LA-UR- 04-7085

*Approved for public release;
distribution is unlimited.*

Title: Detection of Configuration Memory Upsets Causing Persistent Errors in SRAM-based FPGAs (Paper)

Author(s): D. Eric Johnson, Brigham Young University
Keith S. Morgan, Brigham Young University/ISR-3, LANL
Michael J. Wirthlin, Brigham Young University
Michael P. Caffrey, ISR-3, Los Alamos National Laboratory
Paul S. Graham, ISR-3, Los Alamos National Laboratory

Submitted to: 7th Annual Military and Aerospace Programmable Logic Devices International Conference, Washington D.C., 8-10 September 2004

Los Alamos
NATIONAL LABORATORY

Detection of Configuration Memory Upsets Causing Persistent Errors in SRAM-based FPGAs

D. Eric Johnson[1], Keith S. Morgan[1], Michael J. Wirthlin[1],
Michael P. Caffrey[2], and Paul S. Graham[2]

dej23@ee.byu.edu, ksm4@et.byu.edu, wirthlin@ee.byu.edu, mpc@lanl.gov, and *grahamp@lanl.gov*

[1]Department of Electrical and Computer Engineering, Brigham Young University, Provo, UT. 84602

[2]Los Alamos National Laboratory, Los Alamos, NM

Abstract *FPGA designers are becoming increasingly aware of fault tolerance issues in modern FPGA designs, especially designs destined for a radiation environment. We classify errors due to upsets within the configuration bitstream into two categories; namely, persistent and non-persistent. Persistent errors generally cannot be tolerated. However, non-persistent errors can be tolerated in certain types of designs as long as they are properly accounted for. We discuss situations in which non-persistent errors are acceptable, and describe a technique for the detection of upsets causing persistent errors within the configuration memory of an SRAM-based FPGA.*

1 Introduction

FPGAs are increasingly used in radiation harsh environments capable of causing single event upsets (SEUs). These SEUs may potentially modify an FPGA design, causing incorrect design behavior. In order to properly operate in a radiation environment, SRAM-based FPGA designs must employ some sort of SEU mitigation technique, whether it be half-latch removal [1], configuration memory scrubbing [2], or triple module redundancy (TMR) [3]. However, full mitigation techniques such as TMR can be, expensive in terms of power, area, and clock rate [4].

We are investigating the effects of configuration memory SEUs and gain insight into SEU mitigation costs. Certain FPGA designs cannot afford the costs of full mitigation. For these designs, alternate mitigation strategies need to be developed. These strategies may include tradeoffs between reliability, circuit area, power consumption, and clock rate.

In order to aid our investigation of SEU mitigation costs, we propose a new way of classifying the errors caused by configuration memory upsets, dividing them into two categories, namely persistent and non-persistent. We can take advantage of the properties of these errors in order to increase reliability at a low hardware cost. Certain types of errors, namely non-persistent errors, cause only brief interruptions in the correct operation of a design when configuration memory scrubbing is used. By employing a mitigation technique which allows these non-persistent errors while preventing persistent errors, we can increase the reliability in comparison to the non-mitigated design at a low additional hardware cost.

We will begin this paper with a background of fault testing techniques, both fault injection testing and radiation testing. Next, we will further develop the idea of persistent errors and our reasons for investigating them. We will then present the testbed we have created which searches for persistent errors in a design. Finally, we will show the results we have obtained through our persistence testing, and present our conclusions.

2 Configuration Memory Upsets

FPGAs perform very well at custom computationally intensive applications. For this reason, they are often used in specialized signal processing applications. Satellite missions especially serve to benefit from the capabilities provided by FPGAs. However, the radiation environment inherent in space missions can cause severe problems in SRAM-based FPGAs, particularly within the configuration memory.

