Early Analysis of Soft Error Effects for Aerospace Applications Using Probabilistic Model Checking

Khaza Anuarul Hoque[1(✉)], Otmane Ait Mohamed[1], Yvon Savaria[2], and Claude Thibeault[3]

[1] Concordia University, Montreal, Canada
{k_hoque,ait}@ece.concordia.ca

[2] Polytechnique Montréal, Montreal, Canada
yvon.savaria@polymtl.ca

[3] École de Technologie Supérieure, Montreal, Canada
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 and performance of such safety-critical applications can reduce the design effort and increases the confidence. This paper introduces a novel methodology based on probabilistic model checking, to analyze the dependability and performability properties of safety-critical systems for early design decisions. Starting from a high-level description of a model, a Markov reward model is constructed from the Control Data Flow Graph (CDFG) of the system and a component characterization library targeting FPGAs. Such an exhaustive model captures all the failures and repairs possible in the system within the radiation environment. We present a case study based on a benchmark circuit to illustrate the applicability of the proposed approach and to demonstrate that a wide range of useful dependability and performability properties can be analyzed using our proposed methodology.

1 Introduction

Dependability and performability are major concerns in safety-critical and mission-critical applications common in the aerospace industry. 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 [13]. For space missions, the rate of single event effects can be much worse. Due to field programmability, absence of non-recurring engineering costs, low manufacturing costs and other advantages, SRAM-based FPGAs are increasingly attractive. Unfortunately, a great 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 and soft faults [1,21]. Therefore, in aerospace industry, the possibility of cosmic radiation induced soft error grows dramatically at higher altitudes. However, an early analysis of dependability and performance impacts of such errors and faults on the design provides opportunities for the designer to develop more reliable and efficient designs and may reduce the overall cost associated with the design effort. Our work aims at achieving these goals.

This paper proposes a means by which formal verification methods can be applied at early design stages to analyze the dependability and performability of reconfigurable systems. In particular, the focus is on probabilistic model checking [8]. *Probabilistic model checking* is used to verify the systems whose behavior is stochastic in nature. It is mainly based on the construction and analysis of a probabilistic model, typically a Markov chain or a Markov process. These models are constructed in an exhaustive fashion. 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 dependability and performability properties. In contrast, in discrete-event simulations, approximate results are generated by averaging results from large number of random samples. Probabilistic model checking applies numerical computations to provide exact and accurate results.

To analyze a design at high level, we start from its Control Data Flow Graph (CDFG) [17] representation, obtained from a high-level description of the design expressed using a language such as C++. The possible implementation options of the CDFG, with different sets of available components and their possible failures, fault recovery and repairs in the radiation environment are then modeled with the PRISM modeling language [24]. The failure rate of the components are obtained from a worst-case component characterization library. Since the FPGA repair mechanism known as scrubbing [5] can be used in conjunction with other forms of mitigation techniques such as TMR [6] to increase the reliability, we demonstrate in this paper that *rescheduling* [4,16] could be a good alternative candidate in some cases compared to a redundancy-based solution. In the proposed methodology, we show how to use the PRISM model checker tool to model and evaluate dependability, performability and area trade-offs between available design options. Current work in this area either separates the dependability analysis from performance/area analysis, or do not analyze such safety-critical applications at early design stage. Commercial tools for reliability analysis, such as Isograph [15], cannot be used for performance evaluation of such systems as they do not support Markov reward models [27]. Since the probabilistic model checker PRISM allows reward modeling, our work overcomes this limitation. The motivation of the work, the application area, the fault model, considered fault tolerance techniques and the use of probabilistic model checking for system analysis, makes our work unique.

The remainder of the paper is organized as follows. Section 2 reviews motivations and related works. 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 Sect. 4, and in Sect. 5, we present a case study using our proposed methodology. Section 6 concludes the paper with future research directions.

2 Motivation and Related Work

Consider the CDFG of a synchronous dataflow DSP application shown in Fig. 1. Based on data dependencies, this application can be carried out in a minimum of three control steps using the CDFG-1 shown in Fig. 2, with two adders and two multipliers. Such implementation provides a throughput of $1/3 = 0.33$. Another alternative consists of implementing the application with only one multiplier and two adders but in four control steps, as shown by CDFG-2 in Fig. 2. In that case the throughput is 0.25. Based on the priority of throughput or area metric, the appropriate CDFG is selected.

