LA-UR-08-4216

Approved for public release;
distribution is unlimited.

Title:	A Test Methodology for Determining Space-Readiness of Xilinx SRAM-Based FPGA Designs
Author(s):	Heather Quinn, Paul Graham, Keith Morgan, Michael Caffrey, and Jim Krone
Intended for:	Autotestcon 2008 Salt Lake City, UT 9/8-11/2008

Los Alamos
NATIONAL LABORATORY
EST 1943

Form 836 (7/06)

A Test Methodology for Determining Space-Readiness of Xilinx SRAM-based FPGA Designs

Heather Quinn, Paul Graham, Keith Morgan, Michael Caffrey, and Jim Krone
ISR-3 Space Data Systems, Los Alamos National Laboratory, Los Alamos, NM, 87545 USA

Abstract *- Using reconfigurable, static random-access memory (SRAM) based field-programmable gate arrays (FPGAs) for space-based computation has been an exciting area of research for the past decade. Since both the circuit and the circuit's state is stored in radiation-tolerant memory, both could be altered by the harsh space radiation environment. Both the circuit and the circuit's state can be protected by triple-modular redundancy (TMR), but applying TMR to FPGA user designs is often an error-prone process. Faulty application of TMR could cause the FPGA user circuit to output incorrect data. This paper will describe a three-tiered methodology for testing FPGA user designs for space-readiness. We will describe the standard approach to testing FPGA user designs using a particle accelerator, as well as two methods using fault injection and a modeling tool. While accelerator testing is the current "gold standard" for pre-launch testing, we believe the use of fault injection and modeling tools allows for easy, cheap and uniform access for discovering errors early in the design process.*

Keywords: Field programmable gate arrays, Reliability testing, Reliability estimation, Failure analysis, Space technology

1. Introduction

Field-programmable gate array (FPGA) technology, such as the Xilinx Virtex family of devices, has made inroads into space-based platforms over the past decade [1], [2]. These devices have programmable logic and routing that are used to implement user circuits and are well-suited for the digital signal processing algorithms that are often used in space. Unlike the radiation-hardened anti-fuse FPGAs that can be programmed once, the radiation-tolerant devices can programmed an unlimited number of times. The ability to *reconfigure* the device to implement new circuits makes FPGAs interesting to the space community. Unlike other hardware devices that have the circuit fabricated into the silicon, new circuits can be implemented on an FPGA while on orbit. Therefore, reconfiguration can extend the usable lifetime of the system by changing the FPGA's user circuit to meet changing mission and science goals. We have also found that reconfiguration opens up many avenues for pre-launch testing of the user circuits.

Document release number: LA-UR-08-XXXX. This work was funded by the Department of Energy through the Deployable Adaptive Processing Systems and Sensor-Oriented Processing and Networking projects, the Cibola Flight Experiment project at Los Alamos National Laboratory, NASA through the Reconfigurable Hardware in Orbit project under AIST contract #NAG5-13516, and the Air Force Research Laboratory under the FPGA Mission Assurance Center.

Unfortunately, many reconfigurable FPGAs implement logic in SRAM-based technology that is susceptible to radiation-induced faults, called single-event upsets (SEUs), that can affect the programmable logic and routing or affect the entire device. The best practices for FPGA-based spacecraft design encourages the use of triple-modular redundancy (TMR) in the user circuit to mask SEUs on the FPGA and on-line reconfiguration of the programming data, called *scrubbing*, to remove errors. Given the space constraints in this paper, we will focus on issues regarding user circuits. Not only is applying TMR an error-prone process, but sometimes the designers are unable to apply "full" TMR to the user circuit due to device size constraints. The user circuit needs to be tested pre-launch to determine whether it is working as expected, including whether output errors from SEUs can be tolerated, how to respond to output errors, and whether the availability requirements are met.

