# Reliability Analysis of the FPGA Control System with Reconfiguration Hardening

Richard Panek, Jakub Lojda, Jakub Podivinsky, Zdenek Kotasek
Faculty of Information Technology, Brno University of Technology, Centre of Excellence IT4Innovations
Bozetechova 2, 612 66 Brno, Czech Republic
Email: {ipanek, ilojda, ipodivinsky, kotasek}@fit.vutbr.cz

***Abstract*—A computing power is important in space applications where a utilization of FPGAs is very useful. However, the FPGAs are susceptible to manifestations of radiation which can cause malfunction. Particularly dangerous are configuration memory faults known as Single Event Upsets (SEUs), which can lead to the entire system failure. Therefore, the fault-tolerant techniques are used to prevent system failures. The main motivation for the use of these techniques is to maintain the correct behavior of the system despite the occurrence of faults. In addition to fault masking, which only delays system failures due to fault accumulation, the utilization of fault mitigation by partial dynamic reconfiguration was used. Everything needed is provided by the reconfiguration controller, which is a necessary additional component of the entire system. It is also very convenient to be able to detect the occurrence of fault in the system. After that, the system need not be restored unnecessarily, which saves useless work of the controller. The aim is to evaluate the benefit of reconfiguring damaged parts of the system to increase fault tolerance. In all experiments, an experimental platform was used that emulates an electromechanical system, which consists of a robot control unit on an FPGA and a simulation of their behavior on a PC. Artificial faults have been injected into the FPGA configuration memory that corresponds to this controller.**

***Keywords*—Fault Tolerance, Partial Dynamic Reconfiguration Controller, FPGA, Reliability Analysis.***

## I. Introduction

*Field Programmable Gate Arrays (FPGAs)* are widely used in various systems, where the emphasis is mainly on computing power and also on flexibility. However, when used in higher radiation environments such as space applications, configuration memory faults known as *Single Event Upsets (SEUs)* [1] occur. Such configuration corruption will cause an unexpected change in behavior, and this can lead to a total system failure. *Fault Tolerance (FT)* [2] techniques ensure that the system functions properly despite the occurrence of such faults.

The effect of radiation on the FPGA is described in details in [1]. The individual types of failures are classified here and each is accompanied with a thorough description of the interference, in which parts they arise and what they cause. Finally, the authors state the need to be able to repair SRAM-based FPGAs for the use in space applications such as using scrubbing. Not only the use of TMR, but also other options to ensure fault tolerance are discussed in [3]. These are different approaches from fault avoidance to various methods of repair after failure. It also deals with increasing the resilience of *Block Random Access Memories (BRAMs)* that are present on the FPGA chip. The authors of Article [4] also compare different approaches to scrubbing, i.e. repair after a failure by means of reconfiguration. They discuss either the position of the scrubbing control unit or evaluate a new cell for SRAM and its possible use to increase durability. The use of FPGAs in space is also discussed in the paper [5], where the authors introduced nanosatellite equipment into which various research systems can be implemented. Reconfiguration is used here to change the function, but also to recover from a failure. In addition, a strategy is prepared for less and more important parts, so that the management system, for example, has priority over others. There are also several possible stages of reconfiguration, where at first they try to repair only small parts of the FPGA and in case of failure then the whole FPGA. Paper [6] focuses on testing different approaches to increasing fault tolerance. It introduces the simulation of different environments by artificially injecting faults into the design in HDL and subsequent evaluation. Such an evaluation must be more time consuming than direct injection into a running system and, moreover, does not lead to its configuration. The creation of robust system based on softcore processors is dealt with in paper [7]. It is based on two processors in a Duplication with Comparison (DwC) scheme and a reconfiguration controller implemented in other softcore processors and in addition in TMR. The unit comparing the outputs of both processors is further provided with context switching logic. In case of a fault, the processor is shut down, reconfigured and then synchronized using the context recovery block. This is a very useful approach for a system implemented in a processor. We based the design on the approach described in [8], i.e. to secure the system by masking with the help of TMR and to repair the failed module by partial dynamic reconfiguration. In addition, we will compare this approach with previous methods without reconfiguration and thus evaluate the possible contribution to the resilience of the system in a real environment.

