Probabilistic Model Checking Based DAL Analysis to Optimize a Combined TMR-Blind-Scrubbing Mitigation Technique for FPGA-Based Aerospace Applications

Khaza Anuarul Hoque, O. Ait Mohamed
Concordia Univeristy
Montreal, Canada
Email: {k_hoque,ait}ece.concordia.ca

Yvon Savaria
Polytechnique Montréal,
Montreal, Canada
Email: yvon.savaria@polymtl.ca

Claude Thibeault
École de Technologie Supérieure
Montreal, Canada
Email: claude.thibeault@etsmtl.ca

***Abstract*—SRAM-based FPGAs are increasingly popular in the aerospace industry for their field programmability and low cost. However, they suffer from cosmic radiation induced Single Event Upsets (SEUs), commonly known as soft errors. In safety-critical applications, the dependability of the design is a prime concern since failures may have catastrophic consequences. An early analysis of dependability of such safety-critical applications will enable designers to develop a design that meets the high availability and reliability requirements of the DO-254 standard. This paper introduces a novel methodology based on probabilistic model checking, to analyze the dependability properties of safety-critical systems and to suggest required mitigation techniques, such as Triple Modular Redundancy (TMR) or TMR with less frequent scrubs for early design decisions. Starting from a high-level description of a system, a Markov model is constructed from the Control Data Flow Graph (CDFG) expressing the functionality and from failure/mitigation parameters for the targeted FPGAs. Such an exhaustive model captures all the failures and repairs possible in the system within the radiation environment. We present a case study on a benchmark circuit to illustrate the applicability of the proposed approach to demonstrate that a wide range of useful dependability properties can be analyzed using our proposed methodology.**

I. Introduction

Design and verification of highly complex, reliable hardware and software systems have always been quite a challenge. Safety-critical systems such as those are used in avionics, space and medical applications must meet very stringent requirements. A fault in such a system may lead to the loss of life or significant environmental damage. For critical systems, high reliability leads to functional correctness of the design, whereas high availability leads to a very low downtime. In most of the cases, such a system must go through a rigorous certification and quality assurance process. The number and the complexity of electric components in commercial aircrafts have grown dramatically. As a result, it became essential for the Federal Aviation Administration (FAA) to establish a baseline of required design flow steps for airborne component. In 2005, DO-254 [1] was formally recognized as a standard to ensure the highest level of safety in electronic airborne systems. It includes five levels of compliance, commonly known as Design Assurance Levels (DALs). DALs range in severity from A (means that a hardware failure would cause a catastrophic failure of an aircraft) to E (means that a failure would not affect the safety). As expected from the description, meeting a DAL-A level of compliance requires significantly more effort and also greater attention to verification than would DAL-E.

Electronic components are exposed to more intense cosmic rays when flying at high altitude. It has been reported that long-haul aircrafts flying at airplane altitudes experience a neutron-flux roughly 500 times higher than that at ground level in the worst case [2]. For space missions, the rate is much worse [3]. Due to field programmability, absence of non-recurring engineering costs, low manufacturing costs and other advantages, SRAM-based FPGAs are increasingly attractive. Unfortunately, the main disadvantage of these devices is their sensitivity to radiation effects that can cause bit flips in memory elements, and ionisation induced transient faults in semiconductors, commonly known as soft errors or Single-Event Upsets (SEUs) [3], [4]. Therefore, in the aerospace industry, the possibility of cosmic radiation-induced soft error grows dramatically at higher altitudes. An early analysis of availability and reliability impacts of such errors on a design can help designers to develop more reliable and efficient designs that may reduce the overall cost associated with the design effort. Our work aims at achieving these goals.

To deal with SEUs, designers mostly rely on redundancy-based solutions, such as Triple Modular Redundancy (TMR) [5] to obtain high reliability and to configuration scrubbing [6] for fixing soft error effects to obtain high availability. However, frequent scrubbing consumes high power [7] and hence scrub at lower frequency is desired. This paper proposes a means by which formal verification methods can be applied at early design stages to analyze a reconfigurable system in order to validate if the design meets assurance levels compliant with DO-254 and also high-availability requirements. We explore the effects of scrub rate to suggest the lowest possible scrub frequency to meet the availability requirements and also to assess reliability enhancements that can be obtained with a suitably designed architecture leveraging techniques such as TMR. For this purpose, we use *probabilistic model checking*, which is used to verify systems whose behavior are stochastic in nature. It is mainly based on the construction and analysis

978-1-4799-5338-7/14/$31.00 ©2014 IEEE

of a probabilistic model, typically a Markov chain or a Markov process. These models are constructed in an exhaustive manner. Indeed, the models explore all possible states that might occur in a system. Probabilistic model checking can be used to analyze a wide range of reliability and availability properties. In discrete-event simulations, approximate results are generated by averaging results from a large number of random samples. In contrast, probabilistic model checking applies numerical computations to provide accurate results.

To analyze a design at high-level, we start from its Control Data Flow Graph (CDFG) representation [8], obtained from a high-level description of the design expressed using a language such as C++. The methodology described in [9] is used to estimate the resources required to implement the design. Then number of essential bits [10] can be estimated using a library-based approach mentioned in the methodology. With failure rates derived from estimates of the number of essential bits and repair rates derived from the target system characteristics along with fault tolerance strategy (scrub only or scrub with TMR), the possible failures and repairs are then modeled with the PRISM modeling language [11]. Unlike other works where the repair rates are estimated as exponential distributions [12], [13], we model the deterministic repair intervals using the Erlang process [14] for more accuracy. Indeed, nonexponential holding time distributions can be approximated by inserting multiple intermediate states between every main state pairs. Our contributions in this paper can be summarized as follows:

- A novel methodology to analyze reliability and availability at early design stage to verify the DAL requirements for FPGA-based aerospace applications.
- Analytical reliability and availability models of systems with periodic scrub and TMR developed using conventional Markov chains (instead of semi-Markov) to be assessed using a probabilistic model checker.
- A case study showing the effects of scrubbing on availability to guide a designer for choosing the lowest possible scrub rate for low power consumption. The case study also presents a scenario using scrub with TMR ensuring higher reliability.

The remainder of the paper is organized as follows. Section 2 reviews the related works in this area. Section 3 describes the background about soft error effects, soft error mitigation techniques and probabilistic model checking. The proposed methodology and modeling details are discussed in section 4, and in Section 5, we present a case study using our proposed methodology. Section 6 concludes the paper with future research directions.