2.1 Definitions

To aid in our discussion, we make the following definitions:

configuration memory upset a single event upset (SEU) within the configuration memory of an

FPGA, having the potential to *modify* an FPGA design and alter its behavior

error incorrect design behavior resulting from an SEU within either the configuration memory or user memory space

sensitive configuration bit a configuration bit which, when upset, results in an error

non-sensitive configuration bit a configuration bit which, when upset, does not result in an error

2.2 Configuration Upsets

In a radiation environment, any volatile memory source is susceptible to the effects of single event upsets (SEUs). Such an upset can be caused by a high energy particle interfering with the charge state of a memory latch. Typically, such upsets are observed as a change of state in a memory location; a stored logic value of 1 is inappropriately set to 0, or vice versa. When occuring within the configuration memory of an FPGA, this incorrect change of state in the memory is referred to as a configuration memory upset.

SRAM-based FPGAs use volatile memory to store the configuration information for a given design. The configuration memory defines the interconnection and functionality of all programmable logic blocks in an FPGA design. Because of the the volatile nature of the configuration memory of an FPGA, SEUs can cause configuration memory upsets. As a consequence of these configuration upsets, the FPGA design may actually be *modified.*

The occurance of a single event upset within an FPGA configuration memory is illustrated in Figure 1. This figure illustrates a typical logic block of an FPGA. The configuration memory defines the function of the logic block, in this case a four input AND gate followed by a flip-flop, as illustrated in part b of Figure 1. However, an SEU within the configuration memory may potentially change the function of the logic block, in this case, changing the function of the four input and gate to a four input or gate, as shown in part c of Figure 1. For this reason, it is highly important to mitigate against the effects of SEUs within the configuration memory of an FPGA.

The configuration memory is sub-divided into two categories; *sensitive* and *non-sensitive* bits. Sensitive bits are those bits which are crucial to the correct operation of a design. The composition of the set of sensitive bits is design dependent. Those bits which are sensitive generally correspond to parts of the FPGA which are utilized by a given design. Every bit within the configuration memory of an FPGA is classified either as sensitive or non-sensitive.

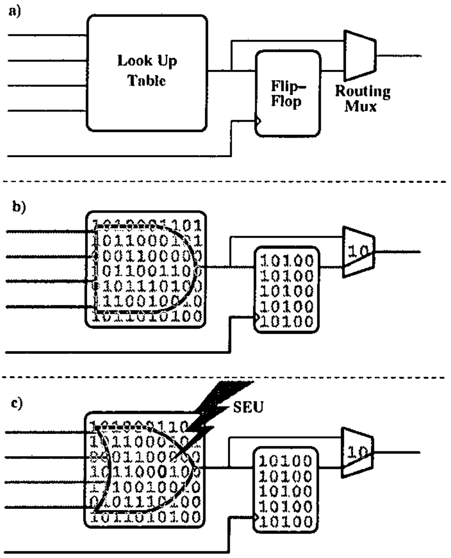

Figure 1: a) typical basic programmable block of an FPGA. b) this figure illustrates how the configuration memory defines the function of an FPGA. c) a fault within the configuration memory of an FPGA can alter the design.

2.3 Single Event Upset Simulation

We have developed a fault injection tool based on the SLAAC1-V computing board, which causes upsets within the configuration memory of an FPGA [5]. With this tool, a design specific map of the sensitive bits can be created. Faults are injected using the partial reconfiguration capabilities of the Xilinx Virtex parts on the board. The behavior of the design is subsequently monitored for errors while the fault is present, and then the fault is repaired. This process is repeated for every bit within the CLB space of the configuration memory.

The fault injection tool is sometimes referred to as a configuration SEU simulator because it simulates the introduction of faults within the configuration memory of an FPGA. In a true radiation environment, these faults occur as a result of SEUs due to high energy particles. With the configuration SEU simulator, we can inject faults into the configuration memory and observe their effects directly. The fault injection tool offers several advantages, including the ability to perform targeted tests. Factors such as fault introduction rate, fault persistence time and fault locations can all be controlled; this is not true for ground-based radiation tests, where the process is random. As such, the fault injection tool is very useful for FPGA designers wishing to validate the performance of configuration SEU mitigation techniques.

