2018 21st Euromicro Conference on Digital System Design

Evaluation Platform for Testing Fault Tolerance Properties: Soft-core Processor-based Experimental Robot Controller

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

***Abstract*—Various electronic systems play an important role in our everyday lives. Some of them serve for fun or to make our lives easier. These systems are useful but not necessary; when they malfunction, the consequences are not critical. On the other hand, there are systems which are more or less critical, and their failure can cause undesirable consequences. For example, a failure in medicine, aviation, the army or automotive systems can cause high economic losses and/or endanger human health. These systems must be protected against the impact of faults, and flawless operation must be ensured. Fault tolerance is one of the techniques that will ensure this. There are many fault-tolerance methodologies targeted towards various systems and technologies, and new methodologies are being investigated. It is also important to verify these techniques; this is the main topic of this paper. An evaluation platform for testing fault-tolerance methodologies targeted towards SRAM-based FPGAs (Field Programmable Gate Arrays) is presented and demonstrated. A robot for seeking a path through a maze and the processor-based robot controller serve as an experimental system case study. Experimental results with the unhardened and hardened versions of the processor-based robot controller are presented and discussed.**

***Keywords*—Soft-core Processor, NEO430, TMR, FPGA, Fault Tolerance, Robot Controller, Reconfiguration.**

I. Introduction

We meet with various electronic systems playing important roles in our everyday lives. These systems are integrated in various commonly used devices such as cars, intelligent buildings, and some entertainment systems. Electronic systems make our lives easier, monitor our health, and provide new opportunities. It is very important to ensure the reliability of systems, the failure of which can cause high economic losses and/or can endanger human health. The current trend is to increase chip-level integration, which allows us to make electronic systems smaller and integrate more functionality into a smaller area on the chip. The problem is that this trend also leads to greater sensitivity to faults. The number of digital systems with a high demand on reliability, such as medicine, space, and industry, is growing as well. It is important to protect these systems against the consequences of faults.

Two main approaches to increase reliability are currently used. The first is called *fault avoidance* [1]. It is a very challenging and expensive approach; the primary goal is to completely prevent failures in the system using more reliable parts, manufacturing processes, etc. The second approach is called *fault tolerance* [2]. Fault tolerance accepts the fact that 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 redundancies are used for this purpose. The most common types are spatial and time redundancy. Time redundancy is based on computation repeating and the results from the independent runs are then compared. On the other hand, spatial redundancy usually uses n-copies of the same functional unit and comparator to guarantee the proper function. Many fault tolerance methodologies exist, which combine and improve these basic methods, e.g. hardware and time redundancy are combined in the approach presented in [3].

Many fault-tolerant methodologies have been developed, among others, to *Field Programmable Gate Arrays* (FPGAs) and new types are under investigation [2], [4], because FPGAs are becoming more popular due to their flexibility and reconfigurability. FPGAs are an alternative solution to *Application Specific Integrated Circuits* (ASICs), which are beneficial in systems that are produced in small series. Fault-tolerance methodologies targeted towards FPGAs are often based on spatial redundancy, specifically on *Triple Modular Redundancy* (TMR), which uses three copies of the same functional unit. The disadvantage is the high consumption of resources, which is leading scientists to develop some improvements. A new technique based on the identification of critical bits of the bitstream and their hardening with TMR is presented in [5]. The practical aspects of TMR implementation on the FPGA and the proper location of triplicated units is discussed in [6]. It is advantageous to place individual copies in the disjoint areas. The unconventional use of TMR combined with High Level Synthesis is presented in [7].

The second reason why so many techniques are inclined towards FPGAs is their sensitivity to faults and their ability to be reconfigured if a fault occurs in the configuration memory. FPGAs are composed of configurable logic blocks [8], which are connected by programmable interconnection. The configuration is stored as a *bitstream* in the SRAM memory. The problem, from the point of view of reliability, is that FPGAs are quite sensitive to faults caused by charged particles [9]. This particles can induce the inversion of a bit in the bitstream, and this may lead to a change in its behavior. This event is called *Single Event Upset* (SEU) [2]. The advantage is that faults which occurred in the configuration memory can be repaired by *Partial Dynamic Reconfiguration* (PDR) [10].

978-1-5386-7377-5/18/$31.00 ©2018 IEEE
DOI 10.1109/DSD.2018.00051

It is important to test and evaluate fault-tolerance 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 [11]. Another approach is the use of artificial fault injection directly into the design implemented in the FPGA. Special evaluation boards are developed for these purposes; one of them is proposed in [12] and [13]. The combination of simulation method and hardware evaluation is discussed in [14].

The *goals of our research* are to develop an evaluation platform for testing fault-tolerance techniques based on functional verification. Our evaluation platform was presented in [15]. The proposed platform is able to monitor the impact of faults on an electro-mechanical system; this means monitoring the impact of faults both on the electronic controller and the mechanical part, because electronic controllers usually control some kind of mechanical part in real applications. Our evaluation platform was tested and demonstrated with the use of an experimental electro-mechanical system (a robot in a maze and its electronic controller) with TMR applied. The next step is to use another experimental system, apply some kind of fault tolerance technique, and demonstrate the use of an evaluation platform, which is the main topic of this paper.

