2014 17th Euromicro Conference on Digital System Design

The Evaluation Platform for Testing Fault-Tolerance Methodologies in Electro-mechanical Applications

Jakub Podivinsky, Ondrej Cekan, Marcela Simkova, Zdenek Kotasek
Faculty of Information Technology, Brno University of Technology
Bozetechnova 2, 612 66 Brno, Czech Republic
Tel.: +420 54114-{1361, 1361, 1362, 1223}
Email: {ipodivinsky, icekan, isimkova, kotasek}@fit.vutbr.cz

***Abstract*—The aim of this paper is to present a new platform for estimating the fault-tolerance quality of electro-mechanical applications based on FPGAs. We demonstrate one working example of such EM application that was evaluated using our platform: the mechanical robot and its electronic controller in an FPGA. Different building blocks of the electronic robot controller allow to model different effects of faults on the whole mission of the robot (searching a path in a maze). In the experiments, the mechanical robot is simulated in the simulation environment, where the effects of faults injected into its controller can be seen. In this way, it is possible to differentiate between the fault that causes the failure of the system and the fault that only decreases the performance. Further extensions of the platform focus on the interconnection of the platform with the functional verification environment working directly in FPGA that allows automation and speed-up of checking the correctness of the system after the injection of faults.**

***Keywords*—Fault Tolerance, Electro-mechanical Systems, Fault Injection, Single Event Upset.**

I. Introduction

In several areas, such as aerospace and space applications or automotive safety-critical applications, fault tolerant electro-mechanical (EM) systems are highly desirable. In these systems, the mechanical part is controlled by its electronic controller. Currently, a trend is to add even more electronics into EM systems. For example, in aerospace, extending of the electronic part results in a lower weight that helps reduce the operating cost [1] [2]. The situation is similar in other sectors, such as automotive [3].

It is obvious that the fault-tolerance methodologies are targeted mainly to the electronic components because they perform the actual computation. However, as the electronics can be realized on different hardware platforms (processors, ASICs, FPGAs, etc.), specific fault-tolerance techniques dedicated for these platforms must be developed.

Our research is targeted to *Field Programmable Gate Array*s (FPGAs) as they present many advantages from the industrial point of view. They can compute many problems hundreds times faster than modern processors. Moreover, their reconfigurability allows almost the same flexibility as processors. FPGAs are composed of *Configurable Logic Blocks* (CLBs) that are interconnected by a programmable interconnection net. Every CLB consists of LUTs *Look-Up Table* that realizes the logic function, a multiplexer and a flip-flop. Structure of CLB is shown in Figure 1. The configuration of CLBs and of the interconnection net is stored in the SRAM memory.

The problem from the reliability point of view is that FPGAs are quite sensitive to faults caused by charged particles [4]. These particles can induce an inversion of a bit in the configuration SRAM memory of an FPGA (or directly to its internal flip-flops) and this may lead to a change in its behaviour. Affecting SRAM or directly the flip-flops can be seen as equivalent in possible consequences. This event is called the *Single Event Upset* (SEU).

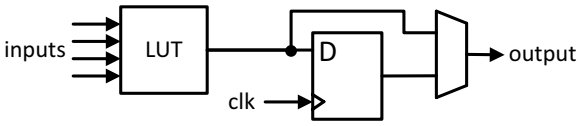

Fig. 1. Structure of *Configurable Logic Blocks*.

The paper is organized as follows. The related work connected to the FPGA reliability is summarized in Section II. The goals of our research and the interconnection scheme of the platform for estimating the quality of EM applications can be found in Section III. The architecture of our experimental design, the robot controller, is provided in Section IV. A detailed description of the fault injection process that is used for artificial injection of faults into the robot controller can be found in Section V. Results of the experiments with the robot controller are available in Section VI. The future work that includes using *functional verification* for automated evaluation of impacts of faults and the test generation process is presented in Section VII and VIII. Finally, Section IX concludes the paper.

II. Related Work

An important feature of FPGAs, which can be utilized for reliability purposes after a fault (we consider SEUs) is detected, is called *Partial Dynamic Reconfiguration* (PDR). PDR can reconfigure the affected part of the FPGA (a faulty module) and restore the electronic system into the correct operation without interrupting other parts of the system. This type of fault repair during the system runtime can be supported by hardware redundancy architectures, such as *Triple Modular Redundancy* (TMR) [5] or duplex system with *Concurrent Error Detection* (CED) [6]. Sensitivity to faults (SEUs) and the possibility of reconfiguration are the main reasons why so

978-1-4799-5793-4/14 $31.00 © 2014 IEEE
DOI 10.1109/DSD.2014.57

many fault-tolerance methodologies inclined to FPGAs have been developed and new ones are under investigation [7],[8].

From the above facts, we have identified two areas that we would like to focus on in our research of fault-tolerant FPGA-based systems:

The first one is that methodologies are validated and demonstrated only on simple electronic circuits implemented in FPGAs. For instance, methodologies focused on the memory in [9] are validated on simple memories without the additional logic around. In [10], the fault-tolerance technique is presented only on a two-input multiplexer, one simple adder and one counter. Other methodology dedicated to harden finite state machines [11] is applied only on a simple finite state machine. Of course, for the demonstration purposes such circuits are satisfactory. However, in real systems different types of blocks must be protected against faults at the same time and must communicate with each other. Therefore, a general evaluation platform for testing, analysis and comparison of alone-working or cooperating fault-tolerance methodologies is needed.

