2012 15th Euromicro Conference on Digital System Design

Dependability Analysis of Fault Tolerant Systems Based on Partial Dynamic Reconfiguration Implemented into FPGA

Jan Kastil, Martin Straka, Lukas Miculka, Zdenek Kotasek
Brno University of Technology
Bozetechova 2, 61266 Brno, Czech Republic
{ikastil,strakam,imiculka,kotasek}@fit.vutbr.cz

***Abstract*—In this paper, a dependability analysis of fault tolerant systems implemented into the SRAM-based FPGA is presented. The fault tolerant architectures are based on the redundancy of functional units associated with a concurrent error detection technique which uses the principles of partial dynamic reconfiguration as a recovery mechanism from a fault occurrence. Architectures are tested by injecting soft errors into partial bitstream in FPGA by an SEU injector and the faults coverage of this architecture is obtained. From faults coverage, the failure rate and repair rate are evaluated. Then, for fault tolerant architecture Markov dependability models are created and how the reliability and availability parameters derived from this model for different configurations of architectures and faulty modules is demonstrated. The reliability analysis results are then shown.**

I. Introduction

The robustness and complexity of digital systems have a significant impact on reliability and diagnostic features of these systems. High reliability and availability are important features which are required in various applications of electronic components. It is reported very often that a particular application is implemented as a Fault Tolerant System (FTS) [1]. Fault-tolerance (FT) is an important system metric for many operating environments (Earth or space application) [2].

Digital systems can be implemented on various platforms. From among those which are widely used in many applications, the reconfiguration hardware can be mentioned. Field Programmable Gate Arrays (FPGAs) play an important role among reconfiguration platforms because their function can be very easily reprogrammed by loading new configuration data (bitstream) into the configuration memory [3]. In the FPGA, the combinational and sequential logic are implemented in programmable Complex Logic Blocks (CLBs) which are configured by bitstream data. In order to store the bitstream, many FPGA devices are based on Static Random Access Memory (SRAM). More than 99% of SRAM memory bits on an FPGA are used for storing the configuration data of the FPGA [4].

SRAM-based FPGAs are becoming increasingly popular for space-based applications due to their high-throughput capabilities and relatively low cost. These SRAM-based devices, however, are susceptible to radiation-induced Single Event Upsets (SEUs). An SEU causes the change in the state of a digital memory element caused by an ionizing particle. As the ionizing particle passes through the device, a charge can be transferred from one node to another. This charge transfer can lower the voltage of a memory cell and change its internal state. SEU occurrence in FPGA memory can be seen as a big problem for many digital systems. Therefore, several FT techniques have been proposed and tested for mitigating SEUs in FPGAs [5],[6]. Many FT techniques use hardware redundancy in order to reduce the probability of failure. By replicating the desired circuitry and comparing the results, faults in the configuration can be detected and reported. Other techniques rely on device reconfiguration to continually scrub the configuration bitstream [7]. By repeatedly configuring the device, SEUs occurring within the configuration bitstream are replaced by the correct value.

Duplex systems with various types of Concurent Error Detection (CED) technique are popular FT architecture for SEU mitigation in SRAM-based FPGA [8]. As examples, CED techniques, on-line checkers or dual-rail logic can be mentioned. Many papers were presented on this topic [9],[10].

Triple Modular Redundancy (TMR) is a well known fault mitigation technique that uses redundant hardware to tolerate faults caused by SEUs as well [11],[12]. A circuit protected by TMR has three redundant copies of the original circuit and a majority voter. A single fault in any of the redundant hardware modules will not produce an error at the output as the majority voter will select the correct result from the remaining two correctly working modules. TMR can be combined with recovery techniques as configuration memory scrubbing or Partial Dynamic Reconfiguration (PDR) [13]. The possibility of modifying or reloading configuration memory while the application is correctly working is seen as the main reason why a PDR has become an available feature in FPGA based implementations. The FTS must include a configuration controller to provide the PDR process driving [14].

Reliability evaluation and dependability analysis are important steps in the process of highly reliable and available SRAM-based FPGA systems design. For these purposes, the Markov dependability models can be used [15]. In [16], a mathematical Markov reliability model for SRAM-based FPGAs is presented. From the model, main dependability parameters are evaluated and gained. Many other papers were presented where the process dependability analysis of an

978-0-7695-4798-5/12 $26.00 © 2012 IEEE
DOI 10.1109/DSD.2012.40

CPS Conference Publishing Services

FPGA-based FT design are described or evaluated [17],[18].

In this paper, a reliability and availability analysis for different types of FT architectures implemented into SRAM-based FPGA is presented.

II. Motivation and Goals of the Research

FPGA based designs offer new possibilities for the activities which aim at designing an FT system with a high reliability and availability of the system.

A. Previous Research in FT Systems Design Area

Our previous activities were oriented on creating a new methodology of FT systems design into SRAM-based FPGA platforms where the main principles of PDR were used as the recovery mechanism in situations when an SEU occurs in the implementation. The principles of the methodology together with properties and experiments of several FT architectures were presented in [19]. The FT structure of our methodology can be seen in Figure 1. The methodology allows for the detection and correction of soft errors in the FPGA-based design and the detection of permanent faults in an FPGA structure.