2.4 Fault Detection and Correction

Systems containing FPGAs intended for use in a radiation environment often find it necessary to employ some sort of strategy for fault detection and correction. Even an FPGA design employing an SEU mitigation technique which guarantees fail-safe operation in the presence of a single configuration memory upset is susceptible to multiple upsets over time if a fault correction strategy is not used. As such, a fault correction mechanism should be implemented for any radiation destined FPGA-based platform.

Two main forms of fault correction exist [2]. The first consists of fault detection and correction. A system utilizing such a method performs periodic readback operations on the configuration memory of the FPGA, searching for faults. When found, these locations are repaired.

The second form of fault correction is commonly referred to as scrubbing. The method simply consists of periodically refreshing the contents of the entire configuration memory of the FPGA. Regardless of when or where a fault occurs, it will eventually be repaired.

Neither technique can guarantee that a design will be 100% free of all configuration faults. The liklihood of the existence of a fault depends on how often the entire configuration memory can be either checked or refreshed.

In our discussion throughout the remainder of this paper, we will assume that all systems employ some sort of technique for fault correction.

3 Persistence

Errors occuring within an FPGA design due to configuration faults can generally be classified into one of two categories, namely *persistent* and *non-persistent* errors. In the following subsections we will explain in more detail what we mean by persistent and non-persistent errors. Further, we will show how we subdivide the category of sensitive bits into two categories based on the type of error that generally results when they are upset. These two subdivisions are *non-persistent* and *persistent* bits.

3.1 Non-Persistent Bits

Non-persistent errors are errors, which, given time and the employment of a fault correction strategy, will flush out of a system. They can be thought of as temporary errors. In other words, if a fault occurs within an FPGA design and is subsequently repaired, errors occuring as a result of that fault will be present for only a finite amount of time. *Non-persistent bits* are those bits in the configuration memory of an FPGA which, when upset, result in a non-persistent error.

cycle	a	b	p	status
0	0x2	0xb	0x16	OK
1	0xa	0x1	0x0a	OK
2	0x6	0x9	0x36	OK
SEU occurance				
3	0x2	0xb	0x96	ERROR
4	0xa	0x1	0x8a	ERROR
5	0x6	0x9	0xb6	ERROR
SEU repaired				
6	0x2	0xb	0x16	OK
7	0xd	0x3	0x27	OK

Figure 2: An upset within the configuration memory of a multiplier design with inputs a and b and output p may cause a temporary failure. Once the configuration memory is repaired, the operation of the multiplier returns to normal without the need of a reset.

A simple example of non-persistence is illustrated in Figure 2. This table shows the operation of a combinational multiplier design with 4-bit inputs a and b, and with an 8-bit output p. The operation of the multiplier over 7 cycles is shown along with the values of the inputs and output. Between cycle 2 and 3 of execution, an SEU occurs at a location which affects the operation of the design. As indicated by the status column, the behavior of the multiplier is incorrect for the next 3 cycles. We assume that a fault correction technique, such as bitstream scrubbing??, is being used for this design. Between cycles 5 and 6, the configuration memory of the design is repaired. Since the multiplier is a purely feed-forward design its operation returns back to normal, as indicated again by the status column.

3.2 Persistent Bits

Persistent errors are defined as those which will never flush out of a design unless a reset is applied. Even then, it is necessary to employ a fault correction strategy in order to correct the configuration memory upset caused by an SEU, otherwise the reset will not be able to fix the problem. Once the configuration upset has been repaired, the error condition will not go away without external intervention of some sort, such as a reset.

cycle	c	status
0	0x0	OK
1	0x1	OK
SEU occurance		
2	0xa	ERROR
3	0xb	ERROR
SEU repaired		
4	0xc	ERROR
5	0xd	ERROR
6	0xe	ERROR
...	...	ERROR

Figure 3: An upset within the configuration memory of a simple 4-bit counter design with output signal c may potentially cause a permanent error condition. Even when the configuration memory is repaired, the operation of the counter does not return to normal without a reset.

