Fault Tolerant ICAP Controller for High-Reliable Internal Scrubbing

Jonathan Heiner[1], Nathan Collins[2], and Michael Wirthlin[3]
Department of Electrical and Computer Engineering
Brigham Young University
Provo, UT 84602
(801) 422-4012
[1] Jonathan.Heiner@byu.net, [2] NathanDrewCollins@gmail.com, [3] Wirthlin@ee.byu.edu

Abstract—High reliable reconfigurable applications today require system platforms that can easily and quickly detect and correct single event upsets. This capability, however, can be costly for FPGAs. This paper demonstrates a technique for detecting and repairing SEUs within the configuration memory of a Xilinx Virtex-4 FPGA using the ICAP interface. The Internal Configuration Access Port (ICAP) provides a port internal to the FPGA for configuring the FPGA device. An application note demonstrates how this port can be used for both error injection and scrubbing [1]. We have extended this work to create a fault tolerant ICAP scrubber by triplicating the internal ICAP circuit using TMR and block memory scrubbing. This paper will describe the costs, benefits, and reliability of this fault-tolerant ICAP controller.

TABLE OF CONTENTS

1. INTRODUCTION

There is growing interest in using FPGAs within space systems due to low non-recurring engineering (NRE) costs, reconfigurability, and the large number of logic, arithmetic, and I/O resources found on modern FPGAs. Further, the ability to reconfigure the FPGA device after the spacecraft has launched allows the FPGA to be updated to changing mission goals or scenarios or even to fix faults within the system. A variety of projects have demonstrated the benefits of using FPGAs in a spacecraft [2], [3], [4].

While FPGAs offer a number of unique benefits for spacecraft electronics, FPGAs are susceptible to single event effects. FPGA devices contain a large number of internal memory cells that can be upset by high energy particles. FPGAs contain memory cells for user flip-flops, internal block memory, and for configuration memory. Upsets within the configuration memory are especially challenging as these upsets may change the behavior of the FPGA. Any system that incorporates FPGAs must provide a strategy for mitigating against these single-event upsets.

The most common mitigation approach for SEUs is to combine triple modular redundancy (TMR) [5], [6] with configuration scrubbing [7]. TMR involves the triplication of circuit resources and the use of majority voters to isolate any single upset within the circuit. Scrubbing involves the continuous configuration of the FPGA to "clean" upsets that occur. Scrubbing prevents the buildup of configuration upsets in order to significantly reduce the probability of getting a multibit upset. Together, these two techniques allow an FPGA to be used reliably in a variety of space environments.

While configuration scrubbing is an important component of a reliable FPGA system, it requires additional system resources and adds to the system complexity. To perform scrubbing, external "radiation hardened" circuits are required to manage the configuration process and load in the configuration data. Further, reliable memories are needed to hold the "golden" FPGA configuration bitstream.

This work demonstrates the use of an alternative form of configuration scrubbing using the internal configuration access port (ICAP). This form of configuration scrubbing eliminates the need for an external configuration memory or a rad-hard configuration controller. The technique used in this work is based on the ICAP scrubber application note published by Xilinx [1]. The design was modified to improve the scrubber reliability by applying TMR to the scrubber and utilizing a number of well-known techniques for improving the system reliability. This high-reliable internal scrubbing circuit was tested at the Crocker Cyclotron at UC Davis to demonstrate the ability to perform scrubbing internally and to demonstrate the improvements in reliability by using TMR and other techniques.

1-4244-1488-1/08/$25.00 ©2008 IEEE
IEEEAC Paper #1256, Updated 14 December 2007.

This paper will begin by reviewing the traditional scrubbing techniques. Next, the technique of internal scrubbing will be introduced along with an overview of the internal ICAP scrubber circuit. The test infrastructure used to test our design will be presented followed by a discussion of the results at our radiation test. The paper will conclude by summarizing our results and suggesting future work.

2. FPGA Configuration Scrubbing

As suggested earlier, configuration scrubbing is an essential component of any high-reliable FPGA based system. Like traditional memory scrubbing, configuration scrubbing "cleans" upsets within the configuration memory to insure a stable configuration memory. Configuration scrubbing prevents the buildup of multiple configuration upsets by identifying upsets and repairing them as soon as they are found. Previous studies have demonstrated significant improvements in overall reliability when scrubbing is used.

Scrubbing is usually performed by continuously writing the valid configuration memory into the device *over* the existing configuration data [7] using partial reconfiguration. If no upsets have occurred, each pass of the scrubber will simply write the same configuration data into the FPGA that is already present. If upsets have occurred, scrubbing will replace the upset configuration bits with correct values. By continuously writing the correct configuration data into the FPGA, scrubbing prevents the buildup of configuration upsets and limits upsets in the configuration bitstream to a short time based on the scrub rate.

Configuration scrubbing requires more infrastructure than that needed by traditional FPGA-based systems. As shown in Figure 1, external memory and a processor or configuration controller are needed to support the scrubbing process. The external memory is required to hold the "golden" configuration bitstream and the configuration controller is required to sequence through the partial reconfiguration steps. If sophisticated scrubbing techniques are used, a full programmable processor is required. Because these components manage the configuration of the FPGA, it is essential that they are protected from SEUs through appropriate mitigation or rad-hard by design techniques.

