RadHard sequentials utilizing local redundancy," in *Proc. IEEE Int. On-Line Test. Symp.*, 2008, pp. 49–50.

[12] D. B. Limbrick, S. G. Yue, W. H. Robinson, and B. L. Bhuva, "Impact of synthesis constraints on error propagation probability of digital circuits," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI and Nanotechnol, Syst. (DFT)*, 2011, pp. 103–111.

[13] W. H. Robinson, M. L. Alles, T. A. Bapty, B. L. Bhuva, J. D. Black, A. B. Bonds, L. W. Massengill, S. K. Neema, R. D. Schrimpf, and J. M. Scott, "Soft error considerations for multicore microprocessor design," in *Proc. IEEE Int. Conf. Integrated Circuit Design Technol. (ICICDT)*, 2007, pp. 1–4.

[14] T. D. Loveless, B. D. Olson, B. L. Bhuva, W. T. Holman, C. C. Hafer, and L. W. Massengill, "Analysis of single-event transients in integer-N frequency dividers and hardness assurance implications for phase-locked loops," *IEEE Trans. Nucl. Sci.*, vol. 56, pp. 3489–3498, Dec. 2009.

[15] T. D. Loveless, L. W. Massengill, W. T. Holman, B. L. Bhuva, D. McMorrow, and J. H. Warner, "A generalized linear model for single event transient propagation in phase-locked loops," *IEEE Trans. Nucl. Sci.*, vol. 57, pp. 2933–2947, Oct. 2010.

[16] J. Wu, Y. C. Ma, J. Zhang, and M. P. Xie, "A low-jitter synchronous clock distribution scheme using a DAC based PLL," *IEEE Trans. Nucl. Sci.*, vol. 57, pp. 589–594, Apr. 2010.

[17] A. Aloisio, R. Giordano, and V. Izzo, "Phase noise issues with FPGA-embedded DLLs and PLLs in HEP applications," *IEEE Trans. Nucl. Sci.*, vol. 58, pp. 1664–1671, Aug. 2011.

[18] N. Seifert, P. Shipley, M. D. Pant, V. Ambrose, and B. Gil, "Radiation-induced clock jitter and race," in *Proc. IEEE Int. Reliab. Phys. Symp.*, 2005, pp. 215–222.

[19] A. H. Johnston, T. F. Miyahira, F. Irom, and J. S. Laird, "Single-event transients in voltage regulators," *IEEE Trans. Nucl. Sci.*, vol. 53, pp. 3455–3461, Dec. 2006.

[20] P. C. Adell, R. D. Schrimpf, B. K. Choi, W. T. Holman, J. P. Attwood, C. R. Cirba, and K. F. Galloway, "Total-dose and single-event effects in switching DC/DC power converters," *IEEE Trans. Nucl. Sci.*, vol. 49, pp. 3217–3221, Dec. 2002.

[21] F. Irom, T. F. Miyahira, P. C. Adel, J. S. Laird, B. Conder, V. Pouget, and F. Essely, "Investigation of single-event transients in linear voltage regulators," *IEEE Trans. Nucl. Sci.*, vol. 55, pp. 3352–3359, Dec. 2008.

[22] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Depend. Secure Comput.*, vol. 1, pp. 11–33, Jan.–Mar. 2004.

[23] N. Seifert and N. Tam, "Timing vulnerability factors of sequentials," *IEEE Trans. Device Mater. Rel.*, vol. 4, pp. 516–522, 2004.

[24] S. Mukherjee, *Architecture Design for Soft Errors*. Burlington, MA, USA: Morgan Kaufmann, 2008.

[25] N. Miskov-Zivanov and D. Marculescu, "Modeling and optimization for soft-error reliability of sequential circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, pp. 803–816, 2008.

[26] D. Black, "Direct ionization-induced transient fault analysis for combinational logic and sequential capture in digital integrated circuits for lightly-ionizing environments," Doctor of Philosophy dissertation, Electr. Eng. Dept., Vanderbilt Univ., Nashville, TN, USA, 2011.

[27] E. Johnson, M. Wirthlin, and M. Caffrey, "Single-event upset simulation on an FPGA," in *Proc. Int. Conf. Eng. Reconfigurable Syst. Algorithms (ERSA)*, 2002, pp. 68–73.

[28] P. Civera, L. Macchiarulo, M. Rebaudengo, M. S. Reorda, and M. Violante, "Exploiting circuit emulation for fast hardness evaluation," *IEEE Trans. Nucl. Sci.*, vol. 48, pp. 2210–2216, Dec. 2001.

[29] F. Irom, "Guideline for ground radiation testing of microprocessors in the space radiation environment," 2008 [Online]. Available: http://trs-new.jpl.nasa.gov/dspace/bitstream/2014/40790/1/08-13.pdf

[30] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Proc. 36th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO-36)*, 2003, pp. 29–40.

[31] V. Sridharan and D. R. Kaeli, "Eliminating microarchitectural dependency from architectural vulnerability," in *Proc. 15th Int. Symp. High-Perform. Comput. Architect. (HPCA-15)*, 2009, pp. 117–128.

[32] P. Ramachandran, P. Kudva, J. Kellington, J. Schumann, and P. Sanda, "Statistical fault injection," in *Proc. IEEE Int. Conf. Depend. Syst. Netw. (DSN)*, 2008, pp. 122–127.