Persistent bits are those bits in the configuration memory which result in a persistent error when upset. The consequences of upsetting a persistent bit are more harsh than those resulting from upsetting non-persistent bits, and consequently more thought should probably be given to mitigating these types of bits within an FPGA design.

An example of a persistent error can be illustrated through the operation of a simple 4-bit counter design shown in Figure 3. Under normal operating conditions, this counter repeats the sequence in order from 0 to 15, or 0x0 to 0xf. From the table we see that the first two cycles of the output c are correct. However, between cycles 1 and 2 an SEU within the configuration memory occurs causing a stuck at one condition on the MSB of the counter. For the next two cycles the counter is in error. Between cycles 3 and 4 the SEU is corrected. However, because of the feedback nature of the counter design, the correct value of the counter never recovers without further intervention. The design has been repaired, but the current bad state of the counter will incorrectly effect al future states as well. The counter assigns the next state of the counter, seen in cycle 4, to be 0xc, based on the previous incorrect value, 0xb. In order to prevent such errors within feedback loops mitigation should be applied through design level techniques or else need to be corrected through external intervention, such as a system reset.

3.3 Persistence Tradeoffs

The characteristic differences between persistent and non-persistent bits allow us to explore tradeoffs in fault-tolerant systems. We can take advantage of the fact that non-persistent errors, when permitted in a system, will cause only momentary lapses in the reliability of an FPGA design. A design which can tolerate such errors can avoid the extra hardware costs involved with mitigating the effects of these errors.

However, designs which have not mitigated against persistent errors will possibly experience permanent error conditions. Unless these errors are mitigated, external intervention will be necessary. One of the simplest forms of such intervention is a system reset. However, such an extreme measure is often unacceptable for a given system. Consequentally, a system will benefit from the mitigation of such errors.

Because all systems cannot tolerate full mitigation techniques, such as exhaustive TMR, for reasons due to area, power or clock constraints, selective mitigation could be investigated. By mitigating against persistent errors only, we may see a relatively big increase in reliability at a low cost in terms of area. Further, certain types of systems may be able to tolerate temporary failures resulting from non-persistent errors, as such errors do not require a system reset in order to remove the error condition as long as some sort of fault correction strategy is employed. Such systems may especially benefit from the application of targetted mitigation techniques which remove the possibility of persistent errors from the design sensitive cross-section, because such a targetted application will in most cases be less expensive than full mitigation in terms of circuit area.

In order to investigate mitigation techniques targetted at removing the potential for persistent errors, we need a method of determining whether a design contains any persistent bits in its sensitive cross-section. Conceptually, we can categorize persistent and non-persistent characteristics by design styles. For example, feed-forward and datapath sections of a design typically contain no persistent bits. On the other hand, control and feedback portions of a design generally do contain persistent bits. Although this high level classification provides useful insight when making design decisions, an accurate analysis of persistence versus non-persistence is necessary in order to effectively apply mitigation techniques. For this purpose we have developed the persistence analysis tool.

4 Testing Methodology

In order to identify persistent bits in a design we have extended the fault injection tool described in Section 2. The tool now finds the subset of sensitive bits which are persistent. The tool operates in the following manner, also illustrated in Figure 4. First, a bit is corrupted in the configuration bitstream data, representing a fault, before the FPGA is programmed. The FPGA is programmed with the corrupt data and the output of the design is monitored for errors. Second, the design is allowed to operate for a finite length of time and then the fault is repaired (see time x in figure 4). If an output error occurs before the bit is repaired, the circuit is cycled for an additional amount of time in order to allow errors to potentially flush out of the system (see time y in figure 4). Finally, the behavior of the design is again monitored for a short amount of time in order to check for errors (see time z in figure 4). An error within this last window of time indicates with a certain probability that upsetting the given configuration bit causes persistent errors.

