

# Probabilistic High-Level Estimation of Vulnerability and Fault Mitigation of Critical Systems Using Fault-Mitigation Trees (FMTs)

Marwan Ammar, Ghaith Bany Hamad, Otmane Ait Mohamed  
Electrical and Computer Engineering Dept., Concordia University  
Montreal, Canada  
e-mails: {m\_amma,ghaith.bany-hamad,ait}@encs.concordia.ca

**Abstract**—The development of safety-critical systems is a rather challenging task, especially due to the cost and complexity associated with this endeavor. For this reason, early fault assessment is a key element towards minimizing vulnerability at the design stage of development. Existing early analysis techniques are often unable to conduct a comprehensive and exhaustive analysis on complex redundant architectures, which may lead to less than optimal risk evaluation. This paper seeks to address some of these issues by proposing a high-level analysis methodology based on probabilistic model checking. This analysis is done by introducing new probabilistic models for repairable fault trees described in the Continuous-Time Markov Decision Process formalism. The models include repairable components and redundancy partitioning to evaluate fault vulnerability across different implementations of the system. The presented approach is very scalable and results demonstrate that the proposed analysis is as reliable as physical FPGA testing, in some scenarios.

**Index Terms**—Fault-tree, mitigation, model-checking, self-repair, redundancy, TMR

## I. INTRODUCTION

High-level techniques for the early analysis of cyberphysical systems have progressively increased in importance as systems grow in complexity, since the costs associated with detecting and correcting vulnerabilities grow exponentially with each subsequent design phase. Fault-mitigation techniques, such as self-repair and triple modular redundancy (TMR) are commonly used for fault tolerance. There exists a rich and extensive literature on the analysis of the impact of these fault-mitigation techniques focused on simulation and FPGA-based approaches. For example, in works such as [1, 2] different mitigation architectures are proposed in the form of built-in self-test (BIST) architectures. These BIST techniques are often implemented in FPGA boards or in design environment softwares (such as Cadence Virtuoso) and are often implemented using different FPGA scrubbing subroutines while performing analyses over emulation runs or statistical simulations in order to find the optimal setup to increase system performance. The work in [3] uses an FPGA implementation to study several different mitigation techniques, including TMR, self-repair,

and combinations of both. These techniques, however, tend to be very time consuming and expensive and their accuracy is arguable, since they can only test for a small subset of the possible state-space.

A good alternative to circumvent these weaknesses lies in the use of formal methods to conduct fault mitigation analysis. In [4, 5], redundant systems are modeled and analyzed through Satisfiability Modulo Theories (SMTs) giving an estimation of the reliability gain across different TMR configurations. These approaches present new problems due to their intrinsic modeling approach, namely the amount of redundancy required to perform the analysis. In [6], fault trees (FTs) have been used to model different systems at high-level of abstraction, and probabilistic model checking (PMC) analyses have been performed to evaluate the impact of different types of TMR to the system reliability. This technique shows that by using FTs to model the system, it is possible to perform exhaustive PMC analysis of complex systems without incurring into state-explosion.

This paper proposes an extension to the work previously reported in [6], by proposing a new technique for fault mitigation analysis that can assess the impact of self-repair routines and TMR may have in a system. This is done by introducing a new probabilistic modeling of FT events and gates, which consist in the creation of a library of behavioral probabilistic automata (PAs) that can be used to compose a model of the system under test (SUT) as Continuous-Time Markov Process (CTMDP). Thereafter, an iterative analysis of the CTMDP model is conducted, in order to estimate the impact that the different mitigation techniques may have in the system and its components. Finally, a report is generated, highlighting system vulnerabilities as well as the best mitigation policies that may be applied to the SUT.