This paper is organized as follows. Section II introduces a previously developed evaluation platform for monitoring the impact of faults on electro-mechanical applications. The experimental electro-mechanical system composed of a robot in a maze and its processor-based robot controller are proposed in Section III. Experiments with proposed experimental systems are presented in Section IV together with their comparison with previously obtained results. Section V concludes the paper and presents the plans for our future research.

II. The Evaluation Platform and The Evaluation Process

An evaluation platform for monitoring the impact of faults on electro-mechanical systems was presented in our previous work [15]. The evaluation process based on the evaluation platform was also presented previously. In this paper, the description of the evaluation platform and the evaluation process are brought to mind, and a case study with a new, experimental electronic controller

A. The Evaluation Platform for Monitoring the Impact of Faults on an Electro-mechanical System

Our evaluation platform is based on *Functional Verification* [16]. The main task of functional verification is to check whether a verified circuit meets its specifications. It compares the outputs of a verified circuit running in an RTL simulator with those of a reference model. In the case of the fault injection, the verified circuit must be implemented into the 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 the impacts of faults injected into an electronic controller implemented into the FPGA.

The two main components of the proposed evaluation platform shown in Figure 1 are a computer and an FPGA development board. The platform is designed to monitor the impact of faults on the electro-mechanical application, so the mechanical part (or its simulation) is an important unit running on the computer. The mechanical part is connected with the FPGA through an Ethernet interface. The software part of the verification environment is also running on the computer and performs the evaluation of the impacts of injected faults on both the electronic and the mechanical parts.

The use of an FPGA development board where an electronic controller is implemented allows us to inject faults directly into the FPGA. The fault injector is one of the components which runs on the computer. Our fault injector [17] is based on the partial reconfiguration. It reads part of the configuration bitstream from the configuration memory, then the specified bits of the bitstream are inverted and a modified part of the bitstream is configured back to the configuration memory. A JTAG interface is used for reading bitstreams from the FPGA and writing modified bitstreams back to the FPGA.

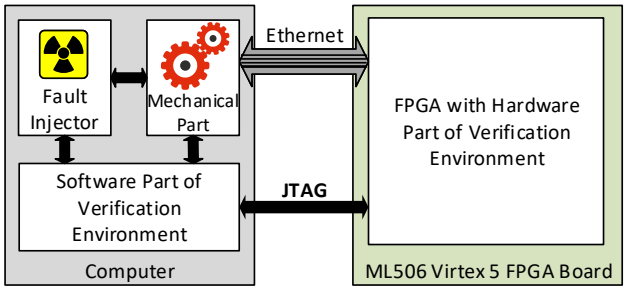

Fig. 1. The architecture of the proposed evaluation platform.

An important metric in functional verification is *coverage*. It measures how well the verification scenarios cover the behavior of the DUT and provide 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 the verification scenarios cover the source code of the DUT. Typical code coverage metrics are toggle, statement, branch, condition, expression, and FSM coverage.

B. The Three Phases of the Evaluation Process

The evaluation process of fault impact monitoring is shown in Figure 2. The proposed process is divided into three main phases. Simulation-based functional verification is performed in the first phase. The VHDL description is used as the DUT and the C/C++ implementation of the electronic controller is used as a reference model. The simulation-based verification environment, which is used in this phase, is usually developed during the development cycle of the whole system. In this phase, the correctness of the electronic controller design is evaluated. The main output of the first phase is a test as to whether the electronic controller works correctly, according to the specification. This is important, because we have to ensure that the electronic controller does not contain any functional errors in its implementation. The generated set of verification scenarios must lead to maximum code coverage, which ensures that much of the code is verified. It is also important to point out that the set of verification scenarios acquired in this phase can be used in the subsequent phase.

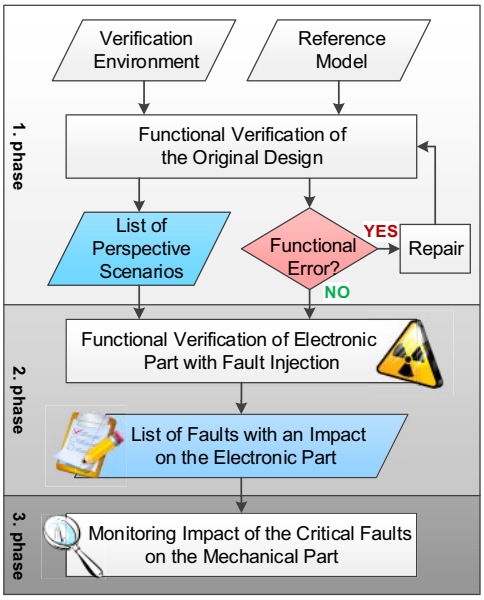

Fig. 2. The flow of phases in the FT evaluation system verification.