As for the second area of the research and the main contribution of our work, we feel that it must be possible to check the reactions of the mechanical part of the system if the functionality of its electronic controller is corrupted by faults. It is either done in simulation or in a physical realization.

III. The Goals of the Research

According to the identified problems we have formulated our goals in the following way:

1) *To develop an evaluation platform based on the FPGA technology for checking the resilience of EM applications against faults.*

2) *To develop and verify a new methodology for increasing fault-tolerance qualities of EM applications using the proposed platform.*

Under the term EM application we understand a mechanical device and its electronic controller implemented in an FPGA. In our experiments, these components are represented by a robot device and its controller, which drives the movement of a robot in a maze.

At this point, we wanted to target also the issue of complexity. The electronic part, the robot controller, is designed as a complex system with specific components that will allow testing and validating individual or cooperating fault-tolerance methodologies based on the FPGA.

As for the first goal of our research, we have already implemented the evaluation platform that consists of three basic parts:

- the Virtex5 FPGA board, where the robot controller is situated after the synthesis and the place and route process,
- the simulation environment Player/Stage [12] for checking responses of the mechanical device to instructions from the robot controller (see Figure 2),
- the external fault injector (PC) which inserts faults into the robot controller [13].

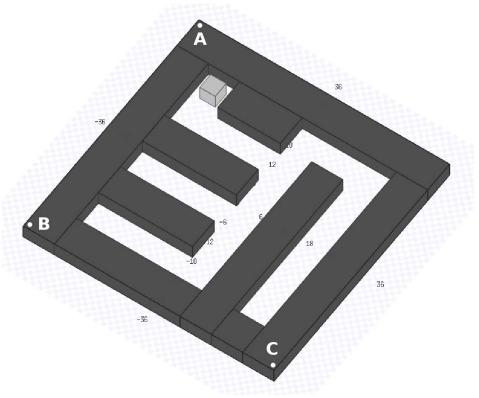

Fig. 2. The robot in a maze in Player/Stage simulation environment.

The second goal of our research is covered by the development of a methodology how to incrementally harden EM systems against faults. We expect to identify clearly the situations when the reconfigurable hardware covers correctly its functions (and the robot works properly) but also the situations when the mechanical functions are corrupted and the robot collapses.

Figure 3 shows the overall interconnection of the PC and the FPGA board in our platform. Note that there are two devices called FITkit [14] in both directions, from the PC to the FPGA and vice versa. FITkit is a hardware platform that was developed for student projects at the Faculty of Information Technology, Brno University of Technology. In our platform, FITkits represent a communication layer and serve as a debugging point for communication between the PC and the FPGA board. The SEU injector runs on the PC and is connected through the JTAG interface directly to the main FPGA board where the robot controller is situated. Via the connection between the SEU injector and the simulation environment (as shown in Figure 3), we are able to control the SEU injection process into the robot controller for every mission and to see effects of faults directly in simulation.

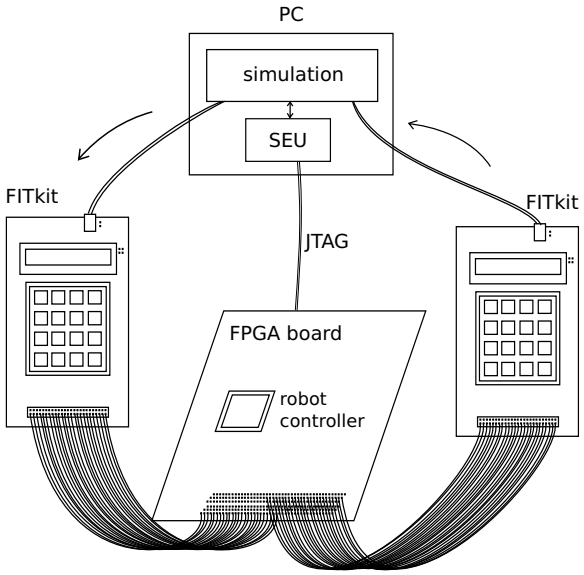

Fig. 3. The platform for testing fault-tolerance methodologies.

In our opinion, it is important to find a relation between the level of functional corruption of the electronic controller and the corruption of the mechanical functionality in the EM applications (i.e. between the robot controller and the simulated mechanical robot). Therefore, it must be possible to introduce various levels of external faults into the controller and check whether the mechanical function: a) was not corrupted, b) was corrupted partially, c) was corrupted completely.

IV. The Robot Controller - Structure and Principles

In Figure 4, the block diagram of the implemented robot controller is available. The control unit is connected to the PC (where the simulation environment is located) via the Interface Block. Through this block, data from the simulation are received (information about barriers, distances from control points, target positions) and in the opposite direction, instructions about the movement of the robot are sent (direction and speed).

