BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

USE OF VERIFICATION FOR TESTING FAULT-TOLERANCE IN FPGA-BASED SYSTEMS

VYUŽITÍ VERIFIKACE PRO OVĚŘOVÁNÍ ODOLNOSTI PROTI PORUCHÁM U SYSTÉMŮ ZALOŽENÝCH NA FPGA

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR — **Ing. JAKUB PODIVÍNSKÝ**
AUTOR PRÁCE

SUPERVISOR — **doc. Ing. ZDENĚK KOTÁSEK, CSc.**
ŠKOLITEL

BRNO 2021

Abstract

Fault tolerance is one of the most commonly used techniques to eliminate the effect of faults on digital systems and increase their reliability. This work presents a platform for testing such fault tolerance techniques targeted to FPGA-based systems. The platform uses the principles of functional verification, while the experimental electronic controller is moved to the FPGA, which allows the use of fault injection directly into the FPGA. The platform makes it possible to use the electro-mechanical application as an experimental system and allows to monitor the effect of faults on both the electronic controller and the behavior of controlled mechanical part. This work presents experiments with two experimental systems – robot for finding a path through a maze and an electronic lock. The platform is designed to allow the use of any experimental system with an electronic control unit implemented in the FPGA.

Abstrakt

Odolnost proti poruchám je jedna z nejčastěji využívaných technik pro eliminaci vlivu poruch na číslicové systémy a zvýšení jejich spolehlivosti. Tato práce popisuje platformu pro testování technik pro zajištění odolnosti proti poruchám v systémech založených na FPGA. Platforma využívá principů funkční verifikace, přičemž experimentální elektronická řídicí jednotka je přesunuta na FPGA, což umožňuje využít injekci poruch přímo do FPGA. Platforma umožňuje využít elektro-mechanickou aplikaci jako experimentální systém a sledovat vliv poruch jak na elektronickou řídicí jednotku, tak na chování řízené mechanické části. V práci jsou představeny experimenty se dvěma experimentálními systémy – robot pro hledání cesty v bludišti a elektronický zámek. Platforma je navržena tak, aby umožnila využití libovolného experimentálního systému s elektronickou řídicí jednotkou implementovanou v FPGA.

Keywords

FPGA, fault tolerance, reliability, fault injection, functional verification, electro-mechanical system.

Klíčová slova

FPGA, odolnost proti poruchám, spolehlivost, injekce poruch, funkční verifikace, elektromechnický systém.

Reference

PODIVÍNSKÝ, Jakub. *Use of verification for testing fault-tolerance in FPGA-based systems*. Brno, 2021. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Zdeněk Kotásek, CSc.

Use of verification for testing fault-tolerance in FPGA-based systems

Declaration

I declare that this PhD thesis was prepared as original author's work under the supervision of doc. Ing. Zdeněk Kotásek, CSc. The supplementary information was provided by Ing. Marcela Zachariášová, Ph.D. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Jakub Podivínský
March 3, 2021

Acknowledgements

I would like to thank my research supervisor Zdeněk Kotásek for professional guidance, his help and expert advice provided during my research. Further, I would like to thank Marcela Zachariášová for professional help, suggestions and ideas, especially at the beginning of my research.

Contents

Chapter 1

Introduction

Digital systems play an important role in our everyday lives, and we meet them more and more frequently in various applications. Digital systems are widely used in industrial production, they are also used as control systems in vehicles, medicine, telecommunications, and other sectors. The current trend is to move more responsibility to digital control systems, which usually leads to a reduction of the weight of the mechanical part and thus to a reduction of operating costs, for example in aviation [18, 5] or automotive [43]. As a result, the complexity of digital systems is rising, which leads to an increase of their integration. Unfortunately, this phenomenon results in an increased susceptibility of such systems to faults. Both to faults caused during the design and implementation of the system, as well as to faults occurred during the system operation. The occurrence of such faults can have far-reaching consequences, not only in the huge financial losses but human lifes can also be endangered (e.g. a failure in the aircraft control system).

This work deals with faults in systems based on *Field Programmable Gate Arrays* (FPGAs). FPGAs are increasingly widespread modern circuits that provide speeds close to the *Application-Specific Integrated Circuits* (ASICs), but at the same time allow easy programmability (a change of the performed function). Thanks to the possibility of partial dynamic reconfiguration, the performed function, even parts of the circuit, can be changed while the system is running without the need to stop it. The FPGA configuration is stored in the configuration memory, currently FPGAs with SRAM configuration memory are mainly used. Unfortunately, the disadvantage is the increased susceptibility to faults caused by charged particles, which can cause a change in the configuration memory. This can lead to a change in the behavior of the whole system. Therefore, it is very important to find ways how to ensure that the occurrence of these faults does not endanger the operation of systems where FPGAs play a significant role.

Two main approaches to increase reliability are currently used: *fault avoidance* and *fault tolerance*. This work focuses on the second approach, fault tolerance. Many fault-tolerant methodologies have been developed, among others, to FPGA-based systems, and new ones are under investigation. This work aims to create a platform for monitoring the impact of faults in FPGA-based systems. The platform combines functional verification approach, such as fault impact monitoring tool, and artificial fault injection. An integral part of the platform is also the developed process, divided into a set of procedures, for monitoring the impact of faults. The proposed platform can be used for automated evaluation of fault-tolerant techniques. The platform proposes the possibility to apply these techniques to a more complex system, nearly real systems. For this reason, an electromechanical application is used as an experimental system, which makes it possible to monitor the

impact of faults not only on the output of the digital part but also on the behavior of the controlled mechanics.

1.1 Goals of the Thesis

The study of the state of the art in the field of fault-tolerant systems based on FPGAs show the need to answer several questions:

- *What will be the results of fault tolerance techniques on real systems?*
- *Is it possible to rely on the fact that not all faults in the electronic part of the system affect the behavior of the controlled mechanical application?*
- *Can functional verification be used to evaluate the effect of faults on fault-tolerant systems?*

These questions form the basic directions of the objectives of this thesis. I define two main goals, which are supplemented by several sub-objectives:

Goal 1. Design and create an evaluation platform targeted to FPGA technology which allows to test fault tolerant techniques and allows to monitor the impact of faults not only on the output of the electronic part, but also on controlled mechanical application.

1.1. Functional verification will serve as the basic technique that will be used to verify the correctness of the outputs of the tested system affected by fault injection.

1.2. An integral part of the whole platform will be a fault injection tool, which was previously developed by the team of doc. Kotásek. Thus, this thesis builds on the earlier work of this team.

1.3. The core of this platform forms an experimental electromechanical application, which means a system consisting of an electronic controller implemented on an FPGA and mechanical applications controlled by this controller. Such a system makes it possible to monitor the effect of faults not only on the output of the electronic controller, but also on the controlled mechanical application.

Goal 2. The developed test platform is followed by the design of a process for monitoring the impact of faults on the electro-mechanical application using the proposed evaluation platform.

2.1. The proposed process for monitoring the impact of faults will reflect the experiences obtained during experimental work with the designed platform and the created experimental system.

2.2. The proposed process also includes a description of the necessary activities related to scalability, it means the use of another experimental system.