The evaluation of the impact of faults on the electronic controller is a task for the second phase, which consists of verification of the electronic controller implemented into the FPGA with the verification scenarios obtained during the previous phase. Modifications of the verification environment used in the previous phase are needed for this phase because functional verification serves merely as a communication observer. It monitors and checks communication between the mechanical part and the electronic controller, and errors in communication are reported and analyzed. In this phase, artificial faults are injected into the FPGAs using an implemented fault injector. The output of this phase is a list of verification runs with information about the injected faults and the results of the verification run (success, failure). The injected faults are divided into two categories: Faults with no impact on the electronic part, and faults which cause mismatches on the output of the electronic part. Various strategies of fault injection may be used in this phase (e.g. one fault per verification run, multiple faults in the same functional unit, or multiple faults in different functional units).

The analysis of the faults that corrupted the mechanical part is the goal of the third phase. The information from the sensors on the mechanical part are used for monitoring its behavior. These sensors usually provide sufficient information about the behavior. Some additional modifications of the verification environment are needed for this phase. It is necessary to implement evaluation of the behavior from the sensor information. The outputs of the third phase also form a list of verification scenarios with the injected faults and their impact on the mechanical part. The faults can cause failure of the mechanical part, with collisions or inaccuracies in the behavior of the mechanical part.

C. Verification Scenario Generation

An important tool in functional verification is verification scenario generation. We need to generate a set of verification scenarios which ensure sufficient code coverage and which also would be suitable for our robot controller. The universal Stimuli Generator was designed for these purposes. It performs pseudo-random generation, which is appropriate to capture the usual and unusual verification scenarios through the whole state space for various systems.

The versatility of the generator is ensured by the probabilistic grammar with constraints which was presented in [18]. Probabilistic grammar is the common context-free grammar which has defined probabilities for its rewriting rules. The constraints are our extension of this grammar, which modifies the probabilities of rewriting rules during the generation. Thanks to this, we are able to control the generation process and get the valid verification scenario for various systems. In probabilistic grammar, the desired verification scenarios are encoded using finite language. In the constraints, there are conditions in which a specific rewriting rule of the grammar gets a new probability value. For this reason, it is ensured that a particular rewriting rule is applied in certain situations, but in other situations, it is not applied. Therefore, we are able to get valid scenarios for a system (a subset of all possible scenarios).

As can be seen from the previous text, in the previous period we dealt with various activities in the area of evaluating the design of fault-tolerant systems. Our new activity, namely the use of a soft-core processor as the robot controller and its use in fault-tolerant system design, certainly belongs to this area.

III. Case Study: Soft-core Processor-based Robot Controller

A robot in a maze, and its electronic controller implemented in the FPGA, were used as an experimental electromechanical system (Figure 3) in our previous work [15]. Unfortunately, we have no real robot device, so we use a Player/Stage [19] tool for the robot and its environment simulation. The task for our robot and its controller is to seek a path through a maze. The electronic robot controller was a ”hard coded” implementation configured into the FPGA. There are various possibilities to implement an electronic controller, one of them is to use soft-core processor implemented on FPGA together with some additional components and create a System on Chip (SoC). The robot controller implemented as an SoC with a processor is used for experiments in this paper.

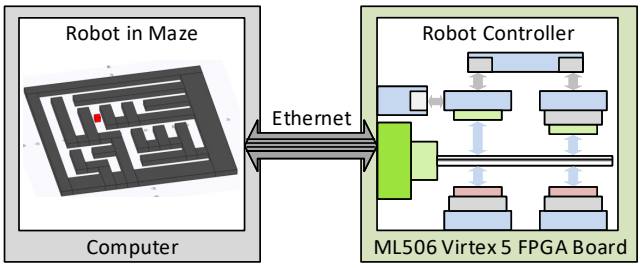

Fig. 3. The robot in the maze and its electronic controller.

As an experimental processor we chose the NEO430 Processor [20], which is a customizable and microcontroller-like processor for FPGA designs. This processor is based on Texas Instruments MSP430 [21] instruction set architecture

and provides compatibility with the original instruction set. The architecture of the processor is shown in Figure 4. The processor already implements standard features like a timer, a watchdog, UART and SPI serial interfaces (implemented together as a USART unit), general purpose IO ports, an internal bootloader, and internal memory for program code and data. All of the peripheral modules are optional; it is possible to exclude them from implementation to reduce the size of the system. Any additional modules can be connected via a Wishbone bus.

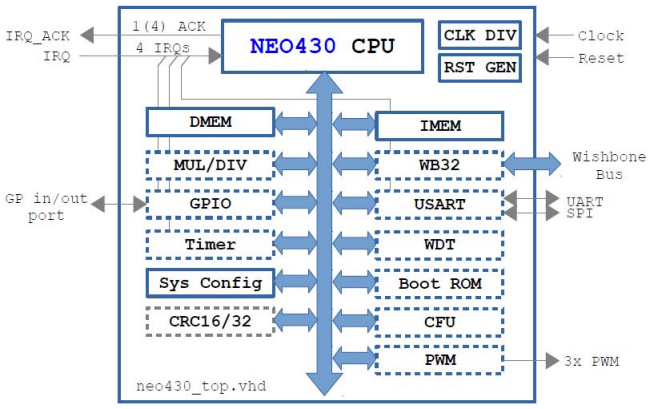

Fig. 4. The architecture of the NEO430 Processor [20].