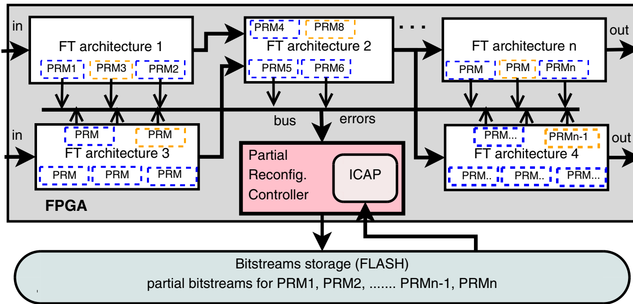

Fig. 1. Fault Tolerant Structure for FPGA Reliability Designs.

Two FT architectures based on Partial Reconfiguration Modules (PRMs) can be seen in Figure 2. The FT architectures using online checkers or other CED techniques for error detection are often reported. On-line checkers can be used for error detection and identification of faulty units. Checkers can be constructed for electronic components on different levels (e.g. module or Register-Transfer Level (RTL) components). An error detected by the on-line checker initiates the reconfiguration process of the faulty PRMs via a special reconfiguration controller (GPDRC - Generic Partial Dynamic Reconfiguration Controller) implemented inside the FPGA. For architecture TMRcmp, a dependability analysis is demonstrated in this paper.

The next goal of our research was to develop an external SEU generator and verify its ability to insert an SEU to the required position in the bitstream. This gives us the opportunity to test the behavior of FT architectures and their reaction to SEUs. The SEU simulation framework allows us to insert multiple SEUs in one run and simulate the occurrence

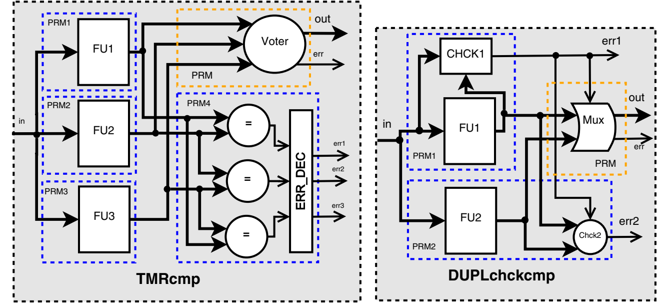

Fig. 2. Fault Tolerant Architectures Based on PRMs for FPGA Designs.

of a higher number of SEUs. The architecture of an SEU simulation framework is shown in Figure 3. The properties of the external SEU generator and experiments with SEU simulation framework were presented in [20]. The results gained from SEU experiments together with investigated FT architectures can be used to compute reliability parameters of dependability models of the methodology. The results from the SEU simulation framework are used in this paper for reliability evaluation of TMRcmp architecture.

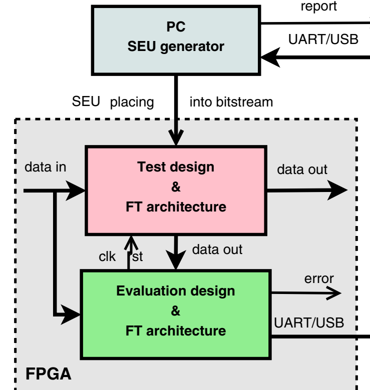

Fig. 3. SEU Simulation Framework for Testing FT Structure.

B. Problem Definition and Goals of the Research

This paper describes the intended following activities of our research based on previous work. In order to verify the ideas of the methodology, a dependability model must be developed. The problem we are solving here is on how to create a dependability model for FT architectures presented above and on how to evaluate their dependabilty parameters. For dependability analysis, Markov models can be used. Finally, the dependability results are compared with other SEU mitigation techniques as TMR or bitstream scrubbing with TMR.

The paper is organized as follows. In Section III, the introduction into system dependability and Markov models together with presenting the principles of reliability parameters evaluation are given. The automatic generation of the Markov dependability model for FPGA-based FT architectures

together with the generation of differential equations and their evaluation by numerical and analytical methods are shown in Section IV. In Section V, the evaluation of the failure rate and repair rate is presented and reliability and availability parameters are derived and summarized for different configurations of architectures and faulty modules. Finally, all experiments together with the goals of our future research are mentioned (Section VI).

