Radiation Impact on Mechanical Application Driven by FPGA-based Controller

Jakub Podivinsky, Marcela Simkova, Zdenek Kotasek
Faculty of Information Technology, Brno University of Technology
Bozetechnova 2, 612 66 Brno, Czech Republic
Email: {ipodivinsky, isimkova, kotasek}@fit.vutbr.cz

***Abstract*—The aim of this paper is to present results of preliminary experiments with our platform for testing the fault-tolerance quality of electro-mechanical applications based on FPGAs. Original work is presented in [1]. 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. In the experiments, the mechanical robot is simulated in the simulation environment where the effects of faults injected into its controller can be recognised. 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.**

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 [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. 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).

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

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

II. The Goals of the Research

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 controlling electro-mechanical applications.

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 [8] are validated on simple memories without the additional logic around. In [9], the fault-tolerance technique is presented only on a two-input multiplexer, one simple adder and one counter. 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. 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).

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

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

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. We have implemented the evaluation platform that consists of three basic parts (as you can see in Figure 1):

- the Virtex5 FPGA board, where the robot controller is situated after the synthesis and the place and route process,
- the simulation environment Player/Stage [10] for checking responses of the mechanical device to instructions from the robot controller,
- the external fault injector (PC) which inserts faults into the robot controller [11].

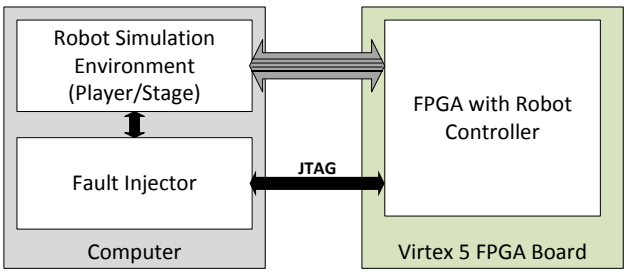

Fig. 1. The platform for testing fault-tolerance methodologies.

III. The Robot Controller - Structure and Principles

In Figure 2, the block diagram of the implemented robot controller is available. The control unit is connected to the PC with simulation environment (SEPC) via the Interface Block. Through this block, data from the simulation are received and in the opposite direction, instructions about the movement of the robot are sent back. The robot controller is composed of various blocks, their function is described in [12]. Here, we only summarize the main characteristics of every component. The central block of the robot controller is a bus through which the communication between blocks is accomplished. The Position Evaluation Unit (PEU) calculates the position of the robot in the maze and provides them to other units as coordinates x and y. The Barrier Detection Unit (BDU) uses four sensors and provides information about the distance to the surrounding barriers as four-bit vector. Map updating provided by the Map Unit (MU) is based on the information about the position of the robot and the four-bit barriers vector. The Map Memory Unit (MMU) stores the information about the up-to-date map. Path Finding Unit (PFU) implements simple iteration

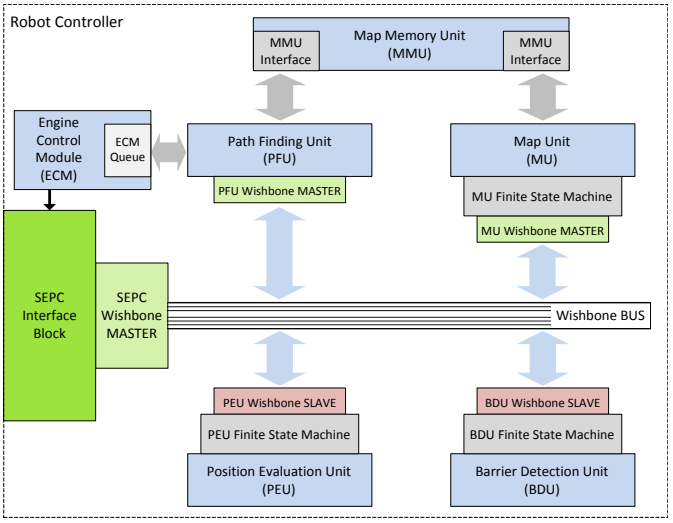

Fig. 2. The block diagram of the robot controller.

algorithm for finding a path through the maze. 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 individual or cooperating fault-tolerance methodologies focused on FPGAs. The controller contains combinational and sequential circuits, finite state machines, memories or buses.

IV. Evaluation of Reliability by Fault Injection

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. The most popular techniques to artificially accelerate fault occurrence are based on *fault injection*.

Therefore, to simulate the 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, a fault injector [11] was implemented which allows to modify single or multiple specified bits of the bitstream in order to simulate single and multiple faults.

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 list of bits representing each component can be obtained through several steps by using the PlanAhead [13] tool for the layout of the components on the FPGA. The knowledge about component layout allows us to use the RapidSmith [14] 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 configured into particular area. The disadvantage of such approach is that this process provides only a list of bitstream bits that correspond to *Lookup Tables* (LUTs).

