A Comparison of Fault-Tolerant Memories in SRAM-Based FPGAs

Nathaniel Rollins, Megan Fuller, and Michael J. Wirthlin
NSF Center for High-Performance Reconfigurable Computing (CHREC)
Department of Electrical and Computer Engineering
Brigham Young University, Provo, UT. 84602
nrollins@byu.edu, mfuller4@byu.edu, wirthlin@ee.byu.edu

Abstract— This paper compares the effectiveness and cost of different fault-tolerant techniques for FPGA memories (BRAMs, LUTRAMs, and SRLs). TMR, parity with duplication, compliment duplicate (CD) with duplication, single-error correction/double-error detection (SEC/DED), and SEC/DED with duplication are the techniques used in this study to protect FPGA memories. Memory scrubbing is also added to each of these techniques. The effectiveness of each technique is measured by the number of sensitive bits in each design as well as the number of critical failures. A critical failure is defined as an upset whose effects can only be repaired through device reconfiguration. Cost is measured in terms of FPGA slices and BRAMs. This study finds that for BRAMs and LUTRAMs scrubbing with TMR provides the best protection. For SRLs scrubbing is unnecessary, and TMR provides the best protection. This study also provides a variety of reliability-area trade-off points with fault-tolerant techniques other than TMR.

TABLE OF CONTENTS

1 INTRODUCTION 1
2 FAULT MECHANISMS IN MEMORIES 2
3 MEMORY FAULT-TOLERANT TECHNIQUES 3
4 MEMORY SCRUBBING 5
5 MEMORY FAULT-TOLERANCE COMPARISONS ... 7
6 CONCLUSION 10
REFERENCES 11

1. INTRODUCTION

SRAM-based field programmable gate arrays (FPGAs) are a popular option for space-based applications because of their flexibility, reprogramability, and low application development costs. They can be reprogrammed while in-orbit to adapt to changing mission needs or correct design errors. However SRAM-based FPGAs are inherently sensitive to the effects of faults caused by high-energy particles. These upsets, also called single event upsets (SEUs), can occur in FPGA user memory bits or in FPGA configuration bits (which control user logic and routing). A lot of research and experimentation has identified techniques to make SRAM-based FPGA logic and routing reliable in the presence of SEUs [1–4]. But there has been limited research into the reliability of FPGA user memories.

Memory elements are an essential part of FPGA designs. FPGA memory elements include block RAMs (BRAMs), LUTRAMs, and SRLs. BRAMs provide single or dual-ported, wide, deep memories. LUTRAMs are smaller, faster memories built of FPGA LUTs that provides better access to memory contents. SRLs are shift registers built efficiently out of FPGA LUTs.

Each of these memories are useful in FPGA designs used for space-based applications, and thus require protection from SEUs. Previous memory reliability studies have focused only on BRAM protection [5] using TMR [6] and scrubbing [7]. But in addition to protecting BRAMs, fault-tolerant techniques must be used to protect LUTRAMs and SRLs.

There are a variety of techniques that could provide fault-tolerance for each of the FPGA memory elements. Triple modular redundancy (TMR) is a popular technique for FPGA logic [1–4] and BRAMs [5]. TMR requires a triplication of all resources and the addition of voting logic. Thus the area and power costs of TMR are at least 3x [8]. Other reliability techniques such as duplication with an error detection code (EDC) which have been applied to FPGA logic [2, 4, 9] may provide adequate protection at a reduced cost. Application specific integrated circuit (ASIC) memory protection techniques such as error control coding (ECC) [9, 10] may also provide less expensive protection.

This paper compares the effectiveness and cost of different fault-tolerant techniques for SRAM-based FPGA memories (BRAMs, LUTRAMs, and SRLs). The following fault-tolerant techniques are compared: TMR, interlaced parity [11] with duplication, complement duplicate (CD) [11] with duplication, single error correction/double error detection (SEC/DED) [11], and SEC/DED with duplication [9]. Additionally, each of these reliability techniques is augmented with the application of scrubbing [7]. The reliability of each of these techniques is measured in terms of single-bit upset sensitivity and critical failures. The cost is measured in terms of area. Despite the reduced significance of area as a cost in ASIC design, the cost of area is still significant for designs implemented in FPGAs. Performance is not considered

978-1-4244-3888-4/10/$25.00 ©2010 IEEE
This work was supported in part by the I/UCRC Program of the National Science Foundation under the NSF Center for High-Performance Reconfigurable Computing (CHREC).

as a cost since the focus of this paper is on FPGA reliability and not on high-speed fault-tolerant technique implementations.

This paper begins by identifying how upsets can affect FPGA memories. Next the fault-tolerant techniques used in this study are introduced. A discussion on scrubbing in each of the different FPGA memory structures follows. Finally, the study results show that although there are some cheaper alternatives to TMR, the most reliable way to protect SRAM-based FPGA memories is with TMR and scrubbing.

2. Fault Mechanisms in Memories

To motivate the need for memory protection techniques, this section considers the types of problems that SEUs can cause in FPGA memories. These problems are measured in terms of *sensitive bits* and *critical failures*. Problems are reduced with the application of memory reliability techniques, however they are rarely eliminated. This section also discusses how sensitivity and critical failures affect each type of FPGA memory (BRAMs, LUTRAMs, and SRLs) so that the effectiveness of each fault-tolerant technique can be evaluated.

Both ASICs and FPGAs are susceptible to SEUs in their user memories, but unlike ASICs, SRAM-based FPGAs are also vulnerable to faults within their configuration memory. FPGA configuration memory controls user logic and routing. This means that, unlike ASICs [12], FPGAs are also subject to faults in the logic and routing protecting the memory bits. So the goal of FPGA memory protection techniques is to prevent faults caused by upsets in either user memory content, or upsets in the logic and routing protecting the memory.

The effectiveness of different fault-tolerant techniques is measured, first in terms of sensitivity to single-bit upsets. A *sensitive* bit refers to a user memory content bit or an FPGA configuration memory bit that, when upset, causes erroneous memory output. Clearly, in the absence of any memory protection technique, every memory bit is sensitive. Every fault-tolerant technique used in this study *eliminates* sensitive bits caused by single-bit upsets in user memory. But to protect memory bits, added logic is required, which is itself sensitive. Thus effective fault-tolerant techniques must remove sensitivity in both the memory content and memory-protecting logic and routing.

In addition to sensitivity, the effectiveness of the memory reliability techniques is measured in terms of the number of *critical failures* caused by upsets in the memory bits and FPGA configuration memory bits. A critical failure is an upset whose effects are lasting, and can only be repaired through device reconfiguration. An upset that causes a critical failure is also considered sensitive, but not all sensitive bits cause a critical failure.