The current "gold standard" for pre-launch testing of user circuits is radiation experiments at a particle accelerator. For these tests the designer has the choice of using either a proton or a heavy ion accelerator, as these radiation sources are the most likely ionized particles to cause problems with the user circuit. Fully space-qualifying a design could take days worth of time and thousands of dollars at an accelerator. Given enough time and money, the experiments will be able to exercise all of the possible radiation-induced failure modes and find all of the problems with a user design. Since radiation-induced faults are statistical in nature, it may be too expensive to get good test coverage and difficult to understand how the errors correlate to faults in the user design. Therefore, we feel the best use of particle accelerators is as a final, pre-launch validation of the user design. Fault injection and modeling tools are much better at providing feedback about the design to the designer. We have both a fault injection tool (the single-event upset emulator) and a modeling tool (the Scalable Tool for the Analysis of Reliable Circuits) that can be used by FPGA designers to augment radiation experiments.

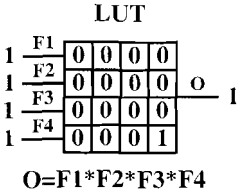
In this paper we will present a three-tiered methodology in this paper that uses all of these technologies for discovering errors in the system before launch. The rest of the paper is organized as follows. Section 3 will introduce the STARC modeling tool, which will help designers identify problems in implementing TMR in user circuits during the design stage. Section 4 will cover fault injection, which will help designers do more in depth, hardware-based testing of user

designs. Finally, Section 5 will cover the final validation of user circuits at a particle accelerator. Given the disparate nature of these three topics, the related work for these topics will be covered in the individual sections. The paper completes with a comparison of these three methodologies in Section 6. Before continuing, though, we will explain in better detail the types of design flaws we are testing for in Section 2.

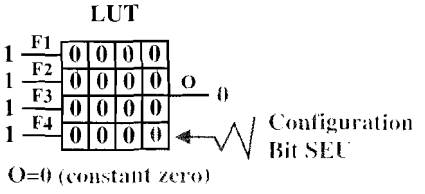
2. Background

By using all three methodologies in conjunction, designers should be able to determine how radiation-induced faults affect the system. In a partially TMR-protected systems, errors could stem from the untriplicated logic or from placement-related issues caused by how the circuit is implemented on the FPGA. In a fully TMR-protected systems, the errors should be confined to specific types of placement-related issues, as long as TMR has been successfully applied. These scenarios will be discussed below.

In user circuits where TMR has been only partially applied, there are a two areas where SEUs can affect the system: untriplicated logic and the programmable routing network. First of all, all untriplicated logic could cause output errors to manifest when the logic is corrupted with an SEU. Errors in the logic can some times be logically masked by the data the circuit is executing. For example, Figure 1(a) shows a programmable logic element, called a *lookup table* (LUT), that is implementing a 4-input AND function. If the one bit that defines the "true" condition is upset, the result is a constant-zero function. For most inputs, the output of the function would still be correct. If the data in the system never exercises the one input combination that causes the error to manifest, the error will be logically masked. Errors that manifest from untriplicated logic can only be fixed by changing the design. Therefore, the amount of these errors in the system are mostly immutable to how the user circuit is placed on the device by the design flow tools, although the location of these errors might change.



(a) Original 4-input AND Function



(b) Upset LUT Function (Constant "0")

Fig. 1. LUT Upset Example

A second set of errors in partially TMR-protected designs stem from the programmable routing network. While errors in untriplicated logic cause the functionality of the circuit to change, errors in the routing network often sever the flow of data through the circuit. For example, an error in the routing network can cause an input to a LUT to float. Unlike untriplicated logic, some of the errors in the routing network can be influenced by the design flow tools that determine how the user circuit is placed and routed on the device. Within blocks of untriplicated logic, the design flow tools will affect the the number of *sensitive bits* in the routing network that cause the design to manifest output errors. Some times the logic in a design might be completely triplicated, but some or all of the input signals might not be. In particular, triplicating clock and reset signals can cause circuits to have problems with timing and signal skew. In the older devices, sometimes it wasn't possible to triplicate these signals. In these systems the number of sensitive bits in the clock and reset trees can be directly influenced by the placement tools. While SEUs in the programmable routing along the main trunks of the clock and reset trees are very likely to affect the entire circuit, SEUs in the leaves of these trees will often be masked if only one module in the TMR-protected design is affected. To some degree, SEUs in these areas can be directly affected by placement, although complete elimination of SEUs in these areas by rerunning the placement tools is not possible. With either set of the errors being caused by the routing network, the number of sensitive bits will be affected by the design flow tools and rerunning the designs through the design flow tools should be avoided after testing has started, since the number of sensitive bits and their location will change.

