A Partial TMR Technique for Improving Reliability at a Low Hardware Cost in FPGAs

D. Eric Johnson
darrel.eric.johnson@gmail.com
Department of Electrical and Computer Engineering,
Brigham Young University,
Provo, UT, 84602

Abstract *The flexibility combined with the computational capabilities of FPGAs make them a very attractive solution for space-based computing platforms. However, SRAM-based FPGAs are susceptible to radiation effects, including Single Event Upsets. In order to increase the fault tolerance of FPGA designs, fault mitigation techniques, such as Triple Module Redundancy, can be applied. Such techniques, however, can be excessive in terms of hardware costs. This work investigates the tradeoffs between fault mitigation techniques for FPGA designs and the corresponding costs of such mitigation. A particular focus is placed upon identifying design components that serve to benefit most from the application of fault tolerance techniques, and investigating the tradeoffs associated with applying mitigation to these most sensitive design sections.*

Introduction

FGPAs are very advantageous for use in custom computing applications because of their computational and reconfigurable capabilities, in addition to their low design entry cost. FPGAs have been successfully used in high throughput signal processing applications, and are suitable for the requirements of space based computing[1],[2].

Though suitable to the computational demands of space based computing, FPGAs are sensitive to radiation effects common in a space environment[3]. This is particularly detrimental for FPGA designs, the configuration of which are defined by the configuration memory. A behavior of a design that is programmed on an SRAM-based FPGA can actually be *altered* due to radiation effects. This is due to a change in state in the FPGA configuration memory, which in turn results in a change in the programmed FPGA design.

Even though SRAM-based FPGAs are sensitive to the radiation effects common in a space environment, FPGA designs can be made fault tolerant through design level mitigation techniques[4],[5]. Mitigation techniques which are exhaustively applied to an FPGA design can remove the possibility of design failures due to single points of failure. However, such exhaustive mitigation techniques, which ensure correct FPGA operation during single points of failure, can be expensive in terms of hardware resource costs and power, as well as detrimental to computational capabilities in terms of latency and throughput.

Certain design failure modes are more critical than others. Additionally, certain types of designs and systems are more tolerant of temporary design failures. For these types of situations, it is not always necessary to apply exhaustive mitigation techniques to an FPGA design. In fact, in such situations an investigation into partial mitigation techniques and the associated tradeoffs proves beneficial.

We have begun an exploration in related to partial mitigation techniques and the associated tradeoffs in terms of reliability and hardware costs. In this paper, we will demonstrate a technique for partial Triple Modular Redundancy (TMR) which, for single points of failure, can remove the dynamic persistent cross section from an FPGA design. We will show how eliminating this particular cross section is beneficial and sufficient for certain types of designs. We will show how system fault tolerance design upon removal of this cross section is simplified. Finally, we will present results obtained through applying an initial version of partial TMR., and will compare the tradeoffs associated with a non-mitigated design, a design with partial TMR, and a design with exhaustive TMR.

Radiation and FPGAs

The flexibility of SRAM-based FPGAs is made possible by the configuration memory which defines the currently programmed design of an

FPGA. Figure 1 illustrates a hypothetical building block of an FPGA architecture. The configuration memory defines the function of the programmed components; for example, Figure 1b shows how the configuration memory could possible define a 4-input AND gate followed by a flip-flop. The configuration memory can be crucial to the correct operation of an FPGA design. As illustrated in Figure 1c, an upset due to radiation can alter the contents of the configuration memory, resulting in a change to the design. In this case, the change is illustrated through a change in logic function, as the 4-input AND gate changes into a 4-input OR gate.