Critical failures are very similar to *persistent* errors [13] in FPGA logic. Persistent errors are SEU-induced errors that cannot be repaired without a global system reset. But unlike FPGA logic, FPGA memories are unaffected by reset signals. Thus critical failures refer to upsets whose effects cannot be repaired without FPGA reconfiguration.

It is possible for the effects of a critical failure to be overcome in user memories when the memory acts as a RAM. So to declare that an SEU-induced fault has caused a critical failure, a window of time must be defined to observe that the effects of the SEU have not been repaired. Corrupted memory locations can be naturally overwritten with new data, repairing upsets caused by an upset. The likelihood of this happening depends on the probability of writing to a given memory location, as well as the size of the critical failure window of observation. Thus it is more likely to observe critical failures in larger memories like BRAMs, which have a smaller probability of writing to any given memory location than smaller memories like LUTRAMs or SRLs.

Upsets in BRAMs

A critical failure will usually occur in an unprotected memory when the memory's input port signals are upset. For example, Figure 1 shows how upsetting the write enable port on a BRAM can have disastrous consequences. Consider a BRAM that is being used as an instruction memory for a soft-core processor [14]. When acting as an instruction memory, the BRAM behaves like a ROM. In other words, the write enable port should never be active. Figure 1 shows that if an SEU causes the write enable port to become active, the memory contents pointed to by the address port will be overwritten with the value on the data input port. If the address port steps through memory (as is common for an instruction memory), the contents of the entire memory can be overwritten, destroying the entire program. When no reliability techniques are present, the only way to recover from this kind of upset is by taking the FPGA off-line and reconfiguring it.

Upsets in LUTRAMs

Like BRAMs, critical failures can be caused in LUTRAMs when any of the input port signals are upset. However the consequences of upsetting the LUTRAM write enable port might not be as catastrophic as upsetting a BRAM write enable port. Figure 2 shows a 16 entry, 16-bit wide LUTRAM. If each of the 16 LUTs that make up the 16-bit wide LUTRAM has its own write enable signal, upsetting one of the write enable ports causes only one bit in one of the 16 words to be upset. Even if the LUTRAM address signal increments, only one bit in each of the 16-bit words is upset. Memory reliability techniques can detect and sometimes correct a single-bit error in a memory word, so this scenario is less severe than upsetting the BRAM write enable port.

However it is possible that the consequences of upsetting the write enable port on a LUTRAM are just as severe as upsetting the write enable on a BRAM. If the write enable ports of all 16 LUTRAMs are tied to the same write enable signal,

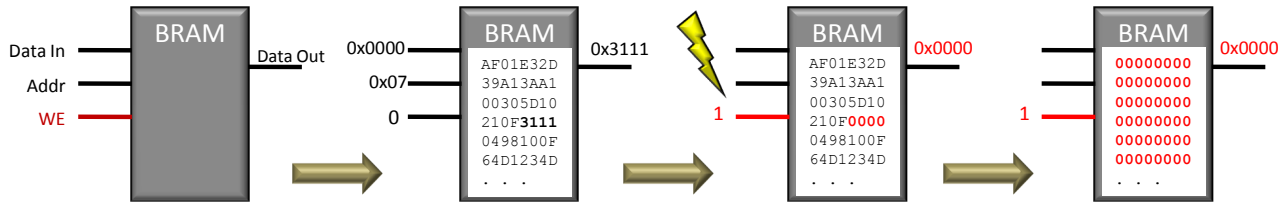


Figure 1. Upsetting a BRAM write enable can cause entire contents to be overwritten - resulting in a critical failure.

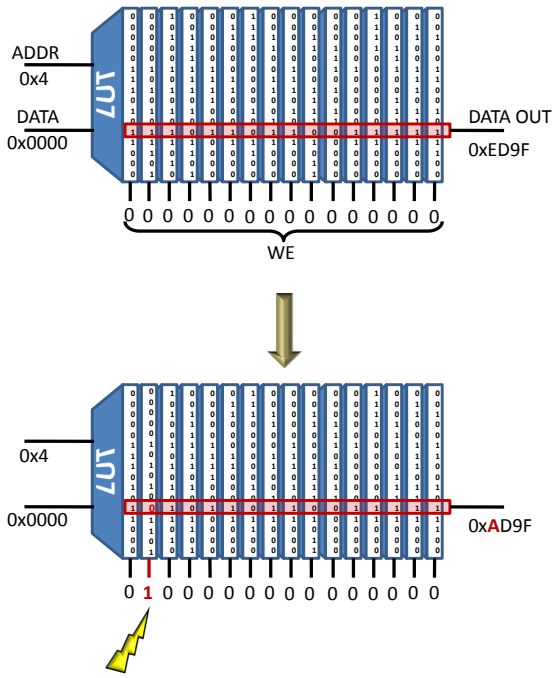


Figure 2. Upsetting a LUTRAM write enable can cause memory contents to be overwritten - resulting in a critical failure.

upsetting that signal will cause every bit in the 16-bit word to be upset. In this case, as the address signal increments, the entire memory contents are wiped out. Thus it is usually desirable to provide (if possible) separate write enable signals to each LUTRAM bit within a LUTRAM memory. However providing individual write enable signals will often require additional design area.

Upsets in SRLs

Critical failures in SRLs are different than in BRAMs and LUTRAMs. SRLs have no true ROM behaviour. Even if they continually cycle through the same memory contents, they have to write their output value back to their input. Thus an upset to any input signal (data input, address, or clock enable), will cause erroneous output in an unprotected SRL. If the SRL feeds its output back to its input, upsetting input signals is likely to cause a critical error. But if the SRL does not feed its output back to its input, a critical error can still occur by upsetting the clock enable signal. In this study both feedback and non-feedback scenarios are studied for each reliability technique.

In the absence of any memory protection, every FPGA memory bit is sensitive, and every upset to a memory content bit can cause a critical failure. But every memory fault-tolerant technique used in this study removes *all* critical failures and *all* sensitive bits due to upsets in the memory content. However the logic and routing required to protect FPGA memories is vulnerable to upsets. Thus, after protecting memory bits, the goal of FPGA memory reliability techniques is to reduce the sensitivity and critical failures in their own logic and routing.

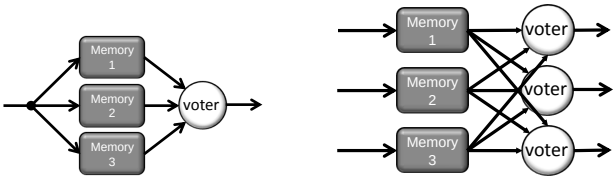
3. Memory Fault-Tolerant Techniques

The fault-tolerant techniques that are compared in this study fall under one of three general categories: TMR, ECC, or duplication with EDC/ECC. TMR is a tested and proven reliability technique for FPGA logic [6, 15], and has even been proposed for FPGA memory reliability [16]. Duplication with EDC/ECC has been proposed for FPGA logic [17] as well as for memories [2], but its effectiveness has not been proven, nor have its practical costs been evaluated. ECC techniques work well for ASICs [9, 10], but they may not be practical for FPGAs.

