SEU Mitigation for SRAM FPGAs: A comparison via Probabilistic Model Checking

Viny Cesar Pereira[1], **Valdivino Alexandre de Santiago Júnior**[1], **Silvio Manea**[1]

[1]Instituto Nacional de Pesquisas Espaciais (INPE)
Av. dos Astronautas, 1758, São José dos Campos, São Paulo – SP – Brazil

{viny.pereira,valdivino.santiago,silvio.manea}@inpe.br

Abstract. *Although there are several Single-Event Upset (SEU) mitigation techniques for SRAM-based Field Programmable Gate Arrays (FPGAs), comparisons are still necessary regarding dependability analyzes of these techniques. Most of these assessments analyze the techniques after design and implementation in FPGA which may be too costly. Stochastic/Probabilistic analysis allow to obtain results in the early stages of design. In this paper, we compare three of these strategies, Scrubbing, Triple Modular Redundancy (TMR), and Hamming code, via Probabilistic Model Checking. Results show that TMR allows upsets to accumulate and must be combined with Error Correction Codes (ECCs), such as Hamming, and that the Scrubbing interval directly affects reliability while safety is more related to the coverage rate.*

1. Introduction

Ensuring safety, reliability and availability in complex systems has been a major challenge for designers, especially in applications where hardware and software failures can cause catastrophic financial and scientific harm. For aerospace systems development, SRAM-based Field Programmable Gate Arrays (FPGAs) have some advantages over the Anti-fuse technology such as decreased cost and reconfiguration flexibility [Berg et al. 2008]. However, the harsh environment of space may trigger the occurrence of Single-Event Effects (SEEs) such as Single-Event Upset (SEUs) in SRAM-based FPGAs. Thus, it is necessary to develop techniques to mitigate the consequences of SEUs in FPGA's configuration and funcional logic paths.

Several techniques of detection, mitigation and correction of SEU have appeared in the past as a way to avoid failures in space systems based on FPGAs [Heiner et al. 2009, Hentschke et al. 2002, Liu et al. 2012, Morgan et al. 2007]. In recent years, researchers have proposed improvements to adapt these techniques to advances in the manufacturing process of integrated circuits. The most common approaches involve hardware redundancy, such as the Triple Modular Redundancy (TMR) [Rollins et al. 2003] which consists of tripling circuit modules that demonstrate greater sensitivity to upsets and, via a voting mechanism, to decide the correct output based on the majority. Other forms of redundancy exist, such as temporal redundancy which masks the occurrence of a fault because it obtains the signal at different times, saves the values obtained and, as in TMR, uses a voter to decide which output is correct.

Another category of technique is known as Error Correction Code (ECC) based on the addition of information (parity bits) next to the input data to detect and mitigate possible errors [Dutta and Touba 2007]. There are several ECCs used in the context of SEUs

such as Hamming code [Kumar and Umashankar 2007], Reed-Solomon code, and Bose-Chaudhuri-Hocquenghem (BCH) codes. By checking the parity bits these techniques are able to detect and correct single-bit errors and, in some more complex implementation, also correct double-bit errors [Kastensmidt 2007].

Comparative analyzes of SEU mitigation techniques have been recently published [Shuler et al. 2009, Morgan et al. 2007, Liu et al. 2012, Hentschke et al. 2002]. Many factors must be taken into account before deciding on one particular technique or even the decision to combine the techniques. Some important aspects such as additional area required, performance impact, ability to correct faults or just mitigate them, robustness against multiple upsets may interfere in deciding which technique is most interesting to use. Studies such as [Ejlali et al. 2006, Sari et al. 2013] suggest combining techniques for best results, however, ensuring performance, reliability and safety can become a challenge given the complexity of the problem. To validate these techniques, most of these works use simulations and tests after design and implementation in FPGAs, and fault-injection tools to simulate the upsets and evaluate the behavior of the techniques. These approaches are not able to evaluate, earlier within the development process, all possible situations and may miss faults that will be costly to detect and correct later.

Formal Verification methods [Baier et al. 2008] have been used in several different application domains. Particularly, Probabilistic Model Checking [Kwiatkowska et al. 2009] has shown to be an efficient and robust technique for modeling a wide variety of problems in diverse fields such as biology, security, network and communication protocols, performance and reliability just to name a few. Probabilistic models such as Discrete-Time Markov Chain (DTMC), Continuous-Time Markov Chain (CTMC) and Markov Decision Process (MDP) have been employed in this context.

In [Hoque et al. 2014, Hoque 2016] the authors analyzed the dependability of SEU mitigation methods on SRAM-based FPGAs via Probabilistic Model Checking. However, the authors' approach deals with a combination of two techniques (TMR and Scrubbing), which when compared, are placed in only two situations: only Scrubbing and TMR with Scrubbing. In addition to not being considered independently, the TMR was analyzed in a simplified way, only as spares components, ignoring important details such as the voter and the input and output data. Hence, in this paper we present a comparison between three popular SEU mitigation techniques for SRAM FPGAs, Scrubbing [Berg et al. 2008], TMR [Kastensmidt et al. 2005], and Hamming Code [Kumar and Umashankar 2007], via Probabilistic Model Checking. Our dependability analysis relied on CTMC and Continuous Stochastic Logic (CSL) extended with Rewards. Results show that TMR allows upsets to accumulate and must be combined with ECCs, such as Hamming, and that the Scrubbing interval directly affects reliability while safety is more related to the coverage rate.

