Fine-Grain SEU Mitigation for FPGAs Using Partial TMR

Brian Pratt, *Member, IEEE*, Michael Caffrey, James F. Carroll, *Student Member, IEEE*, Paul Graham, Keith Morgan, and Michael Wirthlin, *Senior Member, IEEE*

***Abstract*—The mitigation of single-event upsets (SEUs) in field-programmable gate arrays (FPGAs) is an increasingly important subject as FPGAs are used in radiation environments such as space. Triple modular redundancy (TMR) is the most frequently used SEU mitigation technique but is very expensive in terms of area and power costs. These costs can be reduced by sacrificing some reliability and applying TMR to only part of the FPGA design. Our partial TMR method focuses on the most critical sections of the design and increases reliability by applying TMR to continuous sections of the circuit. We introduce an automated software tool that uses the Partial TMR method to apply TMR incrementally at a very fine level until the available resources are utilized. Thus the tool aims to gives the maximum reliability gain for the specified area cost.**

***Index Terms*—Aerospace industry, fault injection, fault tolerance, field programmable gate arrays (FPGAs), proton accelerator, radiation effects, reliability, single-event upset (SEU), triple modular redundancy (TMR).**

I. Introduction

TRIPLE modular redundancy (TMR) is a widely-used fault tolerance method for protecting field-programmable gate array (FPGA) designs against single-event upsets (SEUs) caused by radiation. SRAM-based FPGAs are especially susceptible to these effects due to their reliance on SRAM memory cells to hold the configuration of the device. TMR is popular because of its straightforward implementation and reliable results. TMR has been shown to greatly improve the reliability of FPGA designs subject to SEUs [1], [2].

Although TMR can significantly improve the reliability of a design, it is expensive in terms of circuit area, power, and speed due to the extra logic required for implementation. Since TMR involves the triplication of a circuit and the addition of majority voters, the overhead area cost of TMR exceeds 200% of the original design size. In some cases, the actual area cost of TMR on an FPGA has been shown to be far greater [1], [3].

Due to resource constraints and/or system constraints, TMR of an entire user FPGA design (full TMR) is not always feasible. As commercial off-the-shelf (COTS) parts, FPGAs have a hard limit of configurable resources available on-chip. Due to the overhead of full TMR, unmitigated designs which utilize more than one-third of the FPGA cannot be protected with full TMR. In addition, a user FPGA design may have an artificial resource usage limit due to area, power or other system design constraints. Again, in these situations full TMR is not a viable option.

In situations in which full TMR is not possible, *partial* triplication of the FPGA design may be the next best alternative. Mitigation of part of the design can increase the overall reliability of the design at a lower cost than full TMR. Partial mitigation methods cannot provide the same level of reliability as full mitigation approaches and therefore must focus on the components that will most increase the reliability of the design.

We have previously introduced a method to apply TMR to selective sections of an FPGA design and designed an associated automated software tool which implements this technique [4]. This paper describes the methodology behind this type of mitigation approach along with a demonstration of its effectiveness. An automated software tool we have developed chooses the most critical sections of the design for mitigation and applies TMR incrementally at the logic (LUT) level. The tool focuses on contiguous sections of logic to triplicate in order to reduce the overhead of partial TMR and increase reliability. The automated nature of the tool allows it to work at a level too low for a developer to do by hand, and the fine-grain addition of mitigation logic allows the tool to gain the highest reliability possible by fully utilizing the available resources.

II. Partial Triple Modular Redundancy

In some cases the high cost of full TMR may preclude it from being a mitigation option. In these situations it may be beneficial to apply TMR to only *part* of a target design. The goal of using partial mitigation is to relax the amount of mitigation applied as compared to full TMR to reduce the area overhead cost with a minimal loss in reliability.

Several methods have been proposed for selecting the most critical circuit structures, thus trading hardware cost for SEU immunity. Samudrala *et al.* proposed the *selective triple modular redundancy* (STMR) method which uses signal probabilities to find the SEU-sensitive sub-circuits of a design [5]. Chandrasekhar *et al.* proposed a modification to this method which operates on look-up tables (LUTs) rather than logic gates [6]. We have proposed a partial mitigation method based on the concept of *persistence* [7] called *Partial TMR* [4].

A. Prioritizing Mitigation

Any partial mitigation technique using TMR is based on the idea that a subset of a design's components will be protected

Manuscript received September 7, 2007; revised February 20, 2008. Current version published September 19, 2008. This work was supported by the Department of Energy at Los Alamos National Laboratory. Approved for public release under LA-UR-08-0680; distribution is unlimited.