The robot controller is composed of various blocks, their function is described in [15]. Here, we only summarize main characteristics of every component. The central block of the robot controller is a bus through which the communication between each block is accomplished. Each of components, without the Engine Control Module, is connected to the bus. The Position Evaluation Unit acquires the distance from the control points, which are located in the fixed positions in the maze. From these, the position of the robot in the maze is calculated and provided to other units as coordinates x and y. The Barrier Detection Unit (BDU) uses four sensors; each located on one side of the robot (cubical robot) and provides information about the distance to the surrounding barriers. The output is a four-bit vector that represents the four-neighbourhood of the robot and informs about barriers in this area. Map updating is provided by the Map Unit (MU) and is based on the information about the position of the robot obtained from the Position Evaluation Unit and the information about the occurrence of barriers in a four-neighbourhood provided by the Barrier Detection Unit. The Map Memory Unit (MMU) stores information about the up-to-date map. The memory is realized by the block memory (BRAM) available in the FPGA. The most important block that manages the activity of other blocks in the robot controller is the Path Finding Unit (PFU). It implements the simple iteration algorithm for finding a path through the maze according to the information about the current and the desired target position. The mechanical parts of the robot are driven by the setting of the speed in the required direction of the movement by the Engine Control Module (ECM).

The robot controller is designed as a complex system with specific components that will allow testing and validating various types of fault-tolerant methodologies focused on FPGAs:

- *Combinational circuits*
  Combinational circuits are the basic types of digital circuits, their output is dependent just on the current input. In the robot controller, the Barrier Detection Unit represents a pure combinational circuit.
- *Sequential circuits*
  The output of the sequential circuit, unlike combinational circuit, is not dependent only on the current input but also on the actual state. These circuits also contain a memory for storing a state. Sequential circuits can be explicitly controlled by the finite state machine. Sequential circuits without an explicit control are represented by the Map Unit and the Position Evaluation Unit in the robot controller.
- *Finite state machines*
  Finite state machines also represent sequential circuits, their computational process is modeled by states and transitions between them. In the robot controller, the Path Finding Unit and the Engine Control Module, together with units that provide the bus communication, are implemented as finite state machines.
- *Buses*
  The bus is a central element of our controller. We decided to use freely available *Wishbone bus* [16] that is configured as a shared bus. It means that the communication on the bus can be driven only by one master device and the other units must wait for releasing the bus. All function blocks are connected to the bus via their wrapper.
- *Memories*
  In the robot controller, we can find two occurrences of different types of memory. The first, the Map Memory Unit, is realized as the Block Memory (BRAM) which is available on the FPGA. The second memory is a queue in the Engine Control Module that stores continuously calculated path to the destination.

V. Evaluation of Reliability by Fault Injection

The weak point of FPGAs from the reliability point of view is their configuration memory. The functionality of an FPGA chip is defined by the sequence of configuration bits (called *bitstream*) which is loaded into the configuration memory. In our case, a specific part of bitstream determines the functionality of the robot controller.

However, even the smallest change in the configuration memory can lead to different functionality. When a charged particle strikes a memory cell, the resulting effect is the inversion of the stored value (known as the *Single Event Upset*, SEU) [17].

During testing the resilience of systems against faults, waiting for their natural appearance is not feasible. A typical reason is the *Mean Time Between Failures* (MTBF) parameter that can be in the order of years. Therefore, some special techniques are used in order to artificially accelerate the fault occurrence. The most popular one is called *fault injection*.

Therefore, to simulate effects of faults in the FPGA, it could be done by a direct change of the configuration bitstream which is loaded into the configuration memory. For this purpose we implemented a fault injector [13] which allows to prepare bitstream for our FPGA and also to modify single or multiple bits of the bitstream in order to simulate single and multiple faults. As a consequence, the design placed in the FPGA (determined by the configuration data) is influenced similarly as by a real fault which strikes the hardware architecture of the FPGA in a real environment.

