A Comparison of TMR With Alternative Fault-Tolerant Design Techniques for FPGAs

Keith S. Morgan, Daniel L. McMurtrey, Brian H. Pratt, and Michael J. Wirthlin

***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 additional mitigation techniques and compares them to TMR. These include quadded logic, state machine encoding, and temporal redundancy, all well-known techniques in custom circuit technologies. Each of these techniques are compared to TMR in both area cost and fault tolerance. The results from this paper suggest that none of these techniques provides greater reliability and often require more resources than TMR.**

***Index Terms*—Dynamic testing, error propagation, FPGA, persistence, proton accelerator, quadded logic, radiation, SEU, simulator, state machine encoding, temporal redundancy, TMR.**

I. Introduction

FIELD Programmable Gate Arrays (FPGA) offer many advantages, such as flexibility and high throughput, to the data-intensive signal processing applications often used in space-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.[1] 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.

Manuscript received July 17, 2007; revised September 11, 2007. This work was supported by the Department of Energy at Los Alamos National Laboratory. This paper is approved for public release under LA-UR-07-6132; distribution is unlimited.

K. Morgan is with the Space Data Systems Group, Los Alamos National Laboratory, Los Alamos, NM 87545 USA (e-mail: morgank@lanl.gov).

D. McMurtrey, B. Pratt, and M. Wirthlin are with the Department of Electrical and Computer Engineering, Brigham Young University, Provo, UT 84602 (e-mail: dmcmurt@sandia.gov; brianpratt@byu.edu; wirthlin@ee.byu.edu).

Digital Object Identifier 10.1109/TNS.2007.910871

[1]In some cases the routing and logic may need protection from single-event transients (SETs).

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. Design tools 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 incread power and slower timing.

There are efforts to identify techniques for reducing the overhead costs of TMR [3]–[5], but to the authors' knowledge, only Lima's hybrid method has been shown to provide reliability benefits *equivalent* to TMR at a lower cost [6]. The purpose of this study is to investigate other alternatives to TMR that have been introduced for *non-FPGA* custom circuits (e.g., ASICs). In this study we will investigate temporal redundancy, state machine encoding, and quadded logic *in FPGAs*. The benefits of each technique in custom circuit technologies are well-known and have been the study of several different efforts [7]–[12]. We will examine both the resource overhead cost and reliability benefits of each method *in FPGAs* compared to 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. As a result, our study used different test circuits for each of the three methods. Therefore we will only compare each technique to TMR and will not compare compare each of these techniques against each other.

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 SEU mitigation techniques. Finally, we will present results and compare and contrast each technique to TMR in terms of overhead and reliability.

II. FPGA Single-Event Effects

The primary reliability concern for SRAM-based FPGAs operating in a radiation environment is SEUs. An SEU occurs

TABLE I
MEMORY BITS WITHIN THE VIRTEX XC4VLX60

Memory Type	# Bits	%
Configuration Memory	13,224,960	74.6%
Block RAM	3,379,712	19.1%
Block RAM Interconnect	1,070,592	6.0%
User Flip-flops	53,248	0.3%
Total	17,728,512	100%

when a single charged particle passes through a 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 also worth 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.[2] 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.

[2]This is not always true as upsets within logic and routing may generate transient errors that are latched within the sequential circuitry.

III. MEASURING FPGA SEU SENSITIVITY

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 [13]. Johnson *et al.* showed this method to predict more than 97% of upsets compared to radiation testing. The small difference can 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 [14]. Further, this technique can be used to provide an exhaustive sensitivity map of the configuration bitstream.

For our initial experiments we used fault injection to measure SEU sensitivity. Fault injection is far less expensive than radiation testing and the improvement (or reduction) in reliability can be determined by comparing the SEU sensitivity of the unmitigated to the mitigated design.

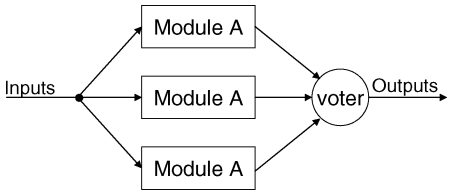

Fig. 1. Triple-modular redundancy fault masking.