B. Pratt, J. F. Carroll, and M. Wirthlin are with the Department of Electrical and Computer Engineering, Brigham Young University, Provo, UT 84604 USA.

M. Caffrey, P. Graham, and K. Morgan are with the Los Alamos National Laboratory, Los Alamos, NM 87544 USA.

Digital Object Identifier 10.1109/TNS.2008.2000852

Authorized licensed use limited to: Univ of Calif Los Angeles. Downloaded on January 19, 2009 at 13:40 from IEEE Xplore. Restrictions apply.

with TMR by triplicating them and adding voters when necessary. This subset must be carefully selected so that the resulting partially mitigated design is as reliable as possible.

Our Partial TMR method uses the concept of persistence, defined in detail in [7], as a first level of prioritization. A *persistent* error is caused by an SEU which corrupts the internal state of the circuit. While *non-persistent* errors are corrected simply by repairing the FPGA configuration (i.e., re-loading the original contents of the FPGA programming memory), persistent errors remain even after the configuration is repaired. Partial TMR gives priority to the circuit components which are susceptible to persistent errors and applies TMR to them. Section III will explain the concept of persistence in more detail.

B. *Voter Placement and Continuous Triplication*

The number and placement of voters is another important consideration when applying partial TMR. For traditional full TMR, voters are needed only in feedback sections of the circuit. These voters are needed in order to correct any errors introduced in one of the three TMR replicates in that feedback section. Three voters are placed in each feedback loop in order to restore the state for all three TMR branches, even if one temporarily has an erroneous value [2].

Partial TMR implies that some of the components in a design are to be triplicated and others are not. When a triplicated component is meant to drive a non-triplicated component, the triplicated output of the first component must be reduced to a single output. When a non-triplicated component drives a triplicated one, three inputs are needed from the single driver.

In the case of a triplicated component driving a non-triplicated component, three outputs can be combined into one by a majority voter, thus producing a single output. In the case of a non-triplicated component driving a triplicated one, a single output wire can be routed to drive three different inputs. In both cases, however, the sensitive cross section of the design is increased. Since both the logic and the routing of an FPGA are sensitive to SEUs, the voters inserted and the added routing both add to the SEU-sensitive cross-section of the design.

Fig. 1 illustrates the difference between choosing adjacent components and non-adjacent components to triplicate. Circuit components are numbered and voters are labeled with the letter "V." Fig. 1(a) represents the unmitigated circuit while Fig. 1(b) and (c) represents two options when triplicating two circuit components. The area of the circuit which is sensitive to SEUs is shaded in the figure. Notice that in Fig. 1(c), there are more circuit components (including voters and routing) that are sensitive to SEUs than in Fig. 1(b).

Due to this added cross section, it is important to limit the transitions from triplicated to non-triplicated logic and vice versa. The addition of hardware to split one domain into three as well as combine three TMR branches into one can cause more chip resources to be used and a larger overall design cross-section sensitive to SEUs. More reliability gains can be achieved by concentrating the triplicated sections to avoid these situations. Any method that triplicates only part of a design should choose contiguous regions for TMR application.

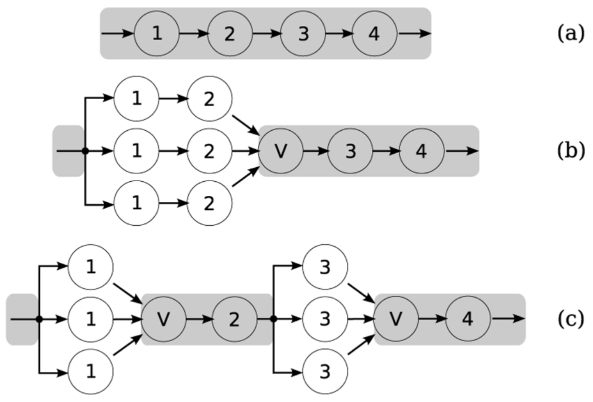

Fig. 1. Circuit flow diagrams illustrating voter placement options. The SEU-sensitive portions of the circuit are highlighted. (a) The unmitigated circuit. (b) Two contiguous modules triplicated. (c) Two non-contiguous modules triplicated.

III. PERSISTENCE

As mentioned in Section II-A, our Partial TMR method uses the concept of *persistence* as a method of prioritizing the application of TMR. We have previously proposed that an FPGA design can be divided into two sections: a persistent section and a non-persistent section [7]. When any SRAM configuration bit is upset, it remains in error until it is corrected. Although upsets in the configuration memory can be repaired, an upset in the persistent section of the design may leave errors in the state of the circuit even after the configuration is corrected. Upsets in the non-persistent section will not produce these persistent errors. This section will explain the difference between these two circuit sections and how this affects the application of partial TMR.

