Scrubbing Optimization via Availability Prediction (SOAP) for Reconfigurable Space Computing

Quinn Martin, Alan D. George
NSF Center for High-Performance Reconfigurable Computing (CHREC)
ECE Department, University of Florida
Gainesville, Florida
{martin, george}@chrec.org

***Abstract*—Reconfigurable computing with FPGAs can be highly effective in terms of performance, adaptability, and power for accelerating space applications, but their configuration memory must be scrubbed to prevent the accumulation of single-event upsets. Many scrubbing techniques currently exist, each with different advantages, making it difficult for the system designer to choose the optimal scrubbing strategy for a given mission. This paper surveys the currently available scrubbing techniques and introduces the SOAP method for predicting system availability for various scrubbing strategies using Markov models. We then apply the method to compare hypothetical Virtex-5 and Virtex-6 systems for blind, CRC-32, and Frame ECC scrubbing strategies in LEO and HEO. We show that availability in excess of 5 nines can be obtained with modern, FPGA-based systems using scrubbing. Furthermore, we show the value of the SOAP method by observing that different scrubbing strategies are optimal for different types of missions.**

Keywords: scrubbing; FPGA; reliability; aerospace

I. Introduction

Reconfigurable computing with field-programmable gate arrays (FPGAs) is known to perform well in space-based processing applications such as hyperspectral imaging, synthetic aperture RADAR, and software-defined radio. FPGAs often outperform general-purpose CPUs with respect to operations per second and power consumption, while offering advantages over ASICs in non-recurring engineering cost and adaptability. Unfortunately, these devices are highly susceptible to radiation effects [1]. Because FPGA hardware in a reconfigurable system is configured by on-chip SRAM, single-event upsets (SEUs) in this memory can corrupt the operation of the device and lead to errors or failure if unmitigated.

Configuration scrubbing is the process of quickly repairing these upset configuration bits before they can accumulate in the configuration memory. Several techniques for scrubbing have been deployed in aerospace missions, but each performs differently with respect to detection and correction time, coverage of different upset patterns, fault localization, and other factors. Further complicating the problem of choosing an optimal scrubbing system, the system designer may need to combine several techniques to achieve the target performance in terms of system availability and other mission goals. This scenario presents a large design space that is difficult to explore without implementation and testing on a specific target platform.

In this work, we develop and present the method of scrubbing optimization via availability prediction (SOAP), which allows the system designer to predict the performance of a set of scrubbing strategies for a given system and mission using only analytical data. Traditionally, scrubbing strategies have been designed around system requirements and tested in the implementation phase. With SOAP, the optimal scrubbing strategy for a given mission can be selected during the design phase and the system designed around its requirements, maximizing the potential system availability.

II. Background

FPGAs are configurable logic devices that implement logic circuits with a fabric that includes lookup tables (LUTs) for implementing logic equations, memories for sequential logic and storage, and routing resources that connect the LUTs and memories [2]. Although one-time programmable FPGAs exist and have been widely deployed in space, reconfigurable FPGAs offer several advantages with their lower cost, greater ease of prototyping, larger and faster available devices, and ability to be reconfigured after deployment. In a reconfigurable FPGA, the configuration memory is a collection of bits called a bitstream. These bits set LUT values, flip-flop and memory initialization values, and states of switch and connection boxes that route signals through the FPGA.

For the Virtex family of devices from Xilinx, the configuration memory is composed of SRAM cells that are arranged in frames of 32-bit configuration words. There are 41 words per frame for Virtex-5 (V-5) [3] and 81 words per frame for Virtex-6 (V-6) [4]. Since the frame is the smallest addressable unit of configuration memory, any reads or writes on the configuration must be performed frame-wise. Several interfaces are provided for accessing configuration memory. The JTAG interface is typically used for initial configuration, and the Xilinx-specific SelectMAP interface is used for runtime readback and reconfiguration. The SelectMAP interface can be configured for a bus width of 8, 16, or 32 bits. In the Virtex-II and newer devices, Xilinx provides the internal configuration access port (ICAP), which exposes the SelectMAP interface to user logic. The ICAP allows the

