Hindawi
International Journal of Reconfigurable Computing
Volume 2020, Article ID 2808710, 19 pages
https://doi.org/10.1155/2020/2808710

Research Article

Dynamic Reliability Management for FPGA-Based Systems

Jahanzeb Anwer, Sebastian Meisner, and Marco Platzner

Department of Computer Science, Paderborn University, 33098 Paderborn, Germany

Correspondence should be addressed to Marco Platzner; platzner@upb.de

Received 13 November 2019; Revised 18 February 2020; Accepted 19 May 2020; Published 13 June 2020

Academic Editor: Martin Margala

Radiation tolerance in FPGAs is an important field of research particularly for reliable computation in electronics used in aerospace and satellite missions. The motivation behind this research is the degradation of reliability in FPGA hardware due to single-event effects caused by radiation particles. Redundancy is a commonly used technique to enhance the fault-tolerance capability of radiation-sensitive applications. However, redundancy comes with an overhead in terms of excessive area consumption, latency, and power dissipation. Moreover, the redundant circuit implementations vary in structure and resource usage with the redundancy insertion algorithms as well as number of used redundant stages. The radiation environment varies during the operation time span of the mission depending on the orbit and space weather conditions. Therefore, the overheads due to redundancy should also be optimized at run-time with respect to the current radiation level. In this paper, we propose a technique called Dynamic Reliability Management (DRM) that utilizes the radiation data, interprets it, selects a suitable redundancy level, and performs the run-time reconfiguration, thus varying the reliability levels of the target computation modules. DRM is composed of two parts. The design-time tool flow of DRM generates a library of various redundant implementations of the circuit with different magnitudes of performance factors. The run-time tool flow, while utilizing the radiation/error-rate data, selects a required redundancy level and reconfigures the computation module with the corresponding redundant implementation. Both parts of DRM have been verified by experimentation on various benchmarks. The most significant finding we have from this experimentation is that the performance can be scaled multiple times by using partial reconfiguration feature of DRM, e.g., 7.7 and 3.7 times better performance results obtained for our data sorter and matrix multiplier case studies compared with static reliability management techniques. Therefore, DRM allows for maintaining a suitable trade-off between computation reliability and performance overhead during run-time of an application.

1. Introduction

The advancement of semiconductor technology to nanometer dimensions has made the design of digital circuits challenging. Future shrinking of device parameters is inevitable due to the increasing need for high computing power and more computational blocks on a single system-on-chip. Besides the conventional performance, area, and power constraints, today's circuits have to conform to reliability requirements. The reliability of the digital circuits however degrades due to probabilistic nature of errors appearing in nanodevices. These errors, being permanent or transient in nature, are mainly caused by process variations, thermal fluctuations, quantum effects, power supply noise, and capacitive/inductive coupling, as few examples [1–4]. These errors are treated for fault tolerance at device, logic, or network layers depending on the feasibility of mitigation at each of these layers. However, when the nanodevice-based circuits are used in high-radiation environments (consisting alpha particles and cosmic rays), an additional source of error emerges which is known as radiation-induced errors. Although the reliability studies for electronics include well-defined mitigation strategies for different sources of errors, they propose redundancy as the most efficient way of countering radiation-induced errors.

The future of space computing is largely dominated by Field Programmable Gate Arrays (FPGAs) due to their capability of run-time modification of functional implementation on hardware. The reconfiguration feature of FPGAs allows them to perform various tasks during different phases of a mission [5]. Moreover, increase in on-board processing requirements on space missions for various image-processing applications goes well with the highly parallel architecture of FPGAs [6]. Most importantly, the space, weight, and power for a satellite payload design can be minimized with the usage of FPGAs due to their ability to perform multiple tasks without having a dedicated hardware for each of these tasks.

The advantages of using FPGAs in space computing brought the attention of researchers and FPGA companies to ensure the fault-tolerant operation of FPGAs in high-radiation space environments. When FPGAs are exposed to high solar or cosmic radiation (involving high-energy electrons, alpha particles, and heavy ions), errors in the form of logic reversals appear in the digital circuit elements which could be as disastrous as causing a system-level failure or as moderate as internally masked errors. For long-term space missions, the accumulation of ionizing dose, measured as Total Ionizing Dose (TID), is an important design parameter which indicates how long the FPGA can withstand the radiation before its transistors begin to degrade [7]. For rather short-term space missions, Single-Event Effects (SEEs) cause temporary errors to appear in the circuit whose mitigation strategies vary from internal masking (using redundancy) to system-reset requirement. Generally, SEEs, which are measureable changes in the state of a microelectronic device [8], are classified into four domains. Single-Event Transient (SET) is a voltage spike causing a glitch in a combinational element; Single-Event Upset (SEU) is a soft error caused by the radiation particle in memory contents, particularly SRAM cells; Single-Event Latchup (SEL) is the high-current state in a device caused by the passage of a single energetic particle through sensitive regions of the device structure or Single-Event Functional Interrupt (SEFI) that cause the component to reset, lockup, or otherwise malfunction in a detectable way. SETs are typically short-term and ineffective unless they are latched into a sequential element, thus behaving as an SEU. SELs can be corrected by power cycling, while SEFIs require resetting the component [7–9]. The remaining category, i.e., SEU, is the major concern for the reliable FPGA operation due to its higher appearance rate compared with other SEEs as well as its accumulation effects. SRAM-based FPGAs are highly susceptible to SEUs appearing in their configuration and application memory elements.

The alternative to SRAM-based FPGAs are radiation-hardened FPGAs which include the foremost Actel (currently known as Microsemi) RTAX antifuse FPGAs [10]. These devices are one-time programmable, and the development of permanent interconnections after configuration makes them immune to SEUs. However, being non-reprogrammable, they lose their charm for utilization in multiple design modification scenarios. The second popular category consists of flash-based FPGAs [11], which offer full reconfiguration though lack partial reconfiguration [6]. Moreover, these devices have typically lower TID rating than SRAM or antifuse FPGAs [12]. However, an additional category of radiation-tolerant FPGAs consists of space-grade versions which utilize the performance of SRAM FPGAs while having built-in radiation-tolerance features, e.g., Xilinx Virtex-5QV [13].

In contrast to inherent radiation-tolerant capabilities of FPGAs, fault-tolerant computation approaches in hardware and software are utilized as well. Redundancy [14, 15] and scrubbing [16, 17] are two most popular techniques for tolerating SEUs and avoiding their accumulation. A classical and widely used form of redundancy is triple modular redundancy (TMR). In principle, TMR instantiates three identical copies of a circuit and places a voter module at the end to take a majority decision for each output. Hence, TMR does not depend on error detection but mitigates the error by passage through the voter. In contrast, scrubbing involves continuously configuring configuration memory to prevent accumulation of errors. The combination of redundancy and scrubbing is considered a widespread optimal fault-tolerant solution in hardware. Additional approaches in the literature include duplication with comparison (DWC) [18], error checking and correcting codes (ECAC) [19], and algorithm-based fault tolerance (ABFT) [20]. However, in this paper, we focus solely on redundancy and its different variations in hardware. Hardware redundancy techniques for FPGAs are more involved than basic TMR with respect to partitioning a circuit into submodules, deciding on how many voters to insert, and where to place the voters in the FPGA design. Tools for automating redundancy insertion in FPGA designs are available, including the TMR tool of Xilinx [21], Precision Hi-Rel software [22], and the BYU-LANL TMR tool [23]. Fault-tolerance mechanisms, particularly modular redundancy, come with an overhead in terms of excessive area consumption as well as latency and power dissipation. Therefore, while providing fault tolerance, the design of a mission critical system also has to limit these overheads to given constraints.

The radiation environment during the operation time of the satellite is variant, especially the radiation particle strike rate increases enormously above Earth's magnetosphere [6, 24]. The fault-tolerance techniques, particularly redundancy, have a constant overhead in performance factors of area, latency, and power dissipation. However, in general, higher stages of redundancy provide more reliability at the cost of increasing overheads in performance factors. Therefore, the realization of reliability-performance trade-off is mandatory before designing a redundant system. In order to avoid a constant degradation in system performance due to a fixed overhead in performance factors, the system should be self-adaptive in a way to optimize the trade-off between reliability and performance factors, at runtime, based on the radiation strength of the environment. We implement this concept named as "Dynamic Reliability Management (DRM)." DRM, based on the partial reconfiguration of FPGAs, is beneficial as it allows for the optimization of the performance overheads and, thus, can save cost and power or free hardware resources for other tasks when feasible.

The contribution of this paper lies around providing a complete approach for using SRAM-based FPGAs in space missions whereby using real-time radiation scenarios for our

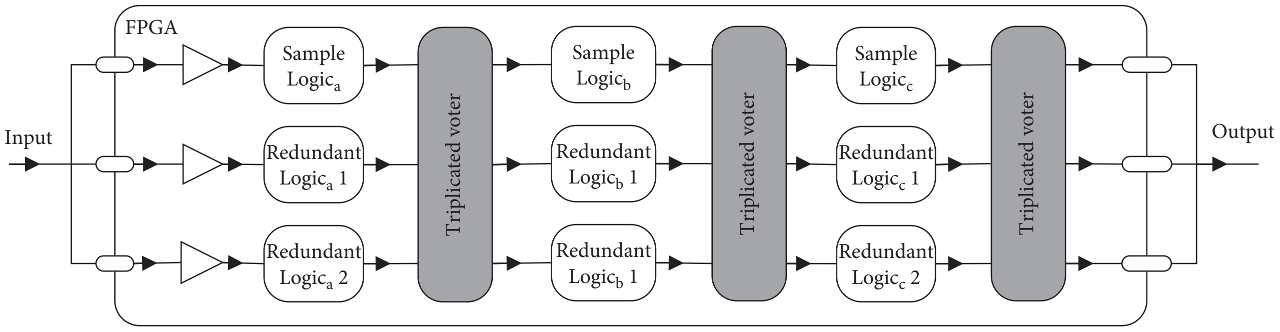

Figure 1: TMR implementation in FPGA.

analysis and experimentation. Most importantly, we focus on visualizing the trade-off of reliability with the three main performance parameters, i.e., latency, area, and power consumption. This is in contrast to the research studies which work on the same research line but fail to show the complete picture of reliability versus three performance factors and how to prefer one redundant structure over another based on the performance constraints. This paper provides such solution where Pareto optimization can be used to filter out the suitable redundant structure based on one or more performance factors. Our developed tool flows are unique and more extensive than previous research works which are comprehensively verified in this paper as well. Last but not the least, we also provide the possible extensions to our experimentation platform, which can be used as future research directions in this vast and emerging research area. We have tried our best to be generic as well as having a bigger scope for our research so that the maximum research community in space computing can benefit from it.

This paper is organized as follows. In the next section, we provide the foundation for understanding the adaptive reliability management concept by describing implementation strategies of redundancy in FPGAs, varying radiation environments, corresponding decision mechanisms, and commonly used reliability theories. Afterwards, we discuss and analyze major research works in self-adaptive reliability management and contrast them to our approach of Dynamic Reliability Management. Section 3 describes the two portions of our DRM approach as design and run-time parts, while verification of each of these parts has been provided with detailed experimentation in Section 4. Section 5 concludes the paper.

2. Background and Related Work

In this section, we describe the basic implementation strategy of TMR in FPGAs followed by an insight into different forms of TMR and its cascaded version. To understand the varying radiation environment of space, we give real radiation scenarios as examples. The reliability computation of FPGA-based circuits can be done by conventional and probabilistic theories, which will be briefly described as well. Finally, we summarize and analyze the major research works similar to our approach of Dynamic Reliability Management.

