Extended Reliability Analysis of Fault-Tolerant FPGA-based Robot Controller

Jakub Podivinsky, Jakub Lojda, Zdenek Kotasek
Faculty of Information Technology, Brno University of Technology, Centre of Excellence IT4Innovations
Bozetechova 2, 612 66 Brno, Czech Republic
Tel.: +420 54114-{1361, 1360, 1223}
Email: {ipodivinsky, ilojda, kotasek}@fit.vutbr.cz

***Abstract*—The reliability of safety-critical systems is very important especially in case of electronic systems which are working in environment with increased occurrence of faults. As an example, space, aerospace or medical systems can serve. Fault tolerance is one of the techniques the goal of which is to avoid the impact of faults on such systems. Lots of fault tolerance techniques exists and new ones are under investigation. This paper is targeted mainly to *Field Programmable Gate Arrays* (FPGAs) which are also the target technology of many fault tolerant techniques. It is important to evaluate and test these techniques. This paper is the continuation of our previously published research results which presents experimental approach to evaluate such fault tolerance techniques by monitoring the impact of faults in the experimental electro-mechanical system utilizing robot navigation in a maze. However, in this paper, we research and compare similarities of the theoretical estimation to various methods of the SEU injection approaches. The theoretical estimation is calculated using known equations. The impact of artificially faults injected into the electronic controller, in which Triple Modular Redundancy is applied, is monitored and used for statistic reliability analysis. This approach serves as a tool for the fast reliability evaluation during the development process of fault tolerance systems.**

***Keywords*—Reliability Analysis, TMR, FPGA, Fault Tolerance, Robot Controller.**

I. Introduction

The reliability of safety-critical electronic systems which are working in environment with increased occurrence of faults is a very challenging topic. A technique called *fault tolerance* [1] is commonly used technique which makes electronic systems more reliable. The goal of this approach is to keep the system functional, even in the presence of faults. It means that fault tolerance accepts the fact a fault can appear in electronic system. Various types of redundancy are the core of such techniques. Hardware and time redundancy are the most common ones. Combination and improvements of these basic methods are still under investigation, e.g. authors of [2] present approach which is based on the combination of hardware and time redundancies.

Many fault-tolerant methodologies targeted to *Field Programmable Gate Arrays* (FPGAs) have been developed and new ones are under investigation [3]. The main reason is that FPGAs are more popular thanks to their flexibility and ability to be reconfigured in case of fault occurrence. Sensitivity of FPGAs to faults caused by charged particles [4] is the problem from the reliability point of view. The configuration of FPGA is stored as a *bitstream* in SRAM memory and charged particle can cause inversion of bit in the *bitstream*. This event is called *Single Event Upset* (SEU) [5].

978-1-7281-1756-0/19/$31.00

A fault-tolerant system development usually starts with a *nondurable* system that does not tolerate faults [6]. This *nondurable* system is usually designed with minimum redundancy and serves as a starting point for the process of hardening against faults. Then the modifications that should be made to the *nondurable* system in order to achieve a higher level of fault tolerance are proposed by an experienced fault-tolerant system designer. After integration of proposed changes into the design, the system must be evaluated to ensure that the applied changes have the expected impact on the reliability of the system. The iteration between the phase of development and the reliability evaluation is the usual approach. Multiple designs with various combinations of fault tolerance methods assigned to the partitions of the design are created. Development ends if two conditions are met: 1) the system complying with the specification or 2) the findings of the specifications not being achievable. We to accelerate this procedure of the development with the capability to evaluate the estimation of reliability of the resulting system even *before* the integration of the method itself. This allows a designer to exclude such combinations of reliability methods that do not look perspective.

This work is the continuation of our previously published paper. Additional experiments were done and experimental results were compared with results obtained in our previous publication [7]. This paper is organized as follows. Section II described reliability analysis and reliability improvement. The experimental platform which allow us to done experimental evaluation on real FPGA is introduced in Section III. Section IV presents reliability analysis and its experimental evaluation. Section V concludes the paper and mentions plans for our future research.