TABLE II
TMR SENSITIVITY AND AREA RESULTS

Design	Mitigation	Area (LUTs/ FFs/ Slices)	Area Overhead (%)	Sensitivity Reduction (%)
8-bit Counters	None	1360/1025/689	n/a	n/a
	TMR	7170/3075/5052	633	94
Synthetic	None	1360/3140/1991	n/a	n/a
	TMR	7740/13864/12277	517	99

IV. 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 seminal paper, numerous studies have introduced variations of this technique and proved various properties of redundant hardware systems [15].

A. TMR in FPGAs

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 [16]. Detailed instructions on using TMR within the Xilinx architecture were created to describe several techniques for efficiently implementing TMR [17].

Several tools have been developed for automating the process of applying TMR [5], [18], [19]. The effectiveness of TMR circuits produced by these tools has been verified in radiation and with fault injection [20]. For example, the BYU-LANL TMR Tool (BLTMR) demonstrated over two orders of magnitude improvement in SEU sensitivity in both fault injection and radiation testing (see Table II) [5].[3] Commercial FPGA vendors have also demonstrated significant improvements in design reliability by automatically applying TMR [19].

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. Carmichael *et al.* suggested several methods for doing this [21]. 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.

The number of allowable upsets at any given time varies with the error correction capabilities of a given reliability improvement scheme. The techniques studied in this paper all have the same requirement as TMR, therefore implementations of these schemes in an FPGA would require configuration memory scrubbing at a rate fast enough to ensure no more than one upset in the FPGA at a given time.

B. TMR Costs

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], 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.

V. TEMPORAL REDUNDANCY

The first alternative SEU mitigation technique we studied was temporal redundancy. Unlike TMR which uses spatial redundancy, temporal redundancy, like its name implies, uses redundancy in time. A computation is repeated on the same hardware at three different times. Many studies have shown the utility of temporal redundancy in custom circuit technologies [7], [8].

The simplest method to implement temporal redundancy is to repeat the exact same computation on the same hardware module three times. This method, however, is inferior to TMR since it only corrects transient errors. A permanent fault in a module would produce incorrect results all three times (except

[3]This work did not perform triplication on clocks and I/O due to limitations on the test fixture. A measurable amount of sensitive circuitry remain in these areas.

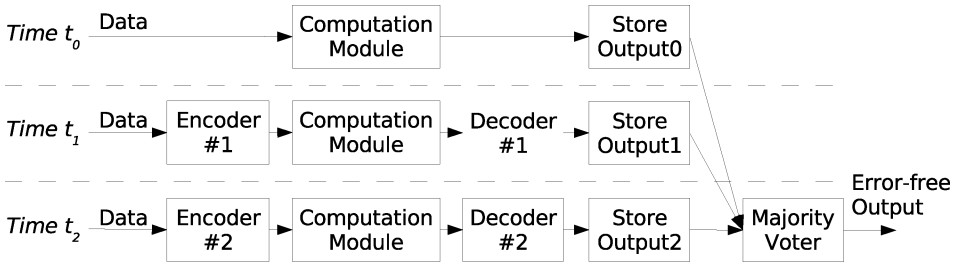

Fig. 2. Each row represents the computations performed at time steps t_0, t_1, and t_2 respectively in temporal redundancy.

when the fault first manifests itself, in which case one, two or all three results would be incorrect).

We will consider an SEU in the FPGA configuration memory to be a permanent fault. After an SEU has changed the configuration of the circuit, the circuit will continue to malfunction until the configuration is repaired by an outside process. Though configuration scrubbing does repair the circuit, it will likely be many clock cycles (thousands on average) after the SEU has occurred, much longer than a typical transient event.

In order to make a fair comparison to TMR we only used forms of temporal redundancy that correct both permanent and transient faults. Fig. 2 shows the basic method. At time t_0 the computation is performed. At time t_1 the inputs are encoded, run through the same computation module, and then decoded. At time t_2 the inputs are encoded with a different algorithm, run through the same computation module, and then decoded with a different algorithm. By uniquely encoding and decoding the inputs on the second and third pass, two of the three executions will correctly compute the output even if the computation module has a single permanent fault. Care should be taken to ensure that the encoding scheme can handle permanent faults which manifest themselves *during* one of the three computations. With two correct executions a final majority voter can select the correct output.