The use of the NEO430 Processor as the main part of our robot controller is shown in Figure 5. Optional peripheral modules which are used in our design are shown. We use a Custom Functional Unit (CFU) as an input interface for data with information about the robot's position in the maze (DIST_A, DIST_B, DIST_C – simplified GPS) and the distances from the barriers in the robot four-neighborhood (S_0, S_1, S_2, S_3). The CFU is connected to the processor system bus and allows writing data to the registers. Information about the robot's position and barriers is written in registers, which makes it available from the CPU.

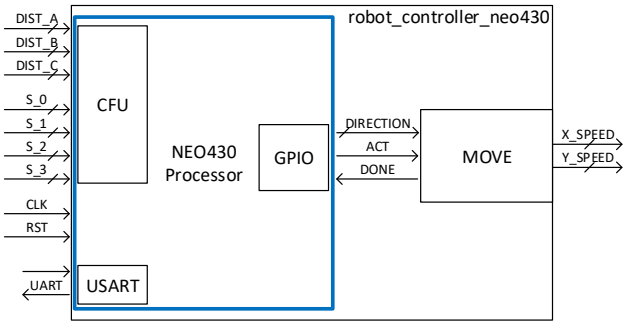

Fig. 5. The architecture of the robot controller composed of the NEO430 Processor.

The input data are processed by the processor and the General-purposes inputs and outputs (GPIO) are used for communication with the MOVE unit. The MOVE unit controls the mechanical robot by setting the speed in the X and Y axis for a specified time, according to the input values. The input is just the direction of the movement and activation signal (ACT). The movement is confirmed by a DONE signal produced by the MOVE unit. The processor must wait for the DONE signal before the next input data are processed and the next movement is activated.

The boot-loader was used only for program debugging. In the experimental version of the processor-based robot controller, the program is stored in the instruction read-only memory (ROM). The UART is connected to the output interface of the whole FPGA and can then be connected to the computer. This allows us to monitor additional information about the program behavior. A simple "left hand on the wall" method is used as a searching algorithm. This means that at each crossroad, the robot turns left. The program is composed of several steps, which are performed until the robot reaches the goal position:

1) Read the information about the robot's position and barriers in the robot 4-neighborhood; the DIST_A, DIST_B and DIST_C values represent represent the distances from the fixed points A, B, and C in a map. From these values, the position coordinates are calculated. The S_x values represent distances from barriers in the 4-neighborhood.
2) Evaluate the position and the barriers and calculate the next position. A simple "left hand on the wall" algorithm is implemented.
3) The command to execute the robot's movement is sent to the MOVE unit, which sets the speed in the X and Y directions for a specified time and the robot moves to the next position.

Our evaluation platform is designed mainly for testing fault-tolerance methodologies, so the current experiments correspond with this. For the experiments discussed in the next section, a hardened version of our experimental system was developed, which allows us to compare the impact of faults on on the electronic and mechanical parts, on both the hardened and unhardened versions of the experimental system. The commonly used *Triple Modular Redundancy* (TMR) was chosen for our experiments because it is a basic method, which is used in many practical applications and forms the basis of more advanced techniques. Of course, it is possible to use other fault-tolerance techniques and the main steps of our evaluation process will be the same.

The use of TMR architecture as an FT technique for our processor-based robot controller is shown in Figure 6. There are three instances of the processor with a majority voter for correct output determination. We use a majority voter that works "per bits". The connection of the UART interface with the outer world is done only for a single instance of the processor.

IV. Case Study: Experimental Results

The main part of this paper deals with experiments and experimental results. The complete evaluation process is performed and reported in the following section. The verification environments needed for the evaluation in each phases of the evaluation process are shown and described together with their practical use during evaluation. A great deal of attention is paid to results analysis and the achieved results are compared with the results obtained during experiments with the original robot controller.

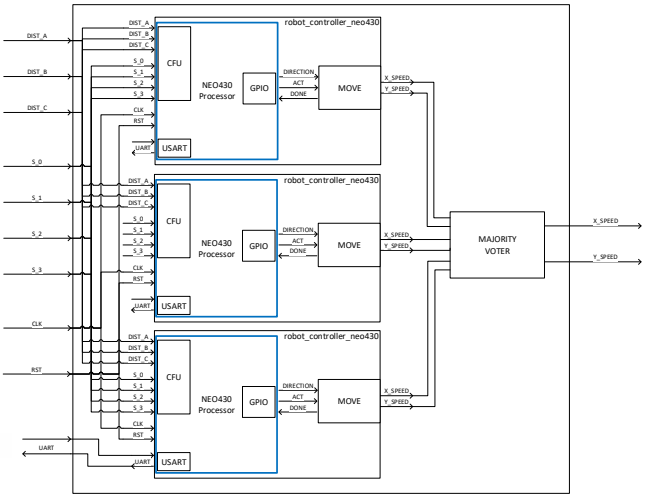

Fig. 6. The architecture of the TMR version of the robot controller composed of NEO430 Processor.