TMR

TMR is the most commonly used reliability technique used to protect SRAM-based FPGAs [6]. This technique simply triplicates the design and votes on the outputs of each triplicated version of the design (Figure 3(a)). If a fault occurs in one of the triplicated versions of the design, its faulty output will be out-voted by the other two correct designs. However since *all* logic and routing in an SRAM-based FPGA are susceptible to SEUs, the voter itself and all routing must also be triplicated in order to prevent errors due to a fault in the voter (Figure 3(b)).

Although triplicated voters and inputs provide better reliability, there are times when only single inputs and voters can be used. For example, a triplicated memory might be part of a non-triplicated system. In that case, the inputs and the outputs cannot be triplicated (Figure 3(a)). But even in a triplicated



(a)TMR with a single voter. (b)TMR with triplicated voters.

Figure 3. Triple modular redundancy (TMR) can be implemented with either a single, or triplicated voters.

system there must be a reliable way to bring the triplicated design back to a single design domain. This can be done on a reliable ASIC or radiation hardened device [18], or by using minority voters [6]. Also in a triplicated system, there must be enough I/O for triplicated signals to come in and out of the FPGA. When one or both of these criteria cannot be met, a single voter must be used on the FPGA (Figure 3(a)). This study investigates the reliability of both TMR with a single voter as well as TMR with triplicated voters.

Duplication with EDC

Another proposed FPGA-based reliability technique (Figure 4) combines duplication with an error detection code (EDC) [11]. Error detection codes encode memory words with redundant bits. The redundancy allows faults to be detected but not corrected (error correction codes can correct and detect errors).

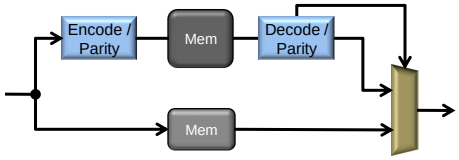


Figure 4. A memory protected by duplication and with an error detection code (EDC).

This study investigates two different error detection code techniques. First, a 2-bit interlaced parity detection scheme is used [11]. 2-bit interlaced, odd parity has the advantage of detecting any single-bit upset, a large percentage of adjacent-bit upsets, an all-zeros upset, an all-ones upset, and all upsets with an odd number of upsets in at least one of the two parity groups. For a 16-bit memory word, 2-bit parity requires only 12.5% more memory. Even after adding duplication, parity with duplication has a 112.5% memory increase - significantly less than the 200% required by TMR.

The second detection method uses a complement duplicate (CD) scheme [11]. CD detects any single-bit upset, 66% of double-bit upsets, and any multiple adjacent unidirectional upset. CD detects more upsets than 2-bit interlaced odd parity, but requires more code bits. Since CD adds an entirely duplicated memory, it has a 100% increase in memory, which when duplicated turns into a 200% increase. However CD has the advantage that its encoder is simply an inverter, which usually takes up no FPGA area. But its decoder requires more than just an inverter; it also requires a comparator to compare the original memory contents with the inverted contents of the CD memory.

At first glance, it appears that duplication with EDC might provide fault-tolerance at a lower cost than TMR since duplication is used instead of triplication. However, additional logic is required for this technique which is not required for TMR. Added logic is required for memory encoders, decoders, and comparators. This added additional logic is comparable in size to the logic required by TMR voters.

The memory encoder, decoder, comparator, and selection logic of duplication with EDC acts as a single point of failure just as the single voter with TMR (Figure 3(a)) acts as a single point of failure. Thus in order to eliminate this single point of failure, this added logic must be triplicated.

Even with triplicated logic, all the input signals still act as single points of failure with this technique (Figure 4). Also since there is only one output signal, any upsets on that signal will lead to a fault. But if the memory feeds in to a triplicated domain, errors on the output can be removed by triplicating the outputs and applying TMR voters to the outputs. Errors on the input cannot be outvoted this way - even if the inputs come from a triplicated domain. The triplicated domain will have to merge to a duplicated or single domain before leading to a memory protected by duplication with EDC.

There is no advantage to using duplicated input signals over a single set of input signals. Suppose for example, a duplicated address signal feed into the duplicated memories. If an upset changes the value of one of the two address signals, the memory word that is written into that memory will still be correctly encoded (even though it's written to the wrong address), or the word that is read from memory will still be correctly decoded (even though the wrong word is read) without any indication of an error. It is possible to *detect* differences between the two address signals by comparing their values, but it is not possible to tell which of the two is incorrect. Thus, in this study, when duplication with EDC is used to protect memories, input signals are not duplicated.

One study uses bitstream scrubbing to correct faults when duplication detects them [19]. When a fault is detected by duplication with compare, design execution pauses until a bitstream scrubber [20] corrects the fault. But pausing design execution can be costly and difficult, therefore this study does not consider the use of duplication with bitstream scrubbing to protect the logic and routing of memory encoders/decoders.

ECC

Error correcting codes (ECCs) are an effective way to protect ASIC memories [9, 10] and may also be effective for pro-

tecting FPGA memories (Figure 5). ECCs protect memories by encoding memory words with redundant bits. The redundancy allows faults to be both *detected* and *corrected*. If there are k data bits in the original memory word, and c redundant code bits are added to make the encoded word n bits wide ($n = c + k$), then an (n, k) code is used. The new n-bit encoded memory word is called a *code word*. This study uses a SEC/DEC encoding to protect a 16-bit memory word. To protect a 16-bit memory word, a (22, 6) SEC/DED code is used, creating a 22-bit code word.

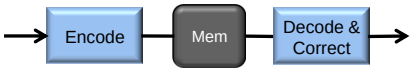


Figure 5. A memory protected by an error correction code (ECC).

A single-error correction / double-error detection (SEC/DEC) code is an efficient way to correct the effects of a single SEU in a memory word, and detect when there are two errors in the word. When there are more than two errors in a memory word, one of three things happen. Either the erroneous word is an incorrect but valid code word (thus no correction or detection occurs and the output is incorrect), or a single error is falsely corrected (and the output is incorrect), or a double error is detected. If a double error is reported when there are more than two upsets, the upsets will be caught, otherwise SEC/DED fails.

An ASIC memory reliability study determined that SEC/DED with duplication (i.e. duplication with ECC) is more effective than SEC/DEC by itself, TMR, or duplication with EDC [9]. To see if this holds true for SRAM-based FPGAs, this paper investigates both SEC/DED (Figure 5) and SEC/DED with duplication (Figure 4).

Like duplication with EDC, single points of failure can be removed by triplicating the outputs and logic of ECC encoders and decoders. Also like duplication with EDC, single points of failure on the input signals (leading to critical failures) cannot be removed. This study investigates both triplicated and non-triplicated versions of the SEC/DED and duplication with SEC/DED designs.