In fully TMR-protected user circuits, no single-bit SEUs should cause output errors unless TMR was not applied properly. With the newer devices, though, multiple bit upsets (MBUs), where a single ionizing particle causes multiple SEUs, have become more common [3]. Under these circumstances, we have observed MBU-induced TMR defeats when the multiple errors manifest across two or more TMR modules [4]. These TMR defeats appear to be strongly influenced by placement issues. Since the design flow tools are not optimized for TMR-protected circuits, it is possible that all three modules can by proximally located on the device. As MBUs cause clusters of tightly-located faults on the device, the TMR defeats tend to occur in regions where the modules neighbor each other. Therefore, faults of this type are considered a placement-related issue.

Good testing should help designers ascertain errors that have been caused by untriplGiven the disparate nature of these three topics, the related work for these topics will be covered in the individual sections. icated logic, placement-related issues, and the application of TMR. With our three approaches to testing, we have varying degrees of coverage of these problems. The modeling tool, STARC, can be used to ascertain problems with the application of TMR and identifying untriplicated logic, but currently does not address placement-related issues. Since the modeling tool works at the design level, tying reliability problems back to the design is trivial. The fault injection tool, the SEU Emulator, and accelerator testing can ascertain problems in all three areas, but tying the errors back to the design is a laborious, manual process. As stated above,

fault attribution with accelerator testing is much harder than with fault injection. Since all three testing styles have their strengths and weaknesses, using them in conjunction is the best approach.

3. Modeling Tools

Reliability analysis is traditionally done with modeling tools. For designers of many types of systems, these tools allow the designers to focus on creating accurate models of their systems, instead of focusing on how to calculate the reliability. FPGA user circuit designs, though, already have an accurate model of the circuit — the hardware description of the circuit. Therefore, with the right modeling tool, the reliability analysis of the FPGA user circuit can be done during the design phase when fixing design flaws in the user circuit is relatively cheaper.

Traditionally, circuit reliability has been determined using purely analytical approaches [5] or techniques that model Boolean networks as probabilistic systems [6]–[9]. These modeling techniques represent circuits as probabilistic transfer matrices, stochastic Petri-nets, Markov chains or Bayesian networks. The combinatorics-based analytical approaches have been found to be error-prone and computationally complex for the analysis of large designs. Similarly, a number of limitations have been identified for many modeling-based approaches. First of all, model and input data set creation greatly increase the time commitment of using these tools. Transforming circuits into intermediate probabilistic system models is an additional, computationally complex task. Calculating the circuit reliability also grows exponentially with circuit size and the number of input vector sets and the computation can take prohibitively long to finish. The exception to these problems is the SETRA tool [10] that directly addresses the state space issues as well as automated model generation.

For these reasons, the traditional tools are not well-suited for the size of designs used in most FPGA systems. All of these limitations have led to the development of the STARC tool, which specifically addresses the limitations of model creation, input data sets and computation complexity with these solutions:

- Uses the industry-standard Electronic Design Interchange Format (EDIF) representation of a circuit as the input model,
- Does not use input vector sets,
- Uses memoization to reduce the computational complexity, and
- Uses combinatorial reliability calculations.

By using the EDIF circuit representation, the designer can assess the reliability of a circuit during the design process, even if the design is not complete, the design doesn't work, or the hardware hasn't even been bought. Without the use of input vector sets reliability is determined through the probability of device or input failure and is not dependent on specific input data sets. Since the calculation is not dependent on the input data set, the reliability of sub-circuits are determined by type, such as a two-bit adder, and memoized for reuse. In this manner, large-scale circuits are analyzed in a fraction of the time required by traditional approaches, making design exploration more worthwhile.