A. *The First Phase - Simulation-based Functional Verification*

The first phase focuses on simulation-based functional verification of the evaluated electronic controller. The output of this phase is a robot controller without implementation faults, which ensures that errors detected in the following phases are caused by injected faults. The verification scenarios (images of mazes) are generated with the use of our universal stimuli generator in order to achieve maximum code coverage. Set of verification scenarios with high code coverage is also one of the outputs of this phase.

The verification environment used in the first phase is implemented according to Universal Verification Methodology (UVM) and is shown in Figure 7. The robot controller as the DUV (Device Under Verification) is equipped with verification components. The inputs for the robot controller (DUV) are the outputs of the simulation of the robot in the maze, which is driven by the outputs of the robot controller. An important component is the Golden Model, which generates the reference outputs for comparison with the outputs of the DUT.

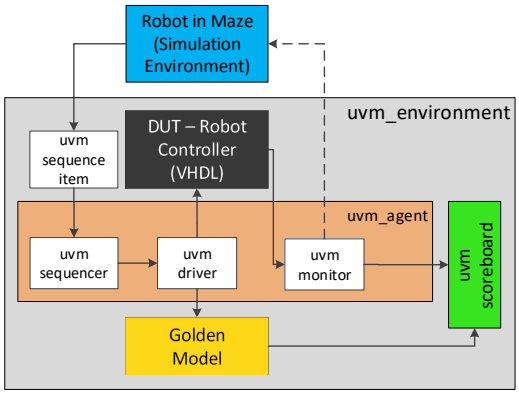

Fig. 7. The architecture of the verification environment for the robot controller.

We evaluated three types of mazes with various dimensions, and our goal is to find which size is good enough for the subsequent phases. Three sizes of maze (shown in Figure 8) were evaluated: 7x7, 15x15, and 31x31 cells. The average number of steps that must be done by the robot on the way to the finish position is shown in Table I. It can be seen that, with the increasing dimensions of a maze, the number of steps the robot has to go through also increases.

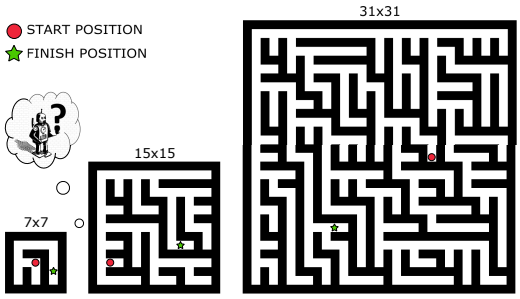

Fig. 8. Three types of mazes - the example of one scenario for each type.

TABLE I. AVERAGE NUMBER OF ROBOT STEPS

Maze size	7x7	15x15	31x31
Average number of steps	15	99	342

For each maze dimension, we generated 1,000 mazes through our generator, which differ in corner composition and also in the start and finish position for the robot in the maze. We have verified the obtained mazes in the process of functional verification for a correct output (the robot reached the finish position after the prescribed number of steps) and obtained the value of the code coverage for these mazes. The number of verification scenarios for the evaluation was chosen as 1, 10, 100 and 1,000. In total, we performed an evaluation of 3,000 verification scenarios with different mazes. The main objective of these experiments is to find the dimension and number of mazes that will provide the highest code coverage.

The result of the experiments with measured code coverage is presented in Table II. The achieved maximum total code coverage is 75.80% for almost all test scenarios. The difference is in the dimension of a maze 7x7 cells with 1 verification scenario, where the total code coverage was 75.49%. The inability to reach 100% total code coverage is due to the complex implementation of the processor used. In the processor, there are many functional units, signals, buses, etc. which are not fully utilized, because the robot controller is, in principle, a simple automaton compared to the processor's possibilities. The table also shows that with the increasing dimension of the maze the achieved coverage does not increase. This is due to the fact that one verification scenario carries several input transactions. It is also clear from the table that just one maze is sufficient to reach maximum coverage.

During experiments, the robot always arrived at the finish position. The robot did not freeze on a place, crash, or behave unusually, so we can say that the robot controller is properly verified on 3,000 input stimuli and, therefore, it does not contain any implementation errors.

Based on the previous paragraphs, we will select one suitable maze for the subsequent phases. The selection of an appropriate maze is done based on the total code coverage for the individual mazes. The coverage is shown in Figure 9, with a box plot graph which shows the range of coverage achieved. The lower dash indicates the minimum achieved coverage,

TABLE II. THE RESULT OF EXPERIMENTS WITH MEASURED CODE COVERAGE.