II. Related Works

For many years, dependability analysis of complex systems has been an active research area in both academia and industry. Also, researchers have put a lot of efforts for hardening SRAM-based FPGA designs to tackle the problem of SEUs. In [15], the authors presented a design flow to implement SEU hardened systems implemented with SRAM-based FPGAs. Three independent strategies were proposed. The proposed strategies are the TMR-based techniques, the TMR coupled with partial reconfiguration and some specific local re-design of the critical portion of the design to overcome TMR failures. In [16], the authors presented a number of possible radiation-induced faults in SRAM-based FPGAs and applicable mitigation methods. However, they did not provide any model for the estimation of the rate at which those errors happen and how to handle them at early stage. Optimal design of TMR systems with more frequent voting strategies has been deeply investigated in [17] and [18]. In the first paper, the authors showed how cleverly inserted voters can improve reliability and their later work showed how partitioned TMR can provide improved reliability by giving protection against distinct single upsets affecting a system at the same time. In [19], the authors presented a methodology to compute the dependability metrics for different fault tolerant architectures. An important class of applications with deterministic maintenance and repair times is discussed in detail in [20]. This paper presented a steady-state analysis of the periodic preventive maintenance problem with general failure and repair time distributions obtained by solving a semi-Markov process.

All the works mentioned above either use semi-Markov models or assume time interval between scrubs to follow an exponential distribution, which is not accurate in real world scenarios. Even though semi-Markov processes have been employed to model deteriorating systems by allowing the holding time distributions to be non-exponential, it is generally assumed that the mathematical formulations of semi-Markov models [21] are so complicated that they are not analytically tractable. Some of the works mentioned above are mostly focused on designing a more robust TMR solution and none of those explored the effect of scrub intervals on the FPGA dependability at early design stage. In our work, we are more interested in using formal verification techniques as they can guarantee exact solutions. We model the periodic scrub using conventional Markov chains. Nonexponential holding time distributions in Markov chains are approximated by inserting multiple intermediate states based on a phase-type distribution. In [22], authors present a methodology for estimation of reliability performance trade-off at early design stage using rescheduling of data flow graphs as a mitigation technique. In this paper, we present a methodology that uses a library-based approach to estimate the reliability and availability of a design to ensure the DO-254 compliance at early design stage using probabilistic model checking. The motivation of the work is also to help designers improve availability with longer scrub intervals to save power. Also reliability assessment at early design stage will help designers to decide if they need to adopt any other mitigation strategy to ensure the required dependability.

III. Background

A. FPGA and Single Event Upsets

FPGAs are configurable logic devices that implement logic circuits with a fabric that includes lookup tables (LUTs), memories and routing resources that connect the LUTs and memories. LUTs are used to implement logic equations, and memories for implementing sequential logic and storage. In a reconfigurable FPGA, the configuration memory is a collection of bits commonly known as a bitstream. Bitstream bits set the values of the LUT, flip-flop and memory initialization values, and states of switches and connection boxes that route

signals through the FPGA. For Virtex devices from Xilinx, the configuration memory is composed of SRAM cells. Those cells are arranged in frames of 32-bit configuration words. In Virtex-5, there are 41 words in each frame [23]. Several interfaces are provided for accessing configuration memory for different purposes. The Joint Test Action Group (JTAG) interface is typically used for initial configuration. The Xilinx-specific SelectMAP interface is used for runtime read-back and reconfiguration. It can be configured for a bus width of 8, 16, or 32 bits. Xilinx provides the Internal Configuration Access Port (ICAP) to expose the SelectMAP interface to user logic. The ICAP eliminates the need for an external runtime-configuration manager by allowing the FPGA to read back and reconfigure itself. The FPGA configuration is stored in volatile SRAMs. Therefore, interaction with high-energy radiated particles that are common in the aerospace environment, such as protons, neutrons, and heavy ions, may corrupt the FPGA configuration. The effects of these particles on electronics are collectively known as Single-Event Effects (SEE) and there are several types of SEE that are relevant to FPGAs. Single-Event Upsets (SEUs) occur when one or more bits in configuration memory changes state due to a radiation event. If only one bit is affected, then it is called a Single-Bit Upset (SBU). If more than one bit is affected, then it is an MBU. The state of the FPGA configuration memory defines the architecture of the application. As a consequence, SEUs in the configuration memory are not only harmful but also could result in catastrophic failure of the design. Many bits in the bitstream that are not employed in a given design do not affect system operation if an upset occurs. A portion of the configuration memory bits that are employed in the design directly affects the system operation if upset occurs, and these critical bits can only be identified by fault-injection techniques. However, for estimation, Xilinx allows generation of a mask file that identifies the essential bits of the design, of which the critical bits are a subset [24].

Altera also offers a variety of SRAM-based FPGAs, such as the Stratix and Cyclone devices [25]. However, Altera FPGAs mostly rely on CRC-based error detection and correction methodology for SEU mitigation [26]. If there was a means of estimating the number of essential bits and of enabling the periodic scrubbing technique, our proposed methodology would apply directly on Altera SRAM-based FPGAs as well.

B. Scrubbing and TMR

For most applications, the FPGA configuration data is loaded upon power-up. For such applications, the desired state of the configuration memory that enables the repair of upset bits through scrubbing will be known. For the implementation of the scrubber, there are mainly two options. The first option is to implement the scrubber as an external device, such as a radiation-hardened microprocessor. The other option is to implement the scrubber internal to the FPGA using the fabric and ICAP [6]. External scrubbing with radiation-hardened parts is reliable. However, as it requires at least one additional processor, it can be expensive in terms of power, size, and cost. Even though internal scrubbing is superior with respect to these constraints, extra care is required for implementing internal scrubbing as the scrubber itself is vulnerable to SEUs. A scrubbing technique is a single algorithm used in the system to mitigate configuration-memory upsets. There are two different type of techniques, specifically detection techniques and correction techniques. Each of these techniques has its own properties with respect to error coding, granularity and redundant data source. A scrubbing strategy is composed of at least one correction technique and optionally, a detection technique. Blind scrubbing is a very popular scrubbing strategy with no detection technique. If at least one detection technique is used in a scrubbing strategy, then it is called read-back scrubbing. In read-back scrubbing, the current state of configuration memory is read back from the device to detect an upset. For this paper, we concentrate on blind scrub, that do not require any detection.