[33] J. P. C. Kleijnen, *Statistical Techniques in Simulation (in Two Parts)*. New York, NY, USA: Marcel Dekker, 1974.

[34] M. Wirthlin, E. Johnson, N. Rollins, M. Caffrey, and P. Graham, "The reliability of FPGA circuit designs in the presence of radiation induced configuration upsets," in *Proc. IEEE Symp. Field-Programmable Custom Comput. Mach.*, 2003.

[35] P. Graham, M. Caffrey, M. Wirthlin, E. Johnson, and N. Rollins, "SEU mitigation for half-latches in Xilinx Virtex FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 50, pp. 2139–2146, 2003.

[36] S. Lee, S. Jeon, S. Baeg, and D. Lee, "Memory reliability analysis for multiple block effect of soft errors," *IEEE Trans. Nucl. Sci.*, vol. 60, no. 2, pp. 1384–1389, 2013.

[37] R. Leveugle, A. Calvez, P. Vanhauwaert, and P. Maistri, "Precisely controlling the duration of fault in ection campaigns: A statistical view," in *Proc. 4th IEEE Int. Conf. Design Technol. Integr. Syst. in Nanoscale Era*, 2009, pp. 149–154.

[38] M. Cukier, D. Powell, and J. Arlat, "Coverage estimation methods for stratified fault-injection," *IEEE Trans. Comput.*, vol. 48, pp. 707–723, July 1999.

[39] C. Constantinescu, "Using multistage and stratified sampling for inferring fault-coverage probabilities," *IEEE Trans. Rel.*, vol. 44, pp. 632–639, Dec. 1995.

[40] J. Teich, "Hardware/software codesign: The past, the present, and predicting the future," *Proc. IEEE*, vol. 100, pp. 1411–1430, 2012.

[41] Z. K. Baker, M. E. Dunham, and K. Morgan *et al.*, "Space-based FPGA radio receiver design, debug, and development of a radiation-tolerant computing system," *Int. J. Reconfigurable Comput.*, vol. 2010, Article ID 546217, 12 pp., 2010, doi: 10.1155/2010/546217.

[42] R. Katz and J. J. Wang, "Using IEEE 1149.1 JTAG circuitry in Actel SX devices," 1998 [Online]. Available: http://www.actel.com/documents/JTAG_SX.pdf

[43] M. Portela-Garcia, C. Lopez-Ongil, M. G. Valderas, and L. Entrena, "Fault injection in modern microprocessors using on-chip debugging infrastructures," *IEEE Trans. Depend. Secure Comput.*, vol. 8, pp. 308–314, Mar.–Apr. 2011.

[44] M. A. Aguirre, J. N. Tombs, A. Torralba, and L. G. Franquelo, "UNSHADES-1: An advanced tool for in-system run-time hardware debugging," *Proc. Field-Programmable Logic Appl.*, vol. 2778, pp. 1170–1173, 2003.

[45] C. Lopez-Ongil, M. Garcia-Valderas, M. Portela-Garcia, and L. Entrena-Arrontes, "Techniques for fast transient fault grading based on autonomous emulation," in *Proc. Design, Autom., Test Eur. Conf.*, 2005, pp. 308–309.

[46] L. Sterpone, N. Battezzati, and V. Ferlet-Cavrois, "Analysis of SET propagation in flash-based FPGAs by means of electrical pulse injection," *IEEE Trans. Nucl. Sci.*, vol. 57, pp. 1820–1826, Aug. 2010.

[47] E. Johnson, M. Caffrey, P. Graham, N. Rollins, and M. Wirthlin, "Accelerator validation of an FPGA SEU simulator," *IEEE Trans. Nucl. Sci.*, vol. 50, pp. 2147–2157, 2003.

[48] G. Swift, S. Rezgui, J. George, C. Carmichael, M. Napier, J. Maksymowicz, J. Moore, A. Lesea, R. Koga, and T. F. Wrobel, "Dynamic testing of Xilinx Virtex-II field programmable gate array (FPGA) input/output blocks (IOBs)," *IEEE Trans. Nucl. Sci.*, vol. 51, pp. 3469–3474, 2004.

[49] M. Wirthlin, E. Johnson, P. Graham, and M. Caffrey, "Validation of a fault simulator for field programmable gate arrays," in *Proc. IEEE Nucl. Space Radiat. Effects Conf.*, 2003.

[50] E. Johnson, K. Morgan, N. Rollins, M. Wirthlin, M. Caffrey, and P. Graham, "Detection of configuration memory upsets causing persistent errors in SRAM-based FPGAs," in *Proc. MAPLD Conf.*, 2004.

[51] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin, "SEU-induced persistent error propagation in FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 52, pp. 2438–2445, 2005.

[52] M. French, M. Wirthlin, and P. Graham, "Reducing power consumption of radiation mitigated designs for FPGAs," in *Proc. 9th Annu. Int. Conf. Mil. Aerosp. Programmable Logic Devices (MAPLD)*, 2006.

[53] P. Ostler, M. Caffrey, D. Gibelyou, P. Graham, K. Morgan, B. Pratt, and H. Quinn, "SRAM FPGA reliability analysis for harsh radiation environments," *IEEE Trans. Nucl. Sci.*, vol. 56, pp. 3519–3526, 2009.