2.1. Triple Modular Redundancy in FPGAs. The concept of triple modular redundancy (TMR) is straightforward as it triplicates a logic design and takes the final output of the design from a voter placed at the outputs of the redundant modules [14]. The function of the voter is to sample three logic outputs and forward the majority result. The limitation of this architecture is the single point of failure, i.e., an error occurring in the voter renders the TMR technique useless. To avoid the single point of failure, we can create a more reliable architecture involving triplicated voters in addition to triplicated logic modules. This architecture runs the three branches in parallel unless they are to be interfaced with the outside world of the FPGA, where they can be converged using reducing voters or could be interfaced in the form of three outputs.

The TMR technique can be used in an FPGA by simply triplicating the inputs, outputs, and logic modules, inserting buffers, and connecting the outputs of logic modules to the triplicated voter. This straightforward implementation is not suitable due to some practical considerations. Firstly, TMR is able to counter one error among the three redundant branches, and a larger length of each branch increases its probability of being erroneous more than once. To deal with this issue, there is a need to break the logic of the branch at regular intervals and place the triplicated voters in intermediate stages of the circuit as shown in Figure 1. Thus, an error occurring in one partition of the logic will not be forwarded to the next partition due to the error-mitigation effect of the triplicated voter. However, the minimum size of the logic partition, or granularity level, could be limited to a single component on an FPGA, e.g., a lookup table and multiplexer. In addition, there are certain locations on an FPGA called illegal-cut locations which should not be triplicated due to the FPGA architecture, e.g., dedicated route connections in a slice [25]. Moreover, voters should not be placed on high-speed carry chains in order to not deteriorate the timing performance of the design. Most importantly, voters should always be added in the feedback paths to avoid data corruption at the outputs of sequential elements being forwarded into the feedback paths [14, 25]. These voters are commonly denoted as synchronization voters.

The process of automatic TMR insertion into a circuit design can be done, for example, via the BYU-LANL TMR (BANL) tool. The BANL tool is able to triplicate the design,

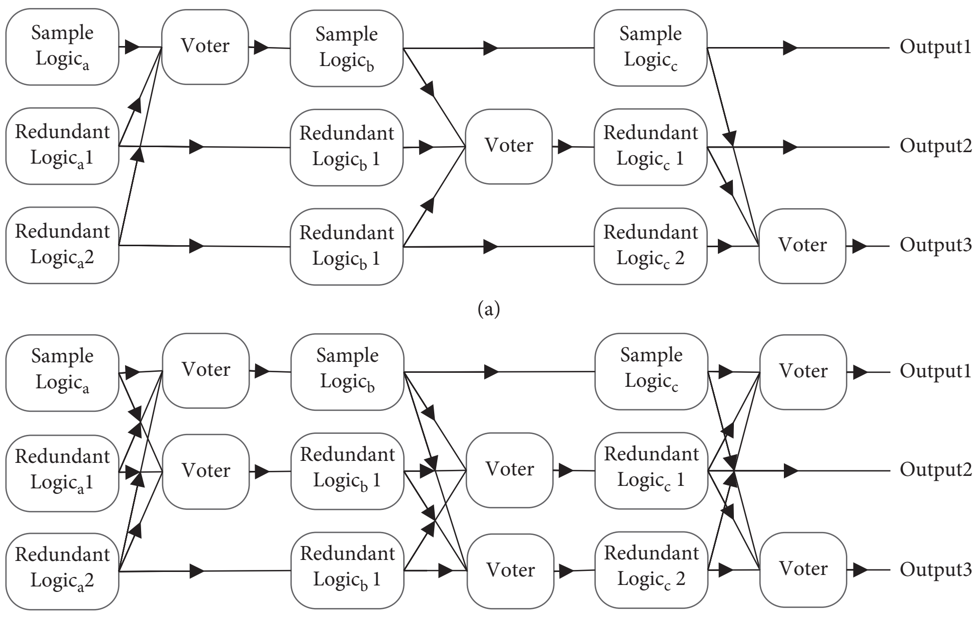

Figure 2: TMR with (a) one alternate voter and (b) two alternate voters.

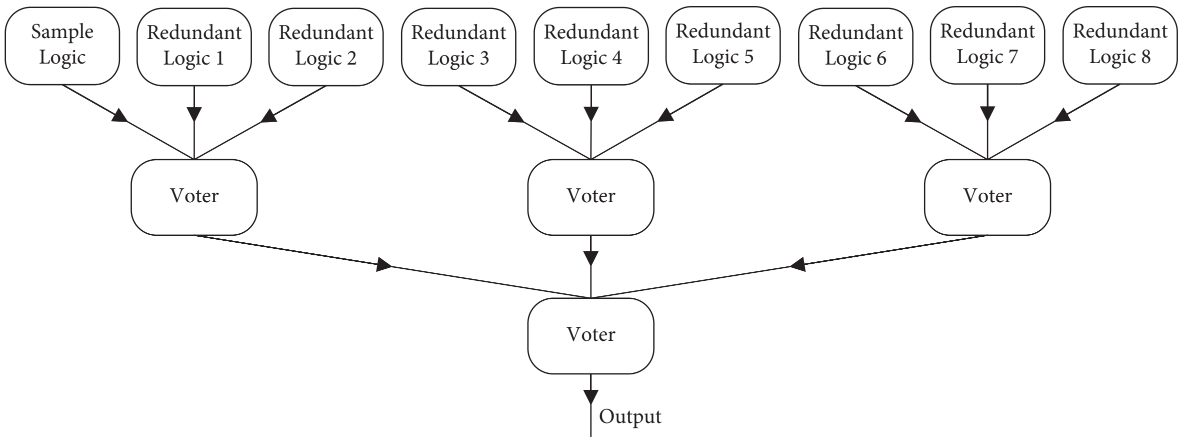

Figure 3: Cascaded TMR-level 1.

insert voters, and use built-in algorithms to take care of the constraints explained above. It is up to the discretion of the circuit designer to request the desired *redundancy configuration*, e.g., TMR with only single voters or with a mixture of single and triplicated voters. Moreover, there is a choice of eight algorithms that decide the placement of voters in the triplicated design. These algorithms are termed *voter-insertion* algorithms. Depending on the type of algorithm, the sets of nets are determined where the voters should be inserted, e.g., using feedback edge set (FES) algorithms or decomposition of strongly connected components (SCCs) in the circuit graph [25]. The details of these algorithms can be found in [5]. The algorithms used in the BANL tool and in our experimentation are abbreviated as follows:

(1) CC: Connectivity cutset

(2) AFC: After flipflop cutset

(3) BFC: Before flipflop cutset

(4) BD: Basic decomposition

(5) HFC: Highest fanout cutset

(6) HFFC: Highest flipflop fanout cutset

(7) HFFIC: Highest flipflop fanin input cutset

(8) HFFOC: Highest flipflop fanin output cutset

2.2. Variation in Voting Structures of TMR and Cascaded TMR. Besides the conventional single and triplicated voter configurations, there are configurations proposed in [26] with single/double voters in the alternate stages as shown in Figure 2. Using Monte Carlo simulations, the authors proved that these alternate configurations are slightly less reliable than triplicated voter configuration though they save

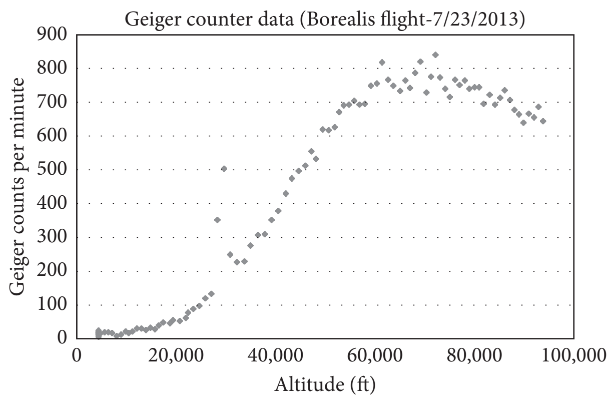

Figure 4: Borealis radiation strike-rate profile [28].

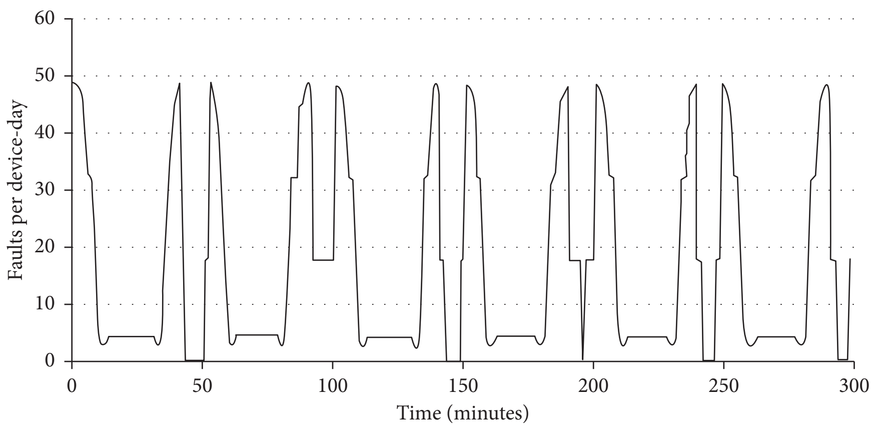

Figure 5: Expected fault rate of LEO (taken from [6]).

the overhead of increased number of voters. Hence, in situations where the radiation environment is not very strong and area consumption is an important issue, the alternate configurations are highly useful. In addition, there is a concept of cascaded TMR which results in more reliable configurations than TMR, though consuming more area [27]. For our analysis, we use the level-1 CTMR shown in Figure 3. It can be noted that the CTMR configuration has a single point of failure as it is the extension of TMR with a single voter. While CTMR can also be improved by using the triplicated voter strategy, we assume that CTMR is always superior to TMR for the time being. Overall and based on the results presented in [14, 26, 27], we rate the reliability of the discussed configurations in the following ascending order:

(1) SV: single voter [14]

(2) OAV: one alternate voter [26]

(3) TAV: two alternate voters [26]

(4) TV: triplicated voter [14]

(5) CTMR: cascaded TMR-Level 1 [27]

Originally, the BANL tool supported only SV and TV configurations while we extended the tool to support the rest of the three configurations, as will be explained in Section 3.1.2. It has to be noted that all configurations have to resort to single voters for illegal-cut locations. In particular, also the configuration TV, which is the default configuration of the BANL tool, combines triplicated and single voter structures.

2.3. Variation in Radiation Patterns.

The main motivation for dynamic reliability management, implemented in a self-adaptive reconfigurable system, is for space missions where radiation strength is high and fluctuating. In this section, based on the literature, we provide four distinct radiation scenarios to strengthen the need for varying reliability levels. The first scenario is referenced with height, the second and third with respect to time, and the fourth with solar conditions. The radiation strikes per minute and the soft error rate are considered proportional measures of radiation strength; however, not all the radiation particles might appear as errors in the hardware due to the their low ionizing energy or masking effects.