However, inclusion of a reliability metric based on a fault recovery mechanism can make the case more complex and difficult to evaluate. When a resource fails (due to a configuration bit flip), an alternative schedule can be derived to continue the system operation using the remaining resources, most likely at a lower throughput. For example, to maximize the throughput, CDFG-1 is implemented. For a single component failure, e.g. a multiplier, the application can be rescheduled to implement CDFG-2 with lower throughput. Such fault tolerance approach was introduced in [4,12,16] for fault-secure microarchitectures and multiprocessors. For FPGA-based designs, such a fault recovery technique can be adopted as well and we explore the dependability, area and performance trade-offs for such systems. We must mention that the controller for rescheduling the operations is assumed to be fault-free. This controller can be implemented in a separate chip with proper fault-tolerance mechanisms. Considering the example again, we observe that, if another multiplier fails, the CDFG cannot be rescheduled and the system fails to continue its operation. For FPGA-based safety-critical applications, systematic system failure at first occurrence of a soft-error is not acceptable. *Scrubbing* with partial reconfiguration capability [5] can repair bit-flips in the configuration memory without disrupting system operations. Scrubbing can be done at a specified rate meaning that there might be

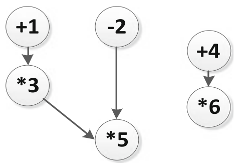

Fig. 1. Sample CDFG

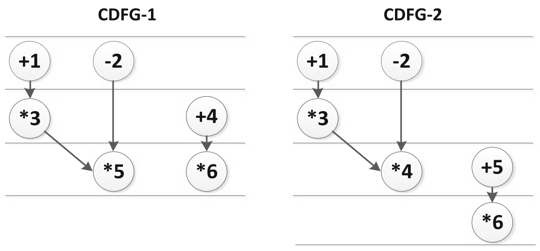

Fig. 2. CDFGs scheduled over available resources

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 [6]. In this work, we use probabilistic model checking to evaluate the dependability and performability vs area trade-offs and demonstrate that in some cases, a redundancy-based solution might not be the best choice as one may expect. Alternatively, for those cases, *rescheduling* in conjunction with *scrubbing* can be a good option.

High-level synthesis algorithms such as forced-directed list scheduling [23] can generate different CDFGs depending on components availability. Switching to another CDFG allows recovering from a failure while a system can continue its operation, possibly with a lower throughput. For many years, fault tolerance techniques and reliability analysis of complex systems have been active research area both in academia and industry. In [29], the authors proposed a reliability-centric high-level synthesis approach to address soft errors. Their framework uses reliability characterization to select the most reliable implementation for each operation fulfilling latency and area constraints. In addition, researchers dedicated lots of efforts in modeling the behavior of gracefully degradable large-scale systems using continuous-time Markov reward models [3,14]. In [26], a case study is presented to measure the performance of a multiprocessor system using a continuous-time Markov reward model. An approach for analyzing performance, area and reliability using a Markov reward model is presented in [19]. The authors used transistor lifetimes to model the reliability and performance, hence the model is composed of non-repairable modules. Use of a non-formal commercial tool makes their approach quite rigid in terms of analysis. Moreover, in their proposed approach, the reward calculation is manual, as the traditional commercial tools for reliability analysis do not support reward modeling.

Even though our model has similarities to performance analysis, our approach is more flexible because we use probabilistic model checking. Our work focuses on a different fault model: cosmic radiation induced configuration bit-flips in FPGAs. Since scrubbing is possible in FPGA designs, we also add repair to our Markov reward model. In consideration of the failure type, repair capability, use of a characterization library to model the system, the application of our work and our methodology is different from and novel when compared to all the related works described above. To our knowledge, this is the first attempt to analyze the dependability and performance to area trade-offs for such safety-critical systems at early design stage using probabilistic model checking.

3 Background

3.1 Soft Errors

In SRAM-based FPGAs, the configuration bitstream determines the routing and functionality of the design. However, a change in the value of one of the SRAM cells can potentially modify the functionality of the design and can lead to catastrophic consequences. The major reason for such inadvertent bit flips in high-altitude is soft errors caused by cosmic radiation. When these particles impact a silicon substrate, they result in the generation of excess carriers, which when deposited on the internal capacitances of a circuit node can result in an upset to the data value stored. The lowering of supply voltages and nodal capacitances with recent technologies have increased the possibility of observing bit flips. Due to this increasing concern, there are several mitigation techniques proposed for tackling the soft error problem.

A mainstream SEU repair technique in SRAM-based FPGAs is configuration scrubbing [11]. Scrubbing refers to the periodic readback of the FPGA's configuration memory, comparing it to a known good copy, and writing back any corrections required. By periodically scrubbing a device, maximum limits may be placed on the period of time that a configuration error can be present in a device. A variation to improve scrubbing is known as *partial reconfiguration* [5]. This is beneficial as it allows a system to repair bit-flips in the configuration memory without disrupting its operations. Configuration scrubbing prevents the build-up of multiple configuration faults. Although scrubbing ensures that the configuration bitstream can remain relatively free of errors, over the long run, there is a period of time between the moment an upset occurs and the moment when it is repaired in which the FPGA configuration is incorrect. Thus the design may not function correctly during that time. To completely mitigate the errors caused by SEUs, scrubbing is used in conjunction with another form of mitigation that masks the faults in the bitstream.