[54] M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, S. Pastore, G. R. Sechi, and R. Weigand, "Evaluation of single event upset mitigation schemes for SRAM based FPGAs using the FLIPPER fault injection platform," in *Proc. 22nd IEEE Int. Symp. Defect Fault Tolerance VLSI Syst. (DFT)*, 2007, pp. 105–113.

[55] M. Berg, C. Perez, and H. Kim, "Investigating mitigated and non-mitigated multiple clock domain circuitry in a xilinx virtex-4 field programmable gate arrays," in *Proc. Single-Event Effects Symp.*, 2008.

[56] G. Swift, C. Tseng, G. Miller, G. Allen, and H. Quinn, "The use of fault injection to simulate upsets in reconfigurable FPGAs," in *Mil. Aerosp. Programmable Logic Devices Conf.*, Annapolis, MD, USA, 2008 [Online]. Available: https://nepp.nasa.gov/mapld_2008/presentations/t/05%20-%20Swift_Gary_mapld08_pres_1.pdf

[57] G. Cieslewski, A. George, and A. Jacobs, "Acceleration of FPGA fault injection through multi-bit testing," presented at the Int. Conf. Eng. Reconfigurable Syst. Algorithms (ERSA), Las Vegas, NV, USA, 2010.

[58] G. Cieslewski and A. George, "SPFFI: Simple portable FPGA fault injector," presented at the Mil. Aerosp. Programmable Logic Devices Conf. (MAPLD), Greenbelt, MD, USA, 2009.

[59] M. G. Valderas, L. Entrena, R. F. Cardenal, C. L. Ongil, and M. P. Garcia, "SET emulation under a quantized delay model," *J. Electron. Testing-Theory Appl.*, vol. 25, pp. 107–116, Feb. 2009.

[60] L. Entrena, M. G. Valderas, R. F. Cardenal, M. P. Garcia, and C. L. Ongil, "SET emulation considering electrical masking effects," *IEEE Trans. Nucl. Sci.*, vol. 56, pp. 2021–2025, Aug. 2009.

[61] M. A. Aguirre, V. Baena, J. Tombs, and M. Violante, "A new approach to estimate the effect of single event transients in complex circuits," *IEEE Trans. Nucl. Sci.*, vol. 54, pp. 1018–1024, Aug. 2007.

[62] J. Carreira, H. Madeira, and J. G. Silva, "Xception: A technique for the experimental evaluation of dependability in modern computers," *IEEE Trans. Softw. Eng.*, vol. 24, pp. 125–136, Feb. 1998.

[63] R. Velazco, S. Rezgui, and R. Ecoffet, "Predicting error rate for microprocessor-based digital architectures through CEU (code emulating upsets) injection," *IEEE Trans. Nucl. Sci.*, vol. 47, pp. 2405–2411, Dec. 2000.

[64] B. Nicolescu, P. Peronnard, R. Velazco, and Y. Savaria, "Efficiency of transient bit-flips detection by software means: A complete study," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Syst.*, 2003, pp. 377–384.

[65] P. A. Ferreyra, C. A. Marques, R. Velazco, and O. Calvo, "Injecting single event upsets in a digital signal processor by means of direct memory access requests: A new method for generating bit flips," in *Proc. 2001 6th Eur. Conf. Radiat. Its Effects Compon. Syst.*, 2002, pp. 248–252.

[66] K. Zick, C. Yu, J. Walters, and M. French, "Silent data corruption and embedded processing with NASA's SpaceCube," *IEEE Embedded Syst. Lett.*, vol. PP, pp. 1–1, 2012.

[67] M. Bucciero, J. P. Walters, R. Moussalli, S. Y. Gao, and M. French, "The PowerPC 405 memory sentinel and injection system," in *Proc. IEEE 19th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, 2011, pp. 154–161.

[68] J. J. Peng, J. Ma, B. R. Hong, and C. J. Yuan, "Validation of fault tolerance mechanisms of an onboard system," in *Proc. 1st Int. Symp. Syst. Control Aerosp. Astronaut. (ISSCAA)*, 2006, vol. 1 and 2, pp. 1230–1234.

[69] M. Rebaudengo and M. S. Reorda, "Evaluating the fault tolerance capabilities of embedded systems via BDM," in *Proc. 17th IEEE VLSI Test Symp.*, 1999, pp. 452–457.

[70] A. V. Fidalgo, G. R. Alves, and J. M. Feffeira, "A modified debugging infrastructure to assist real time fault injection campaigns," in *Proc. IEEE Workshop Des. Diagnost. Electron. Circuits Syst.*, 2006, pp. 174–179.

[71] M. A. Aguirre, J. N. Tombs, F. Munoz, V. Baena, H. Guzman, J. Napoles, A. Fernandez-Leon, F. Tortosa-Lopez, and D. Merodio, "Selective protection analysis using a SEU emulator: Testing protocol and case study over the Leon2 processor," *IEEE Trans. Nucl. Sci.*, vol. 54, pp. 951–956, Aug. 2007.

[72] M. A. Aguirre, J. Tombs, A. Torralba, and L. G. Franquelo, "Improving the design process of VLSI circuits by means of a hardware debugging system: UNSHADES-1 framework," in *-2002: Proc. 28th Annu. Conf. IEEE Ind. Electron. Soc. (IECON)*, 2002, vol. 1–4, pp. 2544–2547.