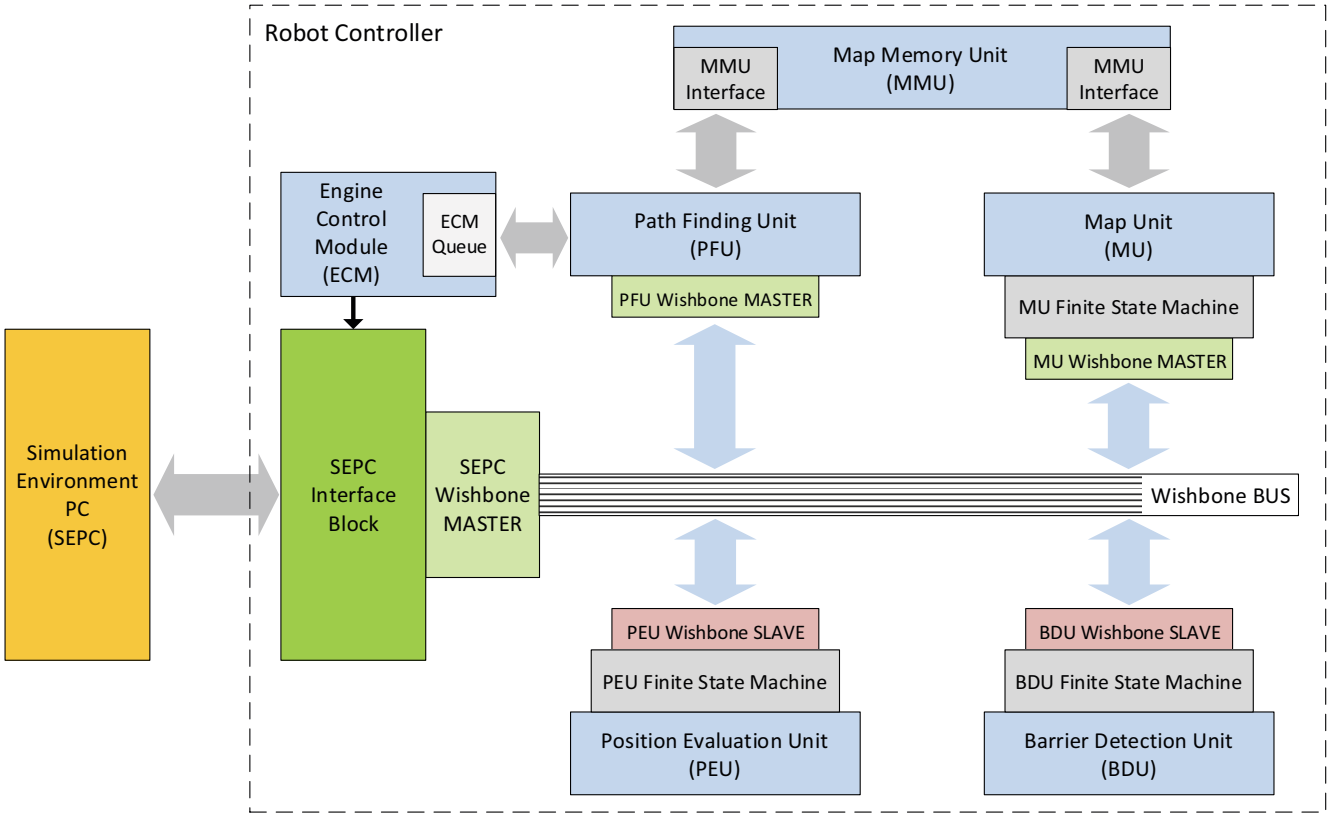

Fig. 4. The block diagram of the robot controller.

For effective testing of fault effects on a system composed of several blocks, we need to determine the block in which the fault will be injected. In the case of injecting faults into the whole FPGA we are not sure which block is affected, or if the useful part of the bitstream is hit. The implemented injector is able to inject faults only to specified bits of the configuration memory, a specification list of these bits is input parameter.

The list of bits representing each component is obtained through several steps. First, we perform synthesis using Xilinx synthesis tools [18]. The result of synthesis is a netlist, which serves as an input for the next step. Next, we use the PlanAhead [19] tool for the layout of the components on the FPGA. Thanks to this, we know where each of components is placed. The bitstream is generated in this step and the FPGA can be programmed. The knowledge about component layout allows us to use the RapidSmith [20] tool for analysing the design. This tool is able to generate a list of the bitstream bits that correspond to the identified areas of the FPGA, while we know what components are in each area. The disadvantage is that this process provides only a list of bitstream bits that correspond to *Lookup Tables* (LUTs). Our goal in the future will be to find a method which allows us to localize also bits of the bitstream corresponding to the interconnection network.

VI. The Experiment with the Robot Controller

The aim of the experiment is to identify which parts of the robot controller are vulnerable to faults. The flow of the experiment is displayed in Figure 5. At first, we initiate the environment of the robot in simulation. We generate a maze

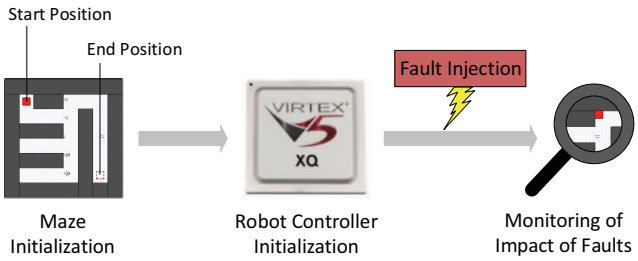

Fig. 5. The flow of one experiment.

together with the start and the end position for the mission of the robot. As the first scenario, we chose a small maze with 8x8 fields. The start position was in the upper left corner and the end position in the lower right corner. Subsequently, the robot controller is initiated. In particular, the bistream for the Virtex5 FPGA board is generated. When loaded, the robot starts to search a path to the end position. It moves quite slowly, one robot mission takes about one minute. At this point, the fault injection takes place. We generate randomly an LUT of every unit of the robot controller into which the fault will be injected. Thanks to the Rapidsmith, only corresponding bits of the bistream are inverted. We want to point out that we really target only bits if the bitstream belonging to the robot controller design. Other bits of the bitstream belonging to the unused parts of the FPGA or to the interconnection network are not affected. Faults are injected one after another (MTBF = 2s) until the robot starts to behave incorrectly or fails. We were monitoring (1) the number of faults that led to the malfunction of the robot and (2) how the behaviour of the robot was changed.