A. *Configuration Scrubbing*

To effectively protect an FPGA design from SEUs, mitigation schemes such as TMR must be accompanied with configuration scrubbing. Since TMR can only function if at least two of the three replicates of the circuit are functioning, any malfunctioning replicates should be repaired as soon as possible. If upsets in the configuration of the FPGA are allowed to build up, even a circuit with full TMR in place will eventually fail.

Configuration scrubbing, or simply scrubbing, is a periodic refresh of the FPGA configuration memory. A sufficiently fast scrubbing rate can insure that only a single fault exists in the configuration at one time, allowing TMR to properly mitigate functional errors.

Scrubbing coupled with TMR is not bullet-proof, however. The random nature of upset occurrence means that no scrubbing rate can guarantee that two faults will not occur within the same scrubbing period. Also, TMR has been shown to fail in the presence of a single upset in some instances [8]. It is also feasible that a single multi-bit upset (MBU) could affect more than one triplicated module and cause TMR to fail.

It is important to understand that scrubbing does not prevent errors from occurring in the circuit. It can only restore the contents of the configuration memory of the FPGA. An unmitigated circuit may operate incorrectly during the time between the upset and the repair of the configuration. The way the circuit

Authorized licensed use limited to: Univ of Calif Los Angeles. Downloaded on January 19, 2009 at 13:40 from IEEE Xplore. Restrictions apply.

behaves after the configuration is repaired determines whether the affected component is labeled persistent or non-persistent.

B. Persistent Circuit Structures

Each of the components in a circuit can be categorized as persistent or non-persistent. When an SEU affects the persistent components of a design, the resulting fault may trap the circuit in an undesired state that can only be corrected by resetting the FPGA system. System resets can dramatically affect system availability due to the time required for a full restart and should be avoided if possible. Non-persistent components do not have the same effect. An upset affecting a non-persistent component does not require a system reset to recover.

The circuit structures that correspond to persistent configuration bits can be identified through static circuit analysis. Specifically, circuit primitives that affect feedback structures within the design are those which cause the persistent error behavior. If a circuit fault propagates an error into a feedback structure, the incorrect values produced by the faulty circuit alter the feedback state. Once the state of the feedback section has been corrupted, the circuit may not behave correctly until the circuit state has been reinitialized with a global reset. Although configuration scrubbing will repair the faulty circuit, it will not restore the proper circuit state.

The sample schematics in Fig. 2 illustrate the major subsections of a simple circuit that may cause persistent configuration faults. Fig. 2(a) highlights a feedback structure in a sample design. A fault within this feedback structure will generate and/or indefinitely propagate incorrect values within the state registers. Fig. 2(b) highlights the logic which feeds *into* feedback structures. Upsets within this section may also cause persistent configuration faults since computations made in these input-to-feedback structures contribute to the state values within the feedback section of the circuit. For example, a metastability filter that feeds into a state machine falls into this input-to-feedback category and is part of the persistent section of the design. Fig. 2(c) highlights logic structures driven by the feedback section or that operate independently of feedback. These sections make up the non-persistent section of the circuit and upsets within these sections do not cause persistent faults.

Dividing the circuit into these three areas guides the choice of which components of the design to triplicate under area constraints. Our Partial TMR approach mandates that the feedback section of the design be given priority when choosing which design components to triplicate. This section will give the highest gains in reliability against persistent errors when mitigated. The second priority is the input-to-feedback section since it also contributes to the persistent cross-section of the design. The feed-forward section is the last priority since it consists only of non-persistent logic.

IV. BYU-LANL TMR (BLTmr) Tool

The Brigham Young University-Los Alamos National Laboratory TMR (BYU-LANL TMR or BLTmr) tool performs triple modular redundancy on an FPGA design automatically. It uses the methods developed for Partial TMR to provide the maximum gain in reliability for a given cost in terms of FPGA resources. The tool gives the designer control over how much mitigation to apply and automatically selects the components to mitigate in order to achieve the highest reliability.

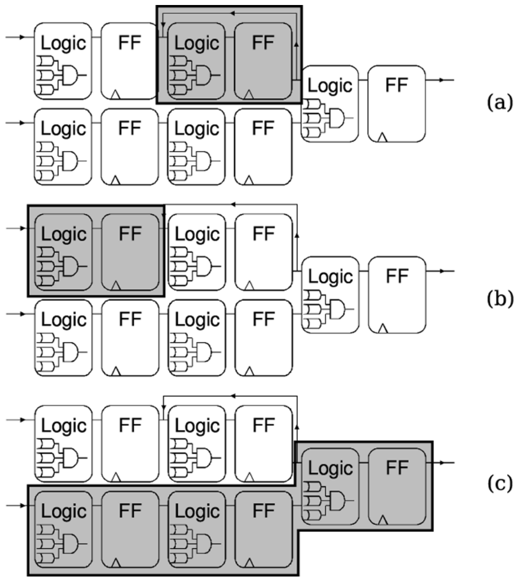

Fig. 2. Simple representation of a circuit with different sections highlighted. (a) The feedback section. (b) The input to the feedback section. (c) The feed-forward logic section.