4. MEMORY SCRUBBING

Memory scrubbing [7] is technique that can be used in conjunction with a fault-tolerant technique to protect a memory from critical failures. Scrubbing refers to the *correction* of upsets within memories. Without scrubbing, upsets within memories can only be *masked* with effective fault-tolerant techniques, but the upsets remain in the memory. As upsets within the memory accumulate, it is less and less likely that a fault-tolerant technique will mask the error. Therefore scrubbing is essential to protect memories against SEUs over time. Scrubbing is also the only way to repair the effects of critical failures in memories.

Memory scrubbing is different than bitstream scrubbing [20]. Bitstream scrubbing is a reliability technique that uses readback [21] and partial reconfiguration to protect the logic and routing (but not the user memory contents) of FPGA designs. Bitstream scrubbing uses readback to compare portions of the FPGA bitstream to a *golden* copy of the bitstream. If there is ever a difference, that portion is reconfigured using partial reconfiguration. Alternatively CRC values of a portion of the bitstream are calculated and compared to a known CRC value for that portion. When the CRC values differ, that portion is reconfigured. Bitstream scrubbing is a great tool to help protect against upsets in FPGA logic and routing, but it cannot protect FPGA memories. Since the contents of memories change, there cannot be an a-priori known CRC value or golden content value that can be used for comparison. Thus memory scrubbers must be implemented independently of bitstream scrubbers.

Memory scrubbing refers to the deliberate repairing of memory words by a memory scrubbing module. A scrubber can work either *deterministically* or *non-deterministically*. A non-deterministic scrubber checks memory words only when they are read. This method is only practical for small memories such as LUTRAMs or SRLs, and requires less area than a deterministic scrubber. But non-deterministic scrubbing does not work well with BRAMs, since it would allow a large number of upsets to potentially accumulate if all memory locations are not read on a regular basis. BRAMs use deterministic scrubbing which *regularly* checks all memory locations for upsets. In this study BRAMs are deterministically scrubbed.

Scrubbing BRAMs

Adding scrubbing to TMR is a common fault-tolerant technique used to protect FPGA BRAMs [7]. Figure 6 shows the TMR BRAM scrubber implemented in this study. Although this subsection discusses how a TMR scrubber protects BRAMs, most of the concepts also apply to ECC BRAM scrubbers and duplication with EDC/ECC BRAM scrubbers. This study adds scrubbing to all three of these memory fault-tolerant techniques.

In order to apply scrubbing to BRAMs, an additional memory port is required. Thus if a design uses single-ported memories, the addition of scrubbing turns them into dual-ported memories. FPGA BRAMs can only be used as single or dual-ported memories [21], thus if a memory is already being used as a dual-ported memory, scrubbing may not be viable. This study assumes the use of single-ported BRAMs. Thus scrubbing is easily provided with the use of the second port.

To properly apply scrubbing to FPGA memories, there are implementation details that must be considered. First in order for every BRAM address to be scrubbed in a deterministic manner, a triplicated counter cycles through the entire address space of all three BRAMs, continually scrubbing their contents. It is essential that this counter be triplicated in a reliable way [1] so that the memory location being scrubbed

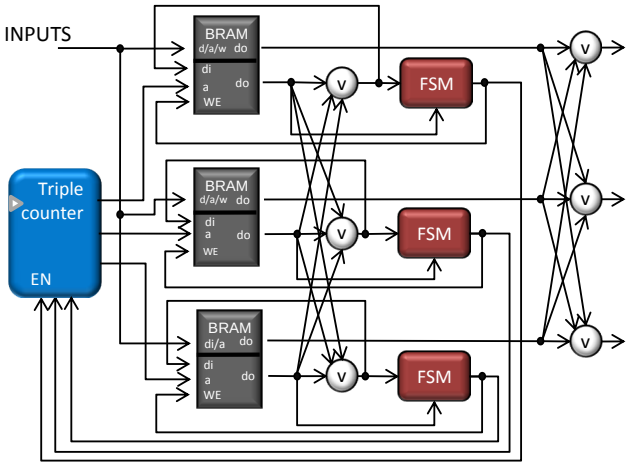


Figure 6. A BRAM protected by TMR with scrubbing.

receives the correct data. BRAM contents are only scrubbed when there is an upset detected in one of the BRAM words.

For ECC BRAM scrubbers and duplication with EDC/ECC scrubbers, when an error is detected at a memory address pointed to by the triplicated scrub address counter, instead of only scrubbing that memory location, the entire BRAM contents are scrubbed. Unlike the TMR BRAM scrubber, the entire BRAM contents are scrubbed because not all upsets are detectable by these other fault-tolerant techniques. Although these fault-tolerant techniques are guaranteed to detect all single-bit errors, they are not guaranteed to detect all multi-bit errors (except for duplication with CD). Thus if an all zeros, or all ones error occurs at a memory location (which can occur if the write enable signal is upset) it may not be caught. But if an upset is detected at some other memory address, the all zeros or all ones error will be scrubbed out since the entire BRAM contents are scrubbed when any error is detected.

The next implementation detail concerns the triplication of BRAM write enable signals. The logic controlling the assertion of *each* of the triplicated (or duplicated) BRAM write enable signals must be completely separate from each other. This detail provides protection against SEUs causing critical failures (Figure 1). If the same logic controls the write enable signals of two BRAMs, then a single SEU could overwrite the contents of both BRAMs. Scrubbing would then overwrite the third BRAM with the invalid contents in the other two BRAMs. On the other hand, when the write enable logic of each BRAM is kept separate, if an SEU causes a critical failure in one of the BRAMs, the effects of the critical failure will be corrected with scrubbing, and the effects of the critical failure are never felt.

Finally, BRAM Scrubbers must also consider address conflicts. An address conflict arises when the BRAM scrubber attempts to scrub the memory contents at an address that is already being accessed by the other BRAM port. Some FPGA architectures will allow the scrubber to scrub the memory contents if the other port is simply performing a read at that address [21]. But an address conflict always occurs if the other port is performing a write at the same address as the scrubber. When this happens, the write takes precedence since it will repair that memory location.

Scrubbing LUTRAMs

In this study LUTRAMs are non-deterministically scrubbed. Figure 7 shows an example of a duplication with EDC/ECC LUTRAM scrubber. Although only the duplication with EDC/ECC scrubber is shown, the design details of this non-deterministic scrubber also apply to TMR LUTRAM scrubbers and ECC LUTRAM scrubbers.