2.3. The proposed process allows scalability, it is generalized in such way that it can be used for evaluation with the use of another experimental electromechanical system. It is also demonstrated on a second experimental electromechanical application.

1.2 The Thesis Outline

This thesis is composed as a collection of published papers. The main ideas of this thesis and the relevant research are, therefore, available in eight key articles published at international conferences or in a journal with an impact factor. These articles are attached in their original form as published. Chapter 2 presents the necessary basic theoretical information. Chapter 3 summarizes the research results achieved in individual articles. Last Chapter 4 concludes the thesis, summarizes the benefits of the work and provides possible direction of further research activities.

Chapter 2

Background

This chapter summarizes the knowledge in the field of this thesis topic, thus in the field of digital systems based on FPGA dependability.

2.1 Digital Systems Dependability

Dependability is not quantifiable in itself and the ČSN IEC 50 (191) [17] standard defines it as follows:

Definice 1 *Dependability is a general property of an object consisting of the ability to perform the required functions and maintain the values of specified operating indicators within the given limits and in time according to specified technical conditions.*

The definition mentions an object, which is a general term for which we can substitute the whole digital system, its part, or a separate component. Reliability cannot be expressed as a single property that includes several different aspects, so the reliability attributes [37] are used to express it, which form the evaluation of partial properties and together they express *dependability*. These are as follows:

- *Reliability* of a system is the probability that the system operates without a failure in the defined interval [*t1, t2*] under given operational conditions.
- *Availabillity* of a system at time t is the probability that a system is in operational state (not in failure) at a given time.
- *Safety* is the probability that the system works correctly or does not work at all in case that the function of another system may be compromised or personal safety may be endangered.
- *Maintainability* is the effort required to maintenance (repair) the system after its failure.

Obviously, the system may not always provide the function for which it is deliberate. According to [7], the causes and consequences of deviations from the required function are called *dependability factors*. We are talking about these three terms:

1. A *fault* is a phenomenon that leads to a disruption of an object's ability to perform a desired function. There is usually an external cause of the fault. We can mention two basic sources of faults in digital systems:

- *Bugs caused by incorrect design* usually lead to a mismatch between the desired and the real behavior of the system. They are often caused by a poor specification of the required behavior, or a misunderstanding of the system specification by the designer.
- *Physical defects in hardware* occur during the manufacture or during the use of the system. Faults caused during the use of the system can be caused by various external influences (e.g. electromagnetic radiation) and when designing a dependable system, we have to consider the possibility of these faults.

2. *Error* in a system is a deviation from the required operation of system which causes the difference between actual processed data and expected correct data. It is usually the result of a fault (see Figure 2.1), but not every fault manifests itself as an error.

3. *Failure* of the system means that it is not producing correct outputs and thus the system is working with the behavior which differs from the required one. A failure is caused by an error.

Figure 2.1: From fault to failure.

Physical faults in hardware that occur during the use of the system can be further classified into several categories, mainly according to the duration time [7]:

1. *Permanent faults* are faults that can occur both during production process and during the life of the system. These faults are permanent and can usually be repaired only by replacing the failed component. As an example can serve fault type permanent 1/permanent 0, short circuit, delay, etc.

2. *Transient faults* appear during the life of the system, arise unexpectedly and may disappear in unpredictable moments. They are most often caused by external influences, such as cosmic radiation, electromagnetic interference or various temperature fluctuations. Sometimes they can also cause permanent damage. Most often we encounter the following cases:.

   - *Single-Event Upset* (SEU) represents a random change of values in memory elements and in sequential circuits.
   - *Single-Event Transient* (SET) manifests as unwanted changes in signal values in the circuit or oscillations in the combinational logic.

3. *Intermittent faults* are usually caused by the degradation of hardware parameters during ages. They appear after a long time and over longer time can convert into permanent faults.

Reliability improvements can be achieved by improving all partial reliability indicators, which is usually impossible or very limited to implement. The biggest influence on the

increase of reliability has the decrease of the intensity of failures, which leads to the improvement of all important reliability indicators. Methods for reducing the intensity of failures are very complex and, above all, too expensive. This approach is often referred as *Fault Avoidance*. A different approach to ensure reliability is *Fault Tolerance*. This approach considers the possibility of faults and tries to prevent affecting the behavior of system caused by fault impact.

2.1.1 Fault Tolerant Systems

A fault-tolerant system is such system that performs its function correctly in case of fault occurrence in its hardware or error in software. The term *performing its function correctly* can be understood in various ways, usually the function is considered correct if the following conditions are met [33]:

1. data processing was not stopped or changed due to a fault,
2. the result is correct,
3. the result was obtained in the prescribed time.

There are two basic types of techniques to provide fault tolerance. The first type is *passive redundancy* [27], which uses masking of faults that occur in the system. In this case, no fault detection and correction techniques are usually used. The most common example is the technique called *Triple Modular Redundancy* (TMR), which consists of a triplication of a part of the system and the output of this part is the result with the majority.

The second approach is *active redundancy* [27], which also uses the ability to detect and repair faults. In this approach, system testing is performed at runtime, informs about the possible occurrence of a fault in the system. Then the fault is localized and isolated. Such isolated fault cannot affect the function of other parts of the system. Finally, the detected fault is repaired and the corrupted part is able to perform its original function correctly again.

The classification of redundancy into several categories is most often given:

1. *Spatial redundancy* [15, 55], sometimes also called hardware redundancy, is the most common way to ensure system fault tolerance. The individual components are duplicated several times and supplemented by a special circuit that monitors whether all units provide the same output. The above mentioned TMR technique can serve as an example.
2. *Time redundancy* [35] is based on repeating the calculation with the same component at different time intervals and comparing the stored results of each calculations. An example of time redundancy is shown in Figure 2.2. The outputs of the combinational logic, which are calculated at different time intervals, are stored in registers for later comparison.
3. *Information redundancy* [61] is used to protect data from fault, the most common form of information redundancy is coding. Various codes are used, which are able to detect and possibly correct an error in the data.
4. *Software Redundancy* [29, 60] is a technique that provides fault tolerance at the level of the software source code. We can use this technique, for example, to ensure fault tolerance in processors.

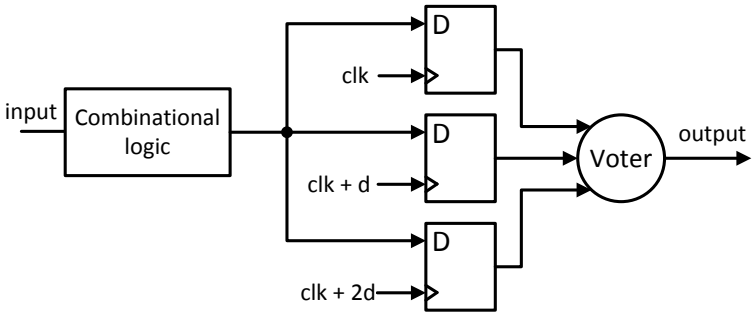


Figure 2.2: Basic time redundancy scheme for ensuring fault tolerance in combinational logic.