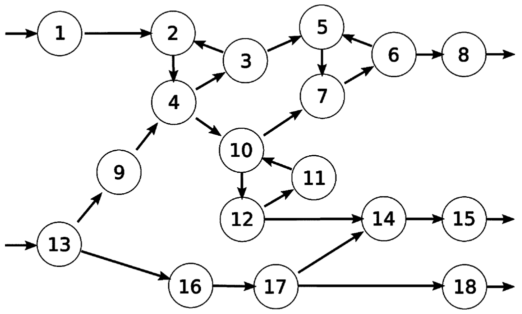

Fig. 3. Circuit flow diagram for a simple sample circuit.

Applying the Partial TMR method, the BLTmr tool focuses on the persistent components of a design. In order to divide the design into persistent and non-persistent components, the connectivity of the circuit is analyzed. The design is separated into the different sections described in Section III-B. As an example, see the circuit flow diagram in Fig. 3. This sample circuit contains three feedback loops, comprised of nodes 2–4, 5–7, and 10–12. These nodes are classified as the feedback section of the circuit. Nodes 1, 9, and 13 all feed into the feedback section of the circuit and together make up the input-to-feedback section. These two sections make up the persistent section of the circuit. The rest of the nodes do not contribute to the feedback state of the circuit and are thus part of the feed-forward or non-persistent section. This sample circuit flow diagram will be used to further explain how the BLTmr tool works.

A. Application of Partial TMR

The BLTmr tool uses the methods suggested in Section II in attempt to obtain the maximum reliability possible for the re-

Authorized licensed use limited to: Univ of Calif Los Angeles. Downloaded on January 19, 2009 at 13:40 from IEEE Xplore. Restrictions apply.

sulting mitigated design. The top priority of the tool is to focus on the persistent section of the design. This reduces the number of system resets needed, increasing the availability of the design. The second priority of the tool is to apply TMR to continuous sections of logic to avoid unnecessary transitions between triplicated and non-triplicated logic. This prevents the addition of unnecessary voters and routing which reduces the SEU-sensitive cross-section of the design.

As explained in Section III-B the primary focus of Partial TMR is the feedback section of a design. These portions of the circuit should be the first to be mitigated. When only a portion of the feedback section of a design can be triplicated, however, it is important to focus on the part that has the greatest effect on the rest of the feedback section. The BLTmr tool gives the highest priority to the components which come first in a topological ordering of the feedback section. These components have the potential to cause the most failures since they can affect more of the circuit than those further down in the topology. In the case of the sample circuit, the feedback loop comprising nodes 2–4 are the highest priority followed by the feedback loop made up of nodes 10–12 and finally nodes 5–7.

Once the feedback section of the circuit has been triplicated, the input-to-feedback section is the next priority. In this section, however, the topologically-first components are not the highest priority. Section II-B explained that transitions between triplicated and non-triplicated components are undesirable. The BLTmr tool therefore begins with the components which feed directly into components in the feedback section of the design which have already been triplicated. Triplication should then continue *backwards* topologically until reaching the primary design inputs. This reduces the number of transitions between triplicated and non-triplicated logic and more effectively increases the reliability of the design. In the sample circuit, nodes 1 and 9 have priority over node 13 since they are topologically closer to the feedback section.

After the feedback and the input-to-feedback sections have been triplicated, the persistent cross-section of the design should be eliminated. Only the non-persistent, or feed-forward, logic remains. Since contiguous sections of logic should be triplicated as much as possible, the BLTmr tool breaks up the feed-forward section further.

The third section that the BLTmr tool focuses on is the portion of the feed-forward section of the circuit which is driven by the feedback section of the design. Motivated by the same reasoning as in the input-to-feedback section, triplication starts with the direct outputs of the feedback section and proceed *forward* topologically until reaching the primary outputs of the circuit. Again, it is important to preserve the continuity of triplication status when choosing components to add. If only a subset of this feedback output section can be triplicated due to area or other constraints, the components closest to the feedback section are chosen first. Nodes 8, 14, and 15 of the sample circuit are all part of this *feedback output* section with nodes 8 and 14 having higher priority than node 15.