2.3.1. Borealis Flight.

In a hot air balloon testing conducted at the University of Montana [28], i.e., Borealis Flight, a Geiger sensor tube was sent to a high altitude of around 100,000 feet. The corresponding Geiger counter logged the radiation strikes per minute capturing most of the low- and high-energy particles. Figure 4 shows the variation of the recorded radiation strike rate with altitude. By analyzing

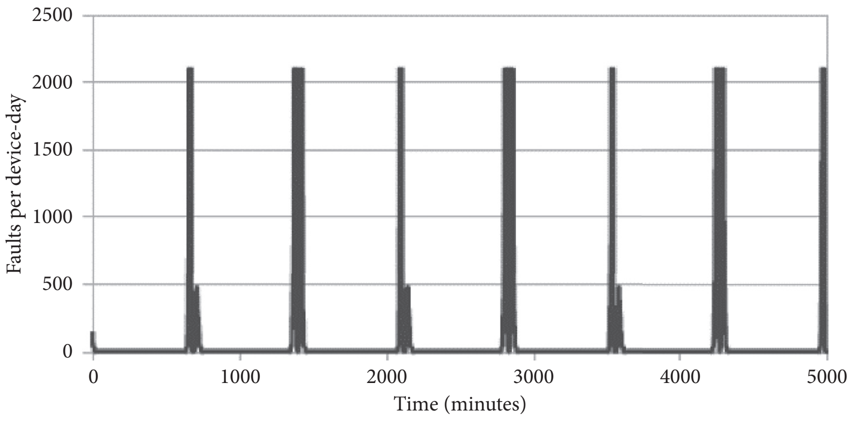

Figure 6: Expected fault rate of HEO (taken from [6]).

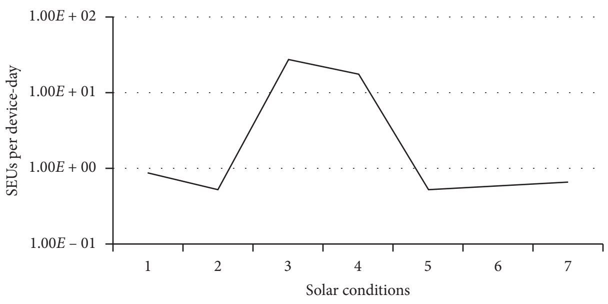

Figure 7: Expected fault rate of LEO orbit under different solar conditions (taken from [29]).

such a radiation profile, we can safely assume that the more redundancy levels we have at hand, the better a system would be able to exploit the trade-off between reliability and performance. Moreover, it is also worth noting that the sensor captured two radiation data points at approximately 30,000 feet, which are uncorrelated to the trend. This emphasizes the need for a system that is able to adapt the reliability level in order to respond to unexpected deviations from an observed trend.

2.3.2. Low-Earth-Orbit Case Study. In the research work presented in [6], a fault-rate model is presented and used to simulate the expected error rate for a path in low earth orbit (LEO). The error rate is presented in the unit of faults per device-day, which is useful to represent the cumulative error rate when a number of devices/FPGAs are monitored in parallel. The orbital track of LEO case study has a mean travel time of 98 minutes due to which we see the repetition of fault pattern. We do not experience huge fault rate due to an altitude of only 700 km which is below the Van Allen radiation belts and completely within Earth's magnetosphere. It can be observed in Figure 5 that there are approximately three discrete radiation/fault-rate levels; hence, three reconfigurable reliability levels would be sufficient.

2.3.3. Highly Elliptical Orbit Case Study. Referring to the research work in [6], an additional case study for expected fault rate was conducted for an elliptical orbit of perigee 1100 km and apogee 39,000 km with a mean travel time of 12 hours. The path is called Molniya orbit and used for the communication satellites in particular. As can be seen in the radiation plot of Figure 6, the fault rate gets high at the end of the time period when it passes the Van Allen radiation belts. Most of the orbit duration has negligible fault rate as compared with a short duration of excessive fault rate at the end of the time period. For such a fault-rate profile, we can expect two reliability levels to be sufficient where the higher reliability level is required only for a very short duration.

2.3.4. Anticipated Error Rate for Different Solar Conditions. In the research work conducted in [29], an expected error rate for another LEO was anticipated for seven different solar conditions, as shown in Figure 7. Although the names of the solar conditions are replaced with numeric numbers in the figure for simplicity, their details can be found in [29]. The graph shows the minimum fault rate of 0.5 SEUs per device-day in solar condition No. 2 and maximum fault rate of 26 SEUs per device-day for solar condition No. 3. For simplicity, the graph can be divided into two regions based on the SEU rate. The first region, i.e., with solar conditions 1, 2, 5, 6, and 7 can be used with the low redundancy, while the other region (solar conditions 3 and 4) can be used with the high-redundancy circuit structures.

2.4. Reliability Evaluation of FPGA-Based Redundant Structures. The traditional way of modeling reliability of electronic systems is to describe the time until the system fails by a random variable. Using an exponential distribution for the time between failures, assuming independent failures, and a constant fault rate λ, we can determine a system's time-dependent reliability $R(t)$ as defined by the following equation:

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

Assuming a constant fault rate is reasonable if we exclude the burn-in and wear-out phases of systems. Thus, based on an estimated or measured fault rate, $R(t)$ can be calculated, and it expresses the probability that a system survives, i.e., is without fault, from its start at time 0 until time t. The reciprocal of λ is denoted as mean time to failure (MTTF) or mean time between failures (MTBF) in case of systems that can be repaired. MTTF and MTBF are widely used as reliability metrics. Though many research works follow this generic reliability equation, they differ in how they determine the fault rate λ. In the following two subsections, we describe major approaches that focus on determining fault rates of FPGA designs with redundant structures followed by a review on probabilistic computational reliability models.

2.4.1. CREME96-Based Reliability Computation. Vaderbilt University's online tool CREME96 [30] is specifically used to compute the fault rate of devices used in aerospace missions. The tool takes as input the device SEU cross section, the solar condition, and specific parameters of space orbit and computes the SEU rate due to heavy ions and protons [31]. The basic fault-rate model of this tool, whether used in its original form or modified for a specific system design, can be found in research works of [6, 24, 32], where FPGAs are used for space computing. The impact of redundancy can be modelled by the Markov fault model [6] or the classical TMR equation [32].

2.4.2. Fault Injection and Testing-Based Reliability Models. A typical way of experimentally checking FPGA reliability is via fault injection into the FPGA bitstream [33]. Since the SEUs can be modelled as bit flips in storage elements, we can revert the bit at the register output and see its effect on the behavior or functionality of the circuit. The bitstream formats of FPGAs are proprietary, which turns the identification of the exact bit locations in the bitstream, that need to be flipped to execute a specific fault in the desired circuit element, into a tedious re-engineering job. Therefore, random fault injection is normally exercised using Monte Carlo techniques [26]. Since an FPGA bitstream contains only a small fraction of critical bits, i.e., the bits that cause an error when affected by an SEU, most of the SEUs do not affect the circuit's functionality. Together with the restricted time for simulation, this leads to a limited coverage.

The FPGA manufacturers, e.g., Xilinx, have their custom testing methods and models that provide reliability measures. They typically report reliability in the FIT (failures-in-time) metric, which defines the number of errors in one billion operation hours of a device [34]. The failure rates of Xilinx FPGAs, for example, can be found for various process technology nodes in [34, 35]. In addition, radiation effects are characterized by the SEU device rates measured by accelerated beam testing and real-time atmospheric testing, particularly for configuration memory and BRAM. Moreover, various other failure rate measurements are conducted by Xilinx and reported in their device reliability reports, though not only radiation-induced errors but also high temperature, humidity, and stress tests.

2.4.3. Probabilistic Computational Reliability Models. Another category of reliability computation is via probabilistic computational models that take as input error probabilities of individual components and compute the output error probability ε_{out} of the overall system by propagating error probabilities from inputs to outputs. A number of publications relate the output error probability to the system's reliability according to $R = (1 - \varepsilon_{\text{out}})$, the most popular works being [36–38]. It is important to note that this notion of reliability and the underlying notion of a system's output error probability are timeless. An apparent interpretation of this reliability measure is that we take a snapshot of the system and make a statement about the probability that it is working error-free. That way, the component error probabilities we use in this model indicate the cumulative effect of all sources of errors. Related work does not detail how to determine the exact component error probabilities and uses arbitrarily set values. The probabilistic computational models are in contrast to the time-dependent reliability measure of equation (1), and the fault-rate method in CREME96 that simulates real device and orbit features and targets mainly radiation-induced errors.

2.5. Comparative Works in Adaptive Reliability Management. In this section, we briefly discuss and analyze research works on self-adaptive reconfiguration mechanism for reliability, with focus on radiation-induced errors.

The authors in [39] present a very detailed approach for an adaptive system denoted as reconfiguration for reliability (R4R). The R4R framework proposes an initial circuit analysis to estimate the cost of different versions of a circuit implementation including the default design and more reliable, i.e., hardened solutions. In the next step, design space exploration is performed including floorplanning to optimize the placement of hardened solutions on the FPGA. The hardened solutions are based on TMR at the system and component levels including the design obtained by the commercial Xilinx TMR tool. The solutions are then Pareto-optimized based on area utilization and reconfiguration time. Although this research work describes the initial analysis and design space exploration steps in detail, it stays abstract due to following reasons. R4R is seen as a broad framework for having an intelligent system exploring different redundant hardware designs though no implementation of such a system is proposed with run-time support. Moreover, the cost parameters do not consider other important factors, e.g., maximum clock frequency of the circuit and power dissipation of different hardened designs. In addition, the authors do not discuss concepts for decision mechanisms for the online reconfiguration.

The concept of reconfigurable fault tolerance (RFT) is provided in [6]. This work targets space applications and simulates processing components' availability based on DWC, TMR, ABFT, and high-performance (HP) fault-tolerance mechanisms. The switching among different fault-tolerance approaches is supported by architecture-level changes through varying partially reconfigurable fault-tolerance regions. The authors use two case studies about low and high earth orbits to represent the radiation strength variation in atmosphere, which is useful for taking appropriate reconfiguration decisions. It has been successfully concluded from the experiments that different fault-tolerance approaches can be useful in different radiation environments. Moreover, it has been clarified how high scrubbing rates can improve system performance. However, the authors neither compare the overheads due to area, latency, and power for different fault-tolerance approaches nor do they consider a design space exploration of different redundant designs. In addition, RFT focuses on the classical, system-level TMR approach, while more efficient component-based redundancy approaches are available.

A custom radiation sensor measuring radiation strikes has been built and proposed for adaptive fault tolerance in [24]. This research employs a radiation sensor whose measured radiation strikes are considered proportionate to the errors appearing in the system. This system is also able to figure out the particular error-hit area and uses partial reconfiguration to mitigate this error. In addition to the TMR and scrubbing, this system even utilizes spare redundant blocks in the event of error detection. The idea is inspired by the fact that handing over computation to a spare computation block is less time-intensive than reconfiguring the partially reconfigurable area to higher redundancy or scrubbing it. When all the spare computation blocks are exhausted, the computation starts from the initial block while reconfiguring all the blocks. The authors verify their approach and show that the combination of redundancy plus scrubbing plus spare resource technique improves the MTBF. However, this technique comes with the increased cost of holding extra resources for a single computation task. In satellite missions where cost is dominated by weight, latency, and power of resources, such a system is indeed fault tolerant but probably not cost-effective.

The authors in [32] utilize the integrated block RAMs on the FPGA to monitor the error rate, which is considered proportionate to the radiation strength to which an FPGA is exposed at run-time. The detected errors are counted and used to derive the probability of failures per hour (PFH). According to the PFH level, a processing module can be switched among three modes: no redundancy, DMR (dual modular redundancy), and TMR. The integrated BRAM Fault Detectors (BFD) implement radiation sensors at virtually no cost because the memory is still available for applications. However, the paper does not consider that the BRAM radiation sensor might detect not only SEEs due to radiation but also faults introduced by other effects like supply voltage instabilities and aging/permanent faults. In addition, the error counter registers are not protected. If an SEU hits an error counter, this might lead to a wrong error count. The reliability of the voter is not considered as well.

