An Overview of Different TMR Design Techniques with Comparison to Alternative Fault-Tolerant Design Methods for FPGAs

Rakan Khraisha

School of Electrical Engineering and Computer Science

ABSTRACT

With growing interest in the use of SRAM-based FPGAs in space and other radiation environments, there is a greater need for efficient and effective fault-tolerant design techniques specific to FPGAs. Triple-modular redundancy (TMR) is a common fault mitigation technique for FPGAs and has been successfully demonstrated by several organizations. This technique, however, requires significant hardware resources. This paper evaluates three TMR design techniques that try to mitigate area overhead. These techniques are compared to other fault tolerant design methods in terms of area overhead, latency, and repair mode.

.

***Index Terms*—** FPGA, TMR, SEU, fault tolerance

1. INTRODUCTION

FIELD Programmable Gate Arrays (FPGA) offer many advantages, such as flexibility and high throughput, to the data-intensive signal processing applications often used inspace-based systems. Unfortunately FPGAs are very sensitive to radiation-induced single-event upsets (SEUs). SEUs are particularly detrimental to FPGAs because they not only can change the state of user flip-flops and internal block memory, but also the contents of the configuration memory which can alter the behavior of the user circuit. This is much different than in custom circuit technologies such as application-specific integrated circuits (ASICs). In these custom technologies, the routing and logic is considered insensitive to SEUs and so only the latches need to be protected. In an FPGA, on the other hand, the latches, logic and routing must all be protected. Furthermore, since any added mitigation circuitry is itself implemented in these radiation-soft resources, care must be taken to eliminate or minimize the potential sensitivity introduced by the mitigation circuitry itself.

In order to reliably operate in a radiation environment, most systems that use FPGAs apply SEU mitigation through some form of design redundancy. The most common technique used within FPGAs is triple-modular redundancy (TMR) [1]. TMR involves triplicating all logic resources and adding simple majority voters to mask an error in one of the three replicates. Designtools have been developed to automatically apply TMR to user FPGA designs. TMR has been shown to be very effective for FPGAs as all resources are replicated including logic, routing and flip-flops.

Although TMR has been shown to significantly improve design reliability, it carries a high overhead cost. At a minimum, full TMR of a design requires three times the hardware to implement three identical copies of a given circuit. In addition, additional logic is required to implement the majority logic voters. In the worst case, TMR can require up to six times the area of the original circuit [2]. The additional hardware resources required to triplicate the original circuit result in other secondary problems such as increased power and slower timing.

There are efforts to identify techniques for reducing the overhead costs of TMR [3]–[5]. The purpose of this study is to investigate other alternatives to TMR that try to mitigate area overhead. In this study we will investigate Partial TMR, fine grained TMR, and Master-Slave TMR in FPGAs. The benefits of each technique is studied .We will examine both the resource overhead cost and reliability benefits of each method in FPGAs compared to full TMR in order to do a cost-benefit analysis.

Each of the techniques works better with different types of circuits and/or applications, and some of the techniques may only work for subsets of a circuit.. Therefore we will only compare each technique to full TMR and will not compare each of these techniques against each other.

There are other self healing techniques like Roving STARS [6], competitive runtime reconfiguration [7]. Please refer to this survey [8] for a complete coverage of fault tolerance techniques.

This paper will begin by reviewing the requirements for mitigating SEU effects in FPGAs and the different methodologies for measuring SEU effects in FPGAs. Next, the TMR technique for FPGAs will be described followed by an introduction to each of the three proposed alternative TMR techniques. Finally, we will present results and compare and contrast TMR to other FT techniques in terms of overhead and reliability.

2. FPGA SINGLE-EVENT EFFECTS

The primary reliability concern for SRAM-based FPGAs operating in a radiation environment is SEUs. An SEU occurs when a single charged particle passes through a

TABLE I
MEMORY BITS WITHIN THE VIRTEX XC4VLX60

Memory Type	# Bits	%
Configuration Memory	13,224,960	74.6%
Block Ram	3,397,712	19.1%
Block Ram Inter.	1,070,592	6%
User Flip-flops	53,248	0.3%
Total	17,728,512	100%

device and transfers charge between the nodes of active regions. This charge transfer can change the state of memory cells in the device.

