2017 Euromicro Conference on Digital System Design

Reliability Analysis and Improvement of FPGA-based Robot Controller

Jakub Podivinsky, Jakub Lojda, Ondrej Cekan, Richard Panek, 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, 1361, 1362, 1223}
Email: {ipodivinsky, ilojda, icekan, ipanek, kotasek}@fit.vutbr.cz

***Abstract*—Faults occurring in the safety-critical systems can lead to the failure of the whole system and cause high economical losses or endanger human health. As an example, space, aerospace or medical systems which are working in the environment with increased occurrence of faults can serve. Fault avoidance and fault tolerance are the main techniques, the goal of which is to avoid such situations. This paper is the continuation of the previously published work and presents an approach to evaluate fault tolerance techniques by monitoring the impact of faults in the experimental electro-mechanical system which consists of the robot in a maze and its robot controller. The experiments with the robot controller hardened against faults are combined with the reliability analysis on a theoretical level in this paper. The impact of faults artificially injected into the robot controller, in which Triple Modular Redundancy is applied, is monitored and used for statistic reliability analysis.**

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

I. Introduction

Various electronic systems play an important role in our everyday lives. We can meet them in various types of commonly used devices such as cars, intelligent buildings, or some entertainment systems. For example, electronic systems make our lives easier, supervise our health or provide new opportunities. The reliability of these systems is a problem, especially in the case of systems in which failure can result in injury or heavy financial losses or can endanger human health. One of the reasons which leads to a higher susceptibility to faults is an increase of chip-level integration. The current trend is to make electronic systems smaller and integrate more functionality to smaller area on the chip which leads to greater sensitivity to faults. The number of digital systems with high demand on reliability, such as medicine, space, industry, is growing as well.

Two main approaches to increase reliability are currently used. The first one is called *fault avoidance* [1]. As the name indicates, the main goal is to completely avoid failures in the system using the means of more reliable parts, manufacturing processes, etc., which is very challenging and expensive.

The second approach is a technique called *fault tolerance* [2]. Fault tolerance accepts the fact a fault can appear, but the goal of this approach is to keep the system functional, even in the presence of faults. Techniques based on the various types of redundancy are used for this purpose. The most common ones are hardware and time redundancy. Hardware redundancy usually uses n-copies of the same functional unit and comparator to guarantee the proper function. On the other hand, time redundancy is based on computation repeating and the results from the independent runs are then compared. Many fault tolerance methodologies exist, which combine and improve these basic methods, e.g. hardware and time redundancy is combined in the approach presented in [3].

There have been many fault-tolerant methodologies inclined, among others, to *Field Programmable Gate Arrays* (FPGAs) developed and new ones are under investigation [4], because FPGAs are becoming more popular due to their flexibility and re-configurability. The second reason why so many techniques are inclined to FPGAs is their sensitivity to faults and ability to be reconfigured in the case of fault occurrence. FPGAs are composed of configurable logic blocks [5] which are connected by programmable interconnection. The configuration is stored as a *bitstream* in SRAM memory. The problem from the reliability point of view is that FPGAs are quite sensitive to faults caused by charged particles [6]. This particle can induce inversion of a bit in bitstream and this may lead to a change in its behaviour. This event is called *Single Event Upset* (SEU) [7]. The advantage is that faults which occurred in configuration memory can be repaired by *Partial Dynamic Reconfiguration* (PDR) [8].

It is important to test and evaluate these techniques. Various approaches to the evaluation of fault tolerance exist, some of them are performed on a theoretical level, for example, a simulation method for SEU emulation is presented in [9]. Another approach is in the use of fault injection directly into the design implemented in FPGA. Special evaluation boards are developed for these purposes, one of them is presented in [10] or [11]. The evaluation of fault tolerance techniques is one of the goals of our research, in our previously published work where we develop the platform for experimental evaluation of the impact of faults that occurred in an experimental system [12]. Our evaluation platform was tested and demonstrated on the experimental electro-mechanical system (a robot in a maze and its controller) without any hardening against faults. *The next step in our experiments is to apply a fault tolerance technique on our experimental system and evaluate it experimentally which is the main topic of this paper. We feel that only a experimental evaluation is not good enough so the theoretical reliability analysis based on experimental results is also mentioned in this paper.*

This paper is organized as follows. Section II introduces reliability analysis and improvement. An evaluation platform and experimental system on which reliability is experimentally evaluated is presented in Section III. Verification scenarios generation presented in Section IV is the important part of evaluation platform. Section V is dedicated to reliability

978-1-5386-2146-2/17 $31.00 © 2017 IEEE
DOI 10.1109/DSD.2017.15

analysis based on experimental evaluation. The possibility of using a dynamic reconfiguration for the faulty module recovery is shown in Section VI. Section VII concludes the paper and presents future plans of our research.