As discussed above, all scrubbing strategies employ at least one technique for correcting the configuration bit upsets. Correction techniques either use data redundancy to recall or calculate the original configuration and then write this configuration to the device. These techniques differ in their coverage of various upset types (e.g., SBU vs. MBU), granularity of correction (e.g., frame vs. device) and correction data source (e.g., off-chip vs. on-chip memory). Depending on the chosen scrubbing strategy, the correction technique may be triggered continuously by a simple timer delay, or by a detection technique. The golden copy correction and error syndrome correction are the two main correction techniques that are widely used for scrubbing. Error syndrome correction is mostly used in read-back scrubbing. In golden copy correction, a trusted golden copy of the original configuration is kept off-chip in non-volatile storage, such as a radiation-hardened PROM, and used to reconfigure the FPGA as needed. Blind scrubbing strategies that employ only golden copy correction are very popular in FPGA-based space platforms because of their effectiveness (they can fix any number of upsets) and simplicity (less implementation complexity). These strategies continuously or periodically reconfigure the FPGA with the golden copy to repair errors quickly after they occur. A known limitation of blind scrub is that the radiation-hardened memories may have limited bandwidth. As a result, the configuration clock often can not be run at its maximum frequency. Scrubbing can be done at a specified rate meaning that there might be a period of time between the moment the upset occurs and the moment when it is repaired. That is why another form of mitigation is required, such as a redundancy-based solution known as TMR [5]. TMR is a technique for enhancing the reliability, in which each module in a circuit or the whole system is triplicated. A majority vote (two out of three) is taken on the TMR outputs to determine the final module output.

C. Probabilistic Model Checking

Probabilistic model checking is based on two main components: the construction and the analysis of a probabilistic model of the system, typically a Markov chain. In this paper, we specifically focus on Continuous-Time Markov chains (CTMCs), that are widely used for reliability and performance analysis. A CTMC involves a set of states S and a transition rate matrix $\mathbf{R} : S \times S \rightarrow \mathbb{R}_{\geq 0}$. The delay before which a transition between states s and s' takes place is specified by the rate $\mathbf{R}(s, s')$. The probability that a transition between the states s and s' might take place within time t is given by $1 - e^{-\mathbf{R}(s,s') \times t}$. Hence, no transitions will take place if $\mathbf{R}(s, s') = 0$. Exponentially distributed delays are appropriate

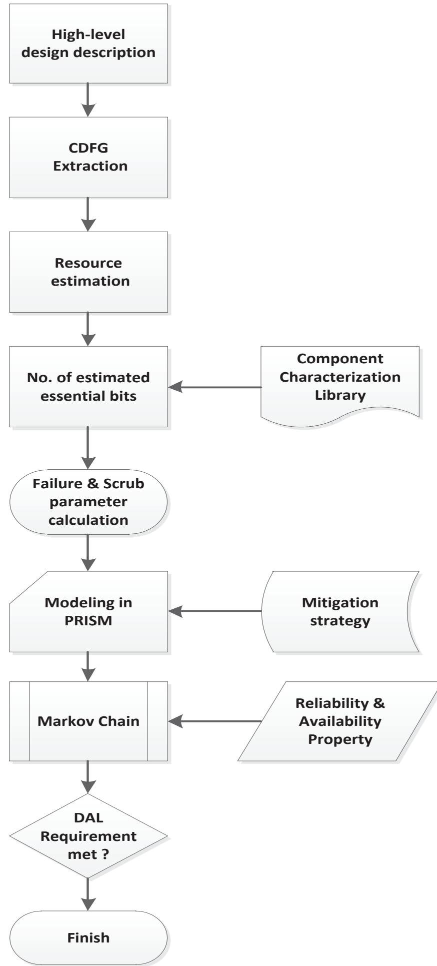

Fig. 1: Proposed methodology

for modelling electronic component lifetimes and inter-arrival times. The model-checking approach to performance and dependability analysis requires both a model of the system under consideration and the desired properties for performance/dependability evaluation. In the case of stochastic modelling, such models are typically CTMCs. The properties are usually expressed in some form of extended temporal logic such as Continuous Stochastic Logic (CSL) [27], a stochastic variant of the well-known Computational Tree Logic (CTL) [28].

D. PRISM modeling and Property Specification

PRISM [29] is a well known tool for the formal modeling and verification of stochastic systems. The current version of the tool supports four types of probabilistic models: discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs), discrete-time Markov decision processes (MDPs) and probabilistic timed automata (PTA).

```
module adder
 s : [0..2] init 0;
 [] (s = 0) -> lambda_1 : (s' = s + 1);
 [] (s = 1) -> lambda_2 : (s' = s + 1);
endmodule
```

Sample PRISM code

Fig. 2: A simple Markov chain to illustrate failure occurrence

A PRISM model is defined as the composition of one or more elementary systems known as `modules`. The state of each module can be represented by a set of finite ranged `variables`. The global state of the model at any point can be determined by evaluating the values of those module (state) variables. A set of guarded commands specifies the behaviour of an individual module. For the case a CTMC, it can be represented as:

```
[] <guard> → <rate> : <action> ;
```

The `guard` is a predicate over the variables of all the modules in the model [30]. The update comprises of `rate` and `action`. A `rate` is an expression which evaluates to a positive real number. The term `action` describes a transition of the module specifying how its variables should be updated. The interpretation of the command is that if the guard is satisfied, then the module is allowed to make the corresponding transition with that associated rate. A very simple command for a module with only one variable z might be:

```
[] <z = 0> → 7.5 : <z' = z + 1> ;
```

which states that, if z is equal to 0, then it will be incremented by one and this action occurs with rate 7.5.

Lets assume that we have a system with error detection capability. The Markov model of such a system as shown in Fig. 2, can be built with three discrete states (S0: fully operational, S1: faulty but with fault undetected, and S2: fault detected, failed) representing the system's status. However, the number of states can be easily changed, depending on the degree of model specificity. The failure rates λ_1 and λ_2 are constant between states. This system can be described using PRISM modeling language as shown in the sample PRISM code.