This paper is structured as follows. Section 2 presents some background to better understand the three mitigation techniques and an overview of Probabilistic Model Checking. Section 3 shows a general explanation of how the CTMC models were created to support our analysis, and what types of CSL properties we considered. Dependability results and discussion are in Section 4. Section 5 presents related work. In Section 6, we point out the conclusions and future directions of this research.

2. Background

This section presents details of the SEU mitigation techniques addressed in the paper and the concepts related to Probabilistic Model Checking.

TMR is one of the most commonly used technique to prevent failures caused by SEUs, especially single bit upsets (SBUs) [Kastensmidt et al. 2005]. In general, the TMR strategy is to use three replicates of the components that are most sensitive to the effects of radiation, because in case of upset the voting mechanism will be able to choose through the majority what will be the correct output [Rollins et al. 2003].

The masking of the component reached by radiation is possible because TMR removes individual points of failure through redundancy however, as in any redundant system, there is a cost. To ensure that a failure does not reach a common point between all replicas and propagates the fault to the output, each redundant component must have individual input and output ports, causing an approximately six-fold increase in the required memory area [Morgan et al. 2007]. Thus, deciding which logical level the TMR will be applied is a complicated task, and the relation between the additional area and the effective gain must be considered.

In addition to area overload, another important feature about TMR is that due to masking of the fault rather than the correction, the hit component will continue to deteriorate, causing an upset buildup [Kastensmidt 2007]. TMR should be considered in applications with a low incidence of MBUs, since it is not capable of handling more than one simultaneous failure in the replicated modules.

Scrubbing is the mechanism that reconfigures the memory using the bitstream with the original configuration, usually stored in a memory that is immune to radiation and SEEs [Sari et al. 2013]. One of the greatest advantages of Scrubbing over full reconfiguration is that Scrubbing does not completely interrupt the operation mode of the system, only part of the configuration is updated with each Scrubbing interval [Berg et al. 2008].

The most common form of Scrubbing is known as Blind Scrubbing because it does not include a fault detection mechanism, only a copy of the original data is used to rewrite the current configuration. One of the decisions in blind Scrubbing is the time interval, i.e. the frequency at which a cycle should occur. The Scrubbing interval depends on the level of reliability desired by the system and how often the upsets occur. We also consider the coverage rate which is responsible for the conditional probability of detecting the fault given that the fault exists. In this way, it is possible to change the coverage rate of the model and observe how it behaves. Other more sophisticated forms of Scrubbing, such as Readback Scrubbing, use detection techniques and are able to restore settings only when a failure is reported [Heiner et al. 2009].

Hamming code is one of the techniques of ECC used in several applications for the detection of single or double bit errors and correction of single bit errors in binary codes [Dutta and Touba 2007]. As in TMR, Hamming code is a technique indicated in applications with low incidence of MBUs.

The operation of the Hamming code is structured in two modules, an encoder and a decoder. The encoder is responsible for calculating the parity bits based on the original data and, as the name says, encode it to the bitstream [Kastensmidt et al. 2005]. The

actual work is done by the decoder which must check the parity bits again to see if there is a error. In case of error, the decoder is also responsible for finding which bit has failed and repairing it [Liu et al. 2012]. There are several Hamming code variations designed to serve applications with different goals. In this research, we used Hamming code (7,4) which encodes three parity bits for each four bits of data.

2.1. Probabilistic Model Checking

Critical complex systems used in aerospace, medical, communication and other applications are very sensitive to software and/or hardware failures. To date, classical systems validation methods include simulation which is still utilized at a modeling level and later, with the model already constructed, the test cycle runs. Although efficient for most problems, when it comes to critical systems, only simulation may not be sufficient to address all possibilities and ensure the desired level of safety.

To increase accuracy, Formal Methods have shown great potential to reduce the time spent and increase the effectiveness of the modeling process by applying mathematical rigor to ensure system correctness [Baier et al. 2008]. Model Checking is a very popular Formal Verification method that exhaustively explores the state space in an automated way and is able to find failures in the solution (Transition System - TS) like unreachable states or deadlocks. Typically, a property is formalized via a variety of temporal logics such as Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) and the approach systematically verifies if the Transition System (model) satisfies this property ($TS \models \Phi$).

Within Model Checking, a category that deals with systems that present stochastic behavior is Probabilistic Model Checking [Baier et al. 2008, Kwiatkowska et al. 2009, Hoque et al. 2014] where, instead of the absolute guarantee of correctness hardly achieved by Model Checking, is able to guarantee, within a specified probability and in a less demanding manner, the correctness of the system. Markov chains (DTMCs, CTMCs) are typically used as the probabilistic model of the system. Variations of CTL, such as Probabilistic Computation Tree Logic (PCTL) for DTMC and CSL for CTMC, are used in this approach.

3. CTMC Models and CSL Properties

For our dependability analysis we used PRISM [Kwiatkowska et al. 2009], a tool that allows to create stochastic models and check their properties. We selected CTMC to model the three mitigation techniques and properties were formalized in CSL. The CSL operators P, S and R indicate the probability of an event occurring, the long-run probability of a condition being satisfied and the values returned by the costs or rewards of the models, respectively.

Our work is based on [Hoque 2016, Hoque et al. 2014] where the authors investigated Scrubbing and TMR combined. The CTMC models represent SEU mitigation techniques in applications that use adders and multipliers because these components are implemented in the memory of SRAM-based FPGAs, therefore, sensitive to upsets. Failure rates (λ) of the components were estimated in [Hoque 2016], using per bit upset rate for Xilinx Virtex-5 in Highly Elliptical Orbit (HEO), which is $7.31\text{x}10^{-12}$ SEUs/bit/sec.