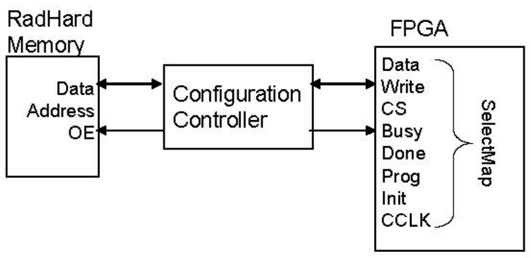

Figure 1. Architecture of traditional scrubbing circuitry

Configuration scrubbing can occur at a fairly fast rate and is only limited by the speed at which the FPGA can be configured. Modern FPGAs provide wide and high-speed interfaces to the configuration port to facilitate high-speed configuration. The Virtex-4 FPGA provides a 32-bit internal configuration port operating up to 100 MHz to provide a 50 MB/sec configuration interface. With a 7.7 Mbit to 15.3 Mbit bitstream, the FPGA can be completely scrubbed in 24 to 278ms [1] given a single SEU per scan. The rate of scrubbing can be slowed, if necessary, to avoid over scrubbing in low-upset environments.

There are several variations of scrubbing that have been proposed and demonstrated [8]. The scrubbing technique described above has been referred to as "blind scrubbing" since it configures the FPGA whether or not upsets have occurred (i.e. blindly). A variation of blind scrubbing is called "readback with correction". In this mode, the internal configuration memory is read back and compared with a golden bitstream. If a difference between the bitstreams is found, the configuration frame with the faulty bit is repaired through configuration.

3. Internal Scrubbing

Another style of scrubbing that has been proposed relies on the internal configuration access port or ICAP [1]. This form of scrubbing is performed by configuring the FPGA from *within* the device using the ICAP. This scrubbing strategy relies on the internal ECC block within the FPGA to identify faults and using the SECDEC code to correct the errors. Scrubbing internally, using the ICAP, removes the need for external circuit components, additional I/O pins, and radiation hardened configuration memories. The purpose of this work is to investigate the use of ICAP scrubbing and demonstrate a low-cost, high-reliable internal ICAP scrubbing circuit.

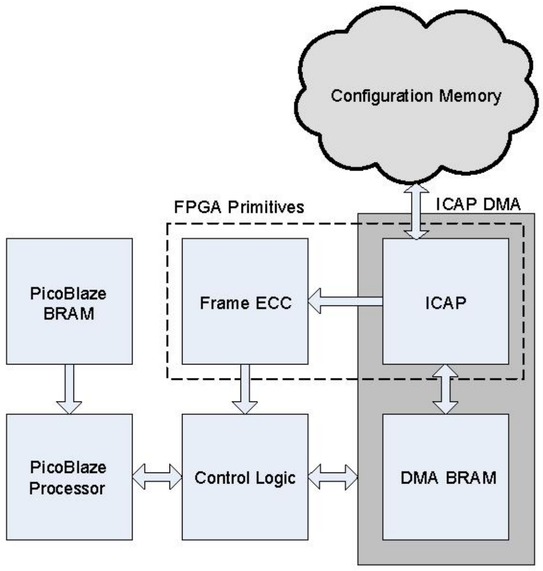

Figure 2. ICAP Based Internal Scrubber

Figure 2 shows the basic architecture of the ICAP based scrubber. Central to this scrubbing circuit are the ICAP and

Frame ECC primitives. These primitives cannot operate independently, additional logic is needed to manage the ICAP and scrubbing process. A programmable soft-core PicoBlaze processor is used to manage the ICAP because of its flexibility and ability to perform complex readback and scrubbing functions. A block ram (PicoBlaze BRAM) is required to store the PicoBlaze scrubbing program. A memory buffer (DMA BRAM) is also required to buffer the data between the ICAP and PicoBlaze processor. The ICAP DMA Engine provides the control logic for interface between the ICAP, DMA BRAM, and additional logic. Additional control circuitry is needed to synchronize the ICAP DMA Engine with the PicoBlaze processor. Each of these blocks will be described in more detail below.

ICAP

The Internal Configuration Access Port (ICAP), shown in Figure 3, is a primitive found within Virtex-4 FPGAs. This primitive is used to perform the readback and re-configuration operations required for scrubbing. The ICAP has direct access to configuration memory through the configuration registers. The interface of the ICAP resembles the interface used by the traditional SelectMap [9].

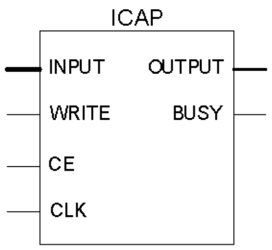

Figure 3. ICAP Block Diagram

The SelectMap interface is the reconfiguration external interface used by traditional scrubbers. The SelectMap interface dedicates user I/O pins for a bi-directional data bus and control signals, and this interface cannot be used with Bitstream encryption. The ICAP requires no dedicated input or output pins because it requires no external components. ICAP separates the data bus into an input bus, and an output bus [10]. The ICAP can perform readback and re-configuration even if the bitstream is encrypted [9]. The ICAP cannot perform initial full configuration, whereas the SelectMap can.