# of verification scenarios	1			10			100			1000		
Size of mazes	7x7	15x15	31x31	7x7	15x15	31x31	7x7	15x15	31x31	7x7	15x15	31x31
Statement coverage	72.01 %	72.24 %	72.24 %	72.24 %	72.24 %	72.24 %	72.24 %	72.24 %	72.24 %	72.24 %	72.24 %	72.24 %
Branch coverage	69.46 %	69.66 %	69.66 %	69.66 %	69.66 %	69.66 %	69.66 %	69.66 %	69.66 %	69.66 %	69.66 %	69.66 %
Expression coverage	59.09 %	59.09 %	59.09 %	59.09 %	59.09 %	59.09 %	59.09 %	59.09 %	59.09 %	59.09 %	59.09 %	59.09 %
Condition coverage	76.92 %	78.02 %	78.02 %	78.02 %	78.02 %	78.02 %	78.02 %	78.02 %	78.02 %	78.02 %	78.02 %	78.02 %
Total coverage	75.49 %	75.80 %	75.80 %	75.80 %	75.80 %	75.80 %	75.80 %	75.80 %	75.80 %	75.80 %	75.80 %	75.80 %

while the upper dash shows the maximum achieved coverage. The middle square defines the first and third quartiles (range of achieved coverages between 25% and 75%). The line between the quartiles represents the median value of the coverage. The figure shows that when increasing the dimensions of the maze, the maximum coverage is hit more frequently, because more steps of the robot are performed. However, this does not change the fact that from each dimension, a certain number of mazes can be selected, because they reached the maximum possible coverage. Based on Table I, which contains information about the average number of steps of the robot (15 steps for 7x7, 99 steps for 15x15, and 342 for 31x31 cells), we chose the maze with dimensions of 15x15 cells. For our experiments, we need a sufficient number of steps to detect a mismatch after the fault injection. For this reason, the 15x15 and 31x31 dimensions are suitable, but the 31x31 maze already contains too many steps that do not bring any benefits and just prolong the time to perform the experiments.

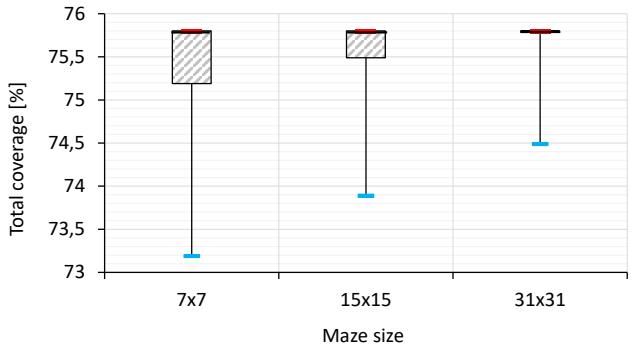

Fig. 9. The box and whisker chart for the selection of the right maze for the robot controller.

The final step is to select the maze with dimensions of 15x15 cells, which has the optimal number of steps of the robot from the start to the finish position. We chose the maze with the maximum code coverage of 75.80% and with 85 steps, which is an optimal number from our point of view. The selected maze, including the start and finish positions, and the path that the robot must follow, are shown in Figure 10.

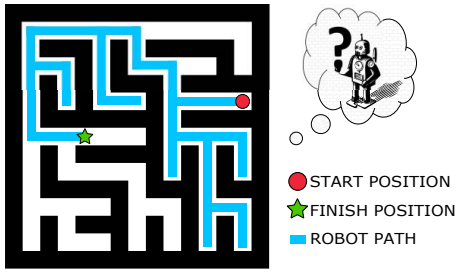

Fig. 10. The selected maze for the robot controller and its path between the start and finish position, which the robot found.

B. The Second and the Third Phases

The second phase focuses on the evaluation of the impact of faults on the output of the electronic controller. We use a processor-based robot controller whose outputs are commands for robot movement. We must monitor whether the commands for the robot in the maze are being generated properly. The main task of the third phase is to monitor the impact of faults on the mechanical part. The mechanical part is the robot in the maze which is equipped with sensors measuring the distances from the walls and the current position. The outputs of these sensors can be used for monitoring the behavior of the robot in the maze. We can detect collision of the robot with the wall, stopping on place and, other behavior of the robot.

The UVM-based verification environment for both the second and the third phases is shown in Figure 11. The proposed verification environment covers the tasks for both the second and the third phases. For the second phase of the evaluation process, this verification environment monitors communication before the simulated robot in the maze and its robot controller running on FPGA. It operates as an observer without any direct intervention in the monitored communication. The correctness of the communication is evaluated by comparison with the outputs of the reference model. The third phase of the evaluation process is done by monitoring the outputs from the sensors and evaluating these outputs. We can detect a small or zero distance from a wall, which is a critical situation. In addition, we can detect when the robot stops in any given place, or when the robot chooses to go in a direction that does not match the implemented searching algorithm.

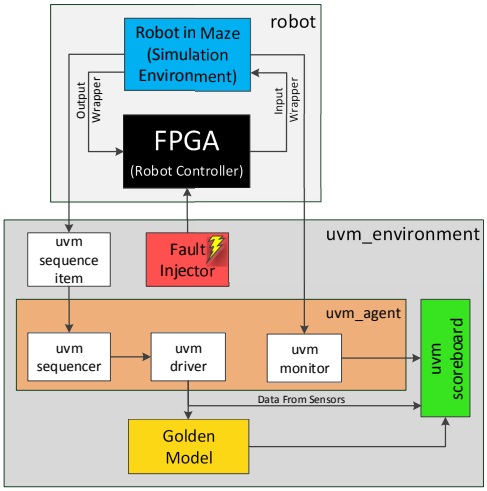

Fig. 11. The architecture of the FPGA-based verification environment for the robot controller.