2.2 Digital Systems Verification

Verification is the process that verifies whether the function of a digital system meets the required specification or not. This is a very important stage in the design and development of digital systems, as it helps us eliminate *bugs caused by incorrect design*. It is important to detect non-compliance with the specification as soon as possible because the later we detect it, the more costly it will be to eliminate it. With the increasing complexity of modern digital systems, verification is becoming increasingly important, but also very time-consuming.

Ideally, the verification output is yes/no, it means information whether it corresponds/does not correspond to the specification. We distinguish two basic types of verification applicable for digital systems: 1) formal verification and 2) functional verification.

2.2.1 Formal Verification

Formal verification [39] uses an abstract mathematical model of the verified system and formally proves its correctness or inaccuracy. This can be considered as a significant advantage over functional verification, which is not able to prove the correctness of the verified system. However, it is very computationally intensive, for example, when searching for state space, there is an extreme increase of checked states, which is referred as the problem called *State Space Explosion*. Similarly, the construction of an abstract mathematical model that is equivalent to a validated system may not always be realistic.

Formal verification uses various methods, such as *Theorem Prooving* [31], *Static Analysis* [24], or *Model Checking* [8].

2.2.2 Functional Verification

Functional verification [51] checks whether the system is in compliance with the specification by monitoring its inputs and outputs in a simulation environment (e.g. *ModelSim* [49] or *Questa Advanced Simulator* [50]). This simulation approach means that functional verification does not provide any proof of system correctness. In general, the use of functional verification is easier for digital system designers, it is an extended sophisticated version of commonly used test environments (*testbench*). At the same time, there is no need to create complex abstract mathematical models. Functional verification is one of the most commonly used approach in the industry [73]. The basic principle of functional verification is shown in the Figure 2.3. The verified system is marked as DUV (*Device Under Verifica-*

tion), the outputs of this system are compared with the reference model (*Reference Model*). The reference model is created by a different developer than the verified circuit, it leads to the fact that more people interpret the required specification. If a mismatch is found between the outputs of the DUV and the reference model, it means that the specification was misinterpreted by one of the developers. The output of the functional verification process is also a report of the coverage of key functions of the verified system (*Coverage Report*).

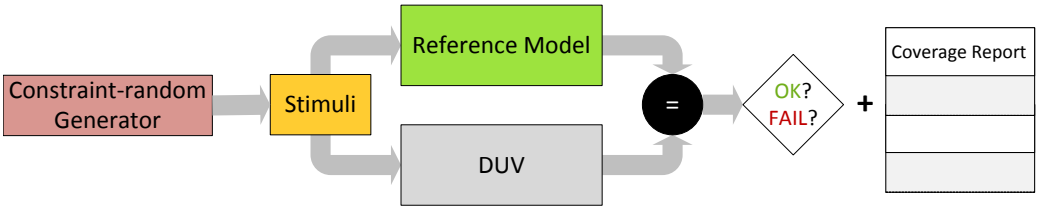


Figure 2.3: The basic principle of functional verification.

To facilitate the creation of verification environments, there is a standardized language SystemVerilog [1, 66, 9], which is an object-oriented language from the family of HDVL (*Hardware Description and Verification Language*) languages, which are languages for hardware description and verification [69]. There are several standardized methodologies for functional verification, one of which is the UVM (*Universal Verification Methodology*) [62], which provides freely available libraries with basic components for the verification environment. The Figure 2.4 shows an example of a verification environment according to the UVM methodology. The basic components are obvious here:

- *uvm_sequence* provides verification stimuli in the form of a sequence of transactions,
- *uvm_sequencer* transforms the sequence of transactions into individual stimuli for the verified circuit,
- *DUV* represents the verified circuit,
- *uvm_driver* converts the received stimuli into signals for the input pins of the verified circuit,
- *uvm_monitor* monitors the output pins of the verified circuit and converts the signals into transactions for further processing,
- *uvm_scoreboard* compares the obtained data with the expected values, generates reports and statistics,
- *Golden Model* produces reference output values of the verified circuit.

SystemVerilog language allows cooperation with other programming languages through an interface called *Direct Programming Interface* (DPI) [25]. It is thus possible to call functions implemented in another language. An example of use is the implementation of a reference model in another language and calling its functions using DPI.

The output report with the coverage of key functions of the verified system (*Coverage Report*) tells us how thoroughly the individual specified functions were verified by used input stimuli. We try to maximize the coverage to ensure that all key features are verified. We distinguish these basic metrics [59]:

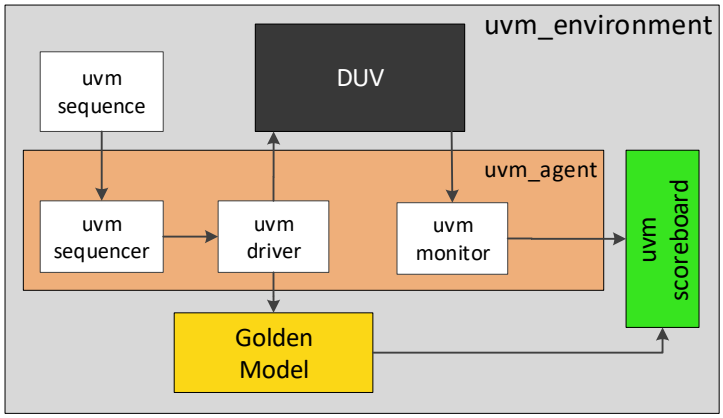


Figure 2.4: The example of environment for functional verification according to UVM.

- *Functional coverage* must be specified manually by the programmer. It focuses mainly on semantics and represents the coverage of system functions and behavior.
- *Structural coverage* is generated automatically by a simulation tool. It focuses on structures in source code. Examples of typical structural metrics are:
  - *Code coverage* measures what constructs and lines of source code were executed during the verification run without any knowledge of its function. It includes more detailed metrics such as coverage of branches, conditions, and expressions.
  - *Finite state machine coverage* focuses on visited states and transitions in finite state machines.
  - *Toggle coverage* represents how often individual signals and registers change their logical values.

In order to obtain high coverage, it is necessary to ensure a sufficient number of suitably selected verification stimuli. This is the task for the generator that generates these inputs based on the specified constraint conditions (*Constraint-random Generator*). In case the coverage is not sufficient, the verification engineer has to adjust the limiting conditions, so the generator generates other suitable stimuli [4].

2.3 Field Programmable Gate Arrays

Field Programmable Gate Arrays (FPGAs) [76] are circuits that can be programmed to gain the required functionality. FPGAs can be programmed before use, but can also be *reconfigured*, which means the programming of the circuit while the application is running. Useful is also *Partial Dynamic Reconfiguration* (PDR) [74], which allows programming only a part of the circuit while the rest of the circuit is running. It is the programmability that differs FPGAs from the *Application Specific Integrated Circuits* (ASICs) [23], which acquire their functionality already during the production process.

The FPGAs are becoming increasingly popular and are used in various applications, primarily due to the programmability mentioned above, quite easy design, flexibility, decreasing power consumption, and also due to falling prices. They are used mainly in applications where small series need to be produced, and the ASIC design is not beneficial, and the conventional microprocessor solution is unsuitable. Advantageously, the FPGA can

be used to prototype more complex customer circuits that allow system testing during the design process. Programmability can also be used to change the behavior of the circuit at the customer, which allows you to correct design bugs or add new features to an already used device.