Devices that contain dense arrays of memory cells are especially sensitive to SEUs due to the large amount of memory state within a relatively small amount of circuit area. SRAM, DRAM and SRAM-based FPGAs are a few devices that fall into this category. Of course SRAM and DRAM devices can hold millions of bits, but most modern FPGAs also contain millions of memory cells for device configuration, internal block memory, user flip-flops, etc. The Xilinx Virtex-4 4VLX60 FPGA, for example, contains almost eighteen million bits of internal state (see Table I). As such, SRAM, DRAM and SRAM-based FPGAs are all especially sensitive to radiation-induced single-event effects.

Although FPGAs and memories (SRAM, DRAM, etc.) have a similar sensitivity to SEUs, it is important to understand that FPGAs have a fundamentally different and unique fault mechanism. In memories, each memory cell simply stores data. Thus an upset of one of these cells corrupts data. In an FPGA, however, the largest component of the memory cells store the functionality of the user-designed FPGA circuit. These cells, called the configuration memory, define the operation of the configurable logic blocks, routing resources, input/output blocks, and other programmable FPGA resources. As a result, an upset in an FPGA memory may actually change the *operation* of the intended user circuit.

It is alsoworth pointing out that FPGAs have a much different fault mechanism than ASICs or other custom circuit technologies. Custom circuit technologies have fixed functionality and thus do not need to store their configuration in memory cells. As such, the routing and logic in ASICs are usually considered insensitive to soft-errors. Consequently, soft-error mitigation techniques usually address only the latches within the circuit. In FPGAs, on the other, hand design reliability techniques must mitigate against errors in the logic, routing, I/O, as well the latches (i.e., flip-flops and block RAM). Since more FPGA resources require error mitigation, the overhead for mitigating failures is much higher in FPGAs.

The fault mechanism in FPGAs also presents another challenge. Since all FPGA resources are sensitive to SEUs, any additional logic added to a circuit to improve the reliability can itself be sensitive to SEUs. In other words, logic added for mitigation can increase the overall SEU sensitive cross-section of a design. If this increase in cross section is greater than the reduction in cross section the scheme provides, there will be a net loss in reliability. As such, any FPGA SEU mitigation technique must eliminate or minimize the potential sensitivity introduced by the mitigation circuitry itself.

3. MEASURING FPGA SUSCEPTIBILITY TO SEU

Each FPGA design has a unique SEU sensitivity profile because each design uses a different set of FPGA resources. This set of FPGA resources corresponds to a custom combination of internal configuration bits that define the behavior of these resources. The number of configuration bits used to define the resources within a design determines the SEU sensitivity of the design. Larger designs using more FPGA resources are generally more sensitive to SEUs than smaller designs with fewer resources. The metric to measure design reliability in this paper is configuration sensitivity, or the number of configuration bits within the device that, when upset, may affect the behavior of the circuit. To determine the benefit of any SEU mitigation approach, it is necessary to accurately measure the SEU sensitivity of a given FPGA design. A variety of techniques have been used to estimate FPGA design sensitivity. Several common techniques are summarized below:

- **Stuck-At Fault Models:** A common approach is to use traditional "stuck-at" fault models for an FPGA design. "Stuck-at" modeling involves gate-level simulation of a design while forcing nodes within the design to zero or one. Stuck-at nodes that modify circuit behavior are considered sensitive. While this approach is relatively straightforward, it significantly underestimates the SEU sensitivity of a design as it does not model the fault mechanism in an FPGA properly.
- **Bitstream Analysis:** Another approach involves analyzing the bitstream of a design to determine how many configuration bits are "active" within the design. While relatively straight forward, this technique generally overestimates design sensitivity as it does not consider logic masking that typically occurs in both mitigated and unmitigated designs.
- **Radiation Testing:** An accurate way of determining sensitivity is to test a design using a radiation source. High-energy particles are applied to the device which induces upsets within all device resources. While statistically accurate, this method is extremely expensive and time consuming and not practical for general design analysis.
- **Fault Injection:** A less expensive way of measuring SEU sensitivity is to artificially insert upsets within the configuration memory using fault injection [9]. this method predict more than 97% of upsets compared to radiation testing. The small difference can