To analyze a system it is required to specify one or more properties. The property specification language in PRISM is mainly based on temporal logic, which offers an unambiguous means of describing a broad range of properties [31]. The language includes operators from PCTL [32], CSL [27] and its extensions [33]. PRISM emphasizes on quantitative properties. For example, PCTL allows the expression of logical states such as "*what is the probability that the system will eventually fail ?*", expressed as `P =? [F fail]`. Here `fail` is the label refer to the faulty states. Property expressed as `P=? [G<=20.0 !"failed"]` uses a bounded form of the operator `G` to make a query about failure probability for a bounded mission time. This property is satisfied by all paths that spend at least 20 time units in the fault-free states, that is, all paths where the first fault occurs after time 20. Rather than asking PRISM to compute probabilities of interest, one can also assert bounds on the probability of certain sets of paths and verify with PRISM whether these bounds actually hold. The property `P>0.99 [G<=20.0 (state!=3)]` is true in a state `s` of the Markov chain if the set of paths that start from `s` and do not reach state 3 in the first 20 time units

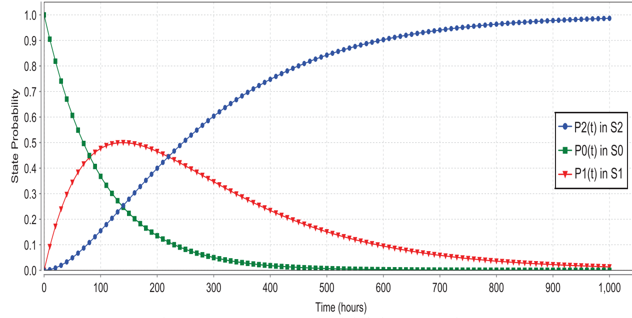

Fig. 3: State probabilities vs time

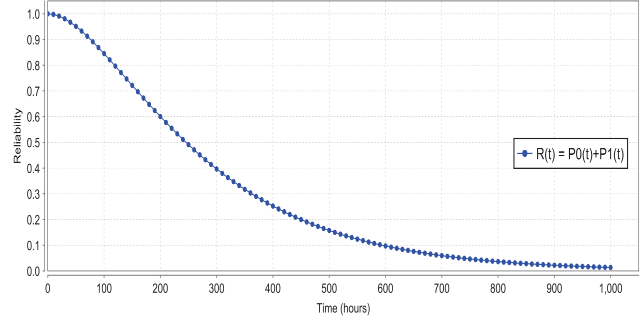

Fig. 4: Reliability vs time

has a probability of at least 0.99. Then this property holds for the system if the initial state satisfies the probability constraint.

IV. Proposed Methodology

In Fig. 1, we present the proposed methodology, which reuses some elements from a methodology proposed in [9], namely the CDFG extraction and the concept of using a characterization library to estimate the number of essential bits (which was created with a different set of tools). We start from the dataflow graph of the application. Different tools such as GAUT [34], SUIF [35] etc. could be used to extract the control dataflow graph from a high-level design description expressed using a language such as C++. Once the CDFG is extracted, a resource estimation algorithm is used to estimate the resource required for the application. Depending on the required resources, using a characterization library, the number of essential bits are estimated for the design. A PRISM model is then built to analyze the design and this model is configured using environmental, target system and mitigation parameters. Different reliability and availability properties are then verified to check if the design meets the reliability and availability requirements. More detail about the different steps of the methodology will be explained in the later parts of the paper.

A. Markov Modeling of Failure and Deterministic Delay:

Markov models are widely used for reliability and availability analysis of complex systems. A Markov model is a directed graph where the nodes represent system states and the arcs represent transition rates between the states. The probability of a state transition for a classic Markov model is assumed to depend only on the current state. This is equivalent to assuming that failure rates are constant and that failure occurrence is a Poisson process. SEU error rates are usually modeled using a Poisson process [36]. This implies that the time between two consecutive events is exponentially distributed.

For the sample Markov chain in Fig. 2, if λ_1 = 0.010 and λ_2 = 0.005, then the corresponding state probabilities, and reliability function of that Markov degradation model can be generated using PRISM as displayed in Fig. 3 and Fig. 4. The reliability function is calculated by summing up all the state probabilities except that of the failed state, S2. In this example, the system is considered tolerant to only one fault. Conversely, a Markov process can also be derived to approximate a given reliability function.

Fig. 5: Markov chain for Erlang process

A system exposed to a harsh radiation environment will eventually fail, requiring repair or replacement. Hence, it is important to have a model that can represent maintenance effects on a system's condition as well as the deterioration process. However, many systems subject to maintenance and degradation may involve state transitions, which depend explicitly on time, or occur discretely. For these reasons, maintenance actions cannot generally be modeled by a simple exponential distribution within the Markov process. For example, a significant repair or periodic inspection time, which may reflect realistic maintenance activities, does not generally follow the exponential distribution. Therefore, we have to develop an approximation methodology to allow the Markov processes to model significant holding times. The concept of a phase-type distribution [37] can be used to approximate a time delay until absorption to one of the states in the Markov chain. It is also known that the Erlang process (i.e., summation of identical exponential distributions as displayed in Fig. 5) minimizes the variance among any phase-type distributions [14]. In other words, non-exponential holding time distributions can be approximated by inserting multiple intermediate states between the two conventional degradation states. In Fig. 5, τ is the total transition interval between S_0 and S_{m+1}, and m is the number of intermediate stages used to approximate it. The rate at which transitions happen is proportional to m to provide a same total transition time. This Erlang process approximation of a constant time delay in a Markov process enables the incorporation of various maintenance activities into the equipment deterioration model.

Fig. 6 illustrates the results of implementing the Markov

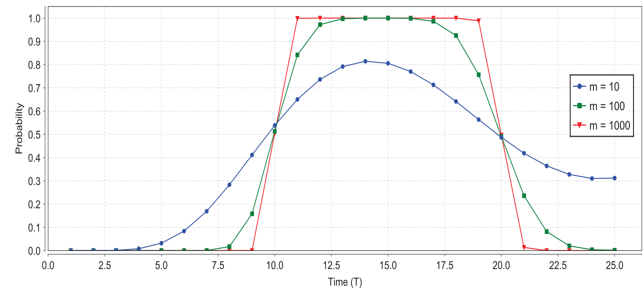

Fig. 6: Deterministic delay modeling with Erlang process

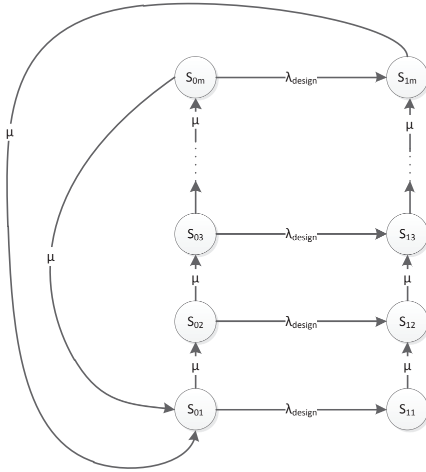

Fig. 7: Markov model of periodic blind scrub using Erlang process

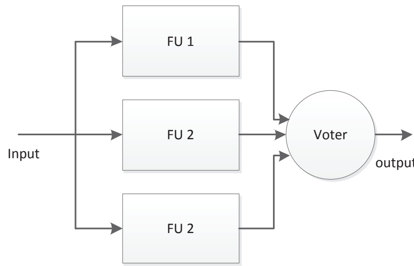

Fig. 8: Conceptual TMR model

chain of Fig. 5 to approximate a constant delay of 10 hours. The figure shows the probability distribution of delay times for different values of *m*. This is how we implement constant time delay for modeling of periodic repair while preserving the Markov property. There is a clear and obvious trade-off here between the accuracy (how close it is to modelling a deterministic time delay) and the resulting expansion in the size of the model.