III. System Dependability and the Use of Markov models

The dependability analysis of FPGA-based systems are based on several attributes that measure the dependability of a system. The main attributes are reliability, maintainability and availability of the system. These attributes are expressed by probabilistic figures and defined as follows:

- **Reliability R(t)** is defined as the probability that a system produces the correct values at the time t under a given set of operating conditions. High reliability means that long time interval elapses before the first system failure occurs. The expected time for a system to fail is expressed as the Mean Time To Fail (MTTF) parameter. It is a statistical value and the length of the observation interval for the calculation of MTTF must be infinite. Generally stated, MTTF $= 1/\lambda$, where λ is the failure rate parameter.
- **Maintainability M(t)** is defined as the probability of performing a successful repair action within a given time t. The maintainability represents the ease and speed with which a system can be restored to operational status after a failure occurs. High maintainability means a short downtime for the system repair. The expected time for a system to be repaired is the Mean Time To Repair (MTTR) parameter. Generally stated, MTTR $= 1/\mu$, where μ is the repair rate parameter (in FPGA technology usually several miliseconds, based on the size of bitstream and the speed of the interface used for the reconfiguration).
  Mean Time Between Failures (MTBF) is the predicted elapsed time between inherent failures of a system during an operation for the successful repair. MTBF parameter can be calculated as the arithmetic mean (average) time between failures of a system. MTBF = MTTF + MTTR.
- **Availability** expresses the average probability that a system delivers correct values. The availability is a performance criterion for repairable systems that accounts for both the reliability and maintainability properties of a component or system. High availability means a long uptime of the system. As a function of MTBF and MTTR, the availability parameter can be evaluated as follows:

$$Availability = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF + MTTR}$$

The relation between MTTF, MTTR and MTBF parameters and lifetime of FPGA-based system are demostrated in Figure 4.

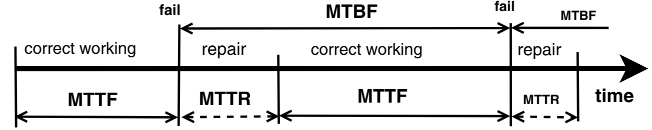

Fig. 4. Lifetime of FPGA-based design, MTTF vs. MTBF.

For evaluation of dependability parameters described above for complex and FT systems, a dependability model must be developed. For these purposes, many different types of dependability models exist. Markov models are very often used for dependability analysis of FT design.

Markov models use state transition diagrams comprised of various possible states of the system and the transitions among various states are described in terms of the rates of transition probabilities. The solution of such a state model using the state space approach then predicts the probability that the system will be in various states after any specified time interval. The sum of these probabilities over non failure states then yields the desired system availability.

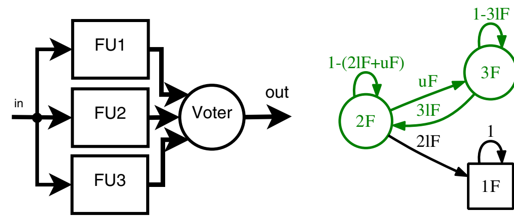

Fig. 5. Markov Model of System using TMR with Scrubbing [21].

The Markov dependability model consists of states, each reflecting the operability (affected by the occurrence of failures) of the system. The states depicted by circles represent ”operational states” while the squares represent the situations when the system does not work correctly. The directed edges between the states are marked either with λ (failure rate) or μ (repair rate).

Figure 5 shows a Markov model of a system using TMR with scrubbing as a repair technique. In this case, State 3F represents the circuit operating correctly. State 2F represents the state in which one of the three FUs of TMR is operating incorrectly. State 1F represents the failure state in which two or more of the three TMR modules are operating incorrectly. SEU can cause a fault in one of the three TMR modules of the circuit and move the circuit from state 3F to state 2F. The probability of failure of one of these domains is equal to the failure rate of the original circuit, λ = lF. The probability of this transition, then, is equal to 3*lF since TMR requires three copies of the original circuit. Configuration scrubbing repairs any faults (at the standard repair rate, μ = uF) that exist in the FPGA and restores the circuit back to State 3F. Alternately, a second SEU could affect another domain and cause a transition from State 2F to State 1F. This transition probability is 2*lF since only two of the redundant modules need be considered. Since the circuit has failed at that point and its output was

incorrect, the model represents no exit from State 1F. Thus the probability of transition from State 1F to State 1F is one [21].

IV. Automatic Generation of Markov Model

The manual construction of Markov models is tedious and error prone work. Many Markov models are required to be generated when it is needed to compare several possible implementations of the system. In order to deal with this problem the software tool for automatic generation of Markov models was developed and the methodology is described in this section. The software tool has as its input the compact description of the architecture and produces the Markov reliability model and dependability evaluation.

Figure 6 shows the flow diagram of the methodology. The implemented design is used as an input into the SEU simulation framework for estimation of reliability parameters, such as λ and μ. The architecture descriptions together with estimated parameters are fed into the Markov Model generator. The generated model is processed by three different output blocks.

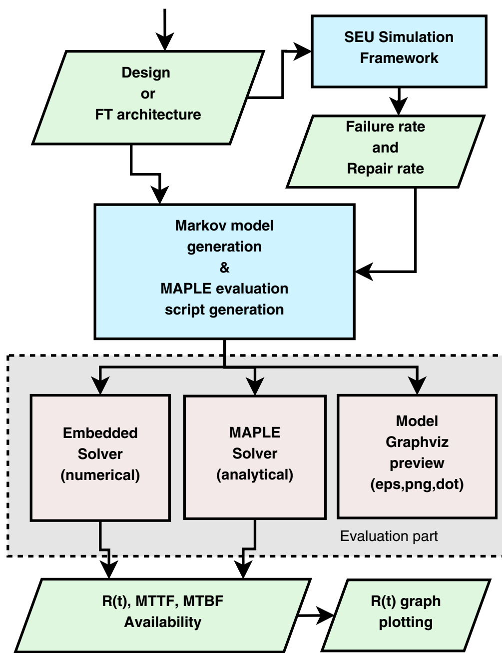

Fig. 6. Dependability Evaluation Flow Diagram.

