How to Measure Dependability Parameters of Programmable Digital Circuits – A Survey

Jaroslav Borecký, Martin Kohlík, and Hana Kubátová

Department of Digital Design
Czech Technical University in Prague
Prague, Czech Republic
{borecjar; kohlimar; hana.kubatova}@fit.cvut.cz

Abstract. Our aim is to create a methodology for FPGA industrial applications with respect to area, speed, power consumption and reliability optimizations (both fail safe and fault-tolerant). We take into account different types of faults, the way they affect the circuit (Single Event Upset, Single Event Latchup, Delay faults etc.) and their injection into design (insertion into bitstream, edif, behavioral description or saboteur method). We need to create formal dependability models that are able to model mentioned faults and reconfiguration ability of FPGAs. We use well-known Markov Chains and Stochastic Petri nets. The usage of both types of models is similar and they are mutually convertible. This paper describes the main problems how to obtain relevant and comparable results.

1 Introduction

During the last decade, systems realized by programmable hardware like Field Programmable Gate Arrays (FPGAs) are more and more popular and widely used in more and more applications due to their capability of implementing complex circuitry within a very short development time, together with the potential for easy reconfiguration or for other actual changes of the implemented circuit. Different types of faults (Single Event Upset, Single Event Latchup, Delay fault etc.) can arise in these FPGAs and if we would use these FPGAs in mission critical application, we must secure design implemented inside.

This paper presents our actual problem: how to design really dependable system based on programmable and reconfigurable hardware (FPGA).

This means not only the design methodology but moreover: how to perform relevant experiments, how to model obtained results by good (and compound) model of reliability, structure and function and how to obtain relevant and comparable results and how to compute relevant dependability parameters on the requested level of preciseness.

Therefore the paper presents our solution of following compound problems: description of possible faults in FPGA, their relevant models and the possible impact to whole design according their classification, the possible methods how to inject these faults into our design and how to obtain relevant values to model and compute dependability parameters. Main aim of this paper is to summarize our actual problem – how to obtain relevant results and how to compare them with others and finally how to construct a good and usable methodology for dependable design for FPGA based systems.

The paper is structured as follows: Fault classification is in Section 2. Section 3 contains the overview of the methods of faults injection into design. Section 4 shows problems that may occur when results from different authors are being compared. Section 5 provides some recommendations how to define unified conditions for reliability calculations that will be necessary to compare results among different research groups.

2 Faults in FPGA

A fault changes a bit in the programming memory of the FPGA. The meaning of this bit depends on the set of the resources of the FPGA and it is determined only by the FPGA architecture. However, the distribution of the faults into following fault groups is defined in the process of the design.

Every fault belongs to one fault group. The fault bit is placed in *Used, Unused* or *Unknown* group in the first approximation.

1. The *unused* bits are outside the design area. Neither static nor dynamic changes of design behavior are expected.
2. *Used* bits are primary created by the design. Any bit from this group influences an active part of this design. A change can lead to the change of the design function. A detailed distinction according to the used category follows:
   (a) Open: represents a wire interruption.
   (b) Alternate: bits alter the design without any conflict on the bus.
   (c) Conflict: is defined by connecting of two or more driven wires. This conflict leads to a short. The conflict can be separated into 2 subcategories:
      – "F–F" (Function–Function), where conflict is between two non-constant functions
      – "0–F" (Zero–Function), where any function conflicts with constant "0"
   (d) Unpredictable: A special case of open, where the change of a bit leads to a change of a selector (transfer gate or mux) from a constant logic value ("1" or "0") to an unconnected wire ("Z").
   (e) Antenna: an unused wire is connected to the data path. This fault leads to worse delays on wires.
3. The *unknown* bits form a class for bits, whose correct class cannot be evaluated. The position in the bitstream implies their usage, but their exact role in the bitstream has not been completely analyzed. Parts of RAM, reset and clock resources are expected to belong in this category. These bits form up to 5% of the bitstream.

These groups are defined in [1] and they are based on the real experiments with the bitstream of FPSLIC [2] device. Each bit of bitstream has been tested.

2.1 Fault Security Calculation

Some redundancy has to be incorporated into the circuit design to improve reliability parameters. We have performed experiments with online testing to obtain Concurrent Error Detection (CED) [3]. Basic reliability criteria are: fault security (FS), self-testing (ST), totally self-checking (TSC), availability, reliability, testability, mean time between failures (MTBF), mean time to repair (MTTR), etc.

To determine whether the circuit satisfies the TSC property, the possible faults are classified and separated into four classes, A, B, C and D [4] according to their impact on the tested circuit design in the FPGA.