II. Reliability Analysis and its Improvement

The reliability itself can be quantified with the support of the *theory of probability* as most of the *reliability indicators* are of a random nature. The length of a time period of the system operation until the failure occurs is an important starting point in the *reliability indicators* computation.

1) Failure Function: If a *random variable* τ expresses a length of a time interval from the systems start of the operation to the point a fault occurs, then the *Cumulative Distribution Function* (CDF) [8] $F(t)$ of *random variable* τ expresses a probability of the system being in a failure state at the time t. In this case, the CDF $F(t)$ is denoted as $Q(t)$ and is called the *failure function*.

2) Reliability Function: Another *reliability indicator* is the so-called *reliability function* which is denoted as $R(t)$. The

reliability function expresses a probability of the system being in an fault-less state at the time t and it is a supplement of the $Q(t)$ as expressed in Equation 1.

$$R(t) = 1 - Q(t) \tag{1}$$

3) Failure Density: The *failure density* $f(t)$ is defined by the time derivative of a CDF $Q(t)$ if the *random variable* [8] is continuous and the derivative exists, as shown in Equation 2.

$$f(t) = \frac{\mathrm{d}Q(t)}{\mathrm{d}t} \tag{2}$$

The product of $f(t)\mathrm{d}t$ then expresses the probability of a fault occurrence for a short period of time $\mathrm{d}t$ that is immediately following after the time t. Although, the case in which the fault occurred earlier before the time t is not taken into account.

4) Failure Rate: The next *reliability indicator* is *failure rate* which is denoted by $\lambda(t)$. The *failure rate* expresses a conditional *failure density* at the time t assuming the failure has not occurred yet. Equation 3 gives a relationship between the $\lambda(t)$ and $Q(t)$.

$$\lambda(t) = \frac{f(t)}{R(t)} = \frac{f(t)}{1 - Q(t)} \tag{3}$$

5) Mean Time To Failure: The *Mean Time To Failure* (MTTF) which is in the following text denoted as T_s represents a mean value of the random variable τ observed. The mean value can be seen as a mean time of all the time period lengths since the system started its operation to the first failure occurrence. If the mentioned system is *non-recoverable*, the value can be considered a *mean time to the first failure* as well. To calculate the T_s, the Equation 4 can be used.

$$T_s = \int_0^\infty R(t)\mathrm{d}t \tag{4}$$

The reliability improvement [9] can be done by several techniques, lots of them are based on *redundancy*. *Triple Modular Redundancy* (TMR, 3MR) which is based on a *triplication* of the component is the most known application of the *hardware redundancy*. In this paper, we plan to analyze the reliability improvement just for TMR. The TMR is based on using of three equivalent functional units, Figure 1 shows the structural schematic. The TMR system is composed of original functional unit and two additional copies of the same functional unit, which are labeled F_1, F_2 and F_3. The input signals x are connected as an input for each of the functional units F_i. The output signals $f_i(x)$ are connected to the *voter* unit which implements *majority function*. The *majority function* can be performed on the level of bits, whole vector, etc. It should be noted, that *voter* used in this paper operates on the per-bit basis.

If we are not taking into account the corner cases, TMR by its nature allow us to mask the failure of one module. Assuming that all F_i units have the same reliability function $R(t)$, then Equation 5 can be used to calculate the reliability function of the whole TMR module. An overview of the reliability indicators of the system with TMR implemented is given in Table I [6], [9].

$$R_{TMR}(t) = 3[R(t)]^2 - 2[R(t)]^3 \tag{5}$$

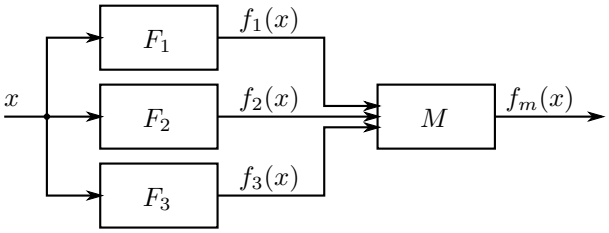

Fig. 1. The schematic representation of the TMR method.

TABLE I. TWO MAIN RELIABILITY INDICATORS OF THE TMR SYSTEMS.