Hsu and Swartzlander devised an alternative method of temporal redundancy for arithmetic computations they called time-shared TMR (TSTMR) [7]. For example, TSTMR splits up an addition module as well as the addition operation into three parts [7]. This way, each of the three partial sums is computed simultaneously on three separate hardware modules as shown in Fig. 3. The addition operation is performed in thirds, starting with the least significant bits and simultaneously performed on three addition modules. This approach has a relatively low hardware cost since the three partitioned addition modules roughly equal the size of the original module. The only hardware overhead comes in the form of control logic and storage for intermediate results. This method has been demonstrated with both adders and multipliers.

Townsend *et al.* developed a slight variation of TSTMR they called quadruple time redundancy (QTR) [8]. This method has a slightly lower hardware cost than TSTMR, but a greater cost in terms of latency.

One challenge with temporal redundancy is finding appropriate encoding and decoding blocks. To the authors' knowledge, a simple set of encoders and decoders has not been found even for trivial arithmetic operations. A second challenge is that,

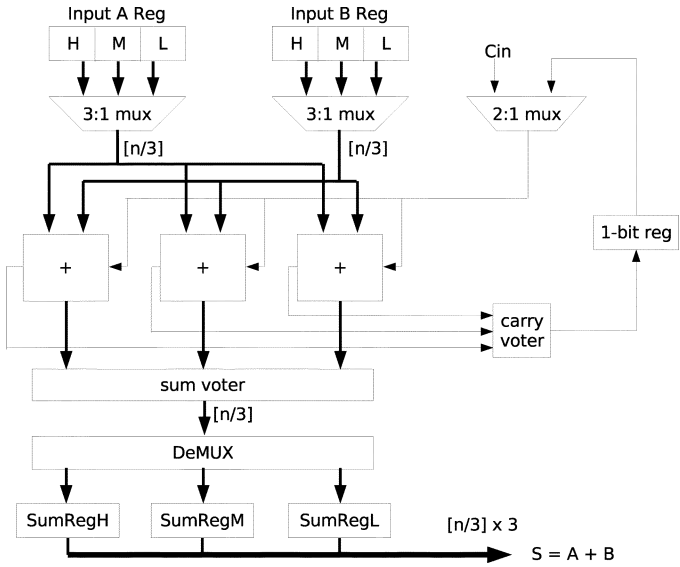

Fig. 3. Time-shared triple-modular redundancy error-correcting adder [7].

in an FPGA, the overhead logic (i.e., encoders, etc.) is susceptible to SEUs and can potentially add more unreliability than the reliability it adds to the original circuit.

A third challenge is that temporal redundancy can alter the timing of the circuit. In addition, temporal redundancy requires $3\times$ or more clock cycles to complete as the original computation, though the clock period may be reduced, depending on the implementation used. Despite these challenges, temporal redundancy methods use fewer resources than TMR, which, in some situations, may be more important than timing considerations.

VI. State Machine Encoding

The next technique we studied was error correcting codes (ECC) to protect finite state machines (FSM) from SEUs. Protecting FSMs with an ECC is a well-studied problem for custom circuit technologies. In independent efforts Rochet and Kumar compared single-error ECCs to TMR in custom ASIC architectures [9], [10].

For our study, we explored the following ECCs: explicit error correction (EEC) [22], implicit error correction (IEC), [22], and a technique proposed by Armstrong [23]. Unlike TMR where an FSM is simply replicated entirely, ECCs redundantly encode the FSM's state variable.

- **Explicit Error Correction:** For explicit error correction the state variables are encoded and additional circuitry is

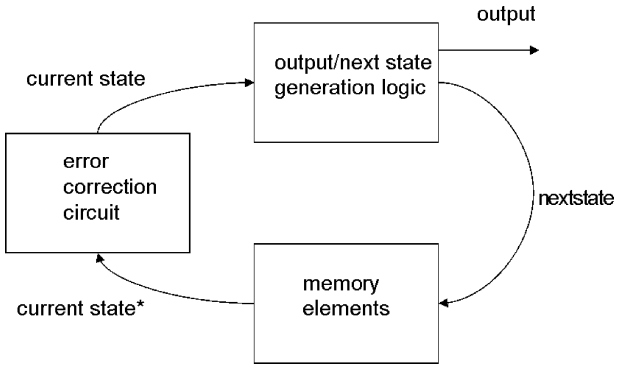

Fig. 4. Finite state machine implementation with explicit error correction.

added to detect and correct errors in the encoded state variable. Fig. 4 shows a block diagram of an FSM protected with EEC. The error correction circuitry is placed between the state storage flip-flops and the next state and output forming logic. This circuitry detects and corrects errors in the state bits, providing a major advantage since the correct codeword is then available to the next state and output forming logic.