This rate is multiplied by the number of critical bits of each component. For the components used, the Mean Time Between Failures (MTBF) was 11.85 days for the Wallace Tree Multiplier and 38.15 days for the Kogge-Stone Adder. Hence, $\lambda = 1/MTBF$.

The mitigation techniques were designed in PRISM by describing the behavior of each technique and the characteristics desired for the system, such as quantity of components, size of input and output data, failure rates and coverage rates. With this information, each model was implemented in modules, where each module has sets of commands that will make the necessary transitions to represent the states of the models.

For Scrubbing, the model was provided in [Hoque 2016] but rather than analyzing the results with adders and multipliers limited to three components each, our model handled 10 adders and 10 multipliers to evaluate the availability, safety and reliability of Scrubbing into a more complex application that requires more components in the configuration. In summary, our aim here is to confirm whether the conclusions presented in [Hoque 2016] are valid for more complex applications. In Scrubbing there are only two modules, one for adders and another for multipliers and the function of these modules is to control the amount of operational and degraded components.

As the model addresses Blind Scrubbing, there is no fault detection technique and the repair is performed regardless of the number of operating components. The Scrubbing interval stipulated in [Hoque 2016] is 1, 4 and 9 days and occurs synchronously between adders and multipliers.

The CTMC model in [Hoque 2016] does not actually address a true TMR. In his analysis, the author used a model with Scrubbing and to represent the TMR, added spare components, but ignored the structure and functioning of the TMR, since it did not include the majority voting mechanism nor the input and output data of each component. In addition to a superficial modeling of the technique, the TMR was not analyzed individually, only combined with Scrubbing. Thus, we decided to create our own TMR model.

Our PRISM code has eight modules: six represent the triple redundancies of the adders and multipliers, and there is one voter for the adders and another for the multipliers. Figure 1 presents PRISM's code of the first adder, the same logic was applied to multipliers. Note that `a1` and `a2` represent the 4-bit inputs, `outA1` represents the output of the module, `Na` is the number of operational adders and `lambda_A` is the failure rate stipulated for the adders. PRISM's language allows synchronization between modules and we have used this feature in our CTMC model. Unlike the model in [Hoque 2016], our solution is more refined in the sense that we explicitly represent inputs (4-bit for adders and multipliers) and outputs (5-bit for adders and 8-bit for multipliers), as well as the modules that actually represent the functioning of a voter. The calculation of the event rates in synchronized commands is the product of the rates stipulated for each command. We chose the passive/active approach: rate 1 in one module and the true λ in the other module.

Initially, all components are operational and receive two 4-bit input values. Based on the stipulated failure rate, each module can function properly and assign the correct value to the output or have an upset and return an incorrect value. The failure rate is multiplied by the number of operating components, so the upset modules will not be taken into account in the next cycle. With the output values returned by the modules,

```
module adder1
  a1 : [0..15] init 0;
  a2 : [0..15] init 0;
  outA1 : [0..31] init 31;
  [input] (a1=0 & a2=0) -> 1: (a1'=5) & (a2'=7);
  [adder] ((a1 > 0 | a2 > 0) & (Na > 0)) -> Na*(1-lambda_A) :
  (outA1'= a1+a2) + Na*lambda_A : (outA1'= a1+a2-1);
endmodule
```

Figure 1. Adder module for our TMR model in PRISM.

the voters take action and decide what the final output will be based on the majority, so TMR is able to ignore the occurrence of an upset because the other two modules have the correct output value. In the case of two simultaneous upsets, the voter will consider the wrong output as a majority, and consequently the result will be incorrect.

To classify the states of CTMC models, PRISM offers the concept of formulas. In our TMR we separated them into three types: operational, where all components are functional; degraded, where at most one component of adders and multipliers suffered an upset, but the output is still correct; and failed, where more than one component has suffered upset and the output generated by the voter is incorrect. A representation of the *failure* formula is shown below:

```
formula fail = ((outMf!=m1*m2) | (outAf!=a1+a2));
```

We considered a third SEU mitigation approach in our research. The model developed for the Hamming code is not directly related to adders and multipliers because it can be implemented to detect errors in any memory component of SRAM-based FPGAs. To represent the operation, two modules, an encoder and a decoder, were created, where the encoder receives a 4-bit input and based on these values, calculates the 3 additional parity bits. The detection and correction of a possible error is performed in the decoder which can receive the unchanged bits from the encoder or insert a error in any of the bits based on a failure rate.

As the model is generic, the failure rate of the Hamming code is dependent on the sensitivity of the FPGA model and the architecture of the protected component. For a fair comparison between the techniques and considering that Hamming code is dealing with upsets in the same components, the failure rate chosen follows the previous techniques with a 10-day MTBF to ensure that the results represent a scenario where the incidence of upsets is greater than stipulated for adders and multipliers in TMR and Scrubbing. When the decoder checks the parity and input bits and does not detect any changes, the value is passed to the output normally, but if the values do not match, the decoder identifies the position of the upset and restores the correct bit value. As well as TMR, we used formulas to classify the states: operational, indicating that the output bits of the decoder coincide with the input bits of the encoder; degraded, which represents the states that detected a fault and are in the process of correction; and failed for the interval between the occurrence of a SEU and the detection by the decoder.

Each property we used is in Table 1 where we show the formula in CSL extended with Rewards (R), related dependability attribute, and its informal meaning. Note that

since only in Scrubbing exists the possibility of a fault occurring and not being detected, only this technique had its safety analyzed. We considered that in both TMR and Hamming code, the occurrence of a fault will always be detected by the voter and the decoder, respectively.