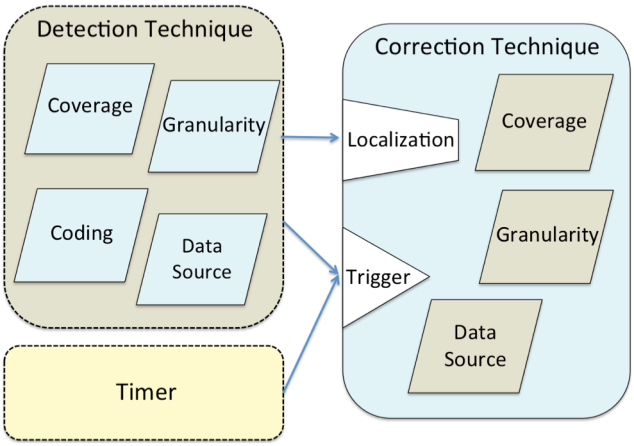

Figure 1. Detection and correction scrubbing techniques

FPGA to read back and reconfigure itself, eliminating the need for an external runtime-configuration manager.

Since the FPGA configuration is stored in volatile SRAM, it can be corrupted by interaction with high-energy radiated particles such as protons, neutrons, and heavy ions that are abundant in aerospace environments. The effects of these particles on electronics are well understood and are collectively known as single-event effects (SEE). Several types of SEE are relevant to FPGAs. Single-event upsets (SEUs) occur when one or more bits in memory changes state due to a radiation event. Since the state of the FPGA configuration memory specifies the application architecture, SEUs in the configuration memory are particularly harmful to system operation. If exactly one bit is affected by the event, it is called a single-bit upset (SBU). Otherwise, it is a multi-bit upset (MBU). As silicon feature size has decreased, MBUs have become more common. Double-bit and triple-bit upsets (DBUs and TBUs) now account for nearly 10% of all upsets in V-5 [5]. Similar to SEU, a single-event functional interrupt (SEFI) occurs when a critical device resource is upset such that device operation is seriously impaired, usually requiring system reset. Although reliable space systems must be able to recover from SEFIs, these events are several orders of magnitude less frequent, so the focus on improving FPGA reliability in space is SEU mitigation.

Many bits in the bitstream are used for routing or device resources that are not employed in a given design. Only a small portion, *critical bits*, directly affect operation of the design if upset. According to Xilinx, these make up about 10% of the configuration memory for an average design [6]. These critical bits can only be identified by a thorough fault-injection campaign. However, Xilinx allows generation of a mask file that identifies *essential bits* to a design, of which the critical bits are a subset [7].

A. Scrubbing

Since for most applications the FPGA is configured upon power-up, the desired state of the configuration memory will be known for such applications. This setup makes it possible to repair upset bits through scrubbing. The scrubber can be implemented as an external device, such as a radiation-hardened microprocessor, or internal to the FPGA using the fabric and ICAP [8]. *External scrubbing*, especially with radiation-hardened parts, is reliable but can be expensive in terms of power, size, and cost because it requires at least one additional processor. *Internal scrubbing* is superior with respect to these constraints, but requires additional care in implementation because the scrubber itself is vulnerable to SEUs and SEFIs.