B. Modeling Scrub and TMR:

A periodic blind scrub mitigation technique can be modeled as a Markov model as illustrated in Fig. 7. The states represent:

S_{0i} : The system is fully operational $(1 < i < m)$;
S_{1i} : The system is faulty with one or more faults $(1 < i < m)$;

In this model, λ_{design} represents the failure rate of the design and μ represent the repair rate, where μ = m/τ , τ = scrub interval. The Markov process is created by stacking the Erlang processes (shown in Fig. 5) on top of the failure model (with 2 main states) shown in Fig. 2. The conceptual block diagram of TMR using device-level redundancy is shown in Fig. 8. Three implementations of the design (known as Functional units or FUs) are used in parallel, and their output goes to a majority voter circuit (the majority voter circuit is assumed to be fault free). The output with the

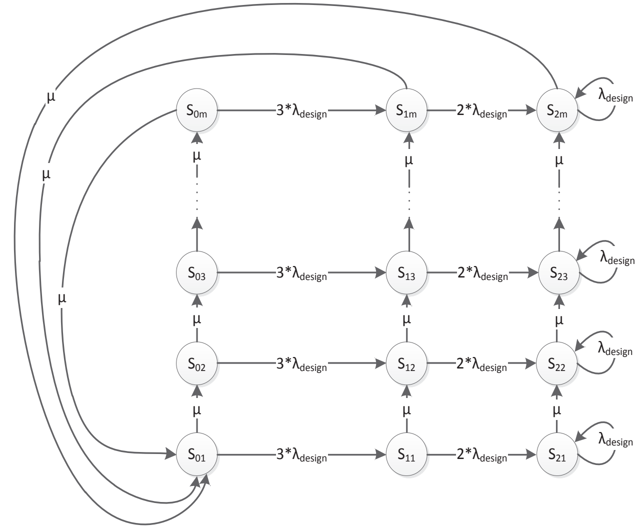

Fig. 9: Markov model of TMR with periodic blind scrub using the Erlang process

majority vote goes to the final output. If more than two FUs encounter errors, then the TMR strategy may fail, since the erroneous output can propagate. The Markov model of a system implementing both TMR and scrub is shown in Fig. 9. This is an extension of the previous scrub only model. The states represent:

S_{0i} : The system is fully operational, e.g. All 3 FUs are fault free. $(1 < i < m)$;
S_{1i} : One of the FUs has encountered at least one SEU and thus the system is under fault, but the output is not erroneous. The system is in a degraded mode $(1 < i < m)$;
S_{2i} : Two of the FUs are faulty, caused by one or more SEUs in each FU, and hence the output may be erroneous. The system has entered a possibly failed mode and will stay there until the next scrub arrives $(1 < i < m)$;

C. Markov Model Parameters:

Once the Markov model is built, we need to populate the model for further analysis. Three different types of parameters are required [38], namely (1) Environmental parameters, (2) Target system parameters and (3) Mitigation parameters. Some of these parameters, such as mitigation parameters, can be varied freely to achieve optimal availability. On the other hand, target system parameters and environmental parameters are fixed for a given target system and environment.

1) Environmental parameters: The key environmental parameter is the upset rate λ experienced in various orbits of interest. As the observed upset rates are dependent on device process technology and architecture, so this parameter may be different for each device family. We use CREME96 [39] with radiation cross sections from [40] to find per-bit upset rate λ_{bit} for Xilinx Vitex-5 in ISS LEO orbit, which is 2.63×10^{-12} SEUs/bit/sec. The failure rate for this system can be calculated as follows:

$$\lambda_{design} = \lambda_{bit} \times Number\ of\ critical\ bits$$

TABLE I: Model construction time and statistics

m	No. of states		No. of transitions		Time (s)	
	Scrub only	Scrub & TMR	Scrub only	Scrub & TMR	Scrub only	Scrub & TMR
50	100	150	150	300	0.006	0.006
100	200	300	300	600	0.008	0.009
200	400	600	600	1200	0.015	0.018

2) Target system parameters: The target system can be defined with three main parameters, namely SelectMAP bus width, B and the configuration clock frequency, f_{clk}. Usually, these parameters are set by the system designer and also limited by the system architecture. They directly impact system availability. We assume a conservative system using a radiation-hardened memory with an 8-bit bus at 33MHz.

3) Mitigation parameters: For a system using periodic scrub, mitigation parameters μ_{design} describes the scrub rate of the system and it can be calculated using the target system parameters. Scrub rate can be determined experimentally, however this rate also can be estimated analytically by using the following equation:

$$\mu_{design} = \frac{B \times f_{clk}}{Total\ configuration\ bits}$$

We use this equation to calculate the repair rate parameter μ in our model, where $\mu = m \times \mu_{design}$.

For the purpose of parameter calculation in our case study, we consider the Xilinx Virtex-5 XC5VLX330 device as the target which has 79,704,832 configuration bits and ISS LEO orbit as the target orbit. Similarly, any other target device or any other orbit can be evaluated using the same methodology.

V. Case Study

For the case study, we consider a 512-tap parallel FIR filter for space application. Using the methodology described in [9], from high-level design description, the data flow graph representation is obtained and the number of essential bits for FPGA implementation is estimated. In our experiments, we consider a worst-case scenario, where all the essential bits are considered critical bits. Fig. 10 shows the block diagram of a 512-tap parallel FIR filter using 32 blocks, each of which embedding a 16-tap FIR filter. The number of essential bits required to implement this design is approximately 5863557 bits. Table I shows the model generation statistics and timings for different values of *m* (number of Erlang steps). For our experiments, we choose the value: *m = 200*.

The experimental results can be divided into two different sections: Analysis and Verification. In the analysis section, we show the use of probabilistic model checking to analyze the system and to evaluate its reliability and availability properties. In the verification section, we show how such analysis can be used to verify that the system meets its DAL and availability requirements.