Reliability indicator	Input variables		Value
Reliability Function $R_{TMR}(t)$	R(t)	R(t) of the original functional unit	$R_{TMR}(t) = 3[R(t)]^2 - 2[R(t)]^3$
Mean Time To Failure $T_{s(TMR)}$	λ	λ of the original functional unit	$T_{s(TMR)} = \frac{5}{6\lambda}$

III. EVALUATION PLATFORM AND EXPERIMENTAL SYSTEM

The development of the evaluation platform for monitoring impact of faults injected into FPGA-based system was the scope, among other, of our previous work [10]. Developed evaluation platform is designed for monitoring impact of faults on electro-mechanical system. The main reason is that lots of digital systems very often control some mechanical part. The use of the electro-mechanical system allows us to monitor not only the impact of faults on the electronic controller but also on the mechanical part. Our evaluation platform uses *Functional Verification* [11] as a tool for checking reactions of experimental system on injected faults. Functional verification is usually used for checking if electronic system corresponds with its specification by monitoring inputs and outputs in design simulation. We propose extended version of the functional verification where verified circuit is running on FPGA as appropriate tool for our purposes. The evaluation platform which is described in our previous work (e.g. [10]) is composed of:

1) software part of verification environment for the electronic controller which checks the reaction of electronic controller and mechanical part on injected faults running on computer,
2) software simulation environment for mechanical part simulation running on computer,
3) electronic controller implemented into FPGA, and
4) external fault injector [12] running on a computer which allows us to simulate real faults in FPGA.

Our experimental electro-mechanical system consists of a robot for searching a path through a maze and its electronic controller implemented in FPGA. Unfortunately, we do not have a real robot device, so we use the simulation tool Player/Stage [13] which allows us to simulate the robot and its environment (in our case the robot in a maze). The robot simulation is executed on a computer which is connected with the FPGA board by the Ethernet interface through which data between the robot and its controller are transmitted. Two versions of the robot controller are used for our experiments and results comparison. The first version is hard-coded robot controller which is composed of various functional units interconnected through the central bus. The second version is processor-based robot controller which consist of soft-core

processor NEO430 [14] and some external components implemented in FPGA. The searching algorithm is implemented in C/C++ and performed on the processor.

IV. Reliability Analysis and Experimental Evaluation

The goal of this work is to evaluate some of the basic *reliability indicators* of the two versions of robot controller which was mention in Section III and their fault-tolerant versions with TMR applied (noted as *noft* and *tmr*).

The fault injection was set up with a constant SEU injection rate. The SEUs were injected to the utilized bits of the bitstream that represent the content of *Look-Up Tables* (LUTs). The important parameter of fault injection is a time delay d_c between two injected faults. The d_c actually does not necessarily have to be constant. We have experimentally chosen the d_c to be described by the *uniform distribution* with a *mean* value of 12 s and a *variance* of 2 s.

The scenario of one verification run was as follows:

1) the robot controller unit was initialized, the maze and starting and target positions were the same during all the verification runs,
2) the *Player/Stage* simulation environment was started with the robot placed on the starting position,
3) after 15 s, for each component c, the SEU injection started with the d_c time period, the bits into which faults were injected were selected *uniformly at random*,
4) the time from the robot start to the first failure was monitored, the ability of the robot to reach the target position was observed as well.

This verification scenario was repeated 3500 times for all versions of the robot controller units. In detail, experimental strategies follow:

- fault injection into unhardened robot controller, component c is whole robot controller (*noft*),
- fault injection into TMR version of robot controller which respect increased area, faults were injected into three component c_1, c_2, c_3 (instances of robot controller) concurrently which led to three times fault intensity for whole robot controller (*tmr*),
- fault injection into TMR version of robot controller, faults were injected into one component c, which represents whole hardened robot controller (fault intensity is the same as in *noft* case) (*tmr1*).

The data acquired included the time of the first failure occurrence and information on whether the robot successfully reached the target position.