II. Reliability Analysis and its Improvement

A fault-tolerant system development usually starts with a *nondurable* system that does not tolerate faults [13]. This *nondurable* system is usually designed with minimum redundancy and serves as a starting point for the process of development. An experienced fault-tolerant system designer then suggests the modifications that are to be made to the *nondurable* system in order to to achieve a higher level of fault tolerance. After these changes are incorporated into the design, the result must be evaluated to be sure the applied redundancy has the desired improvement on the reliability of the system. The usual approach is to iterate between the phases of development and the reliability analysis. Multiple designs with various combinations of fault tolerance methods assigned to the partitions of the design are created. The system development ends either with the system complying with the specification or the findings of the specifications not being achievable. In this research we try to accelerate this procedure of the development with an ability to estimate the reliability of the resulting system even *before* the application of the method itself. This allows for a designer to exclude such combinations of reliability methods that do not look perspective from the final specification needs point of view. The final specification usually contains a list of so-called *reliability indicators* and the corresponding ranges of values that must be achieved in order to accept the resulting solution.

A. Reliability Analysis

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. This variable can be considered the so-called *random variable*. The simplified definition of *random variable* according to [14] is shown in Definition 1.

Definition 1: Random variable X on a sample space S is a function $X : S \to \mathbb{R}$ that assigns a real number $X(s)$ to each sample point $s \in S$.

The *Cumulative Distribution Function* (CDF) is an important concept of studying *random variables*. A CDF expresses a probability the *random variable* takes a value lower than a given non-negative real number t which is defined in Equation 1. The CDF is a *nondecreasing function*.

$$F(t) = \mathcal{P}(\tau < t) \tag{1}$$

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 CDF $F(t)$ 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 2.

$$R(t) = 1 - Q(t) \tag{2}$$

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

$$f(t) = \frac{\mathrm{d}Q(t)}{\mathrm{d}t} \tag{3}$$

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 4 gives a relationship between the $\lambda(t)$ and $Q(t)$.

$$\lambda(t) = \frac{f(t)}{R(t)} = \frac{f(t)}{1 - Q(t)} \tag{4}$$

Again, the product $\lambda(t)\mathrm{d}t$ gives a probability of a fault occurrence for a short period of time $\mathrm{d}t$ immediately following after the time t given the examined system was in a *fault-less* state at the time t.

Throughout the whole lifetime of the system, the $\lambda(t)$ usually forms the so-called *bathtub curve*. The *bathtub curve* can be divided into three main time periods with the first one related to early failures, the middle one related to a constant failure period and the last one corresponding with the wear-out phase [15]. The time interval $\langle t_1, t_2 \rangle$ in which the *failure rate* keeps an approximate constant value is the most important part for us, as this is the part where the random failures *rate* is revealed.

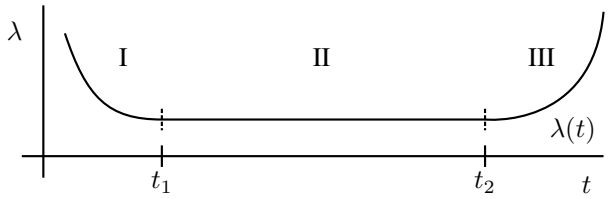

Fig. 1. The usual *failure rate* displayed as a function of a time *t*, the so-called *bathtub curve*.

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 5 can be used.

$$T_s = \int_0^\infty R(t)\mathrm{d}t \tag{5}$$

B. Reliability Improvement

The reliability improvement can be achieved through several means, all of which are based on the concept of *redundancy*. The concept of *time redundancy* is based on increasing the time spent by a particular computation. Another concept of reliability improvement is to utilize redundant information. The *information redundancy* is based on *Error Detection And Correction* (EDAC) codes. Although all of the means of redundancy are closely related, the most fundamental principle is to utilize the *hardware redundancy*.

Probably the most known principle of the *hardware redundancy* is the *Triple Modular Redundancy* (TMR, 3MR) which is based on a *triplication* of the component we intend to improve reliability of [16]. The TMR is based on a *static backup* using three *functionally* and *structurally* equivalent elements. The structural schematic can be seen in Figure 2. The two additional copies of the original functional unit are incorporated into the system. The resulting units are named F_1, F_2 a F_3. The vectors of the input signals x are connected in such a way that each of the functional units F_i works with the same input values. The output signals $f_i(x)$ are connected to the inputs of the so-called *voter* which implements the so-called *majority function*. The *majority function* can be implemented in different ways, the selection of the majority can be performed on the level of bits, whole vector, etc. It is important to note that in the following text, the TMR version using a *voter* that works on the per-bit basis is consulted. The *voter* is built with the use of n instances of the so-called *majority gate* with the n equal to the number of output bits of F_1 (which is the same for all the F_i units).

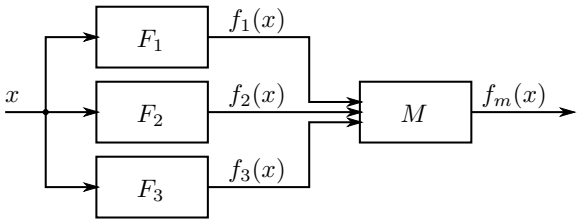

Fig. 2. A system module whose reliability was improved according to the TMR method.