The proposed methodology is experimentally evaluated on different system architectures, including the LEON3FT core processor and the Hermes cubesat HSCOM subsystem. The proposed analysis can estimate the vulnerability of different systems, report the vulnerability of each individual component in the system under test, and evaluate the impact of various different fault mitigation policies on those systems. Our exper-

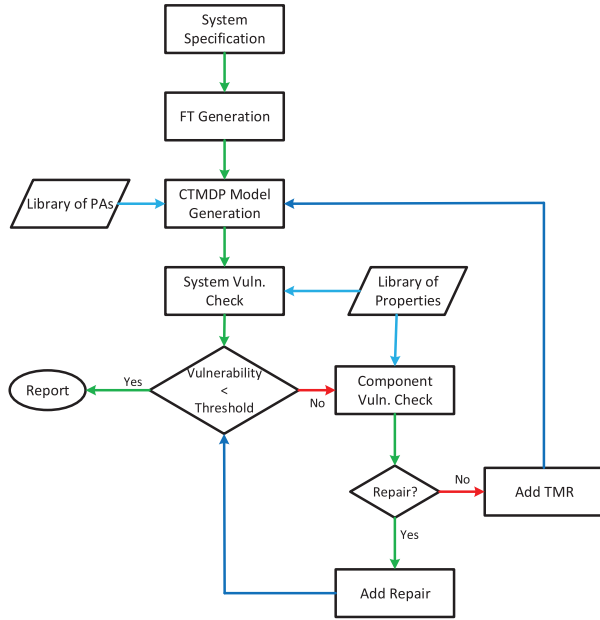


Fig. 1: Main Steps of the Proposed Methodology

iments demonstrate that the proposed methodology is scalable, being capable of handling even very large designs such as the LEON3FT core.

The rest of this paper is structured as follows. In Section II, we present the proposed methodology, with emphasis to the self-repair FT modeling approach, in subsection II-A, and to the TMR modeling approach, in subsection II-B. In Section III, we describe the experiment environment and the results of the experiments performed on the LEON3FT core and on the Hermes HSCOM are reported in Subsection III-A and Subsection III-B, respectively. Finally, in Section IV, we draw some conclusions and discuss future work.

## II. PROPOSED METHODOLOGY

In this section, the proposed probabilistic fault tree analysis of redundant repairable architectures is introduced. The flow of the proposed analysis methodology is shown in Fig. 1. Starting from a system specification description, the fault tree of the system under test (SUT) is derived through the use of the analytical approach known as the Behaviour-Based Method [7]. The failure rate of each component can be estimated from either the system-level specification or from previous experiments conducted on similar hardware. Subsequently, a formal CTMDP model of the system's FT is obtained automatically through the parallel composition of the PAs of the FT events and gates, in the tool PRISM model checker. These PAs are included in a PRISM language library and can be instantiated as needed. Following this, the obtained CTMDP model is exhaustively analyzed with probabilistic model checking (PMC), to provide an estimation of the system's failure rate. If the estimated criticality falls below the acceptable threshold for the SUT, the results are simply

reported. However, if the computed criticality is above the acceptable threshold, then the model is altered to include fault mitigation. When the computed system criticality is above the acceptable threshold, the first step is to perform a detailed vulnerability check of all components with PMC. This is done by isolating each component from the rest of the system and testing them individually. The individual component that is deemed the bigger contributor for the system's vulnerability is then selected. In a first moment, that component is enhanced with self-repair logic, and the vulnerability of the system is reassessed. If the altered component is still critical, then TMR is also applied to it. For each critical component, the fault mitigation tests are incremental and are repeated until the criticality of the component is reduced below the acceptable threshold. This cycle is repeated until the computed system vulnerability falls within the specified threshold. The results of this analysis are reported in the form of proposed improvements to the system design. In the next subsections, the proposed modeling for FTs and for TMRs is discussed in detail.

### A. Modeling of Fault-Mitigation Trees

Fault trees define a set of relations between events, at high-level of abstraction, that may lead to a system failure. Our approach proposes probabilistic models to represent the events and the relationships between events (FT gates). It has been demonstrated in our previous work that the analysis of the vulnerability to SEUs using Markov Decision Process (MDP) [8] and CTMC [6, 9] models can offer accurate results in some cases. MDP models can capture the behavior of SEU events since they permit both probabilistic and non-deterministic choices. However, MDP models cannot be used to evaluate system vulnerability over time, since MDPs do not take into account the influence of the transition time between the states. On the other hand, CTMC models are not able to capture the non-deterministic aspect inherent to SEUs. To address these limitations, as stated earlier, this paper proposes the modeling, fault injection, and analysis using the Continuous-Time Markov Decision Process (CTMDP) formalism. In comparison to MDPs, CTMDPs are able to better model the decision-making process for a system that has continuous dynamics, such as the incidence of random events over time. The general probabilistic model (automaton) of a FT gate is expressed following the definition of a CTMDP model, which can be defined as a tuple  $\{S, S_0, (A(i), i \in S), q(j|i, a), r(i, a)\}$ , where the state space  $S$  is a finite set of fully observable states of the system,  $S_0$  is the initial state in  $S$ ,  $A(i)$  denotes a family of measurable subsets of actions applicable in  $i \in S$ ,  $q(j|i, a)$  is the transition rate function of state  $j$  after performing action  $a \in A(i)$  in state  $i$ , which can satisfy  $q(j|i, a) \geq 0$  and  $i \neq j$ , and  $r(i, a) \in R$  is a reward function such that  $r(i, a)$  is the immediate reward for being in state  $i$  with action  $a$ .

