Reconfigurable Fault Tolerance: A Comprehensive Framework for Reliable and Adaptive FPGA-Based Space Computing

ADAM JACOBS, GRZEGORZ CIESLEWSKI, ALAN D. GEORGE, ANN GORDON-ROSS, and HERMAN LAM, University of Florida

Commercial SRAM-based, field-programmable gate arrays (FPGAs) have the potential to provide space applications with the necessary performance to meet next-generation mission requirements. However, mitigating an FPGA's susceptibility to single-event upset (SEU) radiation is challenging. Triple-modular redundancy (TMR) techniques are traditionally used to mitigate radiation effects, but TMR incurs substantial overheads such as increased area and power requirements. In order to reduce these overheads while still providing sufficient radiation mitigation, we propose a reconfigurable fault tolerance (RFT) framework that enables system designers to dynamically adjust a system's level of redundancy and fault mitigation based on the varying radiation incurred at different orbital positions. This framework includes an adaptive hardware architecture that leverages FPGA reconfigurable techniques to enable significant processing to be performed efficiently and reliably when environmental factors permit. To accurately estimate upset rates, we propose an upset rate modeling tool that captures time-varying radiation effects for arbitrary satellite orbits using a collection of existing, publically available tools and models. We perform fault-injection testing on a prototype RFT platform to validate the RFT architecture and RFT performability models. We combine our RFT hardware architecture and the modeled upset rates using phased-mission Markov modeling to estimate performability gains achievable using our framework for two case-study orbits.

Categories and Subject Descriptors: B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault Tolerance; C.4 [**Performance of Systems**]: *Fault tolerance*

General Terms: Reliability, Performance, Design

Additional Key Words and Phrases: FPGA, Reconfigurable fault tolerance, single-event upsets

ACM Reference Format:
Jacobs, A., Cieslewski, G., George, A. D., Gordon-Ross, A., and Lam, H. 2012. Reconfigurable fault tolerance: A comprehensive framework for reliable and adaptive FPGA-based space computing. ACM Trans. Reconfig. Technol. Syst. 5, 4, Article 21 (December 2012), 30 pages.
DOI = 10.1145/2392616.2392619 http://doi.acm.org/10.1145/2392616.2392619

1. INTRODUCTION

As remote sensor technology for space systems increases in fidelity, the amount of data collected by orbiting satellites and other space vehicles will continue to outpace the ability to transmit that data to other stations (e.g., ground stations, other satellites). Increasing future systems' onboard data processing capabilities can alleviate this downlink bottleneck, which is caused by bandwidth limitations and high latency transmission. Onboard data processing enables much of the raw data to be interpreted,

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422. The authors gratefully acknowledge vendor equipment and/or tools provided by various vendors that helped make this work possible.
Authors' address: A. Jacobs (corresponding author), NSF Center for High-Performance Reconfigurable Computing (CHREC), University of Florida; email: jacobs@hcs.ufl.edu.

DOI 10.1145/2392616.2392619 http://doi.acm.org/10.1145/2392616.2392619

reduced, and/or compressed onboard the space system before the results are transmitted to ground stations or other space systems, thus reducing data transmission requirements. For applications with low data transmission requirements, improved onboard data processing can enable more complex and autonomous capabilities. Finally, increased onboard data processing can enable future space systems to keep up with increasingly stringent real-time constraints. However, increasing the onboard data processing capabilities requires high-performance computing, which has largely been absent from space systems.

In addition to the high performance requirements of these enhanced future systems, space environments impose several other stringent, high-priority design constraints. System design decisions must consider the system's size, weight, and power (SWaP) and heat dissipation, which are dictated by the system's physical platform configuration. The satellite's physical dimensions restrict the system's size, the photovoltaic solar array's capacity restricts the power generation, and all heat dissipation must occur from passive, radiative cooling, which can be slow. In addition to SWaP requirements, system designers must consider system component reliability and system availability because faulty space systems are either impossible or prohibitively expensive to service. Radiation-hardened devices, with increased protection from long-term radiation exposure (total ionizing dose), provide system reliability. While device-level radiation hardening increases component lifetimes and reliability, these hardened devices are expensive and have dramatically reduced performance as compared to nonhardened commercial-off-the-shelf (COTS) components.

In order to design an optimal system, the performance, SWaP, and reliability requirements of future space applications must be considered together. System design philosophy must consider the worst-case operating scenario (e.g., typically worst-case radiation), which may dramatically limit the overall system performance even though the worst-case scenario may be infrequent (e.g., radiation levels typically change based on orbital position). However, if components used for redundancy and reliability can be dynamically repurposed to perform useful, additional computation, future space systems could meet both high-performance and high-reliability requirements. Future systems could maintain high reliability during worst-case operating environments while achieving high performance during less radiation-intensive periods. Current design methodologies do not account for this type of adaptability. Therefore, in order for future space systems to achieve high levels of performance, more sophisticated and adaptive system design methodologies are necessary.

One approach for adaptive high-performance space system design leverages hardware-adaptive devices such as field-programmable gate arrays (FPGAs), which provide parallel computations at a high level of performance per unit size, mass, and power [Williams et al. 2010]. Fortunately, many space applications, such as synthetic aperture radar (SAR) [Le et al. 2004], hyperspectral imaging (HSI) [Hsueh and Chang 2008], image compression [Gupta et al. 2006], and other image processing applications [Dawood et al. 2002], where onboard data processing can significantly reduce data transmission requirements, are amenable to an FPGA's highly parallel architecture. Since reconfiguration enables FPGAs to perform a wide variety of application-specific tasks, these systems can rival application-specific integrated circuit (ASIC) performance while maintaining a general-purpose processor's flexibility. An SRAM-based FPGA can be reconfigured multiple times within a single application, allowing a single FPGA to be used for multiple functions by time-multiplexing the FPGA's hardware resources, reducing the number of concurrently active processing modules when an application does not require all processing modules all of the time. Thus, FPGA reconfiguration facilitates small, lightweight, yet powerful systems that can be optimized for a space application's time-varying hardware requirements.

In order to leverage FPGAs in space systems, the FPGA must operate correctly and reliably in high-radiation environments, such as those found in near-Earth orbits. Currently, most radiation-hardened FPGAs have antifuse-based configuration memories that are immune to single-event upsets (SEUs). However, these hardened FPGAs have reconfiguration limitations and small capacities, reducing the primary performance benefits offered by COTS SRAM-based FPGAs. Fortunately, when combined with special system design techniques, SRAM-based FPGAs can be viable for space systems. An SRAM-based FPGA's primary computational limitation is the possibility of SEUs causing errors within the FPGA user logic and routing resources, which can manifest as configuration memory upsets or logic memory (e.g., flip-flops, user RAM) upsets (resulting in deviations from the expected application behavior). Fault-tolerant techniques, such as triple-modular redundancy (TMR) and memory scrubbing, can protect the system from most SEUs and significantly decrease the SEU-induced errors, but designing an FPGA-based space system using TMR introduces at least 200% area overhead for each protected module. Depending on the expected upset rates for a given space system, other lower-overhead fault tolerance methods could be used to provide sufficient reliability while maximizing the resources available for performance.

When designing a traditional space system, system designers estimate the expected worst-case upset rates and include an additional safety margin. However, since single-event upset (SEU) rates vary based on orbital position and the majority of orbital positions experience relatively low upset rates, a system designed for the worst-case upset-rate scenario contains processing resources that are wasted during the frequent low-upset-rate periods. In order to provide the necessary reliability during high-upset-rate periods and reduce the processing overhead incurred during low-upset-rate periods, the fault tolerance method must change based on the current upset rate. We propose a reconfigurable fault tolerance (RFT) framework that enables FPGA-based systems to dynamically adapt the amount of fault tolerance based on the current upset rate. For example, during high-upset-rate periods, the system can be reconfigured to provide high reliability at the expense of reduced processing capabilities, while during low-upset-rate periods the system can be reconfigured to provide higher performance by reprovisioning the excess hardware (used for high reliability during high-upset-rate periods) to application functionality. This upset-rate-based adaptability provides high performance while maintaining reliability. We present an RFT hardware framework for run-time support of multiple fault-tolerance methods on Xilinx FPGAs, along with several use scenarios. We leverage dynamic partial reconfiguration (PR) to change the fault-tolerance method during run-time without interrupting complete system execution (the regions not under reconfiguration remain active). We develop a fault-rate modeling tool to estimate the upset-rate variations during specific orbits, which is then used to provide more accurate modeling parameters for a Markov-based RFT system performability model. Modeling and performability analysis of the RFT framework is shown in the context of two relevant case studies.

The remaining sections of this article are organized as follows. Section 2 surveys previous work related to this topic. Section 3 describes the RFT hardware architecture, the RFT fault-rate model, and the RFT performability model. Section 4 analyzes results obtained using our RFT fault-rate and RFT performability models with one fault-injection case study and two orbital case studies. Finally, Section 5 presents conclusions and outlines directions for future research.

2. BACKGROUND AND RELATED WORK

In this section, we motivate the need for SRAM-based FPGAs in space systems. The FPGAs used in space systems must be able to withstand the amount of accumulated ionizing dose, measured as total ionizing dose (TID), expected during the mission to ensure long-term device functionality. Single-event effects (SEEs), caused by collisions with high-energy protons and heavy ions (radiation), are the primary short-term reliability concerns in space systems. In order for FPGA-based systems to maintain reliability, a combination of radiation-hardened FPGAs and complex fault-tolerance techniques are required to mitigate errors. Since many of the common fault-tolerance techniques require substantial temporal or spatial area overhead, new, low-overhead techniques allow more reconfigurable logic to be used for actual useful computation instead of for redundancy.

2.1. FPGA Performance and Power Efficiency

SRAM-based FPGAs offer a very large amount of configurable logic and have the ability to modify portions of a design during run-time, giving these FPGAs the ability to efficiently perform a wide range of applications by using a high degree of parallelism while running at low clock rates. Williams et al. [2010] developed computational density metrics to quantify and predict performance of SRAM-based FPGAs for specific types of algorithms and applications. Their analysis showed that FPGAs were capable of providing between 3 and 60 times more performance per unit power than many conventional general-purpose processors, depending on the types of operations being considered. The performance and power efficiency of FPGAs is extremely desirable for the small power budgets of space systems.

2.2. Partial Reconfiguration

Partial reconfiguration (PR) enables a user to modify a portion of an FPGA's configuration while the remainder of the FPGA remains operational. PR time-multiplexes mutually exclusive application-specific processing modules on the same hardware so that only the modules that need to be reconfigured halt operation, which makes PR attractive for real-time systems. Currently, Xilinx supports PR for the Virtex-4 and newer devices [Xilinx 2010a], while Altera has recently announced PR support for Stratix-V devices [Altera 2010].

During the system design phase, designers define the FPGA's partially reconfigurable regions (PRRs) and partially reconfigurable modules (PRMs) and route signals to/from the PRRs through bus macro PR primitives (Xilinx 9.2 PR tool flow) or partition pins (Xilinx 12.1 PR tool flow). Partial bitstreams, communicated through external configuration interfaces (e.g., SelectMAP, JTAG), are used to reconfigure the PRRs with the PRMs. On Xilinx devices, the Internal Configuration Access Port (ICAP) is an internal configuration interface, allowing user logic to directly reconfigure PRRs, removing the need for additional external configuration support devices. Additionally, since partial bitstreams are typically much smaller than full bitstreams (which are used to configure the entire FPGA), PR reduces bitstream storage requirements. Partial bitstreams may be significantly smaller than full bitstreams, depending upon the size of each PRR.

2.3. Single-Event Effects

SEEs occur when high-energy particles, such as protons, neutrons, or heavy ions, collide with silicon atoms on a device, depositing the ion's electric charge into the device's circuit. Protons and electrons are trapped within the Earth's Van Allen belts, while heavy ions are mainly produced by galactic cosmic rays and solar flares. When a high-energy particle collides with a silicon device, the energy of the collision can cause the logical values stored in sequential memory elements to be inverted [Karnik and Hazucha 2004]. Errors caused by these particles are often referred to as soft errors, or SEUs, as there is no permanent circuitry damage and any affected memories can be corrected by rewriting the correct values. Single-event functional interrupts (SEFIs), another type of SEE, can cause a semipermanent fault that requires a circuit to be power-cycled to restore correct operation. Single-event latchups (SELs) are destructive SEEs that occur when a particle causes a parasitic forward-biased structure within the device substrate that can allow destructively high amounts of current to flow through the substrate, potentially damaging the device. SELs can be avoided by using appropriate device manufacturing processes, and devices produced using silicon-on-insulator (SOI) processes are largely immune. SEL immunity is an important property when selecting devices for many space systems.