2.3.1 Structure of the FPGA

The FPGA structure is shown in Figure 2.5.a. The circuit consists of a matrix of *Configurable Logic Block* (CLB), which are connected through a programmable interconnection network. In addition to CLBs, modern FPGAs include a number of other features, such as *Block RAM* (BRAM), fast multipliers, processor cores, and blocks of specialized *Digital Signal Processor* (DSP). *Input/Output Blocks* (IOBs) are used for communication with the surrounding environment.

Configurable logic block (CLB) allows the implementation of any logical function, its structure is shown in Figure 2.5.b. It is the main component for implementing sequential and combinational logic circuits. Each CLB is composed of an SRAM memory cell that implements logical functions using a *Look-Up Table* (LUT). CLBs also contain the D flip-flops and multiplexers.

The configuration of the individual blocks and the interconnection network of the circuit is stored in the SRAM memory in the form of a configuration *bitstream* (bit sequence containing the configuration of the FPGA circuit), which allows easy programming and reconfiguration. On the other hand, it is necessary to configure the circuit each time it starts, because the stored configuration is volatile, which is based on the SRAM principle. Currently, the most commonly used are such FPGA circuits with a configuration stored in SRAM.

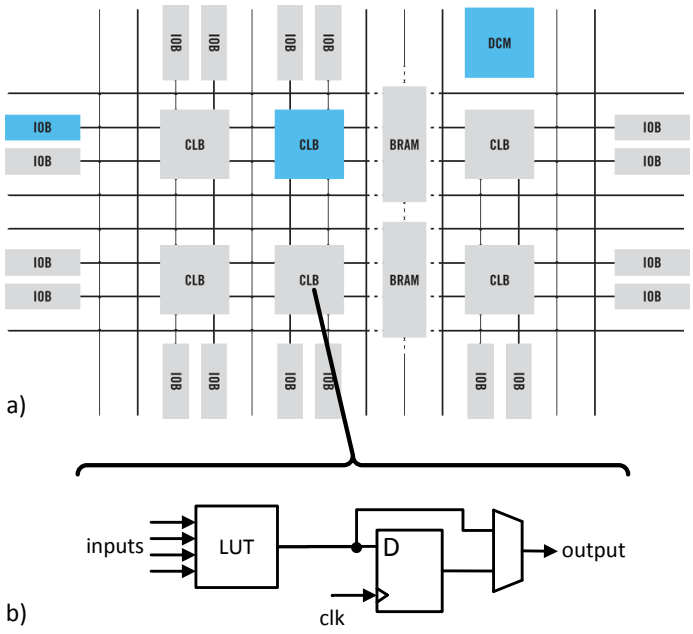


Figure 2.5: Structure of a)*Field Programmable Gate Arrays* (FPGA) and b) *Configurable Logic Block* (CLB).

2.3.2 Faults Occurrence in FPGA

From a general perspective, all the mentioned types of faults can occur in the FPGAs, but the most probable are the transient faults. These faults are typically caused by high-energy particles [14], which occur not only in space but also in the upper parts of the Earth's atmosphere. Due to the influence of solar radiation, various types of particles are formed, we distinguish between charged particles, which can be protons, electrons or ions, and particles created by electromagnetic radiation, which we call photons. The concentration of these particles increases with rising altitude. When designing systems where high reliability is required, we must also take them into account in such systems operating at lower altitudes.

The hit of the high energy particle on the FPGA can cause unwanted jitter on the transmitted signal, an effect called *Single Event Transient* (SET) [19]. When a charged particle hits a memory cell, the voltage of the memory cell may decrease, which leads to a change in the stored value. This effect is called *Single Even Upset* (SEU) [13, 20] and is the most common fault affecting FPGAs. All configuration information about the logic implemented in the FPGA is stored in the SRAM memory. The occurrence of an SEU in this memory can have various impact on the circuit operation. To repair the SEU in the FPGA, the reconfiguration can be advantageously used which ensures the correct functionality of the circuit by restoring the original configuration. In the case of certain conditions, the SEU may cause a permanent fault in the FPGA's internal structure, such as a short circuit in the LUT outputs.

The Figure 2.6 shows possible impact of SEU in FPGA [34]. The occurrence of the fault in the LUT usually leads to a change of the logic function that the LUT implements. SEU can also occur in a D flip-flop in CLB which leads to a fault in the sequence logic. A fault in the part of the configuration memory that stores information about the configuration of the interconnection network can lead to incorrect connection of individual blocks. BRAM block memory is also sensitive to faults that can cause a change in stored data. The sensitivity of the FPGAs to these failures and the possibility of reconfiguration are the main reasons why a number of fault tolerant methodologies focusing on FPGA-based systems have been developed [12, 67].

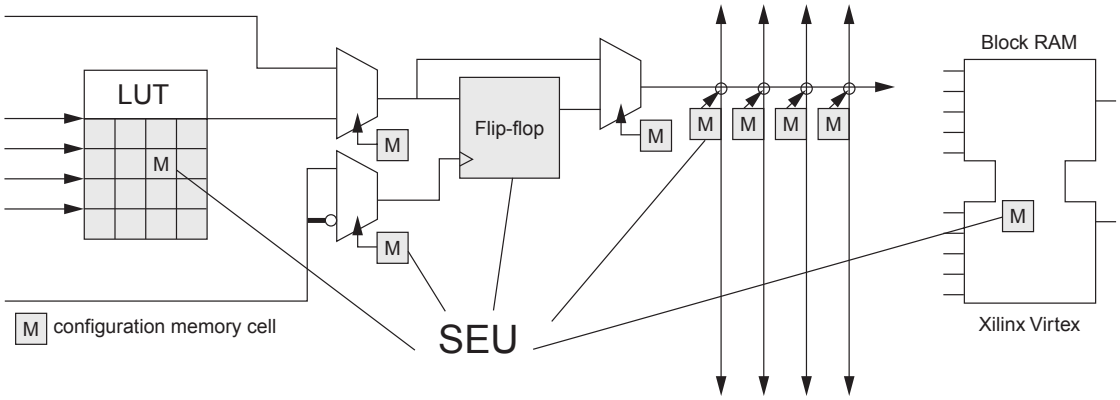


Figure 2.6: Example of SEU impact on FPGAs. [34].

2.3.3 Partial Dynamic Reconfiguration

An important feature of modern FPGAs, which can also be used to correct a detected fault, is the *Partial Dynamic Reconfiguration* (PDR) [74, 30]. Thanks to the partial dynamic re-

configuration, we can modify the specified part of the FPGA configuration memory. At the same time, the rest of the circuit can perform the required functions without interruption. Previously prepared parts of the configuration memory can be stored in the external memory, from where they are read when their use is requested. An example is a simple system where the FPGA part works as a multiplier after initialization, but thanks to the reconfiguration, we can change the function of this part to an adder (without limiting the operation of other FPGA components).