This definition is used to propose a modeling approach for the analysis of different mitigation techniques applied to the FTs. We refer to this approach as Fault-Mitigation Trees (FMTs). FMTs are a combination of conventional FT

gates [10] with newly proposed repairable-events that mimic the behavior of self-repair in the system, resulting in a configurable mitigation architecture that can be easily applied and evaluated in any FT. The proposed schematic for an FMT event is exemplified in Fig. 2, which shows a graphical representation for the proposed repairable events connected to a FT gate. In this work, it is assumed that the SUTs are implemented in FPGAs. Therefore, self-repair events are modeled as data scrubbing routines, which are background tasks that periodically inspect memory/storage units. If an error is found by a scrubbing routine, a repair attempt is performed using redundant data in the form of different checksums or copies of the data.

An example of the PA of a repairable FMT gate with its events is given in Fig. 3. We start by the proposed PA for a repairable event, shown in Fig. 3(a). In the figure, state  $S_0$  represents the absence of fault. The automaton may transition to state  $S_1$ , if the fault rate  $\lambda x$  is greater than the scrubbing rate  $SR$  (i.e., the rate in which the self-repair routine is executed), representing the occurrence of a fault. From state  $S_1$ , the automaton may transition to state  $S_{-s}$  if a successful repair operation occurs, or to state  $S_{-f}$  if the repair operation fails. Alternatively, from state  $S_0$ , if the scrubbing rate is lower than the fault rate, the automaton transitions to state  $S_{-f}$ . State  $S_{-f}$  represents the propagation of the fault (event  $[x]$  in the example), which is transmitted to the FT gate. Continuing the example, Fig. 3(c) shows an example of an FMT AND gate with two input events,  $x$  and  $y$  (Fig. 3(a) and Fig. 3(b), respectively). In this example, the AND gate remains in state  $S_0$  until both events  $[x]$  and  $[y]$  occur, in which case the automaton transitions to state  $S_1$ . From state  $S_1$ , the output event  $[z]$  is generated, transitioning the automaton to state  $S_2$ . The modeling of conventional FT gates follows the approach shown in [6] and the models have been omitted due to space constraints.

### B. Modeling of TMR with FMTs

The assessment of the impact of the optimal redundant architecture configuration is of great importance to the development of safety critical systems. Redundancy is commonly applied to components deemed essential to the system's correct functionality. TMR is a broadly diffused architectural pattern that protects the designs from errors [11, 12, 13, 14]. It consists in triplicating a critical component and feeding the output of each copy to a majority voter. The voter evaluates each component's output and returns the value computed by the majority. Applying TMR to the whole system is very costly and might not be required based on the system's criticality level. Moreover, the efficiency of a TMR is dependent on the