- **Implicit Error Correction:** Unlike explicit error correction, implicit error correction (IEC) does not use additional hardware as error correction circuitry. Instead the next state logic is expanded to include all the "erroneous" states that are a Hamming code distance of one away from valid states. The major advantage of this technique is there is no need for additional error correcting logic. However, with the added states, the next state forming logic is more obtuse than in its original un-encoded form. "Don't cares" in the state logic are reduced because the set of invalid and valid codewords must be handled instead of just the valid codewords. The same principles must be applied to the output forming logic to ensure the output is correct.
- **Armstrong's Error Correction Technique:** Armstrong's FSM encoding method can be seen in Fig. 5 [23]. The outputs of the excitation circuit (including the next state codeword and the system outputs) are divided into r subunits. Each subunit generates p bits of the total set of outputs. The input to each subunit is the only actual state variable. Breaking the next state logic and output forming logic into subunits reduces the chance that an SEU will affect the logic that forms more than one output or next state bit.
  Similar to the EEC technique, the inputs to the subunits must be correct to form the correct outputs and next state. Error correcting circuitry is placed before the inputs of the subunits. The advantage of this system is that the subunits function independently of one another. Only the actual state bits need to be corrected while errors in the check bits need not be corrected as the check bits are not used to generate the next state or output of the system. The check bits are used only to indicate which, if any, of the state bits is in error. This method has the same disadvantages of error correction circuitry that the EEC technique has.

A major advantage of ECCs is that the protected circuit is not $3\times$ larger than the unmitigated circuit like with TMR. ECCs

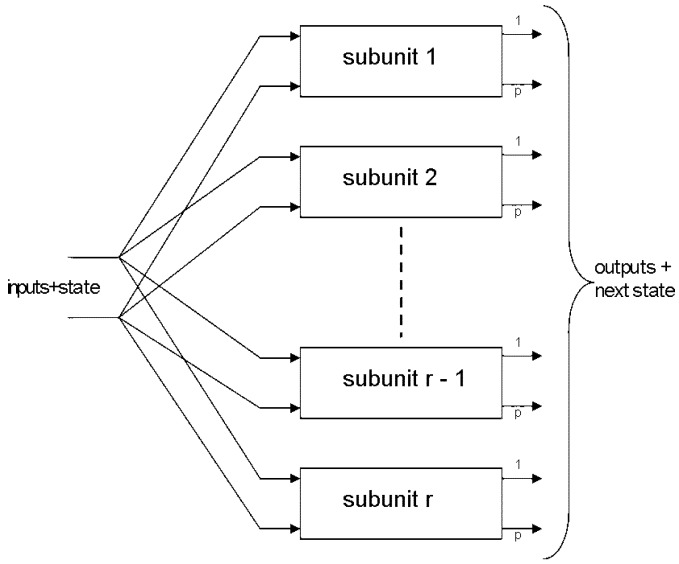

Fig. 5. Armstrong's proposed error correction method.

save resources by using minimal redundancy and no voters, although some ECCs do require additional overhead for an error correction circuit. As is the case with temporal redundancy, in an FPGA the overhead logic can potentially add more unreliability than the reliability it adds to the original circuit. Furthermore, like voters, the overhead circuitry can negatively affect timing.

VII. Quadded Logic

The final alternative SEU mitigation technique we studied was quadded logic. Tryon and Pierce independently developed quadded logic in the early 1960's shortly after von Neumann published his groundbreaking work on reliable computing [1], [11], [12]. Pierce actually called his more generic theory interwoven logic. This work focused on quadded logic since it is the minimally redundant version of interwoven logic.

Quadded logic is built on four main concepts:

1) the network of logic gates is a plane of alternating AND and OR stages,
2) each logic gate is quadruplicated,
3) an error is corrected within two levels of logic from the place of origin, and
4) the uncorrupted wires in the redundant digital signal mask the error [11].

In [11], Tryon outlined the steps required to apply these four principles to a circuit. First, the circuit must be specified as a network of alternating AND and OR stages. Second, each logic gate is replicated four times. Since each gate is quadruplicated, four versions of each of the original signals now exist in the quadded logic circuit. Finally, each gate receives two (of the now four) versions of each input signal it took in the original circuit. Tryon developed a regular pattern for selecting two of the four signals which Pierce later generalized. Fig. 6 illustrates a section of a circuit protected by quadded logic.