[73] C. Lopez-Ongil, L. Entrena, M. Garcia-Valderas, M. Portela, M. A. Aguirre, J. Tombs, V. Baena, and F. Munoz, "A unified environment for fault injection at any design level based on emulation," *IEEE Trans. Nucl. Sci.*, vol. 54, pp. 946–950, Aug. 2007.

[74] J. Napoles, H. Guzman, M. Aguirre, J. N. Tombs, F. Munoz, V. Baena, A. Torralba, and L. G. Franquelo, "Radiation environment emulation for VLSI designs: A low cost platform based on Silinx FPGA's," in *Proc. IEEE Int. Symp. Ind. Electron.*, 2007, pp. 3334–3338.

[75] J. Napoles, H. Guzman-Miranda, M. Aguirre, J. N. Tombs, J. M. Mogollon, R. Palomo, and A. P. Vega-Leal, "A complete emulation system for single event effects analysis," in *Proc. Southern Conf. Programmable Logic*, 2008, pp. 213–216.

[76] M. A. Aguirre, J. N. Tombs, and H. G. Miranda, "Fault injection analysis of bidirectional signals," *IEEE Trans. Nucl. Sci.*, vol. 56, pp. 2179–2183, Aug. 2009.

[77] H. Guzman-Miranda, M. A. Aguirre, and J. Tombs, "Noninvasive fault classification, robustness and recovery time measurement in microprocessor-type architectures subjected to radiation-induced errors," *IEEE Trans. Instrum. Meas.*, vol. 58, pp. 1514–1524, May 2009.

[78] M. G. Valderas, P. Peronnard, C. L. Ongil, R. Ecoffet, F. Bezerra, and R. Velazco, "Two complementary approaches for studying the effects of SEUs on digital processors," *IEEE Trans. Nucl. Sci.*, vol. 54, pp. 924–928, Aug. 2007.

[79] C. Lopez-Ongil, M. Garcia-Valderas, M. Portela-Garcia, and L. Entrena-Arrontes, "Autonomous transient fault emulation on FPGAs for accelerating fault grading," in *Proc. 11th IEEE Int. On-Line Testing Symp.*, 2005, pp. 43–48.

[80] L. Antoni, R. Leveugle, and B. Feher, "Using run-time reconfiguration for fault injection applications," *IEEE Trans. Instrum. Meas.*, vol. 52, pp. 1468–1473, Oct. 2003.

[81] D. B. Limbrick, E. J. Ossi, C. T. Toomey, W. H. Robinson, and B. Bhuva, "Characterization of control bit errors in the MIPS R2000 microprocessor," presented at the 35th Annu. Government Microcircuit Appl. Critical Technol. Conf. (GOMACTech), Reno, NV, USA, 2010.

[82] R. Harada, Y. Mitsuyama, M. Hashimoto, and T. Onoye, "Neutron induced single event multiple transients with voltage scaling and body biasing," in *Proc. IEEE Int. Reliab. Phys. Symp. (IRPS)*, 2011, pp. 3C.4.1–3C.4.5.

[83] S. Pagliarini, F. Kastensmidt, L. Entrena, A. Lindoso, and E. S. Millan, "Analyzing the impact of single-event-induced charge sharing in complex circuits," *IEEE Trans. Nucl. Sci.*, vol. 58, pp. 2768–2775, 2011.

[84] B. T. Kiddie, W. H. Robinson, and D. B. Limbrick, "Single-event multiple-transients (SEMT): Circuit characterization and analysis," presented at the IEEE Workshop Silicon Errors in Logic—System Effects (SELSE), Palo Alto, CA, USA, 2013.

[85] E. L. Petersen, J. C. Pickel, J. H. Adams, Jr., and E. C. Smith, "Rate prediction for single event effects-a critique," *IEEE Trans. Nucl. Sci.*, vol. 39, pp. 1577–1599, 1992.

[86] L. W. Massengill, A. E. Baranski, D. O. Van Nort, J. Meng, and B. L. Bhuva, "Analysis of single-event effects in combinational logic-simulation of the AM2901 bitslice processor," *IEEE Trans. Nucl. Sci.*, vol. 47, pp. 2609–2615, 2000.

[87] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, vol. 35, pp. 677–691, Aug. 1986.

[88] R. E. Bryant, "Symbolic boolean manipulation with ordered binary-decision diagrams," *Comput. Surv.*, vol. 24, pp. 293–318, Sep. 1992.

[89] M.-Z. Natasa and M. Diana, "Circuit reliability analysis using symbolic techniques," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 25, pp. 2638–2649, 2006.

[90] Z. Ming and N. R. Shanbhag, "Soft-error-rate-analysis (SERA) methodology," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 25, pp. 2140–2155, 2006.

[91] S. A. Seshia, W. C. Li, and S. Mitra, "Verification-guided soft error resilience," in *Proc. Design, Autom., Test in Eur. Conf. Exhib.*, 2007, vol. 1–3, pp. 1442–1447.

[92] R. Rajaraman, J. S. Kim, N. Vijaykrishnan, Y. Xie, and M. J. Irwin, "SEAT-LA: A soft error analysis tool for combinational logic," in *Proc. Int. Conf. VLSI Design/Int. Conf. Embedded Syst. and Design*, 2006, p. 4, pp..