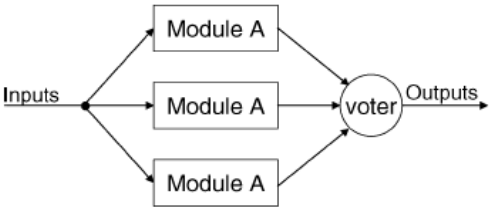

Fig. 1. Triple-modular redundancy fault masking.

be attributed to single-event transients (SET) and SEUs in the user flip-flops and global signals (e.g., configuration logic). Despite this known error, fault injection provides acceptable SEU sensitivity measurements with much less cost and time than radiation testing [10]. Further, this technique can be used to provide an exhaustive sensitivity map of the configuration bitstream.

4. TMR

TMR is a static hardware redundancy scheme for masking single faults in a digital circuit. Fig. 1 depicts the traditional method for TMR. The circuit is replicated three times and a simple majority voter is placed on the outputs. A failure in any one of the three circuit copies will be masked by the majority voter output.

The groundwork for this concept was developed by John von Neumann in 1956 [1]. He proposed a technique of independently computing a signal and using "restoring organs" (e.g., voters) to repair defects in the defective logical "organs" (i.e., circuit replicate). In this work, von Neumann proved mathematically that multiple-line redundancy can improve the reliability of a system composed of unreliable components. Following this influential paper, numerous studies have introduced variations of this technique and proved various properties of redundant hardware systems [11].

TMR is a recognized technique for improving the reliability of FPGAs in a radiation environment. Several projects have demonstrated unique ways of implementing TMR within FPGAs to improve reliability. For example, TMR was used on a 8051-like micro-controller FPGA design and shown to completely remove design failures due to single-bit configuration upsets [12]. Detailed instructions on using TMR within the Xilinx architecture were created to describe several techniques for efficiently implementing TMR [13].

In order for TMR to work properly in an FPGA there should be no more than one upset in the configuration memory at any given time. More than one upset could overwhelm the majority voters and result in a functional error. Since it is inevitable in a radiation environment that upsets will build up over time in the configuration memory, something must be done to periodically remove upsets. In an FPGA configuration memory scrubbing is used. Scrubbing is simply the process of repeatedly correcting upsets in the configuration memory. In [14] they suggested several methods for doing this. If done fast enough for a given environment, scrubbing can ensure there is no more than one upset in the FPGA at a given time.

Used in any technology, TMR comes at great cost. First,TMR can negatively impact timing since the voters inserted are combinational logic and therefore increase path lengths. Second, full TMR of a design requires, at a minimum, three times the hardware to implement three identical copies of a given circuit. Additional hardware is also required to perform the majority voting on the three circuit modules. Some studies have shown that TMR can require up to six times the area of the original circuit [2]. Third, the extra resources ultimately also require more power. While there are efforts to identify techniques for reducing the cost of TMR [3]-[5], any use of redundant hardware to improve reliability will require significant hardware resources. The purpose of this work was to investigate alternative techniques for improving the SEU reliability of FPGA circuits at a lower resource cost than TMR.

5. PARTIAL TMR

This paper [3] describes an efficient approach of applying partial Triple Modular Redundancy (TMR) to an FPGA design to protect against Single Event Upsets. This approach applies TMR selectively to FPGA circuit structures depending on the importance of structure within the design. Higher priority is given to structures causing "persistent" errors within the design. For certain applications, applying TMR to the persistent components can provide higher returns in reliability for the investment in partial TMR cost than full TMR. A software tool is also introduced which automatically classifies circuit structures based on this concept and applies TMR selectively based on the classification of the circuit structure.

5.1 CLASSIFICATION OF SENSITIVE CONFIGURATION BITS

In the paper a new way is introduced to categorize the sensitive configuration bits by separating them into two categories called "persistent" and "non-persistent". A non-persistent configuration bit is a sensitive configuration bit that will cause a design fault when upset through radiation. This will introduce functional errors. When the non-persistent configuration bit is repaired through configuration scrubbing, the design returns to normal operation. Eventually all previously induced functional errors will disappear.