Thanks to the partial dynamic reconfiguration, the part of the FPGA configuration memory affected by the fault can be repaired, and the function of the whole system can be restored into a fault-free state. The repair of a functional unit affected by a fault in the TMR architecture is shown in Figure 2.7. If one of the three FUs is affected by a fault (FU3), the TMR architecture provides the correct result at the system output, and FU3 can be repaired using the PDR without interrupting the operation of the remaining functional units. In the event of a fault occur in the second component (FU1), the TMR architecture would no longer be able to provide the correct output, which leads to failure. Thanks to the reconfiguration, the failure will not occur, because it is able to repair individual faults.

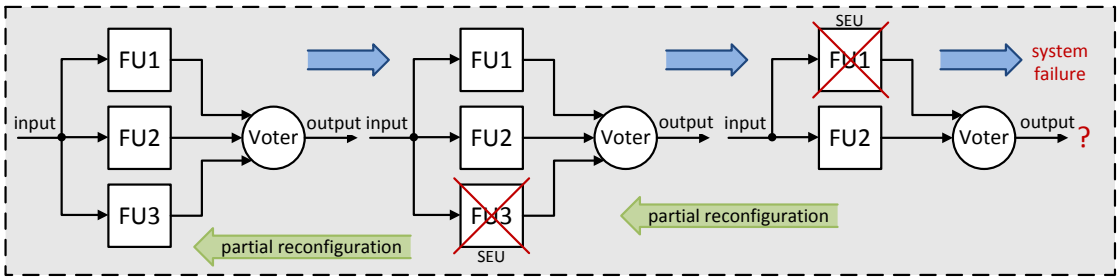


Figure 2.7: The use of PDR to repair the failed functional unit.

2.4 Fault Tolerance Testing

During testing quality of fault tolerance, it is not feasible to wait for the natural occurrence of faults (SEU) in the FPGA configuration memory. The main reason is a long time between natural fault occurrence. The parameters MTTF (*Mean Time To Failure*) [44] and MTBF(*Mean Time Between Failures*) [38] are very limiting here, which can be even several years. These parameters indicate the average operating time of the system before the occurrence of the first fault and the average time between the occurrence of two faults. For this reason, it is necessary to simulate these faults in some way during testing fault tolerance. This chapter introduces fault simulation and fault tolerance testing approaches for FPGA-based systems.

2.4.1 Overview of Fault Tolerance Testing Approaches

Fault tolerance testing can be divided into two categories based on the used fault injection technique. There are two basic approaches [6]:

- fault injection at the level of circuit simulation,
- fault injection into real hardware.

Simulation-based Fault Injection

The use of functional verification for the fault tolerance evaluation with the use of fault injection at the level of a simulation model was presented by the authors of [6]. In this approach, functional verification is extended to include fault injection components at the circuit simulation level. The basic scheme is shown in the Figure 2.8. Two instances of the same implementation of the tested circuit are used, one serving as the tested circuit itself (*Faulty DUT*) and the other as a reference circuit (*Golden DUT*). It includes a number of other components that ensure the injection of faults into defined sensitive areas, monitoring the behavior of the tested circuit, comparison with the outputs of the reference circuit, etc.

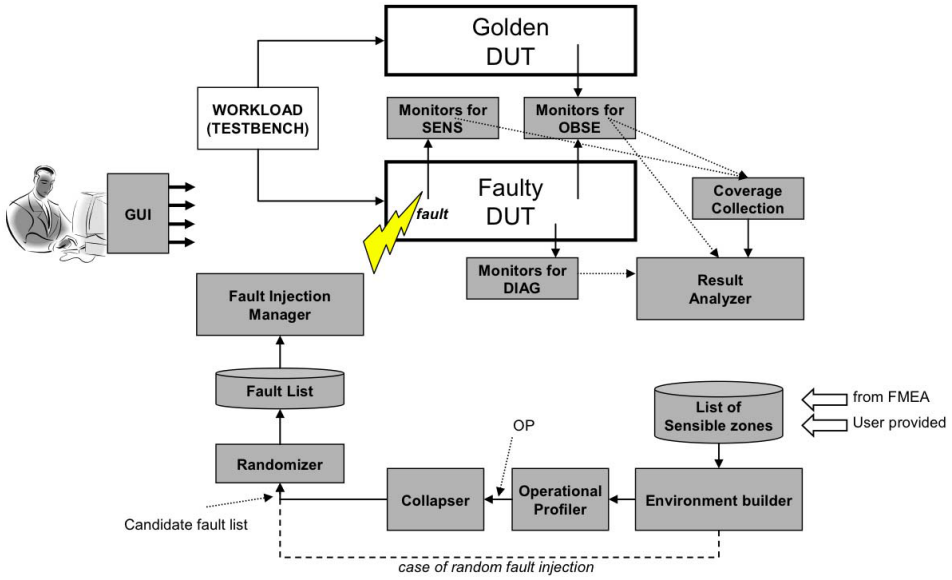


Figure 2.8: The use of functional verification for testing fault tolerance at the level of circuit simulation [6].

A simulation method for emulating the impact of SEU faults in the FPGA configuration memory is presented in [10]. The authors combine simulation and topological analysis of a system that is implemented in the FPGA. The proposed analytical algorithm is able to determine the effects of SEU failures accurately. This simulator was created with the aim to provide designers cheaper alternatives to testing the impact of faults on real, relatively expensive FPGA circuits.

A simulation-based fault injection approach is also presented in [53]. It presents an automated technique based on a simulator that is able to work at different levels of abstraction (e.g. RTL or netlist). The authors also offer a developed environment for monitoring the impact of faults on a finite state machine implemented in an FPGA.

The disadvantage of the simulation-based fault injection approach is the fact that fault tolerance is not evaluated on the real target platform. On the other hand, this type of evaluation can be performed without the existence of real hardware, i.e. in the early stages of development.

Fault Injection into Real Hardware

An FPGA-based fault injection tool, which is presented in [63], supports several synthesizable fault models of digital systems, and is implemented using VHDL. The authors present a real-time fault injection tool for fault injection into real hardware with good controllability and observability. However, the fault injection requires the addition of some extra gates and wires to the original design and thus modifying the original VHDL. There are several types of faults that can be generated. For example, the model of injecting SEU can be seen in Figure 2.9. There are additional signals *Bit* and *FIS* which are connected to the Fault injection component (implemented on the same FPGA). A weak point of this approach is the difference between the Device Under Test (DUT) and the device which will be manufactured.

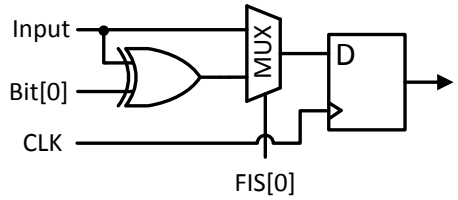


Figure 2.9: The synthesizable SEU model [63].

In [3, 2], techniques based on the fault injection into a real FPGA board without changing the original design were presented. These techniques are based on partial dynamic reconfiguration which allows to read the configuration bitstream, inverse bits, and write the affected bitstream back to the FPGA. The prototype of the evaluation board for the fault injection purposes was presented in [3]. There are two FPGAs, the first one is used as the DUT and the second one is used as the fault injection controller. The block diagram is shown in Figure 2.10. In [2] the authors present platform called FLIPPER. This fault injection platform is composed of two boards with FPGAs – the main board and the DUT board. The fault injection is controlled by the main board driven by the software application running on a PC. It is able to use various types of FPGAs as the DUT board, but only if there are enough input/output pins on the main board. The disadvantage is the need to purchase special hardware for fault injection, while the developer usually has commonly produced development boards from FPGA manufacturers.