If all of the previous sections are able to be triplicated, the last section of the design is the logic that is not related to feedback. That is, it does not affect the feedback state of the design nor is it driven by the logic in the feedback section. When this section cannot be fully triplicated, components are chosen according to their topological ordering, again minimizing the transitions between triplicated logic and non-triplicated logic. In the sample circuit, this section is made up of nodes 16–18. Nodes 16 and 17 have priority over node 18 since they connect directly to previously-triplicated logic.

B. *Fine Grain Partial TMR*

While applying the Partial TMR approach, the BLTmr tool is able to perform TMR at fine increments, giving the FPGA design engineer a large design space to utilize. As previously reported, the BLTmr tool could only apply mitigation at a coarse level. A design was divided into four major sections based on their relationship to the persistent sections of the design [4]. Each section was either triplicated as a whole or left unmitigated.

This coarse level of granularity does not give much flexibility and results in lower reliability than is possible. For example, if one of these four large sections of logic is too large to be triplicated, none of the logic in that section would be protected with TMR. Similarly, if TMR were performed at the module level, a large subcircuit could require more area than is available on the chip to fully triplicate. At this level of granularity, there could be a significant amount of unused resources on the FPGA.

If smaller components in the circuit are considered for TMR, many of these sub-modules may be triplicated even if the full module cannot be. More fully utilizing the FPGA resources by triplicating parts of these larger modules or sections can increase the overall reliability of the FPGA design. The BLTmr tool uses the considerations explained in Section II to work at the logic (LUT) level rather than a higher module level and can effectively use the entire FPGA for the highest reliability possible. The data reported in Section V show that this approach gives results at much finer increments and offers a larger design space of mitigated circuits.

C. *Other Issues*

The BLTmr tool is designed to handle different variations on standard TMR. Standard full TMR, for example, triplicates all inputs and outputs as well as the clock lines. When partially triplicating a design, triplicated clocks and I/O may not be possible. The tool automatically inserts a voter when outputs are not triplicated and splits a signal to three domains when an input is not triplicated. The clock is triplicated where possible, and a single clock line is delivered to non-triplicated logic. As with any non-triplicated logic, SEUs affecting non-triplicated clocks and I/O will cause undetected errors in the circuit.

In addition, the BLTmr tool has unoptimized, default handling for structures such as block memories, bi-directional I/O, and other specialized modules embedded in the FPGA. The user may modify the default behavior of the tool for these types of modules, but the specifics are beyond the scope of this paper.

V. Fault Injection

A. *Test Methodology*

To validate the effectiveness of the mitigation produced by the BLTmr tool, the *sensitivity* and *persistence* of several input

Authorized licensed use limited to: Univ of Calif Los Angeles. Downloaded on January 19, 2009 at 13:40 from IEEE Xplore. Restrictions apply.