The architecture is described as a set of various components and evaluation functions. The software generates all possible states of the system simply by simulating error in every component of the system under evaluation. Together with the generation of states, all error transitions are generated.

```
----Components description of FTS-------------
state = dict()
state["FU"] = 3
state["VOTER"] = 1
state["CHECKER"] = 1
state["GPDRC"] = 1
----------------------------------------------

----Example of Repair function----------------
def repair(state):
   if state["GPDRC"] < 1:
      return None
   repState = state.copy()
   label = "R"
   if state["VOTER"] < 1:
      label = label + "V"
      repState["VOTER"] = 1
      return (repState,label)
   if state["FU"] < 3:
      label = label + "FC"
      repState["FU"] = state["FU"] + 1
      repState["CHECKER"] = 1
      return (repState,label)
   ...
---------------------------------------------
```

The user has to specify the evaluation functions which are used to place each error correctly to the transitions and determine the type of state. The evaluation function for determining the type of state is represented by the set of conditions on the number of correctly operating components. If all conditions are satisfied, then the state is considered as working. Otherwise, the state is considered to be a failure state.

The second evaluation function is used for placing repair transitions. It is represented by the ordered list of conditions. The first condition is used for validation on whether the repair is possible. The repair is possible if all functional units required by the repair subsystem are working. In our methodology, the repair subsystem contains only one functional unit which is GPDRC. It is assumed that the repair system works deterministically. Therefore, all repairable components are ordered according to their repair priority. The repair function checks the component with the highest priority and if it is broken, then it simulates the repair process by placing the repair transition into the model. If not, then the repair function continues with the next condition in the list. It is important to keep in mind that components may have several implementations in the system. For example, the functional unit has three implementations in TMR architecture. However, only one implementation will be repaired in one state. The exceptions on this rule are only possible if coupled units (for example, a functional unit and its checker) are supposed to be repaired concurrently. Then, if a checker reports an error, it may be caused by an error in the functional unit or in its checker. Therefore, the repair subsystem may be configured to the repair checker always together with the functional unit. In this case, it is possible to simulate repair for several components concurrently.

A. Markov Model Graph

The graph generation module of the software tool is responsible for creating the humanly readable Markov model graph. It uses Graphviz [22] tool to draw the graph of the generated model. Figure 7 shows an example of the Markov model generated by the tool for TMR architecture with reliable

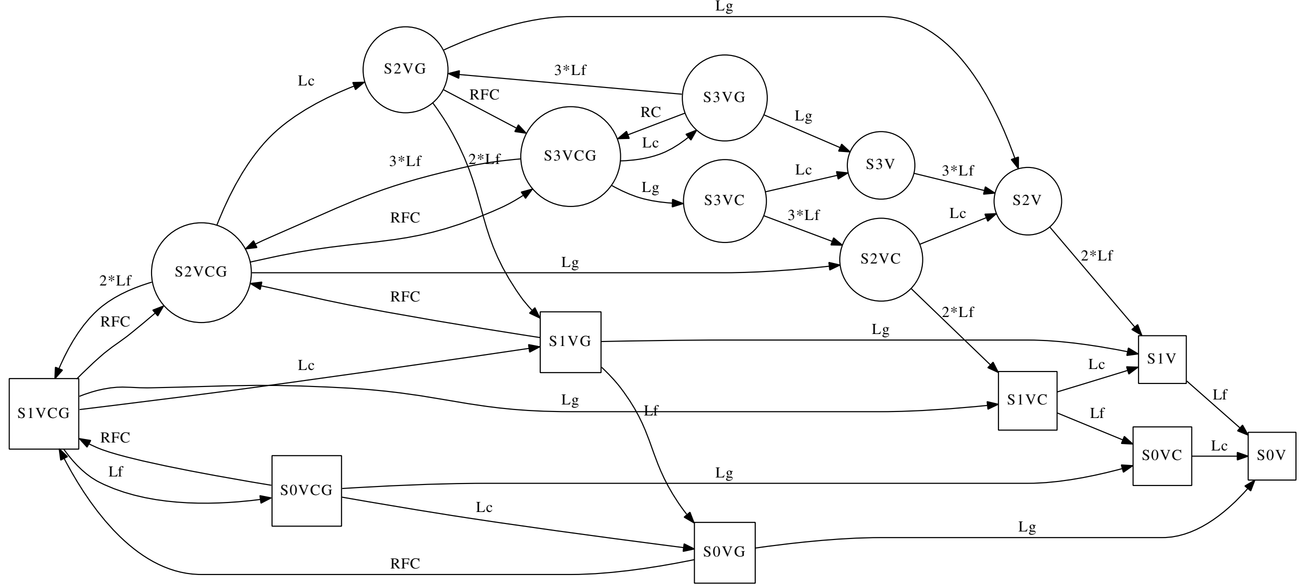

Fig. 7. Markov Reliability Model for TMR with Repair and Reliable Voter.