A. Analysis

The scrub parameter μ is the reciprocal to the time needed to repair an upset once the scrub technique is triggered. From mitigation parameter calculation, we find that μ_{design} = 3.19/sec, which implies that it requires approximately 300ms to repair an upset. One may intuitively conclude that scrubbing

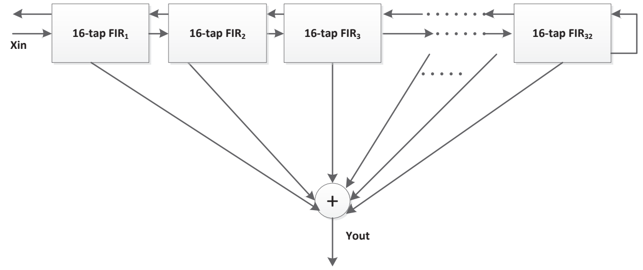

Fig. 10: 512-tap parallel FIR filter

TABLE II: Scrub intervals vs reliability and availability

Scrub Interval (s)	Availability (1 month)	Reliability with scrub, T = 2 hr	Reliability with scrub & TMR, T = 2 hr
0.5	0.99999	0.8658	0.9999
5.0	0.99995	0.8658	0.9999
10.0	0.99989	0.8658	0.9999
100.0	0.99899	0.8658	0.9991
1000.0	0.99001	0.8658	0.9918

continuously is the best solution to ensure the best availability. However, one of the main drawbacks of this technique is its high power consumption, due to the repeated accesses to the large configuration memory of the FPGA [7]. Thus, it is desirable to perform scrubbing with the minimum required frequency.

In Fig. 11, we show the availability for different scrub intervals (τ) for a mission time of T = 300 seconds. In PRISM, this property can be formalized as `R{"up_time"} = ? [C <= T]/T, T = 1 to 300`. Regarding the scrub parameter μ, there are two main things to consider in scrubbing: time when the scrub is triggered and time to repair the bitstreams. For the following experiments, whenever we mention scrub interval, it includes both the time to trigger the scrub and time to repair the bits. We observe that, for τ = 0.5 seconds, the availability is decreasing but stays in the range of five 9s for the whole mission time. On the other hand, for τ = 1 second, the availability drops below five 9s before reaching T = 50 and continues in the range of four 9s. For τ = 1.5 second and 2 seconds, the availability is in the range of four 9s and decreases with time.

In Table II, we show the availability of the system for a mission time of one month for different scrub intervals, τ. We observe that the availability is five 9s for τ = 0.5 second and four 9s for τ = 5 seconds, and for τ = 1000 seconds the availability is only two 9s. The probability that the

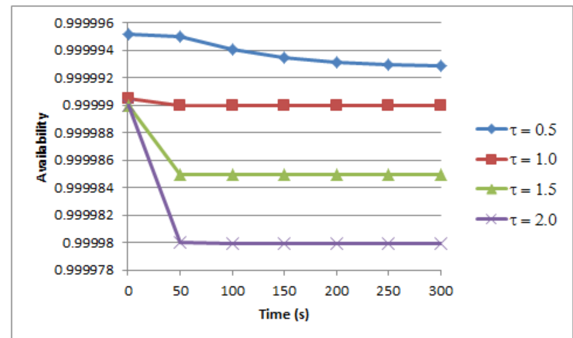

Fig. 11: Availability for T = 300s

TABLE III: Verification of availability requirements

Scrub interval (s)	Availability requirement met ?
0.5	True
1.0	False
1.5	False
2.0	False
2.5	False
3.0	False

system will always be operational (with zero failure events) within the first 2 hours of operation can be formalized in PRISM as : `P = ? [G[0, T] ("operational")], T = 7200`, and the results are shown in Table II. Reliability results show an important observation. That is, periodic scrubbing has no effect on the reliability. This might seem counter intuitive as one might think that the scrub will drastically increase the reliability. In fact, this result reflects the *memoryless* property of exponential distributions. When an SEU occurs, it might corrupt the system, but if the impact has not arrived yet, the observed time can be reset at any time, that means the coming time of the impact will not be delayed by periodic scrubbing. That is the main reason why the periodic scrubbing will not increase reliability. Availability is defined as the ratio of uptime and total runtime (mission time). That is why, for repairable systems, system engineers are more interested in availability analysis. The results also show that even though the reliability is not increased, the availability is significantly improved by the scrubbing. To increase the reliability, the designers need to adopt redundancy-based solutions, such as device-level TMR at a cost of at least 3 times area and power overhead, and the results are shown in column 4. The results show how significant reliability improvements are obtained by adopting the TMR-based solution with scrub. We must mention that in such cases, periodic scrub will have an effect on reliability. The reason is, to reach a failure state, it needs at least 2 FU failures (assuming the voter is error free). So after one FU fails, if the scrub interval is short, the system gets back to "all good" state before the second failure occurs. Otherwise, if the wait time is longer for the second scrub, then it might reach the failure state before the scrub is triggered. For example, for a scrub interval of 1000 seconds, we observe that the reliability drops to 0.99.

Such analysis at an early design stage can help a designer to adopt a proper mitigation strategy considering reliability, availability requirement, power and area constraints. For high availability applications, scrubbing alone might fulfill the requirements. Using our methodology, the designer can choose a proper scrub interval to minimize the power requirements for such applications. On the other hand, for reliability oriented applications, a redundancy-based solution is a must. However, it comes with an extra power and area overhead. The area and power overhead can be reduced by applying the other types of TMRs, such as selective TMR [41], however device-level TMR ensures the best reliability [42].

B. Verification

Depending on the mission goal, a spacecraft can have different levels of reliability and availability requirements. For example, a GPS or communication satellite will need a better availability compared to an earth observation satellite. A Mars rover's landing module will require better reliability than the module responsible for taking photos periodically

TABLE IV: Verification of reliability requirements

Scrub interval (seconds)	DAL-A met (scrub only) (B = 0.0001)	DAL-A met (scrub only) (B = 0.001)	DAL-A met (scrub & TMR) (B = 0.001)
0.5	True	False	True
1.0	True	False	True
1.5	True	False	True
2.0	True	False	True
2.5	True	False	True
3.0	True	False	True

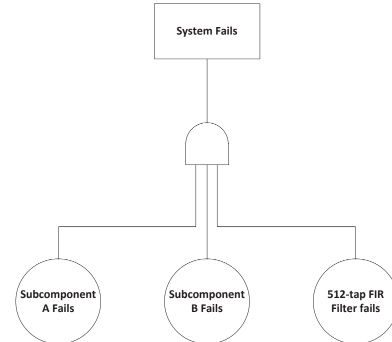

Fig. 12: Fault tree of the system

to send back to earth. FIR filters are widely used in image processing applications. Autonomous landing systems [43] widely use image processing algorithms to find a suitable landing place, hence it is obvious that such a system will require high reliability as the choice of wrong landing place will cause the mission failure. We consider the use of the 512-FIR filter for two different applications: in GPS satellite and lunar landing module. Communication satellites require high availability, usually in the range of five 9s. To be independent of mission time, we calculate the steady state availability to ensure that the availability will maintain that level of service in a long run. So we need to find, *"for a given scrub rate, does the system meet the requirement for five 9s ?"*. Such property can be formalized in PRISM as $S >= 0.99999[\text{"}operational\text{"}]$. To find the appropriate scrub rate, we vary the scrub interval from 0.5 to 3 seconds and verify if the requirement is met. The results are shown in Table III. As we can see from the results, the only scrub interval that meets the requirement is τ = 0.5 second or less.