This process is repeated for every bit within the CLB space of the configuration memory. The lengths of time (x, y, and z) that the simulator monitors the output for errors are user defined parameters. Figure 4 shows these parameters with respect to upsetting and repairing a configuration bit. The pseudo-code for the tool's persistence test loop is shown in Figure 5. With this tool, a design specific map of the persistent bits can be created.

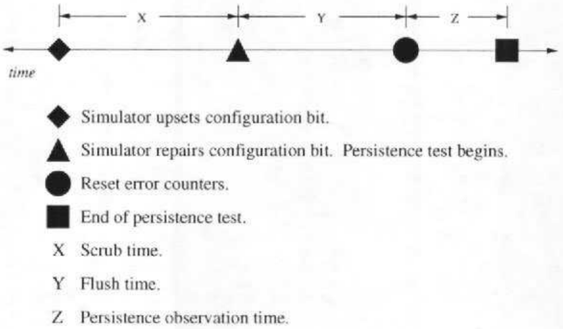
X
Y
Z
time
Simulator upsets configuration bit.
Simulator repairs configuration bit. Persistence test begins.
Reset error counters.
End of persistence test.
X Scrub time.
Y Flush time.
Z Persistence observation time.

Figure 4: Timeline of a persistence test showing the test parameters x, y and z relative to the events which occur during testing.

The degree of confidence with which we can say a bit is persistent is dependent on the design under test and the time parameters x, y, and z, as illustrated in Figure 4. A bit is marked as persistent if an output error occurs during time z. However, if at some point in the future the error flushes out of the system and never appears again, the bit was incorrectly marked as persistent. Conversely, a bit is marked as non-persistent if the output is absent of discrepancies during time z. However, if an error surfaces later and never flushes out of the system, the bit was incorrectly marked as non-persistent.

```
do {
     corrupt configuration bit
     wait for time 'x'
     test for output error
     repair configuration bit
     if output error occured
        mark bit location as sensitive
        wait for time 'y'
        reset error counters
        watch output for time 'z'
        if output error occured
           mark bit location as persistent
        endif
     endif
     reset design
} until all bits have been tested
```

Figure 5: Persistence Check Inner-Loop

5 Test Designs

For preliminary testing of our persistence tool, we have created two FPGA designs. Our first design is a contrived design, intended to emphasize the characteristics inherent in both feed-forward and feedback design styles. We chose this design hoping to see the presence of both persistent and non-persistent bits. The second design is a signal processing kernel implemented by researchers at Los Alamos National Laboratory. The purpose behind testing this design was to get a feel for the persistence characterists of a real world design.

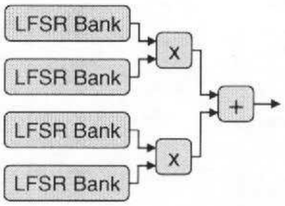

Figure 6: Synthetic design

We refer to our first contrived design as the synthetic design. The synthetic design is comprised of banks of wide LFSRs, whose outputs feed banks of pipelined multipliers (see Figure 6). The output of these multipliers is fed into an adder tree, and the final result is used as the design output. The LFSR portions of the design emphasize the characteristics typical of persistent design styles with feedback. The feed forward multiplier and adder tree exhibits characteristics more indicative of non-persistent design styles.

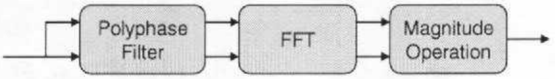

Figure 7: Snapshot recorder design

The signal processing kernel design is called the snapshot recorder design. This design filters incoming data through a polyphase filter bank, separating this data into 32 separate channels. The polyphase filter operation is followed by an FFT and a magnitude operation for each of the 32 channels received from the polyphase filter (see Figure 7). The feed forward nature of this design would suggest that it is dominated by non-persistent characteristics. It is only through more detailed analysis, such as that offered by persistence testing, however, that we can verify such a presumption.

6 Persistence Testing Example

We illustrate a real life example of persistent testing with the snapshot recorder design. Figure 8 illustrates the output obtained from the snapshot recorder design when given random input. In this figure, all 32 channels of the snapshot recorder are overlayed on top of each other. The x-axis indicates time in terms of clock cycles, and the y-axis represents the magnitude of the snapshot recorder output data.