Table 1. Properties formalized in CSL extended with Rewards

Attribute	Formula	Meaning
Availability	$R\{$"*timeOperational*"$\}$=? [$C <= t$]	Accumulated time in the operational mode within the time interval $[0, t]$.
Availability	$R\{$"*timeDegraded*"$\}$=? [$C <= t$]	Accumulated time in the degraded mode within the time interval $[0, t]$.
Availability	$R\{$"*timeFailure*"$\}$=? [$C <= t$]	Accumulated time in the failure mode within the time interval $[0, t]$.
Availability	S=? [*fail*]	The long-run probability of the system fails.
Reliability	P=? [$G[0, t]$ *oper*|*degrade*]	The probability of the system being operational or degraded in the first t days.
Safety	P=? [$G[0, t]$ *oper*|*degrade*|$fail_{safe}$]	The probability of the system being operational, degraded or failed safe, in the first t days.

4. Results and Discussion

This section presents the results obtained with CTMC models for SEU mitigation techniques discussed in this paper and a comparative analysis.

4.1. Availability Analysis

For this analysis, we considered a 90-day mission time to analyze how long each SEU mitigation technique would be in an operational, degraded, or failure state. Figure 2 shows the rewards results obtained with TMR where approximately half of the mission time in failed state demonstrates that, a mitigation technique implemented individually without a correction technique capable of restoring the functioning of the degraded components, can cause problems in a short period of time.

By analyzing the same properties for the Hamming code model, Figure 2 also illustrates a better result even if we have a higher failure rate than the one used in TMR. With more than 80% of the total mission time in operational state, Hamming code becomes an interesting correction technique for systems with low incidence of MBUs. One of the disadvantages of Hamming code is that if the number of protected bits is too large and several memory components implement the encoder and decoder blocks, the impact on FPGA's performance can be very large. An option to work around this problem is to implement TMR and Hamming code together, so the correction will only be performed in the case of upsets accumulations.

Rewards results obtained with Scrubbing using 10 adders and 10 multipliers were compared directly with the results presented in [Hoque 2016] using 2 adders and 2 multipliers. Figure 3 shows that for a 1 day interval, the results presented by the model with less components can keep the system in an operational state for a longer time, while the configuration with 10 adders and 10 multipliers spends more time degraded than operational but remains functional almost 100% of the time. The coverage rate used in both models was 99%.

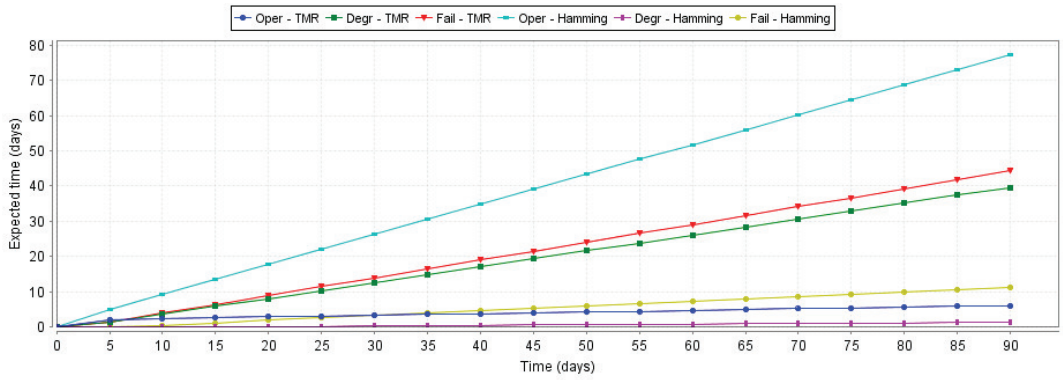

Figure 2. TMR and Hamming code's rewards analysis: 90-day mission.

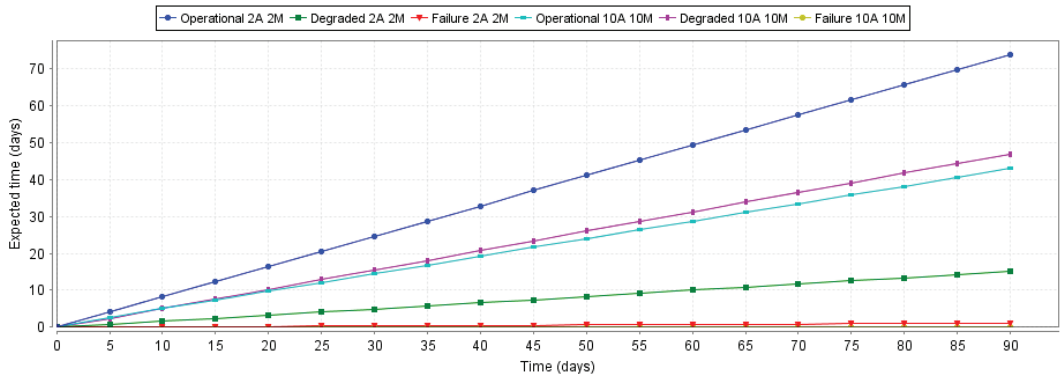

Figure 3. Rewards analysis, 1-day Scrubbing interval: 2A 2M = presented in [Hoque 2016]; 10A 10M = our contribution.

The situation is reversed with a Scrubbing interval of 9 days, even degraded, the configuration of 10 adders and 10 multipliers can keep the system running for much of the mission time, while the time in failure state presented by the other configuration goes up and may become unacceptable for systems that require high reliability, as shown in Figure 4.