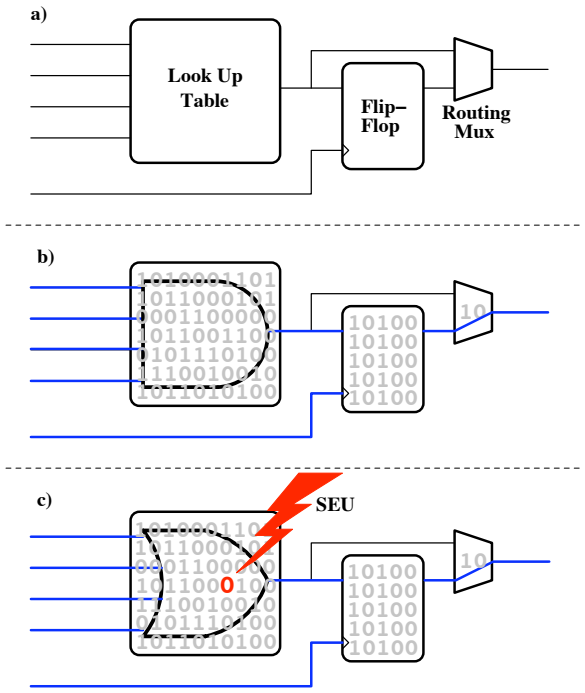

Figure 1: Hypothetical configurable block of an FPGA architecture, illustrating how the configuration memory defines the FPGA design, and how an upset within the configuration memory can alter the design.

The severity of the consequences of an upset within the configuration memory can vary greatly. For example, if an upset occurs within a region of the FPGA that is not currently being used for any design functionality, it is likely that no deviation from expected design behavior will be observed. For a given FPGA design, such configuration memory locations are referred to as non-sensitive or non-critical configuration bits. Those configuration memory upsets which are critical to the correct operation of an FPGA design are referred to as sensitive configuration bits. The set containing all sensitive configuration bits for a given design is referred to as the dynamic sensitive cross section.

In order to guarantee continued correct FPGA design operation in a radiation environment, such as space-based computing, the configuration memory needs to be periodically scrubbed or refreshed[6]. During this process, the contents of the configuration memory of the FPGA are examined and, if an upset is found, it is repaired. Any system intended for operation in a radiation environment that contains SRAM-based FPGAs as part of the computational payload should employ some sort of method for refreshing the contents of the configuration memory, in order to recover from the effects of a sensitive configuration bit upset. The rest of this paper will assume that such an implementation is in place.

The set of sensitive configuration bits can be subdivided into two categories: persistent and non-persistent bits[7]. A non-persistent configuration bit is defined as one which, when upset, results in only a temporary design failure, without the need of a reset[7]. This behavior is better illustrated through Figure 2, which plots the difference between expected design behavior and actual design behavior against time for a non-persistent configuration upset. As the figure illustrates, upon occurrence of the configuration upset, design behavior deviates from expected. Likewise, when the configuration memory upset is repaired, design behavior returns to normal. No external intervention, such as a design reset, is required to cause such non-persistent behavior.

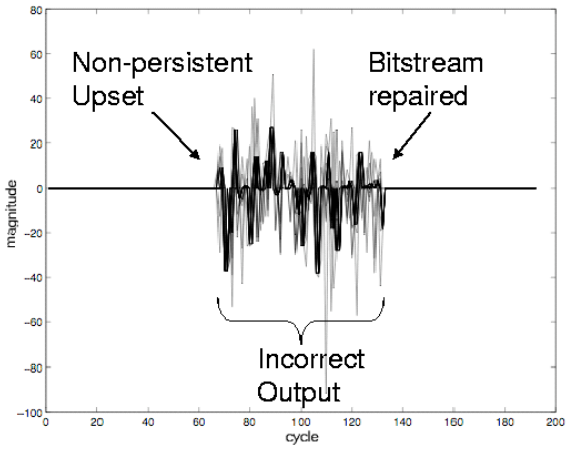

Figure 2: A plot of the difference between expected and actual design behavior against time, illustrating the consequences of a non-persistent configuration bit upset.

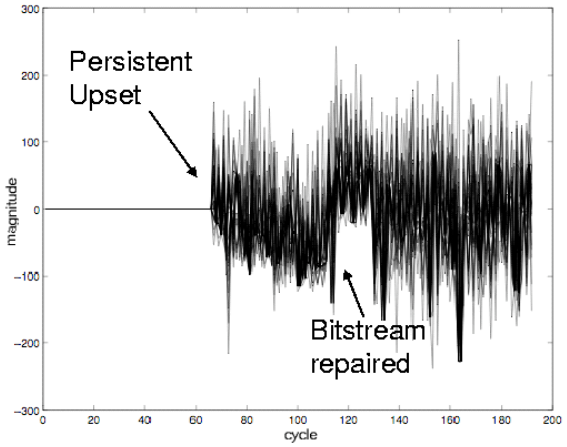

Figure 3: A plot of the difference between expected and actual design behavior against time, illustrating the consequences of a persistent configuration bit upset.