This article is organized as follows. Section II is devoted to the overall platform on which robustness is evaluated. An integral part of it is also a fault injector and a tested system. It also deals with the used reconfiguration controller and discusses the preparation of experiments to evaluate the resilience of the system to failures. The following is an evaluation of the experiments from several perspectives in Section III. These are the system resilience in different environments and the benefits of reconfiguration. Finally, the whole paper is summarized in Section IV.

978-1-6654-2703-6/21/$31.00 ©2021 IEEE
DOI 10.1109/DSD53832.2021.00089

## II. Evaluation Platform for Fault Tolerance Testing

An experimental system designed specifically for this purpose was chosen to investigate fault tolerance. This system consists of an electronic control unit implemented to the FPGA and a controlled mechanical part [9]. This approach allows us to monitor the effect of faults both on the output of the electronic part and on the behavior of the mechanical part. Furthermore, to increase its resilience, a TMR will be used, which will be further equipped with the ability to repair the affected module by means of reconfiguration. The reconfiguration of the damaged module itself will be controlled by its controller, a specially added component to the system. The *Generic Partial Dynamic Reconfiguration Controller (GPDRC)* [10] that is implemented directly into the FPGA was selected.

For our experiments, we used ML506, an evaluation board with Virtex 5 FPGA from Xilinx. The reason for our choice are pre-built proven tools, such as an external fault injector [11], which uses partial reconfiguration to swap one bit in the FPGA configuration memory and communicates via JTAG. The main advantage of this concept is the ability to test the design without any additional components on the FPGA, so the final design is tested without any modifications. Only exclusive access to the configuration memory between the reconfiguration controller and the fault injector must be provided.

The exact composition of the resulting experimental system is shown in Figure 1, where it is possible to see the distribution of individual units between the PC and the evaluation board with FPGA. An evaluation platform runs on the PC, which evaluates the control signals from the examined unit on the FPGA, it also provides its configuration and fault injection into the configuration memory. The control signals are transmitted via Ethernet and other communication is provided via the JTAG interface and Platform Cable, which is connected via USB to PC. Virtex-5 FPGA and flash memory are used from the evaluation board. The investigated robot controller in TMR with the majority that can determine the failed module is on the FPGA. This is exactly the part of the FPGA into which faults are injected, i.e. more precisely into the relevant configuration memory. Another unit on the FPGA provides communication between the robot controller and the monitor on the PC. The last unit is the GPDRC reconfiguration controller, it is based on information from the voter and ensures the reconfiguration of the particular failed module. Thus, it reads the relevant data (golden bitstream) from the flash memory and ensures its loading into the configuration memory via the ICAP interface.

### A. Description and setup of Experiments

Our experimental system, the robot controller, is tested in various degrees of a critical environment, which is given by the number of faults per time unit per bit of configuration memory of the tested system. The very useful unit is $inj/s/bit$ introduced in [12], which describes the same state, because the faults injection corresponds to the faults that occurred naturally.

The individual experimental runs are divided into sets with the same intensity of disturbances. Each such set of experiments consists of 5000 runs. In one run, the robot takes 204 s to pass through the maze from the starting position to the desired target. The maze consists of a square grid, where the individual squares are either a corridor or a wall. If the robot is lost, the experiment is terminated after the 324 s timeout.

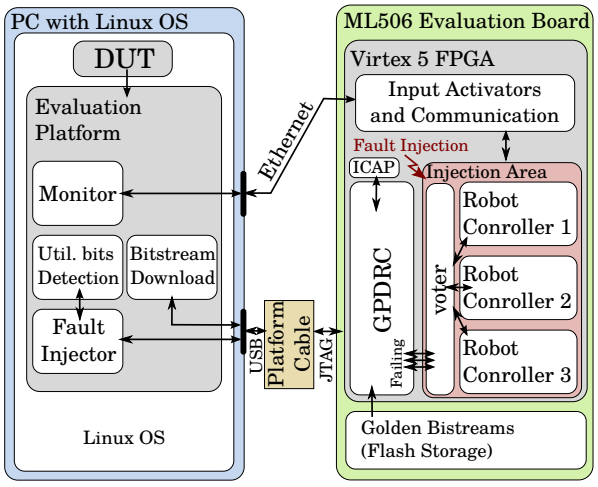


Figure 1: The overview of our evaluation platform alongside with the TMR version of the robot controller, including the GPDRC.