Class A – hidden faults. These are faults that do not affect the circuit output for any allowed input vector. Faults belonging to this class have no impact to the FS property, but if this fault can occur, a circuit cannot be ST.

Class B – faults detectable by at least one input vector. They do not produce an incorrect codeword (valid code word, but incorrect one) for other input vectors. These faults have no negative impact to the FS and ST property.

Class C – faults that cause an incorrect codeword for at least one input vector. They are not detectable by any other input vector. Faults from this class cause undetectable errors. If any fault in a circuit belongs to this class, the circuit is neither FS, nor ST.

Class D – faults that cause an undetectable error for at least one vector and a detectable error for at least one another vector. Although these faults are detectable, they do not satisfy the FS property and so they are also undesirable.

This fault classification is obtained subsequently. A fault from the fault set is inserted into the design. If the check of an output-word code fails, the input word is added to the "test" group. If the comparison of the outputs fails, but the check of the code passes, the input word is added to the "error" group. When all input words are tested, the fault class is determined.

The classification of fault is based on the size of "test" and "error" groups is described in Table 1. Each fault from the fault set is tested. FS and ST property is calculated from numbers of faults and after that TSC property is calculated.

Table 1. Fault Classification

Fault class	"test" group	"error" group
A	empty	empty
B	non-empty	empty
C	empty	non-empty
D	non-empty	non-empty

3 Fault injection methods

The measurement of the behavior of a real designed system or a benchmark circuit (Design Under Test – DUT) affected by fault is required for reliability parameters calculations. The international safety standard IEC-61508 highly recommends fault injection techniques in all steps of the development process in order to analyze the reaction of the system in a faulty environment. Simulated fault injection enables an early dependability assessment that reduces the risk of late discovery of safety related design pitfalls and enables the analysis of fault tolerance mechanisms at each design refinement step [5].

Faults can be injected into design or FPGA chip directly.

3.1 Injection into design

Injecting faults into the DUT requires a fault simulation or emulation to obtain the reaction of the affected design. The injection can be made in the many ways at the different levels of description: Gate insertion, Stuck-at fault insertion, Value change in memory parts, Bitstream change.

A fault insertion by the means of a gate insertion [6, 7] changes the DUT, but it may be the only way how to insert a fault into the design in some cases. The fault simulation can be done by connecting one input of the inserted gate to a primary input of the DUT that allows the change of the value of the output of the inserted gate. The appearance of the fault is controlled by the primary input, so no special test simulation is required.

Stuck-at fault insertion is widely used [8–10]. It can be performed at many levels (behavioral description, RTL, netlist). This method requires the adaptation of a signal carrier (wire) in all cases. The adapted signal carrier must be able to preserve a test value ignoring any operational value of the DUT.

Memories are one of the most significant parts of each FPGA. The fault injection into these parts flips the value of affected bit, so this method is similar to a radiation impact effect. The faults can be injected into Flip-flops [11] or LUTs. The direct access to the LUT memory is available when the DUT is mapped into FPGA primitives.

Bitstream change method is the most close to radiation effects, but the structure of the bitstream used to program the FPGA is unfortunately mostly unknown. The blind test of each bit may be dangerous because it may cause short-circuit inside the FPGA. The bitstreams of some FPGAs have been partially decoded, so the fault injection into the decoded parts can be done [12–14].

3.2 Injection into FPGA chip

These methods suppose to use a real FPGA with the DUT implemented inside. It is necessary to measure the impacts of injected faults. The measurement has to be done by external equipment and/or by an unaffected part of the FPGA.

The injection of the faults into the FPGA can be done by Radiation or Laser beam exposition.

The exposition of the FPGA to radiation effects is the only possibility to obtain the real data based on the negative effects of environment [15, 16]. This kind of experiments can be done e.g. on the orbit or in an ionising chamber. These enviroments are not commonly available, so experiments are hard to perform. Another drawback of these methods is the limited control over the experiment.

The physics of an optical pulsed beam are different from the high-energy radiation, but both effects are similar [17]. A short laser beam burst is able to cause the SEU effect in the same way as a natural radiation. The laser beam can be controlled more precisely than the radiation, therefore it is possible to control the location and the time point of the injected fault. This method of injection requires a chip package to be opened in order to minimize the diffusion of a laser light.

4 Difficulties with experiments comparison

Many papers show experimental results in the area of the dependable design, but there are problems with the comparison of obtained results. Many authors use standard ISCAS benchmarks that are provided in a *bench* format. The *bench* format is close to a netlist representation, but the benchmarks are available also in VHDL or Verilog formats. There are no rules about processing these benchmarks. Published results are difficult not only to compare but also to verify.