2.4. FPGAs in Space Systems

FPGA configuration memories can be constructed using several different technologies (e.g., antifuse, flash, and SRAM), each with performance and reliability tradeoffs. Traditionally, antifuse-based FPGAs have been used in space systems to provide simple processing capabilities and glue logic to interconnect multiple peripherals or to combine/replace the functionality of multiple ASICs. Antifuse-based FPGAs are one-time programmable, where the configuration process creates/fixes the FPGA's physical routing interconnect structure. This configuration process provides an inherent level of fault tolerance from SEUs since the antifuse-based routing cannot be reversed. Additionally, many commercially available antifuse-based FPGAs (e.g., [Actel 2010a]) include replicated flip-flop cells to prevent upsets in the sequential logic. Additionally, antifuse devices generally have a high TID threshold and immunity to SELs. Unfortunately, when compared to flash-based or SRAM-based FPGAs, antifuse-based FPGAs contain a relatively small amount of available logic gates and the fixed-logic structure limits performance potential.

Flash-based FPGAs (e.g., Actel RT-ProASIC3 [Actel 2010b]) attempt to maintain an antifuse-based FPGA's reliability while increasing the amount of configurable logic (logic available for a system designer to implement application functionality) and allowing reconfiguration. Flash-based FPGA configuration memories are composed of radiation-tolerant flash cells that provide reliability for combinational logic, however system designers must insert sequential logic replication to fully protect the FPGA from faults. Even though flash-based FPGAs can be fully reconfigured to support multiple applications, flash-based FPGAs do not support the PR capability that is available on some SRAM-based FPGAs due to a lack of architectural and vendor support for such a capability. Concerns over the TID effects on flash-based logic (floating-gate transistors) have prevented wide-spread acceptance of flash-based FPGAs in space systems [Wang 2003].

SRAM-based FPGAs are the most radiation-susceptible type of FPGA since the design functionality is stored in vulnerable SRAM cells (configuration memory), and configuration memory upsets cause functional changes to the design's logic. Traditionally, this vulnerability has prevented SRAM-based FPGAs from being used in highly critical space applications, however, some space systems use space-qualified SRAM-based FPGAs for onboard processing. Space-qualified FPGAs are similar to COTS FPGAs but are produced using epitaxial wafers, use ceramic, hermetically-sealed packaging, and have been tested to ensure that damaging SEL events will not compromise the system. These devices are rated for TID levels high enough to be used in space systems [Xilinx 2010c]. Even with these reliability techniques, these space systems must still use several fault-mitigation strategies, such as TMR and configuration scrubbing, to ensure that system upsets are detectable and recoverable. (We note that unless specified, all further FPGA references implicitly refer to SRAM-based FPGAs.)

Traditional FPGA-based space system designs leverage spatial TMR. TMR uses a reliable majority voter connected to three identical module replicas in order to detect and mask errors in any one module. In the context of TMR, a module refers to the functional unit being replicated, which can range from a single logic gate to an entire device. There are two primary TMR variations: external and internal. External TMR uses three independent FPGAs working in lockstep, where each FPGA implements a module replica and the outputs are connected to an external radiation-hardened voter that compares the results. External TMR requires significant hardware overhead (each protected module is triplicated and board layout complexity is significantly increased), but is reliable. The RCC board produced by SEAKR engineering [Troxel et al. 2008] uses external TMR to provide reliable computation using Xilinx Virtex-4 FPGAs. Alternatively, internal TMR creates three identical modules within a single FPGA, and the majority voter resides internally or externally [Carmichael et al. 1999]. Internal TMR can reduce the number of physical FPGAs required to implement a space application, but may increase the chance of a common-mode failure, where multiple modules fail simultaneously from a single fault. For example, a SEFI may cause multiple internally-replicated modules to fail, whereas externally-replicated FPGAs would be immune. Several tools assist system designers in incorporating TMR into space system designs [Pratt et al. 2006; Xilinx 2004].

In addition to TMR, configuration scrubbing prevents error accumulation in FPGA configuration memory. While TMR masks individual errors, it does not correct the underlying fault and cannot correct errors that occur in multiple modules. Scrubbing uses an external device to read back the FPGA's configuration memory and compares the read configuration memory to a known good copy. Alternatively, some FPGAs calculate an error correction code (ECC) during configuration read-back for every configuration frame, which can be used to detect and correct configuration faults. If a mismatch is detected, the correct configuration can be written using PR without halting the entire FPGA operation [Xilinx 2010a]. Traditionally, scrubbing is performed by an external radiation-hardened microcontroller to ensure reliability of the reconfiguration process. However, a self-scrubber may be implemented within the FPGA using the ICAP available in Xilinx Virtex-4 and newer devices.

Despite these drawbacks, SRAM-based FPGAs have been used in many space systems, including earth-observing science satellites, communication satellites, and satellites and rovers for the Venus and Mars missions. For instance, space-qualified Xilinx Virtex-1000 (XQVR1000) devices were used on the Mars Exploration Rovers for motor control functions and four XQR4000XLs were used for the lander pyrotechnics [Ratter 2004]. Configuration read-back and scrubbing were used for detection and correction of SEUs, and the full system was cycled once per Martian evening to remove persistent errors. These systems, which contained very little logic as compared to today's standards and lacked the ability for PR, were not used for data processing. The increased logic resources on recent FPGA families enable future systems to use FPGAs for image processing and other computation-intensive applications. The SpaceCube created at NASA Goddard Space Flight Center uses multiple commercial Xilinx Virtex-4 FPGAs along with a radiation-hardened processor to provide onboard processing capabilities for a variety of missions [Flatley 2010]. The SpaceCube provided computational power for the Relative Navigation Sensors (RNS) experiment during Hubble Servicing Mission 4 and is currently being used as an on-orbit test platform aboard the Naval Research Laboratory's MISSE-7 experiment on the International Space Station (ISS). Each FPGA in the system contains a self-scrubber module, protected with TMR, to correct errors and prevent error accumulation.

2.5. Low-Overhead Fault Tolerance Methods

While TMR is the most common fault tolerance method for FPGAs, the high area overhead due to replicating modules partially negates some FPGA benefits. Therefore, much research has focused on developing alternative, low-overhead fault-tolerance methods for low-upset environments, such as replication-based fault tolerance [Laprie et al. 1990], [Rao and Fujiwara 1989], and application-specific optimizations to provide low-cost reliability. Many of these alternative techniques can detect errors quickly, but may require additional processing or complete recomputation to correct the errors. When the expected upset rates are low, the recomputation rates may be acceptably low, depending on application throughput requirements.

Replication-based fault tolerance represents the most commonly used type of fault-mitigation strategy, due to conceptual simplicity and high fault coverage. TMR can detect single errors and can correct/mask single errors using a majority voter. Duplication with compare (DWC) is an alternative replication-based method that compares the outputs of duplicated modules. DWC reduces the resource overhead by one half as compared to TMR, but DWC cannot correct errors and must fully re-compute data when errors are detected [Johnson et al. 2008]. Shim et al. [2004] proposed reduced-precision redundancy (RPR) for numerical computation using either DWC or TMR, but reduced the resource overhead by only replicating the higher-order bits. RPR reduced the number of detectable faults (fault coverage), but resulted in significant resource savings while maintaining sufficient signal-to-noise ratios for many DSP applications. Morgan et al. [2007] investigated the use of temporal redundancy and quadded logic as additional methods for providing fault tolerance through redundancy. However, due to inefficient mapping to the underlying FPGA architecture, their methods did not provide fault tolerance improvement over TMR and imposed a large area overhead. The effects of the fault-tolerant methods were offset by the increased cross-sectional vulnerability of the larger FPGA designs.

Even though these replication-based methods for fault tolerance are suitable for FPGAs, several new fault-tolerant FPGA architectures leverage an FPGA's high capacity and flexibility while maintaining reliability. Alnajiar et al. [2009] proposed a hypothetical coarse-grained multicontext FPGA architecture that supported TMR, DWC, and single-context modes. Their work explored the effects of soft errors and aging on the proposed architecture, with an example Viterbi decoder module mapped to the hypothetical architecture. Kyriakoulakos and Pnevmatikatos [2009] proposed simple modifications to the current Virtex-5 and Virtex-6 architectures to allow native support for DWC and TMR. By adding an XOR-gate between each 5-input LUT within a larger 6-input LUT, the existing architecture and synthesis tools required only minimal changes to support their approach, while incurring only 17.5% to 76% slice utilization overhead for DWC or TMR, respectively.

Algorithm-based fault tolerance (ABFT) is a method that can be used with many linear algebra operations, such as matrix multiplication or LU decomposition [Huang and Abraham 1984]. ABFT augments an original data matrix with row and/or column checksums and the linear algebra operation is performed on the new, augmented matrix. If the linear algebra operation completes successfully, the resulting augmented matrix will contain valid, consistent checksums. ABFT checksum generation and comparison has lower computational complexity than the primary linear algebra operation. ABFT computational overhead is generally low, and as a proportion of total computation, decreases as the matrix size increases. Methods similar to ABFT can also protect data in FFT-based algorithms [Wang and Jha 1994], that are fully comprised of linear operations. Silva et al. [1998] investigated vulnerabilities in traditional ABFT implementations and proposed methods for improving fault coverage using a robust ABFT approach. ABFT may be used in FPGA applications to provide datapath protection with low overhead while other fault-tolerance methods, such as TMR, protect control logic.

3. RECONFIGURABLE FAULT TOLERANCE

RFT leverages COTS components in space systems to achieve high performance and flexibility while maintaining reliability. Beyond traditional, spatial TMR, alternative fault-mitigation methods may be appropriate given a particular application's performance requirements and set of expected environmental factors (e.g., radiation). Other methods, such as temporal TMR, ABFT, software-implemented fault tolerance (SIFT), or checkpointing and rollback, may be suitable for system-level protection. Each alternative fault-mitigation method has tradeoffs between performance, reliability, and overhead, and Pareto-optimal operation may change over a system's lifetime. Therefore, the main goal of our proposed RFT framework is to enable a system to autonomously adapt to Pareto-optimal operation based on the current system's environmental situation.

Our RFT framework consists of three main elements: a PR-based hardware architecture; a fault-rate model for estimating orbital SEU rates; and a methodology for modeling RFT system performability. The hardware architecture, described in Section 3.1, is similar to other traditional SoC architectures used for reconfigurable computing, with multiple, identical PRRs, which are leveraged for module redundancy. The hardware architecture allows the system to execute each processing module (PRM) in several possible fault-tolerance modes, each with differing performance and reliability characteristics. RFT software adapts the amount of per-module redundancy in accordance with the current environment and temporarily pauses only the reconfigured hardware modules to preserve and record application state while changing the fault-tolerance mode, allowing the remainder of the system to continue processing. Additionally, modules with constant state can continue processing during the adaptation process. Section 3.2 presents a model for estimating expected fault rates in potential space-system orbits. Finally, Section 3.3 presents a performability model that quantifies the RFT benefits in environments with varying upset rates.

3.1. RFT Hardware Architecture

Figure 1 shows the high-level architecture of an FPGA-based SoC design with PRRs integrated with an RFT controller. The main architectural components include a microprocessor, a memory controller, I/O ports, PRRs (1 . . . N) for PRMs, and the system interconnect, which connects all of these components to the microprocessor. Since we leverage Xilinx FPGAs, the microprocessor is a MicroBlaze (less resource-intensive processors such as PicoBlaze can also be used) and the system interconnect is a processor local bus (PLB). The MicroBlaze orchestrates PRR reconfiguration using the ICAP, maintains the state of the currently active PRMs, and initiates fault-tolerance mode switching.

All architectural components except for the PRRs are protected using TMR since these components' functionality is crucial to the entire system's reliability. Although the ICAP cannot be replicated, the signals to and from the ICAP are also protected with TMR. Tools such as Xilinx's TMRTool [Xilinx 2004] or BYU's EDIF-based TMR tool [Pratt et al. 2006] automate TMR design creation by applying low-level TMR voting on the design's original, unprotected netlist. For additional SEU protection, FPGA configuration scrubbing should be performed in order to prevent error accumulation. Scrubbing can be performed with an external scrubber and radiation-hardened configuration storage or with an internal scrubber using the internal configuration ECC present in Virtex-4 and later devices.