Unlike TMR which masks errors with voters, quadded logic includes error correction in the same hardware that performs the intended function. Fig. 7 illustrates how this works. In Fig. 7,

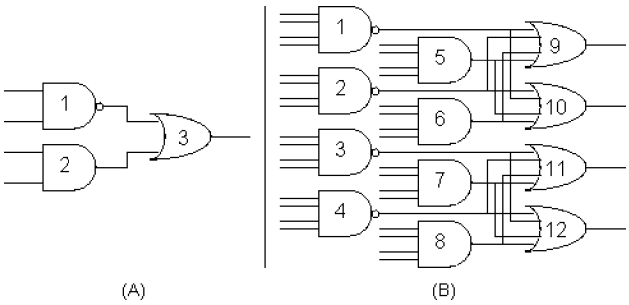

Fig. 6. (a) An example network of two-input logic gates. (b) The resulting circuit after quadded logic is applied to the network in (a).

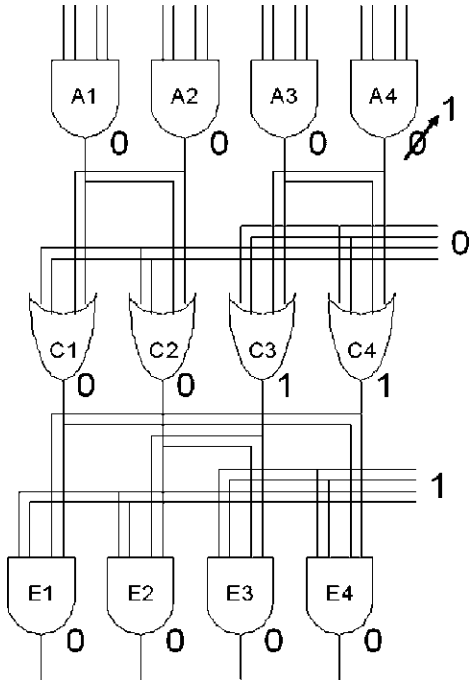

Fig. 7. Erroneous one from an AND being corrected in next level of AND [11].

AND gate A4 erroneously output a logical one. This error spread to OR gates C3 and C4 to which the output of AND gate A4 is connected. This caused gates C3 and C4 to also erroneously output a logic one. However, these two errors were masked in the next level of logic. The zero output of gates C1 and C2 forced AND gates E1, E2, E3 and E4 to all output the correct value since the outputs of AND gates C3 and C4 do not combine anywhere in the next level of AND gates.

Since quadded logic has error correction embedded within the functional logic it does not incur the overhead of voters like TMR. Since voters require extra hardware and negatively impact timing, quadded logic is a potentially attractive solution for FPGAs. However, quadded logic requires $4\times$ more gates and $2\times$ more inputs at each gate. This input count growth, in particular, does not scale well in a look-up table (LUT) based FPGA because LUT memory size grows exponentially with the number of inputs.

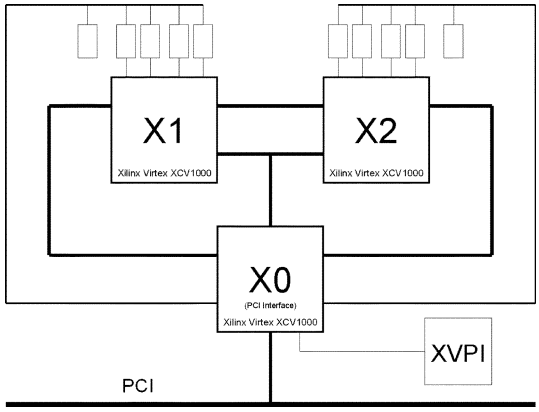

Fig. 8. Simplified schematic of the SLAAC1-V configurable computing platform.

VIII. Testing Methodology

As mentioned in Section III, we used fault injection to measure SEU sensitivity. For this investigation we used a Xilinx Virtex fault-injection tool based on the SLAAC1-V[4] test fixture. The fault-injection tool has been shown to provide over 97% accuracy when compared to radiation testing [14].