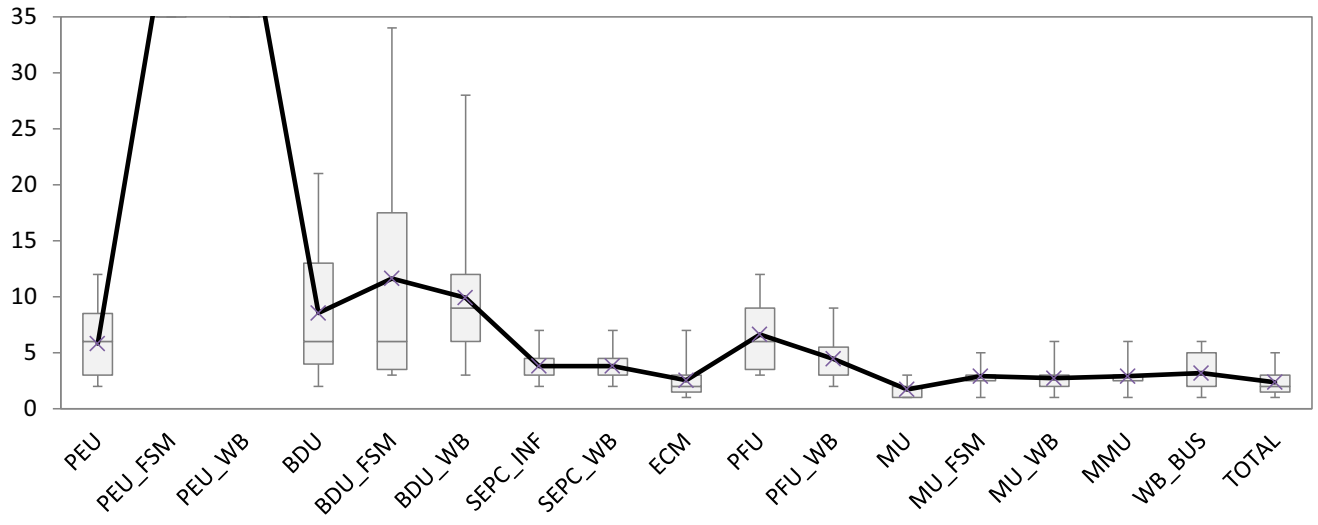

Fig. 6. The quartil graf of the results of experiments.

The results of the experiments are shown in Table I. In the first column, the list of components of the robot controller is provided. In the second column, the total number of bits of the bitstream that belong to the LUTs of corresponding components is shown. The following three columns represent the number of injected faults into particular components which caused incorrect behaviour of the robot. The first number is minimum, the second number is median and the last number is maximum of faults that led to failure. Injecting faults into all bits of the bitstream would be very time-consuming. Therefore, we utilise the statistic evaluation. 20 experimental runs were performed for each component (320 experimental runs in total). The last column of the table contains the state of the robot that was evaluated as the wrong behaviour. These states are described in more detail in the further text.

TABLE I. THE EXPERIMENTAL RESULTS.

Components	Bits of bitstream	Number of injected faults			Consequence
		Min	Median	Max	
PEU	21 632	2	6	12	freezing
PEU_FSM	2 112	>80	-	>80	-
PEU_WB	2 112	41	-	>80	freezing
BDU	320	2	6	21	freezing
BDU_FSM	2 752	3	6	34	freezing
BDU_WB	2 176	3	9	28	freezing
SEPC_INF	1 216	2	3	7	freezing
SEPC_WB	9 088	2	3	7	freezing
ECM	25 664	1	2	7	freezing
PFU	7 488	3	6	12	deadlock
PFU_WB	7 424	2	3	9	freezing
MU	11 840	1	2	3	crashing
MU_FSM	1 280	1	3	5	freezing
MU_WB	7 680	1	3	6	freezing
MMU	3 008	1	3	6	freezing
WB_BUS	5 056	1	3	6	freezing

The statistical data from the measures are also demonstrated in Figure 6. It is a quartile chart that for each component shows the minimum, the first quartile (25%), median, the second quartile (75%) and maximum of the measured number of injected faults that led to the failure. Moreover, the line across all components shows the average number of faults in each component that led to the failure. One interesting conclusion arises from the graph. The incorrect behaviour did not appear immediately after the first injection of a fault. We can conclude that some bits of the bitstream, despite they are identified as related to the robot controller, are not used to store a useful information. This can be seen particularly in components PEU_FSM and PEU_WB. There are several explanations of this, e.g. not all inputs of LUTs are employed or not all states of FSMs are visited during the computation. Nevertheless, we realised that some components contain more critical bits than others and thus they should be preferred while hardening against faults by some fault-tolerance methods.

The most common consequences of injected faults are:

- *Freezing on place*
  Freezing on one spot means that the robot suddenly stopped after the fault injection and did not continue in its mission.

- *Deadlock*
  After injection of certain number of faults the robot began to walk around in a cycle.

- *Crashing into a wall*
  In some cases, the robot did not recognise the occurrence of walls in the maze and repeatedly crashed to the wall.

- *Other*
  In the experiments, we observed a small number of other interesting consequences of faults. An example might be freezing of the robot in one place, then a re-freezing or walking in a cycle. We note also a wrong turn of the robot in the maze, which was followed by freezing.

The proportional representation of these consequences is displayed in Figure 7. As can be deduced from the chart, the most common consequence of injected faults is *Freezing on place*. We can also conclude that stopping of the robot is not so critical as for example, a collision with the wall. This conclusion can be very critical and useful for different kinds of EM applications.

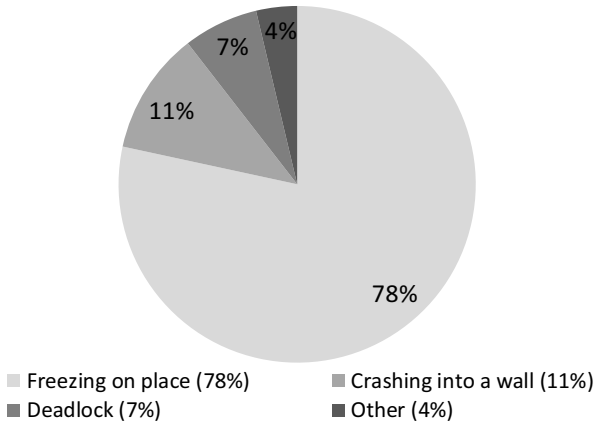

Fig. 7. The chart of typical consequences of injected faults on the mission of the robot.