The data obtained from the previously described experiments were then processed. The *multi-set* of all the times measured from the start of the operation of the system to the first detection of an error on the system outputs was transformed to a discrete *failure function* $Q(t)$ which was then converted to the *reliability function* $R(t)$. The other *reliability indicators failure density* $f(t)$ and *failure rate* $\lambda(t)$ was computed according to proposed equations. All the data are discretized with the time-step of 15 s in Table II. The final values of the *reliability functions* on the bottom of Table II are notclose to the limit value of zero, as in most cases the faults injected did not appear in the form of an error on the outputs of the robot controller unit. The threshold time length the robot had to find its path within was evaluated to 204 s, that is also the maximum time for which the system has been verified.

TABLE II. A discretization of the measured *failure function* Q(t) of the *noft* and *tmr* version of the *hard-coded* and *processor-based* robot controller with the estimation for the *tmr* robot controller.

		Hard-coded robot cont.			Processor-based robot cont.		
Time t $[s]$		First err. detect. $[-]$	$[\%]$	$Q(t)$ $[-]$	First err. detect. $[-]$	$[\%]$	$Q(t)$ $[-]$
$0 - 14.\bar{9}$	*noft*	0	0.0%	0.00	0	0.0%	0.00
	tmr	0	0.0%	0.00	0	0.0%	0.00
	tmr1	0	0.0%	0.00	0	0.0%	0.00
	est.	–	0.0%	0.00	–	0.0%	0.00
$15 - 29.\bar{9}$	*noft*	6	0.2%	0.00	16	0.5%	0.00
	tmr	1	0.0%	0.00	8	0.2%	0.00
	tmr1	0	0.0%	0.00	0	0.0%	0.00
	est.	–	0.0%	0.00	–	0.0%	0.00
$30 - 44.\bar{9}$	*noft*	9	0.3%	0.00	12	0.3%	0.01
	tmr	0	0.0%	0.00	10	0.3%	0.01
	tmr1	0	0.0%	0.00	2	0.1%	0.00
	est.	–	0.0%	0.00	–	0.0%	0.00
$45 - 59.\bar{9}$	*noft*	35	1.0%	0.01	45	1.3%	0.02
	tmr	8	0.2%	0.00	35	1.0%	0.02
	tmr1	3	0.1%	0.00	5	0.1%	0.00
	est.	–	0.0%	0.00	–	0.1%	0.00
$60 - 74.\bar{9}$	*noft*	28	0.8%	0.02	35	1.0%	0.03
	tmr	25	0.7%	0.01	61	1.7%	0.03
	tmr1	4	0.1%	0.00	3	0.1%	0.00
	est.	–	0.0%	0.00	–	0.2%	0.00
$75 - 89.\bar{9}$	*noft*	8	0.2%	0.02	11	0.3%	0.03
	tmr	11	0.3%	0.01	69	2.0%	0.05
	tmr1	1	0.0%	0.00	9	0.3%	0.01
	est.	–	0.0%	0.00	–	0.1%	0.00
$90 - 104.\bar{9}$	*noft*	47	1.3%	0.04	64	1.8%	0.05
	tmr	53	1.5%	0.03	169	4.8%	0.10
	tmr1	18	0.5%	0.01	26	0.7%	0.01
	est.	–	0.2%	0.00	–	0.5%	0.01
$105 - 119.\bar{9}$	*noft*	42	1.2%	0.05	30	0.9%	0.06
	tmr	36	1.0%	0.04	76	2.2%	0.12
	tmr1	8	0.2%	0.01	18	0.5%	0.02
	est.	–	0.2%	0.00	–	0.3%	0.01
$120 - 134.\bar{9}$	*noft*	52	1.5%	0.06	51	1.5%	0.08
	tmr	34	1.0%	0.05	92	2.6%	0.15
	tmr1	4	0.1%	0.01	21	0.6%	0.02
	est.	–	0.2%	0.01	–	0.6%	0.02
$135 - 149.\bar{9}$	*noft*	36	1.0%	0.08	30	0.9%	0.08
	tmr	47	1.3%	0.06	66	1.9%	0.17
	tmr1	4	0.1%	0.01	19	0.5%	0.03
	est.	–	0.4%	0.01	–	0.4%	0.02
$150 - 164.\bar{9}$	*noft*	42	1.2%	0.09	24	0.7%	0.09
	tmr	33	0.9%	0.07	85	2.4%	0.19
	tmr1	6	0.2%	0.01	16	0.5%	0.03
	est.	–	0.3%	0.01	–	0.3%	0.02
$165 - 179.\bar{9}$	*noft*	50	1.4%	0.10	17	0.5%	0.10
	tmr	48	1.4%	0.08	64	1.8%	0.21
	tmr1	8	0.2%	0.02	26	0.7%	0.04
	est.	–	0.6%	0.02	–	0.2%	0.03
$180 - 194.\bar{9}$	*noft*	45	1.3%	0.11	50	1.4%	0.11
	tmr	46	1.3%	0.10	129	3.7%	0.25
	tmr1	15	0.4%	0.02	58	1.7%	0.06
	est.	–	0.7%	0.03	–	0.8%	0.03
$195 - 209.\bar{9}$	*noft*	856	24.5%	0.36	1520	43.4%	0.54
	tmr	787	22.5%	0.32	1373	39.2%	0.64
	tmr1	129	3.7%	0.06	296	8.5%	0.14
	est.	–	21.8%	0.25	–	53.3%	0.57

