# The Fault-tolerant Single-FPGA Systems with a Self-repair Reconfiguration Controller

Richard Pánek, Jakub Lojda
Brno University of Technology, Faculty of Information Technology, Centre of Excellence IT4Innovations
Božetěchova 2, 612 00 Brno, Czech Republic
Email: {ipanek, ilojda}@fit.vut.cz

2023 IEEE 14th Latin America Symposium on Circuits and Systems (LASCAS) | 978-1-6654-5705-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/LASCAS56464.2023.10108372

***Abstract*—Fault tolerance in electronic systems is essential in harsh environments such as space. However, FPGAs that can be used to accelerate various computations are prone to configuration memory faults that determine their function. Repairing these faults is essential to increase system resilience. For this purpose, the partial dynamic reconfiguration controller is necessary. We force the controller to be on the same FPGA with a payload circuit to design a comprehensive system inside one FPGA. We create and thoroughly test a new reconfiguration controller to increase the system's resiliency with the ability to repair itself during its own operation. For this purpose, the FPGA controller is in coarse-grained triple modular redundancy to be able to recover despite the failure of any of its modules. The proposed controller has been tested to increase the resilience of circuits from a set of benchmark circuits. The entire system with the controller was evaluated on an actual FPGA, where faults were injected directly into the configuration memory of this FPGA. Reliability parameters are measured by a platform designed for this purpose, partly directly on the tested FPGA. As we can see from the results, the mean time to failure has been increased by up to 69% compared to a system equipped with only triple modular redundancy with a reasonable amount of hardware resources. The competitive solution brings only a 42% improvement in resilience with similar parameters.**

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

## I. Introduction

*Field Programmable Gate Arrays (FPGAs)* are helpful for computationally intensive applications. Their main contribution includes implementing efficient data processing directly in the hardware. At the same time, they provide a high degree of flexibility, where their function can be changed while the application is running. Due to their flexibility and high performance, FPGAs are often used for space applications [1]. However, using space applications also brings specific problems that must be addressed. Mostly, these are autonomous missions, so the resulting system has to deal with everything without outside interference. Commonly used SRAM-based FPGAs are prone to *Single Event Upsets*, causing a configuration memory bit to flip, which can lead to a change in the implemented function. *Fault Tolerance (FT)* [2] techniques must be used to ensure the expected behavior despite the occurrence of these faults. The primary way is to use redundancy; spatial is the most frequently used redundancy for FPGAs. The most well-known and widely used is *Triple Modular Redundancy (TMR)*. The basic principle is to mask faults by using multiple identical modules and selecting their majority. The problem may occur with faults that remain in the system. Accumulation of these faults over time leads to system failure. Timely fault mitigation will significantly extend the life of the system. The repair of the FPGA must be provided by the so-called *Reconfiguration Controller (RC)*, which uses *Partial Dynamic Reconfiguration* of the FPGA. Depending on the location of RC in the system, it can also be prone to faults.

The paper [3] deals with the design of a robust system based on soft-core processors running the required application. Thus, increasing resilience is limited to processor applications only. The system's repair is based on switching the context between two processors, where the one in the fault is corrected by reconfiguring using the special engine, which is in TMR. In paper [4], an RC based on an application-specific instruction set processor was proposed. It is equipped with TMR and should be able to repair itself, but it has not yet been tested on an actual FPGA with a system that would increase resilience. With a system on multiple FPGAs, the authors came up with article [5]. RCs have on each FPGA and are therefore able to repair one FPGA with the help of controllers on the other FPGAs. An RC based on a soft-core processor in TMR with its own reconfiguration was investigated in paper [6]. In addition, they use unused configuration memory space to store the correct configuration. However, their RC runs continuously because it checks the configuration memory and reconfigures it if a fault is detected. Some of these approaches do not allow working inside only one FPGA or, at the same time, do not provide self-repair capability. However, other approaches that can run on only one FPGA are based on a resource-demanding processor core. For this reason, we researched the principle of designing a reconfiguration controller that would be able to self-repair on the same FPGA. We extended our current controller [7] with the ability to repair itself during its own operation without interrupting its function or the function of the payload circuit. After that, we thoroughly tested the benefits of this new ability to increase the FT of the entire system on a single FPGA. The system's resilience is expressed by the *Mean Time To Failure (MTTF)* value from the runs performed and measured on real FPGAs.