If we omit some edge cases, such as the case when the faults compensate each other, it can be declared that this implementation allows us to mask the failure of one module. If we suppose that each of the F_i modules has an equivalent reliability function $R(t)$, then Equation 6 that can be used to evaluate the resulting reliability function of the whole TMR module exists.

$$R_{TMR}(t) = 3[R(t)]^2 - 2[R(t)]^3 \tag{6}$$

The TMR method is useful for tasks that last for a short period of time where an improvement of the reliability function is desired. For longer lasting tasks an option for faulty modules *recovery* can be added. An overview of the reliability indicators of the TMR systems is shown in Table I [13], [16].

TABLE I. AN OVERVIEW OF THE 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 Robot Controller Case Study

The development of the experimental system and the evaluation platform for monitoring faults injected into FPGA-based system was the scope, among other, of our previous work [12]. A created experimental system can serve as a case study for a discussed techniques demonstration which is also in this paper. Because the digital controlling systems very often control some mechanical part, we decided to create an evaluation platform which can use an electro-mechanical application as an experimental system. It can be stated that such areas exist in which electro-mechanical applications are implemented as fault-tolerant - aerospace and space applications can serve as an example.

A. The Robot Controller - Experimental Electro-mechanical System

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 [17] 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 (see Figure 3) interface through which data between the robot and its controller are transmitted. The robot controller is composed of various functional units which are interconnected through the central bus. There is in total 16 functional units, the main ones are the Position Evaluation Unit (PEU) and the Barrier Detection Unit (BDU) which calculate actual position of the robot in a maze and detect barriers in robot 4-neighborhoods. The Map Unit (MU) stores calculated informations into the Map Memory Unit (MMU) on which path searching realized by Path Finding Unit (PFU) is based. Mechanical parts of the robot are controlled by Engine Control Unit (ECU). Almost all of these functional units are equipped with a control finite stat machine (FSM) and a bus wrapper.

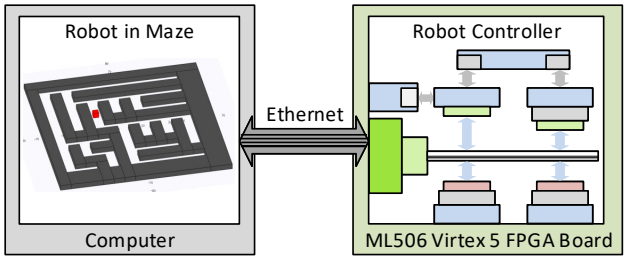

Fig. 3. The robot in maze and its electronic controller.

B. The Evaluation Platform for Monitoring Impact of Faults on Electro-mechanical System

The evaluation platform is based on *Functional Verification* [18]. The main task of the functional verification is to check if

a verified circuit satisfies its specification. It compares outputs of a verified circuit running in a RTL simulator with a reference model implemented in another programming language (e.g. C/C++). In the case of the fault injection, the verified circuit must operate in FPGA, so we do not use classical simulation-based functional verification, but modified FPGA-based functional verification. Our platform uses functional verification as a tool for monitoring impacts of faults injected into electronic controller implemented into the FPGA.

The two main parts of the implemented evaluation platform (see Figure 4) are a computer and an FPGA development board. It allows us to implement a verified electronic controller in FPGA and inject faults directly into FPGA. The fault injector is a component which runs on the computer. Our fault injector [19] is based on the partial reconfiguration. It reads part of the configuration bitstream from the configuration memory, then the required number of specified bits of the bitstream are inverted and a modified bitstream is configured back to the configuration memory through the JTAG interface. The platform is designed to evaluate the impact of faults on the electro-mechanical application, so the simulation of the mechanical part is important and is also runned on the computer. The simulation of the mechanical part is connected with FPGA through the Ethernet interface. The software part of verification environment is also runned on the computer and performs the evaluation of impacts of injected faults on both the electronic and mechanical parts.

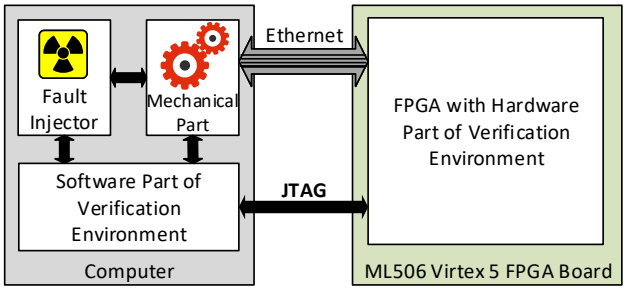

Fig. 4. The architecture of our evaluation platform.

An important metric in functional verification is the *coverage*. It measures how well input stimuli cover the behavior of DUT and provide the feedback that determines when the verification process can be ended. Depending on the required coverage criteria, the *Code coverage* metrics can serve as an example. *Code coverage* measures how well input stimuli cover the source code of DUT. Typical code coverage metrics are toggle, statement, branch, condition, expression or FSM coverage.

IV. Verification Scenarios Generation

Input stimuli are values on the input of the electronic controller on which output values are based on. In the case of the robot controller, input values are changed during the evaluation of one verification scenario (maze, start and goal position). High code coverage was achieved by the set of verification scenarios in our previous work. Actual experiments are simplified because we are using only one verification scenario (one maze) which proposes sufficient code coverage and a suitable number of steps from start to goal position. The principles of finding such a scenario are described in the following text.