V. Experiments 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 3. At first, we initiate the environment of the robot in simulation. As the first scenario, we chose a small maze with 8x8 fields. Subsequently, the robot controller is initiated. Then the robot starts to search a path to the end position. 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, just the corresponding bits of the bistream are inverted. Faults are injected one after another until the robot starts to behave incorrectly. 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.

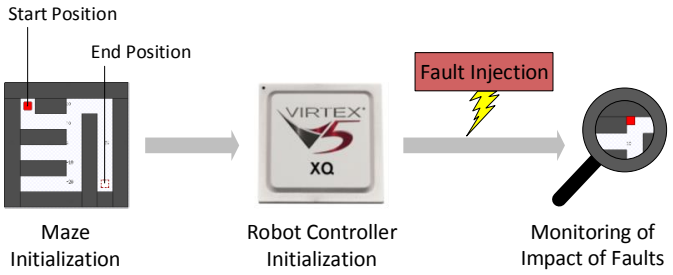

Fig. 3. The flow of one experiment.

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, because behaviour was monitored manually. 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.

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 4. 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. 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.

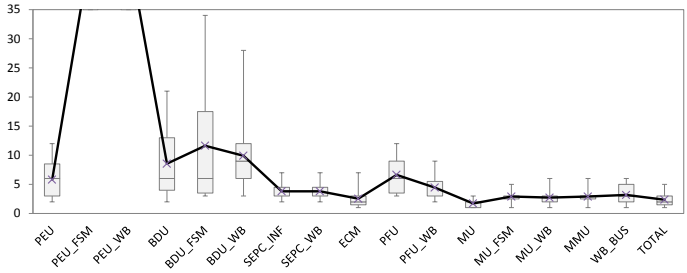

Fig. 4. The quartil graf of the results of experiments.

The proportional representation of consequences is displayed in Figure 5. The most common consequences of injected faults which are presented in table are Freezing on place, Deadlock, Crashing into a wall and some others. As can be deduced from the chart, the most common consequence 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.

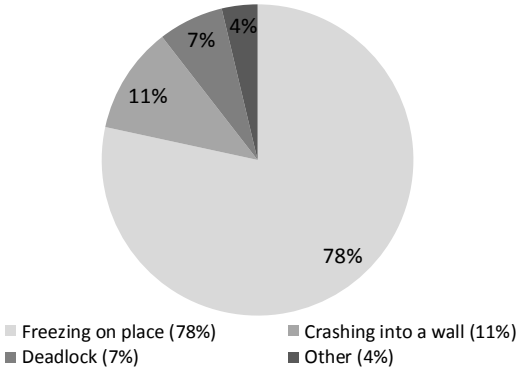

Fig. 5. Typical consequences of injected faults on the mission of the robot.

VI. 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 stimuli, we can prove the correctness and accuracy of the behaviour of the system with respect to the specification. The manual check of outputs of the system for these stimuli is difficult as it requires a full control from the user. The user is responsible for running the test environment, generating stimuli 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. We decided to use an approach called functional verification, as this type of verification fits best to our experiments. Functional verification [15] is the process of verifying that a model of the system, also called DUT or Design Under Test, complies with the specification by monitoring inputs (stimuli) and outputs in simulation. Moreover, the DUT outputs are compared to the outputs of the reference model. 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.

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 [15]. The extension of our evaluation platform with the support of functional verification is shown in Figure 6. The DUT (in our case the robot controller) will be placed in 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.

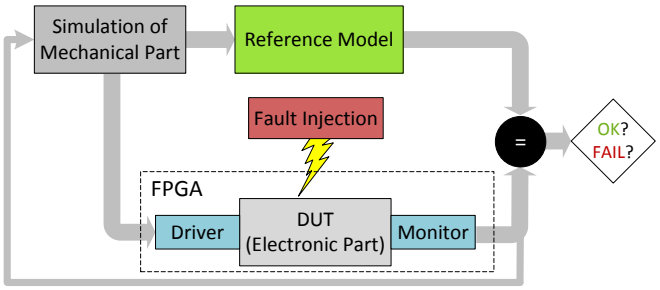

Fig. 6. Functional verification involvement in our platform with the fault injection.

VII. Conclusions 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 the 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.

Acknowledgment

This work was supported by the following projects: National COST LD12036, project Centrum excelence IT4Innovations (ED1.1.00/02.0070), EU COST Action IC1103 ”MEDIAN” and BUT project FIT-S-14-2297.

References

[1] J. Podivinsky, O. Cekan, M. Simkova, and Z. Kotasek, “The evaluation platform for testing fault-tolerance methodologies in electro-mechanical applications,” in *Digital System Design (DSD), 2014 17th Euromicro Conference on*. IEEE, 2014, pp. 312–319.

[2] 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.

[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] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] M. Simkova and O. Lengal, “Towards beneficial hardware acceleration in haven: Evaluation of testbed architectures,” *Lecture Notes in Computer Science*, vol. 2013, no. 7857, pp. 266–273, 2012.