A persistent configuration bit upset is defined as one which, when upset, causes indefinite deviation of design behavior from expected[7]. Even when the configuration memory upset is repaired, design behavior continues to deviate from normal. It is only through external intervention, such as a reset signal, that design behavior will return to expected. Such persistent configuration bits usually define circuit structures which involve some sort of feedback, along with the circuit structures which feed into these feedback paths. Figure 3 illustrates the consequence of a persistent configuration bit being upset. Even though the persistent configuration bit upset is repaired, expected design behavior continues to deviate from normal.

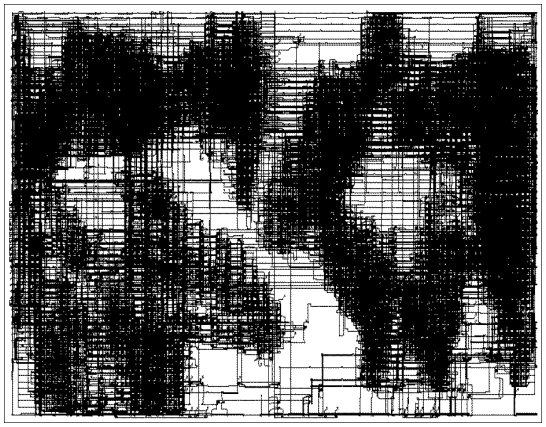

Figure 4: Circuit layout of a signal processing kernel design.

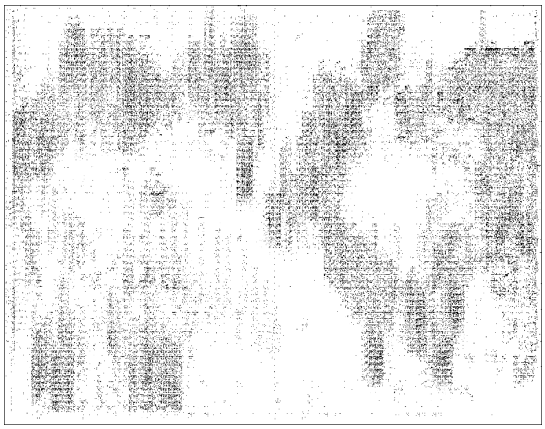

Figure 5: Map of the sensitive configuration memory locations for the signal processing kernel design.

A two-dimensional map can be generated illustrating the physical layout of sensitive locations within the configuration memory for a given FPGA design[8]. For example, the schematic for a design containing a signal processing kernel is shown in Figure 4. This figure illustrates the physical location of utilized FPGA resources, along with the associated routing interconnect. The two-dimensional map representing the location of sensitive configuration bits is presented in Figure 5. Because no fault tolerance techniques were applied to the signal processing kernel design, a high correlation exists between the location of utilized FPGA resources and sensitive configuration memory locations.

Figure 6: Map of the persistent configuration memory locations for the signal processing kernel design.

A map similar to that presented in Figure 5 can also be created for the locations of persistent configuration bits[7]. Such a map for the signal processing kernel design is shown in Figure 6. The set of persistent configuration bits is a subset of the sensitive configuration cross section; consequently, the dynamic persistent cross section is always smaller than or equal to the size of the dynamic sensitive cross section for a given FPGA design.

FPGA Design Fault Tolerance

The severity of errors due to faults within an FPGA configuration memory varies. Additionally, the ability of a given design to tolerate varying degrees of error severity differs from design to design.

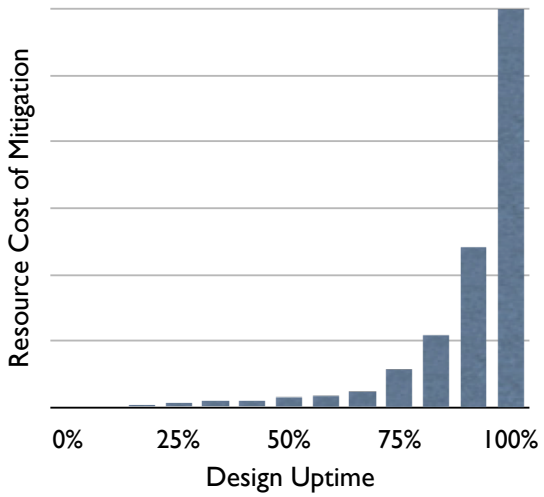

Figure 7: Hypothetical plot of resource cost for fault mitigation versus the corresponding design uptime for a particular design and radiation environment.