VII. The Use of Functional Verification for Automated Evaluation of Fault Impacts

For extensive testing of the behaviour of the robot or any other EM system placed into our evaluation platform, we need to examine various test scenarios. After application of proper test vectors, we can prove the correctness and accuracy of the behaviour of the system with respect to the specification. The manual check of these test vectors is difficult as it requires a full control from the user. The user is responsible for running the test environment, generating test vectors and also analysing the outputs of the system. All these activities are time-demanding and therefore, it is not possible to test the system thoroughly within a reasonable time. It is necessary to apply some kind of automation. An extended technique for automated checking of the correctness of the system is called verification. There are several techniques used in the verification domain, but their description is not crucial for this work. We decided to use an approach called functional verification, as this type of verification fits best to our experiments.

Functional verification [21] is the process of verifying that a model of the system, also called DUT or Design Under Test, compliances with the specification by monitoring inputs and outputs in simulation. Moreover, the DUT outputs are compared to the outputs of the reference model (sometimes also referred to as the golden model) that is typically implemented by a verification engineer or a designer than did not implemented the DUT. On the basis of the compared outputs a discrepancy between the two models can be detected and thus an error in the systems can be discovered. The basic principle of functional verification is demonstrated in Figure 8. An important prerequisite for functional verification is also a good generator of input test vectors for testing all possible scenarios.

To be able to inject faults into the FPGA while performing functional verification, we must carry out verification directly in the FPGA (not in the simulation as usually). Advantageously we can use and modify hardware accelerated verification that uses an FPGA as the acceleration board. An example of such accelerator is the framework HAVEN [21]. The extension of our evaluation platform with the support of functional verification is shown in Figure 9. The DUT (in our case the

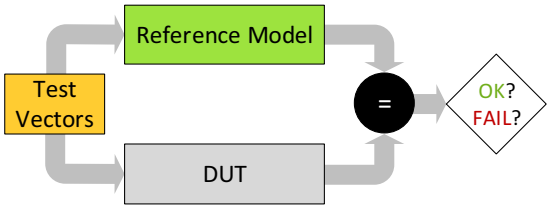

Fig. 8. The main principle of functional verification.

robot controller) will be placed on the FPGA. The outputs from the FPGA are compared to the outputs of the reference model and they represent also the inputs that are propagated to the simulation of the mechanical part. Thus, the output of the DUT stimulates the movement of the mechanical part of the robot in the simulated maze. The inputs for the FPGA and for the reference model are data from the sensors of the mechanical part of the robot.

As the reference model, a second implementation of the control unit, for example in SystemVerilog, C, SystemC, or the same VHDL implementation that is used as the DUT but without injected faults, can be considered. The Fault Injector is a feature that differentiates the current proposal from the classic functional verification. Using this feature we can verify that the fault-tolerance techniques used in the robot controller work properly and the robot behaves correctly also in the presence of faults injected into its controller.

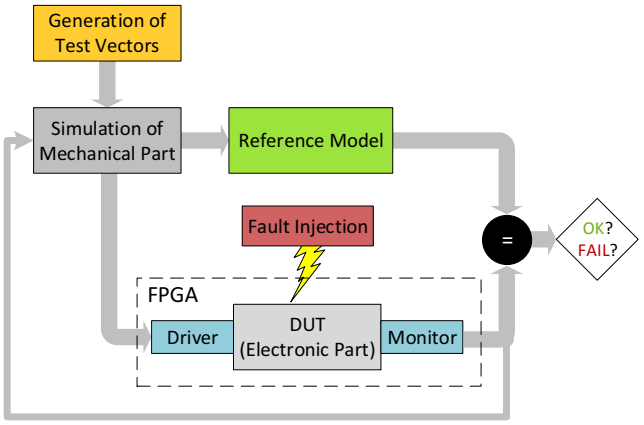

Fig. 9. Functional verification involvement in our platform with the fault injection.

The verification process will be divided into two phases:

1) *Verification of the electronic part only, without monitoring the impact of faults on the mechanical part.*

   Three possible outcomes can arise: (1) The output from the DUT and from the reference model is the same, an error did not appear. (2) The output is not identical but despite this, the robot has completed the mission (the robot reached the end position in the maze). (3) The output is not identical and at the same time, the mission was not accomplished. The last outcome is the most serious one and it will require a thorough analysis of the problem.

2) *Analysis of the faults, which affected the mechanical part.*

In this case, we will examine the faults that caused the failure of the mission of the robot. This activity will be carried out manually, since it is necessary to run the required experiment again and to monitor the behaviour of the mechanical part in simulation as described in the experimental part of this paper.

A very important element in the proposed platform is the generation of test vectors. To be able to check all working scenarios in functional verification and achieve the highest possible coverage of all key functions in the verified circuit the high-quality generator of inputs is needed. In our case, the generation aims at different mazes and different starting and end position of the movements of the robot. We also plan to use the generator for controlling injecting of faults (because now it is configured manually). We will generate signals that will drive the generation of faults and will determine when and into which place a fault should be injected.