Figure 11 shows that the fault injector is an important component. Thanks to the use of the FPGA board we can inject faults directly into the configuration bitstream. We use our

previously implemented fault injector [17], which allows us to invert the specified bit of the bitstream. The fault injector uses reconfiguration of the FPGA configuration bitstream. At first, part of the configuration bitstream is read, then the specified bit is inverted, and the modified bitstream is configured back to the configuration memory. The fault injector is able to find the relation between the bit of the bitstream representing the Look-up Tables (LUTs) and the specified area on the FPGA. This means that we are able to inject artificial faults into the LUTs corresponding with the specified functional units.

Two different strategies of fault injection are used in these experiments: *Multiple faults* and *single faults*. Experiments are done for the mentioned unhardened version and the TMR version of the processor-based robot controller. The number of verification runs that were performed for each version of the robot controller and each fault injection strategy is 5000 verification runs. Experimental results are compared with the same experiments with the original hard-coded robot controller.

1) Multiple fault injection: Permanent bit-flips were injected into utilized LUT contents with a constant period of $15s$. This period was chosen experimentally, based on the system failure manifestation time. This means that in each $15s$ only one SEU was injected into the whole robot controller unit LUT contents (only utilized LUT bits are considered) until the robot failed or reached the finish position.

The experimental results for multiple fault injection strategy are summarized in Table III. It shows the results of both the unhardened and the TMR versions of the processor-based robot controller and it contains a comparison with the original hard-coded robot controller. One can see, see that the unhardened electronic version failed in 44.02% and the TMR version failed in 8.14% of the cases. This confirms that TMR is a beneficial approach, even though the increase in resource consumption is high. The table also shows the impact of faults on the mechanical robot; a large number of electronic failures leads to the robot stopping in a place which is less critical than a collision with a wall. The reliability improvement was calculated according to Equation 1. the In comparison with the original hard-coded robot controller, the processor-based robot controller is more susceptible to faults. This fact is evident both for the unhardened and the TMR version. This phenomenon was expected, because the processor represents a more complex design with lots of partial components. These experiments confirmed our expectations.

TABLE III. A COMPARISON OF THE IMPACT OF *multiple* FAULTS INJECTED INTO THE UNHARDENED AND HARDENED VERSIONS OF THE PROCESSOR-BASED ROBOT CONTROLLER AND THE ORIGINAL HARD CODED ROBOT CONTROLLER.

Monitored impact	*Processor-based RC*		*Original hard-coded RC*	
	noft	*tmr*	*noft*	*tmr*
Electronic OK [−]	2751	4593	3544	4839
Electronic failed [−]	2201	407	1456	161
Electronic failed [%]	44.02%	8.14%	29.12%	3.22%
Finish not reached [−]	2179	403	1429	161
Collision with wall [−]	55	7	11	0
Robot stop on place [−]	2124	396	1418	161
Reliability improvement [%]	81.5%		88.9%	

$$reliab\_improv = \frac{failures_{noft} - failures_{tmr}}{failures_{noft}} * 100 \quad (1)$$

The experimental results for the multiple fault injection strategy are also presented in Figure 12, where number of faults which led to electronic failure is shown. This chart shows that the number of faults which led to an electronic failure of hardened processor-based robot controller is higher than in the case of an unhardened robot controller. The same situation is true in the case of the original robot controller, but there are some differences between the hard-coded original robot controller and the processor-based robot controller. The chart shows that the original unhardened robot controller needs a few more injected faults in order to fail. The situation is different in the case of the TMR versions of the robot controller. In this case, the number of faults which led to a failure is almost the same.

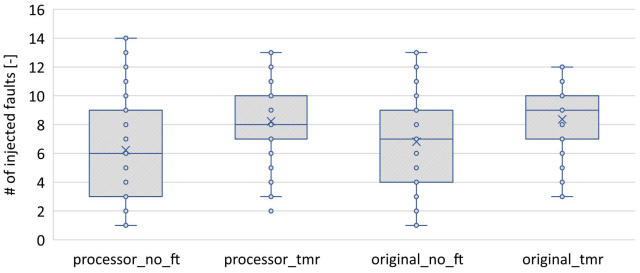

Fig. 12. The box plot shows a statistical comparison of the number of injected faults which led to electronic failure for both the processor-based and the original robot controllers.

2) Single fault injection: Exactly one bit-flip of the utilized LUT contents of a particular component was injected per verification run, and its impact on the behavior of the whole controller unit was observed.