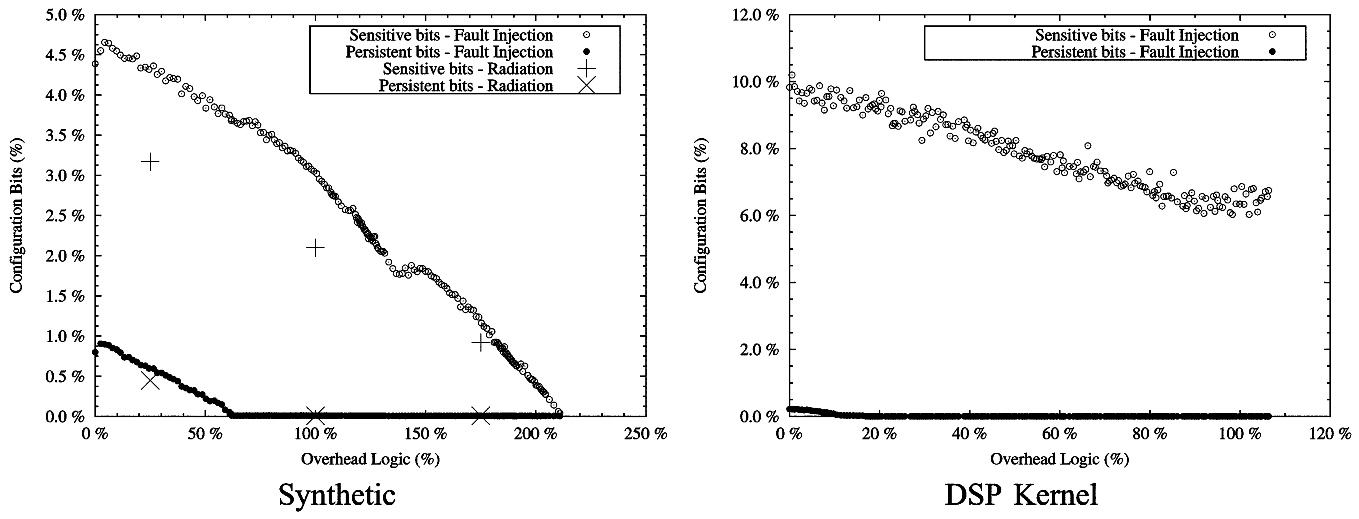

Fig. 4. The sensitivity and persistence vs. amount overhead logic of 200 incremental outputs of the BLTmr tool resulting from a single input file for two different designs. Fault injection results are shown for both design while the synthetic design plot also includes data from radiation experiments for three design points.

designs processed by the tool were measured using fault injection. The sensitivity of a design is the percentage of upsets causing both persistent and non-persistent errors. The persistence is the percentage of upsets causing only persistent errors. This fault-injection tool has been shown to predict sensitivity and persistence rates well as compared to radiation testing [7], [9].

The fault-injection tool used uses an FPGA board on a PCI card with three Xilinx Virtex 1000 (V1000) FPGAs. The first FPGA is loaded with the original, unmitigated design. The second FPGA is configured with the design under test (DUT): the partially mitigated output design from the BLTmr tool. The third FPGA feeds the first two FPGAs with identical inputs and compares their outputs every clock cycle, detecting when the DUT fails to match the original. The three FPGAs are run at 10 MHz. The fault injection tool, which runs as software on the host PC, inserts faults into the configuration of the DUT and records output errors as captured by the third FPGA. Any configuration upset which causes an output error is a *sensitive* upset. Upsets which cause output errors that are not repaired by configuration scrubbing alone are additionally classified as *persistent* upsets. After a persistent error, the FPGA board is reset in order to re-synchronize the first and second FPGAs and testing continues.

This fault-injection tool is designed to test only single-bit upsets in the configuration memory cells of the DUT FPGA. The tool currently does not simulate single-event functional interrupts (SEFIs) nor was it configured to simulate MBUs for the results reported in this paper. In [10], the authors reported that the configuration memory made up 97.4% of the total static cross-section of the V1000 FPGA. Block RAM contribute 2.1% of the cross-section, user flip-flops make up 0.4%, and SEFIs contribute less than 0.0021%. Thus the vast majority of upsets are covered through configuration fault injection. Also, though SRAM FPGAs are susceptible to MBUs [11], the gains in reliability provided by Partial TMR can still be demonstrated by simulating SEUs.

B. Results

The plots in Fig. 4 illustrates the reduction in sensitivity and persistence that can be obtained by incrementally adding TMR to a design with the BLTmr tool. Each plot was generated using our fault injection tool using a single input circuit. The first design, called the Synthetic design, consists of counters, multipliers, and adders. All modules output results every clock cycle and no functional masking occurs. The unmitigated design occupies 24% of the slices in the V1000. The second design is a digital signal processing application called the DSP Kernel design. The unmitigated DSP Kernel occupies 50% of the V1000 slices.

Each point on the plots represents a single mitigated design generated by the BLTmr tool with a different level of TMR requested based on device utilization. For each design, 200 runs of the BLTmr tool were evaluated by our fault injection tool. The vertical axis measures the sensitivity and persistence of the output design in terms of the percentage of FPGA configuration memory cells which are sensitive or persistent. The horizontal axis represents the percentage of overhead logic added to the original design in terms of flip-flops plus LUTs.

The plots show the reduction in both the sensitivity and persistence of the configuration cells as mitigation is incrementally increased. The sensitivity of each design, represented by the open circles, is the percentage of FPGA configuration memory cells that cause the design to fail when their contents are modified by an SEU. The persistence of each design, represented by the closed circles, is the percentage of the configuration cells (a subset of the sensitive cells) which result in a persistent failure in the design when their contents are modified by an SEU. As discussed earlier, a system reset is required in order to recover from a persistent failure.