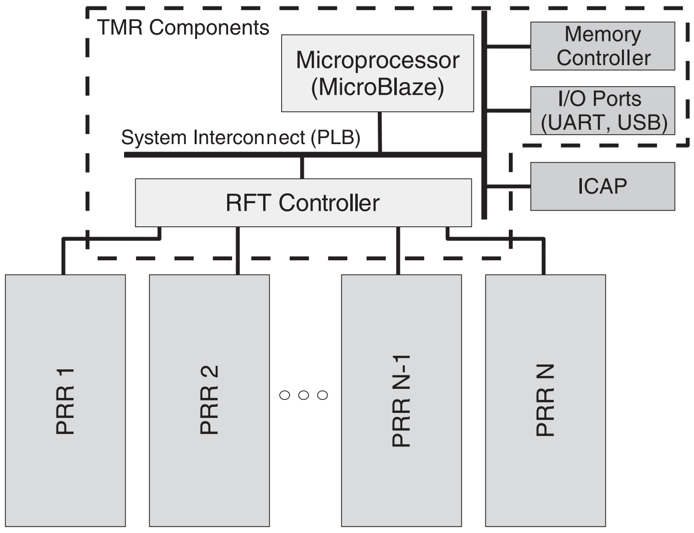


Fig. 1. System-on-chip architecture with RFT controller.

Table I. RFT Fault-Tolerance Modes

Fault-Tolerance Mode	Fault-Tolerance Type	PRRs Required
Triple Modular Redundancy (TMR)	Redundancy	3
Duplication with Compare (DWC)	Redundancy	2
High-Performance (HP) - no fault protection	Single-Module	1
Algorithm-Based Fault Tolerance (ABFT)	Single-Module	1
Internal TMR	Single-Module	1

Each PRM uses a PLB-compatible interface that connects to the RFT controller or directly to the PLB, based on whether the PRM is an RFT-enabled module (a module replicated/instantiated by the RFT controller) or not, respectively. The RFT controller instantiates the bus macros or other low-level components required for interfacing with the PRRs. The RFT controller also contains multiple majority voters and comparators (*voting logic*) that can be used to detect or correct errors by evaluating the replicated PRMs' outputs. The RFT-enabled modules are used in parallel to create redundancy-based, fault-protection modes (e.g., DWC, TMR) by interfacing with the RFT controller's voting logic. Additionally, other single-module fault-protection modes, such as ABFT, can be used for individual PRRs and the RFT controller provides additional fault tolerance components, such as watchdog timers, to detect hanging conditions within PRMs.

Table I lists the currently supported fault-tolerance modes of RFT and the modes' fault-tolerance type and PRR requirements. The MicroBlaze evaluates the system's current performance requirements and monitors external stimuli (radiation) using external sensors to determine when the fault-tolerance mode should be switched, at which time the MicroBlaze reconfigures the appropriate PRRs, and the RFT controller's internal voting scheme between the PRRs' outputs for the new fault-tolerance mode.

3.1.1. RFT Controller Operation. Figure 2 illustrates the interface between PRRs and the PLB for an RFT controller that can operate in single-module and redundancy-based fault-tolerance modes. This RFT controller interface routes input signals from the system interconnect to the appropriate PRRs and routes voting output signals

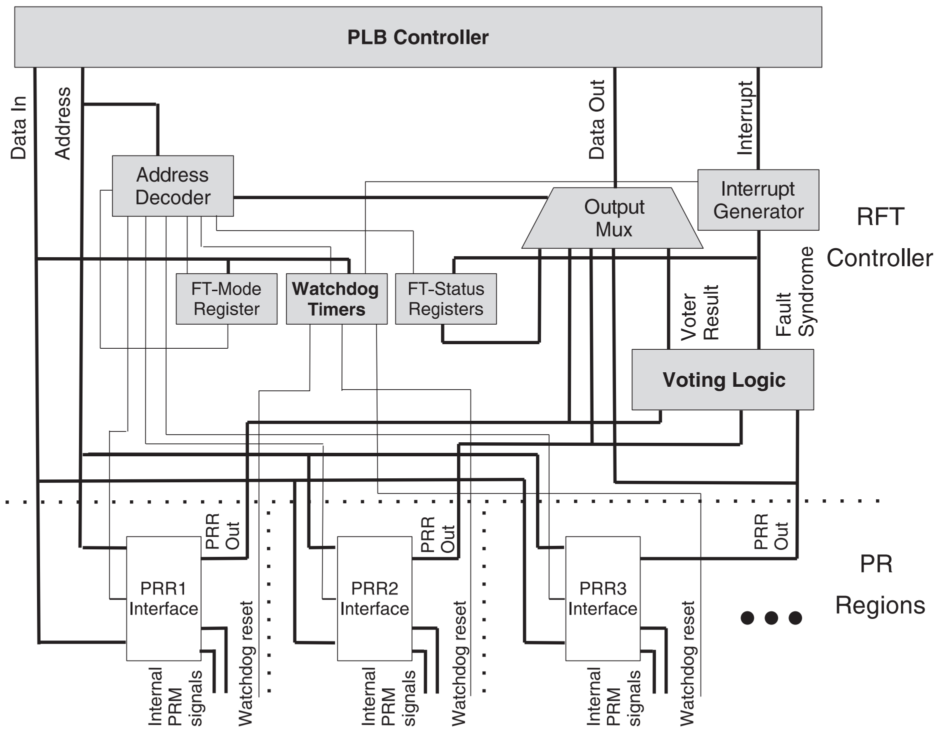


Fig. 2. RFT controller PLB-to-PRR interface.

back to the PLB. The abstraction of this logic into the RFT controller enables preexisting PRMs to leverage the fault-tolerance modes and interface with the RFT controller with minimum modifications. To communicate data and control signals between the MicroBlaze, the PRRs, and the RFT controller, at design time, the system designer assigns a large memory-mapped region of the MicroBlaze's address space to the RFT controller. In order to route signals to specified PRRs, the system designer also subdivides this region into smaller subregions and assigns a subregion to each PRR, while taking into consideration the memory interface requirements of each potential PRM. Each PRM must implement the actual memory interface (e.g., dual-port RAM, FIFO-based interface), which allows preexisting PRMs to interface with the RFT controller with minor modifications, since no specific interface is required. When communication data from the MicroBlaze arrives over the PLB, the RFT controller's address decoder processes the input address and determines the destination PRR(s) based on the current fault-tolerance mode. While operating in single-module fault-tolerance modes, the data is passed directly to the PRR specified by the decoded address. While operating in redundancy-based fault-tolerance modes, the data is passed to multiple PRRs, using appropriate enable signals to the PRRs' interfaces.

The RFT controller routes the outputs from each individual PRR, the outputs from the voting logic via the FT-mode register, or recorded internal status information, over the PLB to the MicroBlaze. When the MicroBlaze requests a PRR's output, the RFT controller's output multiplexer (*Output Mux* in Figure 2) selects the appropriate PRR output to route to the PLB controller based on the decoded address from the MicroBlaze and the current FT-Mode value. In single-module fault tolerance modes, the RFT controller's output multiplexer routes signals directly from the PRRs to the

PLB controller, which bypasses the RFT's internal voting logic. In redundancy-based fault-tolerance modes, the output multiplexer routes the verified outputs from the RFT controller's voting logic to the PLB controller.

In addition to providing routing and voting logic for redundancy-based fault-tolerance modes, the RFT controller supports additional fault-tolerance capabilities that do not require redundant PRMs. If the system is operating in a single-module fault-tolerance mode, each PRM may use the RFT controller's watchdog timers or perform internal fault detection. The RFT controller provides each PRR with an optional watchdog timer interface using a signal that must be asserted periodically within a user-defined time interval (usually on the order of seconds). If the PRM does not assert the watchdog timer reset signal within the time interval, the RFT controller's interrupt generator asserts an interrupt signal to the MicroBlaze, alerting the MicroBlaze of a possible failed PRR. Additionally, PRMs that perform internal fault detection must include an interrupt signal to notify the RFT controller of internally detected errors, which the RFT controller propagates to the MicroBlaze. A PRM using ABFT performs self-checking (or self-correction if the ABFT operation permits) on internally generated checksums and sends an interrupt signal to the RFT controller when data errors are detected. Similarly, PRMs that use internal TMR can also signal detected and corrected errors to the RFT controller. PRMs without any specific fault-tolerance features can also use the RFT controller's watchdog timer, which still allows the RFT controller to detect module hang-ups or other operational errors.

3.1.2. MicroBlaze Operation. In an RFT system, the MicroBlaze enables additional operational features, such as fault-tolerance methods to complement the fault-tolerance modes offered by the RFT controller. Additionally, the MicroBlaze maintains the system's fault-tolerance state (e.g., active PRMs, current PRRs' fault-tolerance modes, etc.) and orchestrates PRR reconfiguration and the fault-tolerance mode switching process.

To protect the application state while reconfiguring PRRs, the MicroBlaze provides support for PRM checkpointing and rollback [Bowen and Pradham 1993; Naeimi and DeHon 2008]. A checkpoint, which the MicroBlaze stores in external memory, consists of the minimum set of application state information needed to restore the application's current state. In the event of a fault, an application can use the previous checkpoint to roll back to a known good state, instead of wasting execution time by beginning execution at an initial starting state. A PRM's state can be checkpointed if the PRM can be read and modified by the MicroBlaze. In an RFT system, the state of all PRMs should be checkpointed periodically in order to reduce wasted computation in the event of a fault-induced PRM restart. The stored checkpoints can then be used during fault recovery procedures to improve system availability.

In addition to handling PRM checkpointing, the MicroBlaze also handles fault recovery and reconfiguration procedures. If the RFT controller's voting logic detects faults in the PRMs' outputs, the RFT controller records fault status information about the faulty PRM (e.g., error location, time, etc.) in internal FT-status registers and sends an interrupt to the MicroBlaze, which initiates the reconfiguration procedure. The FT-status registers record fault status information that may be used by the MicroBlaze to make fault-tolerance mode decisions or to provide system log information to system operators. For single-module fault-tolerance modes, the MicroBlaze corrects the faulty PRM by reconfiguring the PRR with the PRM's original bitstream over the ICAP. For the DWC fault-tolerance mode, both of the associated PRMs must be reconfigured since the faulty PRM cannot be identified. If checkpoints exist for the PRMs, the MicroBlaze initializes the new PRMs using these checkpoints. Alternatively, if the RFT system is operating in the TMR fault-tolerance mode, the MicroBlaze checkpoints one of the nonfaulty PRMs while the faulty PRM is reconfigured. After the nonfaulty PRM has been checkpointed, the RFT controller pauses the nonfaulty PRMs using clock gating to keep the PRMs synchronized. Once the faulty PRM has been reconfigured, the MicroBlaze initializes the newly reconfigured PRM from the most recent checkpoint, and the RFT controller can reenable the clocks for all three of the replicated PRMs to resume operation.

Finally, the MicroBlaze orchestrates the switching procedure between different RFT fault-tolerance modes. Fault-tolerance mode switching can be triggered by external events, by a priori knowledge of the operating environment, or by application-triggered events, and the fault-tolerance mode switching procedure may vary on a per-system basis depending on the specific system performance and reliability requirements. The MicroBlaze can use information from the fault-status registers to make Pareto-optimal decisions about future fault-tolerance modes. Before fault-tolerance mode switching, the MicroBlaze ensures that sufficient PRRs are available for the new fault-tolerance mode. Additional PRRs may be required or PRRs may be freed when switching from a single-module fault-tolerance mode to a redundancy-based fault-tolerance mode or vice versa, respectively. The MicroBlaze signals the RFT controller to change fault-tolerance modes by writing to the RFT controller's FT-Mode register. The MicroBlaze reconfigures the PRRs involved in the fault-tolerance mode switching via the ICAP with partial bitstreams for the appropriate PRM, or a blank partial bitstream if the PRR is not required for the new fault-tolerance mode. When the reconfiguration process is complete, the MicroBlaze signals the RFT controller, by rewriting the FT-Mode register, to resume PRM operation.

3.1.3. Environment-Based Fault Mitigation. RFT fault-tolerance mode switching can be triggered by a priori knowledge of the operating environment, by application-triggered events, or by external events. While a priori knowledge and application-triggered events are convenient for modeling purposes, real-world systems leverage measurements from attached sensors to determine the system's current environmental status. Additionally, due to the unpredictability of space weather conditions, such as solar flare events, an RFT system must be able to respond dynamically to the changing environment.