Critical applications cannot ever tolerate an error, no matter how infrequent or how short lasting the effects of the error are. Not all applications, however, fall in this category. Applications design to gather sensory information, for example space remote sensing, may be able to tolerate temporary lapses in correct design operation. A given application may be able to tolerate a 95% up time in terms of system reliability as long as the corresponding fault mitigation costs are low. An example plot contrasting design up time versus the corresponding resource cost to guarantee such an uptime is illustrated in Figure 7. This hypothetical figure shows that the costs associated with guaranteeing 100% up time may be excessive or prohibitive; however, a design that can tolerate a 95% up time may be able to afford the component costs to implement the necessary mitigation techniques. This tradeoff will vary on a design by design basis, and can be taken advantage of when implementing fault tolerance techniques.

When considering the cost of fault tolerance mitigation techniques, the severity of the types of errors should be taken into account in addition to design uptime. This idea was touched on earlier during the discussion of persistent and non-persistent configuration bits. To better illustrate, consider a system which contains an FPGA design with a dynamic persistent cross section greater than 0%. In order to guarantee that the design will continue to operate correctly, the system will need to monitor the design for the occurrence of a dynamic persistent error. When such an occurrence is observed, the state of the design must be corrected, for example through a system reset. However, correctly identifying such an occurrence can be a difficult, if not impossible, task. Consequently, it may be much simpler to apply fault mitigation techniques that effectively drive the dynamic persistent cross section to 0%. Such a solution removes the necessity for resetting the system, as well as any detection circuitry for determining when a dynamic persistent error has occurred. For this reason, an investigation into mitigation techniques intended to remove the dynamic persistent cross section would prove beneficial.

Fault Mitigation through Partial TMR

Partial application of Triple Module Redundancy (TMR) is one technique that can be used for eliminating the dynamic persistent cross section of an FPGA design. The dynamic persistent cross section is comprised of those configuration memory locations that define design structures containing feedback, as well as those which feed into these feedback structures, as illustrated by the grayed out section of Figure 8. In order to remove the dynamic persistent cross section from an FPGA design, only the structures comprising feedback and feeding into feedback need to have TMR applied to them. A partial TMR technique such as this can remove the entire dynamic persistent cross section, while reducing the mitigation cost when compared to exhaustive TMR techniques. Instead of applying

mitigation to the entire design, mitigation need only be applied to structures likely to propagate dynamic persistent errors. Instead of mitigating all of the sensitive structures, as illustrated in Figure 5, only the persistent structures need mitigation applied, as in Figure 6.

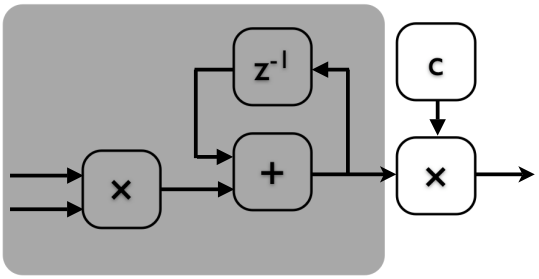

Figure 8: Not all of the components of a design need to be triplicated in order to remove dynamic persistent propogation.

In order to facilitate the identification of feedback structures and the cone of input logic feeding into feedback structures, a Java-based EDIF infrastructure has been developed[9]. EDIF stands for the Electronic Design Interchange Format[10], and is commonly used as an intermediate file format for representing FPGA designs. The Java-based EDIF infrastructure is capable of parsing in an EDIF representation of a digital design, representing the EDIF design structure as a series of Java objects, modifying the EDIF design and implementation, and rewriting the EDIF design structure to file.

We have used the Java-based EDIF infrastructure to parse in FPGA designs and create a graph based representation of a given FPGA design. Once the graph representation of the design has been created, graph algorithms can be used to located feedback structures and identify those design components that need to have TMR applied in order to remove the persistent sensitive cross section. Based upon the identification of these persistent design sections, TMR can be appropriately applied using the Java-based EDIF infrastructure, and the resulting design can be written to file. The Java-based EDIF infrastructure and corresponding tools for graph representation and design manipulation are fast and have been shown to be able to correctly apply TMR mitigation to an FPGA design.