Fig. 2: Example of a Repairable Event schematic

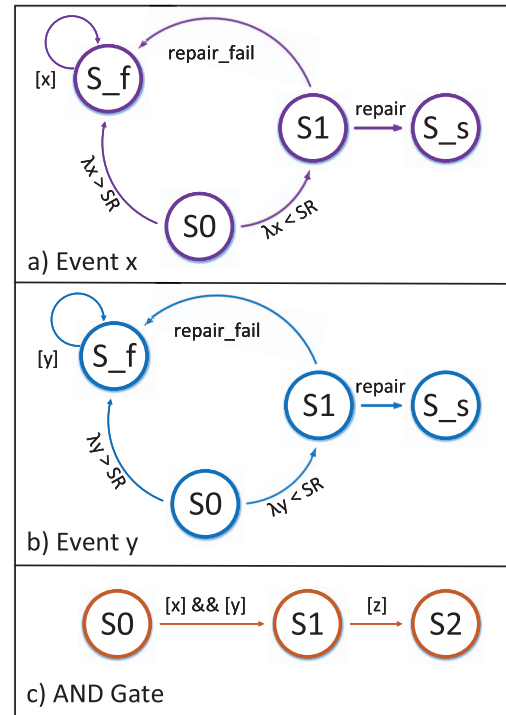


Fig. 3: Example of an AND Gate with Two Repairable Events

reliability of each component and TMR voter, as well as on the TMR configuration. Starting from an evaluation of the most critical components in the system, based on repair policies and presence of TMR, the proposed methodology can assess the optimal TMR partitioning and the best TMR configuration to be used. Following this, the system's failure rate is evaluated for each TMR configuration. This process is repeated for all critical components until the desired system failure rate is achieved.

In our modeling, after identifying the critical component, TMR is applied by triplicating said component in the FT. Each of the redundant copies of the component inherits all the appropriate inputs from all the sub-trees below it, as well as the self-repair logic applied to the original component. The voter logic is added after the self-repair routine takes place. A top-view of the TMR implementation can be seen in Fig. 4.

Through our modeling, component redundancy can be

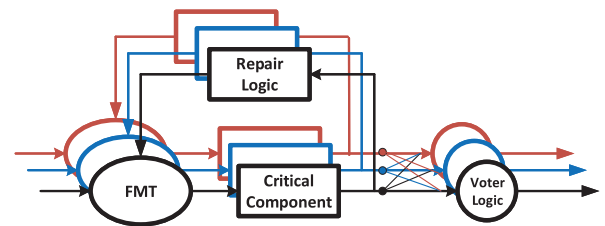
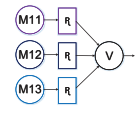
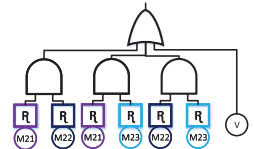
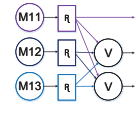
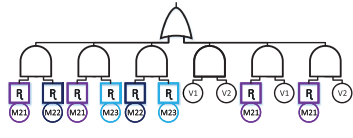
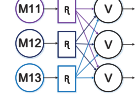
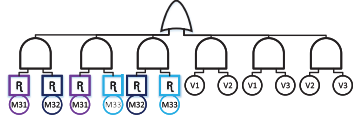


Fig. 4: Representation of Repairable Components with TMR Composition

applied in two ways: 1) the sub-FT of which the critical component is the TLE is triplicated within the system FT and the new analysis is performed. 2) the sub FT of the critical component is extracted from the system FT, triplicated, and analyzed separately. The numerical results of the analysis are then fed back to the system FT, as *external* events. The first way involves increasing the size of the MDP model and, consequentially, the state space. However, the second way reduces the size of the model and increases the efficiency of the verification, while retaining the accuracy. This is achievable through FT modularity [10], combined with the proposed probabilistic modeling, which allows FTs to be divided into sub FTs without losing any properties. Therefore, throughout this work, the second method is applied.

TABLE I: TMR configurations and Respective FTs

a		
b		
c		

In this work, all TMR configurations are analyzed, but for space constraints we only present the most commonly used architectures. Table I shows the main three TMR configurations and their equivalent fault trees. The 1-voter and 2-voters TMR configurations can have different sub-configurations depending on how these TMRs feed their output to the next stage. For instance, a 1-voter TMR can have its output driven by only the voter, or by a combination of the voter output and the copies of the component. The number of outputs of a TMR and their origin are taken into account when building the equivalent FTs. Another modeling option that impacts the analysis of redundant systems is the uniformity of failure rates of the TMR copies of the component. Failure rates can be uniform or non-uniform, meaning that the triplicated components can have either the same or different failure rates. The same principle applies to TMR arrangement, where a 2-voters TMR chain can feature homogeneous or non-homogeneous arrangement of voters at different stages, which can impact fault propagation rates significantly. For example, our results demonstrate that the use of non-homogeneous TMR chains may reduce the failure rate by up to 37% in some cases.