In our previous research [12], we used a set of different verification scenarios (mazes) for monitoring the impact of faults on the robot controller. These mazes achieved sufficient code coverage for the verification of the proper function of the robot controller, but the evaluation was very time consuming. The reason was the execution of a large number of experiments which monitors the influence monitoring on the mechanical part of the robot that simulates path finding in a maze. For these experiments, a set of mazes with size 15x15 cells was used. The set reached the maximal code coverage 91.85%. To accelerate the experiments presented in this paper and also future experiments, we build on the premise of finding the one ideal maze (including the start and goal position of the robot) to ensure the correct behavior of the robot controller and to achieve the previously reached maximal coverage. The found mazes should contain a reasonable number of the robot steps from the start to goal position in order to avoid insignificant prolongation of the experiments. Finding the ideal maze and its positions can be divided into three steps - maze generation, maze selection based on the coverage, and maze selection with the optimal number of steps.

A. Maze Generation

We constantly improve and generalize our test stimuli generator which is based on our designed universal architecture. This architecture allows us to describe the desired test stimuli through two specific input structures. The first structure is used to describe the format of the stimuli, while the second structure defines restrictive conditions on how the format has to be constructed into a valid stimulus. In our previous research, we used to define these structures by using our description (language). The language was not general, therefore, a special functionality to provide a valid stimulus had to be implemented with every new type of supported system. We have managed to generalize the test stimuli generator while maintaining the defined universal architecture. Instead of our own language, probabilistic context-free grammar (PCFG) was used. PCFG was extended by restrictive conditions which dynamically adjust the probabilities for rewriting the rules of grammar during the test stimuli generation. In this way, we define a new grammar the expressive power of which is much higher. More information about PCFG extended with constraints may be found in [20].

Using this new grammar, we are also able to describe and generate a maze, where a way between any two points exists. To generate the maze, we use the principles of the binary tree algorithm which can be encoded into our grammar. The basic principle of the binary tree algorithm is shown in Figure 5. It starts from the basic matrix of the maze (a) in which some cells are tightly specified - either a corridor or a wall. The corridors are represented by a white color and walls by a black color. Cells marked with a question mark represent areas that can take the white or black color. In order to maintain the continuity from any corner of the maze to another, it is necessary to perform a modification of the basic matrix of the maze so that each two adjacent sides of the maze must contain the corridor over its entire dimension (b). In our case, we chose this corridor to the northern and the western side of the maze.

The final and most critical task is to determine cells A, B, C, D which allows us to have the maximal continuous maze (c). If cell A, respectively C, was randomly selected for the corridor in Figure 5.b, then cell B, respectively D, will be a wall and vice versa. Such mazes may be generated directly into the binary file (picture) in Bitmap format (BMP), including start and goal positions which can be differentiated by a color.

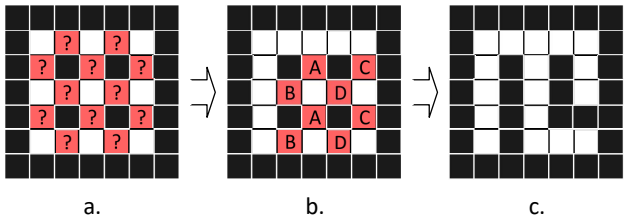

Fig. 5. The demonstration of a conversion of the basic matrix of the maze for needs of the generator.

For the selection of the one maze with maximal coverage, we have generated 300 different mazes with dimensions of 7x7, 15x15, and 31x31 cells by using the described approach. The mazes also have different start and goal positions. The coverage of individual mazes and the differences between different dimensions are shown in Figure 6 with a box plot graph. The upper dash shows the maximal achieved coverage and the lower dash shows the minimal achieved coverage for the set of mazes. The inner line in the chart represents the median value of the coverage. The last shown range defines the first (25% of all values) and third (75% of all values) quartiles. In the figure, it can be seen that with increasing the size of the maze, accomplishment of the maximal coverage occurs more frequently due to the execution of more steps of the robot during the path finding in the maze. It is also evident that some mazes with dimensions 15x15 and 31x31 cells are able to achieve maximal coverage. Therefore, we calculated the average number of steps of the robot for each size of the maze - 13 steps for dimensions of 7x7, 125 steps for 15x15, and 539 steps for 31x31 cells. Based on this information, we have chosen the maze with dimensions of 15x15 cells with maximal coverage 91.85% for further experiments. The inability of achieving an ideal of 100% is caused by the default branches in the source code which are never executed (which is correct), and also by some of the control expressions that are used only when an abnormal situation occurs (e.g. a fault).

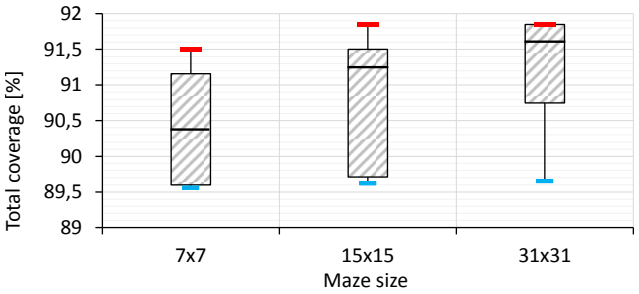

Fig. 6. The Box and whisker chart helps select the right one maze for the robot controller with maximal total coverage in functional verification.