Persistent configuration upsets introduce functional errors which persist after the bit is repaired through scrubbing. Persistent upsets put the design into an incorrect state that cannot self-restore.

The circuit structures that are part of feedback structures within the design contribute to the persistent error behavior. If a circuit fault occurs within a feedback structure, the incorrect values produced by the faulty circuit are propagated into the feedback state. Once the state of the feedback circuit has been corrupted, the circuit will not behave correctly. Although configuration scrubbing will repair the faulty circuit, it will not restore the proper circuit state. The schematics in Fig. 2 illustrate the major subsections of a simple circuit that may cause persistent configuration faults. To maximize the benefits of partial mitigation, redundancy should be added for circuits in Figure 2 (a) and (b) before addressing the circuits of Figure 4 (c).

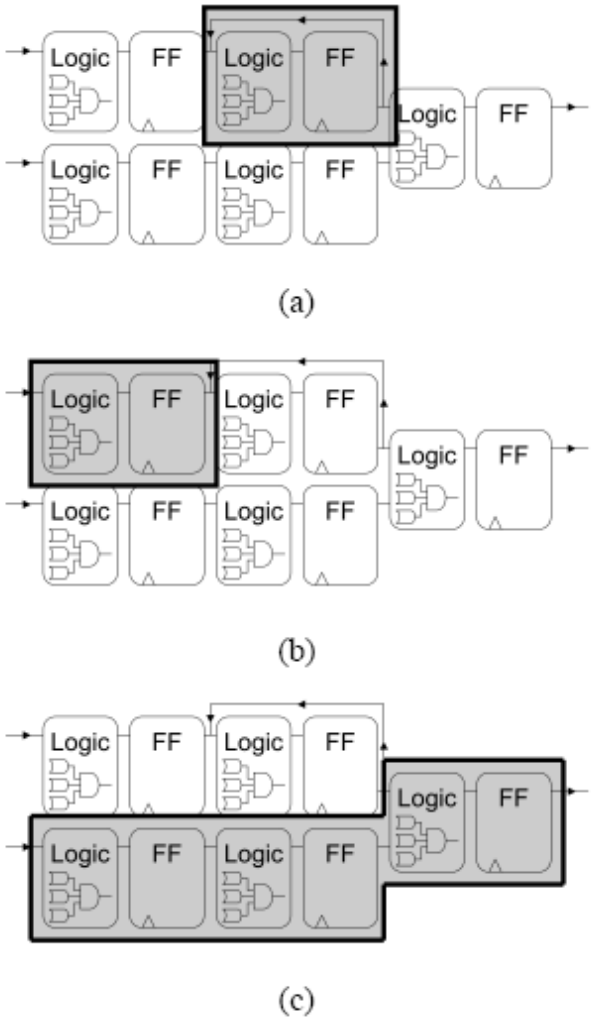

Fig. 2. A simple representation of a circuit with different sections highlighted (a) The feedback section (b) The input to the feedback section (c) The feedforward logic section

5.2 PARTIAL TMR SUMMARY

The selective use of TMR on FPGA circuits was shown to provide improved reliability at a lower cost than full TMR. Since full TMR technique is costly in terms of area and power, the positive benefit of partial TMR is a reduction in mitigation circuitry, and consequently, area and power required. In one of the paper experiments, partial TMR needed 40% area overhead, compared to full TMR which will require 200%. On the negative side, applying only partial TMR, the non-persistent configuration bits are still vulnerable to SEUs.

6. FINE GRAINED TMR

This paper [4] proposes a slight modification to existing SRAM-based FPGA architectures to support fine grain redundancy at an area cost even less than 3x (1.76 x in their experiment). The paper proposes a new architectural-level solution aimed to reduce the cost of TMR. Their approach exploits structures already available in commercial FPGAs, which extends with minor changes the LUT and CLB structure, in order to reduce the cost of mapping fine grain TMR.