The experimental results for single fault injection into the unhardened and TMR versions of the processor-based robot controller are presented in Table IV. It is obvious that the number of failures is lower than in the case of multiple fault injection. As can be seen, almost all faults are tolerated in the TMR version. Even in the case of single fault injection the number of electronic failures which leads to a collision with the wall is low. The comparison with the original hard-coded robot controller is also shown in Table IV. This table confirms our findings realized during the experiments with multiple injections. In the case of single fault injection, the difference is not so significant, but Table IV shows that processor based robot controller is more sensitive to injected single faults than original hard coded robot controller. It is also interesting that both single and multiple faults injected into processor based robot controller without fault tolerance mechanism led to more collisions of robot with wall. This also confirms our assumption that the processor is a more complex system with multiple vulnerable components.

TABLE IV. A COMPARISON OF THE IMPACT OF *single* FAULTS INJECTED INTO THE UNHARDENED AND HARDENED VERSIONS OF THE PROCESSOR-BASED ROBOT CONTROLLER AND THE ORIGINAL HARD CODED ROBOT CONTROLLER.

Monitored impact	*Processor-based RC*		*Original hard-coded RC*	
	noft	*tmr*	*noft*	*tmr*
Electronic OK [−]	4729	4997	4802	4998
Electronic failed [−]	271	3	198	2
Electronic failed [%]	5.42%	0.06%	3.96%	0.04%
Finish not reached [−]	271	3	195	2
Collision with wall [−]	16	0	1	0
Robot stop on place [−]	255	3	194	2
Reliability improvement [%]	98.8%		98.9%	

V. Conclusions and Future Research

The evaluation platform for monitoring the impact of faults on the electro-mechanical system was presented in this paper. The presented evaluation platform is based on functional verification, and the evaluation process is divided into three phases. The first phase uses classical simulation-based functional verification and verifies the correctness of the experimental system. The second phase focuses on fault injection directly into an electronic controller running on an FPGA. In this phase, modified FPGA-based verification environment is necessary. Monitoring the impact of injected faults on the mechanical part is a task for the third phase. The third phase also uses an FPGA-based verification environment modified for monitoring the behavior of the mechanical part.

The whole evaluation process was experimentally evaluated in our research and demonstrated in this paper. A robot for seeking a path through a maze with a new processor-based robot controller serves as an experimental electro-mechanical application. The new robot controller is designed as a system on a chip composed of an NEO430 soft-core processor, equipped with supporting peripheral components. Experiments corresponding with the first phase were performed, and one maze with high code coverage was selected for the subsequent phases. The second and third phases were performed with the use of one combined FPGA-based verification environment. Experiments with fault injection were done for both the unhardened and the TMR versions of a processor-based robot controller. Experimental results show that TMR is beneficial both for multiple and single fault injection strategy. The comparison of the results gained from the processor-based robot controller with the previously evaluated hard-coded robot controller was also mentioned. Our experiments show that the processor-based robot controller is a more susceptible to faults than the original hard-coded robot controller. The experiments confirmed our assumption that a processor is more complex system with a number of critical components.

As a future work, we plan to apply some sophisticated fault tolerance techniques on the presented experimental electro-mechanical system and repeat the complete evaluation process. One of the possible improvements is the use of reconfiguration for faulty module recovery and synchronization of the recovered module (processor in our case study) with failure-free modules.

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 and the BUT project FIT-S-17-3994.

References

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

[2] M. Yang, G. Hua, Y. Feng, and J. Gong, *Fault-Tolerance Techniques for Spacecraft Control Computers*. John Wiley & Sons, 2017.

[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] X. She and N. Li, "Reducing Critical Configuration Bits via Partial TMR for SEU Mitigation in FPGAs," *IEEE Transactions on Nuclear Science*, vol. 64, no. 10, pp. 2626–2632, 2017.

[6] L. Sterpone and L. Boragno, "Analysis of Radiation-induced Cross Domain Errors in TMR Architectures on SRAM-based FPGAs," in *On-Line Testing and Robust System Design (IOLTS), 2017 IEEE 23rd International Symposium on*. IEEE, 2017, pp. 174–179.

[7] A. F. dos Santos, L. A. Tambara, and F. L. Kastensmidt, "Evaluating the Efficiency of Using TMR in the High-Level Synthesis Design Flow of SRAM-based FPGA," in *Circuits & Systems (LASCAS), 2017 IEEE 8th Latin American Symposium on*. IEEE, 2017, pp. 1–4.

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

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] T. Nidhin, A. Bhattacharyya, R. Behera, T. Jayanthi, and K. Velusamy, "Dependable System Design with Soft Error Mitigation Techniques in sram based fpgas," in *Power and Advanced Computing Technologies (i-PACT), 2017 Innovations in*. IEEE, 2017, pp. 1–6.

[15] 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.

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

[17] 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.

[18] O. Cekan and Z. Kotasek, "A Probabilistic Context-Free Grammar Based Random Test Program Generation," in *2017 Euromicro Conference on Digital System Design (DSD)*, Aug 2017, pp. 356–359.

[19] 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.

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

[21] Texas Instruments. (2018, Feb.) Msp430 ultra-low-power sensing & measurement mcus. [Online]. Available: http://www.ti.com/microcontrollers/msp430-ultra-low-power-mcus/overview.html