Results

We have used the partial TMR mitigation toolkit to apply partial TMR to a real world FPGA design. The design consists of a signal processing kernel. The hardware on which the design has been implemented for testing is the SLAAC1-V FPGA board, which consists of three Virtex 1000 FPGAs. This platform has been used extensively to perform dynamic sensitive and persistent cross section estimation[7],[8].

Because of the limitations of the FPGA board, the partial TMR technique that we used did not apply TMR to all feedback structures. In particular, applying TMR to all of the block memory structures used in the feedback path of the design exceeds the block memory resources available on the FPGA. For this reason, block memories were a component that was not included in this particular test of the partial TMR technique. Additionally, pin I/O constraints did not allows for us to apply TMR to the clock domain, nor to the inputs of the design. However, this preliminary test of the partial TMR techniques allowed for us to evaluate the effectiveness of our initial partial TMR infrastructure.

Table 1 contains the results of our preliminary investigation into the effectiveness of our partial TMR infrastructure. This table shows the dynamic sensitive and persistent cross sections for the signal processing kernel design, both before and after partial TMR was applied. Also shown is the FPGA resource utilization required by each design.

	Utilized Slices	Sensitive Bits	Persistent Bits
Unmitigated Design	5,778 47%	514,841 8.86%	9,503 0.16%
Partial TMR Mitigation	8,563 70%	525,947 9.05%	2,179 0.0375%

Table 1: Initial partial TMR test results.

From these initial results we can see that partial TMR was successful at reducing the dynamic persistent cross section of the signal processing kernel design. Indeed, the dynamic persistent cross section was reduced by a factor of 4.26, while the required mitigation resulted in only a 49% increase in resource utilization. This increase in resource utilization favorable in comparison to the theoretical increase of 300% required by exhaustive TMR techniques.

Conclusions and Future Work

The dynamic persistent cross section of an FPGA design can be greatly reduced through the application of partial TMR. Future work will focus on applying partial TMR to all of the feedback structures and logic driving the feedback structures in order to demonstrate that the dynamic persistent cross section can be driven to 0%. Additionally, tradeoffs between various levels of partial TMR and resource cost utilization will be investigated. Finally, low cost techniques for identifying the occurrence of dynamic persistent errors in an FPGA design, will be investigated.

References

[1] A. Ramanathan, R. Teisser, and D. McLaugin, "Acquisition of sensing data on a reconfigurable platform," in International Geosciences and Remote Sensing Symposium, 2001.

[2] M. Caffrey, "A space-based reconfigurable radio," in Proceedings of the International Conference of Engineering of Reconfigurable Systems and Algorithms, P. Plaks and P. Athanas, Eds. ERSA, June 2002, pp. 49–53.

[3] E. Fuller, P. Blain, M. Caffrey, C. Carmichael, N. Khalsa, and A. Salazar, "Radiation test results of the Virtex FPGA and ZBT SRAM for space based reconfigurable computing," in 2nd Anual International Conference on Military and Aerospace Programmable Logic Devices, 1999.

[4] C. Carmichael, "Triple module redundancy design techniques for Virtex FPGAs," Xilinx Corporation, Tech. Rep., November 1, 2001, xAPP197 (v1.0).

[5] N. Rollins, M. Wirthlin, P. Graham, and M. Caffrey, "Evaluating TMR techniques in the presence of single event upsets," in 6th Annual International Conference on Military and Aerospace Programmable Logic Devices, May 2003.

[6] C. Carmichael, M. Caffrey, and A. Salazar, "Correcting single-event upsets through Virtex partial configuration," Xilinx Corporation, Tech. Rep., June 1, 2000, xAPP216 (v1.0).

[7] D. Eric Johnson, Keith Morgan, Michael J. Wirthlin, Michael Caffrey and Paul Graham, "Detection of Configuration Memory Upsets Causing Persistent Errors in SRAM-based FPGAs," Military and Aerospace Applications of Programmable Logic Devices (MAPLD), September 8-10, 2004.

[8] D. Eric Johnson, Michael Caffrey, Paul Graham, Nathan Rollins and Michael Wirthlin, "Validation of an FPGA Fault Simulator," IEEE Transactions on Nuclear Science, Volume 50, Number 6, December 2003.

[9] Java EDIF Home Page. http://reliability.ee.byu.edu/edif

[10] Electronic Design Interchange Format. http://www.edif.org