### III. EXPERIMENTAL RESULTS

The proposed methodology is fully implemented within the well known probabilistic model checker PRISM [15], a free, open source probabilistic symbolic model checker, based on the reactive modules formalism [16]. The probabilistic automata of the fault tree gates are modeled as PRISM modules. Each module is composed of a set of commands. A PRISM command is a tuple  $cmd = (act, guard, rate, action)$  following the format  $[<act>] <guard> \rightarrow <rate>: <action>$ , where:

- *act* is an action label used for synchronization of the different levels of an FT;
- *guard* is a predicate over the inputs of the FT gates;
- *rate* is the probability of occurrence of a FT event;
- *action* is a set of  $n$  updates that will translate into transitions in the FT.

Afterwards, a CTMDP model of the system is automatically generated by PRISM. The failure rate of the system is investigated by verifying a set of Probabilistic Computation Tree Logic (PCTL) properties over the CTMDP model. The maximum probability of a system failure can be obtained by verifying the following property:

$$\text{PCTL 1 : } P = ? [(F \text{ TLE} = 1)]$$

The contribution of the failure of a component to the system failure is obtained by verifying the following property:

$$\text{PCTL 2 : } P = ? [(F x = 1) \& (F \text{ TLE} = 1)]$$

Which is interpreted as: *what is the probability that the failure of component  $x$  will eventually lead to a Top Level Event (TLE) failure?*

#### A. Mitigation Analysis of the LEON3FT Processor

The LEON3FT is a fault-tolerant iteration of the standard LEON3 processor, based on the SPARC V8 architecture. It has been designed for operation in harsh and hazardous environments, such as space. This LEON3 variant includes functionality to detect and correct errors in all on-chip RAM memories. The LEON3FT is an open-source 32-bit soft-core processor standard that is technology independent, and its fault tolerance implemented using Error-correcting code (ECC) memory in all on-chip RAM blocks. This paper considers the LEON3FT core processor, as described in [17].

In the conducted experiments, it is assumed that the LEON3FT processor is implemented on an SRAM-based FPGA, and that the processor is exposed to an intermittent source of SEUs. Our experiments demonstrate how SEU mitigation and repair techniques may improve a system's fault-tolerance and prevent SEU failure modes from occurring. As previously discussed, this is done in two steps: 1) TMR masking, and 2) self-repair through applying scrubbing policies to prevent the accumulation of SEUs that might break the TMRs. As discussed in Section II-B, different types of TMR have been evaluated in our experiments. This is achieved by creating

TABLE II: LEON3FT Probability of Failure. Rate of SEU injection is  $1 \times 10^{-2}$

	Probability of Failure				
	Unmitigated	Scrubbing	Scrubbing+TMR1v	Scrubbing+TMR2v	Scrubbing+TMR3v
Top Level	12.4%	8.3%	3.5%	2.9%	2%
d_cache	6.3%	4.1%	1.7%	1.45%	1%
Reg. File.	9.1%	5.9%	2.5%	2.1%	1.45%
Inst. Memory	3.9%	2.6	1.1%	0.9%	0.6%
Avg. Improvement	0%	34%	72%	77%	84%

multiple copies of the model. In each of the copies, a different TMR configuration is applied to the critical component. Then, all the copies of the model are analyzed, and a decision is made based on the criticality achieved by each TMR configuration and the estimated area overhead of that configuration.

On the other hand, the adopted scrubbing policy follows the self-repair mechanism described in Section II-A, where repair is attempted as the SEUs occur. This policy helps to prevent the failure of TMRs, if they exist, by preventing accumulation of SEUs, as long as the rate of scrubbing is higher than the rate of SEU arrival. For this reason, whenever a critical component is identified, the addition of self-repair through scrubbing is always the first fault-mitigation policy applied to the model.

Although it is safe to assume that all fault mitigation techniques lead to improvements in the target design, the impact of each technique on the overall improvement is not clear. For this reason, the goal of the following experiments is not only to identify possible vulnerabilities, but also to estimate the impact of each individual fault mitigation policy. Therefore, the following five fault mitigation policies were created:

- **Unmitigated:** This policy consists in the analysis of the reference LEON3FT processor without any additional mitigation technique.
- **Scrubbing:** This policy enables the immediate scrubbing of SEUs, based on a predefined scrubbing rate.
- **Scrubbing + TMR1v:** This policy incrementally applies scrubbing with the one-voter TMR configuration.
- **Scrubbing + TMR2v:** This policy incrementally applies scrubbing with the two-voter TMR configuration.
- **Scrubbing + TMR3v:** This policy incrementally applies scrubbing with the three-voter TMR configuration.