A simplified schematic of the SLAAC1-V configurable computing board is shown in Fig. 8. It has three Xilinx Virtex XCV1000 FPGAs and one smaller Xilinx FPGA. FPGA X1 houses the circuit design under test (DUT). FPGA X2 holds a second "golden" circuit (identical copy) programmed into X1. FPGA X0 contains difference circuitry. When the outputs of X1 and X2 do not match, X0 detects the difference. The fourth, smaller FPGA labeled XVPI has the configuration circuitry to program X0, X1 and X2.

The fault injection algorithm, depicted graphically in Fig. 9, is as follows: A bit within the configuration bitstream is toggled from its correct state. After a finite length of time, FPGA X0 is queried to see if it detected differences in the outputs of FPGAs X1 and X2. If X0 detected output errors, the toggled bit is marked as sensitive in a file. Finally, the corrupted bit is toggled again, restoring its original state.

Once all of the sensitive bits in a design have been identified, its sensitive dynamic cross section, σ_d, can be estimated by taking the fraction of "sensitive" configuration bits identified in the design, multiplied by the device's measured static cross section, σ_s, or stated mathematically,

$$\sigma_d = \sigma_s \times \frac{\#\ \text{sensitive bits}}{\#\ \text{total bits}}, \tag{1}$$

where,

σ_d = estimated dynamic cross section,

and

σ_s = measured static device cross section.

Since σ_s has units of cm^2 and the ratio of sensitive to total bits is unit-less, the result has units of cm^2.

[4]Systems Level Applications of Adaptive Computing (SLAAC) board, version 1, with Virtex FPGAs.

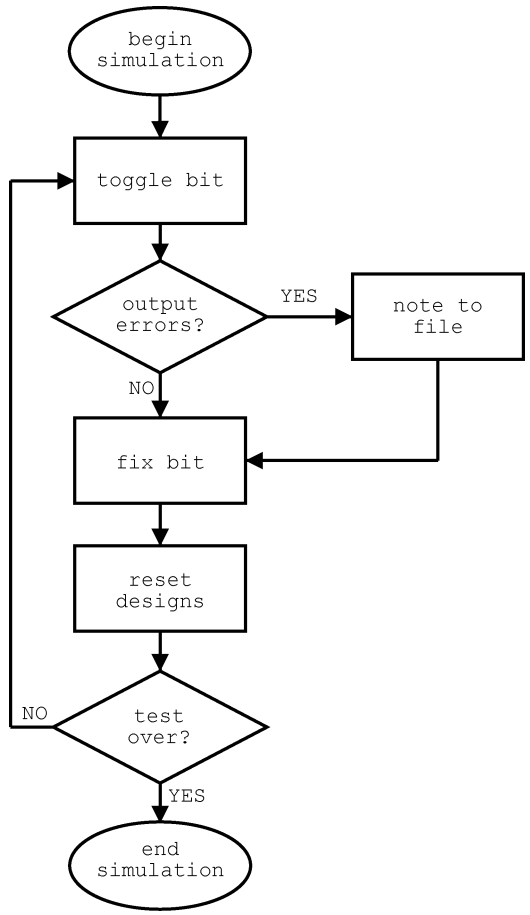

Fig. 9. Fault injection flow diagram.

TABLE III
FAULT INJECTION SEU SENSITIVITY REDUCTION AND AREA OVERHEAD RESULTS

Study	Design	Mitigation	Area (LUTs/FFs/Slices)	Area Overhead (%)	Sensitivity Reduction†(%)
Temporal Redundancy	36-bit Adder	None	109/70/73	n/a	n/a
		TSTMR	121/136/125	71	-66
		TSTMR*	363/340/526	621	44
		QTR	114/138/115	58	-68
		QTR*	345/341/502	588	49
		TMR	327/246/407	458	49
State Machine Encoding	FSM1	None	7/5/5	n/a	n/a
		IEC	35/9/19	280	-108.7
		EEC	70/9/38	660	-94.8
		Armstrong	28/8/16	220	25.2
		TMR	32/15/20	300	54.2
	FSM2	None	35/9/19	n/a	n/a
		IEC	88/14/47	147	-67.9
		EEC	119/12/62	226	-13.8
		Armstrong	82/13/42	121	16.7
		TMR	131/27/72	279	66.7
Quadded Logic	Circuit1	None	3/0/2	n/a	n/a
		TMR	10/0/6	233	27.0
		QL	37/0/21	1133	14.8
	Circuit2	None	3/0/2	n/a	n/a
		TMR	10/0/6	233	27.2
		QL	29/0/16	866	1.9

* Overhead logic protected by TMR
† Mitigation was not applied to clocks and I/O due to limitations on the test fixture.