The ICAP interface can be used for different operation purposes. The main use of the ICAP in this work is to scrub the FPGA configuration memory. Research provided by Zeiniddini [11] and Galerin et al [12], use the ICAP interface for means of self-reconfiguration and partial configuration. Ruano [13] and Groza et al [14], indicate the possibilities to use the ICAP for fault tolerance and fault injection. The application of using the ICAP interface for reconfiguration of an encrypted bitstream was presented by Bossuet et al [15].

We utilized the scrubbing application of the ICAP interface in order to assist in detecting and correcting SEUs. This implementation requires that the configuration registers used by the ICAP are set for both readback and writeback operations. The scrubbing logic uses the ICAP to read the configuration logic. Scrubbing is performed on the data, and then the logic presents the scrubbed data to the ICAP. The ICAP will then write the data back to the configuration logic.

FrameECC

The Frame Error Correcting Code (ECC) interface is a primitive found within Virtex-4 FPGAs. The ECC utilizes a Hamming code algorithm to produce a single error correct, double error detection (SECDED) syndrome value. For any single error in a frame, this syndrome value will identify the bit within the frame that is in error. The syndrome value is represented in 11 bits. Table 1 provides a decomposition of the syndrome value and its corresponding error status. The Frame ECC interface works in collaboration with the ICAP interface in order to detect and correct errors. Using internal FPGA logic, the output of the ICAP interface connects directly to the input of the Frame ECC interface. As the ICAP reads a frame, the Frame ECC reads the data as well. The scrubbing logic uses the outputs of the Frame ECC interface to identify when an error occurs, and the location of the affected bit within the frame [9].

Because the Frame ECC is a “hard” primitive, there are certain limitations to its use. The Hamming Code algorithm can only detect when a multiple bit upset (MBU) in a single frame occurs. It cannot identify where those upsets are within the frame. As the configuration cross sections shrink with improvements in technology, the probability of a high-energy particle causing a MBU increases [16], [17]. Therefore, the Frame ECC logic will need to be replaced in the future with an ECC than can provide locations to the MBUs in the frame. Alternatively, application designers will need to design their own Frame ECC within the configuration memory.

ICAP DMA Engine

The ICAP DMA Engine is responsible for managing the flow of information between the ICAP interface and the PicoBlaze processor. The intermediate memory buffer, DMA BRAM, is necessary due to the differences in the I/O requirements of the ICAP interface and PicoBlaze processor. The PicoBlaze outputs an 8-bit word at a time, whereas the ICAP requires a burst of 32-bit data values.

Despite the need for the ICAP DMA, this primitive presents a couple of limitations. The major limitation is observed during a “fast” scan sequence. The “fast” scan routine is discussed in more detail in the PicoBlaze Processor portion of this section. This routine causes the ICAP interface to read all the configuration memory. The Frame ECC indicates that an error is detected; however, the ICAP DMA cannot stop the ICAP until all frames have been read. This causes the routine to miss the opportunity to fix the error at the time it was detected. This results in a fast detection but slower correction algorithm.

Table 1. Syndrome Value and Corresponding Error Status

Syndrome bit 11	Syndrome bit 10 to 0	Error Status
0	All 0s	No Error
1	Not Equal to 0	SEU, S[10:0] Identifies SEU Bit Location
1	All 0s	SEU, Overall Parity Bit p[11] is in Error
0	Not Equal to 0	MBU, Not Correctable

PicoBlaze Processor

The PicoBlaze Processor is a simple 8-bit programmable soft-core processor embedded into the configuration logic [18]. Figure 4 shows the basic architecture of the processor. After the initialization of the FPGA device, the processor starts to fetch and execute each instruction in the PicoBlaze BRAM.

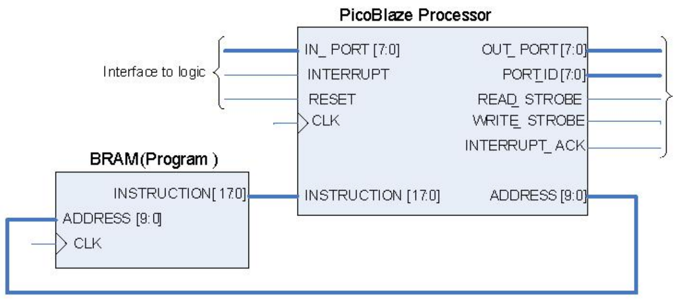

Figure 4. Basic PicoBlaze Processor Architecture

Additional control logic allows the PicoBlaze processor to communicate with all other components. The processor requires an associated BRAM to store the scrubbing program. The following paragraphs describe the functionality of the PicoBlaze code as seen in the flow chart of Figure 5. The flow chart emphasizes four basic blocks that make up the scrubbing logic: Initialization and Synchronization, "Walk" scan, "Run" scan, and correction logic. Each block will be discussed briefly in the following paragraphs.