With these results, it is possible to infer that in models with a smaller area, i.e. with less components susceptible to upsets, a very large Scrubbing interval can become a problem because the upsets will accumulate and it is likely that no operational components will left over. With a larger number of components, the larger Scrubbing interval becomes acceptable, since even if the upsets accumulates, the operating components will be overloaded but will keep the system functional. The accumulation of upsets occurs when more than one component is reached within the same Scrubbing interval, and since there is no detection technique (Blind Scrubbing), the corrections will only occur at the end of the stipulated interval.

Regarding the other perspective of availability, the probability of long-run failure of TMR and Hamming code techniques, Table 2 shows the values obtained in the long run. Again, due to the lack of a technique that prevents the accumulation of upsets, TMR presents a high probability of failure.

For Scrubbing, the failure probability in the long-run directly depends on the cho-

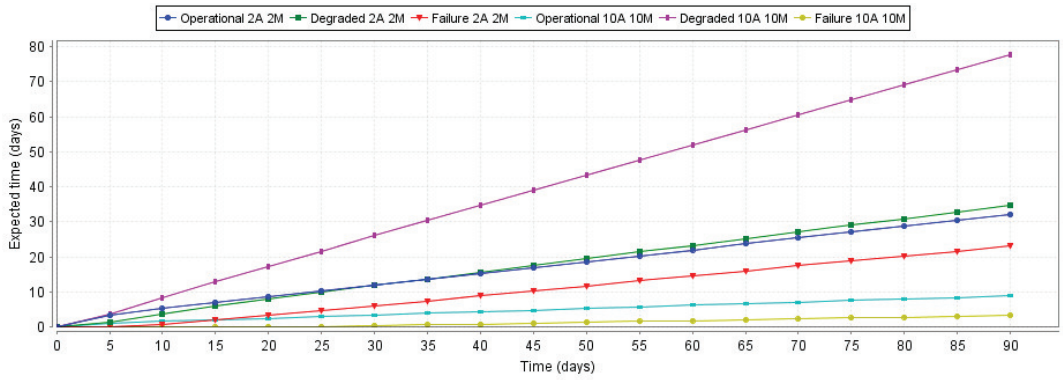

Figure 4. Rewards analysis, 9-day Scrubbing interval: 2A 2M = presented in [Hoque 2016]; 10A 10M = our contribution.

Table 2. TMR and Hamming code failure probability in the long-run.

SEU mitigation technique	Failure probability
TMR	50.5%
Hamming Code	13.5%

sen interval, as shown in Figure 5. For a small interval, the results are very similar between the configurations, but as the interval grows, the amount of components becomes inversely proportional to the probability of failures. In other words, the more components in the system, the less chance that all will suffer an upset within an interval of Scrubbing.

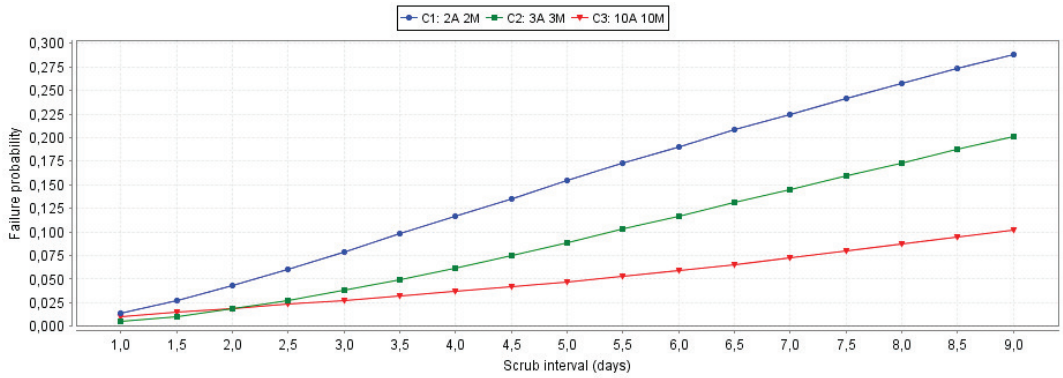

Figure 5. Scrubbing failure probability in the long-run: 2A 2M and 3A 3M = presented in [Hoque 2016]; 10A 10M = our contribution.

4.2. Reliability and Safety Analysis

In the reliability analysis, which is the probability of the system being operational or degraded in the first 90 days, the Hamming code presented a superior result compared to TMR: in half the mission time, TMR was already with zero chances of remaining without failures. Figure 6 shows the results obtained in the analysis and despite the superiority of the Hamming code, both results were lower than expected.

As before, Scrubbing reliability analysis was split into two intervals: 1 day and

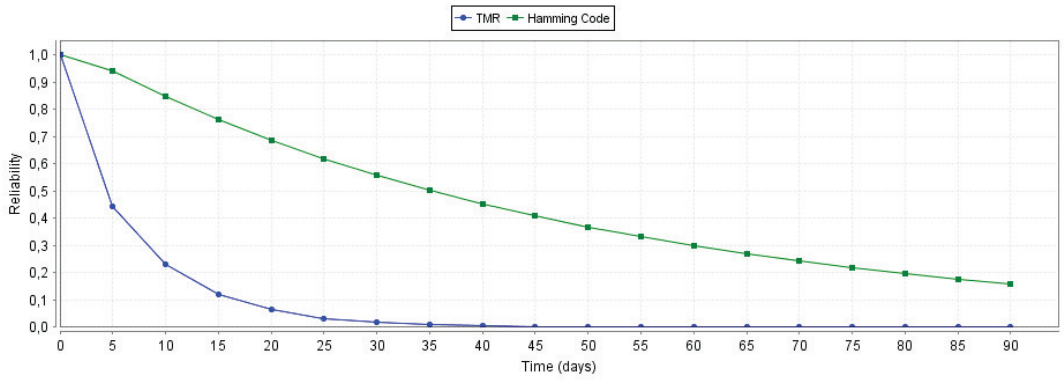

Figure 6. Reliability of TMR and Hamming code: 90-day mission.