6.1 PROPOSED ARCHITECTURE

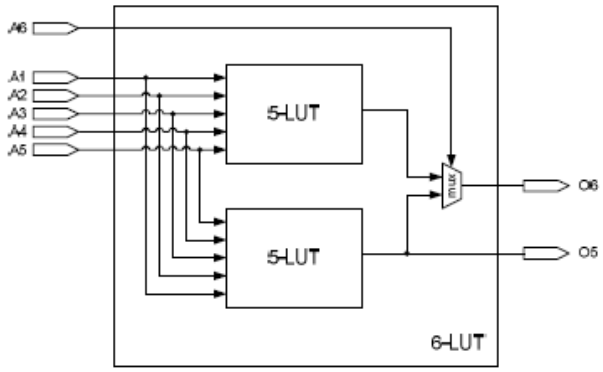

Fig. 3. LUT structure for the Virtex-5 family. Two 5-input LUTs can implement independent functions that share the same inputs. Combined, they implement a 6-input function.

The basic idea is to augment LUTs with TMR functionality, reusing the existing structures as much as possible. The proposal involves the implementation of the voter circuit in within the LUT/slice structures in order to achieve fine grain redundancy and better fault tolerance. The work was done using Xilinx Virtex 5 family architecture. The LUT structure of this family is seen in Fig. 3. Internally, a 6-input LUT is structured as two 5-input LUTs that can be used independently to implement two different Boolean functions as long as these functions use

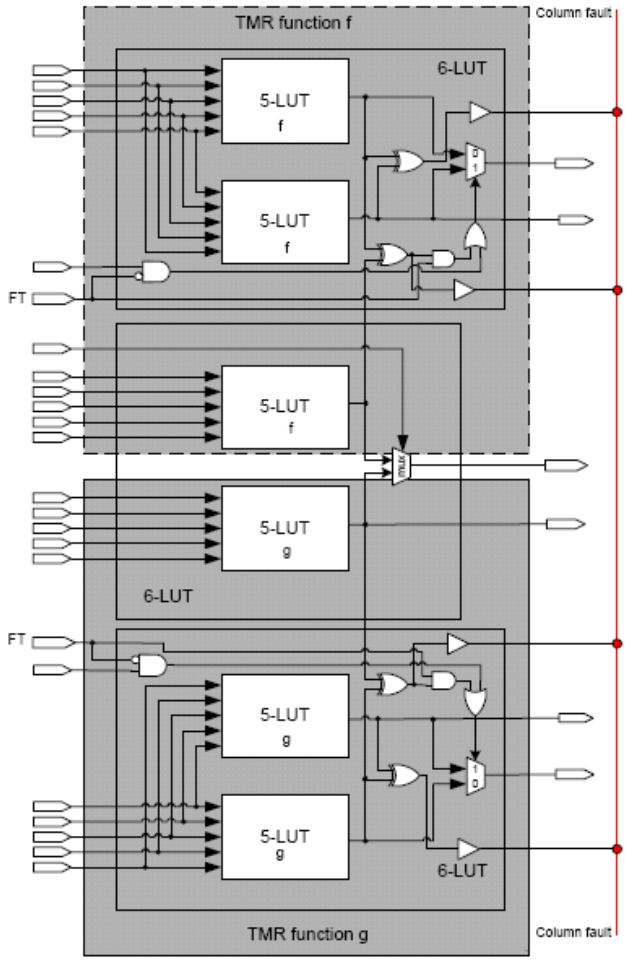

Fig. 4. Tripple-Modular Redundancy in the proposed LUT architecture. An XOR gate compares the first two results and if there is a mismatch the third result is selected.

the same five inputs A1-A5. Under certain conditions the entire LUT is used to implement a *single* five-input function. In that case only the O5 output is used and one of the two 5-LUTS is not used, i.e. it is redundant.

Their approach is to exploit the redundancy and force the design to be mapped and placed only in simple 5-input LUTS so that the other half is reserved. That way they will be able to use the new LUT architecture and apply fine grain TMR with an area cost of about 1.76x compared to more than 3x for the traditional TMR.

In Fig. 4 the fault detection signal can be used to initiate on demand the reconfiguration, avoiding in this way unnecessary reconfigurations. Even better, the column fault signal can be exploited to initiate *partial* reconfiguration instead of a complete scrubbing.