In contrast to the abovemetioned major adaptive fault-tolerance approaches, our technique, Dynamic Reliability Management (DRM), has a broader scope and can be customized for different reliability computation mechanisms as well as application software support. We not only justify our selected experimentation conditions but also propose the alternatives one can choose to have more extensive simulations, which can be observed in Section 4.

3. Dynamic Reliability Management

The dynamic reliability management (DRM) approach is divided into two parts, i.e., design-time and run-time.

3.1. DRM Design-Time Tool Flow

3.1. DRM Design-Time Tool Flow. In this section, we list the stages of our design-time tool flow. The tool flow, illustrated in Figure 8, converts an HDL design into a set of 4-dimensional Pareto-filtered implementations rated by the reliability magnitude, area consumption, latency, and dynamic power consumption. This overall tool flow is constructed by utilizing, extending, and creating various tools as described below:

(i) Tools utilized: Xilinx ISE (Mapping, Placement and Routing, Power Analyzer) and MATLAB Pareto filter

(ii) Tool extended: BANL TMR Tool

(iii) Tool created: MATLAB BDEC Tool

The extended subtools of BANL TMR tool, i.e., JEdifNMRSelection and JEdifVoterSelection, and the newly created tool, i.e., the revised BDEC tool, are marked as dark-shaded blocks in Figure 8 to highlight our contribution areas in the tool chain. The steps used in tool flow are discussed as follows.

3.1.1. Xilinx ISE Synthesis. In the first stage, we synthesize the benchmark HDL design with Xilinx ISE and output an EDIF netlist to be processed by the BANL TMR tool.

3.1.2. Replication via the BANL TMR Tool. The original BANL TMR tool (based on the Java programming language) was extended to support additional features which we explain as follows:

(i) *Original BANL Tool Flow*. The tool performs logic replication in four major stages:

  (1) The first stage comprising *JEdifBuild* and *JEdifAnalyze* performs the technical steps of design flattening, circuit and IOB analysis, etc., and saves the information in intermediate files to be used by the following tools.

  (2) The second stage comprising *JEdifNMRSelection* and *JEdifVoterSelection* determines the type of

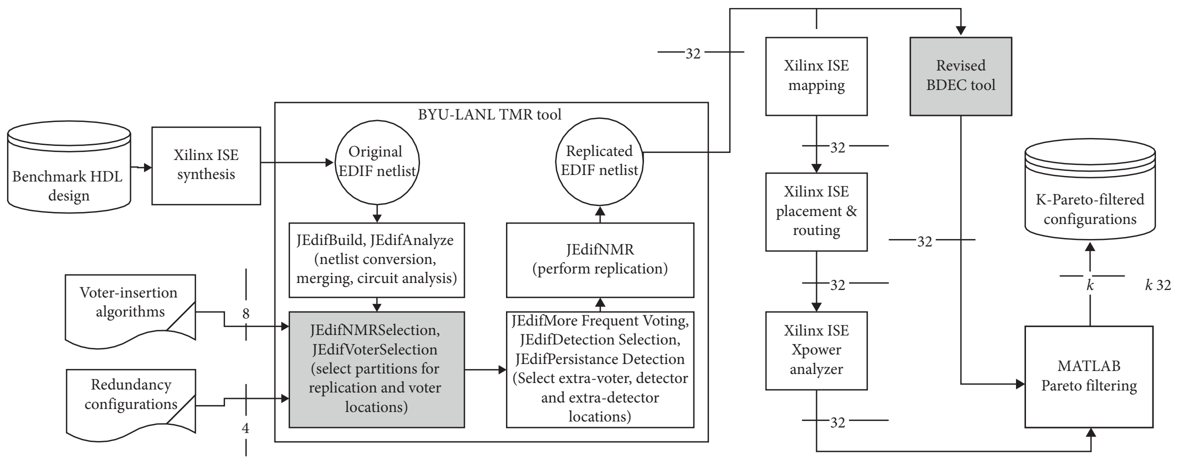
Benchmark HDL design
Xilinx ISE synthesis
BYU-LANL TMR tool
Original EDIF netlist
Replicated EDIF netlist
JEdifBuild, JEdifAnalyze (netlist conversion, merging, circuit analysis)
JEdifNMR (perform replication)
Voter-insertion algorithms
8
Redundancy configurations
4
JEdifNMRSelection, JEdifVoterSelection (select partitions for replication and voter locations)
JEdifMore Frequent Voting, JEdifDetection Selection, JEdifPersistance Detection (Select extra-voter, detector and extra-detector locations)
32
Xilinx ISE mapping
Revised BDEC tool
Xilinx ISE placement & routing
Xilinx ISE Xpower analyzer
K-Pareto-filtered configurations
k
k 32
MATLAB Pareto filtering

Figure 8: DRM deisgn-time tool flow.

configuration and replication to be used, replicates the instances, determines the voter-insertion locations by the specific algorithm used, and makes the necessary wire connections.

(3) The third stage can be run, if desired, for adding more voters and error detectors. For our research, we are excluding this stage as an optional one.

(4) The final stage does the actual replication by reading the intermediate file formats written by the previous tools and by generating the replicated netlist.

(ii) *Extension of BANL Tool.* We have made two extensions to the original BANL tool in order to support the additional redundancy configurations. By default, the tool supported only TMR configuration which we extended to three more configurations, namely, one alternate voter (OAV) [26], two alternate voters (TAV) [26], and cascaded TMR (CTMR) [27]. In addition, we have changed the command-line interface of the *JEdifNMRSelection* tool from replication type to configuration type because there is more than one configuration that uses triplication. As a result, we now can decide on a configuration instead of a particular replication type.

3.1.3. Performing Replication. After running the first stage of the tool, we need to choose one of the four redundancy configurations via the *JEdifNMRSelection* tool and one of the eight voter-insertion algorithms via the *JEdifVoterSelection* tool. The final stage of *JEdifNMR* reads the intermediate file formats of previously run tools and writes the final replicated EDIF netlist. Overall, we run the extended BANL TMR tool for 32 possible combinations of redundancy configurations and voter-insertion algorithms, which sums up the design implementation set.

3.1.4. Xilinx ISE Mapping, Placement/Routing, and Power Analysis. All of the 32 generated implementations of the design are passed through Xilinx ISE mapping, placement/routing, and XPower analyzer tools to obtain slice utilization (for area consumption), pad-pad delay/max. clock frequency (for latency), and dynamic power consumption, respectively. We resort to the dynamic power consumption since the static power almost remains the same throughout the analysis of a single benchmark.

3.1.5. Reliability Evaluation. In Section 2.4, we described three domains for reliability evaluation of redundant circuits on FPGAs. The choice of a specific reliability model depends on the need of the circuit/system designer. For a practical aerospace application, the CREME96 software should be employed to gain realistic reliability values. Testing and simulation-based methods are mostly useful to provide a level of confidence in commercial products. The probabilistic techniques, whether used with actual or arbitrary input error probability values, excel at comparing different redundant circuit configurations. The goal of our work is to analyze, evaluate, and compare fault-tolerant circuit schemes on FPGAs. We have decided to base our work on probabilistic computational reliability schemes due to following reasons:

(i) The mathematical models of these schemes are mature and have been analyzed in the literature [40, 41]

(ii) The probabilistic schemes are able to comprehend very small differences in reliability among slightly different redundant structures

Among the probabilistic computational reliability schemes, we decided to use the Boolean Difference Error Calculator (BDEC) method based on the analysis and merits/demerits of various computation schemes explained in our previous work [40–42]. The basic BDEC model was, however, limited as it did not cover sequential circuits and could not deal with redundant circuit structures. Moreover, no automated tool was available employing the BDEC model to calculate the reliability of circuits. Therefore, we first revised the original BDEC model to perform analysis of redundant as well as sequential circuits. Afterwards, we automated the revised model in the form of a MATLAB tool that takes FPGA-based synthesized Verilog netlist of the HDL design and computes the output reliability based on the provided component and wire error probabilities. This tool is hence utilized to compute the reliability magnitudes of the original nonredundant design and its 32 redundant implementations based on the reference input parameters.

3.1.6. Pareto Filtering of Implementation Points via MATLAB. Using the Pareto Front function of MATLAB [43], we reduce the set of implementation points to nondominated ones to make the selection of trade-off points easier. Each implementation is characterized on the basis of area, latency, power, and measured reliability. The resulting nondominated implementation points can be utilized by the system designer on the basis of one or more constraints of area, latency, power, and reliability.

3.2. DRM Run-Time Tool Flow. The final outcome of the design-time tool flow is the set of nondominated redundant configurations in addition to the original hardware design of the application module. The run-time system, on the other hand, stores and utilizes these configurations for balancing the reliability-performance trade-off using an operating system support. Another component of the run-time system is the decision module, which can be implemented in software or hardware. First, we briefly describe our proposed decision mechanisms followed by an insight into our run-time tool flow.

3.2.1. Decision Mechanisms for Changing Reliability Levels. In this section, we summarize four possible *decision mechanisms* which decide on changing reliability levels of hardware at run-time. The mechanisms are denoted as (i) external, (ii) time, (iii) cooperative, and (iv) radiation/error-rate-based mechanisms. Table 1 lists the four decision mechanisms and shows whether the system, e.g., the satellite, needs a radiation sensor and a so-called decision module that decides on changing the provided level of reliability.

Table 1: Decision mechanisms.

Decision mechanism	Sensor	Decision module
External	No	No
Time	No	Yes
Cooperative	Yes	No
Radiation/error	Yes	Yes

(1) *External*. With an external decision mechanism, a remote user has the control over reconfiguring the reliability configurations. For example, a ground control center transmits a signal to reconfigure the reliability levels of the satellite application based on available information about space weather. In this way, the radiation data is recorded by a source external to the satellite and the decision is made by the control station. Hence, both the components of the decision mechanism are external to the satellite.

(2) *Time*. The decision mechanism based on time can be used for missions where the radiation pattern is already known. For example, in Figure 5, the pattern of radiation can be used to plan the reliability reconfiguration after fixed time intervals calculated on the basis of the travel time of the satellite. With this mechanism, the time-based decision module stays within the application system though there is no sensor involved.

(3) *Cooperative*. Since the decision module is the most critical part of decision mechanism, either we protect it by excessive hardening or we can decide to keep it out of the application system. In this way, the decision is taken out of the system but the radiation data or proportional information, e.g., online error-rate measurement, is taken from the system which can be cross-verified before taking the decision for a reconfiguration. Hence, the term *cooperative* refers to the cooperation between the application system and the control station.

(4) *Radiation/Error*. With this technique, the sensor and the decision module both lie within the application system. Moreover, this only self-adaptive decision mechanism is responsible for collecting radiation data, its interpretation, and the reliability reconfiguration. In comparison with a time-based reconfiguration, which would be sufficient in cases when the radiation plot is known in advance, the radiation/error-based approach can also handle unexpected situations or uncertainty of the radiation plot as, for example, shown in the radiation sensor data plotted in Figure 4. If one wants to maximize system reliability, a self-adaptive decision mechanism is to be adopted even if the probability of such unexpected radiation changes is very low.