9 days. From now on, the configuration presented in [Hoque 2016] with 2 adders and 2 multipliers will be called C1 while our configuration of 10 adders and 10 multipliers will be C2. Figure 7 represents the reliability of Scrubbing for C1 and C2 by varying the coverage percentage between 100% and 90% on a 90-day mission with a 1-day Scrubbing interval. The maximum reliability was obtained in C2 with 100% coverage, but it is possible to observe that the reduction of the coverage rate affected more drastically the reliability of C2 in both 95% and 90%. This shows that the more components in the system, more important is to ensure an efficient fault detection mechanism to keep the coverage always close to 100%.

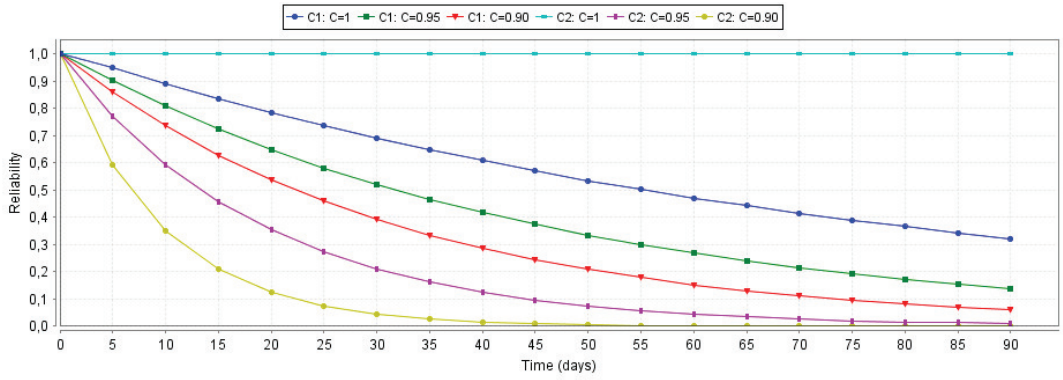

Figure 7. Scrubbing reliability, 1-day interval: 90-day mission.

Keeping the same properties but increasing the Scrubbing interval to 9 days, with the exception of C2 with 100% coverage, all other configurations lost their reliability until the end of the 90 days, as shown in Figure 8. This indicates how much the Scrubbing interval can influence the correct functioning of the system, especially in systems with fewer components.

The safety of a system that has a mitigation method such as Blind Scrubbing is related to the probability of detecting a fault since the fault occurred, i.e. with the coverage rate. This is because there is no fault detection technique in the Blind Scrubbing, so it runs at every preset time interval, even if no fault has occurred. Figures 9 and 10 show how decreasing coverage from 99% to 95% drastically decreases safety in both C1 and C2. This result indicates that for systems that value for safety, ensuring a high coverage rate

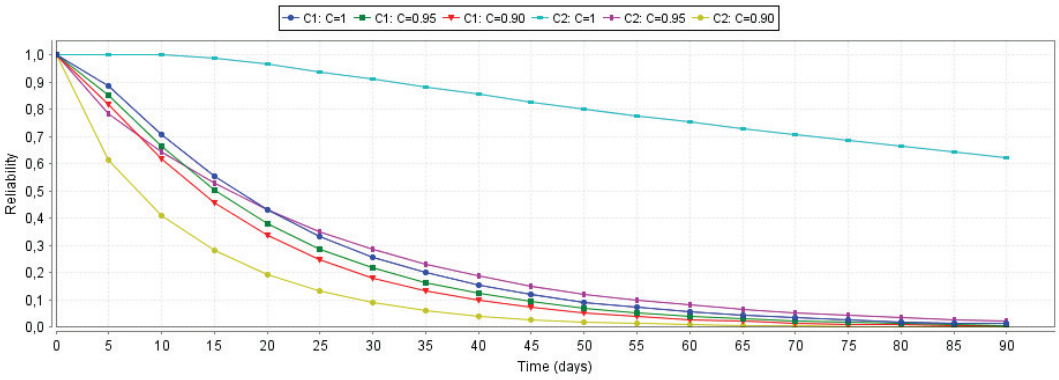

Figure 8. Scrubbing reliability, 9-day interval: 90-day mission.

will yield better results than changes in the Scrubbing interval. In addition, C1 was able to maintain safety greater than C2 at both coverage rates, showing that systems with fewer components decrease the chances of failures to go unnoticed and make the system unsafe.