Because the BLTmr tool is designed to focus on the persistent components of the design first, the plots show that as the number of FPGA resources used for redundancy is increased incrementally, the persistence of each design rapidly declines

TABLE I
RESULTS FROM RADIATION TESTING AT INDIANA UNIVERSITY CYCLOTRON FACILITY (IUCF) COMPARED WITH FAULT INJECTION DATA. THREE PARTIALLY MITIGATED VERSIONS OF THE SYNTHETIC DESIGN WERE IRRADIATED WITH 65 MeV PROTONS

Design Area Overhead	**Flux** ($p/cm^2/s$)	**Duration** (sec)	**Configuration Upsets**	**Sensitive Output Errors**	**Fault Injection Sensitivity**	**% Lower than F.I.**	**Persistent Output Errors**	**Fault Injection Persistence**	**% Lower than F.I.**
25%	4.99×10^7	5165.4	32798	1040(3.17%)	4.32%	26.6%	147(0.448%)	0.596%	24.5%
100%	2.68×10^7	4615.8	23765	520(2.19%)	3.02%	27.6%	2(0.00842%)	0.00926%	9.1%
175%	3.25×10^7	4500.9	28118	260(0.925%)	1.16%	20.4%	2(0.00711%)	0.00704%	−1.0%

until it is nearly eliminated.[1] The sensitivity of each design also decreases since the persistent components are a subset of the sensitive components. Once the persistent components of a design have been protected with TMR and the persistence of the circuit has been eliminated, the focus is shifted to the non-persistent portions of the design. As the number of resources utilized increases, an incremental decrease in overall sensitivity results. In the case of the Synthetic design, both the sensitivity and persistence are reduced to nearly zero. In the case of the DSP Kernel, the design is too large to be fully triplicated and it is not possible to completely eliminate the sensitivity of the design. The persistence, however, is quickly reduced to nearly zero and the sensitivity is reduced by about 40% as compared to the unmitigated design.

These plots illustrate the fine-grain results that can be achieved in terms of reliability and area. Each design point offers a different amount of mitigation and associated resource cost for a different level of sensitivity and persistence. Note that each plot is fairly continuous rather than a step function, which would be the case for a larger level of granularity. Between design points in the Synthetic design, for example, there is an average of 72 LUTs plus flip-flops or about 1,014 sensitive configuration memory cells. The tool can operate at even finer increments as well, but this was not done in this test simply to avoid too much computation in terms of placing and routing and reliability evaluation of each design point. Given reliability and area constraints imposed on a system, the BLTmr tool can be used to find the optimum balance of the two using these incremental results for a particular design and situation.

VI. RADIATION TESTING

A. Test Methodology

The benefits of Partial TMR were also demonstrated in a radiation test. While the fault injection experiments provide a large amount of data at a very low cost, radiation testing more accurately simulates the space radiation environment. Thus the goal of this test was to validate the results of our fault injection experiments through dynamic radiation testing. This study was not concerned with static measurements of device cross section, but the behavior of the partially mitigated designs when exposed to radiation-induced upsets at run-time. The radiation source can further verify the effectiveness of an SEU mitigation technique by providing a random distribution of configuration upsets, user flip-flop upsets, and SEFIs.

[1]Due to limitations in the test fixture, the clock, I/O, and global reset were not triplicated. Measurable amounts of sensitivity and persistence remain even with "full" mitigation.

The test occurred at the Indiana University Cyclotron Facility (IUCF) at Indiana University Bloomington. The IUCF cyclotron was tuned to produce 65 MeV protons and sensitivity and persistence data were collected for each of the chosen design points. Three design points from the set of partially mitigated Synthetic designs were chosen for testing at IUCF: those which added 25%, 100%, and 175% overhead mitigation cost to the original design. In order to accelerate testing, the proton fluxes used (shown in Table I) were much greater than those encountered in orbital environments. For example, the 65 MeV proton flux in a 560 KM, 35.0 degree inclination low earth orbit during solar maximum is approximately 35 particles/cm^2/s. This is approximately six orders of magnitude lower than the flux at the accelerator.

The test fixture used for radiation testing is very similar to the fault injection tool. The same board with three FPGAs is used to perform the test. In this case, the DUT is placed in front of the beam and the rest of the board is shielded. While running, the configuration of the DUT is periodically examined to record upsets caused by the charged particles. When upsets occur, the configuration of the FPGA is refreshed with configuration scrubbing and any output errors are recorded. As with fault injection, output comparisons are made every clock cycle and the sensitivity and persistence of each test design is recorded and the system is reset after persistent errors. In addition, the FPGA board must be power-cycled in the event of a SEFI, which, as mentioned, was not part of our fault-injection experiments. The proton beam is on while the platform is active, including during resets and partial reconfiguration. The beam is stopped only when a SEFI is detected or when changing designs or other test parameters.

B. Results

Table I shows the numerical results collected during radiation testing compared with fault injection results for the same design points. These results are also plotted along side the fault injection data in Fig. 4.

The data presented here shows a similar reduction in sensitivity and persistence as presented in the fault injection data. Notice, however, that these data points show consistently lower percentages in both of these categories. The data collected in the radiation tests measures lower (in general) than the figures predicted by the fault injection experiments. Table I shows the percentage difference between the fault injection and radiation experiments. We feel confident, however, that the trends in the data demonstrate the same reduction in sensitivity and persistence as the finer grain results from the fault injection tool and validate those findings.