3.2.2. Run-Time Tool Flow with ReconOS. The operating system used for DRM on a platform FPGA is ReconOS [44–46]. ReconOS is an operating system for reconfigurable system-on-chip that extends the multithreaded programming model from software to reconfigurable logic cores. ReconOS leverages available host operating systems such as Linux and Xilkernel and allows for managing hardware and software threads. Both thread types can call operating system functions to interact with other threads and the operating system kernel using well-known programming objects such as semaphores, message boxes, and shared memory. An exemplary ReconOS system architecture is shown in Figure 9. The architecture comprises a main CPU, a number of reconfigurable hardware slots, a memory controller, and peripherals. Each hardware slot has two interfaces, an operating system interface (OSIF) for calling operating system functions and a memory interface (MEMIF) enabling direct access to the shared system memory. The main CPU runs an operating system kernel, ReconOS-specific extensions, and the software threads; hardware slots accommodate custom

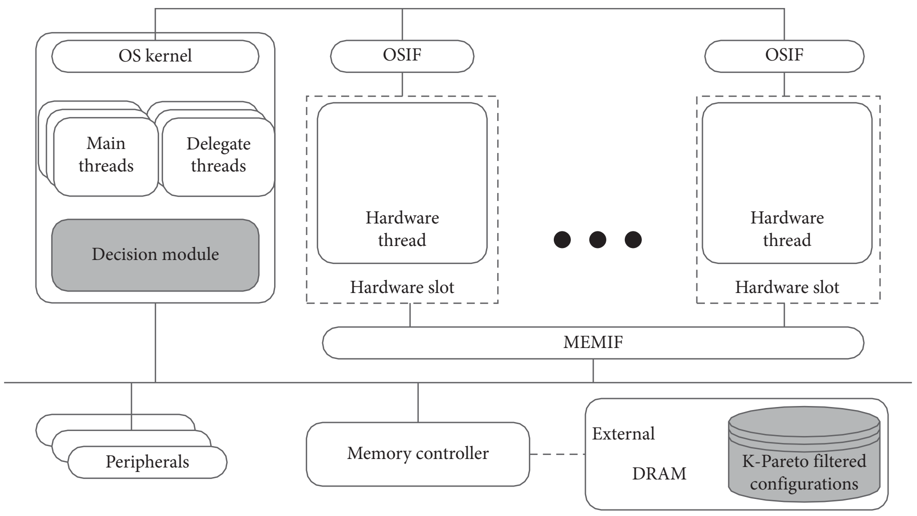

Figure 9: Exemplary ReconOS architecture.

hardware threads. Hardware threads communicate with the operating system kernel by means of delegate threads. These delegates call operating system functions on behalf of their corresponding hardware thread. Since ReconOS supports partial reconfiguration, both hardware and software threads can be instantiated, loaded, and started at run-time.

The DRM concept requires two additional components over a standard ReconOS system, the decision module running as a software thread on the main processor, and the database containing all the implementation variants for the hardware designs. Figure 9 depicts these additional components as dark-shaded blocks. The implementation of decision module in hardware and alternate storage possibilities of redundant configurations apart from DRAM are also possible although investigating the difference in performance due to these alternate strategies correspond to our future work.

The decision mechanisms for reliability can be used with ReconOS as follows. When an application starts and instantiates a hardware design, it creates the corresponding hardware thread and requests ReconOS to load the hardware thread into one of the hardware slots. Subsequently, ReconOS calls the DRM decision module to decide for an actual implementation variant for that thread. Depending on the used approach, the decision module is driven by user commands, time events, or measurements of the radiation level or actual error rates. Afterwards, ReconOS retrieves the selected implementation variant from the database, i.e., external DRAM in our case, and configures it into a hardware slot. Any time during operation, the decision module may request ReconOS to reconfigure a hardware thread with an alternative implementation variant. In this paper, we focus on reliability of hardware designs, which are mapped to ReconOS hardware threads. Eventually, the reliability of the overall ReconOS system will have to be considered including the main CPU, the threads' operating system and memory interfaces (OSIF and MEMIF), buses, and memory controllers. The simplest way is to configure the ReconOS system to highest reliability implementation to ensure reliable operation of the switching mechanism, which adds a fixed cost of redundancy of the base system. The database of implementation variants is stored in external DRAM to allow for a fast reconfiguration. Reliability for external DRAM can be provided through error correction codes.

4. Experimentation

The design-time and run-time parts of the DRM are validated with extensive experimentation using a number of benchmarks. For design-time part, we use a 32-point implementation set to observe how the performance factors scale with the circuit size and architecture. In contrast, the run-time part is verified with a 3-point implementation set to make the reconfiguration process easy for the reader to comprehend. The target device used is a Virtex 5 FPGA, XCVTX150T, with package FF1156. Although our selected FPGA is not amongst the latest Xilinx FPGAs, we are limited by the BANL TMR tool, which does not support the FPGA models above series 5. On the other hand, our technique is generic and equally applicable to all FPGAs; therefore, we do not anticipate a different analysis with the latest FPGA series.

4.1. Validating Design-Time Tool Flow. In order to analyze the variation in performance factors, i.e., area consumption, latency, dynamic power consumption, and reliability with respect to changes in the redundancy configuration and voter-insertion algorithm, we analyzed six benchmark HDL designs from three classes of benchmarks with different circuit architectures [47]: c17 and c3540 from ISCAS'85, s713 and s838 from ISCAS'89, and b8 and b12 from ISCAS'99 benchmark suites. The complete design-time analysis is provided in our research work [48]. In this paper, we will reproduce the results of only two benchmarks, i.e., s713 and s838 so that the paper will not be overwhelmed by extensive design-time experiments.

The experiments use overall four redundancy configurations including OAV (one alternate voter), TAV (two alternate voters), TV (triplicated voters), and CTMR (cascaded TMR-level 1) from Section 2.2, as well as NR, a nonredundant design for comparison. We have set the optimization method for design mapping to *balanced* to ensure the best combination of area and speed efficiency. For latency comparison, we use the maximum pad-pad delay and maximum clock frequency for combinational and sequential circuits, respectively. To determine dynamic power consumption, we have applied random testbench signals to each benchmark while trying to maximize the signal activity among the circuit intermediate nodes. Because of lack of standard testbenches available for these benchmarks, our randomly applied testbenches do not necessarily account for the maximum possible power consumption. However, for the sake of comparison among redundant configurations, they serve the purpose.

The reliability results in this experimentation are interpreted as the average reliability of the outputs of the circuit. The output reliabilities are based on the error probabilites of input, component (i.e., LUT, MUX, RAM, and so on), and voter taken as 1%. The overall output error probability is an indicative of each component and input to be probabilistically affected by a radiation particle at a rate of 1%. In reality, this is an extremely high overestimate of component uncertainty which can be verified from device reliability reports, having error rates in the range of 1–10 errors in one billion hours [32, 34]. Using such low error rates, the reliability is normally represented in MTBF units in conventional time-dependent reliability theories. However, the probabilistic reliability models including BDEC are timeless and there is no notion of failure rate and hence MTBF in these approaches. Therefore, without having the feasibility to use failure rate units when low error probabilities are used in the BDEC model, the reliability results get complicated to understand by the user, i.e., the reliabilities among different implementations vary at smaller decimal places. Since our experimentation depends on *relative* reliability computation among circuit implementations, we resort to a component error of 1% to make the results comprehensible by the reader. It is worth mentioning here that the size of the benchmarks chosen are moderate in this experimentation (max. 800 slices) since the current version of our BDEC-reliability tool is very time-intensive due to the sequential flow of probability in this model. Although we are making efforts to improve the performance of this tool by parallel programming and multicore processing in the future, the reader can still observe the analysis on performance parameters (excluding reliability) for large benchmarks in our previous work [48, 49].

4.1.1. s713 and s838 (ISCAS'89). This series of HDL benchmarks consist of combinational as well as sequential elements due to which the voter placement algorithms result in large variations of the performance factors. Table 2 lists the results for the two benchmarks and highlights the nondominated, i.e., Pareto-optimal, implementations of the HDL design. Those implementation points having similar parameter values are highlighted only once. The benefit of Pareto filtering is obvious as the 32-point set is reduced to 12 and 7 points for s713 and s838, respectively. It can be observed from the table that the span of parameters for OAV, TAV, and TV is not high since they only differ in number of voters as compared with NR and CTMR, which vary in number of modules as well. Moreover, we can observe that different voter-insertion algorithms can greatly vary the trade-off points.

While one would assume that the area utilization always increases in ascending order from NR to CTMR, the experiments prove this assumption wrong. As can be observed for the HFC algorithm and the s713 benchmark, the configuration TV consumes less slices than the configuration TAV, albeit the number of voters for TV is higher than for TAV. The explanation for such anomalies lies again in the automatic mapping, placement, and routing tools. Sometimes, resources such as flip-flops remain unused in slices to balance the timing constraints and, more generally, the optimization possibilities vary from one design to another. Furthermore, the choice of voter-insertion algorithm within each configuration has a high impact on the area consumption, e.g., for the s838 benchmark, the slice utilization varies from 113 to 124 (9.7% variation) for OAV configurations in contrast to 133 to 276 slices (107.5% variation) for CTMR configurations. Similarly, because of different optimization possibilities, the maximum clock frequency also does not always decrease with configurations using higher degrees of replication. For example, the maximum clock frequency increased for the s713 benchmark design and the CC algorithm when going from TAV to TV. However, switching from the nonredundant to redundant configurations drastically impacts the maximum clock frequency. For example, when going from NR to CTMR, we observe a 45% decrease for s713. The maximum clock frequency also varies considerably with variation of the voter-insertion algorithm, e.g., 29% variation for s713 and the TAV configuration. The dynamic power consumption varies minimally for these benchmarks, with a maximum variation of 9.8% observed for s838 and the CTMR configuration. Generally, the reliability always increases from NR to CTMR for a single voter-insertion algorithm; although when comparison is made among different algorithms, it can be easily observable that lower redundancy configuration of one algorithm may be more reliable than higher redundancy version of another algorithm. For example, for s713 benchmark, the TV configuration with the BFC algorithm achieves less reliability than TAV configuration with the BD algorithm. The reliability of CTMR versions is always higher than other configurations even when using different voter-insertion algorithms. However, their variation is very small using different algorithms. A very noticeable advantage we gained from this design space exploration is the implementation point obtained with CTMR and the HFFOC algorithm for s838 benchmark, where the performance parameters are very comparable to the triplicated configuration though having higher reliability. Hence, the placement and routing of the design based on different voter-insertion algorithms can make highly redundant designs cost-effective.

Based on this experiment, we conclude that the performance factors do not follow fixed patterns based on the redundancy structures and voter algorithms. There is a

Table 2: Design space exploration results for the benchmarks s713 and s838 from the ISCAS'89 benchmark suite.