A similar approach presents authors of [46] which focus on the speed of the fault impact evaluation, where the fault injection is fully controlled by a part of the design on the FPGA. Communication with a PC is used only for the initial configuration of the fault injection process. Thanks to a reduction in communication, the fault tolerance testing process is significantly accelerated.

The FPGA-based fault injection method is also presented by the authors of [65], which is demonstrated in [40]. This technique is implemented in Java and is based on the RapidSmith library [41]. Thanks to a tool for finding a relationship between the individual parts of the circuit description and the corresponding part of the FPGA, faults are injected into a specified place on the FPGA. The authors provide a command line interpreter that can operate in batch or interactive mode, and a graphical interface to specify the locations of permanent faults. The authors mention that the method is unique because it is not necessary to perform a new circuit synthesis before injecting the fault into a specified location.

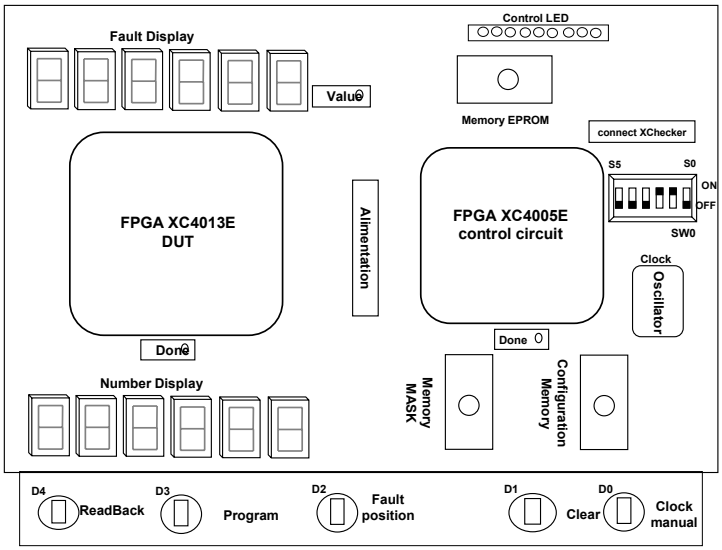


Figure 2.10: The block diagram of a FLIPPER platform board which allows fault injection into an FPGA composed of two FPGAs - DUT and fault injection controller [3].

Multi-platform fault injection is presented in [45]. This technique is based on the use of a *boundary scan* [11] which is a tool for testing integrated circuits, printed circuit boards, or complete systems. Joint Test Action Group (JTAG) interface is used for observing and modifying signals in the design. It is possible to inject faults such as short circuits, connection interruptions, permanent logic 1 or 0, and others.

Techniques that allow faults injection directly into the FPGA without the need to modify the implemented circuit provide results that correspond to real operation. To perform such experiments, it is necessary to have an FPGA, in some cases even a specialized board, which leads to the fact that the experiments are not performed on the target platform and the behavior in real operation may be slightly different.

2.4.2 Fault Injector based on the Partial Dynamic Reconfiguration

Research in the field of artificial fault injection is also covered by some members of the doc. Kotásek team. An external SEU injector was developed that is described in more detail in [68]. This injector is based on the SEU generation outside of the FPGA (in PC), so it is not targeted to a specific FPGA board (testing was performed on the ML506 card with the Virtex 5 FPGA technology). The original and the modified bitstream is transported through the JTAG interface and the subsequent dynamic reconfiguration of the FPGA. The process of the SEU generation is divided into four steps:

1. specifying location of the fault injection,
2. reading the related part of the configuration bitstream,
3. the SEU generation = inversion of the specified bit of the bitstream,
4. applying the bitstream using PDR without stopping the FPGA.

Our fault injector is implemented in TCL in two basic layers, the structure of which is shown in Figure 2.11. The first layer (Bitstream Generation Layer) is responsible for

communication with the FPGA through the standard JTAG interface and uses ChipScope libraries. The SEU injection layer is responsible for the read and writes bitstream according to the specified fault location. The last block (Added Functions) makes it possible to drive the SEU generation by external sources, such as an external program or the UART interface.

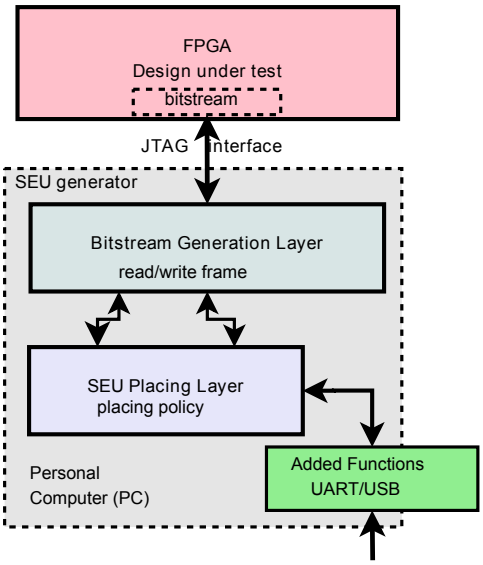


Figure 2.11: An external SEU injector structure [68].

2.4.3 Experimental system

Not only the tool for fault tolerance evaluation but also the experimental system to which the tested techniques are applied is a very important element with an impact on the relevance of the obtained results. The memory-focused fault tolerance techniques presented in [61] are demonstrated and validated on simple memories without another logic. The technique explained in [52] is demonstrated and evaluated only on a multiplexer with two inputs, one simple adder, and a counter. Finite state machines and techniques for ensuring their fault tolerance are proposed in [26]. The evaluation is performed only on a simple finite state machine. It would be possible to give several similar examples, but this is not the scope of this work.

Such simple circuits are fully sufficient for demonstration, but the evaluation results are usually not relevant for their application to real systems, where the various individual fault tolerance techniques must work together. Real systems are generally very complex and contain a number of different blocks that require specific fault tolerance techniques.

Chapter 3

Research Summary

The individual steps to achieve defined goals of this thesis were continuously published at international conferences or in journals. This chapter is based on eight publications included in this thesis (A, B, C, D, E, F, G, H). Individual publications are summarized into thematic sub-chapters. Therefore, for detailed information during reading this summary, the reader is supposed to consult the included papers.

3.1 Experimental System and First Experiments

As a first step, an experimental electro-mechanical system was developed, designed to be as complex as possible, and include various aspects of the digital systems design. This experimental system, together with experiments using the first simplified version of the evaluation platform monitoring the impact of faults, is presented in this chapter.

Key Publication

(A) Podivinsky, J.; Cekan, O.; Simkova, M.; Kotasek, Z.: The Evaluation Platform for Testing Fault-Tolerance Methodologies in Electro-mechanical Applications. *In: 17th Euromicro Conference on Digital Systems Design.* Verona: IEEE Computer Society, 2014, pp. 312-319. ISBN 978-1-4799-5793-4.