Measured data were also transformed to *reliability functions* $R(t)$ for both versions of robot controller. The $R(t)$ functions for hard-coded robot controller are shown in Figure 2. Red lines show, that *tmr* version (injection into 3 component, respect increased area) is better than *noft* version, but significantly worse than estimation. These results were also presented in [7] and additional experiments with version *tmr1* (injection into whole controller, increased area is not taken into

account) were performed. Green line shows, that *tmr1* version is almost the same as estimation.

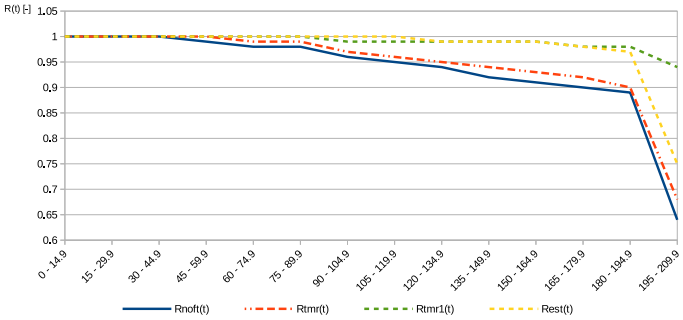

Fig. 2. An experimental evaluation of the measured results of the *reliability function* for the *noft* and the *tmr* versions of the *hard-coded* robot controller.

Additional experiments with processor-based robot controller were performed to confirmation of previous results. Figure 3 show the same chart for processor-based robot controller. The big difference is that *tmr* version (injection into 3 component, respect increased area) is worse than *noft* version. The processor is a complex system and a fault injection with higher intensity led to its worse reliability. On the other hand, *tmr1* version (injection into whole controller, increased area is not taken into account) represented by green line is almost the same as estimated reliability. These experiments confirm, that equation 5 does not take into account increased area of TMR system.

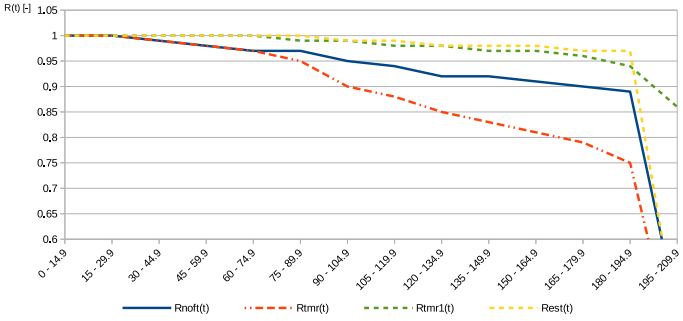

Fig. 3. The measured results of the *reliability function* for the *noft* and the *tmr* versions of the *processor-based* robot controller.