Important processing options influencing the concrete system design are listed below:

- Synthesis tool
- Synthesis options and optimization
- Target architecture
- Place and route options

4.1 Synthesis tool, its options and optimization

Many papers do not specify, which benchmark format is used. Mostly it cannot be determined, whether a synthesis step has been even executed or not. There is no doubt that the different synthesis tools create different results (e.g. results in [18]).

The usage of the same tool does not guarantee the same result. Different optimization techniques (FSM encoding, area, speed etc.) cause the significant changes of the results. Different versions of the same tool may also cause the changes of the results.

4.2 Target architecture

The selected target device has a major impact on the result design, too. Vendors improve their devices all the time and each new family of devices comes with new capabilities. The number of LUT inputs is changed, the modifications of basic blocks are made and new specialized cores are added. These improvements may lead to the changes of the tested circuit at the netlist level (e.g. results in [19]).

The process technology is also an important attribute. It does not change the tested circuit at the netlist level, but it has a significant influence to the sensitivity to the radiation effects, so it must be taken into account in the dependability model and the reliability parameters calculations [20].

4.3 Place and route

The fault insertion into a bitstream requires the Place and Route (P&R) step to be done. The parameters of the P&R (optimization goal, effort level, constrains etc.) can influence the post-P&R design (e.g. results in [21]).

Moreover, a random algorithm is used, therefore the results cannot be repeated even with the same settings. The results of the research previously made in our department show that the multiple runs of the P&R with the same settings provide different results [14].

4.4 Dependability modeling

Formal dependability models are necessary to calculate the level of reliability of a modeled system. Markov chains [22] are well-known and widely used for reliability calculations, but there are more models available.

Markov chains were originally proposed by the Russian mathematician Markov in 1907. Over the many decades since, they have been extensively applied to problems in social science, economics and finance, Computer science, computer-generated music, and other fields.

The availability parameter of the modeled system can be calculated as the steady-state distribution of the probabilities of Markov chain. This distribution can be obtained as the solution of the system of linear equations.

Stochastic Petri nets (SPNs) [23] can be created as easy as Markov chains and allow the same calculations. Mathematical properties, that are available for Petri nets, can be used to analyze SPNs. Moreover, SPNs are not only dependability models, they are able to represent the structure of the design.

Generalized Stochastic Petri nets (GSPNs) [23] have two different classes of transitions: immediate transitions and timed transitions. Once enabled, immediate transitions fire in zero time. Timed transitions fire after a random, exponentially distributed enabling time as in the case of SPNs.

Our results with more details about models relations and comparison can be found in [24].

5 Conclusions and recommendations

Main aim of this paper is to open the problems that we have to solve during the development of the methodology of the dependable systems based on the programmable hardware (FPGAs) in our department. The real experimental results can be obtained by many ways so they cannot be neither replicated nor compared among different research teams.

But there could be a solution of this problem. It is necessary to create the database of benchmarks that will include more levels of descriptions for each of them. Each benchmark will be available in following descriptions: Behavioral, RTL, Netlist, Post-Map, Post-Place and Route, Bitstream.

Users will be able to choose the benchmark at the particular level, which they need for their calculations. User made conversions among the listed levels should be allowed only when the target level will be not available. Users will be able to add their own modifications of a benchmark (e.g. original design with parity check, duplicated design, different architecture implementation etc.) that will be available to other users. The database of this type could allow the correct comparison of the different methods of the reliability modifications and/or the reliability parameters calculations. Authors should be able to reference exact benchmark, the used modification and level in their papers. Results presented in future papers using this database will be easy to reproduce, verify and/or compare using identical benchmark designs and conditions.

Acknowledgment

This research has been partially supported by the projects MSM6840770014, GA102/09/1668 and SGS10/118/OHK3/1T/18.

References

1. Kvasnička, J. and Kubátová, H.: *"Emulation of SEU Effect In Bitstream of FPGA"*, In 4th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science. Brno, 2008, p. 140–147.
2. Atmel Corporation. ATMEL FPSLIC webpage. `http://www.atmel.com/dyn/products/devices.asp?family_id=627`,
3. Pradhan, D.K.: *"Fault-Tolerant Computer System Design"*, Prentice-Hall, Inc., New Jersey, 1996.
4. Kubalík, P.; Fišer, P. and Kubátová, H.: *"Fault Tolerant System Design Method Based on Self-Checking Circuits"*, Proc. 12th International On-Line Testing Symposium 2006 (IOLTS'06), 2006.
5. Perez, J.; Azkarate-askasua, M. and Perez, A.: *"Codesign and Simulated Fault Injection of Safety-Critical Embedded Systems Using SystemC"*, In Proceedings of the IEEE European Dependable Computing Conference (EDCC), 2010, p. 221–229
6. Aftabjahani, S.A. and Navabi, Z.: *"Functional fault simulation of VHDL gate level models"*, In Proceedings of the VHDL International Users' Forum, 1997, p. 18–23.
7. Civera, P.; Macchiarulo, L.; Rebaudengo, M.; Reorda, M.S. and Violante, M.: *"Exploiting circuit emulation for fast hardness evaluation"*, IEEE Transactions on Nuclear Science (Part 1), 2001, p. 2210–2216.
8. Zarandi, H.R.; Miremadi, S.G.; Ejlali, A.: *"Fault injection into verilog models for dependability evaluation of digital systems"*, In Proceedings of the 2nd International Symposium on Parallel and Distributed Computing (ISPDC), 2003, p. 281–287.