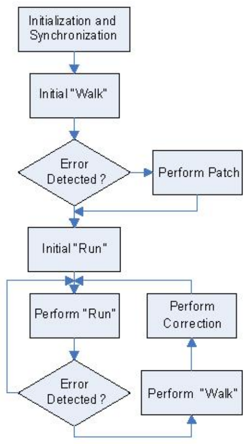

Figure 5. Basic PicoBlaze Program Flow Char

In order to use the ICAP interface, the Frame ECC interface, or even the PicoBlaze processor, each device must be initialized and synchronized to one another. To initialize the ICAP interface, the program sets vital configuration register values, and requests the ICAP DMA Engine to send a synchronization command sequence to the ICAP. Once the ICAP is synchronized with the PicoBlaze processor, the Frame ECC interface is also synchronized so that all three major components are working together to perform the scrubbing operations.

An initial "walk" of the FPGA configuration logic is required in order to fix any errors caused during configuration. A "walk" refers to a slow scan method that performs a scrub on one configuration frame at a time. This scanning method performs a complete scan of the entire device in 24ms to 278ms at a 100MHz clock, depending on the device [1]. If an error is detected during the initial "walk" through the device the patch function is called.

The patch function is performed by dedicated hardware and initiated by the PicoBlaze program. This function is designed to repair errors that occur during the configuration. The patch routine repairs the error by replacing the parity bits within the frame. As suggested by the name, this patch function does not correct the error but rather overlooks the error.

An initial "run" of the FPGA configuration logic is required to verify that all initial errors are corrected. This type of scanning method is labeled "run" because no scrubbing is performed. The program requests the ICAP interface, through the ICAP DMA, to read all configuration logic in the device. The Frame ECC interface is also checking the logic for errors. This scan can perform a complete scan of the device in 1.2ms to 14.6ms at a 100MHz clock, depending on the device [1]. It is important to note that this scan does not correct errors, but performs a fast detection of the errors.

After the initial "run" is completed, the main scrubbing loop starts. This loop starts by performing a fast scan of the device. If the Frame ECC indicates an error occurred during the scan, then the walk routine locates the erroneous frame, and the correction function repairs the erroneous frame. If no error is detected then the fast scan is repeated.

Additional functions not shown in Figure 5 are used for testing and debugging purposes. These include the capabilities to read and write configuration registers, the UART communication protocols between the circuit and a remote computer, transmition of BRAM data to the remote computer, and an error injection interrupt service routine. These functions and their basic design will be described briefly in the Test Structure section of this paper.

4. High Reliability Scrubber

Like all logic in the FPGA fabric, the internal ICAP scrubber circuit is susceptible to singel event upsets. A single upset within the scrubber circuitry could cause the circuit to output incorrect data or cause the circuit to fail. If the scrubber circuit fails, then the entire device configuration could be in jeopardy because it could rewrite the configuration bits to the device incorrectly. Because of this, it is essential that the scrubber circuit is reliable and immune to single event upsets.

The goal of this project was to modify the ICAP scrubber to make it more reliable. The ICAP scrubber was made more reliable by modifying the original design and applying mitigation techniques. Two modifications were made to the design to improve reliability. The first was triple modular redundancy (TMR) to mitigate against single-event upsets in the logic. The second involves BRAM scrubbing because the ICAP scrubber does not scan the BRAM section for errors. Each of these techniques will be described in more detail in the following sections.

TMR

TMR was applied to the ICAP scrubber to make it more reliable. The purpose of TMR is to reduce single points of failure [19]. It accomplishes this by triplicating each component in the design so that three identical circuits are running at the same time. TMR reduces single points of failure because it allows for multiple components within a design to fail and still allow the system to output correct data.

A special implementation of TMR that we used in our design is called Feedback TMR. Feedback TMR uses voters on the feedback loops within the circuit [20], [5]. Feedback TMR reduces the risk of a next state calculation error by reducing the number of single points of failure. Since the ICAP scrubber uses many state machines to control the data between key components, a feedback TMR implementation made it a more reliable circuit.

TMR was applied to this design using the BL-TMR tools. This project utilized these tools on every component of the ICAP scrubber except for the ICAP, Frame ECC, and PicoBlaze program BRAM. The ICAP and Frame ECC cannot be triplicated because there are not enough resources to apply triplication. A BRAM scrubber was used to protect the PicoBlaze program BRAM because BRAMs is not scrubbed by the ICAP scrubber.

Block Memory Scrubbing

Although TMR is used to mitigate any single failure within the BRAMs, there is no way to repair upsets within the memory since the internal scrubber does not scrub the BRAM memories. These memories are not scrubbed since the contents of the memories change with time and the scrubbing circuit does not have a “golden” copy of the contents. Since the data is dynamic, there is no way to differentiate between a user input and an SEU. The ICAP scrubber omits the BRAM section of the FPGA from its scanning routine and continues with the rest of the configuration frames.

The ICAP scrubber was designed with three BRAM memories. The PicoBlaze processor uses BRAMs for a stack, scratch pad, store, and register memory. This memory will change with time as the processor executes. The other BRAMs in the design are the DMA BRAM and the PicoBlaze program BRAM. The DMA BRAM was described previously. The PicoBlaze program BRAM stores the instructions used in the PicoBlaze processor. The instructions are never changed after initialization so this BRAM can be thought of as a ROM.