A scrub rate describes how often a scrub cycle should occur. It is denoted by either a unit of time between scrubs, or a percentage (scrub cycle time divided by the time between scrubs). There are direct relationships between scrubbing rate, device size, device reliability and device safety, hence the scrub rate should be determined by the expected upset rate of the device for the given application.

3.2 Probabilistic Model Checking

Model checking [8] is a well established formal verification technique to verify the correctness of finite-state systems. Given a formal model of the system to be verified in terms of labelled state transitions and the properties to be verified in terms of temporal logic, the model checking algorithm exhaustively and automatically explores all the possible states in a system to verify if the property is satisfiable or not. If not, a counterexample is generated. *Probabilistic model checking* deals with systems that exhibit stochastic behaviour, such as fault-tolerant systems. Probabilistic model checking is based on the construction and analysis of a probabilistic model of the system, typically a Markov chain. In this paper, we focus on the continuous-time Markov chains (CTMCs) and Markov reward models [27], widely used for reliability and performance analysis.

A CTMC comprises a set of states S and a transition rate matrix $\mathbf{R} : S \times S \rightarrow \mathbb{R}_{\geq 0}$. The rate $\mathbf{R}(s, s')$ defines the delay before which a transition between states s and s' takes place. If $\mathbf{R}(s, s') \neq 0$ then the probability that a transition between the states s and s' might take place within time t can be defined as $1 - e^{-\mathbf{R}(s,s') \times t}$. No transitions will take place if $\mathbf{R}(s, s') = 0$. Exponentially distributed delays are suitable for modelling component lifetimes and inter-arrival times.

In the model-checking approach to performance and dependability analysis, a model of the system under consideration is required together with a desired property or performance/dependability measure. In case of stochastic modelling, such models are typically CTMCs, while properties are usually expressed in some form of extended temporal logic such as Continuous Stochastic Logic (CSL) [2], a stochastic variant of the well-known Computational Tree Logic (CTL) [8]. Below are a number of illustrative examples with their natural language translation:

1. $P_{\geq 0.98}[\Diamond\ complete]$ - "The probability of the system eventually completing its execution successfully is at least 0.98".
2. $shutdown \Rightarrow P_{\geq 0.95}[\neg\ fail\ U^{\leq 200}\ up]$ - "Once a shutdown has occurred, with probability 0.95 or greater, the system will successfully recover within 200 h and without any further failures occurring".

Additional properties can be specified by adding the notion of rewards. Each state (and/or transition) of the model is assigned a real-valued reward, allowing queries such as:

1. $R = [\Diamond\ success]$ - "What is the expected reward accumulated before the system successfully terminates?"

Rewards can be used to specify a wide range of measures of interest, for example, the number of correctly delivered packets or the time that the system is operational. Of course, conversely, the rewards can be considered as costs, such as power consumption, expected number of failures, etc.

4 Proposed Methodology

In Fig. 3, we present the proposed methodology, which reuses some elements from a methodology proposed in [28], namely the CDFG extraction and the concept of using a characterization library (which was created with a different set of tools). We start from the dataflow graph of the application. Different tools such as GAUT [9], SUIF [10] etc. could be used to extract the dataflow graph from a

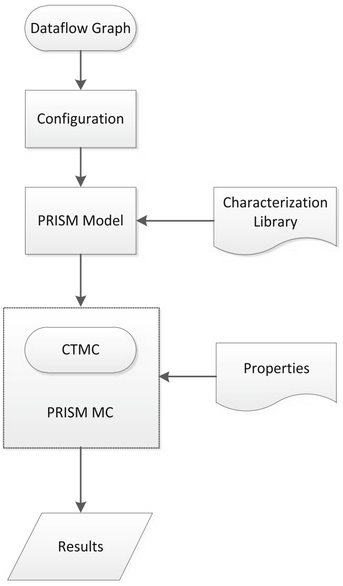

Fig. 3. Proposed methodology

high-level design description expressed using a language such as C++. As mentioned earlier, a CDFG can be implemented with different component allocations (design options). We will refer to the term *design options* as *configurations* in the rest of the paper. Upon a failure, if possible with available resources, the CDFG is rescheduled for fault recovery and the system continues its operation -that is reflected in the CTMC as the next states. For rescheduling the CDFG with available components, a high-level synthesis algorithm, such as *forced-directed list scheduling* [23] can be used. To analyze each configuration, we model them with the PRISM modeling language. Such a model is described as a number of modules, each of which corresponds to a component of the system. Each module has a set of finite-ranged variables representing different types of resources. The domain of the variables represent the number of available components of a specific resource. The whole model is constructed as the parallel composition of these modules. The behaviour of an individual module is specified by a set of guarded commands. For a CTMC, as is the case here, it can be represented in the following form:

$$\texttt{[] <guard>} \rightarrow \texttt{<rate> : <action>} ;$$

The `guard` is a predicate over the variables of all the modules in the model. 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 in terms of how its variables should be updated. The interpretation of the command is that if the `guard` is satisfied, then the module can 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. For another example, consider an application that requires 2 adders and 2 multipliers and such a configuration in the PRISM modeling language can be described as follows:

```
module adder
 a : [0..num_A] init num_A;
 [] (a > 0) -> a*lambda_A : (a' = a - 1);
 [] (a < num_A) -> miu : (a' = num_A);
endmodule

module mult
 m : [0..num_M] init num_M;
 [] (m > 0) -> m*lambda_M : (m' = m - 1);
 [] (m < num_M) -> miu : (m' = num_M);
endmodule
```

In the PRISM code above, `num_A` and `num_M` represent the number of adders and multipliers available in the initial state of the configuration. The `lambda_A` and the `lambda_M` variable represents the associated failure rates of the adders and multipliers whereas `miu` represents the repair rate. Each repair transition (scrub) leads back to the initial state reflecting the scenario that the configuration bit flips have been repaired. The value of `lambda_A` and `lambda_M` is obtained from a component characterization library, that will be explained later in the paper. PRISM then constructs, from this, the corresponding probabilistic model, in this case a CTMC. The resulting CTMC for this configuration is shown in Fig. 4. PRISM also computes the set of all states which are reachable from the initial state and identifies any deadlock states (i.e. reachable states with no outgoing transitions). PRISM then parses one or more temporal logic properties (e.g. in CSL) and performs model checking, determining whether the model satisfies each property.

4.1 Markov Model for Dependability

CTMC models are very commonly used for reliability and dependability modeling. To analyze each configuration, a separate CTMC is built with the help of the PRISM tool and a wide range of dependability properties are verified. For the FIR application in Fig. 5, at a minimum, an adder and a multiplier pair is required for successful operation, hence any state that does not fulfill the minimum resource availability, is labeled as a *failed state*. At the end, the state labeled as *all fail* represents a state where all the components in the system have

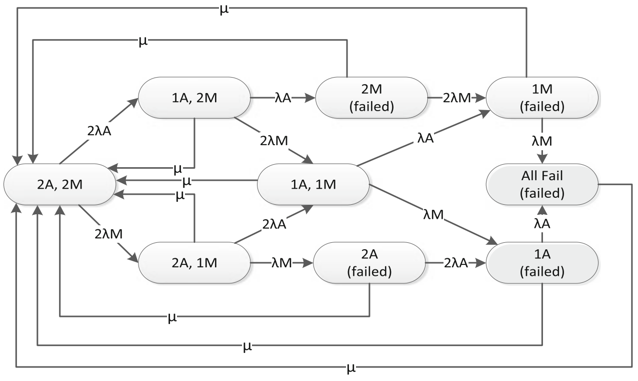

Fig. 4. Sample Markov model

failed due to soft errors one-by-one. The initial state of the configuration has the maximum throughput and all the components are functional. The edges between the states represent transition rates. The assumptions for our model are defined as follows:

Assumption 1: The time-to failure for a component due to a configuration bit flip is exponentially distributed. Exponential distribution is commonly used to model the reliability of systems where the failure rate is constant. The scrubbing behavior is assumed to follow Saleh's probabilistic model [25], e.g. *scrubbing* interval is distributed exponentially with a rate $1/\mu$, where μ represents the scrub rate.
Assumption 2: Only one component can fail at a time due to a soft error. This assumption is made to ensure the complexity in the Markov model is managable.
Assumption 3: *Cold spare* components are used to provide redundancy and are actived only when a same type of component fails. The *cold spare* components are only error prone to cosmic radiation when they are active.
Assumption 4: The reconfiguration and rescheduling times (i.e. the time taken for the system to reschedule when a component fails and the time taken for repair via partial reconfiguration) are extremely small compared to the times between failures and repairs. The time required for rescheduling is at most few clock cycles and the time required for scrubbing is only a few seconds, which is significantly smaller than the failure and repair rate.
Assumption 5: All the states in the CTMC model can be classified into three types: *operational*, -where all the component are functional and the system has the highest throughput; *degraded*, -where at least one of the components is faulty; and *failed*, -where the number of remaining non-faulty components is not sufficient to perform successful operation and hence has a throughput of 0. In PRISM, a *formula* can be used to classify such states as follows:

```
formula operational = (a = num_A) & (m = num_M) ;
```