voter and repair transitions. The names of states are derived from the status of the system. The same state names are used in all output modules. For compatibility reasons, the state name starts with character S. The S is followed by the number of correctly operating functional units. All other units have only one implementation in the architecture. Therefore, they are denoted by the first character of their name. For example, the starting state of the model from figure 7 is S3VCG. It means that the architecture consists of three functional units, a Voter, a Checker and a GPDRC. It can be seen that the model contains all possible error combinations under the assumption that the voter is not vulnerable to SEU attacks. The edges corresponding to failures are labeled by L followed by the symbol of the failing component while repair edges are labeled by R followed by capital first letters of repaired units. For example, the edge RFC represents repair of one functional unit and a checker. It is important to say that such edge connects states S2VCG and S3VCG. Even if the checker is working correctly in both states, it is going to be repaired.This is due to parameters of the repair subsystems which is unable to verify if the checker works correctly with one failing functional unit.

B. Analytical Solution

To evaluate R(t) and MTTF reliability parameters, analytical calculation by means of program MAPLE [23] was used. As its input, the sets of differential equations reflecting the model and λ and μ values gained by fault injection were used. These equations are then solved by Laplace transformation with initial conditions in t = 0 which are equal to 0 for all states with the only exception of the initial state in which the probability that the FT architecture works correctly is equal to 1 (S3VCG state). As the calculations are rather complex, the R(t) and MTTF evaluations for one model can last several minutes. From the Markov model, the following equations can be derived:

$- -$

$pc_0 : S3VCG'(t) = -3Lf * S2VCG(t) - Lg * S3VC(t) + RCF * S2VG(t)...$

$pc_1 : S2VCG'(t) = 3Lf * S3VCG(t) - 2Lf * S1VCG(t) + RFC...$

$...$

$pc_n : S2V'(t) = -2Lf * S1V(t) + Lg * S2VG(t) + Lc * S2VC(t)...$

$- -$

$pf_0 : S1VCG'(t) = 2Lf * S2VCG(t) - RCF * S2VCG(t) - Lf * S0VCG...$

$pf_1 : S0VCG'(t) = -Lf * S0VCG(t) + RCF * S1VCG(t) + Lc * S0VG(t)...$

$...$

$pf_m : S0V'(t) = -\lambda p1(t)$

$- -$

$R(t) = \sum_{i=0}^{n} pc_i(t)$
$Rfail(t) = \sum_{j=0}^{m} pf_j(t)$
$MTTF = \int_0^\infty R(t)dt$

$- -$

C. Numerical Solution

The implemented tool supports a numerical solution of generated models implemented by Scipy python module [24]. One differential equation is generated for every state in the Markov model. The differential equation for each state reflects all incoming and outgoing edges for this state, incoming edges are positive, outgoing edges are negative. The resulting set of differential equation is then solved by the ”zvode” solver.

V. Experimental Results and Discussion

The main issue in the reliability modeling is to know the correct intensity of failures. In FPGA, only a small number of configuration bits is used to represent the given functionality.

According to [25], only 10 – 20 percent of the configuration bits contains information about the design. However, the actual number depends on the type of the implemented system and even on the setting of place and route tools. Therefore, we used the experiments done in [26] to compute the number of bits that will affect the correct behaviour of a simple system.

The first three lines in Table I contain the measured values, while all other lines were gained by an approximation based on the number of LUTs in the design.

The values of λ were computed by means of equation 1 while the values of μ were evaluated by means of equation 2. The important parameter is the failure rate F which is the number of errors occurring within one time unit. This parameter depends on the environment. The value 100 in the equation is the average number of bits that had an effect on the correct function of one LUT in the design tested in [26]. This number is computed as the fraction between all bits that affected the design and the number of LUT in the design. Therefore, the routing configuration bits are also taken into account.

$$\lambda = \frac{\#LUT * 100}{1000000} * F \quad (1)$$

The values of μ are evaluated by means of equation 2. The equation consists of two parts. The first is caller $Preparetime$ which is the time required to prepare the data for reconfiguration process. For example, if the repair subsystem has to deal with permanent error in the FPGA fabric, it uses $Preparetime$ to reroute the unit. The $Preparetime$ for the SEU equals zero. The second part of the equation represents the length of the actual repair process. The limitation of the repair process is the speed of the external reliable memory. The external flash memory used in our experiments works on the 8 MHz frequency and offers an 8-bit interface.

$$\mu = \frac{1}{Preparetime + \frac{|bitstream|}{f_g * buswidth}} \quad (2)$$

The values of λ in Table I are evaluated under the assumption that one Mb of FPGA configuration is affected by 100 faults in one hour which corresponds to the extremely harsh environment. The values of μ and μ_{10} are only different in the size of the $Preparetime$ parameter which was set to zero and ten seconds. The results for ITC benchmark circuits (B4, B12, B14 and B15- I80386)[27] are shown in Table I.