In an RFT system, the current expected fault rate can be estimated either directly or indirectly. An external radiation sensor can be directly interfaced with the FPGA, allowing the MicroBlaze to track the current fault rate and predict future fault rates. Alternatively, the RFT system can indirectly determine fault rates by tracking the number of data and configuration faults detected during operation. Since an FPGA's fabric is composed of large SRAM arrays, these arrays can be used as makeshift radiation detectors. If a fault is detected in the FPGA configuration or data memory, either through readback during scrubbing or from the RFT controller's logic, the fault can be recorded or can be used to make decisions about which RFT fault-tolerance mode to use. One simple, rule-based method for choosing the RFT fault-tolerance mode is to use a sliding window approach. For instance, a system may use the following rules.

(1) If there were any faults in the past 5 minutes, use the DWC fault-tolerance mode.
(2) If there were more than 5 faults in the past 5 minutes, use the TMR fault-tolerance mode.
(3) Only transition from the TMR fault-tolerance mode to a lower-reliability mode after 5 minutes of fault-free operation.

The size of the sliding window and the RFT fault-tolerance mode choices are system- and mission-dependent. A larger window size can provide a more conservative fault-tolerance strategy, while a small window size can more quickly adapt to spikes in

Table II. RFT Controller Resource Usage

Module Name	LUTs Used	FFs Used	Slices Used	FPGA Utilization
PLB Controller	479	375	345	1.4%
Address Decoder	238	0	136	0.5%
RFT Registers	82	64	48	0.2%
Watchdog Timers	498	390	366	1.4%
Voting Logic	438	0	234	0.9%
Output Mux	227	0	132	0.5%
Total	1962	829	1261	5.0%

the experienced fault rate. Implementing a rule with hysteresis, such as rule (3), can produce a more reliable strategy, while requiring a smaller window size. The time required for transitioning between RFT modes is dominated by the PRR reconfiguration time, which will be discussed in the next section.

3.1.4. RFT Controller Resource and Performance Overheads. To quantify the RFT controller's resource overhead, we implemented an RFT-based system on a Virtex-4 FX60-based platform, similar to the SpaceCube. The design uses six PRRs connected to a MicroBlaze through a PLB-connected RFT controller and operates at 100 MHz. Each PRR contains 2000 slices for user logic. Each PRR can operate in high-performance (HP) mode, DWC-mode with the PRR's neighboring PRRs, or TMR-mode with two consecutive, neighboring PRRs. The resources (LUT, FF, and slice) required for the RFT controller and constituent modules are detailed in Table II. The final column of Table II shows the percentage of the Virtex-4 FX60 used by each module. The RFT controller logic, excluding the PLB controller that would be required for non-RFT designs, requires approximately 900 slices. The largest modules within the RFT controller are the optional watchdog timers and the voting logic. Overall, the complete RFT controller uses approximately 5% of the total FPGA.

Since frequent PRR reconfiguration combined with lengthy PRR reconfiguration time can impose a performance overhead, we measured the reconfiguration of a single PRR. The PRR reconfiguration time was measured using timing functions on a MicroBlaze with a PLB-attached ICAP controller running at 100 MHz. We measured the PRR reconfiguration time for a 2000 slice PRR as approximately 15 ms. Even though the RFT mode-switching overhead is dominated by this PRR reconfiguration time, the mode-switching time incurs a negligible performance penalty due to the likely low frequency of mode-switching. Additionally, the switching overhead only occurs when increasing the fault tolerance redundancy (switching from DWC to TMR). When decreasing the fault tolerance redundancy, the retained modules can continue running without interruption.

3.2. RFT Fault-Rate Model

In order to analyze RFT's reliability benefits, a suitable fault-rate model that incorporates varying fault rates, system performance capabilities, and system fault-tolerance modes, is required. Traditional reliability analysis focuses on quantifying permanent hardware failures in systems with long lifetimes. Since many processors and FPGAs have a high TID and can withstand decades of use in near-Earth environments, permanent hardware failures due to long-term, end-of-life failures can be ignored. Therefore, reliability analysis for space systems focuses on short-term failure analysis by modeling SEU-induced computational failures. For FPGA-based systems, these failures can cause either single or multiple data errors through corrupted data or configuration memory. FPGA upset rates for space systems are correlated with the magnetic field strength of the system's current orbital position. For example, as a system passes

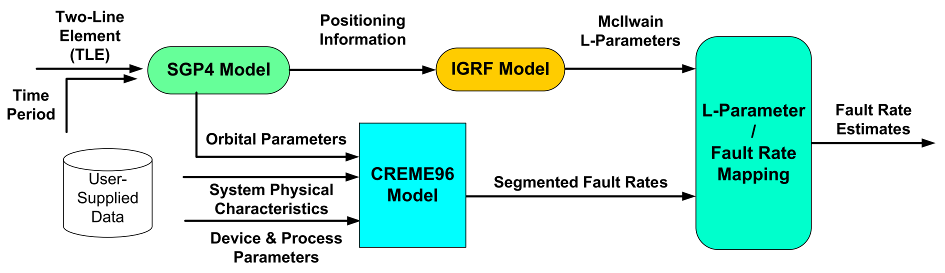


Fig. 3. RFT fault-rate model.

into the Van Allen radiation belt, the trapped charged particles in the belt have a higher likelihood of interacting with the system. For systems in a low-earth orbit, only a portion of the orbit passes through the inner Van Allen belt, at a location known as the South Atlantic Anomaly (SAA). The SAA represents the low-altitude area with a large number of trapped particles from the inner Van Allen belt's closest point to the Earth.

Existing fault-rate modeling generally produces a single, average orbital upset rate, however this average is not sufficient for RFT. In order for RFT to adapt the fault-tolerance mode, the fault-rate model must estimate the expected fault rates based on the instantaneous orbital position. To account for the orbit-dependent, time-varying fault rates, data from several sources can be combined to form a more accurate estimate than a single average rate. Our fault-rate model combines orbital position and trajectory, magnetic field strength, and cosmic ray particle interaction data to provide an accurate estimate of instantaneous fault rates. We use these fault-rate estimates as input into multiple system-level Markov models in order to calculate reliability, availability, and performability of RFT systems.

Our RFT fault-rate model combines three existing models to estimate time-varying fault rates. Figure 3 illustrates the three models used as well as the inputs and outputs to each model. To generate accurate fault estimates, a system's time-varying orbital position must be modeled. Orbital position can be estimated using the SGP4 model, which is a simplified general perturbation modeling algorithm. NORAD, which is responsible for tracking space objects and space debris, developed SGP4 for tracking near-earth satellites [Hoots and Roehrich 1980]. SGP4 accurately calculates a satellite's position given a set of orbital elements (apogee, perigee, inclination, etc.) collectively referred to as a two-line element (TLE). Given a system's TLE, the RFT fault-rate model uses SGP4 to generate the system's position information, in terms of latitude, longitude, and altitude, over the user-defined modeling time period. Next, the RFT fault-rate model passes the SGP4 positioning information for each point along a specified orbit to the International Association of Geomagnetism and Aeronomy's (IAGA) International Geomagnetic Reference Field (IGRF) model [Maus et al. 2005], which models the Earth's magnetosphere. IGRF combines magnetosphere data collected from satellites and observatories around the world in order to create the most accurate and up-to-date model possible (the model is updated every five years). For a given orbital position, IGRF outputs a McIlwain L-parameter, representing the set of magnetic field lines that cross the Earth's magnetic equator at a number of Earth-radii equal to the value of the L-parameter. The inner Van Allen radiation belt corresponds to L-values between 1.5 and 2.5. The outer Van Allen belt corresponds to L-values between 4 and 6. In addition to identifying regions with trapped particles, the McIlwain L-parameter can be used to estimate the effect of geomagnetic shielding (cutoff rigidity) from galactic cosmic rays. The estimated L-parameters are then used, along with the outputs of CREME96, to estimate SEU rates.

The CREME96 model [Tylka et al. 1997], a publically available SEU estimation tool, generates fault-rate estimates and has been used extensively to predict heavy ion and proton-induced SEU rates, as well as estimate the expected total ionizing dose in modern electronics. CREME96 combines orbital parameters, space system physical characteristics, and silicon device process information, to create a highly accurate SEU simulation. Traditionally, CREME96 generates an average fault-rate for a particular orbit, generated by averaging hundreds of orbits together. However, CREME96 can also generate fault rates for orbital segments, which can be segmented using the McIlwain L-parameter. By running several simulations with very narrow L-parameter segments, CREME96 can obtain estimated fault rates for each segment. As the width of each segment decreases, the generated fault rates become more precise and continuous. The L-parameter outputs from the IGRF model are then mapped to the appropriate orbital segment and the associated fault rate.

The SGP4, IGRF, and CREME96 models collectively generate a time-varying fault-rate estimate over the course of a specified orbit. We implemented the RFT fault-rate model as a C++-based program that connects the separate models together and passes data between them. SGP4's algorithms, along with C++/Java reference code, are publically available via the Internet. A FORTRAN-based implementation of the IGRF algorithm for calculating position-based McIlwain L-parameters is publically available from NASA Goddard [Macmillan and Maus 2010]. Fault-rate information can be generated from the CREME96 model through a Web-based interface, which is stored for efficient offline use. Orbits described by TLEs can be visualized using open-source tools, such as JSatTrak [Gano 2010]. The RFT fault-rate model program accepts TLE data as input and generates time-varying fault-rate estimates as output. These fault-rate estimates are then used by the RFT performability model.

3.3. RFT Performability Model

System reliability is the probability that a system is operating without faults after a specified time period. Assuming exponentially-distributed random faults at rate λ, the system reliability is traditionally defined as

$$R(t) = e^{-\lambda t}. \tag{1}$$

Mean-time-to-failure (MTTF) and *Mean-time-to-repair (MTTR)*, are the average amounts of time before a system encounters a failure (or repair) event. For a fault rate λ (or repair rate μ), MTTF (or MTTR) is defined as

$$MTTF = \int_0^\infty t \cdot R_\lambda(t)\,\mathrm{d}t \qquad MTTR = \int_0^\infty t \cdot R_\mu(t)\,\mathrm{d}t. \tag{2}$$

System availability, which is similar to system reliability, estimates the long-term, steady-state probability that the system is operating correctly, and is defined as

$$A = \frac{MTTF}{MTTF + MTTR}. \tag{3}$$

System unavailability is the opposite of availability, and is often used for convenience when discussing systems with very high availability. Unavailability is defined as

$$UA = 1 - A = \frac{MTTR}{MTTF + MTTR}. \tag{4}$$

Space system reliability and availability can be accurately modeled using Markov models. A Markov model is composed of states and transition rates. A state represents the current operating state of the system and the transition rates represent the transitions from an operating state to a failure state (failure rates), or from a failure state to an operating state (repair rates). The Markov model can be transformed into a series of equations, which can be solved or approximated numerically using tools such as SHARPE, an open-source fault-modeling tool [Sahner and Trivedi 1987], to determine probabilities of each state. System reliability and availability can be directly determined from the calculated state probabilities. For Markov models, the *instantaneous availability* of repairable systems measures the probability of being in an available as opposed failed state at a given point in time. These types of models are frequently used to estimate the effects of TMR, scrubbing, and other fault-tolerance methods in FPGAs and other electronics [Dobias et al. 2005; Garvie and Thompson 2004; Pratt et al. 2007].

Markov reward models, a type of weighted Markov model, can be used to extend the concept of system availability to measure *system performability* of adaptable systems [Ciardo et al. 1990; Meyer 1982]. Performability is a metric that combines system availability with the amount of work produced by the system and gives a measure of total work performed. Performability is especially useful for gracefully degradable systems or other systems that have changing characteristics over time. Assuming that $X(t)$ is a semi-Markov process with state space S and is continuous over time $t > 0$, the instantaneous performability is defined by

$$Performability(t) = \sum_{a \in S} Perf(a) \cdot P\{X(t) = a\}, \tag{5}$$

where $Perf(a)$ is the system performance in state a. System performance can be defined using any desired performance metric (e.g., throughput, execution time, etc.) and performability is measured similarly. In this context, *instantaneous availability* can be viewed as a special case in which $Perf(a) = 1$ in available states and $Perf(a) = 0$ otherwise.

For reconfigurable FPGA systems, where the system configuration changes over time (e.g., our RFT architecture), the system must be modeled as a phased-mission system. A phased-mission system is described using a set of unique models for each phase of the mission. The states at the end of a given phase's model map to the states of the following phase's model during phase transitions, and the phase duration can be modeled as either probabilistic or deterministic [Alam et al. 2006; Kim and Park 1994].