			CC	AC	BFC	BD	HFC	HFFC	HFFIC	HFFOC
s713	NR	#Slice	42							
		Max Freq (MHz)	336							
		Dynamic PD (W)	1.476							
		Reliability	0.9568							
	OAV	#Slice	93	78	86	93	81	85	85	85
		Max Freq (MHz)	233	262	249	233	263	273	255	273
		Dynamic PD (W)	1.997	1.988	1.986	1.997	1.993	1.993	1.988	1.993
		Reliability	0.9767	0.9673	0.9763	0.9767	0.9765	0.9763	0.9763	0.9763
	TAV	#Slice	101	97	97	101	98	106	93	106
		Max Freq (MHz)	205	265	218	205	220	231	246	231
		Dynamic PD (W)	1.933	1.993	2.006	1.993	1.990	1.999	1.986	1.999
		Reliability	0.9772	0.9767	0.9767	0.9772	0.9769	0.9767	0.9766	0.9767
	TV	#Slice	102	117	94	102	86	117	92	117
		Max Freq (MHz)	227	207	222	227	240	207	233	207
		Dynamic PD (W)	2.002	1.996	2.001	2.002	1.998	1.996	1.998	1.996
		Reliability	0.9774	0.9769	0.9767	0.9774	0.9771	0.9769	0.9766	0.9769
	CTMR	#Slice	211	214	223	211	173	214	223	214
		Max Freq (MHz)	219	232	228	219	219	232	228	232
		Dynamic PD (W)	2.083	2.073	2.079	2.083	2.050	2.073	2.079	2.073
		Reliability	0.9790	0.9789	0.9789	0.9790	0.9789	0.9789	0.9789	0.9789
s838	NR	#Slice	33							
		Max Freq (MHz)	260							
		Dynamic PD (W)	0.287							
		Reliability	0.9384							
	OAV	#Slice	113	117	124	113	117	117	124	117
		Max Freq (MHz)	222	205	228	214	205	205	228	205
		Dynamic PD (W)	0.317	0.305	0.316	0.317	0.305	0.305	0.316	0.305
		Reliability	0.9735	0.9736	0.9732	0.9735	0.9736	0.9735	0.9732	0.9736
	TAV	#Slice	147	131	133	147	131	131	133	131
		Max Freq (MHz)	217	212	205	217	212	212	205	212
		Dynamic PD (W)	0.317	0.320	0.322	0.317	0.320	0.320	0.322	0.320
		Reliability	0.9746	0.9746	0.9740	0.9746	0.9746	0.9746	0.9740	0.9746
	TV	#Slice	166	173	140	166	173	173	140	173
		Max Freq (MHz)	203	227	208	203	227	227	208	227
		Dynamic PD (W)	0.328	0.327	0.325	0.328	0.327	0.327	0.325	0.327
		Reliability	0.9752	0.9752	0.9745	0.9752	0.9752	0.9752	0.9745	0.9752
	CTMR	#Slice	276	133	212	276	133	133	212	133
		Max Freq (MHz)	203	219	203	203	219	219	203	219
		Dynamic PD (W)	0.336	0.306	0.321	0.336	0.306	0.306	0.321	0.306
		Reliability	0.9793	0.9793	0.9793	0.9793	0.9793	0.9793	0.9793	0.9793

different optimization potential for different circuit structures on FPGAs. Therefore, the Pareto-optimization and such design space exploration are very helpful in selecting the right redundant structure that follows one's specific performance constraints.

4.2. Validating Run-time Tool Flow. The Pareto-filtered implementations generated from the design-time tool flow need to be utilized and reconfigured at run-time using the run-time tool flow discussed in Section 3.2. For experimentation, instead of a 32-point implementation set, we use only a 3-point set using configurations NR, TMR, and CTMR while using the voter algorithm CC. The reason for using this small set is to analyze and demonstrate the reconfiguration process effectively instead of presenting numerous reconfigurations required in a larger implementation set. Moreover, the process of reconfiguration holds similar for all sizes of implementation sets. The design-time analysis of these hardware designs would be similar to benchmarks used in the previous section; it is also skipped here since we use only three implementation points for demonstration purposes instead of using a fully Pareto-optimized implementation set. To analyze the trade-off between reliability and performance given by different implementation variants of a design, we used two case studies of hardware designs, namely, *data sorter* and *matrix multiplier* [50].

The DRM implementation on a reconfigurable SoC platform is faced with the challenge of area efficiency. The problem we observed before implementing the DRM concept was that the area of a reconfigurable slot has to be limited to a fixed size, which refers to the maximum area of a redundant implementation we use in the reconfigurable partition, i.e., CTMR. Hence, when the reconfigurable partition is utilized with relatively smaller redundant

Table 3: Hardware slot combinations.

Application Configuration	Hardware slot #1	Hardware slot #2	Hardware slot #3
Static-maximum-reliability	CTMR	CTMR	CTMR
Static-varying-reliability	CT MR	TM R∗2	NR∗8
Reconfigurable	CTMR|TMR∗2| NR∗8	CTMR|TMR∗2| NR∗8	CTMR|TMR∗2| NR∗8

structures such as TMR, we do not obtain an area efficiency since the unused portion of the partition cannot be utilized for other logic. Therefore, we use the concept of parallelism to fully utilize the reconfigurable partition for each of the redundant implementations smaller than CTMR, i.e., the NR and TMR structures will be implemented as parallelized versions. Hence, the reconfigurable slot is fully utilized while providing a higher throughput due to parallel application engines.

4.2.1. Data Sorter. The data-sorting hardware thread uses a bubble sorting algorithm to sort data in an ascending order. The sorting thread operates on 8 KB blocks of 32-bit integer data, and its performance is measured by the sorting rate, in blocks/minute. The sorting application has been chosen due to its block-based processing usage which is considered typical for signal/multimedia processing, data compression, and encryption tasks [51, 52].

In our experiments, we allocate a certain hardware area for implementing the sorting function. The area is chosen such to fit the sorter implementation with the highest level of reliability, i.e., CTMR. The same area can also be used for an internally parallel instance of the hardware sorter in the TMR version and for a sorter in the NR version that employs eight parallel sorting engines. We have developed these parallel versions of a data sorter for TMR and NR, as well as the CTMR version as ReconOS hardware threads that are designed to operate at 100 MHz. We denote the resulting reliability versions as CTMR, TMR∗2, and NR∗8.

We experiment with a ReconOS system, as shown in Figure 9, employing three hardware slots and compare three different configurations for the data sorter application, respectively, for utilizing them. The configurations are denoted as static-maximum-reliability, static-varying-reliability, and reconfigurable. In the static-maximum-reliability configuration, we strive for maximum reliability and employ the CTMR sorter in each of the three slots in a static way, i.e., without partial reconfiguration. The so-called static-varying-reliability configuration is also static but uses all three redundancy versions of the sorter, i.e., NR∗8, TMR∗2, and CTMR at the same time, each one in a separate hardware slot. During run-time, we can switch the hardware threads on and off based on the reliability requirements. Finally, in the reconfigurable configuration, we utilize partial reconfiguration to reconfigure the hardware slots with sorter threads matching the reliability requirements. That is, at a particular instant, all the three threads are configured either NR∗8 or TMR∗2 or CTMR. The organization of these configurations is illustrated in Table 3.

The entire ReconOS base system was designed using Xilinx EDK. The run-time reconfiguration and performance measurements were performed via software developed with Xilinx SDK and downloaded to a Microblaze processor implemented on FPGA. The partial bitstreams for the dynamically reconfigurable regions, i.e., the hardware slots, are generated by the Xilinx Partial Reconfiguration tool flow [53]. There are nine partial bitstreams representing NR∗8, TMR∗2, and CTMR versions for each of the three hardware slots. The full bitstream is always generated with the NR versions of the hardware threads.

The decision mechanism we envision for our experiments is radiation-based, utilizing the radiation profile obtained from the Borealis flight (duration 103 minutes) shown in Section 2.3 (though the radiation strike rate in Figure 4 is shown with respect to altitude, the time-dependent variation, as used in this experimentation, has a similar trend). Since we have three reliability versions for the data sorter, i.e., NR, TMR, and CTMR, we divide the radiation levels into the following three ranges corresponding to three reliability levels:

(i) Reliability level 1: 0–300 counts/min

(ii) Reliability level 2: 300–600 counts/min

(iii) Reliability level 3: 600–900 counts/min

The radiation data are stored on the Microblaze, and each radiation sample is read and interpreted after a time interval of one minute, according to the data frequency of the Borealis flight. The NR∗8, TMR∗2, and CTMR implementations, as partial bitstreams, are loaded into the SDRAM associated with the Microblaze and used for reconfiguration via the ICAP interface of the FPGA. The evaluation platform used is Xilinx ML605 Board, which is equipped with a Virtex 6 XC6VLX240T FPGA. The data to be sorted are continuously provided to the hardware threads until the completion of experiment or Borealis flight duration. Each block of data is comprised of 2048 32-bit words.

The sorting performance for the three configurations is compared in Figure 10. For reliability level 1, i.e., 0–44 minutes, we can observe that the sorting rate of reconfigurable configuration is more than double and around eight times higher than static-varying-reliability and static-maximum-reliability configurations, respectively. From 44–46 minutes, the sudden increase in radiation rate, as recorded by the sensor, makes the three threads reconfigure to TMR∗2 equivalents in the reconfigurable configuration. Similarly, for the static-varying-reliability configuration, the NR∗8 thread is switched off while dropping the sorting rate by 3.7 times since the total sorting

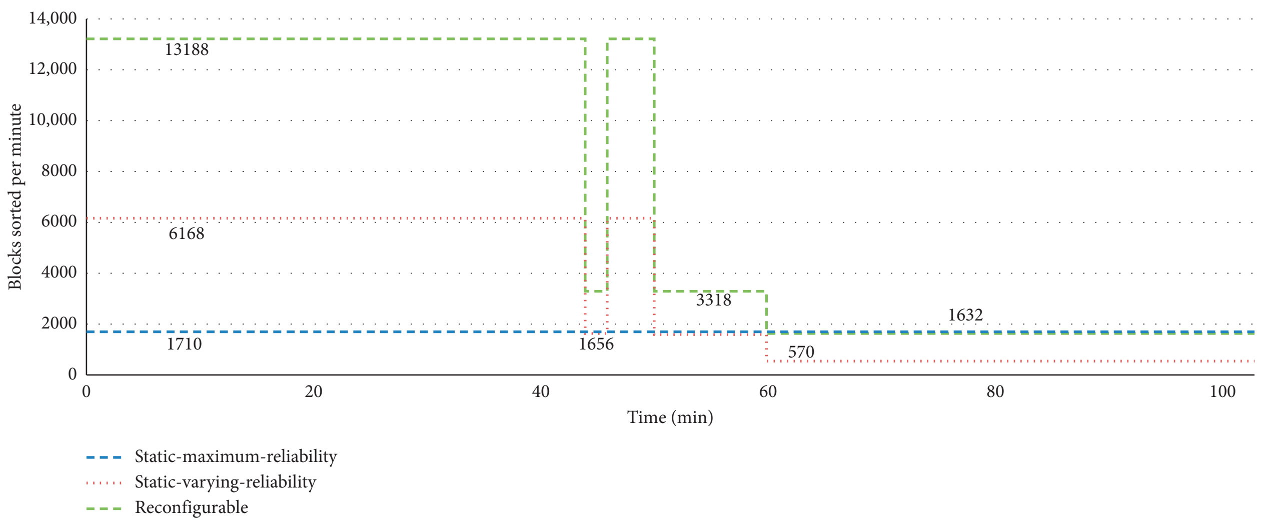

Figure 10: Performance results for static-maximum-reliability, static-varying-reliability, and reconfigurable configurations of data sorter.

units decreased by the same magnitude as well. The similar transition occurs for the 46–50 min range corresponding to reliability level 1. For the 50–60 min range, the reliability level shifts to the second category. In this region, the total working units for the reconfigurable configuration are six, compared with three units for each of static-varying-reliability and static-maximum-reliability configurations. However, the sorting rate of static-varying-reliability configuration is slightly lower than static-maximum-reliability configuration due to extra complexity of the software code used for continuous radiation monitoring in the static-varying-reliability case. Similarly, the last reliability level, with range 60–103 min, has three sorting units working in reconfigurable and static-maximum-reliability configurations, while only a single unit is utilized for the static-varying-reliability case. The slight difference for the sorting rate for reconfigurable case compared with static-maximum-reliability one is due to the same code-complexity reason described above. However, the static-varying-reliability configuration is ineffective even compared with static-maximum-reliability configuration in highest reliability requirement.