The second experiment compares different realizations of the TMR implementation of the viper processor from the ITC benchmark which is circuit B14. The parameters for the model were established in the previous experiment. The main difference is in the components which may fail under the influence of an SEU. The first experiment setting considers all units vulnerable to SEUs. This setting is denoted as *all fails* in graphs. Other settings consider one type of component invulnerable to SEUs. Such settings are denoted by the name of the invulnerable unit.

The probability of the correct function of all systems during the first two hours of the system operation is shown in Figure 8. The y-axis contains the probability in percentages and x-axis contains time measured in milliseconds. The probability that the system works correctly in the given time is computed as a sum of the probabilities of all working states. It can be seen that the system with never failing GPDRC achieves reliability close to 1 during the whole experiment.

XC5VSX50T TMRcmp	Bitstr. [bits]	# LUT -	λ [ms-1]	μ [ms-1]	μ_{10} [ms-1]
PRM-FU-CNT8	47232	15	4,166E-08	0,1355	9.992E-05
PRM-COMP-CNT8	47232	14	3,888E-08	0,1355	9.992E-05
PRM-VOTER-CNT8	47232	8	2.222E-08	0,1355	9.992E-05
PRM-FU-ITC-B4	236160	165	4,583E-07	0,2710	9.996E-05
PRM-COMP-B4	47232	20	5,555E-08	1,3550	9.999E-05
PRM-VOTER-B4	47232	11	3,055E-08	1,3550	9.999E-05
PRM-FU-ITC-B12	330624	247	6,861E-07	0,1935	9.994E-05
PRM-COMP-B12	47232	11	3,055E-08	1,3550	9.999E-05
PRM-VOTER-B12	47232	6	1,666E-08	1,3550	9.999E-05
PRM-FU-ITC-B14	1464192	1209	3,358E-06	0,0437	9.977E-05
PRM-COMP-B14	47232	112	3,111E-07	1,3550	9.999E-05
PRM-VOTER-B14	47232	54	1,500E-07	1,3550	9.999E-05
PRM-FU-ITC-B15	2172672	1839	5,108E-06	0,0294	9.966E-05
PRM-COMP-B15	47232	154	4,277E-07	1,3550	9.999E-05
PRM-VOTER-B15	47232	70	1,944E-07	1,3550	9.999E-05
PRM-GPDRC-5PRM	330624	266	7,388E-07	–	–

TABLE I
FAILURE RATES AND REPAIR RATES FOR ITC BENCHMARK CIRCUITS BASED ON PRMS.

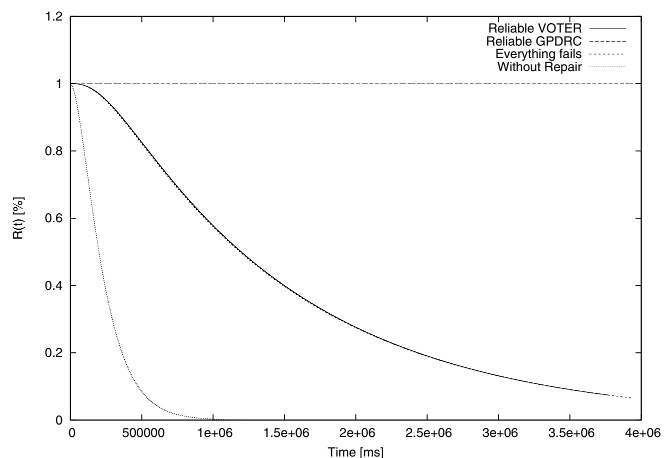

Fig. 8. Reliability of different systems.

The graph further shows that the reliability of the system is significantly decreased if the GPDRC is vulnerable to faults. Therefore, it is very important to implement GPDRC with the highest possible reliability. The *Reliable VOTER* line shows reliability of the system in the case of a voter which is not vulnerable to faults. It can be seen that the reliability of this system is similar to the system where all components may fail. The last line shows the reliability of the system without the repair. The experiment proves that the automatic repair process is a very efficient technique for increasing reliability of the system implemented into FPGA.

Figure 9 shows the difference between the system with invulnerable voter and the system where all units may fail. The graph focuses on the detail at the beginning of the

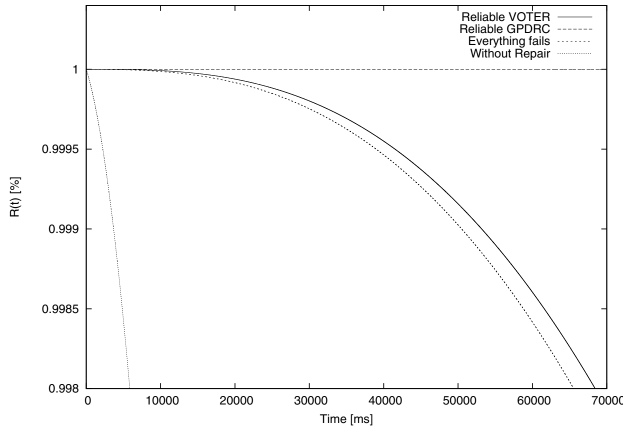

Fig. 9. Detail of the beginning of lifetime with short repair.

system operation. It can be seen that the reliability is higher if the voter is considered to be invulnerable to faults. This difference is the error that is represented by the assumption of an invulnerable voter. In this setup, the voter consists only of 54 LUTS, which is a relatively small number. It is possible to have much larger voters implemented into the FPGA designs. Therefore, it can be concluded that it is not possible to assume invulnerable components in the FPGA reliability analysis.