The BRAM scrubber uses TMR and is based on the design in [21]. The BRAM scrubber uses a dual port BRAM to make

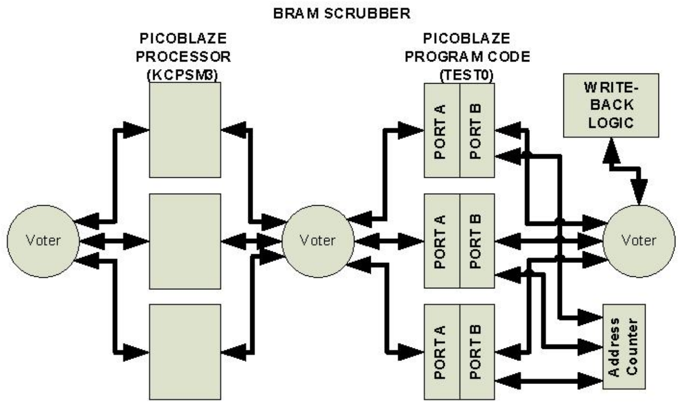

Figure 6. Architecture Block Diagram of BRAM Scrubber

a BRAM scrubber. Port A can be freely used for any data transfer while port B is used for the BRAM scrubbing. A counter connected to the B address port, counts sequentially through every address in the BRAM. The output data from port B is voted on and if an error was found the correct data is rewritten to the BRAM. Figure 6 shows the architecture of the BRAM scrubber.

5. Radiation Test Organization

The high reliability scrubbing circuit was tested in a radiation environment to determine its effectiveness in repairing configuration upsets as well as operating in the presence of upsets within the scrubber itself. The goals of this test were to (1) demonstrate a working scrubber in the presence of upsets, (2) determine the improvements in reliability provided by TMR and memory scrubbing, and (3) identify ways of improving the reliability and on-line functionality of the circuit. This section will summarize how this test was organized.

Test Fixture and Design

The test was constructed to irradiate a single FPGA that contains an operational internal scrubbing circuit. No other circuits were configured within the FPGA. The test was performed on the Avnet Virtex-4 LX-25 evaluation board and the device under test was a Virtex-4 LX-25 FPGA. A custom aluminum shield was created for this board to shield all other components from the high-energy protons (see Figure 7).

Two communication interfaces are provided between this text fixture and a host computer (see Figure 7). A JTAG programming cable is used to configure the device under test. A UART cable links the host computer to the PicoBlaze processor. The PicoBlaze provides status information during the scrubbing process to the host computer. The UART operated at a relatively low baud rate that limited the amount of information that could be transmitted during the test (38400 baud).

Two designs were used for testing. The designs operating on the FPGA both implemented the internal scrubber circuit described earlier in Section 3. However, the first design did not implement TMR described in section 4, while the second design did implement TMR. Table 2 provides the size of each design. No other application-specific or non-scrubbing circuitry was added to the design[1]. The design simply performed internal scrubbing continuously to detect and repair errors.

Table 2. ICAP-Based Scrubber Design Utilizations

Resource	non-TMR	TMR
Flip Flops	680 (3%)	1082(5%)
Slices	736 (6%)	1308(12%)
BRAM	2	6

There were several limitations of the TMR implemented design that reduced its reliability. First, the design used a single clock for all TMR circuit domains. This introduced a single point failure on the clock line (i.e. a single failure on the clock may break all three circuit domains). Second, the I/O pins for the UART were not triplicated due to board constraints (i.e. there was only a single pin for the UART RX/TX signals). This introduced a single point failure on the communication channel between the scrubber and the host computer.

As mentioned, the main goal is to identify the advantages of utilizing TMR with the ICAP over traditional methods or using the ICAP alone. A third design implementing the traditional SelectMap method was not implemented because of the lack of resources needed to create the design.

Host Computer

A host computer is connected to the device via a serial connection. The host computer runs an application to control input and output data to and from the device. This application governs the serial data streams, decodes signals transmitted from the device, and monitors the flow of the scrubbing logic. All data received from the UART is stored by the host program into text-based files for further analysis. The particulars on what type of data is transmitted is discussed in the next section.

Test Program

This section will briefly discuss the portions of the program that are specific to the testing and data collection required for the radiation test. A description of all other sections of the program is referenced in the PicoBlaze Processor section of the Internal Scrubbing section. The portions of the program referenced here include the capabilities to read and write configuration registers, the UART communication protocols between the test design and a host computer, transmition of BRAM data to the host computer. These functions can be accessed by the user from the host computer, after the scrubbing function is disabled.

In order to achieve realtime collection of data from the test, a UART module was implemented within the configuration logic. The UART module is designed to relay back to a host computer important information pertaining to the test such as, watch-dog timer signals, SEU or MBU detection signals with corresponding frame address register (FAR) and syndrome values, scan completion signals, and, if requested, BRAM content. For many testing and debugging purposes, it is often advantageous to look inside and see the different values of the configuration registers, or the current BRAM content. Using the UART the program is able to transmit all this data to a remote host computer for data analysis. The program also allows the user to transmit data back through the UART in order to change the values of the configuration registers or BRAM content.