B. Maze Selection With the Optimal Number of Steps

The final step is to select the one maze with dimensions of 15x15 cells which has the optimal number of steps of the robot from the start to the goal position. This condition is important because a maze with the long way has the same coverage as the maze with the short way, but multiple steps do not bring any profit and just prolong the time to perform the experiments. On the other hand, the short mazes can cause a problem with the detection of a fault which may not occur in a short time. Among our test set of generated mazes with dimensions of 15x15 cells, we chose the maze with the maximal coverage of 91.85% and with the number of steps equal to 51 which is an optimal number from our point of view. The selected maze, including start and goal positions, and the way that the robot must follow, is shown in Figure 7.

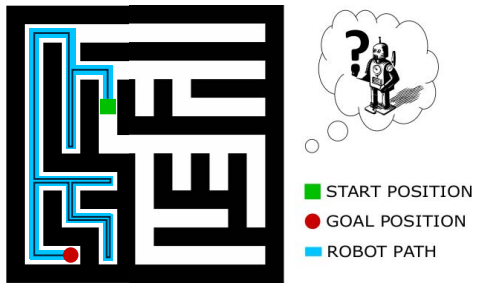

Fig. 7. The selected maze for the robot controller and its path between the start and goal position which the robot found.

V. Reliability Analysis with Experimental Evaluation

The subject of this research is to evaluate some of the basic *reliability indicators* of the robot controller unit which had already been used in our previous research presented in Section III and its fault-tolerant version that is implemented the TMR method. The other subject of this research is to find out if the process of the evaluation technique of the fault-tolerant robot controller could be estimated by an analysis of the *nondurable* robot controller system followed by an application of equations to convert the *reliability indicators* to estimate the indicators of the fault-tolerant controller.

A. Data Acquisition

At first, to be able to practically evaluate the parameters of the robot controller and to verify that the equation mentioned is applicable in our concept, two robot controller versions were created. The first version that is referenced to as to the *noft* version is actually the initial *nondurable* system we started with. The *noft* version includes one component only – the instance of robot controller alone named the *rc*. The second version that is labeled the *tmr* was implemented using three instances of the *rc* robot controller component from the previously mentioned *noft* version and a voter. The components are named *rc1*, *rc2* and *rc3*. It is important to note the voter itself was not subject to evaluation. The resources consumed for both of the versions are shown in Table II.

TABLE II. Resources consumed for both versions of the robot controller units.

Version	Occupied slices [–]	Slice reg. [–]	Slice LUTs [–]	Max freq. [MHz]	LUT bits used [–]
noft	1080	1617	1708	93.76	108480
tmr	2991	4755	5165	93.76	329824

For the practical evaluation of these two robot controller versions, *verification environment* presented in Section III was

used. The fault injection our *verification environment* utilizes was set up with a constant SEU injection rate. The SEUs were injected to the bits of the bitstream that are utilized in the design and represent the content of *Look-Up Tables* (LUTs) at the same time. If we suppose that component c is subject to SEU fault injection, then the important parameter to unambiguously describe this type of fault injection is a time delay d_c between two consecutive SEUs injected to c. The d_c actually does not necessarily have to be constant for the whole time of the SEU simulation, it can be represented by a *random variable* with a particular *probability distribution*. In our experiments, 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 configured into its initial state, the maze map as well as its starting and target positions were the same for all the verification runs,
2) the *Player/Stage* simulation environment was started, the robot was placed on the starting position,
3) after 15 s, for each component c, the SEU injection started with the d_c time period between SEUs based on the *uniform distribution* with the *mean* value of 12 s and *variance* of 2 s, the bits to inject SEU to were selected *uniformly at random*,
4) mainly, the time from the robot start to the first failure observed was monitored, moreover, the ability of the robot to reach the target position was observed as well.

This verification scenario was repeated 3500 times for both of the two versions of the robot controller units. The data acquired included the time of the first failure occurrence and information on whether the robot successfully reached the target position.

B. Method of Reliability Indicators Calculation

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* included are the *failure density* $f(t)$ and *failure rate* $\lambda(t)$.

All the data for the *noft*, *tmr* robot controller unit versions and the estimation of parameters for the *tmr* version are discretized with the time-step of 15 s in Table III. The rows that are marked with *est.* contain the estimations of the *reliability function* for the given time-steps calculated using Equation 6. From there, the other reliability indicators (*failure function*, *failure density* and *failure rate*) were calculated using Equations 2, 3 and 4 from Section II respectively. The final values of the *reliability functions* on the bottom of Table III are not close 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 III. A DISCRETIZATION OF THE MEASURED RELIABILITY PARAMETERS OF THE *noft* AND *tmr* ROBOT CONTROLLER UNITS WITH THE ESTIMATION OF THE PARAMETERS FOR THE *tmr* ROBOT CONTROLLER.