We use the RFT fault-rate model's generated fault-rate estimates to drive multiple system-level Markov models in order to calculate reliability, availability, and performability of RFT systems. In order to incorporate varying fault rates (due to orbital position) and varying system topologies (due to RFT), we leverage a phased-mission Markov approach. We model the RFT system as a collection of individual phases, with each phase consisting of a period of time where the fault-tolerance mode, failure rates, and repair rates are constant. The phase lengths are both application-dependent and orbit-dependent. At the end of each phase, the pretransition state probabilities are mapped onto initial probabilities for the posttransition Markov model. Figure 4 illustrates an example high-level model transitioning from TMR to DWC at time t_1, and then transitioning from DWC back to TMR at time t_2. Fault rates (λ) and repair rates (μ) are represented as directed graph edges between states. At each phase transition (denoted by the dashed vertical lines), state probabilities are remapped (dashed arrows). When remapping state probabilities from TMR to DWC, two TMR states are merged into a single operational state. When remapping from DWC to TMR, the single operational DWC state maps to the most-similar TMR state. Each fault-tolerance mode

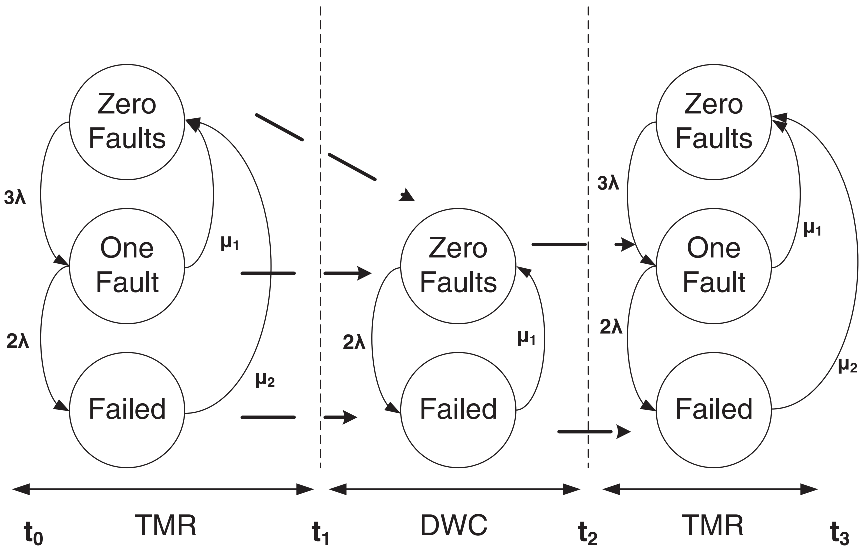


Fig. 4. Phased-mission Markov model transitioning between TMR and DWC modes.

has an associated Markov model. Each state of the Markov model corresponds to the number of operational devices in a system (e.g., PRRs in an FPGA-based SoC). Transitions between states occur when a device changes state from operational to failed, or vice-versa. FPGA upset rates are estimated using the RFT fault-rate model described in Section 3.2. The system repair rates are based on the system-designer-specified *scrub rate* and *checkpointing rate* and the system and PRR reconfiguration times, which can be obtained experimentally. Transitions between fault-tolerance modes are mission-dependent. State-mapping functions ensure a continuous availability function, although performability may contain discontinuities at phase transitions.

Figure 5 shows Markov model representations for each fault-tolerance mode in a system with six PRRs. Figures 5(a) and 5(b) represent systems where each PRR is operating in the HP or ABFT fault-tolerance modes, respectively. Figure 5(c) represents a system using three independent pairs of PRMs operating in the DWC fault-tolerance mode. Figure 5(d) represents a system using two independent sets of three PRMs operating in the TMR fault-tolerance mode. Solid circles represent the Markov model's available operating states and dashed circles represent failed operating states. Using the definition of performability from Eq. (5), each state in a Markov reward model is assigned a performance throughput value. For this analysis, system performance is normalized to the work performed by a single PRR. The performance throughput value of each state is represented in the Markov model by the value in the rectangles. For example, six independent PRRs running concurrently in the HP fault-tolerance mode would have a performance throughput value of 6 while a system using six PRRs with two independent sets of PRRs operating in the TMR fault-tolerance mode would have a performance throughput value of 2.

The ABFT Markov model is used to demonstrate a generic reliability model for modules that contain some form of internal fault tolerance and are capable of self-detecting data errors. In particular, ABFT provides a low-overhead method for detecting errors in certain linear algebra operations. While hardware-implemented ABFT may not have 100% fault coverage, hardware-implemented ABFT provides improvements over the HP model, which cannot detect when corrupt data is returned.

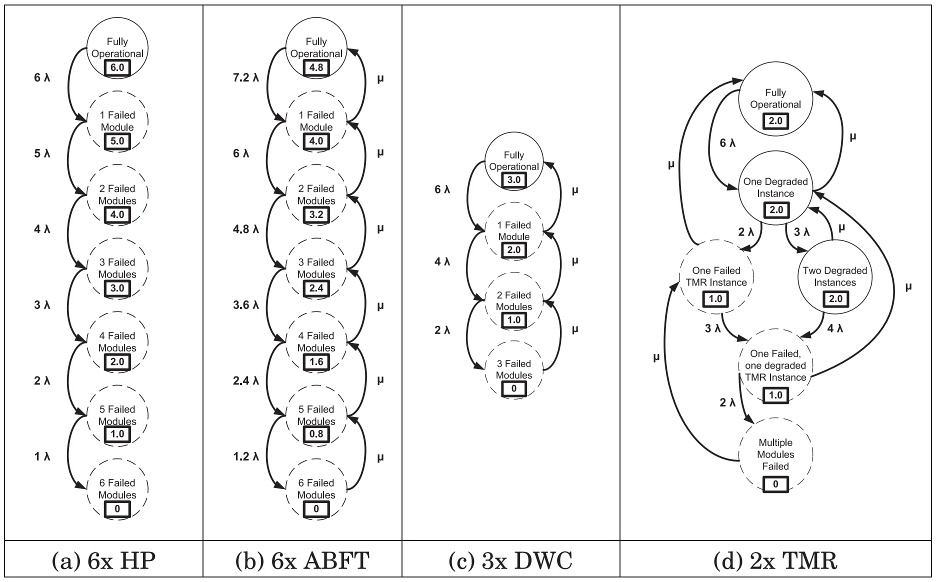


Fig. 5. Markov models of RFT modes.

In the Markov model for the single-module ABFT mode, we estimate the performance of a single module as 80% of the default, unprotected module due to the performance overhead associated with generating and comparing checksums for fault detection [Acree et al. 1993]. Additionally, ABFT fault rates are modeled as having a 20% higher fault rate than unprotected modules due to the increased module size from the ABFT logic. The repair rate for the ABFT model uses the system's scrubbing rate, supplying a worst-case repair rate for cases when ABFT logic does not detect (or even introduces) an error, relying on external scrubbing and configuration readback to detect errors. Although these numbers are application- and implementation-specific, the numbers represent overhead costs that must be considered. The ABFT model can also be used to model other modules by user-implemented fault tolerance techniques. For a given phased-mission Markov model, a TLE is used to determine the expected fault rates using the RFT fault-rate model in Section 3.2. These fault rates, along with a description of the mission-specific criteria for using each fault-tolerance mode, are used to split a full space mission into distinct phases for Markov modeling. RFT phases are periods of time with constant fault rates, repair rates, and fault-tolerance modes. For each phase, the previous phase's state probabilities are mapped onto the new phase's Markov model as initial probabilities. SHARPE processes the new model to numerically solve the state probabilities and reliability metrics for the current phase. SHARPE's results provide the initial probabilities for the next phase. This process is repeated iteratively for each phase for the entire mission's duration and SHARPE's results are aggregated to produce overall reliability results.

4. RESULTS AND ANALYSIS

In this section, we present one validation case study and two orbital case studies to evaluate the potential reliability and performance benefits of our RFT framework for space systems. The validation case study uses FPGA fault injection to estimate fault rates and error coverage, which can then be used by the other case studies. The orbital

Table III. Fault-Injection Results for RFT Components

System Component	Faults Injected	Data Errors	System Hangs	Vulnerable Bits (est.)	DVF (%)
Matrix Multiply (MM)	100,000	1,501	71	41,497	0.197%
RFT Controller (RFT)	100,000	63	157	4,576	0.022%
MicroBlaze (MB)	100,000	150	1,219	86,584	0.342%
Full FPGA					0.955%

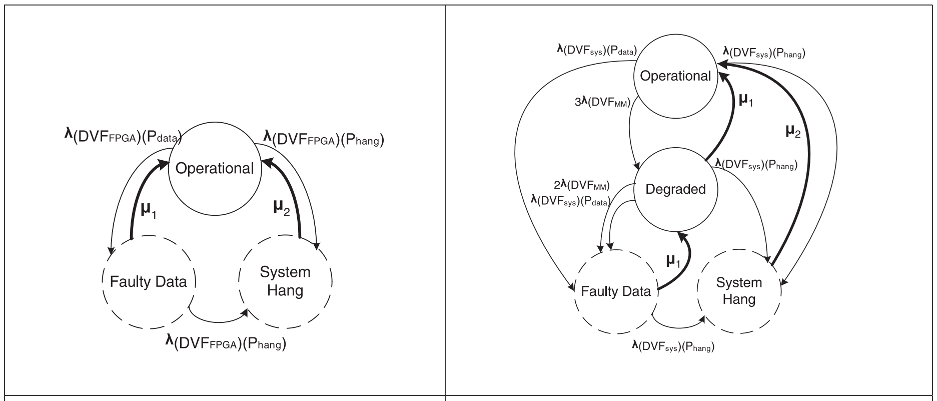
case studies represent FPGA-based space systems operating in two common orbits, with multiple performance and reliability requirements. The first case study represents a space system operating in low-Earth orbit (LEO), the Earth Observing-1 (EO-1) satellite. Space systems in a LEO experience relatively low radiation. The second case study represents a space system operating in a highly elliptical orbit (HEO), which is a much harsher radiation orbit. Each case study will compare multiple adaptive fault-tolerance strategies to a traditional static TMR strategy.

4.1. Validation Case Study

In order to validate our reliability models, faults must be injected into an executing RFT system. In this section, we present FPGA fault-injection results gathered using the Simple, Portable Fault Injector (SPFI) [Cieslewski et al. 2010]. SPFI performs fault injection using both full and partial reconfiguration, which reduces the time required to modify configuration memory and improves the speed of fault injection. However, due to the time required to inject and test each fault, testing every bit in the configuration memory may be infeasible. Instead, SPFI uses random sampling and a statistical approach to estimate design vulnerability. We validate our reliability models by correlating SPFI's results with analytical Markov model results.

For the validation case study, we implemented a simplified RFT-based system on a Xilinx ML505 FPGA development platform. The RFT-based system had a 3-PRR RFT controller that allowed HP and TMR voting and had watchdog timer functionality. Each PRR contained a matrix multiplication (MM) PRM; a MicroBlaze processor in the static region streamed data from a UART to an MM PRM and streamed results back to the UART. The SPFI fault-injection tool enabled individual system components (e.g., PRMs, RFT controller, MicroBlaze) to be tested independently without modifying the entire system. Table III shows the RFT systems' fault-injection results. For each system component, Table III indicates the number of injections performed and the number of data errors and system hangs detected. The fault vulnerability for each component is scaled to the FPGA's total number of configuration bits to estimate the components' design vulnerability factor (DVF). A component's/device's DVF represents the percentage of bits that are vulnerable to faults and can result in observable errors.

Most Xilinx FPGA designs have a DVF that ranges from 1%–10% [Xilinx 2010b] due to the large amount of configuration memory devoted to routing. The DVF for each component is calculated by measuring the component's fault rate, estimating the number of vulnerable bits by scaling the fault rate to the size of the area occupied by the component, and dividing the number of vulnerable bits by the total number of FPGA configuration bits. For the following experiments, 100,000 bits were randomly selected from each region under test in order to estimate the DVF within a narrow 95% confidence interval. For a single MM PRM with the RFT controller using HP mode, only 1.6% of faults that occurred in the PRR were found to cause observable errors in the output. Approximately 41,497 vulnerable bits in each PRR, which occupies approximately one-eighth of the FPGA, results in a DVF_{MM} of 0.197%. In this case, the majority of the PRR is unused, resulting in very few vulnerable bits and a



(a) HP mode	(b) TMR mode
$\mu_1 = ScrubRate$	$\mu_2 = ScrubRate + ConfigurationTime$

Fig. 6. RFT validation Markov models.

low DVF. Faults injected into the RFT controller resulted in a DVF_{RFT} of 0.022%. The MicroBlaze was not protected using TMR or other design techniques and had a DVF_{MB} of 0.342%. Based on these component fault-injection results, the total FPGA DVF is estimated to be 0.955% ($3DVF_{MM} + DVF_{RFT} + DVF_{MB}$).