Figure 9 shows the difference of the expected and actual output of the snapshot recorder for all 32 channels. This data was obtained from actually executing the design in hardware. A value of zero indicates that the snapshot recorder design is operating normally, whereas a non-zero result indicates a deviation from normal behavior.

Within Figure 9, we can see that an error condtion occurs. This error is due to an upset within the configuration bitstream memory of the FPGA design, which has been inserted using the fault injection tool. A short time after the fault is inserted, it is repaired. From the plot of the difference between expected and actual output, we see that the snapshot recorder design operation returns to normal. This particular instance illustrates the behavior of a non-persistent error.

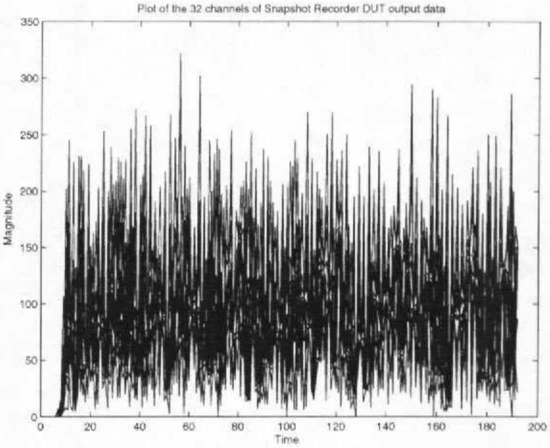

Figure 8: Snapshot recorder normal recorder output

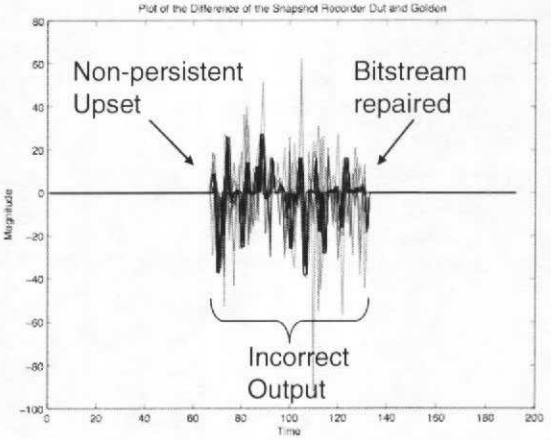

Figure 9: Snapshot recorder non-persistent error example. This graph shows the difference between expected and actual output. A value of 0 indicates correct operation.

Figure 10 also illustrates the difference between the expected and actual output of the snapshot recorder. Again, an error in the operation of the design occurs due to an upset within the configuration bitstream memory. Similar to the previous example, the upset is repaired shortly after its occurance. However, in this instance the design never recovers back to normal operation. We can clearly see that the error condition continues in spite of the configuration memory having been repaired. This is an example of a persistent error, and the particular configuration memory bit that was upset is an example of a persistent bit.

Design	Logic Slices	Sensitive Bits	Sensitivity	Persistent Bits	Persistence
Snapshot	5,775	568,660	9.79%	12,382	0.213%
Synthetic	5,924	421,874	7.26%	59,369	1.02%

Table 1: Preliminary results obtained with our persistence tool for two designs

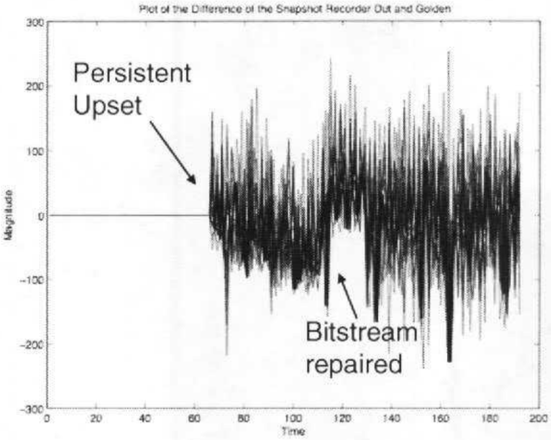

Figure 10: Snapshot recorder persistent error example. This graph shows the difference between expected and actual output. A value of 0 indicates correct operation.