Time t [s]		First err. detect. [−]	[%]	$Q(t)$ [−]	$R(t)$ [−]	$f(t)$ [−]	$\lambda(t)$ [−]
$0-14.\bar{9}$	*noft*	0	0.0%	0.00	1.00	0.0000	0.0000
	tmr	0	0.0%	0.00	1.00	0.0000	0.0000
	est.	–	0.0%	0.00	1.00	0.0000	0.0000
$15-29.\bar{9}$	*noft*	6	0.2%	0.00	1.00	0.0001	0.0001
	tmr	1	0.0%	0.00	1.00	0.0000	0.0000
	est.	–	0.0%	0.00	1.00	0.0000	0.0000
$30-44.\bar{9}$	*noft*	9	0.3%	0.00	1.00	0.0002	0.0002
	tmr	0	0.0%	0.00	1.00	0.0000	0.0000
	est.	–	0.0%	0.00	1.00	0.0000	0.0000
$45-59.\bar{9}$	*noft*	35	1.0%	0.01	0.99	0.0007	0.0007
	tmr	8	0.2%	0.00	1.00	0.0002	0.0002
	est.	–	0.0%	0.00	1.00	0.0000	0.0000
$60-74.\bar{9}$	*noft*	28	0.8%	0.02	0.98	0.0005	0.0005
	tmr	25	0.7%	0.01	0.99	0.0005	0.0005
	est.	–	0.0%	0.00	1.00	0.0000	0.0000
$75-89.\bar{9}$	*noft*	8	0.2%	0.02	0.98	0.0002	0.0002
	tmr	11	0.3%	0.01	0.99	0.0002	0.0002
	est.	–	0.0%	0.00	1.00	0.0000	0.0000
$90-104.\bar{9}$	*noft*	47	1.3%	0.04	0.96	0.0009	0.0009
	tmr	53	1.5%	0.03	0.97	0.0010	0.0010
	est.	–	0.2%	0.00	1.00	0.0001	0.0001
$105-119.\bar{9}$	*noft*	42	1.2%	0.05	0.95	0.0008	0.0008
	tmr	36	1.0%	0.04	0.96	0.0007	0.0007
	est.	–	0.2%	0.00	1.00	0.0001	0.0001
$120-134.\bar{9}$	*noft*	52	1.5%	0.06	0.94	0.0010	0.0011
	tmr	34	1.0%	0.05	0.95	0.0006	0.0007
	est.	–	0.2%	0.01	0.99	0.0002	0.0002
$135-149.\bar{9}$	*noft*	36	1.0%	0.08	0.92	0.0007	0.0007
	tmr	47	1.3%	0.06	0.94	0.0009	0.0010
	est.	–	0.4%	0.01	0.99	0.0003	0.0003
$150-164.\bar{9}$	*noft*	42	1.2%	0.09	0.91	0.0008	0.0009
	tmr	33	0.9%	0.07	0.93	0.0006	0.0007
	est.	–	0.3%	0.01	0.99	0.0002	0.0002
$165-179.\bar{9}$	*noft*	50	1.4%	0.10	0.90	0.0010	0.0011
	tmr	48	1.4%	0.08	0.92	0.0009	0.0010
	est.	–	0.6%	0.02	0.98	0.0004	0.0004
$180-194.\bar{9}$	*noft*	45	1.3%	0.11	0.89	0.0009	0.0010
	tmr	46	1.3%	0.10	0.90	0.0009	0.0010
	est.	–	0.7%	0.03	0.97	0.0004	0.0004
$195-209.\bar{9}$	*noft*	856	24.5%	0.36	0.64	0.0163	0.0254
	tmr	787	22.5%	0.32	0.68	0.0150	0.0221
	est.	–	21.8%	0.25	0.75	0.0146	0.0193

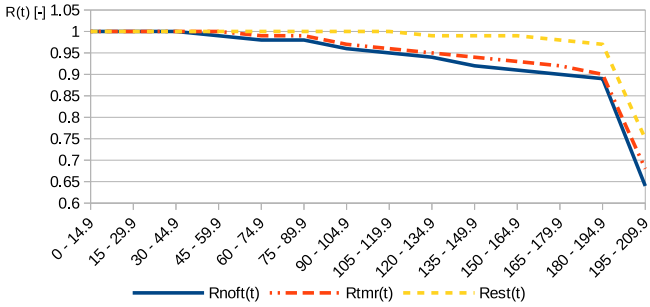

Fig. 8. An experimental evaluation of the measured results of the *reliability function* for the *noft* and the *tmr* versions.

The results of our research can be seen from two different points of view. The first point of view is from the improvement of the reliability by the TMR application. As can be seen in the chart on Figure 8, it is evident that the implementation of the TMR method improved the reliability of the robot controller unit in each of the time-steps we used to discretize the calculation. As a known fact, the chart shows the TMR without faulty modules *recovery* is useful mainly for shorter mission times. From the second point of view, there is a considerable difference between the estimated and measured parameters for the *tmr* version of the robot controller unit. One reason we believe could possibly cause this phenomenon

is the fact the evaluation was done with the fault injections to the LUTs only. Another consideration includes a possible problem within the method of the calculation. In order to find out the true reason for this, disproportion is seen by us as a good direction for future research.