4.2 Markov Reward Modeling

Markov chains can be augmented with *rewards* to specify additional qualitative measures, known as a Markov Reward Model (MRM). In a Markov reward model, a reward rate function or reward structure $r(X_i)$ where $X \rightarrow \mathbb{R}$ ($\mathbb{R}$ is a set of real numbers) is associated with every state X_i such that r represents the performance of the system when in state X_i. The transient performability is defined as the *expected value* of a random variable defined in terms of a reward structure :

$$E[X(t)] = \sum_{X_i \in X} P_{X_i}(t) * r(X_i)$$

A steady-state accumulated mean reward is obtained by integrating this function from start to an convergent time beyond which rewards are invariant. For performance analysis, we use the throughput metric, hence each state in the MRM is augmented with associated throughput (in a non-pipelined design, throughput is the inverse of latency). The throughput reward at each state in the CTMC is obtained using the forced-directed list scheduling algorithm and all the *failed states* are augmented with a throughput reward of zero. In our MRM model, the area that is required, to implement the design on the FPGA, is assumed to be invariant between the states for a specific configuration. The reason is, once the system is implemented on FPGA, the area is fixed and if a fault occurs, then the system will be rescheduled. So only the control signals will change, not the components. For *overall reward* calculation e.g. to evaluate the throughput-area trade-offs for a configuration, we use the following equation:

$$Overall\ reward = (1/A) * E[X]$$

In the above equation, A represent the area of the design and $E[X]$ represents the expected throughput. This equation is similar to [20] , however instead of calculating the reward up to a specified time-step, we use the notion of steady-state throughput. Such modeling can be considered as a direct optimization of throughput, area and reliability. Rewards can be weighted based on designer's requirements. In the case study, the rewards are set to equal weight.

4.3 Characterization Library

The reliability of a particular device can be calculated by multiplying the estimated nominal SEU failure rate that is expressed in failure-in-time per megabyte (FIT/Mb) and the number of *critical bits*. A bit that is important for the functionality of a design can be categorized as a *critical bit*. For the analysis of *critical bit*, we follow the procedure from [7]. The components to be analyzed are implemented on Virtex-5 xc5vlx50t device. According to Rosetta experiment [21] and the recent device reliability report [30], a Virtex-5 device has a nominal SEU failure rate of 165 FIT/Mb.

The above failure rate estimation was done for atmospheric environment. At places with high elevation above the sea, the SEU rates can be three or four

Table 1. Characterization library

Component	No. of LUTs	No. of essential bits	MTBF (years)
Wallace tree multiplier	722	133503	9.22
Booth multiplier	650	130781	9.41
Brant-Kung adder	120	29675	41
Kogge-Stone adder	183	41499	30

times higher than at the sea-level. Long-haul aircrafts flying at altitudes near 40,000 ft along flight paths above 60 °C latitude experience the greatest neutron flux of all flights, roughly 500 times that of a ground-based observer in New York City [13]. However, results from the Rosetta experiment [21] for different altitude and latitude shows a worst-case derating factor of 561.70, and hence for commercial avionics applications the worst-case derating factor should be used.

In order to build a characterization library for the first-order estimate of soft error effects, we use the *bitgen* feature of Xilinx ISE tool to identify the *essential bits*, also known as *potentially critical bits*. It is well known that the number of *critical bits* is less than the number of *potentially critical bits*. More accurate SEU susceptibility analysis can be performed using the fault injection techniques [18,22], however, for first-order worst-case estimation, it is a valid assumption that all the *essential bits* are considered as *critical bits*. This is important to mention that we use the characterization library to obtain the failure rate of the components for the CTMC model and the methodology is generic enough to be used with a different characterization library with more precise and accurate data, without any major changes.

Table 1 presents the first-order worst-case estimate of component failures due to soft errors. We characterize different adder and multiplier components, namely 32-bit Brent-kung adder, 32-bit Kogge-stone adder, 32-bit Wallace-tree multiplier and 32-bit Booth multiplier. The Xilinx Synthesis Technology (XST) tool is used to synthesize the components from their HDL codes and the number of required LUTs to implement them is also obtained. We observe that a 32-bit Wallace-tree multiplier has about 0.134 million bits that are sensitive to SEUs. So this multiplier has a worst-case MTBF of 9.22 years for avionic applications.