Each of the 3 robot controller modules in the TMR occupies 331 kB of configuration memory, which together corresponds to almost 1 MB and thus 8 million bits that can be affected by the fault. Therefore, to speed up the evaluation of fault tolerance, we use injection only into configuration bits that correspond to the design LUTs used. There are only 353 440 bits for comparison, i.e. about 25 times less. The obtained results will be comparable, because this method is used in all experiments.

### B. Reconfiguration of Sequential Circuits

Our initial intention to create a fault-tolerant control system was to extend the majority voter by identifying the module affected by the fault and adding a reconfiguration controller that reconfigures the failed module. For the subsequent synchronization of individual TMR modules, their `Reset` signal would be used, which would reset them to the default state. Then the operation of the whole system would be virtually uninterrupted. This original assumption of ours turned out to be wrong, because after the repair, the system cannot return to its initial state, but must at least partially retain its state. It is therefore necessary for each system, based on its careful analysis, to design a method of synchronization of individual modules, i.e. to ensure that the reconfigured module gets into the same state as other undamaged modules before commissioning.

After a thorough analysis, the moment when the robot in the middle of the maze field decides where to go next was chosen for the synchronization of the modules after the reconfiguration. At this point, only the direction in which the robot got to this field needs to be shared between the modules. So after the reconfiguration repair, the repaired module is kept in reset until the appropriate time for synchronization occurs. Then the values from the other modules are copied to this module via the majority and thus the functionality is completely restored.

## III. Experimental Results

The results of the experiments are divided into two parts, which correspond to the following views:

- At first perspective, the contribution of reconfiguration to fault tolerance depending on the growing level of the critical environment is investigated.
- Furthermore, the benefit of reconfiguration to increase system fault tolerance is compared to previous versions, where the system was unprotected or only equipped with TMR to increase the resilience to faults.

### A. Fault Tolerance of the Reconfigured System

Thanks to the chosen evaluation platform, the resilience of the robot controller itself is monitored, but also the impact on its physical activity. Therefore, there may be a situation where, despite the failure of the electronics, the robot arrives at its destination. The "natural" resilience of the physical system to control failure is also taken into account.

The results of the individual sets are summarized in the Table I. The first columns are the environment specification, which is expressed by the number of faults injected per second per bit of configuration memory. For better imagination, the intensity of faults is expressed as the frequency of faults, i.e. the average time between the occurrence of individual faults. This interval is randomly selected with a uniform distribution from a 2 second range with the mean value given in the table. The reason is to get as close as possible to the natural environment. Each fault is injected only into the configuration memory of the robot controller, i.e. into all its modules in the TMR. The next columns show the number of runs in which the robot reached the expected goal, as well as their percentage. The next columns, on the other hand, represent the number and the percentage of runs in which the goal was not reached. The last columns summarize the runs during which the electronics failed, i.e. the robot controller does not provide the expected control signals.

TABLE I: The Classification of Results According to Goal Achievement and Electronics Failure Depending on the Intensity of Faults

| Intensity of faults | | Goal reached | | Goal failed | | Electronics failure | |
|---|---|---|---|---|---|---|---|
| avg [s] | [inj/s/bit] | [-] | [%] | [-] | [%] | [-] | [%] |
| 28 | 1.01E-07 | 4964 | 99.28% | 36 | 0.72% | 48 | 0,96% |
| 21 | 1.35E-07 | 4951 | 99.02% | 49 | 0.98% | 56 | 1,12% |
| 15 | 1.89E-07 | 4901 | 98.02% | 99 | 1.98% | 117 | 2,34% |
| 10 | 2.83E-07 | 4783 | 95.66% | 217 | 4.34% | 275 | 5,50% |
| 8 | 3.54E-07 | 4717 | 94.34% | 283 | 5.66% | 343 | 6,86% |
| 6 | 4.72E-07 | 4517 | 90.34% | 483 | 9.66% | 627 | 12,54% |
| 4 | 7.078E-07 | 4083 | 81.66% | 917 | 18.34% | 1123 | 22,46% |

The expected trend was confirmed, with the increasing number of injected faults per second per bit, also the number of runs of experiments in the set increases, during which the robot does not reach the expected target in the maze. The number of runs when the electronics failed is growing at a faster rate. With the intensity of faults, the number of runs during which the robot reached the goal despite the failure of the electronics also increases. For a better view of these dependencies, they are plotted in the chart in the Figure 2. The second half of the chart, i.e. the higher intensity of faults, shows in both cases a practically linear increase in the incidence of failures. In contrast, the part with a lower incidence of disturbances indicates a gradual, slow convergence to zero. We explain this in such a way that, despite the sufficient time for reconfiguration and subsequent synchronization of the modules, in some cases a situation may occur in which the fault simply results in an error. Another possibility may be that a fault in the configuration memory will cause an error in the data, and therefore its repair itself may not immediately lead to the restoration of a fault-free state.