6.2 FINE GRAINED TMR SUMMARY

In the paper they propose a new LUT structure for efficient TMR support in Virtex-5. We reuse as much as possible the existing LUT circuitry and augment it to provide TMR support at reduced cost. The approach offers several advantages: it allows the use of TMR with 76.5% area overhead compared to 242% for the traditional implementations. It also provides a fault detection mechanism and column-based fault location, minimizing the number of required reconfigurations, saving both reconfiguration time and power.

7. MASTER-SLAVE TMR

The proposed method [5] is based on triple modular redundancy (TMR) combined with master-slave technique, an exploiting partial reconfiguration to tolerate permanent faults. In the paper, both single and double faults affecting configurable logic blocks (CLBs) are addressed. The proposed MSU architecture consists of two types of CLBs; CLB-M and CLBS. The relationship between these two architectures is managed by the master-slave technique (MST). When a defect occurs in a CLB-S, an application of partial reconfiguration on the MSU in order to replace the defective CLB-S by another among the three CLBs of the CLB-M.

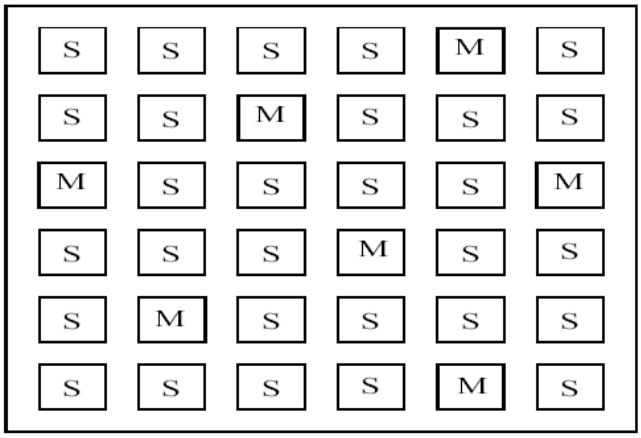

Fig. 5. Proposed SRAM-based FPGA architecture based on MSUs, where each CLB-M (M) is surrounded by four or three CLB-S (S).

7.1 MASTER-SLAVE TMR APPROACH

The proposed approach requires fault detection and diagnosis method as a preprocessing step. The key element of our fault-tolerant approach is partially reconfiguring the SRAM-based FPGA to alternate configuration in response to a fault. Fig.5 illustrates the proposed architecture of SRAM based FPGA based on MSUs, within each MSU a CLB-M surrounded by two, three or four CLB-Ss. In their approach the interface between the MSU and the surrounding areas is fixed and the MSU's function remains unchanged.

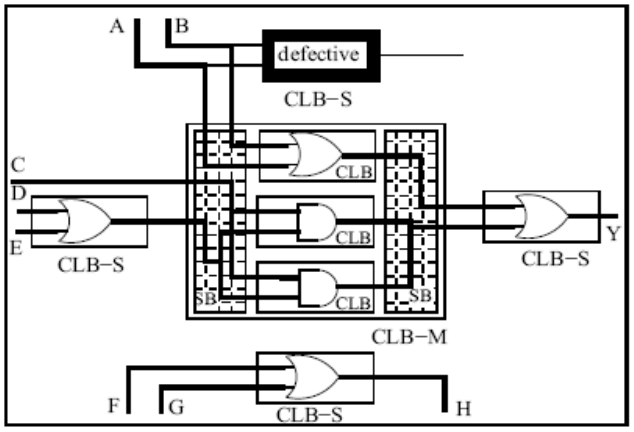

Fig. 6. Overcoming single defect.