[93] R. R. Rao, K. Chopra, D. T. Blaauw, and D. M. Sylvester, "Computing the soft error rate of a combinational logic circuit using parameterized descriptors," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, pp. 468–479, Mar. 2007.

[94] L. Sterpone, N. Battezzati, F. L. Kastensmidt, and R. Chipana, "An analytical model of the propagation induced pulse broadening (PIPB) effects on single event transient in flash-based FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 58, pp. 2333–2340, Oct. 2011.

[95] L. Sterpone, F. Margaglia, M. Koester, J. Hagemeyer, and M. Porrmann, "Analysis of SEU effects in partially reconfigurable SoPCs," in *Proc. NASA/ESA Conf. Adapt. Hardware Syst. (AHS)*, 2011, pp. 129–136.

[96] L. Sterpone, M. Violante, A. Panariti, A. Bocquillon, F. Miller, N. Buard, A. Manuzzato, S. Gerardin, and A. Paccagnella, "Layout-aware multi-cell upsets effects analysis on TMR circuits implemented on SRAM-based FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 58, pp. 2325–2332, Oct. 2011.

[97] K. R. Walcott, G. Humphreys, and S. Gurumurthi, "Dynamic prediction of architectural vulnerability from microarchitectural state," in *Proc. Int. Symp. Comput. Architect.*, 2007, pp. 516–527.

[98] X. D. Li, S. V. Adve, P. Bose, and J. A. Rivers, "Online estimation of architectural vulnerability factor for soft errors," in *Proc. Int. Symp. Comput. Architect.*, 2008, pp. 341–352.

[99] P. E. Dodd and L. W. Massengill, "Basic mechanisms and modeling of single-event upset in digital microelectronics," *IEEE Trans. Nucl. Sci.*, vol. 50, pp. 583–602, 2003.

[100] P. E. Dodd, "Physics-based simulation of single-event effects," *IEEE Trans. Device Mater. Rel.*, vol. 5, pp. 343–357, 2005.

[101] G. C. Messenger, "Collection of charge on junction nodes from ion tracks," *IEEE Trans. Nucl. Sci.*, vol. 29, pp. 2024–2031, 1982.

[102] SEUs Simulation Tool, 2012 [Online]. Available: http://www.nebrija.es/~jmaestro/esa/sst.htm, SEUs Simulation Tool Available:

[103] R. A. Reed, M. Xapsos, G. Santini, M. Law, J. D. Black, and T. Holman, "Modeling the space radiation environment and effects on microelectronic devices and circuits," in *IEEE Nucl. Space Radiat. Effects Conf. Short Course*, 2006.

[104] V. Srinivasan, A. L. Sternberg, A. R. Duncan, W. H. Robinson, B. L. Bhuva, and L. W. Massengill, "Single-event mitigation in combinational logic using targeted data path hardening," *IEEE Trans. Nucl. Sci.*, vol. 52, pp. 2516–2523, Dec. 2005.

[105] Y. S. Dhillon, A. U. Diril, and A. Chatterjee, "Soft-error tolerance analysis and optimization of nanometer circuits," in *Proc. Design, Autom., Test Eur.*, 2005, vol. 1, pp. 288–293.

[106] S. J. Wen, D. Alexandrescu, and R. Perez, "A systematical method of quantifying SEU FIT," in *Proc. 14th IEEE Int. On-Line Testing Symp.*, 2008, pp. 109–114.

[107] S. Yoshimoto, T. Amashita, D. Kozuwa, T. Takata, M. Yoshimura, Y. Matsunaga, H. Yasuura, H. Kawaguchi, and M. Yoshimoto, "Multiple-bit-upset and single-bit-upset resilient 8T SRAM bitcell layout with divided wordline structure," in *Proc. IEEE 17th Int. On-Line Testing Symp. (IOLTS)*, 2011.

[108] D. Alexandrescu, E. Costenaro, and M. Nicolaidis, "A practical approach to single event transients analysis for highly complex designs," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, 2011, pp. 155–163.

[109] H. Chapman, E. Landman, A. Margalit-Ilovich, Y. Fang, A. Oates, D. Alexandrescu, and O. Lauzeral, "A multi-partner soft error rate analysis of an infiniband host channel adapter," in *Proc. Silicon Errors in Logic-Syst. Effects (SELSE)*, 2010.

[110] J. Bergeron, *Writing Testbenches-Functional Verification of HDL Models*. Norwell, MA, USA: Kluwer Academic, 2002.

[111] M. Benjamin, D. Geist, A. Hartman, Y. Wolfsthal, G. Mas, and R. Smeets, "A study in coverage-driven test generation," in *Proc. 36th Design Autom. Conf.*, 1999, 2010, pp. 970–975.

[112] J. R. Schwank, M. R. Shaneyfelt, and P. E. Dodd, "Radiation hardness assurance testing of microelectronic devices and integrated circuits: Radiation environments, physical mechanisms, and foundations for hardness assurance," *IEEE Trans. Nucl. Sci.*, vol. 60, no. 3, pp. 2074–2100, Jun. 2013.