Figure 6 shows Markov model representations for each fault-tolerance mode in a system with three PRRs. Figures 6(a) and 6(b) represent systems where each PRR is operating in the HP or the TMR fault-tolerance mode, respectively. These models are similar to the models presented in Section 3.3, without additional states for performability, and with the addition of state transitions to account for fault coverage. In a TMR system, coverage refers to the percentage of faults that cause the system to immediately enter the failed state. These noncovered faults occur due to designs not being fully protected by TMR.

Initial fault-injection testing revealed that our system appeared to have two possible fault scenarios. In the first scenario, a fault could cause the system to remain operational but produce erroneous data. This state was recoverable using periodic scrubbing. In the second scenario, a fault could cause the system to hang until a full reconfiguration was performed. We expanded the Markov models to include these behaviors as additional states. Figure 6(a) shows the HP Markov model with two unavailable states that account for the two fault scenarios. The probabilities of transitioning to "Faulty Data" or "System Hang" were determined from the component fault-injection testing. The DVF_{FPGA} was estimated from the sum of each of the components tested.

A similar approach was taken for the TMR Markov model shown in Figure 6(b). The additional "degraded" state is used to model faults that have been masked by the TMR protection provided by the RFT controller. The probability of transitioning to the degraded state is provided by the DVF_{MM} from previous testing. The DVF_{sys} term represents faults in the RFT controller (DVF_{RFT}) and MicroBlaze (DVF_{MB}). From fault-injection testing, we estimate the DVF_{sys} to be approximately 0.364%.

By assigning fault rates, repair rates, and coverage to the Markov model, we can calculate the system availability. Table IV shows the fault and repair rates used in this analysis. Using a scrub rate of 1 fault per 10 seconds, and a repair rate of 1 scrub per 5 seconds, the RFT system will have a 98.82% availability in HP mode and 99.31% availability in TMR mode. When the fault rate to scrub rate ratio is increased, the benefits of TMR become more pronounced. Using a scrub rate of 1 fault per 2 seconds, and

Table IV. RFT Markov Model Validation

Fault Period (time/fault)	Repair Period (time/scrub)	FT-Mode	Markov Model Availability	Experimental Availability	Model Error
10s	5s	HP	98.82%	98.99%	0.2%
10s	5s	TMR	99.31%	99.11%	0.2%
2s	10s	HP	92.25%	92.59%	0.4%
2s	10s	TMR	95.32%	93.89%	1.5%

a repair rate of 1 scrub per 10 seconds, the RFT system will have a 92.25% availability in HP mode and 95.32% availability in TMR mode. These high availabilities, even in HP mode, are due to frequent scrubbing and the very low DVF of the FPGA design.

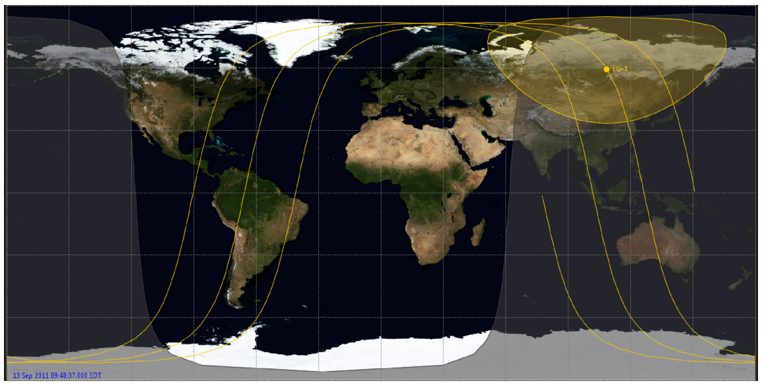
Finally, we validate the Markov model results using fault injection. In a continuously-running RFT system, faults are injected at a specified rate, which is randomly varied using a Poisson distribution to simulate the error model used in the Markov model. Scrubbing, using partial reconfiguration, occurs at user-defined periodic intervals. Full reconfiguration occurs at the next scrubbing cycle after a PRM error has been detected by the RFT controller. Full reconfigurations also occur if the external testing program detects that the FPGA system has entered the System Hang state. Availability can be experimentally determined by the ratio of time the system is operating correctly to the total experiment run-time. For each run, 10,000 faults were randomly injected into a running system. Table IV shows the results of the availability experiment and the relative error from the analytical model. At low fault rates, the HP and TMR modes both provide approximately 99% availability. With high fault rates, the system had an availability of 92.59% in the HP-mode and 93.89% in the TMR-mode.

The Markov model availability methodology provides a simple and effective method for determining the effects of fault and repair rates on system availability without exhaustive testing. In general, the HP models predicted slightly lower unavailability than what was observed during experimental testing, while the TMR models overestimated the availability of the system. All availability results were within 1.5% of the Markov models' predictions. The Markov model accuracy can be improved by providing more accurate fault-injection results. The availability values obtained through experiments can be improved by increasing the length of the testing period.

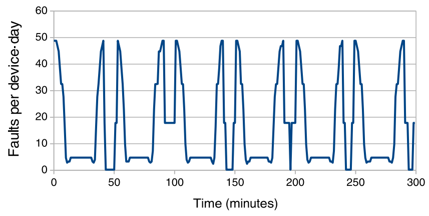
Fault-injection testing did highlight implementation issues that must be handled in any high-reliability design. The use of TMR had a lower than expected benefit due to the unprotected MicroBlaze processor. Since the DVF_{MB} was larger than the DVF_{MM}, the availability of the system was dominated by the MicroBlaze's availability. For high reliability, the MicroBlaze must be protected with TMR or an alternative fault-tolerant processor (e.g., FT-LEON3).

4.2. Orbital Case Studies

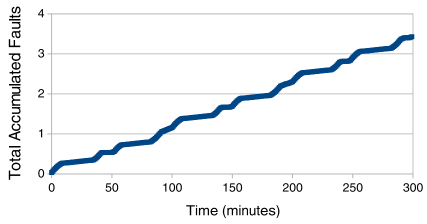
For each orbital case study, the system under test is an FPGA system-on-chip implementation of the RFT hardware architecture described in Section 3.1. In order to calculate device vulnerability, the parameters for the Xilinx Virtex-4 FX60 will be used as inputs to the RFT fault-rate model described in Section 3.2. The radiation susceptibility parameters of the Virtex-4 device family are obtained from the Xilinx Radiation Test Consortium's (XRTC) published results [Swift et al. 2008]. The generated fault rate is linearly scaled from a full device to the size of the PRR to produce PRR fault rates. The RFT controller is connected to 6 PRRs, allowing for several combinations of the TMR, DWC, and ABFT fault-tolerance modes discussed in Section 3.3. The performability model from Section 3.3 is used to evaluate the effectiveness of each fault-tolerance strategy.



(a) Visualization of the EO-1 TLE data using JSatTrak [Gano 2010]



(b) Expected fault rates over several orbits



(c) Total accumulated faults over several orbits

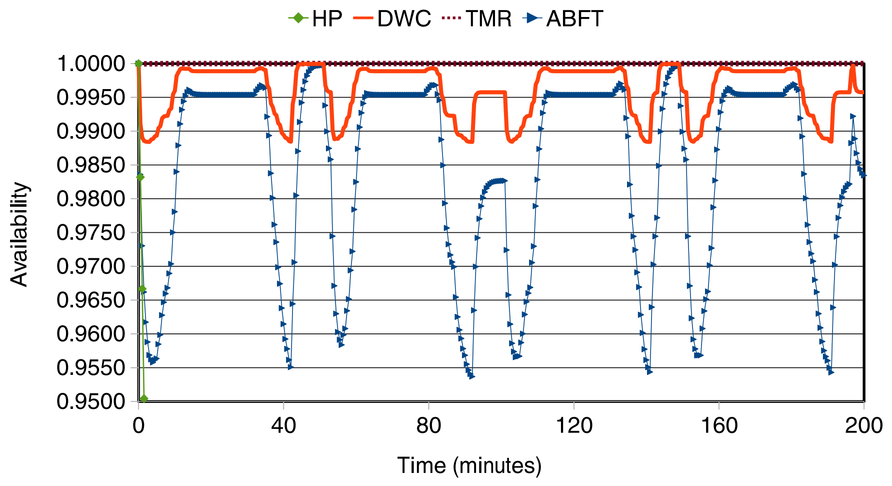
Fig. 7. LEO fault rates using the RFT fault-rate model.

4.2.1. Low-Earth Orbit Case Study. Space systems in LEO are commonly used for earth-observing science applications, such as HSI or SAR. Both HSI and SAR have large datasets, which can be significantly reduced through onboard processing using a variety of algorithms that are decomposable into basic kernels that can be parallelized and implemented on an FPGA system for high performance and power efficiency. For example, HSI can be decomposed into a sequence of matrix multiplications and matrix inversions, and SAR can be decomposed into vector multiplication and FFTs. Since these mathematical operations are linear, ABFT can be used to protect the computation results from SEUs. For applications that cannot be protected with ABFT, TMR or DWC must be used to provide fault tolerance.

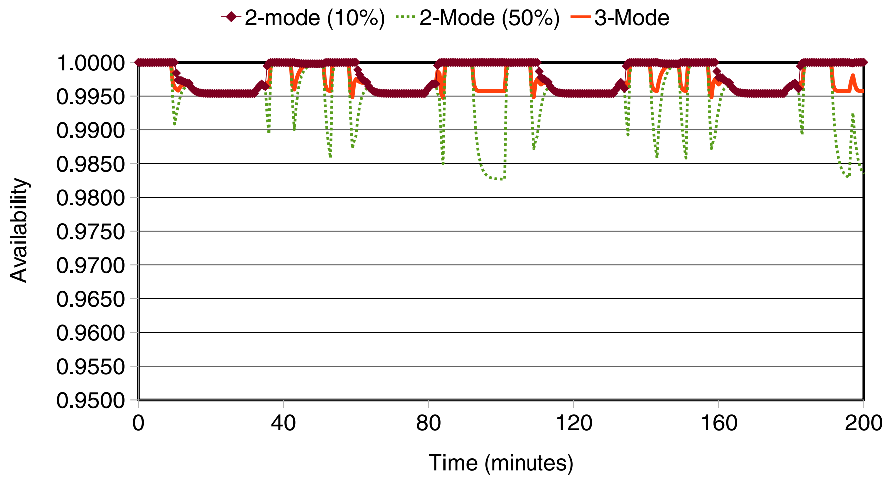
The TLE used to generate the fault rates for this LEO case study is from the EO-1 satellite. The EO-1 orbit is circular at an altitude of 700 km and a 98.12° inclination with a mean travel time of 98 minutes. Figure 7(a) shows the orbital track of EO-1 and the shaded circle represents the EO-1's field of view. Figure 7(b) shows the estimated number of upsets per hour that occur in the EO-1 orbit over several orbital periods. The average fault rate of the Virtex-4 FX60 in the EO-1's orbit is 16.5 faults per device-day (combined configuration memory and BRAM vulnerability). Each local maximum occurs when the satellite is closest to the Earth's magnetic poles. Figure 7(c) shows the total number of accumulated faults over time. Fault rates in EO-1's orbit are low because the orbit is lower than the Van Allen Belts and is fully within the Earth's magnetosphere, which deflects a large amount of radiation. We examine the availability and performability of TMR, DWC, and ABFT fault-tolerance modes in LEO, as well as three adaptive fault-tolerance strategies to maximize application performability: 10% two-mode, 50% two-mode, and three-mode adaptive strategies. For the adaptive fault-tolerance strategies, the fault-tolerance mode switching is determined by comparing the current upset rate with a fault-rate threshold. The 10% two-mode adaptive strategy uses the ABFT fault-tolerance mode when upset rate is in the lowest 10% of the expected fault rates and the TMR fault-tolerance mode otherwise. The 50% two-mode adaptive strategy uses a similar strategy as the 10% two-mode, but with a higher fault-rate threshold. The three-mode adaptive strategy uses ABFT when the upset rate is in the lowest 10% of the expected fault rates, TMR when the upset rate is in the highest 50% of the expected fault rates, and DWC otherwise. In all modes, the system performs scrubbing to ensure that configuration memory errors are removed from the system. For this LEO case study, the system uses a 60-second scrub cycle, which is also used as the system repair rate for the Markov models. Figure 8(a) shows the availability of the LEO system while statically using the four fault-tolerance models described in Figure 5. While the static HP strategy's availability quickly declines (due to the lack of any fault-tolerance mechanisms), the static TMR, DWC, and ABFT strategies all maintain availability above 95%. The dynamically changing availability is directly related to the current fault rate, however, since the repair rate of the system (through scrubbing) is much larger than the expected fault rates, most configuration memory faults are rapidly mitigated. The system availability while using the three adaptive fault-tolerance strategies is shown in Figure 8(b). The average availability for each adaptive strategy improves availability over the static ABFT strategy, and the adaptive strategies can increase the minimum availability to above 99.5%. Table V displays the availability results for each fault-tolerance strategy in terms of unavailability, showing the probability of a system failure. The 10% two-mode adaptive strategy reduces average unavailability by 88% as compared to the static ABFT strategy. For extremely high availability, static TMR is required, as the maximum unavailability for the adaptive strategies is more than 100 times higher than TMR.