Overall, the reconfigurable configuration, upon which the DRM technique is based, outperforms both the other configurations. In lowest reliability requirements, the performance is 2.1 and 7.7 times higher compared with static-varying-reliability and static-maximum-reliability configurations, respectively. For highest reliability requirement, the reconfigurable configuration has a performance comparable with static-maximum-reliability and 2.8 times higher than static-varying-reliability configuration. The static-varying-reliability configuration, on the other hand, is 3.6 times faster and three times slower than the static-maximum-reliability configuration in highest and lowest reliability requirements, respectively. The data throughput of the three configurations can also be compared by the total data sorted at the end of the experiment, i.e., $7.43e5$, $3.40e5$, and $1.76e5$ blocks for reconfigurable, static-varying-reliability, and static-maximum-reliability configurations, respectively. The time spent during each reconfiguration stage is 228 ms (76 ms for each thread). Since our radiation sampling rate is one minute, the reconfiguration time is negligible and hence, not suitable to be represented on the graph, having time scale in minutes. Moreover, the reconfiguration time can be further decreased by using different FPGA architectures [54] or utilizing processor-independent partial reconfiguration [55].

4.2.2. *Matrix Multiplier.*

Matrix multiplication is a heart of many image-processing algorithms [56, 57]. The matrix multiplication application we use as our case study multiplies integer matrices of 128 columns and 128 rows. It works by reading in Matrix B completely and Matrix A row-wise, thereby performing a row-wise multiplication and then writing back the result row-wise until the whole matrix multiplication is completed. The hardware thread is part of an application that uses the Strassen algorithm [58] to split the multiplication of a 512 columns by 512 rows matrix into 49 multiplications of smaller (128×128) matrices. This way, the matrix multiplication is parallelizable and the workload can be distributed among many hardware threads.

The experimentation platform for this application, including hardware generation and experimental duration is the same as for the data sorter. However, because of different size of this hardware application, the parallelization is different, i.e., the CTMR version of matrix multiplier accommodates two instances of TMR and five instances of NR, denoted as CTMR, TMR∗2, and NR∗5 respectively, in contrast to eight NR versions for the data sorter application. Moreover, the reconfiguration time for each slot is double compared to data sorter due to double the size of hardware utilized, i.e., two clock regions compared with one for the data sorter.

The performance results of the matrix multiplier are shown in Figure 11. For reliability level 1, i.e., 0–44 minutes, we can observe that the multiplication rate of reconfigurable configuration is 1.7 and 3.7 times higher than static-varying-

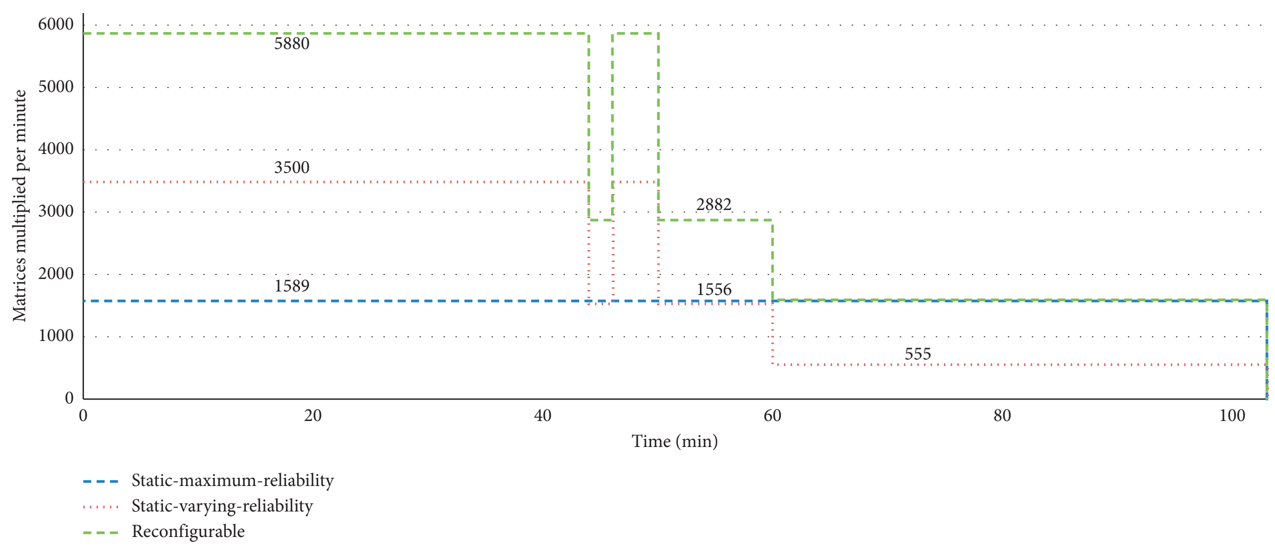

FIGURE 11: Performance results for static-maximum-reliability, static-varying-reliability, and reconfigurable configurations of the matrix multiplier.

reliability and static-maximum-reliability configurations, respectively, whereas the number of working units are higher by a factor of almost double and five times, respectively. The performance does not exactly scale with the number of working units and depends on the architecture and how efficiently a hardware design is parallelized. Overall, the reconfigurable configuration again outperforms both of the other configurations. In lowest reliability requirements, as mentioned before, the performance is 1.7 and 3.7 times higher compared with static-varying-reliability and static-maximum-reliability configurations, respectively. For highest reliability requirement, the reconfigurable configuration has comparable performance to static-maximum-reliability and 2.86 times higher than static-varying-reliability configuration. The static-varying-reliability configuration, on the other hand, is 2.2 times faster and 2.86 times slower than the static-maximum-reliability configuration in lowest and highest reliability requirements, respectively. The data throughput of the three configurations can also be compared by the total matrices multiplied at the end of the experiment, i.e., 3.85*e*5, 2.11*e*5, and 1.64*e*5 matrices for reconfigurable, static-varying-reliability, and static-maximum-reliability configurations, respectively.

4.3. Possible Extensions to the Experimentation Platform. During the course of validation of DRM, we developed an experimentation platform that could be extended in multiple ways. It is practically hard to enlarge the scope of our experimentation to cover all the possible system architectures in our experimentation. However, our future work will address some or all of these improvements as discussed below:

(i) Our focus was studying the reliability-performance trade-off for the application module only. However, the overall operating system and its interface should be accounted for similar reliability requirements. The simplest approach of ensuring the reliability of the base system is to implement it to the highest reliability/redundancy level, as also proposed in [6, 32], thereby adding a constant overhead in the performance of the system including application modules. Our future work will involve making the base system redundant including the processor Microblaze and then observing the performance benefit of DRM.

(ii) The performance of the system has been studied only while utilizing redundancy in this work, whereas the reliable systems employ scrubbing as an additional reliability-enhancement technique. In the presence of scrubbing, the reliability of the system will be computed according to a more complex mathematical model [59]. Moreover, since scrubbing is performed via ICAP interface of the FPGA and so is the partial reconfiguration for our DRM approach, it may cause performance degradation while the reconfiguration process is blocked by the scrubbing cycle. This effect will be investigated in our future work. Moreover, the power consumption due to the reconfiguration process will also be documented in the run-time experiments.

(iii) The implementation of decision mechanism in DRM can be studied for different approaches based on the error rate or radiation-level measurements. In our work, we utilized a basic approach of dividing radiation data into different domains and taking the reconfiguration decision based on crossing the domain thresholds. However, we can also use the real-time error-rate measurements while taking the reconfiguration decisions with more accuracy, using, for example, an extended BRAM sensor [32] implementing our proposed extensions in Section 2.5. Moreover, the performance of decision mechanism can be improved by implementing it in hardware in contrast to a software approach. It is also worth mentioning that utilizing a nonredundant (NR) implementation for lowest reliability requirement is just for the sake of understanding the reconfiguration concept; it does not guarantee that the system will be reliable being nonredundant.

(iv) The storage mechanism of partial bitstreams has to ensure the integrity of bitstream since we cannot afford the corruption of these bitstreams representing the whole functionality of hardware design. In our work, we stored them in DRAM while proposing ECC to protect them against corruption. The storage mechanism can be made more robust by additional reliability techniques, which correspond to our future work.

(v) In our work, the reliability of different redundant implementations was calculated offline using a known analytic reliability model. This approach can be made more compact by characterizing reliability at run-time using mathematical approaches. The decision algorithm can be made more compact which takes into account the parameters based on the hardware design, environmental conditions, and system constraints.

(vi) The real space applications involve extensive simulations considering space weather and solar conditions. In this work, we demonstrated the approach in a broader picture without stressing that our approximate reliability computation method is comparable with the real-time simulations for space environments. In our future work, we will utilize the real aerospace environment and use the CREME96 tool to observe the difference in performance of a DRM-based application.

5. Conclusion

In this paper, we describe the need for building an adaptive system for optimizing reliability-performance trade-off for redundant FPGA hardware, at run-time, based on the radiation strength of the environment. We provide a survey of all the major approaches proposing such an adaptive system concept and contrast them with our new approach of Dynamic Reliability Management (DRM). Furthermore, we discussed some real radiation profiles and the possible approaches to build decision mechanisms for adaptive reconfiguration in radiation-sensitive applications. The design-time and run-time tool flow of DRM have been described in detail including tools utilized, extended, and developed. The design-time tool flow has been verified with various circuit benchmarks that show how the performance factors vary due to placement and routing decisions of the FPGA design software. This performance optimization has been made easy using our design space exploration technique, which analyses various redundant implementations of a hardware design. The run-time tool flow has been tested for data sorting and matrix multiplication case studies under the radiation data profile of Borealis flight experiment. To investigate the benefits obtained by the partial reconfiguration approach, we developed a system-on-chip experimentation platform which provides the performance comparison for different application configurations, i.e., static-maximum-reliability, static-varying-reliability, and reconfigurable. Our results prove that the dynamic partial reconfiguration, on which the DRM technique is based, provides the best performance results, up to 7.7 and 3.7 times higher throughput for our data sorting and matrix multiplication case studies, respectively, as compared with static reliability management techniques. The usage of radiation data in our experimentation is one possible way of implementing decision mechanism. However, its implementation in hardware or usage of different decision mechanisms is up to the implementation preferences of the system designer. In the future, we intend to experiment with different decision mechanisms and their implementation using different storage mechanisms in hardware and software.

Data Availability

The experimental data used to support the findings of this study are included within the articleas well as in authors' previous research works [40–42, 48–50].

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the International Graduate School of Dynamic Intelligent Systems (IGS-DIS) and Paderborn Center for Parallel Computing (PC2) of the Paderborn University.

References

[1] C. Huang, *Robust Computing with Nanoscale Devices: Progresses and Challenges*, Springer, Berlin, Germany, 2010.

[2] M. Stanisavljevic´, A. Schmid, and Y. Leblebici, *Reliability of Nanoscale Circuits and Systems: Methodologies and Circuit Architectures*, Springer, Berlin, Germany, 2010.

[3] M. Tehranipoor, "Emerging nanotechnologies: test, defect-tolerance and reliability, series," *Frontiers in Electronic Testing*, Vol. 37, Springer, Berlin, Germany, 2008.

[4] A. Abdollahi, "Probabilistic decision diagrams for exact probabilistic analysis," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 266–272, San Jose, CA, USA, November 2007.

[5] J. M. Johnson, "Synchronization voter insertion algorithms for FPGA designs using triple modular redundancy," Master's thesis, Department of Electrical and Computer Engineering, Brigham Young University, Provo, UT, USA, 2010.