VIII. Test Vector Generation

Generation of test vectors is our further goal. To prove the correct behaviour of the system according to its specification, testing the system on a wide set of input values is needed. We plan to adjust the generation of input test vectors to functional verification purposes and as an advantageous method seems to be an approach called *Coverage Directed Test Generation* (CDTG) [22] [23]. This method generates test vectors according to the defined design conditions and limitations which are called constraints. The main challenge in the generation of test vectors is to achieve maximal coverage of the system key functions. If a system function remains unverified, this method will define additional constraints in order to get this feature covered. At the end, the coverage report which is the result of the simulation runtime of verification is created.

Thanks to CDTG we will acquire two important advantages. The first is the possibility that the uncovered features of the system become covered and a higher level of coverage will be achieved. The second advantage is in testing certain scenarios multiple times for different input values.

Figure 10 shows the proposed method of generating test vectors. This method is not limited only to generation of inputs for the robot controller which will be described in the next paragraph. It represents a universal approach that can be used to generate inputs for different kinds of systems. The basic elements of the universality of the generator are two separate pseudo-formal models. The first model labelled as the *Problem Description* contains information about the scenario we want to generate. It may contain information about variables, data types, static values or substitutes that we want to generate. In simple words, this model defines what we want to generate. The second model labelled as the *Constraints for the Problem* describes how the scenario defined in the Problem Description should be generated. This model thus contains constraints that should be taken into account while generating the scenario. This is essentially a limit for data values, such as a variable cannot take certain values from the range of the data type, or restriction of dependency, such as some combination of variables cannot occur after the currently generated combination. Both of these models are inputs to the generator of test vectors that is currently in the implementation phase.

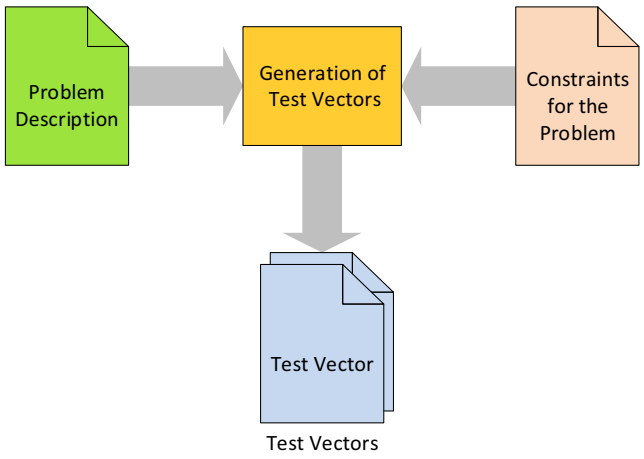

Fig. 10. The principle of the constraint generator.

Figure 11 shows an example of generating the mazes for the robot device. This is a simple example that shows the function of above mentioned approach. The problem of generating the maze is defined as the generation of lines that are represented by the boolean array of specific size. The constraints restrict the minimal width of the corridor of the maze, the walls of the maze can be only rectangular and a room that have no path cannot appear in the maze. The result obtained by the generator is a sequence of rows that consists of zeroes or ones. Zeroes represent the corridors, ones represent the walls. This generated output may be further processed. In our case, this output is regenerated into a bitmap image representing the desired maze for the robot.

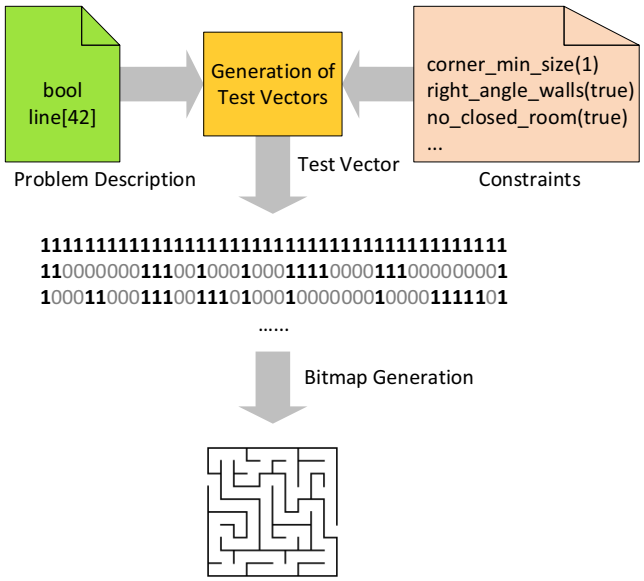

Fig. 11. An example of generating a maze for the robot controller.

IX. Conclusion and Future Work

In this paper, we introduced the evaluation platform for estimating reliability of FPGA designs. As our research focuses on testing EM applications, we presented the experimental design which is composed of the mechanical robot and its electronic controller situated in the FPGA. The robot controller contains a variety of components. During the experiments, random faults were artificially injected into these components and we were monitoring impact of these faults on the behaviour of the robot in the simulation environment. These experiments showed that some faults have an impact on the behaviour of the robot, and others do not have. According to this result we were able to identify the parts/components of the robot controller that need to be hardened by some fault-tolerance techniques.

In addition, we have recognised from the experiments that some kind of automation is unavoidable in our future experiments, especially in the early phases of testing. The reason is that monitoring the behaviour of system in simulation is very time-demanding. Therefore, we have already prepared an innovative extension of our platform - interconnection of fault injection and functional verification environment with advanced test generation. Using this approach we will be able to automatically verify an EM system during the fault injection. The automation is achieved by comparing the outputs of the verified system to the reference model that is in our case represented by the same design but without injected faults.