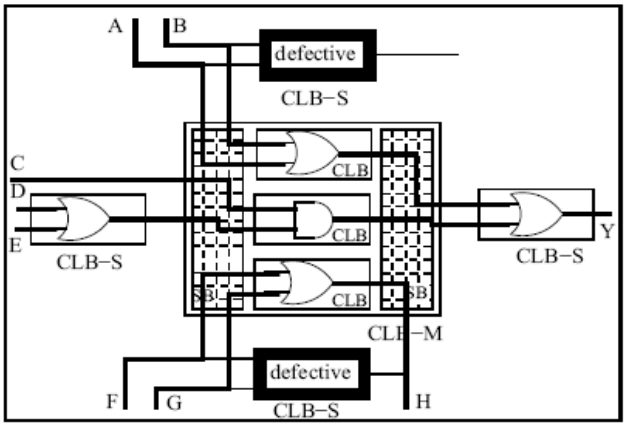

Fig. 7. Overcoming single defect.

Fig.6 shows some cases of overcoming single fault, on the other side Fig.7 illustrates a case of tolerating double faults affecting two CLB-Ss within the same MSU.

7.2 MASTER-SLAVE TMR SUMMARY

This approach has three main benefits compared to redundancy-based fault-tolerance: low overhead, the option of runtime management, and complete availability. Low area overhead is achieved. In the FT-based MST technique only one CLB among a set of five CLBs is subjected to TMR. Finally this approach is targeting both single and double faults by exploiting the master-slave technique.

8. ROVING STARS

In [6] they have demonstrated how self testing of small portions of an FPGA can be carried out simultaneously with normal functioning of other parts. A Self Testing Area, so called STAR is first offloaded by reconfiguring its functionality on another area so that normal functioning is not affected at all, thus enabling online self testing.

However, employing self testing for FT has its shortcomings. Some transient faults that appear and go within a complete test cycle cannot be detected. The time to detect a fault is upper-bounded by the time to test the complete FPGA, but this requires continuous self testing and repeated reconfiguration of STARs. If the failure rate is low, a high rate of self testing may not be justifiable due to the reconfiguration time and power overhead consumed.

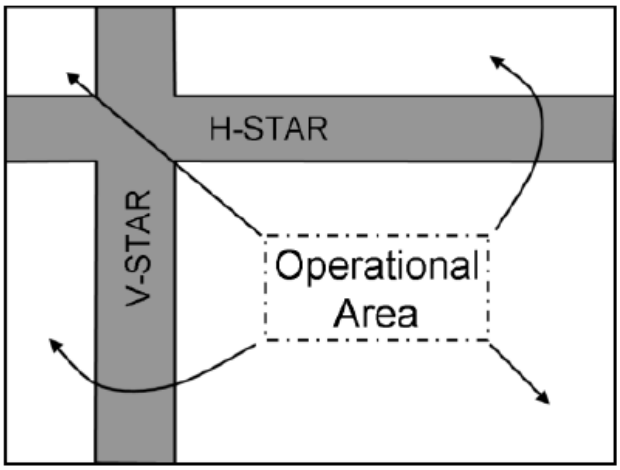

Fig. 8. Roving STARS within FPGA.

9. COMPETITIVE RUNTIME RECONFIGURATION

In [7] a method is proposed as complete system for achieving fault tolerance, based on a pool of competitive configurations. To detect errors and assign a ‘fitness’ to each configuration, two functionally identical configurations are invoked from the pool and the outputs are compared. Over time, different pairs are selected and each configuration accumulates a score representing the number of mismatches it has generated. Configurations that exceed a certain threshold are put through a mutation process to attempt to correct the fault.

TABLE II : COMPARISON OF VARIOUS FAULT-TOLEARNT TECHNIQUES FOR FPGA

FT Parameter	STARS[6]	TMR[3],[4],[5]	CRR[7]
Mode of operation: Online or offline	Test area is offline. While the remaining portion of FPGA is online	Online by majority voting	Online. Uses data input for testing
Ability to handle transient faults	No. time to detect fault is upper bounded by time to reconfigure and test the full FPGA area	yes	Yes, through repetitive comparison of competitive configurations
Area overhead due to added FT functionality	BIST circuitry	Area overhead to implement TMR and voter. Area overhead varies by the technique used	Area overhead to implement CED and checker and algorithm controller
Area overhead due to communication requirement	Routing of clock, test data input and output through scan chain path	Routing input signals to triplicate logic and output to voter	Routing input signals to the modules and output to checker
Memory space requirement	Stores the signatures of the circuit	none	Memory used to hold competitive configurations and GA algorithm
Time overhead	Significant time due to testing(offline) and reconfiguration and roving the star	Time for voting on multiple results	Time to find a solution through evolution
Diagnostic capability	Provides very high coverage	Identify the minority unity as faulty. Cant diagnose routing errors	Provides high coverage
Recovery Mechanism	Bypass faulty blocks by reconfiguring their functionality	Use the majority output as correct	Through evolution of GA algorithm.