5 Case Study

To illustrate the applicability of the proposed methodology for early design decision, this section presents a case study from a high-level synthesis benchmark. Figure 5 shows the CDFG for a 16-point FIR Filter [16]. For the experiments, we consider the 32-bit Kogge-stone adders and 32-bit Wallace tree multipliers as available components from the characterization library. To achieve a schedule with minimum number of control steps, the minimum allocation is two adders and two multipliers for the FIR filter application. At a minimum a pair of one adder and one multiplier is required for successful operation. The first part of the

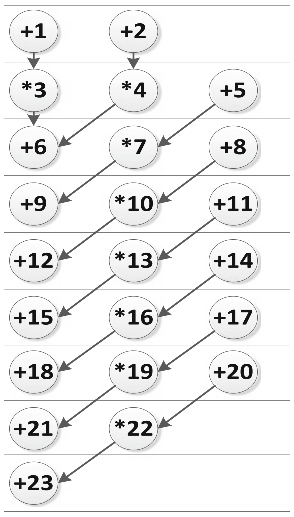

Fig. 5. FIR filter

case study presents the dependability analysis on different configurations. The later part of the case study focuses on the performance-area trade-off analysis using overall reward calculation.

Table 2 shows the statistics and model construction time in PRISM for four different configurations. The first configuration consists of two adders and two multipliers with no redundancy. The second and third configuration consists of one spare multiplier and one spare adder respectively used as redundant components (*coldspare*). Configuration 4 is equipped with full component-level redundancy, with a spare of each type of components. All the four configurations have approximately the same model generation time around 0.002 s. Configuration 4 has maximum number of states and and maximum number of transitions in the generated Markov model.

Probabilistic model checking allows us to reason about the probability of occurrence of an event or of reaching a state within a specific time period, or at any point of time in the lifetime of the system. Such measures can be formalized using CSL as `P = ?` ($\texttt{F}[\texttt{t}_1, \texttt{t}_2]$ "`operational`"), which must be read as follows: *"the probability that the system is operational within the specified*

Table 2. Model construction statistics

Configuration	No. of states	No. of transitions	Time (s)
2A 2M	9	24	0.002
2A 3M	12	34	0.002
3A 2M	12	34	0.002
3A 3M	16	48	0.002

Table 3. Configurations vs classes of states

Configurations	Operational (days)	Degraded (days)	Failed (days)
2A 2M	3212.16	419.81	18.02
2A 3M	3212.16	434.64	3.20
3A 2M	3212.16	421.45	16.39
3A 3M	3212.16	436.28	1.55

time-bound where $[t_1, t_2] \in \mathbb{R}$". In Table 3, we analyze the number of days the design spends in different classes of states for a mission time of 10 years with a scrub rate of 6 months. The first column of the table shows the different configurations for evaluation. The second, third and fourth column presents the number of days the design spends in different classes of states. All the configurations spend approximately similar number of days in *operational state* (rounded to 2 decimal points). Configuration 1 spends around 18 days in *failed state*. Interestingly, we observe that adding an extra adder as spare does not help much whereas adding an extra multiplier as spare significantly reduces the number days spent in *failed state*. In configuration 4, the added spares for both adder and multiplier provide the best result in terms of dependability. This is obvious but will cost more area on the FPGA. Configuration 1 spends the least number of days in degraded state and configuration 4 spends the highest number of days in *degraded state*. For many safety-critical applications, low performance for a period of time is acceptable. For such systems the number of days spent in *failed state* is a major concern and hence, configuration 4 and configuration 2 are the two best candidates.

Choice of scrub rate affects the dependability of the system. Table 4 shows the effects of different scrub rates on configuration 2 for a mission time of 10 years. From the experimental results, we observe that the increase in the scrub rate increases the number of days spent in $failed$ and *degraded* states. Thus, it decreases the number of days spent in *operational* state. For a scrub rate of 10 months, the system spends around 10 days in *failed state* whereas for a scrub rate of 4 months, the design spends only around 1 days in $failed$ state. For a scrub rate of 1 month, the system spend only around 1.5 h in *failed state*. Such an analysis can help designers to choose an effective scrub rate best suited for the application.

In Fig. 6 and Table 5, we compare the four available configurations with respect to different scrub rates to calculate their failure probability for the same mission time. The experimental results show that for configuration 1, the failure probability varies from 0.020 to 0.145. Configuration 2 has a lower failure probability than configuration 3 for all the scrub rates. The failure probability of configuration 4 for all different scrub rates shows the best result with associated extra area overhead.