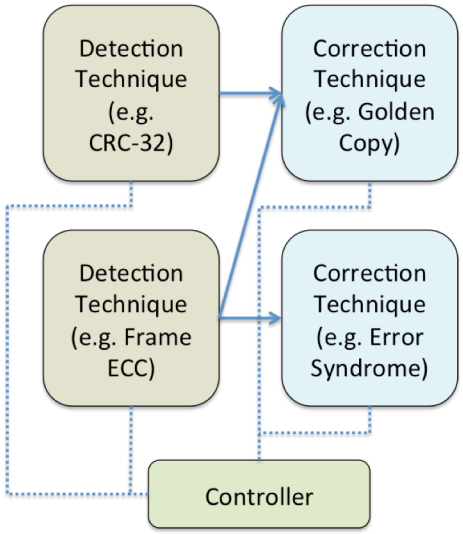

Figure 2. Complex scrubbing strategy with controller

A *scrubbing technique* is a single algorithm that can be used in a system to mitigate configuration-memory upsets. There are two types of techniques, *detection techniques* and *correction techniques*, each having properties such as error coding, coverage, granularity, and redundant data source that influence performance of the scrubber. A *scrubbing strategy* is composed of one or more correction techniques and, optionally, a number of detection techniques. *Blind scrubbing* is a scrubbing strategy with no detection technique, whereas a scrubbing strategy with at least one detection technique is called *readback scrubbing*, because the current state of configuration memory is read back from the device to detect an upset. The scrubbing strategy may also include a controller to process the information from various detection mechanisms and arbitrate access to the configuration interface among detection techniques, correction techniques, and application reconfiguration needs [9]. Fig. 1 illustrates a simple scrubbing strategy where a correction technique is triggered by a timer (blind) or a detection technique (readback). If the detection technique triggers the correction, it also passes localization information to the correction technique. Fig. 2 shows a more complex readback strategy with multiple detection and correction techniques and a controller.

B. Correction Techniques

All scrubbing strategies employ at least one technique for correcting upset configuration bits. Correction techniques use data redundancy to recall or calculate the original configuration and then write this configuration to the device. In general, these techniques differ in their coverage of various upset types (e.g., SBU vs. MBU), granularity of correction (e.g., frame vs. device) and correction data source (e.g., off-

chip vs. on-chip memory). Depending on the chosen scrubbing strategy, the correction technique may be triggered continuously, by a simple timer delay, or by a detection technique.

The two correction techniques widely used for scrubbing are *golden copy correction* and *error syndrome correction*. In golden copy correction, a trusted "golden" copy of the original configuration is maintained off-chip in non-volatile storage, such as a radiation-hardened PROM, and used to reconfigure the FPGA as needed. Blind scrubbing strategies that employ only golden copy correction are widely used in FPGA-based space platforms because of their effectiveness and simplicity. These strategies continuously or periodically reconfigure the FPGA with the golden copy to repair errors quickly after they occur. This method can correct any configuration upset, but such radiation-hardened memories may have limited bandwidth, such that the configuration clock often can not be run at its maximum frequency.

To avoid the memory-access penalty of golden copy correction, as well as the expense of additional off-chip memory, the error syndrome correction technique can be used. In this case, the correction technique decodes an error syndrome to determine the original configuration. In practice, Hamming error-correcting codes (ECC) are often used to correct upsets. In Virtex-II devices and above, Xilinx embeds a single-error correcting, double-error detecting (SECDED) Hamming code in each configuration frame at the time of configuration bitstream generation [3]. Xilinx also provides a Frame ECC primitive that reports the error syndrome as each frame is read back over the SelectMAP interface. This error syndrome can then be used to correct a single error in the damaged frame and reconfigure the device.

C. Detection Techniques

For Xilinx FPGAs, readback scrubbing strategies use the SelectMAP interface to read back the current configuration and detect errors. This readback process combined with an error-detection algorithm forms a detection technique. Although there are many reasons to want upset detection, certain types of missions require detection in order to trigger a system reset, shutdown, or restoration of state to prevent catastrophic failure. We refer to these missions as upset-critical (UC) and distinguish between these and non-UC missions in our treatment of scrubbing strategies. The reset times for UC missions are highly system dependent and can vary from milliseconds for a purely hardware-based system to tens of seconds for a system with a master controller running the Linux operating system, for example.