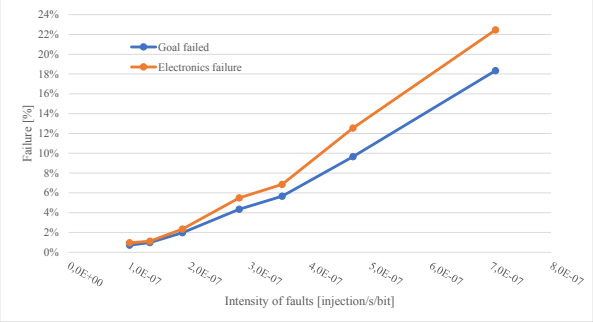


Figure 2: Chart of failure dependencies (runs of experiment where the goal is not reached and when the electronics failed) on the level of critical environment expressed by the number of faults per unit time and area.

### B. Improving Fault Tolerance by Using Reconfiguration

The goal of using reconfiguration is to provide a higher level of fault tolerance and thus not only to disguise the faults but also to be able to recover from them. Therefore, the obtained results of the experiment are compared with the results of the previous research [13]. The robustness of the basic system without providing techniques to increase the FT and the version in TMR were investigated there. The results are simply comparable, as it is an incremental increase in fault tolerance by first only masking the faults and then adding the possibility to reconfigure the modules affected by the fault. The same conditions were always set, i.e. 5000 runs and injections every 15 seconds into the control unit in the TMR (corresponding to approximately 45 seconds into each of the units). This comparison is summarized in Table II. At the top there are the numbers of successful runs in which the robot reached the desired target and also the runs in which it failed. The stated values are given both in absolute terms and their percentage in individual sets. The lower part of this table then expresses the percentage improvement in reliability according to the Equation 1, where *new* is with higher robustness and with *original* it is compared.

TABLE II: The Comparison of Fault Tolerance of the Original Version, Version Using Only TMR and TMR with Reconfiguration

| | noFT[13] | | TMR[13] | | Reconfiguration | |
|---|---|---|---|---|---|---|
| | [-] | [%] | [-] | [%] | [-] | [%] |
| Goal reached | 3571 | 71.42% | 4839 | 96.78% | 4901 | 98.02% |
| Goal failed | 1429 | 28.58% | 161 | 3.22% | 99 | 1.98% |
| Reliability improvement | noFT | | 88.73% | | 93.07% | |
| | | | TMR | | 38.51% | |

$$reliab\_improv = \frac{failures_{original} - failures_{new}}{failures_{original}} \cdot 100 \quad (1)$$

The above comparison shows a more than $90\%$ improvement in reliability over the original unprotected version. Adding reconfiguration to TMR version increases resiliency by almost $40\%$. This improvement is at the expense of increasing the occupied area on the FPGA by approximately $17\% - 24\%$ compared to the version of the robot controller in TMR. More details about the occupied area of each version are given in Table III, where it is possible to see how much space the individual designs of Registers, LUTs and Slices occupy. Then there is an increase in the version with reconfiguration compared to other versions in individual types of resources.

TABLE III: Utilization of FPGA Resources in Individual Versions of the Robot Controller Design

| Version | Register | LUT | Slice |
|---|---|---|---|
| noFT [14] | 1617 | 1708 | 1080 |
| TMR [14] | 4755 | 5165 | 2991 |
| Reconfiguration | 5887 | 6264 | 3495 |
| Increase Reconfiguration resources to: | | | |
| noFT | 264% | 267% | 224% |
| TMR | 24% | 21% | 17% |

## IV. Conclusions

The fault tolerance of the control system on the FPGA equipped with fault masking by means of TMR and correction by means of reconfiguration was evaluated in this paper. This combination was chosen so that the system could operate continuously without interruption to resume operation after a failure occurred. These approaches were provided with a robot controller, the aim of which is to find its way to the target by navigating the maze. The main advantage of its use was that it is a part of a platform that evaluates fault tolerance in terms of both electronics and physical expression. Thus, the effects of controller failures on the behavior of the robot itself and whether it still achieves the desired goal were evaluated. The faults were artificially injected directly into the FPGA configuration memory.