Authorized licensed use limited to: Univ of Calif Los Angeles. Downloaded on January 19, 2009 at 13:40 from IEEE Xplore. Restrictions apply.

We do not fully understand why the results from the radiation testing do not correlate better with the fault injection results. In the course of our investigation, we learned that the FPGA chips used in these radiation experiments used a different manufacturing process than the chips originally used to compare with our fault injection tool. These new Virtex 1000 parts, which use a smaller geometry than the previous versions, do not correlate as closely with our fault injection tool. Past experimental results have shown a uniform distribution of upset probability across all configuration bits. We suspect that this is no longer true in these new chips, which would cause our data to be skewed somewhat.

Initial experiments to verify our conjectures show that the upset rate on these new parts is about half that of the old parts in the same environment. We have also seen that by running our fault injection experiments with the configuration upsets observed during the radiation tests rather than injecting random faults assuming uniform upset probability, the results are more highly correlated with the data obtained at the accelerator. We are actively pursuing this line of reasoning in order to obtain more concrete results.

VII. Conclusion

This paper introduced an automated tool for applying TMR to selective portions of an FPGA design. The BLTmr tool used the *Partial TMR* method to automatically choose the most critical sections of the design and applies TMR to those areas first. The tool gives higher priority to *persistent* components which, when upset by an SEU, cause a design failure that must be corrected with a system reset. The tool also focuses on providing continuous sections of triplicated logic in order to maximize the reliability gains achieved.

The BLTmr tool is designed to provide the maximum reliability possible under FPGA resource constraints. For situations in which FPGA resources are limited, the tool can be used to apply TMR to as much of the design as possible in small increments in order to fully utilize those resources. The fine granularity of the BLTmr tool is more effective at using all of the available resources than a higher-level, coarse-grain solution. By more fully utilizing the available resources in conjunction with a focus on the most critical design components, the BLTmr tool provides better reliability for resource-limited FPGA designs in an automated tool.

References

[1] N. Rollins, M. Wirthlin, M. Caffrey, and P. Graham, "Evaluating TMR techniques in the presence of single event upsets," in *Proc. Conf. Military and Aerospace Programmable Logic Devices (MAPLD)*, Washington, DC, Sep. 2003, pp. P63–P63.

[2] C. Carmichael, Triple Module Redundancy Design Techniques for Virtex FPGAs, xAPP197 (v1.0), Xilinx Corp., 2001.

[3] S. Rezgui, G. Swift, K. Somervill, J. George, C. Carmichael, and G. Allen, "Complex upset mitigation applied to a re-configurable embedded processor," *IEEE Trans. Nucl. Sci.*, vol. 52, no. 6, pp. 2468–2474, Dec. 2005.

[4] B. Pratt, M. Caffrey, P. Graham, E. Johnson, K. Morgan, and M. Wirthlin, "Improving FPGA design robustness with partial TMR," presented at the IRPS Conf., Mar. 2006.

[5] P. K. Samudrala, J. Ramos, and S. Katkoori, "Selective triple modular redundancy (STMR) based single-event upset SEU tolerant synthesis for FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 51, no. 6, pp. 2957–2969, Oct. 2004.

[6] K. Veezhinathan, S. N. Mahammad, V. Muralidaran, V. Narayanan, and V. Chandrasekhar, "Reduced triple modular redundancy for tolerating SEUs in SRAM-based FPGAs," presented at the MAPLD Conf., Sep. 2005.

[7] 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. 51, no. 6, pp. 2438–2445, Dec. 2005.

[8] L. Sterpone, M. Violante, and S. Rezgui, "An analysis based on fault injection of hardening techniques for SRAM-based FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 53, no. 4, pp. 2054–2059, Aug. 2006.

[9] E. Johnson, M. Caffrey, P. Graham, N. Rollins, and M. Wirthlin, "Accelerator validation of an FPGA SEU simulator," *IEEE Trans. Nucl. Sci.*, vol. 50, no. 6, pp. 2147–2157, Dec. 2003.

[10] P. Graham, M. Caffrey, M. Wirthlin, D. E. Johnson, and N. Rollins, "Reconfigurable computing in space: From current technology to reconfigurable systems-on-a-chip," in *Proc. IEEE Aerospace Conf.*, Big Sky, MT, Mar. 2003, pp. T07_0603.1–12.

[11] 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, no. 6, pp. 2455–2461, Dec. 2005.

Authorized licensed use limited to: Univ of Calif Los Angeles. Downloaded on January 19, 2009 at 13:40 from IEEE Xplore. Restrictions apply.