There are a few disadvantages to this approach. First, since EDIF does not contain information about the routing and the placement on the device, routing is currently being statistically estimated from case studies of routing placement. Furthermore, currently there is no way to assess placement-related issues, such as MBU-induced TMR defeats. We are currently working on a solution for this limitation for designs that have gone all the way through the design flow tools. Second, without input vector sets logic masking cannot be taken into account, and STARC estimates the worst case failure rate. While this value may be lower than the value determined by other tools [11], STARC provides a useful lower bound on the circuit's reliability. Furthermore, the scalability gained from not using input vector sets would most likely be compromised if input vector sets are used.

One of the advantages of using the EDIF circuit representation is that hierarchy in the circuit should be preserved, as long as the hierarchy is not removed during the synthesis process. Since designers tend to create complex circuits by creating less complex sub-circuits, maintaining this structure can be very useful. There are many advantages of calculating the reliability hierarchically. In particular, STARC can readily exploit memoization if there is a high degree of sub-circuit reuse. By exploiting sub-circuit reuse the state space and the computation grows polynomially instead of exponentially. This hierarchical nature allows circuits to be examined at the highest level of abstraction or the most minute level of detail. STARC automatically determines the appropriate level of the hierarchy that needs to be explored.

During hierarchical exploration, dependency graphs for each primary output at each level of the hierarchy is determined. The dependency graph has all of the sub-circuits between the output and the reachable inputs. Since not all logic or inputs are reachable from every output, this technique removes unrelated logic from the reliability calculation. Once the dependency graph for an output is determined, the reliability can be calculated. In unmitigated designs, the cross-section is the total area of the dependency graph: $A(O) = (\sum_{i=0}^{m} A(C_i))$, where $A(X)$ is the sensitive area of X (where X is either a wire or a cell) and $C = \{C_0, ..., C_m\}$ is the set of cells that can be reached from output wire O. The reliability of basic architectural elements, such as LUTs and user flip-flops, are pre-determined and are statically loaded when STARC starts.

STARC was designed to help designers find problems in TMR-protected circuits by detecting imbalances between the modules and by finding all unprotected logic. For mitigated circuits, the sensitive area is confined to the part of the design that is not triplicated, as triplication will mask errors. As long as there is one voter for each redundant module, the sensitive area should be zero. There are cases where the design flow tools, in particular synthesis tools, will alter the circuit so that the TMR modules are no longer functionally equivalent. If the modules lack functional equivalence with each other, often times two of the modules are sharing a partial calculation with the third module. While the TMR-protected circuit in this case will be functionally equivalent to the unmitigated circuit, the

shared partial calculation is untriplicated and can cause single points of failure. Feedback loops in TMR-protected systems are also sensitive to *persistent errors* [12], if the feedback loops are not triplicated and the feedback signals cut with voters. If the feedback loops are not handled in this manner, the design could be protected by technically correct TMR, but errors in the feedback loop's state will not be able to autonomously resynchronize after the SEU is removed. While the first SEU in the feedback loop will be masked, another SEu in the feedback loop is not guaranteed to be masked. Finally, we have found that how the design flow tools implement logical constants, such as the zeroth bit of a carry chain adder or unused inputs that have been tied off, can cause single points of failures in TMR-protected systems.

In all of these cases, STARC provides warnings and information about the design to designer. The output of the tool provides the designer a list of sub-circuits that are untriplicated, and warnings about potential single points of failures from functionally unequivalent modules and logical constants. Since 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.

4. Fault Injection Testing

Once a design is completed and hardware has been bought, it is possible to move onto fault injection. Unlike modeling tools, fault injection works with the hardware that the user circuit is meant to be implemented on. Therefore, the placement-related issues can be assessed through fault injection. Because the hardware is being used, this type of analysis has much better fidelity to accelerator testing and on-orbit behavior, if done right. Finally, since input vectors are used with the design, any logical masking and placement-related issues can be discovered.

Since the interfaces that control configuration of the device 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. This process is called *fault injection*. While LANL designed one of the first fault injection testbeds for FPGAs with the SLAAC1-V SEU Emulator [13], since then many other organizations have created them [14]–[16] as a testament to their usefulness in preparing and measuring a design's space-readiness. We have also gone on to make other versions of our fault injection tool to support newer hardware devices and fault injection tools that injection MBUs.