The previous experiments focused on fast repair. However, if the hard error is taken into account, the repair process may require much more time. Figure 10 shows the detailed graph at the beginning of the system life time if the repair operation takes 10 seconds to be performed. It can be seen that the reliability of the system with invulnerable GPDRC is lower than in the previous one in Figure 9. The main difference with the previous experiment lies in the behaviour of system with an invulnerable voter. At the beginning of the system operation, the voter system achieves its highest reliability.

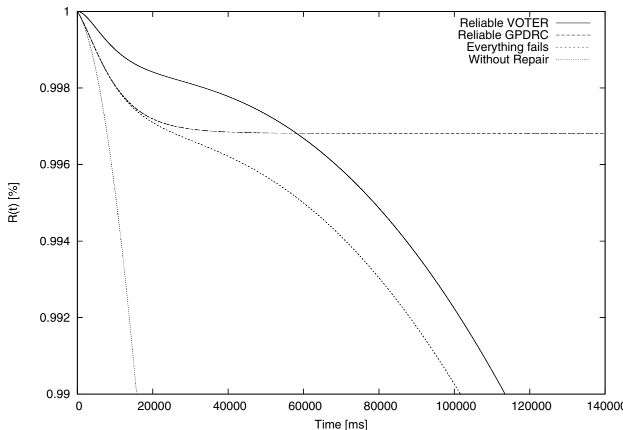

Fig. 10. Detail of the beginning of lifetime with long repair.

VI. Conclusions

This work presents effective methodology for the automatic generation of reliability models for the given architecture. The methodology was applied in order to compute the reliability of TMR based systems with repair mechanisms implemented into FPGA. The presented experiments indicate that the repair process can significantly increase the reliability of the design solution even if the repair mechanisms are vulnerable to faults. The paper presented a comparison with the models that consider some parts of the system invulnerable to faults and concludes that it is not possible to consider any part of the system invulnerable to faults in the FPGA based implementations.

The current implementation of the software tool is restricted only to simple architectures. The main focus of the future work is to extend the tool to support combinations of simple architectures to simulate complex systems, such as several sequentially connected TMRs. This extension will further increase the volume of required computations. Therefore, the research will focus on the possible optimization of the numerical solver implementation and models themselves. The current solution only works with two states of each component and the states are ”working” and ”failing”. However, the fault may affect the component in many different ways. Future research will focus on the effect of the fault by assigning other possible states to the component. For example, the checker may only work correctly for two of the three functional units.

Acknowledgment

This research was supported by the following projects: National COST LD12036 - ”Methodologies for Fault Tolerant Systems Design Development, Implementation and Verification”; RECOMP project - ”Reduced Certification Costs Using Trusted Multi-core Platforms”; research project No.MSM 0021630528 - ”Security-Oriented Research in Information Technology”, GACR No.102/09/H042 - ”Mathematical and Engineering Approaches to Developing Reliable and Secure Concurrent and Distributed Computer Systems” and grant FIT-S-11-1. This work was supported also by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070).

References

[1] J. A. Cheatham, J. M. Emmert, and S. Baumgart, “A survey of fault tolerant methodologies for fpgas,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 11, no. 2, pp. 501–533, 2006.

[2] L. Sterpone, M. Aguirre, J. Tombs, and H. Guzmán-Miranda, “On the design of tunable fault tolerant circuits on sram-based fpgas for safety critical applications,” in *DATE ’08: Proceedings of the conference on Design, automation and test in Europe*. New York, NY, USA: ACM, 2008, pp. 336–341.

[3] U. Sharma, “Fault tolerant techniques for reconfigurable platforms,” in *A2CWiC ’10: Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India*. New York, NY, USA: ACM, 2010, pp. 1–4.

[4] F. L. Kastensmidt, G. Neuberger, L. Carro, and R. Reis, “Designing and testing fault-tolerant techniques for sram-based fpgas,” in *CF ’04: Proceedings of the 1st conference on Computing frontiers*. New York, NY, USA: ACM, 2004, pp. 419–432.

[5] G.-H. Asadi and M. B. Tahoori, “Soft error mitigation for sram-based fpgas,” in *Proceedings of the 23rd IEEE Symposium on VLSI Test*, ser. VTS ’05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 207–212.

[6] B. Osterloh, H. Michalik, S. A. Habinc, and B. Fiethe, “Dynamic partial reconfiguration in space applications,” *Adaptive Hardware and Systems, NASA/ESA Conference on*, vol. 0, pp. 336–343, 2009.

[7] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb, “Fpga partial reconfiguration via configuration scrubbing,” in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, 31 2009-sept. 2 2009, pp. 99 –104.