[6] A. Jacobs, G. Cieslewski, A. D. George, A. Gordon-Ross, and H. Lam, "Reconfigurable fault tolerance: a comprehensive framework for reliable and adaptive FPGA-based space computing," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 5, no. 4, pp. 1–30, 2012.

[7] H. Quinn, D. Roussel-Dupre, M. Caffrey et al., "The cibola flight experiment," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 8, no. 1, pp. 1–22, 2015.

[8] JEDEC, *Global Standards for the Microelectronic Industry*, 2019, https://www.jedec.org/standards-documents/dictionary.

[9] R. Do, "Automated triple modular redundancy," *Web-Seminar, Mentor Graphics*, http://www.mentor.com/products/fpga/multimedia/automated-triple-modular-redundancy-how-and-when-to-use-it, 2011.

[10] Actel RTAX-S/SL FPGAs, 2015, http://www.microsemi.com/products/fpga-soc/radtolerant-fpgas/rtax-s-sl.

[11] Actel RT ProASIC3 FPGA, 2015, http://www.microsemi.com/products/fpga-soc/radtolerant-fpgas/rt-proasic3.

[12] H. Quinn, *An introduction to mission risk and risk mitigation for xilinx SRAM FPGAs*, Technical Report, Los Alamos National Laboratories, Santa Fe, New Mexico, USA, 2008, ftp://ftp.lanl.gov/public/hquinn/quinn_intro_to_rad2.pdf.

[13] Space-grade Xilinx Virtex 5QV, 2015, http://www.xilinx.com/products/silicon-devices/fpga/virtex-5qv.html.

[14] C. Carmichael, "Triple module redundancy design techniques for virtex FPGAs," Xilinx Application Note, XAPP197 (v1.0.1), 2006.

[15] F. L. Kastensmidt, L. Carro, and R. Reis, *Fault-Tolerance Techniques for SRAM-Based FPGAs (Frontiers in Electronic Testing)*, Springer-Verlag, New York, NY, USA, 2006.

[16] I. Herrera-Alzu and M. Lpez-Vallejo, "Self-reference scrubber for tmr systems based on xilinx virtex fpgas," in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization, and Simu- Lation, Ser. Lecture Notes in Computer Science*, J. Ayala, B. Garca-Cmara, M. Prieto, M. Ruggiero, and G. Sicard, Eds., vol. 6951, pp. 133–142, Springer, Berlin, Heidelberg, Germany, 2011.

[17] G. Miller, C. Carmichael, and G. Swift, "Single-event upset mitigation for xilinx FPGA block memories," Xilinx Technical Report, XAPP962 (v1.1), 2008.

[18] J. Johnson, W. Howes, M. Wirthlin et al., "Using duplication with compare for on-line error detection in FPGA-based designs," in *Proceedings of the 2008 IEEE Aerospace Conference*, pp. 1–11, Big Sky, MT, USA, March 2008.

[19] D. L. Foster, *Area Constrained Partial Fault Tolerance*, ProQuest, UMI Dissertation Publishing, ProQuest, UMI Dissertation Publishing, 2011.

[20] G. Bosilca, R. Delmas, J. Dongarra, and J. Langou, "Algorithm-based fault tolerance applied to high performance computing," *Journal of Parallel and Distributed Computing*, vol. 69, no. 4, pp. 410–416, 2009.

[21] TMR Tool, 2012, http://www.xilinx.com/ise/optional_prod/tmrtoolhtm.

[22] Precision Hi-Rel Synthesis Software, 2012, http://www.mentor.com/products/fpga/synthesis.

[23] BYU EDIF Tools Homepage, 2012, http://reliability.ee.byu.edu/edif.

[24] J. S. Hane, B. J. LaMeres, T. Kaiser, R. Weber, and T. Buerkle, "Increasing radiation tolerance of field-programmable-gate-array-based computers through redundancy and environmental awareness," *Journal of Aerospace Information Systems*, vol. 11, no. 2, pp. 68–81, 2014.

[25] J. M. Johnson and M. J. Wirthlin, "Voter insertion algorithms for FPGA designs using triple modular redundancy," in *Proceedings of the 18th annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, pp. 249–258, 2010.

[26] S. Lee, J.-i. Jung, and I. Lee, "Voting structures for cascaded triple modular redundant modules," *IEICE Electronics Express*, vol. 4, no. 21, pp. 657–664, 2007.

[27] D. Bhaduri and S. K. Shukla, "NANOPRISM: a tool for evaluating granularity vs. Reliability trade-offs in nano architectures," in *Proceedings of the 14th ACM Great Lakes symposium on VLSI*, pp. 109–112, Boston USA, 2004.

[28] Flight Results: 2013 High Altitude Balloon Design Program, MSGC BOREALIS Flight on 7/23/13, 2013, https://slideplayer.com/slide/5904101/.

[29] M. Caffrey, K. Morgan, D. Roussel-Dupre et al., "On-orbit flight results from the reconfigurable cibola flight experiment satellite (CFESat)," in *Proceedings of the 17th IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, pp. 3–10, Napa, CA, USA, April 2009.

[30] CREME96 Tool Website, 1996, https://creme.isde.vanderbilt.edu.

[31] J. D. Engel, K. S. Morgan, M. J. Wirthlin, and P. S. Graham, *Predicting On-Orbit Static Single Event Upset Rates in Xilinx Virtex FPGAs*, Tech. Rep., Brigham Young University, Provo, UT, USA, 2006.

[32] R. Glein, B. Schmidt, F. Rittner, J. Teich, and D. Ziener, "A self- adaptive SEU mitigation system for FPGAs with an internal block RAM radiation particle sensor," in *Proceedings of the IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 251–258, Boston, MA, USA, May 2014.

[33] E. Johnson, M. J. Wirthlin, and M. Caffrey, "Single-event upset simulation on an FPGA," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, CSREA Press, Las Vegas NV, USA, pp. 68–73, June 2002.

[34] Xilinx, *Device Reliability Report UG116 (v10.2.1)*, http://www.xilinx.com/support/documentation/userguides/ug116.pdf, 2015.

[35] J. Hussein and G. Swift, *Mitigating Single-Event Upsets, Xilinx White Paper (WP395) (v1.1)*, http://www.xilinx.com/support/documentation/white_papers/wp395-Mitigating-SEUs.pdf, 2015.

[36] J. Han, H. Chen, E. Boykin, and J. Fortes, "Reliability evaluation of logic circuits using probabilistic gate models," *Microelectronics Reliability*, vol. 51, no. 2, pp. 468–476, 2011.

[37] S. Krishnaswamy, G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Probabilistic transfer matrices in symbolic reliability analysis of logic circuits," *ACM Transactions on Design Automation of Electronic Systems*, vol. 13, no. 1, pp. 1–35, 2008.

[38] N. Mohyuddin, E. Pakbaznia, and M. Pedram, "Probabilistic error propagation in a logic circuit using the Boolean difference calculus," in *Advanced Techniques in Logic Synthesis, Optimizations and Applications*, K. Gulati, Ed., Springer, New York, NY, USA, 2011.

[39] C. Bolchini, A. Miele, and C. Sandionigi, "A novel design methodology for implementing reliability-aware systems on SRAM-based FPGAs," *IEEE Transactions on Computers*, vol. 60, no. 12, pp. 1744–1758, 2011.

[40] J. Anwer and M. Platzner, "Evaluating fault-tolerance of redundant FPGA structures using Boolean difference calculus," *Microprocessors and Microsystems*, vol. 52, pp. 160–172, 2017.

[41] J. Anwer and M. Platzner, "Analytic reliability evaluation for fault-tolerant circuit structures on FPGAs," in *Proceedings of the 17th IEEE Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pp. 177–184, Amsterdam, Netherlands, October 2014.

[42] J. Anwer and M. Platzner, "Boolean difference based reliability evaluation of fault- tolerant circuit structures on FPGAs," in *Proceedings of the 2016 Euromicro Conference on Digital System Design (DSD)*, pp. 1–8, Limassol, Cyprus, August 2016.

[43] Y. Cao, *Pareto Front*, 2007, http://www.mathworks.de/matlabcentral/fileexchange/17251-pareto-front.

[44] E. Lubbers and M. Platzner, "ReconOS: an RTOS supporting hard-and software threads," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pp. 441–446, Amsterdam, Netherlands, August 2007.

[45] E. Lubbers and M. Platzner, "Cooperative multithreading in dynamically reconfigurable systems," in *Proceedings of the IEEE International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4, IEEE, Prague, Czech Republic, August 2009.

[46] E. Lubbers and M. Platzner, "ReconOS: multithreaded programming for reconfigurable computers," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 9, no. 1, 2009.

[47] Benchmarks homepage, 2007, http://www.pld.ttu.ee/maksim/bench-marks.

[48] J. Anwer, M. Platzner, and S. Meisner, "FPGA redundancy configurations: an automated design space exploration," in *Proceedings of the 2014 IEEE International Parallel Distributed Processing Symposium Workshops*, pp. 275–280, Phoenix, AZ, USA, May 2014.

[49] J. Anwer, S. Meisner, and M. Platzner, "Dynamic reliability management: reconfiguring reliability-levels of hardware designs at runtime," in *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pp. 1–6, Cancun, Mexico, December 2013.

[50] J. Anwer, *Dynamic reliability management*, Paderborn University, Paderborn, Germany, PhD dissertation, 2017.

[51] S. Dong, X. Wang, and X. Wang, "A novel high-speed parallel scheme for data sorting algorithm based on FPGA," in *Proceedings of the 2nd International Congress on Image and Signal Processing (CISP)*, pp. 1–4, Tianjin, China, October 2009.

[52] D. Mihhailov, V. Sklyarov, I. Skliarova, and A. Sudnitson, "Optimization of FPGA-based circuits for recursive data sorting," in *Proceedings of the 2010 12th Biennial Baltic Electronics Conference (BEC)*, pp. 129–132, Tallinn, Estonia, October 2010.

[53] Partial Reconfiguration User Guide, 2012, http://www.xilinx.com/support/documentation/sw_manuals/xilinx14.5/ug702.pdf.

[54] K. Papadimitriou, A. Dollas, and S. Hauck, "Performance of partial reconfiguration in FPGA systems: a survey and a cost model," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 4, no. 4, pp. 36:1–36:24, 2011.

[55] K. Vipin and S. A. Fahmy, "ZyCAP: efficient partial reconfiguration management on the xilinx Zynq," *IEEE Embedded Systems Letters*, vol. 6, no. 3, pp. 41–44, 2014.

[56] S. Belkacemi, K. Benkrid, D. Crookes, and A. Benkrid, "Design and implementation of a high performance matrix multiplier core for xilinx virtex FPGAs," in *Proceedings of the 2003 IEEE International Workshop on Computer Architectures for Machine Perception*, pp. 4–159, New Orleans, USA, 2003.

[57] S. Aslan, C. Desmouliers, E. Oruklu, and J. Saniie, "An efficient hardware design tool for scalable matrix multiplication," in *Proceedings of the 2010 53rd IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1262–1265, Seattle, WA, USA, August 2010.

[58] V. Strassen, "Gaussian elimination is not optimal," *Numerische Mathematik*, vol. 13, no. 4, pp. 354–356, 1969.

[59] D. McMurtrey, K. S. Morgan, B. Pratt, and M. J. Wirthlin, *Estimating TMR Reliability on FPGAs Using Markov Models*, Tech. Rep., Brigham Young University, Provo, UT, USA, 2008.