IX. RESULTS

For each of the different mitigation techniques we tested the reliability of one or two example circuits using that method. We also tested the reliability of those same circuits with just TMR (using triplicated voters). We did not use the same circuits for each method. Instead we picked sample circuits that had the best chance of producing reliability and/or cost results better than TMR.

For temporal redundancy we performed experiments on a 36-bit adder module using TSTMR and QTR. For state machine encoding we tried EEC, IEC and Armstrong's method on two different FSMs. The first FSM is a Mealy state machine with two inputs, one output, and eight states. The second FSM is also a Mealy state machine, but it has seven inputs, four outputs, 16 states and more complicated state transitions. For quadded logic we tested two arbitrary combinational circuits. Each has 10 inputs and one output. The first is an alternating plane of two-input AND and OR logic gates. The second is an arbitrary combination of two-input logic gates (i.e., AND, OR, XOR, NOR etc.). Because no automated approaches are available for implementing these techniques, our tests were limited to small, test circuits that can be modified by hand. If initial results demonstrate that these techniques do in fact provide improvements in reliability or a reduction in cost, tools for automating the insertion of these techniques may be created. Such tools would be used to test full applications and complex circuits.

The results, reported in Table III, suggest that *none* of the techniques evaluated provide greater reliability than TMR and that these techniques are often more costly than TMR. In fact, the following mitigation techniques actually decreased the design reliability: TSTMR, QTMR, EEC and IEC. This was the result of unprotected mitigation logic.

Despite the reliability decrease, TSTMR and QTR did have a much lower area overhead than TMR—only 71% and 58% more than the original circuit compared to 458% for TMR. As a follow-on experiment, the overhead logic was itself protected with TMR (indicated in Table III by an asterisk next to the design name) and tested again. In this case the circuits' reliability were comparable to the strictly TMR implementation, but the area increase jumped to 621% and 588%.

Armstrong's method and quadded logic did actually improve reliability, but not by as much as TMR. For the FSM2 design, TMR outperformed Armstrong's method in sensitivity reduction by 299% and only required 131% more area than the Armstrong implementation. In other words, TMR had much higher gains in reliability for a slightly larger area cost.

Quadded logic reduced the single-bit SEU sensitivity of circuit1 by 14.8% compared to 27% with TMR. For circuit2, quadded logic reduced the sensitivity only 1.9% while TMR reduced the sensitivity about 27%. Unlike Armstrong's method, quadded logic also required more area than TMR with 866% to 1133% area overhead compared to 233% for TMR. In other words, TMR yielded much higher reliability in terms of single-bit upsets at substantially smaller area costs.

X. CONCLUSION

In this work, we have evaluated three mitigation techniques in an attempt to identify a technique that is more cost effective at increasing reliability than TMR. This study evaluated the techniques of quadded logic, state machine encoding, and temporal redundancy and compared them in both area and SEU sensitivity to TMR. The results of our study suggest that *none* of the

techniques evaluated provide greater reliability than TMR and that these techniques are often more costly than TMR.

The first observation from these results is that the reliability of all of the techniques was worse than the reliability of TMR. The increased sensitivity is due to the mitigation logic added to the design. Unlike ASICs, the configuration of logic and routing structures in FPGAs are stored in SRAM cells and thus are sensitive to SEUs, in addition to SETs. As a result, any mitigation technique that adds unprotected mitigation circuitry will add to the SEU-sensitive cross section of the design. TMR does not experience this increase in sensitivity because the added redundant logic is protected by majority voters. All of the logic, including the original unmitigated logic, is protected from single event upsets due to the single error correction behavior of the majority voters. Like the use of voters in TMR, any SEU mitigation technique that adds additional logic to a circuit must itself be protected against SEUs.

The second observation from these results is that the area overhead of these mitigation techniques was usually *greater* than the area overhead required by TMR. These techniques were initially investigated due to the prospect of lower overhead. When mapped to FPGAs, however, these techniques proved to be more costly than TMR. The high overhead of these techniques can be attributed to the mismatch between the mitigation technique and the LUT-based architecture of the FPGA. These techniques require much larger input functions thus exponentially increasing the number of look-up tables used in the design. TMR, however, involved a linear increase in logic to implement the two additional levels of redundancy and voters. Any SEU mitigation technique mapped to FPGAs must be adapted to the LUT-based architecture prevalent in modern FPGA devices.