[8] M. Straka, J. Kastil, and Z. Kotasek., “Modern fault tolerant architectures based on partial dynamic reconfiguration in fpgas,” in *13th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*. New York, NY, USA: IEEE Computer Society, 2010, pp. 336–341.

[9] M. G. Gericota, L. F. Lemos, G. R. Alves, and J. M. Ferreira, “On-line self-healing of circuits implemented on reconfigurable fpgas,” in *IOLTS ’07: Proceedings of the 13th IEEE International On-Line Testing Symposium*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 217–222.

[10] P. Kubalik, R. Dobias, and H. Kubatova, “Dependable design for fpga based on duplex system and reconfiguration,” in *DSD ’06: Proceedings of the 9th EUROMICRO Conference on Digital System Design*, Dubrovnik, Croatia, 2006, pp. 139–145.

[11] C. Bolchini, A. Miele, and M. D. Santambrogio, “Tmr and partial dynamic reconfiguration to mitigate seu faults in fpgas,” in *DFT ’07: Proceedings of the 22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 87–95.

[12] R. Oliveira, A. Jagirdar, and T. J. Chakraborty, “A tmr scheme for seu mitigation in scan flip-flops,” in *ISQED ’07: Proceedings of the 8th International Symposium on Quality Electronic Design*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 905–910.

[13] C. Pilotto, J. R. Azambuja, and F. L. Kastensmidt, “Synchronizing triple modular redundant designs in dynamic partial reconfiguration applications,” in *SBCCI ’08: Proceedings of the 21st annual symposium on Integrated circuits and system design*. New York, NY, USA: ACM, 2008, pp. 199–204.

[14] X. Iturbe, M. Azkarate, I. Martinez, J. Perez, and A. Astarloa, “A novel seu, mbu and she handling strategy for xilinx virtex-4 fpgas,” in *International Conference on Field Programmable Logic and Applications, 2009. FPL 2009.* Washington, DC, USA: IEEE Computer Society, 2009, pp. 569–573.

[15] R. Noji, S. Fujie, Y. Yoshikawa, H. Ichihara, and T. Inoue, “Reliability and performance analysis of fpga-based fault tolerant system,” in *Proceedings of the 2009 24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, ser. DFT ’09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 245–253.

[16] P. Kubalik, R. Dobias, and H. Kubatova, “Dependability computation for fault tolerant reconfigurable duplex system,” in *DDECS ’06: Proceedings of the 2006 IEEE Design and Diagnostics of Electronic Circuits and systems*, Prague, Czech Republic, 2006, pp. 98–100.

[17] O. Heron, T. Arnaout, and H.-J. Wunderlich, “On the reliability evaluation of sram-based fpga designs,” in *Field Programmable Logic and Applications, 2005. International Conference on*, aug. 2005, pp. 403 – 408.

[18] P. Ostler, M. Caffrey, D. Gibelyou, P. Graham, K. Morgan, B. Pratt, H. Quinn, and M. Wirthlin, “Sram fpga reliability analysis for harsh radiation environments,” *Nuclear Science, IEEE Transactions on*, vol. 56, no. 6, pp. 3519 –3526, dec. 2009.

[19] M. Straka, J. Kastil, and Z. Kotasek, “Fault tolerant structure for sram-based fpga via partial dynamic reconfiguration,” in *13th EUROMICRO Conference on Digital System Design DSD 2010*. Washington, DC, USA: IEEE Computer Society, 2010.

[20] M. Straka, J. Kastil, and Z. Kotasek, “Seu simulation framework for xilinx fpga: First step towards testing fault tolerant systems,” in *Digital System Design (DSD), 2011 14th Euromicro Conference on*, 31 2011-sept. 2 2011, pp. 223 –230.

[21] B. Pratt, M. Caffrey, D. Gibelyou, P. Graham, K. Morgan, , and M. Wirthlin, “Tmr with more frequent voting for improved fpga reliability.” in *The International Conference on Engineering of Reconfigurable Systems and Algorithms*, New York, NY, USA: IEEE Computer Society, 2008, pp. 1–6.

[22] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull, “Graphviz and dynagraph static and dynamic graph drawing tools,” in *GRAPH DRAWING SOFTWARE*. Springer-Verlag, 2003, pp. 127–148.

[23] W. M. Inc., “Maple 13,” 2008. [Online]. Available: http://www.maple.com/

[24] E. Jones, T. Oliphant, P. Peterson *et al.*, “SciPy: Open source scientific tools for Python,” 2001–. [Online]. Available: http://www.scipy.org/

[25] XILINX, “Xapp864: Seu strategies for virtex-5 devices.” [Online]. Available: www.xilinx.com

[26] M. Straka, L. Miculka, J. Kastil, and Z. Kotasek, “Test platform for fault tolerant systems design properties verification,” in *15th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*. New York, NY, USA: IEEE Computer Society, 2012, pp. 1–6.

[27] S. Davidson, “Itc99 benchmark,” 1998. [Online]. Available: http://www.cerc.utexas.edu/itc99-benchmarks/bendoc1.html