[113] J. R. Schwank, M. R. Shaneyfelt, and P. E. Dodd, "Radiation hardness assurance testing of microelectronic devices and integrated circuits: Test guideline for proton and heavy ion single-event effects," *IEEE Trans. Nucl. Sci.*, vol. 60, no. 3, pp. 2101–2118, Jun. 2013.

[114] S. Buchner, F. Miller, V. Pouget, and D. McMorrow, "Pulsed laser testing for Single-Event Effects investigations," *IEEE Trans. Nucl. Sci.*, vol. 60, no. 3, pp. 1852–1875, Jun. 2013.

[115] K. M. Horn, B. L. Doyle, and F. W. Sexton, "Nuclear microprobe imaging of single-event upsets," *IEEE Trans. Nucl. Sci.*, vol. 39, pp. 7–12, Feb. 1992.

[116] F. W. Sexton, "Microbeam studies of single-event effects," *IEEE Trans. Nucl. Sci.*, vol. 43, pp. 687–695, Apr. 1996.

[117] P. Chu, D. L. Hansen, B. L. Doyle, K. Jobe, R. Lopez-Aguado, M. Shoga, and D. S. Walsh, "Ion-microbeam probe of high-speed shift registers for SEE analysis—Part I: SiGe," *IEEE Trans. Nucl. Sci.*, vol. 53, pp. 1574–1582, June 2006.

[118] J. A. Pellish, R. A. Reed, D. McMorrow, G. Vizkelethy, V. F. Cavrois, J. Baggio, P. Paillet, O. Duhamel, K. A. Moen, S. D. Phillips, R. M. Diestelhorst, J. D. Cressler, A. K. Sutton, A. Raman, M. Turowski, P. E. Dodd, M. L. Alles, R. D. Schrimpf, P. W. Marshall, and K. A. LaBel, "Heavy ion microbeam- and broadbeam-induced transients in SiGe HBTs," *IEEE Trans. Nucl. Sci.*, vol. 56, pp. 3078–3084, Dec. 2009.

[119] F. Miller, N. Buard, G. Hubert, S. Alestra, G. Baudrillard, T. Carriere, R. Gaillard, J. M. Palau, F. Saigne, and P. Fouillat, "Laser mapping of SRAM sensitive cells: A way to obtain input parameters for DASIE calculation code," *IEEE Trans. Nucl. Sci.*, vol. 53, pp. 1863–1870, Aug. 2006.

[120] D. McMorrow, W. T. Lotshaw, J. S. Melinger, S. Buchner, and R. L. Pease, "Subbandgap laser-induced single event effects: Carrier generation via two-photon absorption," *IEEE Trans. Nucl. Sci.*, vol. 49, pp. 3002–3008, Dec. 2002.

[121] J. S. Melinger, S. Buchner, D. Mcmorrow, W. J. Stapor, T. R. Weatherford, and A. B. Campbell, "Critical-evaluation of the pulsed-laser method for single event effects testing and fundamental-studies," *IEEE Trans. Nucl. Sci.*, vol. 41, pp. 2574–2584, Dec. 1994.

[122] C. Godlewski, V. Pouget, D. Lewis, and M. Lisart, "Electrical modeling of the effect of beam profile for pulsed laser fault injection," *Microelectron. Rel.*, vol. 49, pp. 1143–1147, Sep.–Nov. 2009.

[123] V. Pouget, A. Douin, G. Foucard, P. Peronnard, D. Lewis, P. Fouillat, and R. Velazco, "Dynamic testing of an SRAM-based FPGA by time-resolved laser fault injection," in *Proc. IEEE Int. On-Line Testing Symp.*, 2008, pp. 295–301.

[124] V. Pouget, D. Lewis, and P. Fouillat, "Time-resolved scanning of integrated circuits with a pulsed laser: Application to transient fault injection in an ADC," *IEEE Trans. Instrum. Meas.*, vol. 53, pp. 1227–1231, Aug. 2004.

[125] H. Quinn, P. Graham, and B. Pratt, "An automated approach to estimating hardness assurance issues in triple-modular redundancy circuits in xilinx FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 55, pp. 3070–3076, Dec. 2008.

[126] H. M. Quinn, P. S. Graham, M. J. Wirthlin, B. Pratt, K. S. Morgan, M. P. Caffrey, and J. B. Krone, "A test methodology for determining space readiness of xilinx SRAM-based FPGA devices and designs," *IEEE Trans. Instrum. Meas.*, vol. 58, pp. 3380–3395, Oct. 2009.

[127] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 1990.

[128] H. Quinn, G. R. Allen, G. M. Swift, C. W. Tseng, P. S. Graham, K. S. Morgan, and P. Ostler, "SEU-susceptibility of logical constants in xilinx FPGA designs," *IEEE Trans. Nucl. Sci.*, vol. 56, pp. 3527–3533, Dec. 2009.

[129] M. Caffrey, M. Echave, C. Fite, T. Nelson, A. Salazar, and S. Storms, "A space-based reconfigurable radio," in *Proc. 5th Annu. Int. Conf. Mil. Aerosp. Programmable Logic Devices (MAPLD)*, 2002.

[130] H. Quinn, P. Graham, K. Morgan, Z. K. Baker, M. Caffrey, D. Smith, and R. Bell, "On-orbits results for the xilinx virtex-4 FPGA," presented at the IEEE Nucl. Space Radiat. Effects Conf. (NSREC) Radiat. Data Workshop, Miami Beach, FL, USA, 2012.