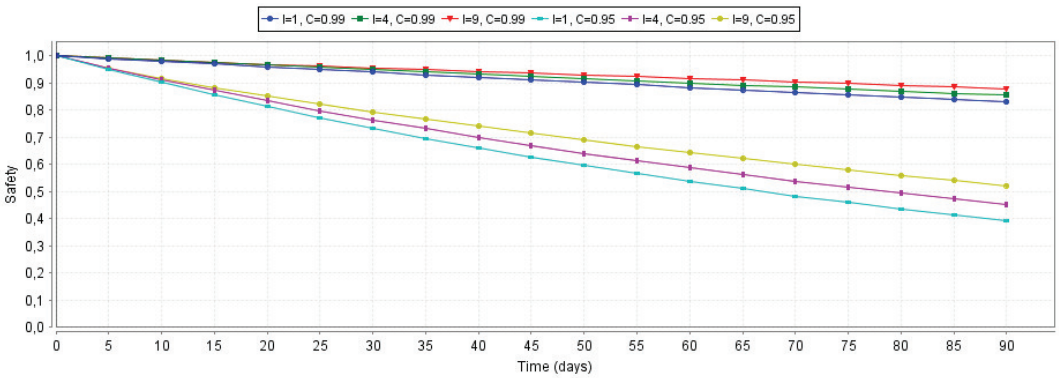

Figure 9. Scrubbing safety, 2 adders and 2 multipliers (C1): 90-day mission.

5. Related Work

SEU mitigation techniques in SRAM-based FPGAs used today have also undergone evolutions, and several works over the past few years have brought comparisons, combinations, and variations of these techniques to fit their needs. In [Morgan et al. 2007], the authors compare TMR with three additional techniques of SEU mitigation for FPGAs: quadded logic, state machine encoding and temporal redundancy. To simulate the harsh environment of space, they used a Xilinx Virtex fault-injection tool and analyzed the results based on reliability, area overload and reduced sensitivity of FPGAs. The conclusion was that these techniques presented a greater area overload and a smaller sensitivity reduction when compared with TMR, guaranteeing TMR a higher reliability. In [Shuler et al. 2009], TMR is also compared with other mitigation methods such as radiation-hardened flip flops and dual-rail logic. Considering area efficiency, TMR presented the best result with 95% efficiency.

In [Berg et al. 2008], the authors compare the effectiveness of two different forms of Scrubbing. The first form is internal, implemented by Xilinx, and the second one is external, but without a frame by frame readback and no method of detection of upsets.

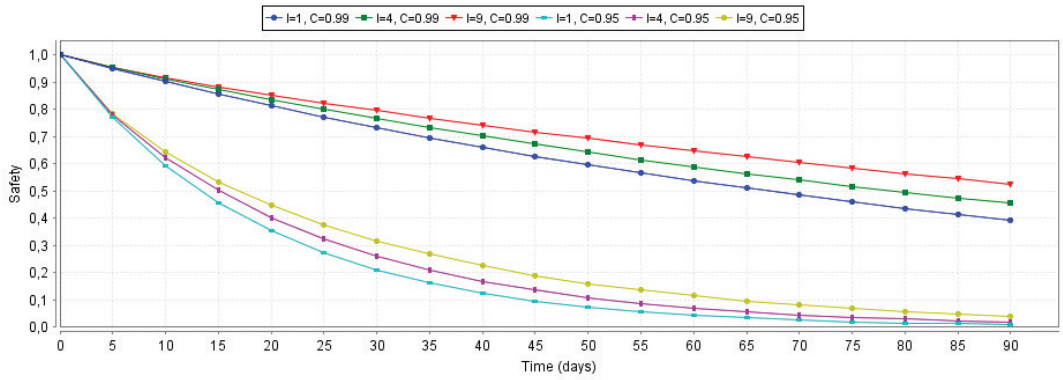

Figure 10. Scrubbing safety, 10 adders and 10 multipliers (C2): 90-day mission.

Results showed that even the internal scrubber being more complete, its efficiency was not superior to the external scrubber, being little ahead of an FPGA with no Scrubbing technique.

In [Liu et al. 2012], the authors compare Hamming code with another ECC, the Difference-Set Cyclic Codes (DSCC). The objective was to analyze performance, cost design and failure rate. The results showed that although the DSCC is more suitable for the Xilinx FPGA architecture, in practice Hamming code showed a lower failure rate and a better energy consumption, resulting in a more feasible cost design. A direct comparison between the TMR and the Hamming code was presented in [Hentschke et al. 2002] analyzing the impact on performance and area. As expected, the results showed advantages and disadvantages in each of the techniques. The TMR significantly increases the area of memory cells but performs better because it has a lower delay while Hamming code has only a small increase in memory cells, but depending on the amount of bits it must protect, it can add a significant delay. In our analysis, TMR had a lower performance.

All previous comparative analyzes have been conducted after implementing the techniques in FPGAs and simulate upsets with fault-injection tools, but this type of approach can be costly, since it presents the results only at the end of the simulations. On the other hand, stochastic models can be used in initial stages of the project and possibly help traditional approaches achieve better results in a complementary manner. Probabilistic analyzes via Formal Verification methods are still scarcely found in the literature, but in [Hoque et al. 2014] and [Hoque 2016] the authors have presented case studies using Probabilistic Model Checking for dependability analysis of SEU techniques. Our work is different from theirs in several ways, starting with the individually analyzed techniques rather than a single model combining two techniques. Moreover, our TMR not only deals with spare components but also includes the input and output bits and the voter. In Scrubbing, a different scenario was addressed, where 10 adders and 10 multipliers showed significant results. In addition, Hamming code was another technique analyzed and showed promising results.

6. Conclusions

In this work, we presented a comparison between three well-known techniques of mitigation of SEU in SRAM-based FPGAs. By using Probabilistic Model Checking, we claim that the benefits are high because we are able to get earlier results of these tech-

niques without the cost and effort spent designing and implementing in FPGA and using fault-injection tools. Eventually, our analysis may not replace such implementation-based approaches, but our results may complement such strategies with an additional probabilistic perspective. We relied on CTMC models to verify possible failures and to analyze the availability, reliability and safety of the techniques within a radiation environment with the characteristics described in this work. The properties formalized in CSL were automatically verified by PRISM [Kwiatkowska et al. 2009].