VI. Reliability Improvement by Partial Dynamic Reconfiguration

The results of our research demonstrated in the previous parts of this paper is the increase of system reliability by using the TMR approach. TMR is naturally only a passive approach [21], which is capable of delivering correct output when a fault occurs. However, just fault masking means that FPGA is not fully utilized – namely the ability of changing its configuration. Therefore, the extension of the passive by active approach is very effective. The active approach of avoiding the impact of fault occurrence is based on a reconfiguration. The combination of passive and active approaches is described in [22]. The utilization of a reconfiguration controller to eliminate faults in FPGA based applications is a key technique for such approaches. This controller is responsible for scrubbing and other activities combined with bitstream relocation. A Partial Dynamic Reconfiguration (PDR) is very advantageous when used because it does not interrupt other functions implemented in FPGA. Therefore, PDR is a very important approach for critical applications and Generic Partial Dynamic Reconfiguration Controller (GPDRC) [23] is an example of such a controller used for this purpose. An extension of the TMR majority voter function is demanded to acquire information on which module a fault has occurred. This extension resides in adding an output from the voter, which identifies the malfunction module whose output value is different from the other two. Mitigation of faults that occur in individual TMR modules is possible due to the GPDRC existence in the design. Such an approach increases the reliability of a particular system.

A. PDR Controller Requirement

The requirement of including more components to FPGA necessarily represents sufficient FPGA size. However, larger and more expensive FPGA will be a requisite for achieving fault tolerance improvement. Possibly, a reconfiguration controller can be configured in FPGA with the application, or as another component which has to be included into the system (i.e. an extra FPGA). However, the increase in occupied FPGA area is manifested in both cases. The probability of hitting the utilized part of FPGA by some fault, which can cause a circuit malfunction, increases concurrently with its reliability. Faults, which hit the TMR module, are not important because the reconfiguration controller will fix them. Nevertheless, some faults can hit the reconfiguration controller that will behave in an incorrect way. It can cause ultimate system breakdown even if this system behavior is presumed. The malfunctioning controller might occasionally destroy a correct circuit configuration due to a reconfiguration which is performed unexpectedly. If the controller area in FPGA is considerably smaller than the application in TMR, the controller should provide longer failure-free time compared with triplication.

The reconfiguration controller is utilized to deliver bitstreams into the FPGA configuration memory during the PDR procedure. These bitstreams are called *golden bitstreams*. Special memory can be used to save them. When a fault is identified, the controller then loads and applies corresponding bitstreams from this memory which should be protected against faults as well. The records can be equipped with a correction code. Another component which can possibly be included in the FPGA produces higher FPGA area requirements. Another possibility is the utilization of module triplication (TMR). When there is a fault in one module, the remaining two modules are still faultless. Therefore, three bitstreams of three modules are downloaded from the FPGA configuration memory. Then, a new bitstream is prepared as a majority value of each bit of these bistreams. This new bitstream is used for fixing faults instead of using golden bitstream. This approach is called *Lazy Scrubbing* [24] by M. Garvie.

B. Fault Tolerant PDR Controller

Fault tolerance of a reconfiguration controller might be also required in some applications. The same technique as for protected circuit can possibly be used (i.e. TMR). The reconfiguration controller will then be implemented three times in FPGA and each of its instances will operate in parallel with the remaining two. The outputs from each instance will be compared and when a controller malfunction is detected, the other two correctly functioning controllers will be capable of reconfiguring the incorrectly working one. Even in this case, when two controllers reconfigure the third one simultaneously, it is possible to check fault occurrence. Because outputs of the two correctly working controllers are compared and if they are equal, no fault exists there. This approach corresponds to the Duplication with Comparison (DwC) method. Nevertheless, in the case of fault occurrence in DwC architecture it is impossible to determine which controller works incorrectly. It is important to note that the fault tolerance for reconfiguration controller has a negative impact on FPGA area and circuit delay – because of a majority voter addition to the system.

C. Research in the Area of GPDRC

Two versions of the PDR controller for FPGA, which we designate as GPDRC, were designed and implemented within our research group. The first version of the controller [23] is capable to mitigate transient faults i.e. SEUs. The second expanded version of the controller [25] is focused on the mitigation of permanent faults too. The permanent fault is a fault when some bits of FPGA configuration memory are in an incorrect state without a possibility to change it. Some spare modules for TMR are available in the FPGA to be used in these situations. These modules are able to be used instead of another TMR module with a permanent fault. When all spare modules are exhausted and one more permanent fault occurs, fault protection degrades to only DwC. The next permanent fault causes circuit function termination.

Previous research aims at ensuring fault tolerance for the circuit function itself. However, fault tolerance for the reconfiguration controller itself is not provided. The controller has to be in a radiation protection component outside FPGA, which performs its function. It is certainly not the only alternative of protecting the reconfiguration controller against faults.