[131] J. M. Benedetto, P. H. Eaton, D. G. Mavis, M. Gadlage, and T. Turflinger, "Variation of digital SET pulse widths and the implications for single event hardening of advanced CMOS processes," *IEEE Trans. Nucl. Sci.*, vol. 52, pp. 2114–2119, 2005.

[132] J. R. Ahlbin, J. D. Black, L. W. Massengill, O. A. Amusan, A. Balasubramanian, M. C. Casey, D. A. Black, M. W. McCurdy, R. A. Reed, and B. L. Bhuva, "C-CREST technique for combinational logic SET testing," *IEEE Trans. Nucl. Sci.*, vol. 55, pp. 3347–3351, 2008.

[133] A. D. Tipton, J. A. Pellish, R. A. Reed, R. D. Schrimpf, R. A. Weller, M. H. Mendenhall, B. Sierawski, A. K. Sutton, R. M. Diestelhorst, G. Espinel, J. D. Cressler, P. W. Marshall, and G. Vizkelethy, "Multiple-bit upset in 130 nm CMOS technology," *IEEE Trans. Nucl. Sci.*, vol. 53, pp. 3259–3264, 2006.

[134] R. A. Weller, R. D. Schrimpf, R. A. Reed, M. H. Mendenhall, K. M. Warren, B. D. Sierawski, and L. W. Massengill, "Monte carlo simulation of single event effects," *RADECS Short Course*, 2009.

[135] K. M. Warren, "Sensitive volume models for single event upset analysis and rate prediction for space, atmospheric, and terrestrial radiation environments," Ph.D. dissertation, Electrical Eng. Dept., Vanderbilt Univ., Nashville, TN, USA, 2010.

[136] K. M. Warren, A. L. Sternberg, J. D. Black, R. A. Weller, R. A. Reed, M. H. Mendenhall, R. D. Schrimpf, and L. W. Massengill, "Heavy ion testing and single event upset rate prediction considerations for a DICE flip-flop," *IEEE Trans. Nucl. Sci.*, vol. 56, pp. 3130–3137, 2009.

[137] R. A. Weller, R. A. Reed, K. M. Warren, M. H. Mendenhall, B. D. Sierawski, R. D. Schrimpf, and L. W. Massengill, "General framework for single event effects rate prediction in microelectronics," *IEEE Trans. Nucl. Sci.*, vol. 56, pp. 3098–3108, 2009.

[138] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Architect. News*, vol. 39, pp. 1–7, 2011.

[139] E. H. Cannon and M. Cabanas-Holmen, "Heavy ion and high energy proton-induced single event transients in 90 nm inverter, NAND and NOR gates," *IEEE Trans. Nucl. Sci.*, vol. 56, pp. 3511–3518, 2009.

[140] B. Sierawski, "LibsingleEvent library module," Inst. for Space and Defence Electronics, Vanderbilt Univ., Nashville, TN, USA, 2010, Engineering Research and Development.

[141] N. M. Atkinson, A. F. Witulski, W. T. Holman, B. L. Bhuva, J. D. Black, and L. W. Massengill, "Single event characterization of a 90 nm bulk CMOS digital cell library," in *Proc. Govern. Microcircuit Appl. Critical Technol. Conf.*, 2010.

[142] J. D. Black, L. Massengill, B. Bhuva, W. T. Holman, and K. M. Warren, "Final report to Boeing phantom works," Inst. for Space and Defense Electronics, Vanderbilt Univ., Nashville, TN, USA, 2008.

[143] A. J. Tylka, J. H. Adams, Jr., P. R. Boberg, B. Brownstein, W. F. Dietrich, E. O. Flueckiger, E. L. Petersen, M. A. Shea, D. F. Smart, and E. C. Smith, "CREME96: A revision of the cosmic ray effects on micro-electronics code," *IEEE Trans. Nucl. Sci.*, vol. 44, pp. 2150–2160, 1997.

[144] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, "The soft error problem: An architectural perspective," in *Proc. 11th Int. Symp. High-Perform. Comput. Architect. (HPCA-11)*, 2005, pp. 243–247.

[145] T. N. Vijaykumar, I. Pomeranz, and K. Cheng, "Transient-fault recovery using simultaneous multithreading," in *Proc. 29th Annu. Int. Symp. Comput. Architect.*, 2002, pp. 87–98.

[146] R. Hyman, K. Bhattacharya, and N. Ranganathan, "Redundancy mining for soft error detection in multicore processors," *IEEE Trans. Comput.*, vol. 60, pp. 1114–1125, 2011.

[147] G. P. Saggese, N. J. Wang, Z. T. Kalbarczyk, S. J. Patel, and R. K. Iyer, "An experimental study of soft errors in microprocessors," *IEEE Micro*, vol. 25, pp. 30–39, 2005.

[148] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel, "Characterizing the effects of transient faults on a high-performance processor pipeline," in *Proc. Int. Conf. Depend. Syst. Netw. (DSN)*, Florence, Italy, 2004, pp. 61–70.

[149] O. A. Amusan, L. W. Massengill, B. L. Bhuva, S. DasGupta, A. F. Witulski, and J. R. Ahlbin, "Design techniques to reduce SET pulse widths in deep-submicron combinational logic," *IEEE Trans. Nucl. Sci.*, vol. 54, pp. 2060–2064, 2007.