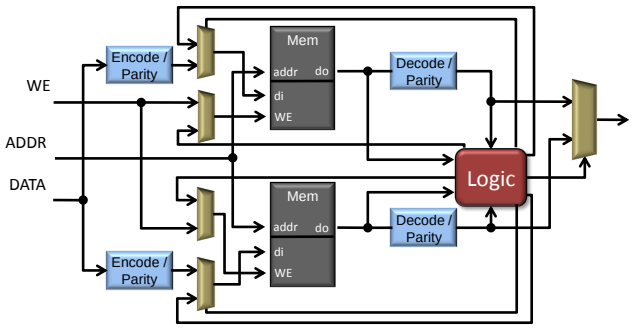


Figure 7. A LUTRAM protected by duplication with EDC/ECC and scrubbing.

LUTRAM memory locations are only scrubbed when they are read during normal design execution (non-deterministic scrubbing). Unlike BRAM deterministic scrubbers, all LUTRAM locations are not regularly scrubbed. This means that the scrubbing logic is smaller, but it also means that upsets could accumulate in memory. But non-deterministic scrubbing is realistic for LUTRAMs since there are only a small number of memory locations. In other words there is little chance that upsets will collect in memory locations before they are either read and scrubbed, or repaired by a memory write.

Just like BRAM scrubbers, in order for LUTRAMs to be scrubbed an additional memory port is required. In order to simplify this study, only single-ported LUTRAMs are considered. Thus the second port is available for a scrubbing module. Although scrubbing may be possible without the need of the second port, it facilitates the logic required for scrubbing.

Also like BRAM scrubbers, the logic controlling the LUTRAM write enable signal must be properly designed in order to prevent critical failures. The write enable signals leading to each duplicated domain (or triplicated domain in the case of TMR LUTRAM scrubbers) must be completely independent. Further, within a domain it is possible to have either a single write enable signal for the entire LUTRAM memory,

or a write enable signal for each single-bit LUT of the memory (Figure 2). Some fault-tolerant techniques such as TMR and CD with duplication do not need more than just a single write enable signal. Other techniques such as parity with duplication have no way of determining which write enable signals to assert, and therefore use a single write enable signal. Finally some techniques such as SEC/DED or SED/DED with duplication can take full advantage of multiple write enable signals. In this study all LUTRAM scrubbers use a single write enable signal except for the SEC/DED scrubbers. SEC/DED with duplication works well enough with a single write enable signal that it is not worth the extra area to have multiple signals.

Unlike BRAM scrubbers, there is no need to check for address conflicts. Although LUTRAMs are dual-ported, one port is used for reading and one is used for writing. Thus there can never be a write conflict. Also, LUTRAMs do not suffer from the same problem that BRAMs do in some FPGA architectures when a read and a write are requested from the same port. This means that there is less logic required to build a LUTRAM scrubber.

Scrubbing SRLs

At first glance, scrubbing SRLs seems difficult since, unlike LUTRAMs and BRAMs, there are no dual-ported SRLs. This means that deterministic, scrubbing cannot be performed the way it can be for BRAMs or LUTRAMs. Also, to access any SRL value other than the one available on the output, the SRL contents must be cycled through. Despite these difficulties, it is possible to build a relatively simple non-deterministic scrubber (Figure 8).

Figure 8 shows an example of an EDC with duplication SRL scrubber. When the EDC in one of the two SRLs detects an error, the contents of the other SRL are used to scrub the faulty SRL using a clock that is $n + 1$ times faster than the normal clock (when n is the length of the SRL). The first n cycles scrub the SRL contents, and the extra cycle is used to perform a write when the clock enable signal is high. The duplicated SRL also feeds its own data back into itself in order to preserve its contents. A TMR or ECC SRL scrubber can be similarly implemented.

Although the ability of the scrubber shown in Figure 8 (as well as a TMR and ECC version) have been verified, this study finds that there is no need to scrub SRLs. The natural operation of SRLs which causes new values to be written to the SRL is good enough to protect SRLs from critical errors. Even when not written to on every clock cycle, SRL scrubbing is unnecessary.

5. Memory Fault-Tolerance Comparisons

This section compares the fault-tolerance of FPGA memories protected with different reliability techniques, and evaluates each of their costs. The fault-tolerant techniques applied include TMR, duplication with EDC (parity and CD), ECC (SEC/DED) and ECC with duplication. Fault-tolerance is measured in terms of the number sensitive bits, and the number of critical failures. The cost of each fault-tolerant technique is measured in terms of FPGA slices, and BRAMs (where applicable).

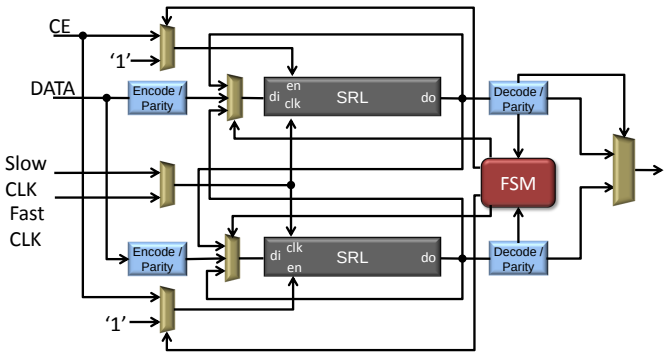


Figure 8. An SRL protected by duplication with EDC and scrubbing.

Fault-injection tests are run on SEAKR's XRTC board. The XRTC board contains 3 FPGAs: the configmon, the funcmon, and the DUT (Figure 9). The DUT (design under test) holds the mitigated FPGA memory design that will injected with faults. The DUT is a Xilinx Virtex4 FPGA. The funcmon (functional monitor) holds a *golden* copy of the memory design and compares the outputs of the golden and DUT memories. The configmon (configuration monitor) is responsible for injecting the faults into the DUT bitstream, and communicating with a host through a USB bus. The configmon configures both the funcmon and DUT FPGAs and reports sensitive bits and critical failures back to the host.

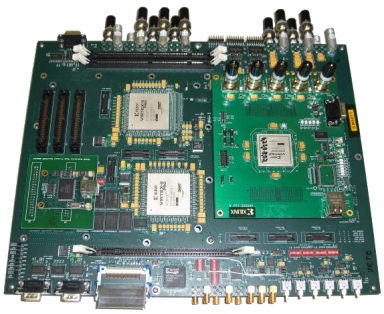


Figure 9. Fault-injection is performed using SEAKR's XRTC board.

In order to determine the sensitivity of each design, the configmon design upsets each bit of the DUT bitstream. After a bit is upset, the configmon allows the DUT to cycle x times (where x is a variable). During the x cycles, the memory on the DUT and the golden memory on the funcmon are fed pseudo-random inputs. If the output of the memory on the DUT ever differs from the output of the golden memory, an

error signal is sent back to the host, indicating which bitstream bit is sensitive. Every memory address is compared in order to ensure that all errors are caught. After x cycles, the bitstream bit is repaired, and the next bit is upset. For all BRAM reliability tests, $x = 5000$. For all LUTRAM and SRL tests, $x = 200$.