As mentioned previously, the baud rate of the UART limits the amount of data that can be passed from the device to the computer. In order to meet this limit the program transmits coded ASCII signals that are interpreted by the higher-level program on the host computer. The only real data transmitted are the FAR and syndrome values. All other data represent either watchdog timer signals, or test procedure signals. During the testing of the reliability of the ICAP-based scrubber, the circuit was set to constantly perform the scrubbing program. This requirement closed off all debugging and testing options previously mentioned. Therefore, in order to assist in determining when something fails, and what that thing is, we implemented watchdog timers throughout the different layers of the program. These timers indicate when a scan fails to complete, and if specific signals failed to be asserted or deasserted. The test procedure signals indicate when a scan has completed, if scrubbing is disabled, if an error occurred, and the type of error that occurred.

The FAR and syndrome values are transmitted to the host

[1]There were plenty of free resources available on the FPGA but additional circuitry was avoided to reduce the complexity of the test.

computer after the transmition of the error signal. These values and their associated error type are recorded for further data analysis. For further analysis, it is important to record the number of errors that occur within a scan, the high-level program on the host computer uses the signal indicating a completion of a scan in order to know how many errors occurred between the previous scan and the latest one. When the program on the host computer receives the ASCII value that represents a disabling of the scrubbing program, it then reports the total number of errors that were recorded. It is important to note that since the scrubbing mechanism cannot fix MBUs, the logging program should not record repeat MBUs when they are reported again. All signals transmitted from the UART to the host computer are recorded for further analysis.

Test Organization

In order to test the reliability of the ICAP circuitry, we provided some measure of protection from radiation for the other devices on the development board. A 1"-thick aluminum shield was designed to shield the other components and for mounting of the board. A hole was drilled in the shielding to expose the FPGA to the high energy particles. Pictures of the board and shield are shown in Figure 7.

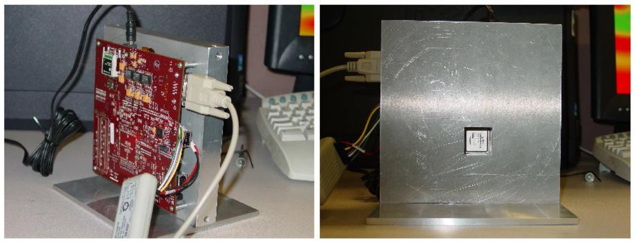

Figure 7. Picture on the left shows the back and side view of the development board used in the test with shield, the picture on the right shows the front view.

Because of budget constraints, this test was performed simultaneously behind another experiment. The shielded board was placed behind another board and we did not have control over the beam flux. The proton beam was set to 63 MeV but the energy reaching this test was reduced due to presence of another board. We were unable to measure the beam energy but estimate the energy to be roughly 10% lower than the beam setting. The beam flux was set by another experiment and varied from $1.79e7$ $protons/cm^2/s$ to $8.89e8$ $protons/cm^2/s$ with and an average flux of $2.47e8$ $protons/cm^2/s$ for all tests.

6. Test Results

The design was tested at the Crocker Cyclotron at the Davis Campus of the University of California on the dates of August 28th and 29th 2007. The first day's tests lasted for 5 hours, and the second days tests ran for 7.5 hours. As mentioned previously, two different designs were tested during that time. The following sections are results, and main issues associated with the test.

MBUs

Multiple bit upsets (MBU) within a single frame is the biggest limitation of this scrubbing circuit. A multiple bit upset in adjacent cells can be caused by a single charged particle. Multiple bit upsets can also occur when two independent particles upset two cells within the same frame. Due to test setup limitations, we are unable to determine whether MBUs were caused by a single particle or multiple particles.

As mentioned previously, the scrubbing logic performs two types of scans, a "run" scan, and a "walk" scan. The optimal solution is to perform many fast scans and very few slow scans. The "walk" scan will only occur when an error is detected during the "run" scan. However, with MBUs the fast scan will detect an error every scan cycle, which slows down the speed of the scrubbing program. This is because the SECDED code of the FRAME ECC interface cannot provide locations of the multiple erroneous bits within the frame for repair.

Our test results showed that an average of 24.75 MBUs per failure occurred for the mitigated TRM design. The design without TMR tolerated only 10.68 MBUs per failure. Data from the test indicates that 1.7% of all upsets were MBUs. Other data retrieved from the test shows that the average number of SEUs detected and corrected per scan increase as the number of MBUs detected increase.

Failure Mechanisms

During the testing of the two designs at times the system would fail. These faults manifested themselves in different manners. The following list mentions the types of failures noticed. Following the list is a hypothesis of what may have caused the failures. The conclusion section discusses some of the future fixes that may help in avoiding these faults.

1. Program crash
2. Invalid response from UART
3. Repeat FAR & syndrome values
4. Repeat FAR but different syndrome values
5. Repeat sets of FAR & syndrome values
6. FAR increments till end of FPGA row
7. Errors detected after test finished
8. Failed during reconfiguration

Many of the errors above are difficult to identify their cause due to the limitations in our testing methodology. Our hypothesis is that the errors that affect the FAR values are single event fault interrupt (SEFI) failures in the configuration logic. The rest of the items on the list are caused by the accumulation of uncorrectable errors or single point failures. All faults can be fixed by a reconfiguration of the device. Due to limitations on test variables, we are unable to prove this property at the test.