Fault injection tools for FPGAs have the same basic algorithm, as shown in Figure 2. With this algorithm, faults can be injected throughout the entire programming data, except user flip-flops. As user flip-flops make up on only a very small fraction of the programming data and only affect the circuit state, not injecting faults into the user flip-flops does not adversely affect the fault injection accuracy. It is important to run a number of input vectors through the system after the fault is injected to avoid logical masking. Good test coverage of input vectors is important, as running a complete set of test vectors is often infeasible due to time constraints as test vectors increase exponentially with input data width. It is always possible to run complete set of test vectors on a limited set of locations in subsequent fault injection tests. It is also important that the circuit is reset and resynchronized after the fault is removed. If not, then it is possible that the combination of the two error states causes an output error, causing false-positive errors to be reported.

There are usually two types of fault injection systems based on whether one or two FPGAs are used. In our SEU Emulator tool two FPGAs are used. Faults are injected into the *design under test* (DUT) FPGA and then run in lockstep with the same input vectors with the *golden* FPGA. The advantage of this system is that sharing input vectors, detecting output errors, and testing the system for resynchronization is very easy, but the timing of a lockstep system is difficult to design. In the one FPGA fault injection systems, the input vectors are run through the system twice: once without fault injection and once with fault injection. The advantage of this system is that it takes less hardware and is easier to design than a lockstep system, but the disadvantage is that the input vectors and correct output vectors need to be saved in the system. Furthermore, the output from the fault injection needs to be compared to the expected output results. While it is possible to do the comparison in software, a lockstep system can do it on the golden FPGA and report only the miscompares to the software.

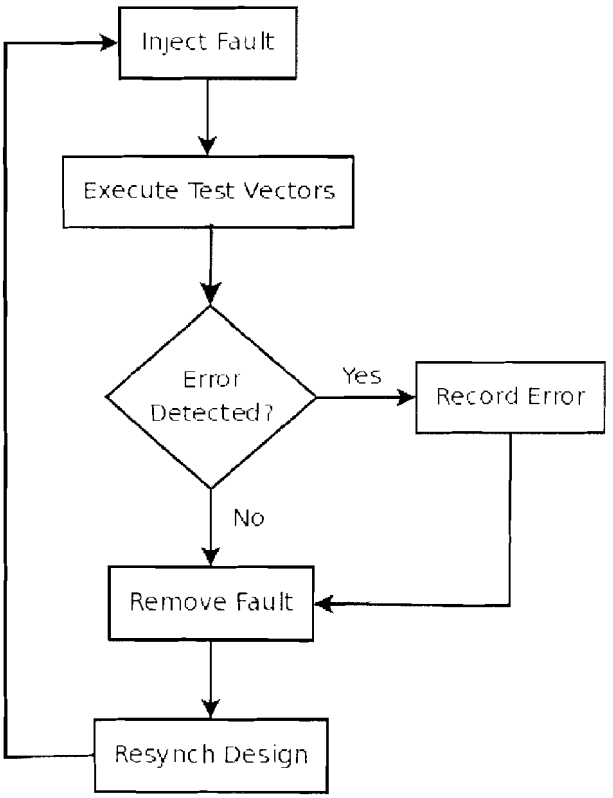


Fig. 2. Basic Fault Injection Algorithm [13]

In general, there are several aspects to good fault injection systems, such as the ability to handle different types of user circuits. With many fault injection systems, often times the

number of clock and reset pins, the width of input and output buses, and the pin locations are often set. Due to these restrictions, some times the design has to be changed to fit the fault injection system. These types of changes need to be minimized so the assessment of placement-related errors accurately reflects the user circuit the spacecraft will deploy. On occasion, we have found some systems do not lend themselves to fault injection. In these systems, the use of modeling tools will be even more important.