While the techniques described in this paper did not demonstrate improvements in FPGAs, it is likely that they will fare more favorably for ASICs and other non-volatile FPGAs. In fact, some of these techniques have been implemented in ASICs and have shown improvements in reliability in this technology. The reason these techniques are more favorable on ASICs is that the routing and logic functions are not susceptible to upsets that can last thousands of clock cycles as they are in FPGAs. The logic added to implement these techniques do not appreciably increase the dynamic sensitive cross section of an ASIC device as they do for SRAM-based FPGAs. While the added logic may be sensitive to single event transients (SETs), the effects of SETs are small compared to the overall improvements in reliability provided by the given technique.

Even though this study was not able to identify a technique that is more cost effective than TMR, it helped clarify several important issues pertaining to FPGA SEU mitigation. First, this study reaffirmed the importance of TMR as a SEU mitigation technique. To date, no other technique has been demonstrated that produces lower-cost reliability. Second, the results of this study emphasize the importance of adding SEU "insensitive" mitigation circuitry to a design. Techniques that add unprotected logic to a circuit may actually increase the sensitivity of the design. Third, this study suggests that mitigation techniques must be adapted to a LUT based architecture in order to provide cost effective SEU mitigation. We will continue to investigate other mitigation techniques, including variations of TMR, to identify improved methods for reducing the SEU sensitivity of FPGA designs in a radiation environment.

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] P. Samudrala, J. Ramos, and S. Katkoori, "Selective triple modular redundancy for SEU mitigation for FPGAs," in *Proc. 6th Annu. Int. Conf. Military an d Aerospace Programmable Logic Devices (MAPLD)*, Sep. 2003, C1.

[4] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin, "SEU-induced persistent error propagation in FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 52, no. 6, pp. 2438–2445, Dec. 2005.

[5] B. Pratt, M. Caffrey, P. Graham, E. Johnson, K. Morgan, and M. Wirthlin, "Improving FPGA design robustness with partial TMR," in *Proc. IRPS Conf.*, Mar. 2006.

[6] F. G. de Lima Kastensmidt, G. Neuberger, R. F. Hentschke, L. Carro, and R. Reis, "Designing fault-tolerant techniques for SRAM-based FPGAs," *IEEE Design Test Comput.*, vol. 21, no. 6, pp. 552–562, Nov.-Dec. 2004.

[7] Y.-M. Hsu, J. Earl, and E. Swartzlander, "Time redundant error correcting adders and multipliers," *Defect Fault Tolerance VLSI Syst.*, 1992.

[8] W. Townsend, J. Abraham, and E. Swartzlander, "Quadruple time redundancy adders," *Defect Fault Tolerance VLSI Syst.*, p. 250, 2003.

[9] R. Rochet, R. Leveugle, and G. Saucier, "Analysis and comparison of fault tolerant FSM architectures based on SEC codes," *Defect Fault Tolerance VLSI Syst.*, p. 9, 1993.

[10] N. Kumar and D. Zacher, "Automated FSM error correction for SEUs," in *Proc. MAPLD*, 2004.

[11] R. Wilcox and W. Mann, *Redundancy Techniques for Comp. Systems*. Washington, DC: Spartan Books, 1962.

[12] W. H. Pierce, *Fault-Tolerant Computer Design*. New York: Academic, 1965.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] J. Wells, "Automatic radiation hardening of FPGA designs using synthesis tools," *Electron. Eng. (London)*, pp. 63–66, 1998.

[19] Xilinx XTMR Tool [Online]. Available: http://www.xilinx.com/esp/mil_aero/collateral/tmrtool_sellsheet_wr.pdf

[20] N. Rollins, M. Wirthlin, P. Graham, and M. Caffrey, "Evaluating TMR techniques in the presence of single event upsets," in *Proc. 6th Annu. Int. Conf. Military Aerospace Programmable Logic Devices (MAPLD)*, Sep. 2003.

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

[22] S. Niranjan and J. Frenzel, "A comparison of fault-tolerant state machine architectures for space-borne electronics," *IEEE Trans. Reliability*, vol. 45, no. 1, pp. 109–113, Mar. 1996.

[23] D. B. Armstrong, "A general method of applying error correction to synchronous digital systems," *Bell Syst. Techn. J.*, vol. 40, no. 2, pp. 577–593, 1961.