The paper is further organized as follows. Section II is dedicated to improving the FT of the reconfiguration controller itself. Emphasis is placed on the ability to repair itself. Subsequently, such a controller is used to increase the resilience of systems, i.e., selected circuits from a set of benchmarks, which is summarized in Section III. Furthermore, this section deals with the results of experiments with RC and benchmark circuits. The whole paper is summarized in Section IV.

## II. Reconfiguration Controller Hardening

The resilience of the RC itself is critical to the system's overall resilience on the FPGA. A damaged RC with direct

access to the configuration memory can insert additional faults affecting the function and thus cause the entire system to fail. In addition, faults accumulate in the configuration memory that contains the RC, significantly reducing its MTTF [8].

Since the RC is located directly on the same FPGA as the circuit, it can be repaired similarly to this. The controller must perform as expected, even when faults affect it. We chose the approach where TMR masks faults until they are repaired by reconfiguration [9]. Unlike scrubbing, which detects faults directly in the configuration memory, our approach detects only errors at the application level. Faults in the configuration memory that do not affect the user logic, i.e., the required function performed by the FPGA, do not need to be repaired. Faults that lead to a circuit error can be detected by augmenting the *majority voter* with the ability to identify a module with a different output. Due to this information, the controller as a whole, therefore, repairs its one failed module. The majority voter ensures that the remaining two functional modules manage this reconfiguration. The commands from the failed module are masked in this way. A diagram of this approach is shown in Figure 1. The entire *Generic Partial Dynamic Reconfiguration Controller (GPDRC)* [7] selected by us to increase its resilience is there three times, i.e., in the *Coarse-grained TMR (CGTMR)*. The majority voters at the FPGA output and the ICAP input, i.e., at the interface between the triple and single systems, cannot be repaired using partial dynamic reconfiguration within a single FPGA. Therefore, these parts should be as small as possible to minimize the probability of being hit by a fault. Therefore, the fault detection of a module with a different output has been moved to the *Partial Reconfiguration Module (PRM)* of each controller module. *Fault Detection* will therefore be able to be repaired together with this PRM. The PRM_X indicates the parts of the FPGA that are reconfigured together. *PRM Error bus* carries requests for reconfiguration of payload circuits. This bus has a separate signal dedicated to each such request so that the reconfiguration of several modules can be requested simultaneously. These requirements are combined in Fault Detection to the requirements for reconfiguring the individual modules of the entire GPDRC in CGTMR. The flash memory, which is supposed to be FT, stores the *golden bitstreams*, i.e., the correct FPGA configuration.

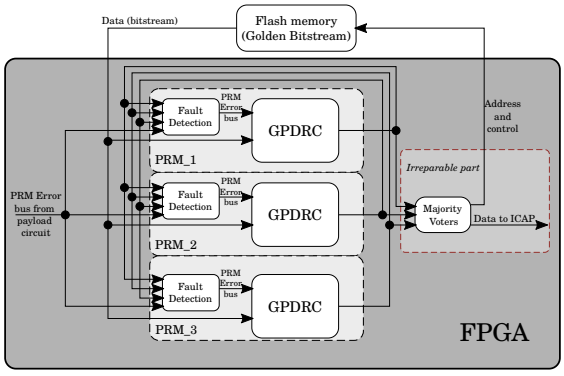



Figure 1: A GPDRC scheme in CGTMR with the ability to self-repair by reconfiguration.

The resulting self-repairing CGTMR GPDRC occupies more than three times the area on the FPGA compared to the original simple version. Related to this, it will also be three times more prone to faults, but this shortcoming should be offset by the ability to recover from these faults by self-reconfiguration. The Virtex-5 FPGA resource usage for each controller version is summarized in Table I. The values for one CGTMR GPDRC module are also given for a more straightforward comparison. The more considerable than expected triple increase in resources is significant for *LUT* and *CARRY*. The reason is the logic added to the individual modules, which is used to detect faults in both the controller and the payload circuit itself. This overhead depends on the number of bits of the circuit's outputs because these outputs need to be compared.