Results obtained in the availability analysis showed that Hamming code was the technique that stayed the longest in operational state even if we considered the worst-case failure rate. Scrubbing comes in the sequence with interval of 1 day and with interval of 9 days. Comparing the results of the two Scrubbing configurations, smaller systems (less amount of components) can spend less time in failure if the Scrubbing interval is as small as possible. For larger systems, like our configuration with 10 adders and 10 multipliers, larger intervals make the system degraded, but still functional. The worst result was TMR which spent half the total mission time failing because it could not fix the upsets, generating a buildup that makes the system inoperable. One of the alternatives is to combine TMR with some correction technique such as Scrubbing or Hamming code.

In the reliability analysis, TMR also presented the worst result but Hamming code was worse than presented in our availability analysis. For Scrubbing, the greatest influence on reliability is the length of the interval between corrections, especially on smaller systems. Thus, if the application focus is to ensure reliability, the ability to detect faults (coverage rate) can be relaxed, but the Scrubbing interval should be as small as possible. The safety analysis presented the inverse result, since the coverage rate was more relevant in the results than the variation in the Scrubbing interval in both configurations.

We need to realize about the effectiveness of this Formal Verification and probabilistic analysis. Therefore, our next efforts will focus on implementing these SEU mitigation techniques in FPGA and compare whether the results of our analysis meet the practical results and, if the conclusion is positive, in the future one may rely on such probability comparison rather than really implementing the techniques in FPGAs.

References

Baier, C., Katoen, J.-P., and Larsen, K. G. (2008). *Principles of model checking*. MIT press.

Berg, M., Poivey, C., Petrick, D., Espinosa, D., Lesea, A., LaBel, K. A., Friendlich, M., Kim, H., and Phan, A. (2008). Effectiveness of internal versus external seu scrubbing mitigation strategies in a xilinx fpga: Design, test, and analysis. *IEEE Transactions on Nuclear Science*, 55(4):2259–2266.

Dutta, A. and Touba, N. A. (2007). Multiple bit upset tolerant memory using a selective cycle avoidance based sec-ded-daec code. In *VLSI Test Symposium, 2007. 25th IEEE*, pages 349–354. IEEE.

Ejlali, A., Al-Hashimi, B. M., Schmitz, M. T., Rosinger, P., and Miremadi, S. G. (2006). Combined time and information redundancy for seu-tolerance in energy-efficient real-time systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(4):323–335.

Heiner, J., Sellers, B., Wirthlin, M., and Kalb, J. (2009). Fpga partial reconfiguration via configuration scrubbing. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 99–104. IEEE.

Hentschke, R., Marques, F., Lima, F., Carro, L., Susin, A., and Reis, R. (2002). Analyzing area and performance penalty of protecting different digital modules with hamming code and triple modular redundancy. In *Proceedings. 15th Symposium on Integrated Circuits and Systems Design*, pages 95–100. IEEE.

Hoque, K. A. (2016). *Early Dependability Analysis of FPGA-Based Space Applications Using Formal Verification*. PhD thesis, Concordia University Montréal, Québec, Canada.

Hoque, K. A., Mohamed, O. A., Savaria, Y., and Thibeault, C. (2014). Probabilistic model checking based dal analysis to optimize a combined tmr-blind-scrubbing mitigation technique for fpga-based aerospace applications. In *2014 Twelfth ACM/IEEE Conference on Formal Methods and Models for Codesign*, pages 175–184. IEEE.

Kastensmidt, F. L. (2007). See mitigation strategies for digital circuit design applicable to asic and fpgas. In *IEEE NSREC Short Course*.

Kastensmidt, F. L., Sterpone, L., Carro, L., and Reorda, M. S. (2005). 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*, pages 1290–1295. IEEE Computer Society.

Kumar, U. and Umashankar, B. (2007). Improved hamming code for error detection and correction. In *Wireless Pervasive Computing, 2007. ISWPC'07. 2nd International Symposium on*. IEEE.

Kwiatkowska, M., Norman, G., and Parker, D. (2009). Prism: probabilistic model checking for performance and reliability analysis. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):40–45.

Liu, S., Sorrenti, G., Reviriego, P., Casini, F., Maestro, J., Alderighi, M., and Mecha, H. (2012). Comparison of the susceptibility to soft errors of sram-based fpga error correction codes implementations. *IEEE Transactions on Nuclear Science*, 59(3):619–624.

Morgan, K. S., McMurtrey, D. L., Pratt, B. H., and Wirthlin, M. J. (2007). A comparison of tmr with alternative fault-tolerant design techniques for fpgas. *IEEE transactions on nuclear science*, 54(6):2065–2072.

Rollins, N., Wirthlin, M., Caffrey, M., and Graham, P. (2003). Evaluating tmr techniques in the presence of single event upsets. In *Proceedings of the 6th Annual International Conference on Military and Aerospace Programmable Logic Devices*, page P63.

Sari, A., Psarakis, M., and Gizopoulos, D. (2013). Combining checkpointing and scrubbing in fpga-based real-time systems. In *VLSI Test Symposium (VTS), 2013 IEEE 31st*, pages 1–6. IEEE.

Shuler, R. L., Bhuva, B. L., O'Neill, P. M., Gambles, J. W., and Rezgui, S. (2009). Comparison of dual-rail and tmr logic cost effectiveness and suitability for fpgas with reconfigurable seu tolerance. *IEEE Transactions on Nuclear Science*, 56(1):214–219.