Steady state analysis of a design is useful to evaluate its dependability in the long-run. The steady-state properties can be formalized using CSL as `S = ? [fail]`, which must be read as follows: *"the long-run non-availability*

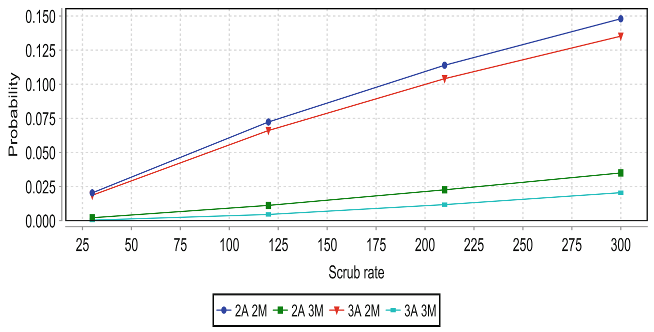

Fig. 6. Failure probability vs scrub rate (days)

Table 4. Scrub rate vs Classes of states

Scrub rate (months)	Operational (days)	Degraded (days)	Failed (days)
1	3567.06	82.87	0.06
4	3343.21	305.49	1.30
7	3151.41	494.09	4.49
10	2985.99	654.35	9.65

Table 5. Scrub rate vs configurations

Scrub rate (Months)	2A 2M	2A 3M	3A 2M	3A 3M
1	0.020	0.002	0.019	3.36E-4
4	0.071	0.011	0.066	0.004
7	0.111	0.022	0.104	0.011
10	0.145	0.035	0.135	0.020

of the system", i.e. the steady-state probability that the system is in *failed state.* The results of steady-state analysis is presented in Table 6 for a scrub rate of 4 months. From the results, we observe that configuration 2 is really an attractive alternative to configuration 4. On the other hand, configuration 1 and configuration 3 offer similar results (rounded to 2 decimal points) over the long-run.

For throughput-area trade-off analysis, Table 7 shows the long-run *overall reward* calculation for the configurations with a scrub rate of 4 months. The rewards are setup so that the area and expected throughput have equal weights. For every configuration, the maximum throughput is used to normalize the throughput for other states in the Markov reward model. Similarly, the maximum area is used to normalize the other area values among different configurations. The normalized long-run expected throughput for each configuration is shown in column 2. Column 3 shows the area of each configuration and their normalized value is shown in column 4. Column 5 shows the overall area-throughput reward for each configuration. The reward for each configuration is calculated

Table 6. Steady state analysis

Class	2A 2M	2A 3M	3A 2M	3A 3M
Fail	0.002	3.86E-4	0.002	1.58E-4
Degraded	0.084	0.086	0.084	0.086
Operational	0.913	0.913	0.913	0.913

Table 7. Overall reward calculation

Configurations	Expected throughput	Area	Normalized area	Overall reward
2A 2M	0.983	1710	0.667	1.46
2A 3M	0.991	2432	0.948	1.04
3A 2M	0.990	1834	0.715	1.39
3A 3M	0.999	2565	1.000	0.99

by multiplying the value of column 2 with the reciprocal of the normalized area. Based on the equal reward weighting, configuration 1, which has no redundancy (spare components), shows the best throughput-area reward. This indicates that the extra reliability provided by the redundancy is not always useful to suppress the extra area overhead. However, rescheduling with scrubbing is good enough to serve as a fault recovery and repair mechanism in such cases. Another important observation is that adding a spare adder significantly improves the throughput-area reward, much more than adding a spare multiplier. It clearly show, how the inclusion of throughput-area metrics can influence design decisions toward solutions that differs from those resulting from an analysis based on a dependability metric alone, as in Table 3. Such an analysis, using the proposed methodology, can be very useful at early design stages for designers of safety-critical applications concerned with dependability, performance and area constraints.

6 Conclusion

This paper illustrated how probabilistic model checking, a formal verification technique which has already been applied to a wide range of domains, can be used to analyze designs at early stage for avionic applications. The design options are modeled using a Markov reward model that captures the possible failures, recoveries and repairs possible in high-altitude radiation environments. Afterwards, a wide range of properties are verified to evaluate the design options, in terms of throughput, area and dependability. Such analysis is useful to reduce the overall design cost and effort. A FIR filter case study demonstrated how the proposed methodology can be applied to drive the design process. Future works include automation of the process to generate the PRISM code for a given configuration and to analyze designs in the presence of other kinds of faults such as Single-Event Functional Interrupts (SEFI).

Acknowledgments. This research was performed as 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 and the Canadian Space Agency (CSA) for their technical guidance and financial support.