To verify the DAL requirement, we consider a hypothetical system that uses the FIR filter as a subcomponent in the image processing component of a lunar landing module. As such image analysis needs to be done in real time and a fault during the landing operation will cause a catastrophic failure, so the overall failure probability of such a system must be less than 10^{-9} (DAL-A design) according to Item Design Assurance Level (IDAL) standard. From the fault tree [44] in Fig. 12, we observe that if any of the subcomponents, namely A, B and the FIR filter fails, the module fails. If subcomponent A and subcomponent B both have a failure probability of 0.0001, then to avoid a catastrophic failure the failure probability of the FIR filter must be less than 0.1. It takes around 4 days to reach lunar orbit and touchdown to the moon requires around 11 minutes starting from the descending time. So to be safe, we consider 20 minutes, hence we find out *"For a given scrub interval, what is probability that the FIR filter will fail in last 20 minutes of the flight ? "*. Such property can be formalized in PRISM as

`P < 0.1 [F [344400,345600] ("failure")]`. We also evaluate another case, where the failure probability of subcomponent B is 0.001. For this experiment, a system with only scrub and a system with both TMR and scrub, are evaluated while varying the scrub rates (as scrub rate affects the system using TMR with scrub). From the result shown in Table IV, we observe that, all the scrub intervals from 1 to 3 seconds meet the DAL-A requirement, whereas in the latter case, if B = 0.001, then it fails to satisfy the DAL-A requirement. However, if TMR is adopted with scrub, even with B = 0.001, the system meets the DAL-A requirement for all the scrub intervals. Such results can help a designer obtain the maximum scrub interval (in this case 3 seconds) to save power.

VI. Conclusion

We presented a novel methodology to analyze the radiation induced errors and some popular traditional mitigation techniques. The proposed method allows evaluating the dependability metrics of specific designs. Such analysis at an early design stage can help a designer to build a robust design for harsh radioactive environments, such as in outer space, reducing the overall design effort, and it may also reduce the associated cost. Using a FIR filter case study, we also showed how such an analysis using probabilistic model checking can be used to verify the high-availability requirements and the design assurance level compliance at high-level. Modeling results showed how an appropriate scrub interval (slowest scrub rate) can be found to save power while meeting the dependability requirements. Future works include automation of the process to generate the PRISM code for a given mitigation strategy and to analyze designs in the presence of other kinds of possible failures due to SEUs, such as aging, electromigration, hot electron effects, Negative-Bias Temperature Instability (NBTI) and Single-Event Functional Interrupts (SEFI). Also, inclusion of a read-back repair technique in the model is part of our future works.

Acknowledgments

This research work is a part of the AVIO-403 project financially supported by the Consortium for Research and Innovation in Aerospace in Quebec (CRIAQ), Fonds de Recherche du Québec - Nature et Technologies (FRQNT) and the Natural Sciences and Engineering Research Council of Canada (NSERC). The authors would also like to thank Bombardier Aerospace, MDA Space Missions, Regroupement Stratǵique en Microsystèmes du Québec (ReSMiQ) and the Canadian Space Agency (CSA) for their technical guidance and financial support.

References

[1] P. S. Miner, V. A. Carreño, M. Malekpour, and W. Torres, "A case-study application of RTCA DO-254: design assurance guidance for airborne electronic hardware," in *Digital Avionics Systems Conference. Proceedings. DASC. The 19th*, vol. 1. IEEE, 2000, pp. 1A1–1.

[2] C. Hu and S. Zain, "NSEU mitigation in avionics applications (XAPP1073 v1.0)," October 2011.

[3] A. Lesea, "Continuing experiments of atmospheric neutron effects on deep submicron integrated circuits (WP286 v1.1)," October 2011.

[4] P. Adell, G. Allen, G. Swift, and S. McClure, "Assessing and mitigating radiation effects in Xilinx SRAM FPGAs," in *Radiation and Its Effects on Components and Systems (RADECS), 2008 European Conference on*, 2008, pp. 418–424.

[5] C. Carmichael, "Triple module redundancy design techniques for Virtex FPGAs (XAPP197 v1.0.1)," Xilinx Corporation, 2006.

[6] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K. LaBel, M. Friendlich, H. Kim, and A. Phan, "Effectiveness of internal versus external SEU scrubbing mitigation strategies in a Xilinx FPGA: Design, test, and analysis," *Nuclear Science, IEEE Transactions on*, vol. 55, no. 4, pp. 2259–2266, 2008.

[7] G. Nazar, L. Santos, and L. Carro, "Scrubbing unit repositioning for fast error repair in fpgas," in *Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2013 International Conference on*, Sept 2013, pp. 1–10.

[8] K. Kavi, B. Buckles, and U. N. Bhat, "A formal definition of data flow graph models," *Computers, IEEE Transactions on*, vol. C-35, no. 11, pp. 940–948, 1986.

[9] C. Thibeault, Y. Hariri, S. R. Hasan, C. Hobeika, Y. Savaria, Y. Audet, and F. Z. Tazi, "A library-based early soft error sensitivity analysis technique for SRAM-based FPGA design," *J. Electronic Testing*, vol. 29, no. 4, pp. 457–471, 2013.

[10] K. Chapman, "Virtex-5 SEU critical bit information: Extending the cabability of the Virtex-5 SEU controller," Xilinx Corporation, 2010.

[11] PRISM, http://www.prismmodelchecker.org.

[12] D. Chen and K. S. Trivedi, "Closed-form analytical results for condition-based maintenance," *Reliability Engineering & System Safety*, vol. 76, no. 1, pp. 43–51, 2002.

[13] L. M. Maillart, "Maintenance policies for systems with condition monitoring and obvious failures," *IIE Transactions*, vol. 38, no. 6, pp. 463–475, 2006.