Future research in the area of GPDRC will be oriented to bringing fault tolerance into the GPDRC design. The utilization of the TMR approach for the design of a fault tolerant GPDRC is an idea for future research. This approach is based on placing three equivalent controllers into the FPGA and connecting their outputs to a special majority voter. The experiments

with this approach will be executed and compared with the version which contains just one reconfiguration controller. In both cases, the reconfiguration controller will guarantee the proper function of a robot controller, which is described herein. Certainly, this is only a passive approach for fault tolerance, which is not sufficient for proper attenuation of the SEU impact on the FPGA. Therefore, we will solve how to perform reconfiguration of the malfunction controller by another two controllers which operate correctly. Obviously, the malfunction reconfiguration controller must not interfere in its reconfiguration. However, that problem will be solved by a majority voter, which masks one incorrect output due to another two correct outputs. We will ensure fault tolerance for majority voters after thorough testing and comparison with previous versions. The majority voters remain as the last unprotected component of the system.

VII. Conclusions and Future Research

In this paper we introduced the combination of fault injection-based experimental evaluation and theoretical reliability analysis of our previously developed robot controller. Our previous work was targeted mainly towards the experimental evaluation, but we feel that the theoretical reliability analysis has also important place in our evaluation process. We applied a commonly used TMR on the top level of the robot controller (there were three instances of the robot controller complemented with the majority voter). The first step was the fault injection-based experimental evaluation of the robot controller without TMR applied and its reliability indicators calculation. The calculation of reliability indicators of the TMR version was done in the second step, then we gained estimated reliability indicators. Reliability indicators of the TMR version were also evaluated by fault injection. These experiments show us that TMR has a significant impact on the reliability indicator and improves reliability of the hardened robot controller. When we compare a measured and estimated reliability indicator we found that the measured values are not so good as the estimated ones. We outlined possible causes which are in the scope of our future research.

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, ARTEMIS JU under grant agreement no 621439 (ALMARVI) and BUT project FIT-S-14-2297.

References

[1] J.-C. Geffroy and G. Motet, *Design of Dependable Computing Systems*. Kluwer Academic Publishers, 2002.

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

[3] 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.

[4] 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.

[5] XILINX. (2014, Nov.) FPGA. [Online]. Available: http://www.xilinx.com/fpga/index.htm

[6] 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.

[7] 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.

[8] XILINX, "Partial Reconfiguration User Guide," http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ug702.pdf, Apr. 2012, accessed: 2016-09-15.

[9] C. Bernardeschi, L. Cassano, A. Domenici, and L. Sterpone, "Accurate Simulation of SEUs in the Configuration Memory of SRAM-based FPGAs," in *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 115–120.

[10] M. Alderighi, S. D'Angelo, M. Mancini, and G. R. Sechi, "A Fault Injection Tool for SRAM-based FPGAs," in *On-Line Testing Symposium, 2003. IOLTS 2003. 9th IEEE*. IEEE, 2003, pp. 129–133.

[11] M. Alderighi, F. Casini, S. d'Angelo, M. Mancini, S. Pastore, and G. R. Sechi, "Evaluation of Single Event Upset Mitigation Schemes for SRAM-based FPGAs Using the FLIPPER Fault Injection Platform," in *Defect and Fault-Tolerance in VLSI Systems, 2007. DFT'07. 22nd IEEE International Symposium on*. IEEE, 2007, pp. 105–113.

[12] J. Podivinsky, O. Cekan, J. Lojda, and Z. Kotasek, "Verification of Robot Controller for Evaluating Impacts of Faults in Electro-mechanical Systems," in *Digital System Design (DSD), 2016 19th Euromicro Conference on*. IEEE, 2016, pp. 487–494.

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

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

[15] S. Sangwine, *Electronic Components and Technology, Third Edition*, ser. Tutorial guides in electronic engineering. CRC Press, 2007. [Online]. Available: https://books.google.cz/books?id=vVHvBQAAQBAJ

[16] 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

[17] 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.

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

[19] 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.

[20] O. Cekan, J. Podivinsky, and Z. Kotasek, "Random stimuli generation based on a stochastic context-free grammar," in *Proceedings of the 2016 International Conference on Field Programmable Technology*. IEEE Computer Society, 2016, pp. 291–292.

[21] J. Jiang and X. Yu, "Fault-tolerant control systems: A comparative study between active and passive approaches," *Annual Reviews in Control*, vol. 36, no. 1, pp. 60–72, 2012.

[22] X. Yu and J. Jiang, "Hybrid fault-tolerant flight control system design against partial actuator failures," *IEEE Transactions on Control Systems Technology*, vol. 20, no. 4, pp. 871–886, July 2012.

[23] M. Straka, J. Kastil, and Z. Kotasek, "Generic partial dynamic reconfiguration controller for fault tolerant designs based on fpga," in *NORCHIP 2010*, Nov 2010, pp. 1–4.

[24] M. Garvie, "Reliable electronics through artificial evolution," Ph.D. dissertation, University of Sussex, January 2005.

[25] L. Miculka and Z. Kotasek, "Generic partial dynamic reconfiguration controller for transient and permanent fault mitigation in fault tolerant systems implemented into fpga," in *17th International Symposium on Design and Diagnostics of Electronic Circuits Systems*, April 2014, pp. 171–174.