The system performability for the static and adaptive fault-tolerance strategies for a LEO is also shown in Table V. Due to the low overall upset rates and good availability of the static ABFT strategy, the static ABFT strategy achieves the highest performability. The 50% two-mode adaptive strategy achieves an average performability throughput of 4.01, a 100% improvement over static TMR, while improving unavailability over the static ABFT strategy by 73%. The 10% two-mode strategy has lower average performability, while maintaining better system availability. The three-mode strategy has better performability than the static DWC mode, while having better availability than the static DWC or ABFT modes, but is outperformed by the 50% two-mode strategy. We point out that the fault-rate threshold for the two-mode strategies can be used to adjust the availability and performability parameters for a space system. Figure 9 illustrates the effect of changing the threshold of the two-mode adaptive strategy for the LEO case study. As the threshold is raised, more time is spent using the ABFT mode, which lowers system availability while increasing performability. For the LEO case study, most of the performance gains from using ABFT can be obtained by using a low threshold value because much of the orbit will be under this threshold. With a 10% threshold, an RFT system would spend an approximately equal amount of time in each of the ABFT and TMR modes. Raising the adaptive threshold higher than 50%, results in limited performance gains at the expense of decreased availability. Further analysis of these thresholds in mode-switching strategies can enable optimization toward Pareto-optimal goals.

4.2.2. Highly-Elliptical Orbit Case Study. The HEO is a common type of orbit used mostly by communication satellites. From the ground, satellites traveling in an HEO can appear stationary in the sky for long periods of time. HEOs also offer visibility of the Earth's polar regions, where most geosynchronous satellites do not. The HEO used



(a) System availability of static strategies



(b) System availability of adaptive strategies

Fig. 8. LEO system availability.

for this case study is a Molniya orbit, named for the communication satellites that first used this orbit. A TLE for a Molniya-1 satellite was used to generate fault rates for the HEO case study. This orbit has a perigee of 1100km, an apogee of 39,000km, and a 63.4° inclination with a mean travel time of 12 hours. The average amount of radiation throughout the orbit is much higher than the LEO case study, and much larger amounts of radiation are encountered when the satellite passes through the Van Allen belts. The average fault rate in the Molniya-1 orbit is 62 faults per device-day. For most of the orbit, the fault rate averages 7 faults per device-day, but the large fault-rate peaks that occur near perigee increases the overall fault rate. Figure 10(a) illustrates the Molniya-1 orbit and Figure 10(b) shows the estimated number of upsets per hour that an FPGA might experience in an HEO. Figure 10(c) shows the total number of accumulated faults over time. Space systems outside of

Table V. Unavailability and Performability for LEO Case Study

	Average Unavailability	Maximum Unavailability	Average Performability	Minimum Performability
TMR	9.8×10^{-6}	3.8×10^{-5}	2.00	2.00
DWC	3.9×10^{-3}	1.1×10^{-2}	3.00	2.98
ABFT	1.6×10^{-2}	4.7×10^{-2}	4.79	4.76
2-Mode (10%)	1.9×10^{-3}	4.6×10^{-3}	3.51	2.00
2-Mode (50%)	4.4×10^{-3}	1.8×10^{-2}	4.01	2.00
3-Mode	2.7×10^{-3}	5.4×10^{-3}	3.69	2.00

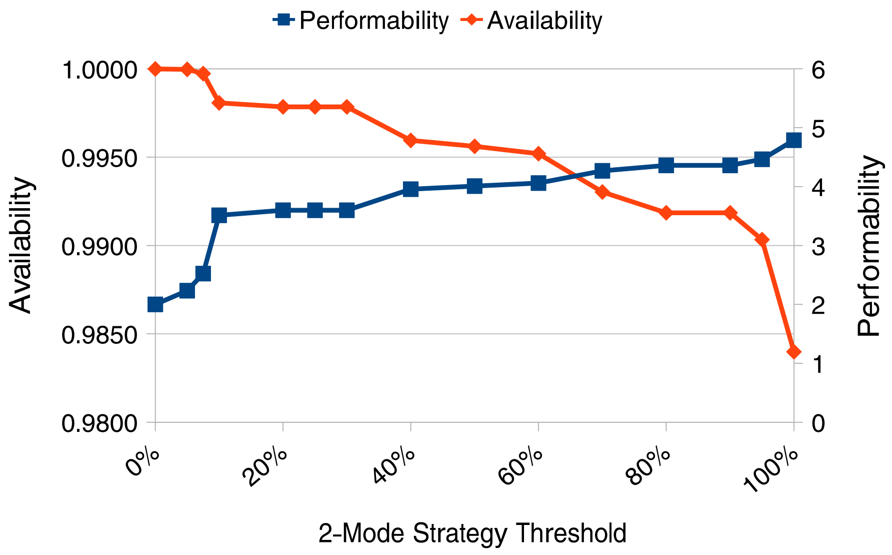
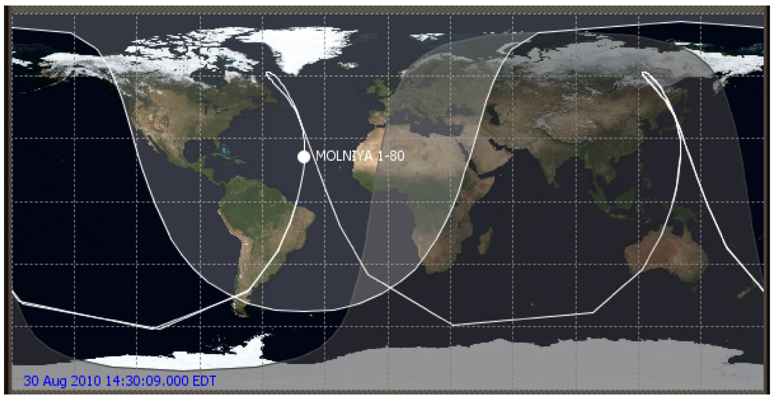


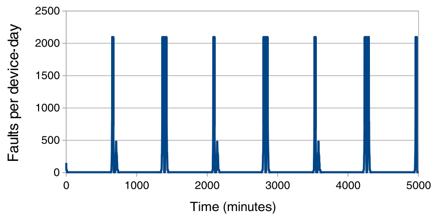
Fig. 9. Effects of adaptive threshold on availability and performability.

the Earth's magnetosphere, either in geosynchronous orbit or in interplanetary space, experience constant fault rates that vary based on the occurrence of solar flares and other space weather conditions. Solar flares can send a wave of high-energy particles into space, causing a brief period of extremely high fault rates. These fault-rate spikes, while different in origin, look similar to the fault-rate peaks that occur in this case study. The analysis used in this case study can also be used to estimate RFT system performance in the presence of different space weather conditions.

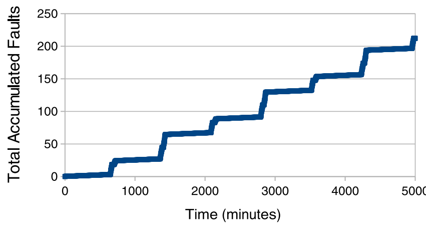
We evaluate the availability and performability of TMR, DWC, and ABFT fault-tolerance modes in an HEO. In order to maximize performability of applications in an HEO, we also examine two adaptive fault-tolerance strategies. The ABFT/TMR adaptive strategy uses the ABFT fault-tolerance mode when upset rates are in the lowest 5% of expected fault rates and the TMR fault-tolerance mode otherwise. The DWC/TMR adaptive strategy uses the same fault-rate thresholds as ABFT/TMR, but switches between the DWC and TMR modes. In each mode, the system performs scrubbing to ensure that configuration memory errors are removed from the system. For the HEO case study, the system uses a 10-second scrub cycle to account for the increased average fault rates experienced in an HEO, which is used as the system repair rate for the Markov models. Figures 11(a) and 11(b) show the availability of the HEO case study system while using three static fault-tolerance strategies and two adaptive strategies. The static TMR strategy maintains an average availability of 99.93%, but the availability drops as low as 95.1% while passing through the peak fault-rate



(a) Visualization of Molniya-1 TLE data using JSatTrak [Gano 2010]



(b) Expected fault rates over time

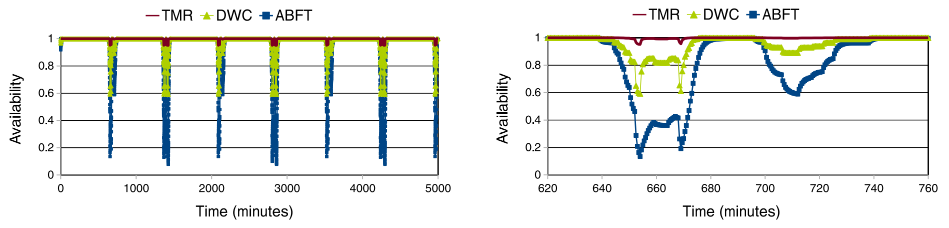


(c) Total accumulated faults over time

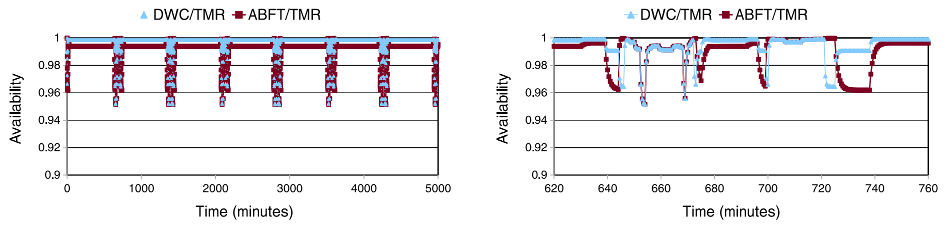
Fig. 10. HEO fault rates using the RFT fault-rate model.

periods. While using the static DWC and ABFT strategies, the availability drops significantly during the peak fault-rate periods, making these strategies unsuitable for systems that must maintain continuous operation (due to the extremely high upset rate, many systems shut down operation during peak fault-rate periods). However, outside of the peak fault-rate periods, the minimum availability for the static DWC (99.8%) and ABFT (99.3%) strategies are high enough to be tolerable by many applications. The two adaptive strategies use DWC and ABFT fault-tolerance modes during low fault-rate periods and TMR fault-tolerance mode during the peak fault-rate periods. Using the adaptive strategies, the availability never falls below the TMR strategy's minimum availability of 95.1% during peak fault-rate periods, while maintaining higher availability at other times. (Note the change of scale in Figure 11(b).) Table VI shows the unavailability and performability of the fault-tolerance strategies described in this section. The DWC/TMR adaptive strategy reduces average unavailability by 80% as compared to the static DWC strategy. The ABFT/TMR adaptive strategy reduces average unavailability by 85% as compared to the static ABFT strategy.

The system performability for the static and adaptive fault-tolerance strategies for an HEO are shown in Table VI. While the TMR strategy exhibits the lowest performability, it also has the lowest variation in performability and highest availability, allowing for predictable performance levels. The performability of the static ABFT and DWC strategies is significantly reduced in the peak fault-rate periods, but quickly returns to acceptable levels after passing through the peak fault-rate periods. The maximum unavailability for each of the adaptive strategies occurs during the peak fault-rate periods. Since the RFT system is using the TMR fault-tolerance mode during these segments, the adaptive strategies have the same maximum unavailability



(a) System availability of static strategies (zoomed image on right)



(b) System availability of adaptive strategies (zoomed image on right)

Fig. 11. HEO system availability.