First, it was evaluated how our system will perform in environments with different intensity of failures. As expected, with the increasing intensity of failures, the probability of system failure also increased. However, even the very low intensity of failures led to a significant probability of system failure. Therefore, to further increase the resilience, it would be appropriate to add another stage, for example, the possibility of reconfiguring the entire FPGA with an external way.

Furthermore, the benefit of reconfiguration compared to a system using only TMR and a non-resilient system was evaluated. A system with reconfiguration repair is by $93\%$ more reliable than the original system. Compared to the TMR version, the probability of failure has been reduced by $38\%$.

Future work will be aimed at increasing the resilience of our reconfiguration controller itself, which is one of the last remaining unprotected elements of the entire robustness system. This is an important part of our work, where we want to compare the various approaches to the implementation of these controllers and evaluate their impact on the resulting system. We consider it crucial to compare different approaches to control the reconfiguration of an autonomous system, where such a controller can be directly on the FPGA with the system, i.e. either hardcoded or as a processor (soft or hard core), as well as an external component.

## Acknowledgements

This work was supported by the BUT project FIT-S-20-6309 and the JU ECSEL Project SECREDAS (Product Security for Cross Domain Reliable Dependable Automated Systems), Grant agreement No. 783119.

## References

[1] H. Quinn, "Radiation effects in reconfigurable FPGAs," *Semiconductor Science and Technology*, vol. 32, no. 4, p. 044001, mar 2017. [Online]. Available: https://doi.org/10.1088/1361-6641/aa57f6

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

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

[4] T. S. Nidhin, A. Bhattacharyya, R. P. Behera, and T. Jayanthi, "A Review on SEU Mitigation Techniques for FPGA Configuration Memory," *IETE Technical Review*, vol. 35, no. 2, pp. 157–168, 2018. [Online]. Available: https://doi.org/10.1080/02564602.2016.1265905

[5] C. Fuchs, N. Murillo, A. Plaat, E. Kouwe, D. Harsono, and T. Stefanov, "Fault-Tolerant Nanosatellite Computing on a Budget," 09 2018.

[6] T. S. Nidhin, A. Bhattacharyya, R. P. Behera, T. Jayanthi, and K. Velusamy, "Dependable system design with soft error mitigation techniques in SRAM based FPGAs," in *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*, 2017, pp. 1–6.

[7] H. Pham, S. Pillement, and S. J. Piestrak, "Low-Overhead Fault-Tolerance Technique for a Dynamically Reconfigurable Softcore Processor," *IEEE Transactions on Computers*, vol. 62, no. 6, pp. 1179–1192, 2013.

[8] C. Bolchini, A. Miele, and M. D. Santambrogio, "TMR and Partial Dynamic Reconfiguration to Mitigate SEU Faults in FPGAs," in *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*, Sept 2007, pp. 87–95.

[9] 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.

[10] M. Straka, J. Kaštil, and Z. Kotásek, "Generic Partial Dynamic Reconfiguration Controller for Fault Tolerant Designs Based on FPGA," in *NORCHIP 2010*. IEEE Computer Society, Nov 2010, pp. 1–4.

[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. Lojda, J. Podivínský, Z. Kotásek, and M. Krčma, "Majority Type and Redundancy Level Influences on Redundant Data Types Approach for HLS," in *2018 16th Biennial Baltic Electronics Conference (BEC)*. IEEE Computer Society, 2018, pp. 1–4. [Online]. Available: https://www.fit.vut.cz/research/publication/11600

[13] J. Podivinsky, J. Lojda, O. Cekan, and Z. Kotasek, "Evaluation platform for testing fault tolerance properties: Soft-core processor-based experimental robot controller," in *2018 21st Euromicro Conference on Digital System Design (DSD)*, 2018, pp. 229–236.

[14] J. Podivínský, J. Lojda, O. Čekan, R. Pánek, and Z. Kotásek, "Reliability Analysis and Improvement of FPGA-based Robot Controller," in *Proceedings of the 2017 20th Euromicro Conference on Digital System Design*. IEEE Computer Society, 2017, pp. 337–344. [Online]. Available: https://www.fit.vut.cz/research/publication/11425