Unlike STARC, tying design problems found through fault injection to the design can be quite difficult and time consuming. While the fault injection tool will give a location for the fault, most designers do not know how to translate that location into a physical location on the device. Once the physical location is determined, it is possible to go into a design flow tool, called FPGA Editor, to determine what part of the user circuit is in that location. Since the design flow tools do not mangle the naming convention within a user circuit design, it is then possible to tie the physical location to part of the design. There are times, though, that even knowing the part of the design that is causing the problem does not help. Since many errors manifest in the routing network of TMR-protected designs, it's possible the fault is caused by a signal that is passing through a neighboring routing switch. In these cases, unless the fault injection tool finds a number of similar errors, it can be very difficult to determine exactly what is failing. Because of these problems, some times it is easier to disambiguate fault injection results using the STARC results. At least in this manner, designers can confirm specific results, such as problems with untriplicated logic. Finally, if fault injection is only reporting a handful of errors, the designers can decide that the user circuit meets the availability requirements for the system and that further design exploration to fix design flaws is unnecessary.

5. Accelerator Testing

Related work.

One of the advantages of doing accelerator testing after the use of fault injection and modeling tools is that the designers should be better prepared for the accelerator testing. To this end, the designers should know the areas of the circuit design that should cause output errors from the modeling tools, and know the locations of these faulty areas on the device through fault injection. Furthermore, if the designer has been using a lockstep fault injection tool, the fault injection hardware can be used as the test fixture for the accelerator testing. Even the lockstep fault injection software can be used with minor modifications as part of the accelerator test fixture. If a lockstep system was not used for fault injection, a test fixture that can easily determine miscompares on the fly might be needed, as turning the beam on and off frequently can waste time and money.

The algorithm for the software aspect of the test fixture is very similar to the fault injection tool's algorithm. Instead of injecting faults artificially, though, the particle accelerator will be injecting the faults. Unlike fault injection, controlling the number of upsets that occur during one loop of the algorithm is more difficult. Furthermore, error removal and single-event functional interrupts (SEFIs) that affect the functionality of the entire device complicate the situation. These three problems will be discussed below.

The arrival time of radiation-induced faults are a Poisson random processes. As the designer will want to reduce the probability of multiple independent upsets (MIUs) causing an output error, the beam's flux is tuned so that on average only one upset occurs per algorithm loop. Even still, Poisson statistics tell us that, if the beam's flux is tuned to one upset per algorithm loop, there is a 37% chance that no upsets occur, a 37% chance that one upset occurs, and a 26% chance that two or more upsets occur during the given time period. Since each time the algorithm iterates without an upset or with multiple upsets wastes time and money, tuning the beam's flux properly is very important. Further complicating this issue is detecting the location where the upset occurred. In the larger devices, it take can take appreciably longer to determine where the upset occurred on the device, which slows down each iteration of the algorithm's loop. Therefore, a smaller device could possibly be tested at a much higher flux than a larger device, speeding up the qualification process.

Going hand in hand with detecting the location of the SEU is removing the SEU and allowing the circuit to resynchronize. There are two ways to remove an SEU during an accelerator test. One method is to do a complete reconfiguration of the device and the other is to use a scrubbing circuit that fixes SEUs through on-line reconfiguration that only partially reconfigures the device. The advantage of using the scrubbing circuit is that the SEU's location can be determine at the same time, speeding up the time for each iteration of the algorithm's loop. Usually access to both methods is necessary, though. Often times the only way to recover from a SEFI is a full reconfiguration of the device. SEFIs are usually detected when an unreasonably high number of SEUs are found at one time by the scrubbing circuit. Therefore, an effective design for a scrubbing circuit is to have a threshold limit for SEUs per cycle. Once the number of SEUs is above that threshold, the scrubbing circuit does a complete reconfiguration of the device. In either case, the circuit needs to enough time to resynchronize with the golden FPGA before the next SEU occurs to reduce the probability of false-positive fault attribution from MIUs.

After the accelerator test is completed, the results need to be examined so that correlations between output errors and SEU locations can be determined. Since the SEUs in accelerator testing do not present themselves in the system uniformly or at specified time intervals, correlating output errors to specific SEU locations can be a challenge. In some cases, if there is a long latency in the user circuit, an output error might be correlated with an SEU that happened several iterations before the output error manifested. In otherGiven the disparate nature of these three topics, the related work for these topics will be covered in the individual sections. cases, sometimes the software reports the output error before the SEU location is determined and the problem location is after the output error in the results log. Often times all of the results around a SEFI event will need to be tossed, since removing the SEFI is time

consuming and the system will likely report output errors for several iterations until the circuit state resynchronizes.