A program crash indicates that the UART has stopped transmitting data; this can be caused by no more data is presented

Table 3. Fault types and quantities detected per design. Fault numbers correspond to itemized list.

Design Type	Failure Types 1	2	3	4	5	6	7	8	Failure Total	Total # Tests	% Failed
w/ TMR	7	3	11	2	4	1	2	5	35	77	45.45%
w/out TMR	10	4	3	0	1	2	0	3	23	31	74.19%

to it or that it has failed. The invalid response from the UART indicates that the UART is transmitting either invalid data, or data that was not expected. There are no interrupts implemented by the design and the host computers program is designed to know the sequence of events provided by the scrubbing logic.

All the FAR related failures cause a repetition to show up. The first two prevent a scan from completing, the other two do not necessarily prevent a scan from completing, however once the repetition occurs the subsequent scans are irrelevant. Table 3 shows the dispersal of the types of faults per design type. As was discussed, due to limitations on control over the test variables these values do not show an accurate representation of the level of improvement between designs.

Reliability Improvement

The host computer retrieved data from both test designs for all tests performed on each design. Due to the limitations of test parameters, the designs were not tested with the same number of tests and the same amount of time. The mitigated design was tested 3.85 times longer than the design that did not implement the TMR.

The use of TMR and scrubbing demonstrated significant improvements in reliability. The mitigated scrubber showed a fluence to failure that was 3.6 times greater than the unmitigated design. Further, the mitigated design was able to tolerate far more MBUs than the unmitigated design. Figure 8 and Figure 9 show the number of MBUs and SEUs that occurred before a failure for each type of design. These figures clearly indicate that the mitigated ICAP-based scrubber could detect and correct a far greater number of SEUs.

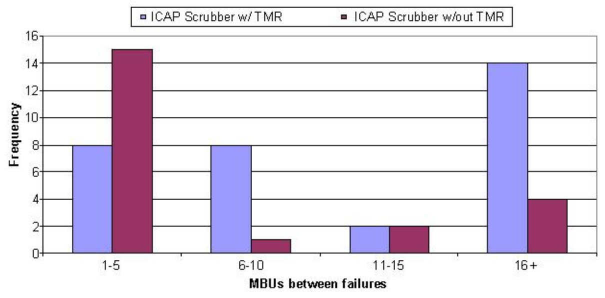

Figure 8. MBU to Failure Histogram

7. Conclusions

This paper describes a fault tolerant internal scrubber circuit using the ICAP configuration controller. The internal scrubber was successfully demonstrated in a high-energy proton test. The scrubber circuit was able to detect real-time failures in the bitstream and repair the bitstream when a single upset occurred within a frame. The scrubber operated continuously repairing upsets until a SEFI occurred or a functional failure in the design.

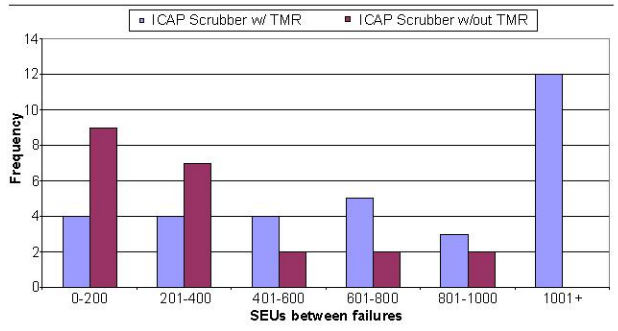

Figure 9. SEU to Failure Histogram

Both a standard non-triplicated scrubber and high reliable triplicated version of the scrubber were tested. Both scrubbers operated correctly and, as anticipated, the high reliability scrubber operated demonstrated significant improvements in fluence to failure. However, the triplicated circuit did not improve the reliability as much as anticipated because of limitations of the test fixture and limited visibility into the design. Future work will improve the visibility into the design and remove several single-point failures within the triplicated design.

The major limitation of this work was the inability of the scrubber to repair multiple bit upsets (MBU). While the SECDEC code used by the FrameECC was able to detect single and multiple bit upsets, it was unable to correct more than one upset within any given frame. Because of this, the scrubber would skip over these MBUs and allow them to accumulate. While the design was able to operate correctly with a surprisingly large number of MBUs, the system would eventually fail and required full reconfiguration. Future work will investigate the possibility of correcting multiple upsets within a frame.

The ICAP scrubber offers an alternative low-cost method of identifying and repairing upsets within an FPGA. Used in conjunction with other techniques, the ICAP scrubber can provide additional reliability for FPGA circuits at lower cost. The ICAP scrubber will be used in a variety of other future projects including internal fault injection and fault recovery.

Acknowledgments

This work was supported by Lockheed Martin under a grant from the University Projects program with collaboration by Tim Gallagher and Jon Wilson.

References

[1] L. Jones, "Single event upset (seu) detection and correction using virtex-4 devices," Xilinx Ltd., San Jose, CA, Application Note XAPP714, January 2007.

[2] D. Ratter, "FPGAs on Mars," Xilinx, Tech. Rep., August 2004, xCell Journal #50.