Table VI. Unavailability and Performability for HEO Case Study

	Average Unavailability	Maximum Unavailability	Average Performability	Minimum Performability
TMR	7.2×10^{-4}	4.9×10^{-2}	2.00	1.95
DWC	1.4×10^{-2}	4.1×10^{-1}	2.98	2.46
ABFT	4.9×10^{-2}	9.2×10^{-1}	4.73	2.62
DWC/TMR	2.6×10^{-3}	4.9×10^{-2}	2.92	1.95
ABFT/TMR	7.2×10^{-3}	4.9×10^{-2}	4.56	1.95

as the static TMR strategy. The DWC/TMR adaptive strategy increases performability over the TMR strategy by 46%. The ABFT/TMR adaptive strategy increases average performability over the TMR strategy by 128%, or reduces performability by 4% over the static ABFT strategy, while significantly improving unavailability over the ABFT strategy.

5. CONCLUSIONS

In this work, we have presented a novel and comprehensive framework for reconfigurable fault tolerance capable of creating and modeling FPGA-based reconfigurable architectures that enable a system to self-adapt its fault-tolerance strategy in accordance with dynamically varying fault rates. The PR-based RFT hardware architecture enables several redundancy-based fault-tolerance modes (e.g., DWC, TMR) and additional fault-tolerance features (e.g., watchdog timers, ABFT, checkpointing and rollback), as well as a mechanism for dynamically switching between modes. The combination of these fault-tolerance features enables the use of COTS SRAM-based FPGAs in harsh environments. Future work will automate and optimize the creation RFT controllers for specific system configurations in order to increase developer productivity, facilitate adoption, and reduce system overhead.

In addition to the hardware architecture, we have demonstrated a fault-rate model for RFT to accurately estimate upset rates and capture time-varying radiation effects for arbitrary satellite orbits using a collection of existing, publically available tools and models. Our model provides important characterization of fault rates for space systems over the course of a mission that cannot be captured by average fault rates. The HEO case study demonstrated the large range of fault rates experienced in certain elliptical orbits, and the potential for performance improvements when using RFT.

Using the results from the fault-rate model, our Markov-model-based RFT performability model was used to demonstrate the benefits of using an adaptive fault-tolerance system architecture. The performability model was validated using FPGA fault injection on an experimental RFT system and multiple static and adaptive fault-tolerance strategies for space systems in dynamically changing environments were evaluated. The RFT performability model demonstrated that while TMR provides a lower bound on availability, less reliable, low-overhead methods can be used during low-fault-rate periods to improve performance. We evaluated two case study orbits and observed that adaptive fault-tolerance strategies are able to improve unavailability by 85% over static ABFT and performability by 128% over traditional, static TMR fault tolerance. This additional performance can lead to more capable and power-efficient FPGA-based onboard processing systems in the future.

REFERENCES

ACREE, R., ULLAH, N., KARIA, A., RAHMEH, J., AND ABRAHAM, J. 1993. An object-oriented approach for implementing algorithm-based fault tolerance. In *Proceedings of the 12th Annual International Phoenix Conference on Computers and Communications*. 210–216.

ACTEL. 2010a. Actel product page. http://www.actel.com/products/milaero/rtsxsu/default.aspx.

ACTEL. 2010b. Actel product page. http://www.actel.com/products/milaero/rtpa3/default.aspx.

ALAM, M., SONG, M., HESTER, S., AND SELIGA, T. 2006. Reliability analysis of phased-mission systems: A practical approach. In *Proceedings of the Annual Reliability and Maintainability Symposium (RAMS)*. 551–558.

ALNAJIAR, D., KO, Y., IMAGAWA, T., KONOURA, H., HIROMOTO, M., MITSUYAMA, Y., HASHIMOTO, M., OCHI, H., AND ONOYE, T. 2009. Coarse-grained dynamically reconfigurable architecture with flexible reliability. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*. 186–192.

ALTERA. 2010. Stratix V FPGAs: Ultimate flexibility through partial and dynamic reconfiguration. http://www.altera.com/products/devices/stratix-fpgas/stratix-v/overview/partial-reconfiguration/stxv-part-reconfig.html.

BOWEN, N. AND PRADHAM, D. 1993. Processor- and memory-based checkpoint and rollback recovery. *Comput. 26*, 2, 22–31.

CARMICHAEL, C., FULLER, E., BLAIN, P., AND CAFFREY, M. 1999. SEU mitigation techniques for Virtex FPGAs in space applications. In *Proceedings of the 2nd Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference*.

CIARDO, G., MARIE, R., SERICOLA, B., AND TRIVEDI, K. 1990. Performability analysis using semi-Markov, reward processes. *IEEE Trans. Comput. 39*, 10, 1251–1264.

CIESLEWSKI, G., GEORGE, A., AND JACOBS, A. 2010. Acceleration of FPGA fault injection through multibit testing. In *Proceedings of the Engineering of Reconfigurable Systems and Algorithms Conference*.

DAWOOD, A., VISSER, S., AND WILLIAMS, J. 2002. Reconfigurable FPGAs for real time image processing in space. In *Proceedings of the 14th International Conference on Digital Signal Processing (DSP)*. Vol. 2, 845–848.

DOBIAS, R., KUBALIK, P., AND KUBATOVA, H. 2005. Dependability computations for fault-tolerant system based on FPGA. In *Proceedings of the 12th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. 1–4.

FLATLEY, T. 2010. Advanced hybrid on-board science data processor - SpaceCube 2.0. In *Proceedings of the Earth Science Technology Forum*.

GANO, S. 2010. JSatTrak. http://www.gano.name/shawn/JSatTrak/index.html.

GARVIE, M. AND THOMPSON, A. 2004. Scrubbing away transients and jiggling around the permanent: Long survival of FPGA systems through evolutionary self-repair. In *Proceedings of the 10th IEEE International Online Testing Symposium (IOLTS)*. 155–160.

GUPTA, A., NOOSHABADI, S., TAUBMAN, D., AND DYER, M. 2006. Realizing low-cost high-throughput general-purpose block encoder for JPEG2000. *IEEE Trans. Circuits Syst. Video Technol. 16*, 7, 843–858.

HOOTS, F. R. AND ROEHRICH, R. L. 1980. SPACETRACK REPORT NO. 3-Models for propagation of NORAD element sets. http://celestrak.com/NORAD/documentation/spacetrk.pdf.

HSUEH, M. AND CHANG, C.-I. 2008. Field programmable gate arrays (FPGA) for pixel purity index using blocks of skewers for endmember extraction in hyperspectral imagery. *Int. J. High Perform. Comput. Appl. 22*, 408–423.

HUANG, K.-H. AND ABRAHAM, J. 1984. Algorithm-based fault tolerance for matrix operations. *IEEE Trans. Comput. 33*, 6, 518–528.

JOHNSON, J., HOWES, W., WIRTHLIN, M., MCMURTREY, D., CAFFREY, M., GRAHAM, P., AND MORGAN, K. 2008. Using duplication with compare for on-line error detection in FPGA-based designs. In *Proceedings of the IEEE Aerospace Conference*.

KARNIK, T. AND HAZUCHA, P. 2004. Characterization of soft errors caused by single event upsets in CMOS processes. *IEEE Trans. Depend. Secure Comput. 1*, 2, 128–143.

KIM, K. AND PARK, K. 1994. Phased-mission system reliability under Markov environment. *IEEE Trans. Rel. 43*, 2, 301–309.

KYRIAKOULAKOS, K. AND PNEVMATIKATOS, D. 2009. A novel SRAM-based FPGA architecture for efficient TMR fault tolerance support. In *Proceedings of the International Conference on Field Programmable Logic and Applications(FPL)*. 193–198.

LAPRIE, J.-C., ARLAT, J., BEOUNES, C., AND KANOUN, K. 1990. Definition and analysis of hardware- and software-fault-tolerant architectures. *IEEE Trans. Comput. 23*, 7, 39–51.

LE, C., CHAN, S., CHENG, F., FANG, W., FISCHMAN, M., HENSLEY, S., JOHNSON, R., JOURDAN, M., MARINA, M., PARHAM, B., ROGEZ, F., ROSEN, P., SHAH, B., AND TAFT, S. 2004. Onboard FPGA-based SAR processing for future spaceborne systems. In *Proceedings of the IEEE Radar Conference.* 15–20.

MACMILLAN, S. AND MAUS, S. 2010. IGRF10 Model Coefficients for 1945-2010. http://modelweb.gsfc.nasa.gov/magnetos/igrf.html.

MAUS, S., MACMILLAN, S., CHERNOVA, T., CHOI, S., DATER, D., GOLOVKOV, V., LESUR, V., LOWES, F., LHR, H., MAI, W., MCLEAN, S., OLSEN, N., ROTHER, M., SABAKA, T., THOMSON, A., AND ZVEREVA, T. 2005. The 10th generation international geomagnetic reference field. *Phys. Earth Planetary Interiors 151*, 3–4, 320–322.

MEYER, J. 1982. Closed-form solutions of performability. *IEEE Trans. Comput. 31*, 7, 648–657.

MORGAN, K., MCMURTREY, D., PRATT, B., AND WIRTHLIN, M. 2007. A comparison of TMR with alternative fault-tolerant design techniques for FPGAs. *IEEE Trans. Nucl. Sci. 54*, 6, 2065–2072.

NAEIMI, H. AND DEHON, A. 2008. Fault-tolerant sub-lithographic design with rollback recovery. *Nanotechnol. 19*, 11, 115708.

PRATT, B., CAFFREY, M., GRAHAM, P., MORGAN, K., AND WIRTHLIN, M. 2006. Improving FPGA design robustness with partial TMR. In *Proceedings of the 44th Annual IEEE International Reliability Physics Symposium.* 226–232.

PRATT, B., WIRTHLIN, M., CAFFREY, M., GRAHAM, P., MORGAN, K., QUINN, H., AND SHELLEY, S. 2007. Improving FPGA reliability in harsh environments using triple modular redundancy with more frequent voting. In *Proceedings of the Prentice Hall. Military and Aerospace FPGA Applications Conference*.

RAO, T. AND FUJIWARA, E. 1989. *Error-Control Coding for Computer Systems*.

RATTER, D. 2004. FPGAs on Mars. *Xcell J.*, 8–11.

SAHNER, R. A. AND TRIVEDI, K. S. 1987. Reliability modeling using SHARPE. *IEEE Trans. Rel. 36*, 2, 186–193.

SHIM, B., SRIDHARA, S., AND SHANBHAG, N. 2004. Reliable low-power digital signal processing via reduced precision redundancy. *IEEE Trans. VLSI Syst. 12*, 5, 497–510.

SILVA, J., PRATA, P., RELA, M., AND MADEIRA, H. 1998. Practical issues in the use of ABFT and a new failure model. In *Proceedings of the 28th Annual International Symposium on Fault-Tolerant Computing*. 26–35.

SWIFT, G., ALLEN, G., TSENG, C. W., CARMICHAEL, C., MILLER, G., AND GEORGE, J. 2008. Static upset characteristics of the 90nm Virtex-4QV FPGAs. In *Proceedings of the IEEE Radiation Effects Data Workshop*. 98–105.

TROXEL, I., FEHRINGER, M., AND CHENOWETH, M. 2008. Achieving multipurpose space imaging with the ARTEMIS reconfigurable payload processor. In *Proceedings of the IEEE Aerospace Conference*. 1–8.

TYLKA, A., ADAMS, J.H., J., BOBERG, P., BROWNSTEIN, B., DIETRICH, W., FLUECKIGER, E., PETERSEN, E., SHEA, M., SMART, D., AND SMITH, E. 1997. CREME96: A revision of the cosmic ray effects on micro-electronics code. *IEEE Trans. Nucl. Sci. 44*, 6, 2150–2160.

WANG, J. 2003. Radiation effects in FPGAs. In *Proceedings of the 9th Workshop on Electronics for LHC Experiments*.

WANG, S.-J. AND JHA, N. 1994. Algorithm-based fault tolerance for FFT networks. *IEEE Trans. Comput. 43*, 7, 849–854.

WILLIAMS, J., MASSIE, C., GEORGE, A. D., RICHARDSON, J., GOSRANI, K., AND LAM, H. 2010. Characterization of fixed and reconfigurable multi-core devices for application acceleration. *ACM Trans. Reconfigur. Technol. Syst. 3*, 1–29.

Xilinx 2004. *XTMR Tool User Guide*. Xilinx. Xilinx User Guide UG156.

Xilinx 2010a. Partial Reconfiguration User Guide. Xilinx. Xilinx User Guide UG702.

Xilinx 2010b. SEU Strategies for Virtex-5 Devices. Xilinx. Xilinx Application Note XAPP864.

Xilinx 2010c. Space-Grade Virtex-4QV Family Overview. Xilinx. Xilinx Product Specification DS653.

Received May 2011; revised September 2011, February 2012; accepted June 2012