Error detection using test vectors, such as CRC-32 and Hamming codes, has gained popularity because the redundant data accessed for detection is greatly compressed compared to the original bitstream. This approach allows for much quicker detection of certain upset types (e.g., burst errors and SBUs). However, MBU rates are increasing with smaller feature size, to the point that 3-bit upsets are no longer uncommon. If the upsets occur in the same frame and Frame ECC alone is used the MBU can go undetected (if the number of upsets is a multiple of 4) or even be corrected erroneously (for an odd number of bits upset greater than 1). The fact that Frame ECC alone allows errors to accumulate has caused recent interest in more robust codes.

The first test vectors proposed for error detection on FPGAs were cyclic redundancy check (CRC) codes [10]. A 32-bit CRC code can detect up to a 32-bit burst error, making it much more robust against MBUs than Frame ECC. Increasing MBU susceptibility of newer devices has prompted Xilinx to include built-in, 32-bit readback CRC hardware on Virtex devices since the V-5. This fixed-logic hardware constantly reads back frame data from the configuration interface whenever it is not in use. It calculates a CRC over the entire device configuration and compares this value with a golden value calculated upon issuing a special "Reset CRC" SelectMAP command. When a CRC mismatch is detected, it is reported on a pin of the Frame ECC primitive. This method of error detection is robust against the accumulation of errors because even if there is a detection code collision with the current upset pattern it is improbable that the next upset will be missed.

D. Markov Availability Modeling

Markov models can be used to model systems with several states, each of which have constant transition rates to one or more other states [11]. By solving these models, we can find the time spent in each state over a simulated run of the system. Discrete solvers such as SHARPE can solve complex Markov models in a small amount of real time [12]. They are useful in modeling reliability for systems with constant failure and repair rates, because we can find the amount of time spent in functional or "up" states for the system. Using this information, we can find the availability, A, of the system by

$$A = \frac{Uptime}{Total\ runtime}. \quad (1)$$

Since the availability for reliable systems is very close to 1, it is common to express it as the number of nines in the decimal representation of the availability, e.g., .99990 would be exactly 4 nines of availability.

III. Approach

The broad array of scrubbing options available to system designers can be daunting, and this problem is worsened by conflicting opinions on the efficacy of various scrubbing techniques. We propose the method of scrubbing optimization via availability prediction (SOAP) by which to compare scrubbing strategies and choose the best strategy for a given mission (Fig. 3). Using availability as a metric, we develop Markov models to describe scrubbing strategies of interest. We then populate the models using three types of parameters: *environmental parameters*, *system parameters*, and *scrubbing parameters*. Some parameters are fixed for a given system choice, while others can be varied to achieve optimal performance. The parameters are developed analytically in this paper, but could also be derived experimentally to obtain greater prediction accuracy.

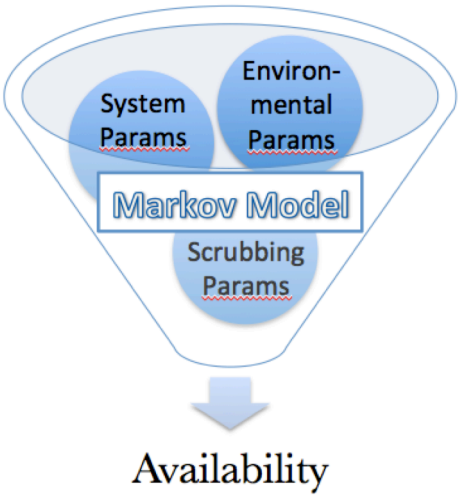

Figure 3. Concept diagram for the SOAP method

TABLE I. ENVIRONMENTAL PARAMETERS

Environmental Parameters	Virtex-5		Virtex-6	
	LEO	HEO	LEO	HEO
λ_{bit} (SEUs/bit/s)	2.63E-12	7.31E-12	1.82E-12	4.623E-11
α_{SBU}	0.9		N/A	N/A
α_{DBU}	0.09		N/A	N/A
α_{TBU}	0.01		N/A	N/A