Critical failures are observed by allowing the DUT to cycle an extra x cycles after repairing a bit, and before upsetting the next bit. Before these second x cycles, the funcmon allows the DUT to first cycle y times (where y is a fraction of x) before comparing the DUT and golden memories. These y cycles provide a window of opportunity for upsets in the memory to be repaired. If, after the y cycles, any differences between the DUT and golden memories are observed, a critical failure is indicated to the host. After a critical failure is observed, the DUT is reconfigured, and the golden memory is reinitialized. For all BRAM tests $y = 2000$, and for all LUTRAM and SRL tests $y = 50$.

When comparing reliability techniques it is important to have a baseline to compare effectiveness and cost. For each FPGA memory type (BRAM, LUTRAM, and SRL) the sensitivity, number of critical failures, and area cost of an *unprotected* memory is provided to compare the values of each fault-tolerant technique against.

To test the effectiveness of each of the fault-tolerant techniques on each of the FPGA memory types, different sets of tests are defined. First, all of the reliability techniques are applied to BRAMs and LUTRAMs while acting as a ROM, and then again while acting as a RAM. For SRLs, all of the fault-tolerant techniques are applied, first when the SRL outputs are fed to its inputs, and then when random inputs are fed to the SRL. For all three memory types, comparisons are done first among non-triplicated, non-scrubbing techniques, and then among triplicated, non-scrubbing techniques. For BRAMs and LUTRAMs, comparisons are also done among triplicated scrubbing techniques.

BRAM Fault-Tolerant Technique Comparisons

For each BRAM reliability design, 16-bit wide memory words are used. With 16-bit memory words, a Xilinx Virtex4 BRAM can hold 1024 words. The Virtex4 BRAM will actually hold 1024 18-bit words, but only 16 bits of the 18 are used in the baseline design. The extra two bits per word allow parity to be added for no additional area cost. However other error coding techniques such as CD and SEC/DED will require additional BRAM area.

The cost for each BRAM protection technique (with and without scrubbing) is shown in Table 1 in terms of BRAM area. The table shows the number of memory bits that each technique requires, and compares that cost to the original baseline design. Unfortunately, since BRAMs contain a fixed number of memory words, very few reliability techniques can make efficient use of the BRAM area. For example, the SEC/DED encoding technique causes a 16-bit memory word to grow to 22-bits. Since there is no natural way to store 22-bit words in a Xilinx BRAM, that 22 bits is broken into two 11-bits halfs, and each half is stored in a separate BRAM. Since only 11-bits are being used in each word of the BRAM, the other 7 bits per word are wasted. Only the parity, CD, and TMR techniques use BRAMs efficiently.

Design	No Scrubbing			Scrubbing		
	BRAM Bits		BRAMs	BRAM Bits		BRAMs
Original	16384		1	16384		1
Parity Dup	34816	2.1x	2	36864	2.3x	2
CD Dup	49152	3x	3	65536	4x	4
SEC/DED	22528	1.4x	2	N/A	N/A	N/A
SEC/DED Dup	38912	2.4x	3	45056	2.8x	4
TMR	49152	3x	3	49152	3x	3
Xilinx ECC	23552	1.4x	2	N/A	N/A	N/A

Table 1. The cost of fault-tolerance in terms of BRAM bits.

Table 2 compares the fault-tolerant techniques for the case when no triplication is performed in the ECC or duplication with EDC/ECC techniques, and when the TMR design has only a single voter (Figure 3(a)). In addition to these techniques, Xilinx's ECC-protected BRAM block [21] is included for comparison. None of the designs in Table 2 implement scrubbing. The table includes the area cost in slices, as well as the number of sensitive bits and critical failures for each technique.

BRAM Non-Triplicated, Non-Scrubbing						
Design	ROM			RAM		
	Slices	Sens. Bits	Crit. Fail	Slices	Sens. Bits	Crit. Fail
Original	0	16725	17	0	17103	306
Parity Dup	12	674	9	16	1184	787
CD Dup	12	329	17	28	817	306
SEC/DED	21	949	34	29	1463	829
SEC/DED Dup	34	1073	23	42	1128	527
TMR (1 Voter)	16	484	10	16	900	288
*Xilinx ECC	0	340	20	0	3673	1544

*Virtex4 ECC BRAM contents cannot be initialized and is therefore not useful as a ROM

Table 2. Non-triplicated, non-scrubbing reliability and cost comparisons for BRAMs.

Since each technique prevents all single-bit upsets within the memory content, all of the techniques have significantly fewer sensitive bits than the original unprotected BRAM. When treated as a ROM, most of the critical failures are due to upsets in the signals leading to the write enable signals of the BRAMs. When treated as a RAM, most of the critical failures are due to upsets in logic and signals leading to input ports. Table 2 shows that when treated as a ROM, the

parity with duplication, and CD with duplication techniques provide the best protection for the lowest cost. Although the Xilinx ECC BRAM provides the lowest cost (no added slices needed), the Virtex4 version of the primitive cannot be initialized, which makes it useless as a ROM. When a BRAM is treated as a RAM, TMR with a single voter provides the best protection at the lowest cost.

In general, triplicating the logic in the duplication with EDC/ECC and ECC techniques, and triplicating the TMR voters (Figure 3(b)) improves the reliability of memory protection. Table 3 compares the techniques when triplication is added to the designs from Table 2. This table shows that when no scrubbing is used TMR provides the best protection at the lowest cost. The sensitivity in all of these designs is due, mainly, to upsets in the input logic and signals.

BRAM Triplicated, Non-Scrubbing						
Design	ROM			RAM		
	Slices	Sens. Bits	Crit. Fail	Slices	Sens. Bits	Crit. Fail
Original	0	16725	17	0	17103	306
Parity Dup	37	611	9	41	655	495
CD Dup	40	503	5	56	1050	824
SEC/DED	66	548	34	100	845	736
SEC/DED Dup	102	544	23	136	1001	900
TMR (3 Voters)	34	135	10	34	494	110

Table 3. Triplicated, non-scrubbing reliability and cost comparisons for BRAMs.

Table 3 shows that even with TMR and triplicated voters and inputs, not every sensitive bit is eliminated. Some of the sensitivities in each fault-tolerant technique design in all of the tables correspond to single FPGA configuration bits that affect multiple routes [4]. Although this study does not focus on eliminating these kinds of sensitivities, in order to reduce them the memory elements in each design are hand placed. Even though hand placing the memories reduces sensitive bits which affect multiple routes, the sensitivities reported in every table will include these kind of sensitive bits.

In order to reduce, or even eliminate critical errors, scrubbing is required. Table 4 compares different reliability techniques that implement scrubbing. The logic and voters of all of these designs are also triplicated. There is no SEC/DED scrubber in this table since SEC/DED on its own cannot overcome critical failures due to upsets on the write enable port (Figure 1). This table leaves not doubt that TMR with scrubbing provides the best protection for BRAMs.