As long as the user circuit that is being tested is the same one tested in fault injection, the results from fault injection can be used to disambiguate the accelerator test results. Due to the problems described with attributing SEUs to output errors, the most effective approach for analyzing accelerator results is to look at several SEU locations before and after the output error in the log. This "window" of SEU locations can then be compared to fault injection results to determine if any of these SEUs occurred in fault injection. While this method can usually help a designer correlate output errors with fault injection results, some output errors cannot be completely correlated. In these cases, sometimes the accumulation of errors in the circuit state caused by multiple errors caused the output error. For these cases, sometimes part of an accelerator test can be "played back" using the fault injection tool, where the tool uses the accelerator log to inject faults in specific locations in a particular order. In this way, the designer can determine whether the output error can be explained and whether further design exploration is needed to address potential design flaws that caused the output error.

For fully or nearly-fully mitigated designs, accelerator testing should be uneventful and the user circuit should be able to operate for minutes or longer without any output errors. For example, if fault injection only found 100 sensitive bits in a device with 75 million bits, there is only a 0.000133% chance that an output error will manifest for any given SEU. With such a low probability of occurrence, the designer could wait hours for an output error to occur and SEFIs might be a more common cause of output errors. Some designers will do tests with different flux levels and different durations. In particular, one test might be very low flux over several hours, mimicking average operation on orbit, and another test might have a very high flux over a couple of minutes, mimicking solar flare conditions. If at the end of these tests, the design is able to operate either error-free or within the availability requirements, the design is considered space ready.

On the other hand, if the error rate is much higher than indicated by the fault injection tool, either the flux could be too high or there might be problems with either the fault injection or accelerator test fixture. When designing new fault injection and accelerator test fixtures it is important to test the setup by correlating the results. If the results cannot be correlated, then the methodologies for both systems need to be examined.

6. Results

In this section, we will compare the use of these three methodologies on a circuit. This circuit, an adder tree, is fully triplicated and was designed originally to test for placement-related issues from both MBUs and logical constants. This design was implemented for a Xilinx Virtex-II part (XC2V1000). All three methodologies were used on this design. In the following paragraphs, we will describe the amount of time, the quality of the results, and the cost of using these methodologies.

In terms of time, STARC is comparitively 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 bits in the design could cause output errors. In terms of cost, STARC is free to government users.

In terms of test coverage, the SEU Emulator was much more complete than the other two methods. With fault injection, we were able to find 285 single-bit SEU locations 18,733 2-bit SEU locations, 11,264 3-bit SEU locations, and 19,464 4-bit SEU locations that cause the design to output bad data. Each pass through the fault injection test takes two hours per run and each MBU test is a separate run through the test. As the MBU tests are run with specific MBU shapes based on our analysis of how MBUs affect the Virtex-II, we were able to constrain the MBU tests to the six most common shapes. In all, fault injection tests took 14 hours for seven tests. In terms of cost, the fault injection hardware is about $6,000 and the software is free to government users.

As validation for both of these tests, we did a two hour long test at at the Univerity of Indiana proton accelerator. During this test we were able to observe 31 output errors, of which 42% we were able to later correlate to known fault injection error locations. At one upset a second, we would have been able to test all of the single bit errors in no less than four hours of testing, assuming that no single-bit fault location was exercised multiple times. Since the MBU-related issues have only a 2% chance of occuring in proton, completing the test would be prohibitively expensive. In terms of cost, we were able to use the hardware and software from fault injection and only had to pay the accelerator fees of $1,200 and $500 for the FPGA. Had we completed the single-bit test, we would have to pay for four to eight hours of testing and two to four FPGAs for a total of $3,400-6,800.

While the initial cost of the hardware for the fault injection tool is the highest of the three test methodologies, the cost is amortized across across all of the fault injection tests and the accelerator testing. Since the hardware infrastructure can be reused an unlimited number of times, if the FPGA is not irradiated, the cost is reasonable. When the test coverage is factored in, the amount of time and cost invested in the fault injection tool is the best option. While fault injection should never replace accelerator testing, the accelerator was shortened when we were able to confirm that our fault injection results.