A. Environmental Parameters

The environmental parameters are the upset rates (λ) in various orbits of interest, and the MBU coefficients (α). These parameters are different for each device family, because upset rates are highly dependent on device process technology and architecture. We used CREME96 with radiation cross-sections from [5] to find the per-bit upset rates for the V-5 in the ISS LEO orbit and the Molniya HEO orbit (Table 1). We used the linear regression method proposed in [13] to estimate cross-sections for the V-6 and found the upset rates for the same LEO and HEO orbits. From [5], we estimated the MBU coefficients for the V-5 device. MBU data was not available for the V-6 device, but general trends from the Virtex to V-5 indicate an increase in MBU rates with smaller process technology [14]. The equations for calculating the device upset rate, λ_{device}, and adjusted MBU rates, λ_{MBU}, from these parameters are given in (2) and (3). Using these equations, we derive the rates λ_{SBU}, λ_{DBU}, and λ_{TBU}.

$$\lambda_{device} = \lambda_{bit} \times (\#\ configuration\ bits)(\%\ critical) \quad (2)$$

$$\lambda_{MBU} = \lambda_{device} \times \alpha_{MBU} \quad (3)$$

B. System Parameters

The system parameters are the SelectMAP bus width, B, and the configuration clock frequency, f_{CCLK}. These parameters are set by the system designer and are generally limited by the configuration memory bandwidth and the available routable pins on the device. Increasing either of these parameters has a direct effect on availability, since they are further used to calculate the detection and correction rates. Here, we assume a conservative hypothetical system using a radiation-hardened memory with an 8-bit bus at 33 MHz.

C. Scrubbing Parameters

The scrubbing parameters, μ and γ, describe the correction and detection rates of the system respectively. We derive these parameters from the particular scrubbing technique characteristics combined with the system parameters. These rates can be found experimentally be injecting faults into the system and measuring the time delay to detection and correction of the fault. Fortunately, we can usually estimate these rates analytically for a given technique, assuming an efficient hardware implementation.

$$\mu_{device} = \frac{B \times f_{CCLK}}{Total\ configuration\ bits} \quad (4)$$

$$\mu_{frame} = \frac{B \times f_{CCLK}}{Total\ bits\ per\ frame} \quad (5)$$

$$\gamma_{CRC} = \frac{32 \times f_{CCLK}}{Total\ configuration\ bits} \quad (6)$$

Each correction technique available for use in a scrubbing system has a corresponding correction rate, μ, which is the reciprocal of the time taken to repair an upset once the technique is triggered. The correction rate equations for device and frame golden copy correction are shown in (4) and (5), respectively. In this case, $\mu_{ECC} = \mu_{frame}$ because the reconfiguration time is the same, assuming reconfiguration is performed in parallel with accessing the golden copy memory. Detection techniques have a corresponding detection rate, γ, which is the reciprocal of the time taken to detect an upset once it has occurred. The detection rate for CRC-32 detection is shown in (6), assuming the readback bus width is set at 32 because the fixed-logic CRC circuit is used. For these devices, $\gamma_{ECC} = \mu_{device}$, because the time to read back the entire configuration memory is approximately the same as the reconfiguration time. In all cases, we neglect the overhead contributed by configuration command sequences. Correction and detection rates for several devices are shown in Table 3.

TABLE II. SYSTEM PARAMETERS

System Parameters	Value
B	8 bits
f_{CCLK}	33 MHz

TABLE III. SCRUBBING PARAMETERS

Scrubbing Parameters	Rate (s^{-1})					
	Virtex-5			Virtex-6		
	LX20T	LX110T	LX330T	LX75T	LX240T	LX760T
μ_{device}	42.24	8.48	3.19	10.08	3.57	1.43
μ_{frame}	2.01E+05			1.35E+05		
μ_{ECC}	2.01E+05			1.35E+05		
γ_{CRC}	168.96	33.93	12.77	40.31	14.29	5.71
γ_{ECC}	42.24	8.48	3.19	10.08	3.57	1.43