The author participation: 32%, conference rank: B1 (Qualis)

This is the first conference publication, which presents the ideas of testing the impact of faults on the electro-mechanical system and at the same time the experimental system itself. It contains the definition of research objectives in this area as well as the definition of the basic components of the verification platform and the implementation of simplified experiments with fault injection. At the same time, the basic ideas of a platform based of functional verification were defined in the article.

Robot for Searching Path through Maze

The robot for searching the path through the maze was chosen as the first experimental electro-mechanical application. The electronic part is represented by the robot controller (RC) implemented in the FPGA, while the mechanical part is represented by the robot in the maze. Unfortunately, no real robot is used, but the robot and its environment are

only simulated, using the freely available simulation environment *Player/Stage* [28]. The use of simulation makes it very easy to change verification scenarios, i.e. the start and the end positions and the maze that the robot goes through. The evaluated FT methodologies can be applied to the electronic robot controller implemented in the FPGA. This controller is designed to represent a complex system composed of components that represent various aspects of digital system design (sequential and combinational circuits, finite state machines, memories and buses), which will allow testing of various FT techniques. The central part is the Wishbone bus [56], to which the partial functional units are connected. Individual functional units process data from the sensors and search the path through the maze.

The First Version of Evaluation Platform for Testing Fault Tolerance

The first version of the evaluation platform consists of two basic blocks - a computer and an FPGA board. These blocks are interconnected using two communication channels. The first is the JTAG interface used by the fault injector, the second channel is used for data transfer between the simulation environment of the robot and its controller. In this version, the data transfer between the PC and the FPGA was realized proprietary as the transformation of the USB interface to the input and output pins of the FPGA. The first version of the platform is divided into three basic parts:

1) Xilinx ML506 board with FPGA Virtex 5 [75] where robot controller is implemented,
2) simulation environment *Player/Stage* [28] for robot simulation and monitoring of mechanical part behavior running on computer, and
3) fault injector [68] running on computer which is used for artificial fault injection.

Not only the fault injector itself is used, but also a tool for searching used bits of the configuration bitstream which represent individual functional units is used in this work. The PlanAhead [22] tool allows to define the positions of individual functional units on the FPGA, which allows to use the RapidSmith [42] tool to analyze the FPGA and generate bits that are related to the defined area on the FPGA. These are mainly configuration bits that represent the used *Look-up Tables* (LUTs).

Experimental Results

The experiments monitoring the impact of faults on the FPGA were performed in order to estimate the susceptibility of the robot controller to faults and to identify typical forms of mechanical failure. After programming the FPGA and starting the robot, faults were injected into the selected functional unit with a period of 2 s, until the robot started to behave unusually. During experiments, two metrics were monitored: (1) the number of fault which leads to failure (2) the way of mechanical part failure. The results are shown in the graph in Figure 3.1. This is a statistical box plot, which shows for each functional unit the range of the number of faults leading to the failure. It is evident that each unit is differently prone to faults, two units (PFU_FSM and PFU_WB) could not even be damaged by the injected faults.

The most common consequences of injected faults observed during experiments were: (1) freezing on place, (2) deadlock (walk around in cycle), (3) crashing into a wall and (4) other, e.g. freezing the robot in one place, then refreezing and walking in a circle, the wrong turn of the robot in the maze, followed by freezing. The proportional representation of these consequences is shown in Figure 3.2.

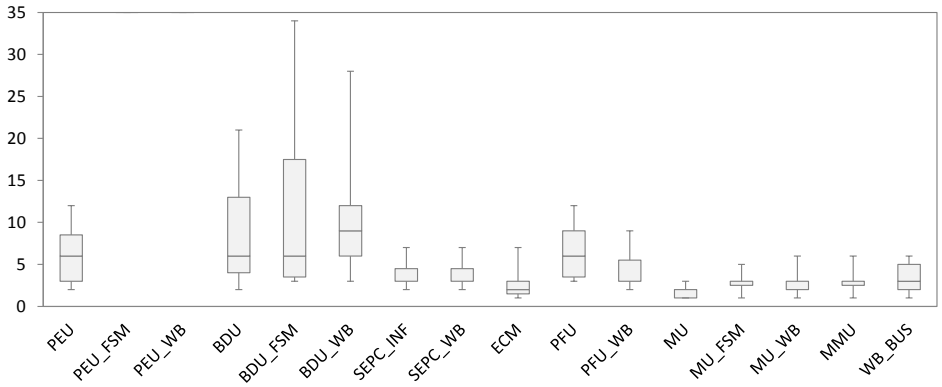


Figure 3.1: Statistical box plot summarizing the number of faults injected into each component which leads to failure.

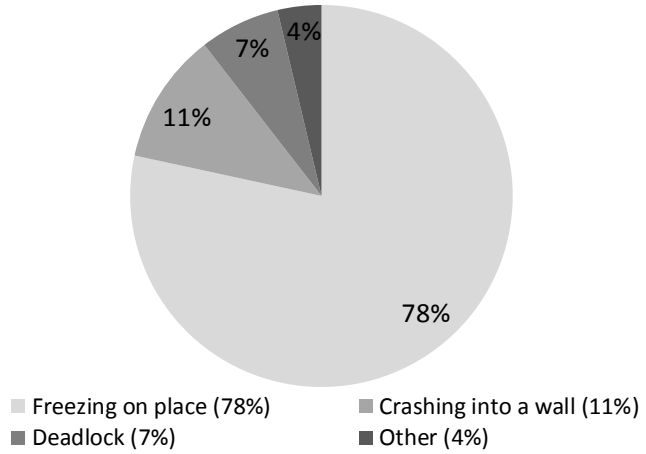


Figure 3.2: The chart representing typical consequences of injected faults on the mission of the robot.

These experiments show that some faults have an impact on the behavior of the robot, and others do not. In some cases, it was necessary to inject a large number of faults (e.g. 34 into BDU_FSM). 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.

3.2 Evaluation Platform for Monitoring Impact of Faults

The aim of this chapter is to introduce a platform for monitoring the impact of faults on an electromechanical system with a control unit implemented in an FPGA. This platform is created based on the techniques and principles used in functional verification. It is the fundamental part of further work because later experiments are based on this platform.

Key Publications

(B) Podivinsky, J.; Cekan, O.; Lojda, J.; Zachariasova, M.; Krcma, M.; Kotasek, Z.: Functional Verification Based Platform for Evaluating Fault Tolerance Properties. *In: Microprocessors and Microsystems. Amsterdam: Elsevier Science*, 2017, vol. 52, no. 5, pp. 145-159. ISSN 0141-9331.

The author participation: 38%, impact factor: 1,045 (Q3), number of citations: 4

(C) Podivinsky, J.; Lojda, J.; Kotasek, Z.: An Experimental Evaluation of Fault-Tolerant FPGA-based Robot Controller. *In: Proceedings of IEEE East-West Design & Test Symposium.* Kazan: IEEE Computer Society, 2018, pp. 63-69. ISBN 978-1-5386-5709-6.

The author participation: 46%

The article published in the Microprocessors and Microsystems journal covers several topics, the main one is the platform for testing the impact of faults on the electro-mechanical system controlled by FPGA. Everything is proposed both from a general point of view and also implemented and demonstrated on a specific example of a robot in a maze. In the second article of this chapter another experiments with the same experimental system extended by the application of fault tolerance technique to some components are presented.