7 Results

We have performed extensive testing on the synthetic and snapshot recorder designs with our persistence detection tool. The results from our tests are shown in Table 1.

In column 1 of the table is listed the number of logic slices occupied by each design. The Virtex part on which these designs were tested contains a total of 12,288 logic slices. These numbers give us an idea of the design size, a useful figure for comparing the relative sensitivy and persistence results.

The second column shows the number of sensitive bits found for each design, and column 3 shows the percentage of configuration memory bits which are sensitive. The total number of configuration memory bits on the Virtex part used for these designs is 5,810,048. This shows that on average the snapshot recorder design, with a sensitivity of 9.79%, is more susceptible to single event upsets than the synthetic design, with a sensitivy of 7.26%.

Finally, columns 4 and 5 show the number of persistent bits found and the persistence, respectfully. The persistence is defined as the percentage of configuration memory bits which were found to cause persistent errors. Attention should be given to the relatively small number of bits which cause persistent bits for these two designs.

It is interesting to note that the snapshot recorder design, which is more sensitive to SEUs than the synthetic design, actually contains less persistent bits than the synthetic design by almost a factor of 5. However, this is as expected due to the nature of the two designs. The snapshot recorder consists largely of a datapath feed forward structure, whereas the synthetic design was purposely constructed to contain several feedback structures. This feedback is present mainly in the LFSRs which drive the multipliers (see Figure 6). Feedback structures will typically cause errors to be persistent in nature.

We have also conducted a preliminary radiation test in order to verify the validity of the results obtained with our persistence tool. These results will be forthcoming upon the completion of their analysis.

8 Conclusions

We have developed a technique for the detection of persistent errors due to the upset of persistent configuration memory bits within an FPGA design. Preliminary results have been obtained, and indicate that we can use the persistence tool for detection of persistent bits within an FPGA design, as well as for the validation of mitigation techniques targetted at removing persistent bits from the sensitive cross section of a given design.

Not all FPGA designs require bullet-proof mitigation techniques. The extreme consequences of persistent errors indicate that they are an excellent candidate for targetted mitigation techniques. The relative increase in reliability due to the removal of persistent bits from the sensitive cross section is much higher than that gained from removing non-persistent bits, as the consequences of persistent errors are much more extreme than those of non-persistent errors. Additionally, we can infer from the results shown in Table 1 that the cost of applying mitigation for the persistent bits only should be small, as the overall number of persistent bits relatively small.

Mitigation techniques targetted at removing persistent bits from the design sensitive cross section can be validated using the persistence tool which we have developed. As part of our future work, we wish to develop techniques for detecting the presence of structures having characteristics common to persistent design styles, and applying mitigation to these structures.

References

[1] Paul Graham, Michael Caffrey, Eric Johnson, Nathan Rollins, and Michael Wirthlin. SEU mitigation for half-latches in Xilinx Virtex FPGAs. In *IEEE Transactions on Nuclear Science*, volume 50, pages 2139–2146, December 2003.

[2] Carl Carmichael, Michael Caffrey, and Anthony Salazar. Correcting single-event upsets through Virtex partial configuration. Technical report, Xilinx Corporation, June 1, 2000. XAPP216 (v1.0).

[3] Carl Carmichael. Triple module redundancy design techniques for Virtex FPGAs. Technical report, Xilinx Corporation, November 1, 2001. XAPP197 (v1.0).

[4] Nathan Rollins and Michael Wirthlin. Effects of TMR on power in an FPGA. submitted to MAPLD'04.

[5] Eric Johnson, Micheal J. Wirthlin, and Michael Caffrey. Single-event upset simulation on an FPGA. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, pages 66–73, June 2002.