D. Markov Availability Modeling

We use Markov models to describe the scrubbing strategy algorithm using one or more correction techniques and zero or more detection techniques. In this work, we explore three different scrubbing strategies: blind; readback CRC-32; and Frame ECC. The strategy of blind scrubbing with simple upset correction is shown in Fig. 4. In this model, we have an "Up" state denoting a functional system and an "Upset" state to indicate failure. The failure rate of transitioning from "Up" to "Upset" is described by the λ_{device} factor, while the repair rate of moving from "Upset" to "Up" is μ_{device}. For the other two strategies, we build on the blind scrubbing model. In the CRC-32 scrubbing strategy (Fig. 5) we add a state to represent upsets detected by the built-in readback CRC-32, with a transition rate defined by γ_{CRC}. Finally, we propose a Frame ECC-based scrubbing strategy (Fig. 6) that uses Frame ECC detection and correction with a CRC-32 "safety net" for catching falsely corrected three-bit upsets. In this strategy, the CRC-32 is run each time after using the error syndrome correction method, and if it still detects an upset then a TBU is assumed and the entire device is reconfigured.

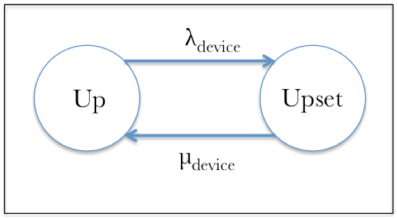

Figure 4. Markov model for blind scrubbing

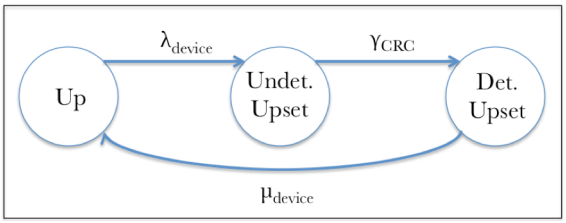

Figure 5. Markov model for CRC-32 scrubbing

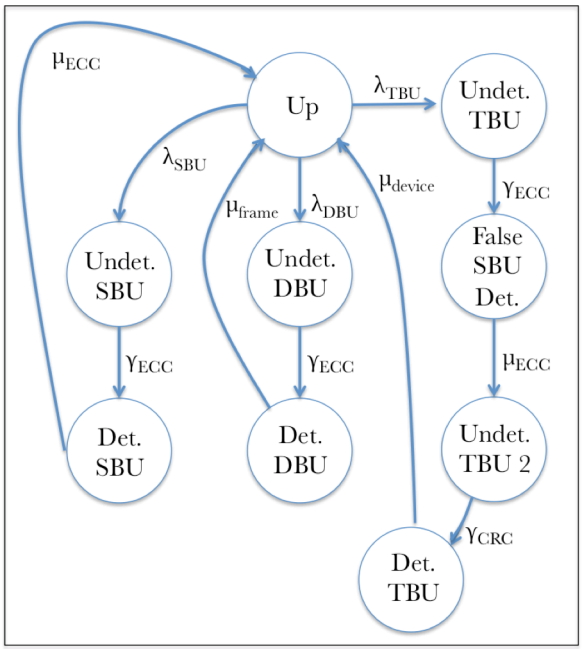

Figure 6. Markov model for Frame ECC scrubbing with CRC-32

For the analytical model, we assume the worst-case scenario that all MBUs occur within a single frame. Further, we assume that only one SEU occurs before the fault is detected, and that the upset frame, current reconfiguration frame, and current readback frame are selected at random. These assumptions provide an accurate estimate as long as the SEU rate is low compared to the total detection and correction time.