9. Na, J.: *"A Novel Simulation Fault Injection using Electronic Systems Level Simulation Models"*, IEEE Design & Test of Computers, Iss. 99, 2009.
10. Boue, J.; Petillon, P. and Crouzet, Y.: *"MEFISTO-L: A VHDL-based fault injection tool for the experimental assessment of fault tolerance"*, In Proceedings of the 28th Annual International Symposium on Fault-Tolerant Computing, Digest of Papers, 1998, p. 168–173.
11. Espinosa-Duran, J.M.; Trujillo-Olaya, V.; Velasco-Medina, J. and Velazco, R.: *"Bit-flip injection strategies for FSMs modeled in VHDL behavioral level"*, In Proceedings of the 11th Latin American Test Workshop (LATW), 2010, p. 1–5
12. Sterpone, L.; Violante, M. and Sterpone, L.: *"A New Partial Reconfiguration-Based Fault-Injection System to Evaluate SEU Effects in SRAM-Based FPGAs"*, IEEE Transactions on Nuclear Science, Part 2, 2007, p. 965–970.
13. Legat, U.; Biasizzo, A. and Novak, F.: *"Automated SEU fault emulation using partial FPGA reconfiguration"*, In Proceedings of the IEEE 13th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2010, p. 24–27.
14. Kvasnička, J. and Kubátová, H.: *"Single Event Upset Tolerant FPGA Design"*, In Proceedings of the Work in Progress Session SEAA 2009 and DSD 2009, 2009, p. 37–38.
15. Karlsson, J.; Liden, P.; Dahlgren, P.; Johansson, R. and Gunneflo, U.: *"Using heavy-ion radiation to validate fault-handling mechanisms"*, IEEE Micro, Vol. 14, Iss. 1, 1994, p. 8–23.
16. Ceschia, M.; Bellato, M. and Paccagnella, A.: *"Heavy Ion Induced See in Sram Based FPGAs"*, Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation, book from Frontiers in Electronic Testing series, Vol. 23, Part 2, 2003, p. 95–107.
17. Ceschia, M.; Bellato, M. and Paccagnella, A.: *"Investigation of SEU sensitivity of Xilinx Virtex II FPGA by pulsed laser fault injections"*, In Proceedings of the 15th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis, Vol. 44, Iss. 9–11, 2004, p. 1709–1714.
18. Czajkowski, T.S. and Brown, S.D.: *"Functionally Linear Decomposition and Synthesis of Logic Circuits for FPGAs"*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 27, Iss. 12, 2008, p. 2236–2249
19. Mishchenko, A.; Chatterjee, S. and Brayton, R.: *"DAG-aware AIG rewriting: a fresh look at combinational logic synthesis"*, In Proceedings of the 43rd ACM/IEEE Design Automation Conference, 2006, p. 535–535.
20. Ohlsson, M.; Dyreklev, P.; Johansson, K. and Alfke, P.: *"Neutron SEU In SRAM-Based FPGAs"*, Xilinx Appnotes www.xilinx.com/appnotes/FPGA_NSREC98.pdf
21. Sterpone, L. and Violante, M.: *"A New Reliability-Oriented Place and Route Algorithm for SRAM-Based FPGAs"*, IEEE Transactions on Computers, Vol. 55, Iss. 6, 2006, p. 732–744.
22. *"Empirical Techniques in Finance – Chapter: Basic Probability Theory and Markov Chains"*, Springer Berlin Heidelberg, 2006, p. 5–17.
23. Bause, F. and Kritzinger, P.: *"Stochastic Petri Nets"* An Introduction to the Theory (2nd edition). Vieweg Verlag, Germany.
24. Kohlík, M.: *"Dependability models based on Petri nets and Markov chains"*, In Počítačové architektury & diagnostika. Soláň: Universita Tomáše Bati ve Zlíně, 2009, p. 95–103.