V. Conclusions and Future Research

In this paper we present the combination of experimental and theoretical evaluation of robot controller reliability. We applied a commonly used TMR on the robot controller (there were three instances of the robot controller complemented with the majority voter). The first step was fault injection into unhardened version of robot controller, reliability indicators calculation and then estimated reliability of TMR version were calculated according to commonly used equation 5. Next step was experimental evaluation of estimated reliability indicators. The first experiments were done with fault injection into all robot controller instances concurrently with respect to increased area. The experimentally measured results indicated significantly worse reliability than the estimation predicted. The second experiment was done with fault injection just into the whole robot controller and the obtained results correspond with the estimated reliability. These experiments confirm, that equation 5 does not take into account increased area of TMR system.

Presented results were obtained using TMR without faulty module recovery. The faulty module recovery significantly increases the operation time without failure. The scope of our future research is to apply reconfiguration as a tool for faulty module recovery and perform similar experiments and examine benefits and negatives.

Acknowledgements

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science – LQ1602, the BUT project FIT-S-17-3994 and the JU ECSEL Project SECREDAS (Product Security for Cross Domain Reliable Dependable Automated Systems), Grant agreement No. 783119.

References

[1] I. Koren and C. M. Krishna, *Fault-Tolerant Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[2] F. L. Kastensmidt, G. Neuberger, L. Carro, and R. Reis, "Designing and testing fault-tolerant techniques for sram-based fpgas," in *Proceedings of the 1st conference on Computing frontiers*. ACM, 2004, pp. 419–432.

[3] F. Siegle, T. Vladimirova, J. Ilstad, and O. Emam, "Mitigation of Radiation Effects in SRAM-Based FPGAs for Space Applications," *ACM Comput. Surv.*, vol. 47, no. 2, pp. 37:1–37:34, Jan. 2015.

[4] D. White, "Considerations surrounding single event effects in fpgas, asics, and processors," http://www.xilinx.com/support/documentation/white_papers/wp402_SEE_Considerations.pdf, Mar. 2012, accessed: 2016-09-15.

[5] M. Ceschia, M. Violante, M. Reorda, A. Paccagnella, P. Bernardi, M. Rebaudengo, D. Bortolato, M. Bellato, P. Zambolin, and A. Candelori, "Identification and Classification of Single-event Upsets in the Configuration Memory of SRAM-based FPGAs," vol. 50, no. 6, 2003, pp. 2088–2094.

[6] J. Hlavička, *Číslicové systémy odolné proti poruchám*. ČVUT, 1992.

[7] J. Podivinsky, J. Lojda, O. Cekan, R. Panek, and Z. Kotasek, "Reliability Analysis and Improvement of FPGA-Based Robot Controller," in *Digital System Design (DSD), 2017 Euromicro Conference on*. IEEE, 2017, pp. 337–344.

[8] K. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. Wiley, 2016. [Online]. Available: https://books.google.cz/books?id=_yOXDAAAQBAJ

[9] B. Dhillon, *Applied Reliability and Quality: Fundamentals, Methods and Procedures*, ser. Springer Series in Reliability Engineering. Springer London, 2007. [Online]. Available: https://books.google.cz/books?id=rRp7DkTegMEC

[10] J. Podivinsky, O. Cekan, J. Lojda, M. Zachariasova, M. Krcma, and Z. Kotasek, "Functional Verification based Platform for Evaluating Fault Tolerance Properties," *Microprocessors and Microsystems*, vol. 52, pp. 145 – 159, 2017.

[11] A. Meyer, *Principles of Functional Verification*. Elsevier Science, 2003. [Online]. Available: http://books.google.cz/books?id=qaIiX3hYWL4C

[12] M. Straka, J. Kastil, and Z. Kotasek, "SEU Simulation Framework for Xilinx FPGA: First Step Towards Testing Fault Tolerant Systems," in *14th EUROMICRO Conference on Digital System Design*. IEEE Computer Society, 2011, pp. 223–230.

[13] B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th international conference on advanced robotics*, vol. 1, 2003, pp. 317–323.

[14] S. Nolting, "NEO430 Processor," https://github.com/stnolting/neo430, 2018.