IV. Experimentation

We evaluated the three scrubbing strategies for steady-state availability using the SHARPE reliability-modeling tool. For these tests, we choose the largest V-5 and V-6 devices, and test them for both non-UC and UC mission conditions. For the non-UC mission, we assume that only 10% of upsets are critical and result in a failure. We scale the upset rates by this factor before evaluating the models, except in the case of three-bit upsets in the Frame ECC method since these could trigger a false correction affecting a critical bit. For the UC mission we assume that all detected upsets result in reset, and measure the availability for various system reset times. However, UC missions can use the essential bits file to determine if a detected SBU or DBU is critical. Of the scrubbing strategies under test, it can only be used with Frame ECC with the CRC-32 safety net, because this strategy identifies the upset bit, which can then be correlated with the mask file. We add this "Essential Bits" method as an additional test case, and scale the SBU and DBU rates accordingly assuming 10% essential bits. We do not test blind scrubbing for the UC mission because UC missions require detection.

A. Non-UC Results

Results for non-UC missions are shown in Table 4. The same trends hold for both devices in both orbits, so we observe that optimal scrubbing strategies may be independent of device and environment. We see that blind scrubbing offers the highest overall availability because we do not wait to correct errors until they are detected. We recommend CRC-32 scrubbing in cases where detection is desired, because it offers comparable availability with little additional implementation effort. Finally, the Frame ECC method offers the lowest availability for V-5, because TBUs can result in improper correction leading to increased downtime.

TABLE IV. Availability for Non-UC Missions

Scrubbing Techniques	Availability (nines)			
	Virtex-5 LX330T		Virtex-6 LX760T	
	LEO	HEO	LEO	HEO
Blind	5.17	4.72	4.62	3.22
CRC	5.07	4.63	4.53	3.13
Frame ECC	5.01	4.64	N/A	N/A

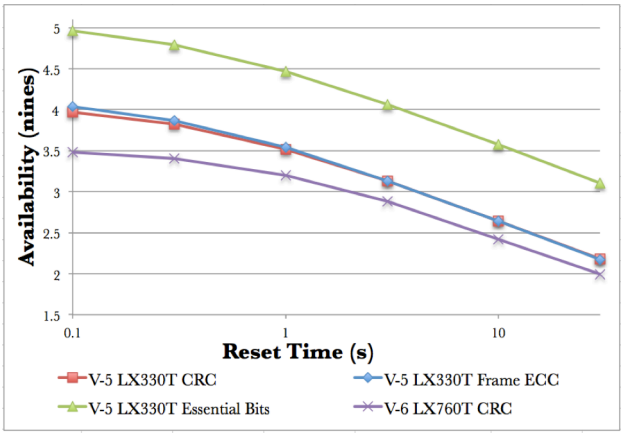

Figure 7. Availability vs. reset time in LEO for UC mission

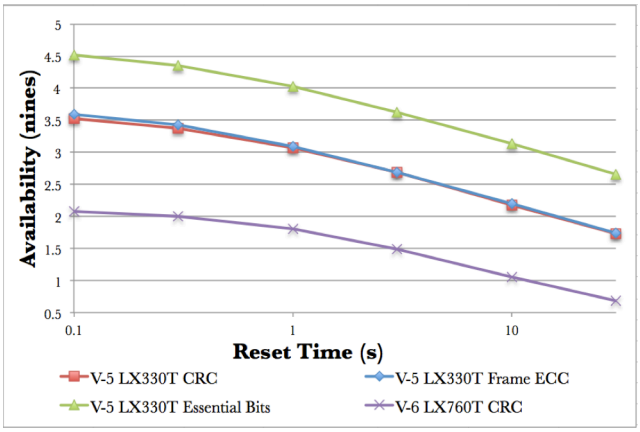

Figure 8. Availability vs. reset time in HEO for UC mission

B. *UC Results*

For the UC mission (Figs. 7 and 8), similar trends hold to the non-UC mission, with the exception of the new Frame ECC method using the essential bits mask. This approach offers roughly one additional nine of availability over CRC-32 or Frame ECC without the essential bits mask. We recommend this method for mission-critical systems that need the highest availability, despite the additional implementation effort and parts required for readback and essential bits mask storage and comparison. Overall availability decreases across the board with increased system reset time. For HEO orbits, the V-6 device shows particularly low availability because of its large λ_{bit} in HEO and large number of total configuration bits.