10. COMPARISON OF DIFFERENT FAULT TOLERANT TECHNIQUES

We have discussed about various types of FT techniques. Different applications may require the use of different techniques. A lot of factors must be taken into account: performance, power, cost etc. Table II provides a comprehensive summary of the comparative analysis between various FT techniques.

11. CONCLUSION

In this work, we have evaluated three TMR mitigation techniques in an attempt to identify a technique that is more cost effective at increasing reliability and reducing area overhead than the classical TMR. This study evaluated the techniques of partial, fine grained, and master-slave TMR and compared them in both area and SEU sensitivity to full TMR. The results show that the reliability of these techniques does compare well to TMR, and achieve much less area overhead. Finally a comparison is made with other FT techniques.

REFERENCES

[1] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata Studies, no. Ann. Math Studies*, no. 34, 1956.

[2] M. J. Wirthlin, N. Rollins, M. Caffrey, and P. Graham, "Hardness by design techniques for field-programmable gate arrays," in *Proc. 11th Annu. NASA Symp. VLSI Design*, Coeur d'Alene, ID, May 2003, pp. WA11.1–WA11.6.

[3] Pratt, B.; Caffrey, M.; Graham, P.; Morgan, K.; Wirthlin, M.," Improving FPGA Design Robustness with Partial TMR*," IEEE 44th Annual Reliability Physics Symposium Proceedings*, 2006.

[4] Kyriakoulakos, K.; Pnevmatikatos, D.," A novel SRAM-based FPGA architecture for efficient TMR fault tolerance support ," *International Conference on Field Programmable Logic and Applications, 2009.*

[5] Lahrach, F.; Doumar, A.; Châtelet, E.; Abdaoui, A.," Master-Slave TMR Inspired Technique for Fault Tolerance of

SRAM-Based FPGA ," *IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2010*.

[6] EMMERT, J. M., STROUD, C. E. and ABRAMOVICI, M. Online Fault Tolerance for FPGA Logic Blocks. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 15, 2 216–226, 2007.

[7] DEMARA, R. F. and ZHANG, K. Autonomous FPGA fault handling through competitive runtime reconfiguration. In *Proceedings of the NASA/DoD Conference on Evolvable Hardware*, Washington D.C., U.S.A. (June 29–July 1 2005), 109–116.

[8] M.G. Parris, C.A. Sha rma, and R. F. DeMara, "Progress in Autonomous Fault Recovery of Field Programmable Gate Arrays," accepted to ACM Computing Surveys, December 27, 2009.

[9] E. Johnson, M. J. Wirthlin, and M. Caffrey, T. P. Plaks and P. M.Athanas, Eds., "Single-event upset simulation on an FPGA," in *Proc.Int. Conf. Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Jun. 2002, pp. 68–73.

[10] 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.

[11] R. A. Short, "The attainment of reliable digital systems through the use of redundancy – A survey," *IEEE Comput. Group News*, vol. 2, no. 2, pp. 2–17, Mar. 1968.

[12] F. Lima, C. Carmichael, J. Fabula, R. Padovani, and R. Reis, "A fault injection analysis of Virtex FPGA TMR design methodology," in *Proc. 6th Eur. Conf. Radiation Effects Compon. Syst. (RADECS 2001)*, 2001.

[13] C. Carmichael, "Triple Module Redundancy Design Techniquesfor Virtex FPGAs," Tech. Rep. Xilinx Corporation, 2001, vol. 1.0,xAPP197

[14] C. Carmichael, M. Caffrey, and A. Salazar, "Correcting Single-EventUpsets Through Virtex Partial Configuration," Tech. Rep. Xilinx Corporation, 2000, vol. 1.0, xAPP216.