Acknowledgment

This work was supported by the following projects: National COST LD12036 - "Methodologies for Fault Tolerant Systems Design Development, Implementation and Verification", project Centrum excelence IT4Innovations (ED1.1.00/02.0070), EU COST Action IC1103 - MEDIAN - Manufacturable and Dependable multIcore Architectures at Nanoscale and BUT project FIT-S-14-2297.

References

[1] S. Cutts, "A collaborative approach to the more electric aircraft," in *Power Electronics, Machines and Drives, 2002. International Conference on (Conf. Publ. No. 487)*, June 2002, pp. 223–228.

[2] J. Bennett, A. Jack, B. Mecrow, D. Atkinson, C. Sewell, and G. Mason, "Fault-tolerant control architecture for an electrical actuator," in *Power Electronics Specialists Conference, 2004. PESC 04. 2004 IEEE 35th Annual*, vol. 6, June 2004, pp. 4371–4377 Vol.6.

[3] G. Leen and D. Heffernan, "Expanding automotive electronic systems," *Computer*, vol. 35, no. 1, pp. 88–93, Jan 2002.

[4] 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," *Nuclear Science, IEEE Transactions on*, vol. 50, no. 6, pp. 2088–2094, 2003.

[5] C. Bolchini, A. Miele, and M. D. Santambrogio, "TMR and Partial Dynamic Reconfiguration to Mitigate SEU Faults in FPGAs," in *DFT '07: Proceedings of the 22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 87–95.

[6] J. Emmert, C. Stroud, B. Skaggs, and M. Abramovici, "Dynamic Fault Tolerance in FPGAs via Partial Reconfiguration," in *FCCM '00: Proceedings of the 2000 IEEE Symposium on Field-Programmable Custom Computing Machines*. Washington, DC, USA: IEEE Computer Society, 2000, pp. 165–170.

[7] J. A. Cheatham, J. M. Emmert, and S. Baumgart, "A Survey of Fault Tolerant Methodologies for FPGAs," vol. 11, no. 2. New York, NY, USA: ACM, 2006, pp. 501–533.

[8] L. Sterpone, M. Aguirre, J. Tombs, and H. Guzmán-Miranda, "On the Design of Tunable Fault Tolerant Circuits on SRAM-based FPGAs for Safety Critical Applications," in *DATE '08: Proceedings of the conference on Design, automation and test in Europe*. New York, NY, USA: ACM, 2008, pp. 336–341.

[9] N. Rollins, M. Fuller, and M. Wirthlin, "A comparison of fault-tolerant memories in sram-based fpgas," in *Aerospace Conference, 2010 IEEE*, 2010, pp. 1–12.

[10] M. Naseer, P. Sharma, and R. Kshirsagar, "Fault tolerance in fpga architecture using hardware controller - a design approach," in *Advances in Recent Technologies in Communication and Computing, 2009. ARTCom '09. International Conference on*, 2009, pp. 906–908.

[11] L. Frigerio and F. Salice, "Ram-based fault tolerant state machines for fpgas," in *Defect and Fault-Tolerance in VLSI Systems, 2007. DFT '07. 22nd IEEE International Symposium on*, 2007, pp. 312–320.

[12] 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.

[13] 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.

[14] Z. Vasicek. (2014, Apr.) FITkit. [Online]. Available: www.fit.vutbr.cz/FITkit

[15] J. Podivinsky, M. Simkova, and Z. Kotasek, "Complex Control System for Testing Fault-Tolerance Methodologies," in *Proceedings of The Third Workshop on Manufacturable and Dependable Multicore Architectures at Nanoscale (MEDIAN 2014)*. COST, European Cooperation in Science and Technology, 2014, pp. 24–27.

[16] OPENCORES. (2014, Apr.) Wishbone B4: WISHBONE System-on-Chip (SoC) Interconnection Architecture Portable IP Cores. [Online]. Available: http://cdn.opencores.org/downloads/wbspec_b4.pdf

[17] R. Oliveira, A. Jagirdar, and T. J. Chakraborty, "A TMR Scheme for SEU Mitigation in Scan Flip-Flops," in *ISQED '07: Proceedings of the 8th International Symposium on Quality Electronic Design*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 905–910.

[18] XILINX, "Xst User Guide."

[19] N. Dorairaj, E. Shiflet, and M. Goosman, "Planahead software as a platform for partial reconfiguration," *Xcell Journal*, vol. 55, no. 68-71, p. 84, 2005.

[20] C. Lavin, M. Padilla, P. Lundrigan, B. Nelson, and B. Hutchings, "Rapid prototyping tools for fpga designs: Rapidsmith," in *Field-Programmable Technology (FPT), 2010 International Conference on*, Dec 2010, pp. 353–356.

[21] M. Simkova, O. Lengal, and M. Kajan, "Haven: An open framework for fpga-accelerated functional verification of hardware," Tech. Rep., 2011. [Online]. Available: http://www.fit.vutbr.cz/research/view_pub.php.en?id=9739

[22] M. George and O. Ait Mohamed, "Performance analysis of constraint solvers for coverage directed test generation," in *Microelectronics (ICM), 2011 International Conference on*, 2011, pp. 1–5.

[23] H. Shen, P. Wang, Y. Chen, Q. Guo, and H. Zhang, "Designing an effective constraint solver in coverage directed test generation," in *Embedded Software and Systems, 2009. ICESS '09. International Conference on*, 2009, pp. 388–395.