TABLE I: Average FPGA resource utilization by GPDRC versions according to statistics from Xilinx PlanAhead tool.

| GPDRC version | FLOP_LATCH | LUT | CARRY | BRAM |
|---|---|---|---|---|
| Simple | 288 | 327 | 223 | 1 |
| One PRM of CGTMR | 298 | 510 | 301 | 1 |
| Whole CGTMR | 894 | 1531 | 903 | 3 |

## III. Evaluation of the Improved Reconfiguration Controller

To evaluate the benefits of the reconfiguration controller, we need an application to extend its life by increasing its resilience. The use of benchmark circuits for FPGA seems to us to be conclusive. The prepared systems with the circuit and the controller must then be tested in a harsh environment. Therefore, we will use fault injection directly into the FPGA configuration memory to simulate such an environment accurately. Faults are injected by our fault injector tool [10], allowing us to target only the utilized part of LUTs. This approach significantly speeds up the FT evaluation, and simultaneously, the accuracy remains similar, which we verified in [11]. For managing experiments and subsequent data collection, we use the *Fault Tolerance ESTimation (FT-EST)* framework [12].

### A. Evaluation Platform and Circuits

The resilience evaluation platform, FT-EST, must be adapted to test individual benchmark circuits on the current FPGA. For each system, it is necessary to set the appropriate criticality of its future environment. In terms of testing, a value of $2 \times 10^{-5}\,\mathrm{inj/s/bit}$ was experimentally chosen, which is a reasonable compromise between the situation in the Earth's orbit and the time required to obtain a sufficient number of results to evaluate the experiments. The proposed approaches are verified on the Xilinx ML506 development board [13] equipped with FPGA from the Virtex-5 family *(XC5VSX50TFF1136)*. The 16-bit onboard flash memory P30 from Intel *(JS28F256P30T95)* is used for memory with golden bitstreams. We selected circuits from the ITC'99 [14] benchmark set for testing. Specifically, we chose three circuits so that it was possible to evaluate the effect of their size on the resulting resilience of the entire system. The largest possible circuit synthesizable on an FPGA was chosen, i.e., *b12* benchmark, a single-player game (guess a sequence). The second selected circuit is the smallest, i.e., *b01*, FSM that compares serial flows. *B05*, elaborating the contents of memory, is the last selected circuit because it has the average

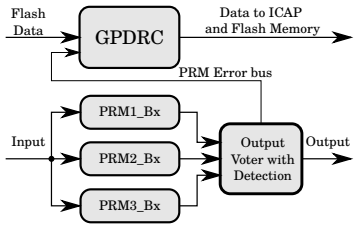
TABLE II: Use of FPGA resources by benchmark circuits.

| Benchmark | LUT | FLOP_LATCH | CARRY |
|---|---|---|---|
| b01 | 10 | 10 | - |
| b05 | 182 | 36 | 521 |
| b12 | 279 | 141 | - |

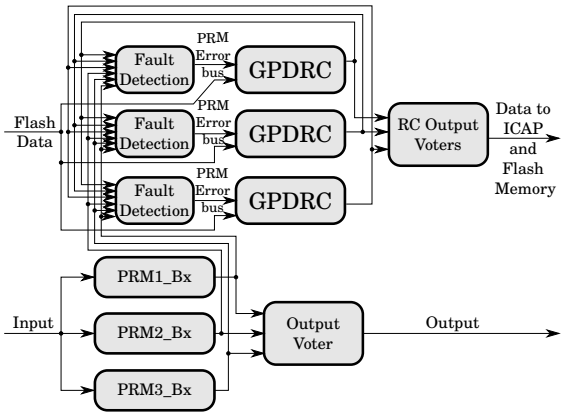
size among the previous ones. Table II summarizes the FPGA resource utilization by the selected circuit.

The prepared test systems with the RCs and the selected benchmark circuits have the following structure on the FPGA shown in Figure 2. Figure 2a shows a system with a simple controller, i.e., the benchmark circuit in TMR, the simple version of GPDRC, and the majority voter, which also detect a faulty module. In this case, each of the three PRMs with the entire benchmark circuit can be repaired separately. Figure 2b, on the other hand, shows a system with a self-repairing GPDRC in CGTMR. In addition to PRM with benchmark circuits, such a controller can also repair its modules. Thus, each of the three PRMs with the controller module and fault detection can be reconfigured separately. In this way, we prepared nine experimental systems to evaluate the benefits of our self-repairing RC. For each of the three selected benchmark circuits, which is always in TMR, three scenarios are prepared:

1) reference solution without RC,
2) with simple GPDRC,
3) with GPDRC in CGTMR (self-repair version).




(a) System with a simple controller.




(b) System with the controller in CGTMR with self-repair.

Figure 2: Schematic of the tested system on an FPGA.