[150] O. A. Amusan, L. W. Massengill, M. P. Baze, B. L. Bhuva, A. F. Witulski, J. D. Black, A. Balasubramanian, M. C. Casey, D. A. Black, J. R. Ahlbin, R. A. Reed, and M. W. McCurdy, "Mitigation techniques for single event induced charge sharing in a 90 nm bulk CMOS process," in *Proc. IEEE Int. Rel. Phys. Symp. (IRPS)*, 2008, pp. 468–472.

[151] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J. C. Fabre, J. C. Laprie, E. Martins, and D. Powell, "Fault injection for dependability validation: A methodology and some applications," *IEEE Trans. Softw. Eng.*, vol. 16, pp. 166–182, 1990.

[152] J. A. Clark and D. K. Pradhan, "Fault injection: A method for validating computer-system dependability," *Computer*, vol. 28, pp. 47–56, 1995.

[153] M. C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, pp. 75–+, Apr. 1997.

[154] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. S. Mukherjee, and R. Rangan, "Computing architectural vulnerability factors for address-based structures," in *Proc. 32nd Int. Symp. Comput. Architect. (ISCA)*, Madison, WI, USA, 2005, pp. 532–543.

[155] A. Biswas, C. Recchia, S. S. Mukherjee, V. Ambrose, L. Chan, A. Jaleel, A. E. Papathanasiou, M. Plaster, and N. Seifert, "Explaining cache SER anomaly using DUE AVF measurement," in *Proc. IEEE 16th Int. Symp. High Perform. Comput. Architect. (HPCA)*, 2010, pp. 1–12.

[156] X. Li, S. V. Adve, P. Bose, and J. A. Rivers, "Architecture-level soft error analysis: Examining the limits of common assumptions," in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw. (DSN)*, 2007, pp. 266–275.

[157] P. N. Sanda, J. W. Kellington, P. Kudva, R. Kalla, R. B. McBeth, J. Ackaret, R. Lockwood, J. Schumann, and C. R. Jones, "Soft-error resilience of the IBM POWER6 processor," *IBM J. Res. Develop.*, vol. 52, pp. 275–284, 2008.

[158] A. A. Nair, L. K. John, and L. Eeckhout, "AVF stressmark: Towards an automated methodology for bounding the worst-case vulnerability to soft errors," in *Proc. 43rd Annu. IEEE/ACM Int. Symp. Microarchitect. (MICRO-43)*, 2010, pp. 125–136.

[159] X. Li, S. V. Adve, P. Bose, and J. A. Rivers, "SoftArch: An architecture-level tool for modeling and analyzing soft errors," in *Proc. Int. Conf. Depend. Syst. Netw. (DSN)*, Yokohama, Japan, 2005, pp. 496–505.

[160] M.-L. Li, P. Ramachandran, U. R. Karpuzcu, S. Hari, and S. V. Adve, "Accurate microarchitecture-level fault modeling for studying hardware faults," in *Proc. 15th Int. Symp. High Perform. Comput. Architect. (HPCA)*, 2009, pp. 105–116.

[161] M. Gschwind, V. Salapura, C. Trammell, and S. A. McKee, "SoftBeam: Precise tracking of transient faults and vulnerability analysis at processor design time," in *Proc. IEEE 29th Int. Conf. Comput. Design (ICCD)*, 2011, pp. 404–410.

[162] S. K. S. Hari, S. V. Adve, H. Naeimi, and P. Ramachandran, "Relyzer: Exploiting application-level fault equivalence to analyze application resiliency to transient faults," in *Proc. 17th Int. Conf. Architect. Support for Programming Lang. Operat. Syst. (ASPLOS)*, London, England, U.K., 2012, pp. 123–134.

[163] J. L. Hennessy and D. A. Patterson*, Computer Architecture A Quantitative Approach*, 4th ed. San Francisco, CA, USA: Morgan Kaufmann, 2007.

[164] H. Quinn, J. Tripp, T. Fairbanks, and A. Manuzzator, "Improving microprocessor reliability through software mitigation," presented at the Silicon Errors in Logic-Syst. Effects (SELSE), 2011.

[165] N. Karimi, M. Maniatakos, A. Jas, and Y. Makris, "On the correlation between controller faults and instruction-level errors in modern microprocessors," in *Proc. IEEE Int. Test Conf. (ITC)*, 2008, pp. 1–10.

[166] M. Maniatakos, N. Karimi, Y. Makris, A. Jas, and C. Tirumurti, "Design and evaluation of a timestamp-based concurrent error detection (CED) method in a modern microprocessor controller," presented at the IEEE Int. Symp. Defect Fault Tolerance in VLSI Syst. (DFT), Cambridge, MA, USA, 2008.

[167] N. Pinckney, T. Barr, M. Dayringer, M. McKnett, J. Nan, C. Nygaard, D. M. Harris, J. Stanley, and B. Phillips, "A MIPS R2000 implementation," in *Proc. 45th ACM/IEEE Design Autom. Conf. (DAC)*, 2008, pp. 102–107.

[168] E. Perelman, G. Hamerly, M. V. Biesbrouck, T. Sherwood, and B. Calder, "Using SimPoint for accurate and efficient simulation," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, San Diego, CA, USA, 2003, pp. 318–319.