LUTRAM Fault-Tolerant Technique Comparisons

Like BRAMs, each fault-tolerant LUTRAM technique design uses 16-bit memory words. But instead of 1024 memory locations, each design uses a 16 entry LUTRAM. Unlike BRAMs,

BRAM Triplicated, Scrubbing						
Design	ROM			RAM		
	Slices	Sens. Bits	Crit. Fail	Slices	Sens. Bits	Crit. Fail
Original	0	16725	17	0	17103	306
Parity Dup	106	568	15	113	1895	1630
CD Dup	112	393	10	127	1253	856
SEC/DED Dup	181	525	180	297	735	616
TMR (3 Voters)	103	1	0	102	6	0

Table 4. Triplicated, scrubbing reliability and cost comparisons for BRAMs.

LUTRAMs do not suffer from inefficient memory use, and all area costs are determined by slice count.

The reliability and cost for the non-triplicated, non-scrubbing fault-tolerant LUTRAM techniques are shown in Table 5. The TMR technique used in these comparisons does not triplicate voters (Figure 3(a)). When treated as a RAM, all critical failures are repaired through natural memory writes. But these critical failures cannot be repaired when treated as a ROM. When treated as a ROM or a RAM, CD with duplication provides the best reliability at a reasonable cost.

LUTRAM Non-Triplicated, Non-Scrubbing						
Design	ROM			RAM		
	Slices	Sens. Bits	Crit. Fail	Slices	Sens. Bits	Crit. Fail
Original	8	1113	55	8	1430	0
Parity Dup	28	341	24	32	420	0
CD Dup	36	308	2	36	387	0
SEC/DED	32	617	61	40	1056	0
SEC/DED Dup	53	553	34	60	617	0
TMR (1 Voter)	40	448	12	40	384	0

Table 5. Non-triplicated, non-scrubbing reliability and cost comparisons for LUTRAMs.

Triplicating logic and voters (Figure 3(b)) generally improves the sensitivity of these designs. Table 6 compares the fault-tolerant techniques when triplication is added to the designs in Table 5. Again, CD with duplication provides the best reliability at a reasonable cost.

In order to eliminate critical failures, scrubbing and triplication are combined. Triplicated scrubbing designs are compared in Table 7. Like BRAMs, this table shows that combining TMR and scrubbing provides the best protection with a relatively low cost.

When treated as a ROM, Table 7 shows that sometimes critical failures are not eliminated. In the case of the TMR scrubber, the one critical failure occurs at a location where two

LUTRAM Triplicated, Non-Scrubbing						
Design	ROM			RAM		
	Slices	Sens. Bits	Crit. Fail	Slices	Sens. Bits	Crit. Fail
Original	8	1113	55	8	1430	0
Parity Dup	64	472	26	78	310	0
CD Dup	76	323	2	78	241	0
SEC/DED	102	267	60	139	834	0
SEC/DED Dup	119	495	18	156	181	0
TMR (3 Voters)	56	409	7	58	1	0

Table 6. Triplicated, non-scrubbing reliability and cost comparisons for LUTRAMs.

LUTRAM Triplicated, Scrubbing						
Design	ROM			RAM		
	Slices	Sens. Bits	Crit. Fail	Slices	Sens. Bits	Crit. Fail
Original	8	1113	55	8	1430	0
Parity Dup	106	126	8	140	367	0
CD Dup	136	278	0	173	595	0
SEC/DED	110	95	1	158	284	0
SEC/DED Dup	255	42	2	338	207	0
TMR (3 Voters)	124	2	1	145	25	0

Table 7. Triplicated, scrubbing reliability and cost comparisons for LUTRAMs.

of the triplicated clock signals run very close to each other. It is likely that upsetting this bit affects both of these clock routes [4]. The other scrubbing designs that have only one or two critical failures might also be caused by a single bit affecting multiple domains. These failures can be removed by constraining the placement and routing of the designs more carefully.

SRL Fault-Tolerant Technique Comparisons

Unlike BRAMs and LUTRAMs, SRLs cannot operate as a ROM. Thus instead of including ROM and RAM modes for each fault-tolerant test, SRLs operate in *feedback* mode and *non-feedback* mode. In feedback mode, the SRL outputs are fed back to its inputs. In non-feedback mode random inputs feed into to the SRL. Also unlike BRAMs and LUTRAMs, the address of an SRL determines the length of the SRL. In this study only static addressing is considered. Like the LUTRAM reliability designs, the SRL designs use a 16 entry, 16-bit wide SRL. Thus the address value is fixed for a length of 16 bits.

The non-triplicated, non-scrubbing SRL memory protection comparisons are shown in Table 8. In feedback mode, all of the designs are susceptible to critical failures caused by upsets to either the input or output logic and routing. In this mode, Table 8 shows that some techniques actually reduce reliability. In this mode TMR provides the best protection. In non-feedback mode, TMR provides the best protection, but parity with duplications provides relatively good protection for the lowest cost.

SRL Non-Triplicated, Non-Scrubbing						
Design	Feedback			Non-Feedback		
	Slices	Sens. Bits	Crit. Fail	Slices	Sens. Bits	Crit. Fail
Original	16	1002	584	16	617	0
Parity Dup	48	1023	578	49	319	0
CD Dup	68	1080	626	76	336	0
SEC/DED	51	677	412	51	483	0
SEC/DED Dup	76	637	458	79	493	0
TMR (1 Voter)	64	325	113	64	207	0

Table 8. Non-triplicated, non-scrubbing reliability and cost comparisons for SRLs.

When triplication is added to the non-scrubbing designs, Table 9 shows that all critical failures are eliminated when the SRL is in non-feedback mode. In feedback mode, all critical failures are eliminated when TMR is used. This is true even though the clock enable port is not always asserted (the clock enable port value is pseudo-randomly determined). Since all critical failures are eliminated, no SRL scrubbing designs are provided in this study. With triplication in feedback and non-feedback modes, TMR provides the cheapest protection, TMR provides complete protection at the lowest cost.

SRL Triplicated, Non-Scrubbing						
Design	Feedback			Non-Feedback		
	Slices	Sens. Bits	Crit. Fail	Slices	Sens. Bits	Crit. Fail
Original	16	1002	584	16	617	0
Parity Dup	89	963	796	87	307	0
CD Dup	102	887	708	116	245	0
SEC/DED	144	196	195	144	74	0
SEC/DED Dup	178	199	188	159	159	0
TMR (3 Voters)	80	0	0	80	0	0

Table 9. Triplicated, non-scrubbing reliability and cost comparisons for SRLs.

6. CONCLUSION

This study compares the reliability and cost of different FPGA memory fault-tolerant techniques. TMR, parity with duplication, CD with duplication, SEC/DED, and SEC/DED with duplication are the techniques used to protect BRAMs, LUTRAMs, and SRLs. Memory scrubbing is also added to each reliability technique. Reliability is measured in terms of the number of sensitive bits in a design, and the number of critical failures. This study defines a critical failure as an upset whose effects can only be repaired by reconfigur-

ing the FPGA. The cost of a reliability technique is measured in terms of area (slices and BRAMs).