## B. Experimental Results

The results of the performed experiments are summarized in Table III for easy comparison, all with the same intensity of faults. The total *size* column is expressed in terms of the number of configuration memory bits identified as LUT configuration bits used for the system's function. The *Mean Time Between Fault Injection (MTBFI)* column shows the average time between each fault injection (FI) into the configuration memory. This time results from the size of the circuit and the fault intensity. The resulting system resilience is expressed in MTTF from all runs. The FT improvement column compares the resilience of both controllers' versions, always with benchmarks only in TMR without RC and between the versions with controllers against each other (values in parentheses). The column Mean FI into Failure shows how many injected faults cause the failure of the complete system and, thus, how well injected faults are masked regardless of circuit size. The last two columns compare the FI to Failure and the sizes of the entire test circuits, i.e., the cost of increasing resilience. There is always a comparison of the increase to the non-RC version and the simple controller version (in parentheses). The numbers of individual separate experimental runs for each of the nine combinations are given by the convergence of the results and are in the order of thousands.

The resulting resilience of individual systems provided only by the simple version of GPDRC shows significant shortcomings of such a solution. The susceptibility of the controller to faults manifested itself, as it could not further repair the circuits. The resulting resilience of systems with self-repairing GPDRC shows its advantages. In all cases, the resulting resilience of the system is increased. We observe that this increase does not directly correspond to the size of the circuits. So, we assume that the circuit to which the controller is added affects the resilience. The structure of this circuit will probably be essential as to how it uses the FPGA resources and how its parts influence the outputs. We can see that such a controller needs a significant amount of FPGA resources. We observe a direct relationship between size and increased resilience when comparing the system with individual controller versions. The increase in the time that the system is in faultless operation decreases with increasing circuit size, apparently due to the unfavorable ratio of the controller to the circuit size. However, the controller's ability to repair itself positively affects the resulting system resilience. From the fault-to-failure point of view, we can see that the systems tolerated a significant amount of injected faults. The results show that systems with self-repairing GPDRC withstand more faults as their size increases.

The scrubber introduced by the authors of paper [6] is the closest to our self-repairing GPDRC; the difference is the type of FPGA used. We assume that technology significantly affects the occurrence of faults in FPGAs. However, the influence of faults on the manifestation of errors in user circuits is minimal, and we simulate precisely this part. Their scrubber is built on a PicoBlaze soft-core processor and is also in TMR. Our solution needs two to three times fewer resources and, at the same time, brings a slightly better improvement in FT for the selected circuit. The circuit with their scrubber was 42% better than just TMR. Our controller is 69% better for the same scenario, the b05 benchmark.

The fault detection approach can also have an impact on the results achieved. Our approach saves energy because faults in unused configuration memory do not lead to an error and therefore are not needlessly corrected. At the same time, however, latent faults that may be hidden in currently unused

TABLE III: Influence of various RC versions on the overall FT system, measured on benchmark systems on Xilinx Virtex-5 FPGA technology with $2 \times 10^{-5}\,\mathrm{inj/s/bit}$ fault intensity.

| Bench-mark | GPDRC version | Size [b] | MTBFI [ms] | MTTF [ms] | FT improvement over None (Simple) RC | Mean Num. of FI into Failure | Num. of FI into Failure Comparison | Size Comparison to None (Simple) RC |
|---|---|---|---|---|---|---|---|---|
| b01 | none | 2432 | 20 559 | 483 595 | - | 24 | - | - |
| | simple | 21 120 | 2367 | 371 477 | -23% | 157 | 6.7× | 8.7× |
| | self-repairing | 149 120 | 335 | 568 085 | 17% (53%) | 1694 | 72× (11×) | 61× (7.1×) |
| b05 | none | 43 584 | 1147 | 67 263 | - | 59 | - | - |
| | simple | 61 248 | 816 | 85 877 | 28% | 105 | 1.8× | 1.4× |
| | self-repairing | 220 224 | 227 | 113 749 | 69% (32%) | 501 | 8.5× (4.8×) | 5.1× (3.6×) |
| b12 | none | 55 552 | 900 | 174 721 | - | 194 | - | - |
| | simple | 69 184 | 723 | 168 200 | -4% | 233 | 1.2× | 1.2× |
| | self-repairing | 201 024 | 249 | 191 734 | 10% (14%) | 771 | 4× (3.3×) | 3.6× (2.9×) |