V. Conclusions

We have observed that the common scrubbing strategies in use for aerospace FPGA systems can be decomposed into correction and detection techniques for detailed analysis. Scrubbing parameters can be found for these techniques by simple analytical equations and combined with environmental and system parameters to populate a Markov model describing the scrubbing strategy. This model can be solved to find system availability, which we use as a metric to compare the various strategies across different environments and possible systems to find the best solution for a given set of parameters. This method allows us to design a system for maximum availability given our design constraints, rather than designing a scrubbing strategy around an implemented system.

By employing the SOAP method for hypothetical cases of V-5 and V-6 systems, we found that blind scrubbing, the simplest method, was also the most effective for non-UC missions. However, Frame ECC with the essential bits mask offered the highest availability for UC missions, although the implementation complexity is high. With increasing FPGA size, it will become even more difficult to maintain high availability for large designs in UC systems, necessitating faster and more advanced scrubbing techniques. Furthermore, as silicon feature size decreases, MBUs will become more prevalent and require more robust techniques to scrub them. The SOAP method aids the development of these new techniques by allowing designers to test and compare many hypothetical techniques through modeling rather than implementation.

Acknowlegments

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant Nos. EEC-0642422 and IIP-1161022. The authors gratefully acknowledge vendor equipment and tools provided by Xilinx and the SHARPE tool from Duke University that helped make this work possible.

References

[1] E. Fuller, M. Caffrey, P. Blain, C. Carmichael, N. Khalsa, and A. Salazar, "Radiation test results of the Virtex FPGA and ZBT SRAM for Space Based Reconfigurable Computing," 1999 MAPLD.

[2] K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," in ACM Computing Surveys, vol. 34, no. 2, June 2002, pp.171-210.

[3] "Virtex-5 FPGA configuration user guide," Xilinx UG191 (v3.9.1), Aug. 20, 2010.

[4] "Virtex-6 FPGA configuration user guide," Xilinx UG360 (v3.4), Nov. 18, 2011.

[5] H. Quinn, K. Morgan, P. Graham, J. Krone, and M. Caffrey, "Static proton and heavy ion testing of the Xilinx Virtex-5 device," Radiation Effects Data Workshop, 2007 IEEE.

[6] K. Chapman, "SEU strategies for Virtex-5 devices," Xilinx XAPP864 (v2.0), Apr. 2010.

[7] R. Le, "Soft error mitigation using prioritized essential bits," Xilinx XAPP538 (v1.0), Apr. 2012.

[8] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, et al., "Effectiveness of internal vs. external SEU scrubbing mitigation strategies in a Xilinx FPGA: Design, test, and analysis," 2007 RADECS.

[9] J. Heiner, N. Collins, M. Wirthlin, "Fault tolerant ICAP controller for high-reliable internal scrubbing," 2007 MAFA.

[10] R. Andraka, P. Brady, J. Brady, "A low complexity method for detecting configuration upset in SRAM based FPGAs," 2003 MAPLD.

[11] I. Koren and C. Krishna, Fault-Tolerant Systems. San Francisco, CA: Morgan-Kaufman, 2007.

[12] R. Sahner, S. Trivedi, "Reliability modeling using SHARPE," in IEEE Transactions on Reliability, vol. R-36, no. 2, June 1987, pp. 186-193.

[13] N. Wulf, A. George, and A. Gordon-Ross, "A framework to analyze, compare, and optimize high-performance, on-board processing systems," IEEE Aerospace Conference, 2012.

[14] H. Quinn, P. Graham, J. Krone, M. Caffrey, et al., "Radiation-induced multi-bit upsets in Xilinx SRAM-based FPGAs," 2005 MAPLD.