This study finds that the best way to protect BRAMs and LUTRAMs is with a TMR scrubber. Despite the potential cost advantages that ECC and duplication with EDC/ECC sometimes have, they have a major weakness that TMR doesn't suffer from - they all have single points of failure on their input and output port signals. Also this study finds that scrubbing is unnecessary to protect SRLs. When an SRL is used in feedback or non-feedback mode, TMR provides complete protection.

Although TMR or TMR with scrubbing seems to provide the best protection, there are often cheaper alternatives whose reliability is almost as good. This study reveals these reliability-area trade-offs points provided by the different memory protection techniques. For example, a LUTRAM protected with CD and duplication, whose logic is not triplicated, provides good protection at a low cost. A SEC/DED scrubber with triplicated logic provides better protection at a greater cost, while a TMR scrubber provides the best protection, but at an even greater cost. The trade-off points provided by this study will be valuable when designing for space-based applications, where area and reliability are of concern.

REFERENCES

[1] N. Rollins, M. Wirthlin, M. Caffrey, and P. Graham, "Evaluating TMR techniques in the presence of single event upsets," in *Proceedings of the 6th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, September 2003.

[2] A. Tiwari and K. Tomko, "Enhanced reliability of finite-state machines in FPGA through efficient fault detection and correction," *Reliability, IEEE Transactions on*, vol. 54, no. 3, pp. 459–467, Sept. 2005.

[3] F. Kastensmidt, C. Filho, and L. Carro, "Improving reliability of SRAM-based FPGAs by inserting redundant routing," *Nuclear Science, IEEE Transactions on*, vol. 53, no. 4, pp. 2060–2068, Aug. 2006.

[4] L. Sterpone, M. S. Reorda, M. Violante, F. L. Kastensmidt, and L. Carro, "Evaluating different solutions to design fault tolerant systems with SRAM-based FPGAs," *Journal of Electronic Testing: Theory and Applications*, vol. 23, pp. 47–54, 2007.

[5] C. Carmichael and C. W. Tseng, "Correcting single-event upsets in virtex-4 platform FPGA configuration memory," Xilinx Corporation, Tech. Rep., March 13, 2008, xAPP988 (v1.0).

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

[7] D. McMurtrey, K. Morgan, B. Pratt, and M. Wirthlin, "Estimating TMR reliability on FPGAs using markov models," Brigham Young University Department of Electrical and Computer Engineering, Tech. Rep., 2007. [Online]. Available: http://hdl.handle.net/1877/644

[8] N. Rollins, M. Wirthlin, M. Caffrey, and P. Graham, "Evaluatiion of power costs in applying TMR to FPGA designs," in *Proceedings of the 7th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*, September 2004.

[9] L. Schiano, M. Ottavi, and F. Lombardi, "Markov models of fault-tolerant memory systems under SEU," in *Memory Technology, Design and Testing, 2004. Records of the 2004 International Workshop on*, Aug. 2004, pp. 38–43.

[10] J. Gaisler, "A portable and fault-tolerant microprocessor based on the SPARC v8 architecture," *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pp. 409–415, 2002.

[11] D. P. Siewiorek and R. S. Swarz, *Reliable computer systems (3rd ed.): design and evaluation*. Natick, MA, USA: A. K. Peters, Ltd., 1998.

[12] T. Ganesh, V. Subramanian, and A. Somani, "SEU mitigation techniques for microprocessor control logic," in *Dependable Computing Conference, 2006. EDCC '06. Sixth European*, Oct. 2006, pp. 77–86.

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

[14] K. Chapman, "Picoblaze 8-bit microcontroller for virtex-e and spartan-ii/iie devices," Xilinx Corporation, Tech. Rep., February 4, 2003, xAPP213 (v2.1).

[15] M. Wirthlin, E. Johnson, N. Rollins, M. Caffrey, and P. Graham, "The reliability of FPGA circuit designs in the presence of radiation induced configuration upsets," in *Proceedings of the 2003 IEEE Symposium on Field-Programmable Custom Computing Machines*, K. Pocek and J. Arnold, Eds., IEEE Computer Society. Napa, CA: IEEE Computer Society Press, April 2003.

[16] *Single-Event Upset Mitigation for Xilinx FPGA Block Memories, Xilinx Application Notes 962, v1.1*, Xilinx, Inc., March 2008.

[17] F. Lima, L. Carro, and R. Reis, "Designing fault tolerant systems into SRAM-based FPGAs," in *Proceedings of the 41nd Design Automation Conference (DAC 2003)*, June 2003, pp. 650–655.

[18] L. Rockett, D. Patel, S. Danziger, B. Cronquist, and J. Wang, "Radiation hardened FPGA technology for space applications," *Aerospace Conference, 2007 IEEE*, pp. 1–7, March 2007.

[19] E. Heiko, "Development of a fault tolerant softcore CPU for SRAM base FPGAs," Master's thesis, Heidelberg University, June 2009.

[20] "Correcting single-event upsets through virtex parital

configuration," Xilinx Corporation, Tech. Rep., June 1, 2000, xAPP216 (v1.0).

[21] "Virtex-4 FGPA user guide," Xilinx Corporation, Tech. Rep., December 1, 2008, uG070 (v2.6).

Nathaniel Rollins is currently an Electrical Engineering Ph.D. candidate at Brigham Young University. He received both his B.S. degree in Computer Engineering with a minor Mathematics, and his M.S. degree in Electrical Engineering from Brigham Young University in 2004 and 2007 respectively. He has worked in the reconfigurable computing lab of the Electrical Engineering department at BYU since 2001, and has taught a number of Electrical Engineering undergraduate classes at BYU. He has authored or coauthored a number of papers in the field of FPGA design reliability.

Megan Fuller graduated from Cibola High School in December 2007. She spent two summers as an intern at Los Alamos National Laboratories and is currently a junior majoring in electrical engineering at Brigham Young University, where she now works as a research assistant.

Michael Wirthlin is currently an Associate Professor in the Department of Electrical and Computer Engineering at Brigham Young University in Provo, Utah. He has been actively involved in FPGA design for over 20 years and is active in the FPGA design, architecture, and tool research communities. He is currently a leading researcher in FPGA reliability modeling and fault tolerant design techniques. He and his students have developed tools for automatically inserting fault tolerant structures into FPGA designs and have tested these techniques on the orbiting Cibola Flight Experiment (CFE) satellite launched by Los Alamos National Laboratory. His research interests include FPGA reliability modeling, FPGA fault tolerant design techniques, Configurable Computing Systems, high-level synthesis, and computer-aided design for application-specific computing.