The results of the experiments are summarized in Table II. The table estimates the probabilities of failure of the most critical internal components in the analyzed architecture. For this experiment, the rate of SEU injection is  $1 \times 10^{-2}$  and the scrubbing rate varies from  $1 \times 10^{-1}$  to  $1 \times 10^{-3}$ . The experiment estimates the vulnerability of the design and quantifies the improvement obtained with different fault mitigation policies. These results may provide valuable insight to system designers, since the choice of the optimal mitigation technique may not be straightforward. For example, Scrubbing+TMR2v provides results that are marginally better than Scrubbing+TMR1v. However, the area overhead resulting from the use of 2-voter TMR may not be viable in some cases. It is also noticeable that there is a great improvement

leap between the scrubbing design and the Scrubbing+TMR designs. However, if the design is able to meet the vulnerability requirements with Scrubbing only, the area overhead and power consumption requirements may be drastically reduced.

### B. Mitigation Analysis of the HERMES CubeSat HSCOM

A similar experiment has been conducted on the high speed communications system (HSCOM) of the Hermes Cubesat project [18]. Hermes is a CubeSat technology demonstration mission of the Colorado Space Grant Consortium at the University of Colorado, USA. The HSCOM component is part of a Microhard MHX-2400 S-band modem with a monopole antenna tuned to 2.4 GHz frequency range and capable of data rates of up to 50 kbit/s. The fault tree of the Hermes HSCOM subsystem is shown in Fig. 5. For simplicity, all the events of this FT are assigned the same failure rate of  $5.0 \times 10^{-2}$ . This is done because real failure rates are unavailable. *Transfer* events (represented by a triangle) have rates of failure equal to  $1 \times 10^{-2}$  each. Similarly to the LEON3FT experiment, it is assumed that the HSCOM subsystem under analysis is implemented on an SRAM-based FPGA. The same five fault mitigation policies were utilized. This experiment is conducted by evaluating the failure probability after each of the FT gates in the HSCOM fault tree. Furthermore, for simplicity, it is assumed that all the components have the same area overhead. It is also assumed that the area of a voter is 10% of the area of a regular component.

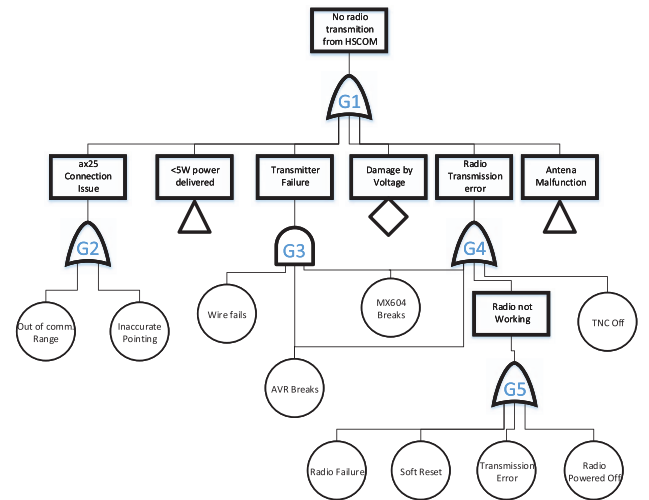


Fig. 5: Fault Tree of the Hermes Cubesat HSCOM Component



TABLE III: Probability of Failure of the Hermes HSCOM Subsystem

	Probability of Failure				
	Unmitigated	Scrubbing	Scrubbing+TMR1v	Scrubbing+TMR2v	Scrubbing+TMR3v
Top Level (G1)	37.2%	25.2%	9.3%	8.6%	7.8%
Avg. Improvement	0%	42%	75%	77%	79%
Area Overhead	0%	0%	100%	120%	140%

The results in Table III show a similar rate of improvement between the vulnerability estimations across the different mitigation approaches. Due to its reduced size, we were able to estimate the area overhead that resulted from the employment of TMR in the HSCOM subsystem. It can be seen that the vulnerability is greatly reduced between the scrubbing-only approach and any of the TMR approaches. However, even the smallest TMR is expected to double the area required to implement the system. It can also be seen that, in the case of this system, increasing the number of voters in the TMR is hardly beneficial to the system, since the increase in the area overhead greatly outbalances the vulnerability improvements. It must be noted though that these numbers may vary drastically from SUT to SUT.