References

1. Adell, P., Allen, G., Swift, G., McClure S.: Assessing and mitigating radiation effects in Xilinx SRAM FPGAs. In: 2008 European Conference on Radiation and its Effects on Components and Systems (RADECS), pp. 418–424 (2008)
2. Baier, C., Katoen, J.-P., Hermanns, H.: Approximate symbolic model checking of continuous-time Markov chains (extended abstract). In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, p. 146. Springer, Heidelberg (1999)
3. Beaudry, M.D.: Performance-related reliability measures for computing systems. IEEE Trans. Comput. **C–27**(6), 540–547 (1978)
4. Borgerson, B.R., Freitas, R.F.: A reliability model for gracefully degrading and standby-sparing systems. IEEE Trans. Comput. **24**(5), 517–525 (1975)
5. Salazar, A., Carmichael, C., Caffrey, M.: Correcting single-event upsets through virtex partial configuration (XAPP216 v1.0), Xilinx corporation (2010)
6. Carmichael, C.: Triple module redundancy design techniques for virtex FPGAs (XAPP197 v1.0.1), Xilinx corporation (2006)
7. Chapman, K.: Virtex-5 SEU critical bit information: extending the capability of the virtex-5 SEU controller, Xilinx corporation (2010)
8. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. **8**, 244–263 (1986)
9. Coussy, P., Chavet, C., Bomel, P., Heller, D., Senn, E., Martin, E.: GAUT: a high-level synthesis tool for dsp applications. In: Coussy, P., Morawiec, A. (eds.) High-Level Synthesis, pp. 147–169. Springer, Netherlands (2008)
10. Aigner, G., et al.: The SUIF program representation. http://suif.stanford.edu/suif/suif2/index.html, January 2010
11. Heiner, J., Sellers, B., Wirthlin, M., Kalb, J.: FPGA partial reconfiguration via configuration scrubbing. In: International Conference on Field Programmable Logic and Applications 2009, FPL 2009, pp. 99–104 (2009)
12. Hong, I., Potkonjak, M., Karri, R.: Heterogeneous BISR-approach using system level synthesis flexibility. In: Proceedings of the Asia and South Pacific Design Automation Conference 1998, ASP-DAC '98, pp. 289–294 (1998)
13. Hu, C., Zain, S.: NSEU mitigation in avionics applications (XAPP1073 (v1.0) 17 May 2010), October 2011
14. Huslende, R.: A combined evaluation of performance and reliability for degradable systems. In: Proceedings of the 1981 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pp. 157–164. ACM (1981)
15. ISOGraph. http://www.isograph-software.com
16. Karri, R., Orailoglu, A.: High-level synthesis of fault-secure microarchitectures. In: 30th Conference on Design Automation 1993, pp. 429–433 (1993)
17. Kavi, K.M., Buckles, B.P., Narayan Bhat, U.: A formal definition of data flow graph models. IEEE Trans. Comput. **C–35**(11), 940–948 (1986)

18. Kenterlis, P., Kranitis, N., Paschalis, A.M., Gizopoulos, D., Psarakis, M.: A low-cost SEU fault emulation platform for SRAM-based FPGAs. In: IOLTS, pp. 235–241 (2006)
19. Kumar, V.V., Verma, R., Lach, J., Bechta Dugan, J.: A markov reward model for reliable synchronous dataflow system design. In: 2004 International Conference on Dependable Systems and Networks, pp. 817–825 (2004)
20. Kumar, V.V., Lach, J.: IC modeling for yield-aware design with variable defect rates. In: Proceedings of the Annual Reliability and Maintainability Symposium, 2005, pp. 489–495 (2005)
21. Lesea, A.: Continuing experiments of atmospheric neutron effects on deep submicron integrated circuits (WP286 v1.1), October 2011
22. Mansour, W., Velazco, R.: SEU fault-injection in VHDL-based processors: a case study. J. Electron. Test. **29**(1), 87–94 (2013)
23. Paulin, P.G., Knight, J.P.: Force-directed scheduling for the behavioral synthesis of asics. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **8**(6), 661–679 (1989)
24. PRISM. http://www.prismmodelchecker.org
25. Saleh, A.M., Serrano, J.J., Patel, J.H.: Reliability of scrubbing recovery-techniques for memory systems. IEEE Trans. Reliab. **39**(1), 114–122 (1990)
26. Smith, R.M., Trivedi, K.S., Ramesh, A.V.: Performability analysis: measures, an algorithm, and a case study. IEEE Trans. Comput. **37**(4), 406–417 (1988)
27. Stewart, W.J.: Introduction to the Numerical Solution of Markov Chains. Princeton University Press, Princeton (1994)
28. Thibeault, C., Hariri, Y., Hasan, S.R., Hobeika, C., Savaria, Y., Audet, Y., Tazi, F.Z.: A library-based early soft error sensitivity analysis technique for SRAM-based FPGA design. J. Electron. Test. **29**(4), 457–471 (2013)
29. Tosun, S., Mansouri, N., Arvas, E., Xie, Y.: Reliability-centric high-level synthesis. In: Proceedings of DATE (2005)
30. Device reliability report: Second quarter (UG116 v9.1), Xilinx corporation (2012)