[3] M. Caffrey, "A space-based reconfigurable radio," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, T. P. Plaks and P. M. Athanas, Eds. CSREA Press, June 2002, pp. 49–53.

[4] A. S. Dawood, S. J. Visser, and J. A. Williams, "Reconfigurable FPGAs for Real Time Image Processing in Space," in *14th International Conference on Digital Signal Processing (DSP 2002)*, vol. 2, 2002, pp. 711–717.

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

[6] F. Lima, C. Carmichael, J. Fabula, R. Padovani, and R. Reis, "A fault injection analysis of Virtex FPGA TMR design methodology," in *Proceedings of the 6th European Conference on Radiation and its Effects on Components and Systems (RADECS 2001)*, 2001.

[7] C. Carmichael, M. Caffrey, and A. Salazar, "Correcting single-event upsets through virtex partial configuration," Xilinx Corporation, Tech. Rep. XAPP216 v1.0, June 2000.

[8] M. Berg, "The NASA Goddard space flight center radiation effects and analysis group virtex 4 scrubber," in *Xilinx meeting*, 2007.

[9] *Virtex-4 Configuration Guide*, 1st ed., Xilinx Ltd., San Jose, CA, January 2007.

[10] *Virtex-4 Libraries Guide for HDL Design*, 9th ed., Xilinx Ltd., San Jose, CA, 2007.

[11] A. H. S. Zeineddini, "Secure partial reconfiguration of fpgas," Master's thesis, George Mason University, Fairfax, VA, 2005.

[12] B. Rousseau, P. Manet, D. Galerin, F. Dedeken, Y. Gabriel, D. Merkenbreack, and J.-D. Legat, "Enabling certification for dynamic partial reconfiguration using a minimal flow," in *Proceedings of the Conference on Design, Automation and Test in Europe*, July 2007, pp. 983–988.

[13] O. Ruano, P. Reyes, J. Maestro, L. Sterpone, and P. Reviriego, "An experimental analysis of seu sensitiveness on system knowledge-based hardening techniques," in *Design and Diagnostics of Electronic Circuits and Systems*, April 2007, pp. 261–s66.

[14] V. Groza, R. Abielmona, M. H. Assaf, M. Elbadri, M. El-Kadri, and A. Khalaf, "A self-reconfigurable platform for built-in self-test applications," *IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT*, vol. 56, no. 4, pp. 1307–1315, 2007.

[15] L. Bossuet, G. Gogniat, and W. Burleson, "Dynamically configurable security for sram fpga bitstreams," *Int. J. Embedded Systems*, vol. 2, no. 1/2, pp. 73–85, 2006.

[16] K. Johansson, M. Ohlsson, N. Olsson, J. Blomgren, and P.-U. Renberg, "Neutron induced single-word multiple-bit upset in sram," *IEEE Transactions on Nuclear Science*, vol. 46, no. 6, pp. 1427–1433, December 1999.

[17] J. Maiz, S. Hareland, K. Zhang, and P. Armstrong, "Characterization of multi-bit soft error events in advanced srams," *Proc. IEEE International Electron Devices Meeting*, pp. 519–522, May 2003.

[18] K. Chapman, *KCPSM3 8-bit Micro Controller for Spartan-3, Virtex-II and Virtex-IIPRO*, 7th ed., Xilinx Ltd., San Jose, CA, October 2003.

[19] N. Rollins, M. Wirthlin, M. Caffrey, and P. Graham, "Evaluating TMR techniques in the presence of single event upsets," in *Proceedings fo the 6th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*. Washington, D.C.: NASA Office of Logic Design, AIAA, September 2003, p. P63.

[20] R. Katz, R. Barto, P. McKerracher, B. Carkhuff, and B. Koga, "Seu hardening of field programmable gate arrays (fpgas) for space applications and device characterization," *IEEE Transactions on Nuclear Science*, vol. 41, no. 6, pp. 2179–2186, 1994.

[21] G. Miller and C. Carmichael, "Single-event upset mitigation for xilinx FPGA block memories," Xilinx Corporation, Tech. Rep., February 1, 2007, xAPP962 (v1.0).

Biography

Jonathan Heiner received the B.S. degrees in Computer Engineering Technology and Software Engineering Technology from the Oregon Institute of Technology, Klamath Falls, OR, in 1996. He is currently pursuing the M.S. degree in Electrical and Computer Engineering at Brigham Young University, Provo, UT. His research interests include reconfigurable computing, and hardware/software codesign.

Nathan Collins is pursuing the B.S. degree in Computer Engineering from Brigham Young University, Provo, UT. His research interests include reconfigurable computing and business management.

Michael J. Wirthlin received his B.S. and Ph.D. degrees from Brigham Young University (BYU) in 1992 and 1997 respectively. After completing his Ph.D., he worked as a Staff Research Engineer in the Systems Architecture Laboratory at National Semiconductor Corporation in Santa Clara, CA. He is currently an Associate Professor in the Department of Electrical and Computer Engineering at BYU. His research interests include Configurable Computing Systems, FPGA reliability, fault tolerance computing, high-level synthesis, and computer-aided design for application-specific computing.