parts of the circuit are not repaired. Then, if these parts of the circuit become active, faults will result in errors in multiple parts of the circuit simultaneously. If these faults occur in multiple TMR modules simultaneously, masking will fail and lead to system failure. This shortcoming strongly depends on the circuit itself, i.e., how its parts control its outputs over time. Therefore, latent faults are eliminated if the entire system continuously influences the output. Conversely, the number of latent faults increases when parts of the circuit affect the output for only a short time. This problem is especially critical for the RC itself, which, most of the time, only waits for a reconfiguration request. However, even at this time, the RC is affected by faults, which do not manifest themselves in error as they do not affect the controller's output. So errors usually occur after running reconfiguration. If faults accumulate on over half of the TMR modules, the RC can no longer fix anything.

## IV. Conclusions

This paper focuses on increasing the fault tolerance of systems on a single FPGA using partial dynamic reconfiguration. The results show that the resilience of the reconfiguration controller is essential to increase the fault tolerance of the entire system. The overall fault tolerance was reduced when using a fault-prone RC in two of the three systems. On the other hand, a self-repairing RC has increased system resiliency in all cases. The MTTF of the individual systems tested was raised in the range of 10% to 69% compared to the system consisting only of the TMR version of the payload circuit. For comparison, on a similar system, the authors of paper [6] achieved a resilience improvement of 42% with their scrubber. From another point of view, reasonably large circuits require 4 to 8.5 times more configuration memory faults to fail a system with a self-repairing controller. Compared to systems equipped with only the simple version of the controller, i.e., prone to faults, the MTTF was increased by 14% to 53% for individually selected benchmark circuits. However, using a self-repairing controller significantly increases the demands on FPGA resources, but it is practically a constant overhead. As the payload circuit increases, this overhead will decrease, and the benefit of using the controller will be even more effective. In future work, we will address a more effective way of fault detection. Distinguishing between latent faults and those that never lead to an error is advantageous. Such approaches should save resources and extend the lifetime of the controller and, therefore, the reliability of the entire system.

## Acknowledgements

This work was supported by the Brno University of Technology project FIT-S-20-6309.

## References

[1] S. Habinc, "Suitability of reprogrammable FPGAs in space applications," http://microelectronics.esa.int/techno/fpga_002_01-0-4.pdf, September 2002, Accessed: 2022-01-12.

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

[3] 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.

[4] L. Gong, A. Kroh, D. Agiakatsikas, N. T. H. Nguyen, E. Cetin, and O. Diessel, "Reliable SEU monitoring and recovery using a programmable configuration controller," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–6.

[5] C. Bolchini, L. Fossati, D. M. Codinachs, A. Miele, and C. Sandionigi, "A Reliable Reconfiguration Controller for Fault-Tolerant Embedded Systems on Multi-FPGA Platforms," in *2010 IEEE 25th International Symposium on Defect and Fault Tolerance in VLSI Systems*, Oct 2010, pp. 191–199.

[6] R. Giordano, D. Barbieri, S. Perrella, R. Catalano, and G. Milluzzo, "Configuration Self-Repair in Xilinx FPGAs," *IEEE Transactions on Nuclear Science*, vol. 65, no. 10, pp. 2691–2698, Oct 2018.

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

[8] J. Lojda, R. Panek, L. Sekanina, and Z. Kotasek, "Automated Design and Usage of the Fault-Tolerant Partial Dynamic Reconfiguration Controller for FPGAs," in *article submitted to the Microelectronics Reliability journal*.

[9] 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.

[10] 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.

[11] R. Panek, J. Lojda, J. Podivinsky, and Z. Kotasek, "Reliability Analysis of Reconfiguration Controller for FPGA–Based Fault Tolerant Systems: Case Study," in *2020 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, 2020, pp. 1–4.

[12] J. Lojda, J. Podivinsky, O. Cekan, R. Panek, and Z. Kotasek, "FT-EST Framework: Reliability Estimation for the Purposes of Fault-Tolerant System Design Automation," in *2018 21st Euromicro Conference on Digital System Design (DSD)*, 2018, pp. 244–251.

[13] Xilinx Inc., "Ml506 Evaluation Platform User Guide," *UG347 (v3. 1.2)*, 2011.

[14] F. Corno, M. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *Design and Test of Computers, IEEE*, vol. 17, no. 3, pp. 44–53, Jul 2000.