7. Conclusions

In this paper we presented a three-tired methodology that finds design flaws in FPGA user circuits and locates the faulty locations on the FPGA. One methodology used a circuit representation to find design flaws through modeling. The second methodology used fault injection to locate how the design flaws translated to physical locations on the FPGA. The final method was an acclerator test to validate the previous results. We were also able to show how these three methodologies compared in terms of test coverage, time, and cost. While

the modeling tool was the fastest, fault injection was the best methodology in terms of cost and test coverage.

References

[1] E. Fuller, M. Caffrey, P. Blain, C. Carmichael, N. Khalsa, and A. Salazar, "Radiation test results of the Virtex FPGA and ZBT SRAM for space based reconfigurable computing," in *Proceeding of the Military and Aerospace Programmable Logic Devices International Conference(MAPLD)*, Laurel, MD, September 1999.

[2] M. Caffrey, M. Echave, C. Fite, T. Nelson, A. Salazar, and S. Storms, "A space-based reconfigurable radio," in *Proceedings of the 5th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, September 2002, p. A2.

[3] H. Quinn, P. Graham, J. Krone, M. Caffrey, and S. Rezgui, "Radiation-induced multi-bit upsets in SRAM-based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2455 – 2461, December 2005.

[4] 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 Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2037 – 43, 2007.

[5] J. A. Abraham and D. P. Siewiorek, "An algorithm for the accurate reliability evaluation of triple modular redundancy networks," *IEEE Transactions on Computers*, vol. 23, no. 7, pp. 682–692, July 1974.

[6] S. Krishnaswamy, G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Accurate reliability evaluation and enhancement via probabilistic transfer matrices," in *Design, Automation and Test in Europe (DATE'05)*, vol. 1. New York, NY, USA: ACM Press, 2005, pp. 282–287.

[7] C. Hirel, R. Sahner, X. Zang, and K. Trivedi, "Reliability and performability using SHARPE 2000," in 11^{th} *Int'l Conf. on Computer Performance Evaluation: Modeling Techniques and Tools*, vol. 1786, 2000, pp. 345–349.

[8] G. Norman, D. Parker, M. Kwiatkowska, and S. Shukla, "Evaluating the reliability of NAND multiplexing with PRISM," *IEEE Transactions on CAD*, vol. 24, no. 10, pp. 1629–1637, 2005.

[9] F. V. Jensen, *Bayesian Networks and Decision Graphs*. New York: Springer-Verlag, 2001.

[10] D. Bhaduri, S. K. Shukla, P. S. Graham, and M. B. Gokhale, "Reliability analysis of large circuits using scalable techniques and tools," *IEEE Transactions on Circuits and Systems - I: Fundamental Theory and Applications*, vol. 54, no. 11, pp. 2447 – 60, NOV 2007.

[11] D. Bhaduri and S. Shukla, "NANOLAB—a tool for evaluating reliability of defect-tolerant nanoarchitectures," *IEEE Transactions on Nanotechnology*, vol. 4, no. 4, pp. 381–394, 2005.

[12] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin, "SEU-induced persistent error propagation in FPGAs," *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2438 – 45, 2005.

[13] E. Johnson, M. Caffrey, P. Graham, N. Rollins, and M. Wirthlin, "Accelerator validation of an FPGA SEU simulator," *IEEE Transactions on Nuclear Science*, vol. 50, no. 6, pp. 2147–2157, December 2003.

[14] M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, S. Pastore, G. Sechi, and R. Weigand, "Evaluation of single event upset mitigation schemes for sram based fpgas using the flipper fault injection platform," in *Proc. of the 22th IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems (DFT07)*, September 2007, pp. 105–113.

[15] 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 *Single-event effects symposium*, 2008.

[16] G. Swift, C. W. Tseng, G. Miller, G. Allen, and H. Quinn, "The use of fault injection to simulate upsets in reconfigurable FPGAs," in *Submitted to the military and aerespace programmable logic devices conference (MAPLD08)*, 2008.