[14] A. David and S. Larry, "The least variable phase type distribution is erlang," *Stochastic Models*, vol. 3, no. 3, pp. 467–473, 1987.

[15] C. Bolchini, A. Miele, C. Sandionigi, N. Battezzati, L. Sterpone, and M. Violante, "An integrated flow for the design of hardened circuits on SRAM-based FPGAs," in *Test Symposium (ETS), 2010 15th IEEE European*. IEEE, 2010, pp. 214–219.

[16] H. Quinn, P. S. Graham, K. Morgan, J. Krone, M. P. Caffrey, and M. J. Wirthlin, "An introduction to radiation-induced failure modes and related mitigation methods for Xilinx SRAM FPGAs." in *ERSA*, 2008, pp. 139–145.

[17] F. L. Kastensmidt, L. Sterpone, L. Carro, and M. S. Reorda, "On the optimal design of triple modular redundancy logic for SRAM-based FPGAs," in *Proceedings of the conference on Design, Automation and Test in Europe-Volume 2*. IEEE Computer Society, 2005, pp. 1290–1295.

[18] B. H. Pratt, M. P. Caffrey, D. Gibelyou, P. S. Graham, K. Morgan, and M. J. Wirthlin, "Tmr with more frequent voting for improved FPGA reliability." in *ERSA*, 2008, pp. 153–158.

[19] J. Kastil, M. Straka, L. Miculka, and Z. Kotasek, "Dependability analysis of fault tolerant systems based on partial dynamic reconfiguration implemented into FPGA," in *Digital System Design (DSD), 2012 15th Euromicro Conference on*. IEEE, 2012, pp. 250–257.

[20] D. Chen and K. S. Trivedi, "Analysis of periodic preventive maintenance with general system failure distribution," in *Dependable Computing, 2001. Proceedings. 2001 Pacific Rim International Symposium on*. IEEE, 2001, pp. 103–107.

[21] T. K. Das, A. Gosavi, S. Mahadevan, and N. Marchalleck, "Solving semi-markov decision problems using average reward reinforcement learning," *Management Science*, vol. 45, no. 4, pp. 560–574, 1999.

[22] K. A. Hoque, O. Ait Mohamed, Y. Savaria, and C. Thibeault, "Early analysis of soft error effects for aerospace applications using probabilistic model checking," in *Formal Techniques for Safety-Critical Systems*, ser. Communications in Computer and Information Science. Springer International Publishing, 2014, vol. 419, pp. 54–70.

[23] "Virtex-5 FPGA configuration user guide," Xilinx UG191, 2012.

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

[25] "Configuration Devices for SRAM-Based LUT Devices," Altera Corporation, 2012.

[26] "Test Methodology of Error Detection and Recovery using CRC in Altera FPGA Devices," Altera Corporation, 2014.

[27] C. Baier, J.-P. Katoen, and H. Hermanns, "Approximate symbolic model checking of continuous-time markov chains (extended abstract)," 1999.

[28] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems*, vol. 8, pp. 244–263, 1986.

[29] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: verification of probabilistic real-time systems," in *Computer aided verification*. Springer, 2011, pp. 585–591.

[30] ——, "Controller dependability analysis by probabilistic model checking," *Control Engineering Practice*, vol. 15, no. 11, pp. 1427–1434, 2007.

[31] ——, "PRISM: Probabilistic model checking for performance and reliability analysis," *SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, pp. 40–45, 2009.

[32] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Aspects of Computing*, vol. 6, no. 5, pp. 512–535, 1994. [Online]. Available: http://dx.doi.org/10.1007/BF01211866

[33] B. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen, and C. Baier, "Model checking performability properties." Press, 2002, pp. 103–112.

[34] P. Coussy, C. Chavet, P. Bomel, D. Heller, E. Senn, and E. Martin, "GAUT: A high-level synthesis tool for dsp applications," in *High-Level Synthesis*, P. Coussy and A. Morawiec, Eds. Springer Netherlands, 2008, pp. 147–169.

[35] G. A. et al., "The SUIF program representation," http://suif.stanford.edu/suif/suif2/index.html, January 2010.

[36] B. Bridgford, C. Carmichael, and C. W. Tseng, "Single-Event Upset mitigation selection guide," *Xilinx Application Note, XAPP987 (v1. 0)*, 2008.

[37] T. Osogami and M. Harchol-Balter, "Closed form solutions for mapping general distributions to quasi-minimal ph distributions," *Performance Evaluation*, vol. 63, no. 6, pp. 524–552, 2006.

[38] Q. Martin and A. George, "Scrubbing Optimization via Availability Prediction (SOAP) for reconfigurable space computing," in *High Performance Extreme Computing (HPEC), 2012 IEEE Conference on*, Sept 2012, pp. 1–6.

[39] A. Tylka, J. Adams, P. Boberg, B. Brownstein, W. Dietrich, E. Flueckiger, E. Petersen, M. Shea, D. Smart, and E. Smith, "Creme96: A revision of the cosmic ray effects on micro-electronics code," *Nuclear Science, IEEE Transactions on*, vol. 44, no. 6, pp. 2150–2160, Dec 1997.

[40] H. Quinn, K. Morgan, P. Graham, J. Krone, and M. Caffrey, "Static proton and heavy ion testing of the xilinx virtex-5 device," in *Radiation Effects Data Workshop, 2007 IEEE*, July 2007, pp. 177–184.

[41] P. Samudrala, J. Ramos, and S. Katkoori, "Selective triple modular redundancy (stmr) based single-event upset (seu) tolerant synthesis for fpgas," *Nuclear Science, IEEE Transactions on*, vol. 51, no. 5, pp. 2957–2969, Oct 2004.

[42] S. Habinc, "Functional Triple Modular Redundancy (FTMR). VHDL design methodology for redundancy in combinatorial and sequential logic," *Gaisler Research, Design and Assessment Report (Version 0.2)*, 2002.

[43] D. Meng, C. Yun-feng, W. Qing-xian, and Z. Zhen, "Image processing in optical guidance for autonomous landing of lunar probe," in *Intelligent Unmanned Systems: Theory and Applications*, ser. Studies in Computational Intelligence, A. Budiyono, B. Riyanto, and E. Joelianto, Eds. Springer Berlin Heidelberg, 2009, vol. 192, pp. 1–10.

[44] J. Bechta Dugan, S. J. Bavuso, and M. A. Boyd, "Dynamic fault-tree models for fault-tolerant computer systems," *Reliability, IEEE Transactions on*, vol. 41, no. 3, pp. 363–377, 1992.