#### IV. CONCLUSION AND FUTURE WORK

In this paper, an efficient probabilistic modeling and analysis of fault mitigation techniques using fault trees is proposed. A new probabilistic approach for modeling repairable fault trees was developed. These models are used to construct the CTMDP model the fault tree of the system under test. Next, efficient probabilistic model checking is utilized to perform an exhaustive analysis, with the objective to identify vulnerabilities and to estimate and report the impact of different fault mitigation techniques on the system. Notably, the proposed analysis can apply and estimate the impact of self-repair and of different configurations of TMR and approximate the total area overhead derived from each TMR instance.

Our results indicate that the proposed methodology is very scalable and has good potential for future extensions, such as the addition of analysis of repairable systems over time and the addition of component decay to our fault tree notation.

#### REFERENCES

- [1] P Sivakumar, G Karthy, and K Vidya Bharani. Fault tolerant improvement mechanism for 3d memories using built-in self repair scheme. In *Electrical, Instrumentation and Communication Engineering (ICEICE), 2017 IEEE International Conference on*, pages 1–6. IEEE, 2017.
- [2] Sanjay Patnaik and V Ravi. A built-in self-repair architecture for random access memories. In *Nanoelectronic Materials and Devices*, pages 133–146. Springer, 2018.
- [3] Andrew M Keller and Michael J Wirthlin. Benefits of complementary seu mitigation for the leon3 soft processor on sram-based fpgas. *IEEE Transactions on Nuclear Science*, 64(1):519–528, 2017.
- [4] Marco Bozzano, Alessandro Cimatti, and Cristian Mattarei. Automated analysis of reliability architectures. In *Engineering of Complex Computer Systems (ICECCS), 2013 18th International Conference on*, pages 198–207. IEEE, 2013.
- [5] Marco Bozzano, Alessandro Cimatti, and Cristian Mattarei. Efficient analysis of reliability architectures via predicate abstraction. In *HaiFa Verification Conference*, pages 279–294. Springer, 2013.
- [6] M. Ammar et al. Comprehensive vulnerability analysis of systems exposed to seus via probabilistic model checking. In *RADECS*, pages 1–4, 2016.
- [7] A. Rae et al. A behaviour-based method for fault tree generation. In *Int. System Safety Conference, System Safety Society*, pages 289–298, 2004.
- [8] M. Ammar et al. Efficient probabilistic fault tree analysis of safety critical systems via probabilistic model checking. In *Forum on Specification and Design Languages (FDL)*, pages 1–8, 2016.
- [9] M. Ammar et al. System-level analysis of the vulnerability of processors exposed to single-event upsets via probabilistic model checking. *IEEE Transactions on Nuclear Science*, 64(9):2523–2530, 2017.
- [10] W.E. Vesely et al. Fault tree handbook, 1981. In <http://www.hq.nasa.gov/office/codeq/doctree/fthb.pdf>.
- [11] Jacob A Abraham and Daniel P Siewiorek. An algorithm for the accurate reliability evaluation of triple modular redundancy networks. *Computers, IEEE Transactions on*, 100(7):682–692, 1974.
- [12] Peter A Lee and Thomas Anderson. *Fault tolerance: principles and practice*, volume 3. Springer Science & Business Media, 2012.
- [13] Michele Favalli and Cecilia Metra. Tmr voting in the presence of crosstalk faults at the voter inputs. *Reliability, IEEE Transactions on*, 53(3):342–348, 2004.
- [14] Darshan D Thaker, Rajeevan Amirtharajah, F Impens, IL Chuang, and Frederic T Chong. Recursive tmr: Scaling fault tolerance in the nanoscale era. *Design & Test of Computers, IEEE*, 22(4):298–305, 2005.
- [15] M. Kwiatkowska et al. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV’11)*, volume 6806, pages 585–591, 2011.
- [16] Rajeev Alur and Thomas A Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.
- [17] LEON3FT-RTAX Fault tolerant Processor. [Online]. Available: <https://www.gaisler.com/doc/leon3ft-rtax-ag.pdf>.
- [18] Felix Bidner. Fault tree analysis of the hermes cubesat. *University of Colorado at Boulder, USA*, 2010.