The Process of Monitoring the Impact of Faults on Electro-mechanical System

The proposed process of the fault impact evaluation is divided into three phases. In the first phase,the simulation-based functional verification is used. It tests whether the electronic controller works correctly, according to the specification. In this phase, a set of verification scenarios is acquired. Implementation bugs are detected so they are not later mixed with the impact of faults. It is necessary to create a verification environment including a reference model. It should be noted that the verification environment is connected with the mechanical part.

During the *second phase*, the functional verification is modified so that the electronic controller runs directly on the FPGA and faults can be injected into it. In this phase, previously obtained verification scenarios are used. This ensures that all incorrect behavior is caused by the injected fault, not by the implementation bug. Each verification scenario is repeated several times. During each repetition, a different fault or set of faults is injected (according to the chosen fault injection strategy) and their impact is monitored. The output is a list of faults that, in combination with the given verification scenario, caused an incorrect output of the electronic controller. In the last *third phase*, a detailed examination of situations is performed where the injected fault corrupts the output of the electronics and the impact of this fault on the mechanical part is evaluated. For monitoring the effect of faults on the mechanical part, it is possible to use sensors used by the electronic controller as feedback. Data from these sensors can be analyzed to determine whether the mechanical part behaves according to the specification. Each run is then assigned to one of the following categories: (1) The output from the FPGA and the reference model match, the fault did not occur. (2) The outputs do not match, nevertheless the mechanical part passes its mission. (3) The outputs do not match and at the same time the function of the mechanical part has been disturbed.

In the *second and third* phases, a verification environment that uses an FPGA is used. In this verification environment, the electronic controller is not simulated, but is moved to the FPGA. The electronic controller on the FPGA and the mechanical part thus form an independent unit, which works autonomously. The verification environment only monitors its communication, it functions as an observer. The experimental platform which is composed of a few components running on a computer or on an FPGA evaluation board was designed for these purposes:

1) software part of verification environment for the electronic controller running on a computer,
2) software simulation environment for mechanical part simulation running on a computer,
3) electronic controller implemented into FPGA, and
4) external fault injector [68] running on a computer which allows us to simulate real faults in the FPGA.

Robot in Maze as an Experimental system

The verified circuit into which the faults are injected is the electronic controller of robot for searching a path through the maze. For the first phase, a verification environment implemented according to the UVM (Universal Verification Methodology) was created. The mechanical part, more precisely its simulation, is controlled by DUT outputs and at the same time generates new inputs for DUT.

The verification environment using FPGA (Figure 3.3) is based on the simulation-based verification environment. The verification environment is divided into two parts. The first part is a robot simulation which communicates with the controller on the FPGA. This part works completely autonomously, between the FPGA and the simulation environment an information from the sensors and instructions for the robot movement flow. The second part is the verification environment itself, which assumes most of the components from the original environment. This verification environment operates as an observer; it means that it is only passive monitor of the communication between the FPGA and the robot simulation. The verification environment is also ready to be used in the third phase, in which the values from the sensors are analyzed. The data from sensors are used to monitor the behavior of the robot itself.

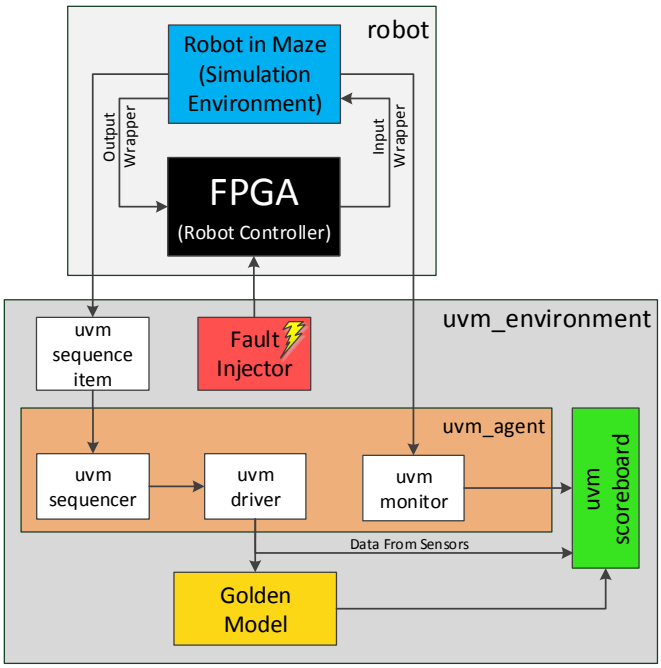


Figure 3.3: The architecture of the extended FPGA-based verification environment used in second and third phase of evaluation process.

Faults can be injected according to the chosen strategy, for example single faults into a selected functional unit or multiple faults into one or more functional units and others. The output of this process is a report summarizing the performed experiments. For each run, the position of the injected fault is also stored for the use in the next phase.

Experimental results

The Figure 3.4 shows the results of experiments, in which a single fault was injected during each run of the verification scenario. The aim of these experiments was to evaluate the susceptibility of individual functional units to single failures and at the same time to demonstrate the use of the verification platform. The blue bars show the number of failures of the robot electronic controller. During these experiments, the robot either stopped or collided with the wall. Collision is considered as the worst possible impact. The red bars show the number of collisions with the wall, in other cases the robot stopped moving. These experiments again show that each functional unit is otherwise prone to failure. However, in some cases the results are completely different from the experiments in the previous chapter. This is probably due to a bug in the fault injector, which was repaired after the first experiments.

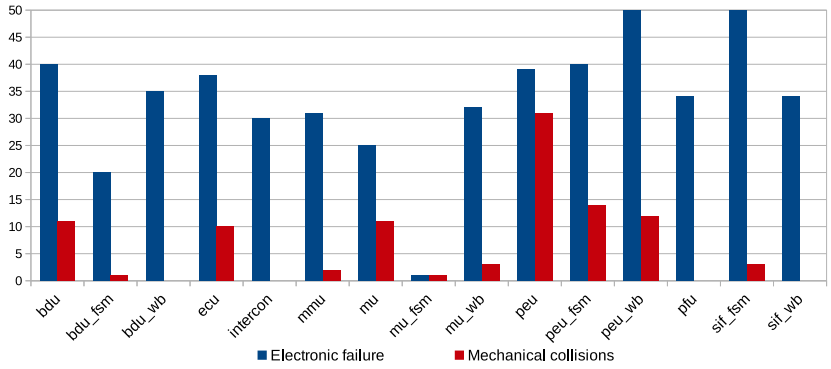


Figure 3.4: Experimental results in functional verification.

Three functional units (PEU, BDU and ECU) were selected for further experiments, these are significant units in terms of the role in the entire robot controller. A fault tolerant version of the robot controller was prepared, where these function blocks were hardened using Triple Modular Redundancy (TMR). These experiments were performed with the injection of single faults into selected functional units, both in unhardened and hardened versions. One fault was injected during each scenario. The analysis of the effect of injected faults on the electronic controller is summarized in Table 3.1. The table clearly displays how the susceptibility to faults decreased after TMR application. Some failures also occurred in the hardened version - this can be the result of the fact that an unhardened voting circuit (*voter*) was used. The advantage is that if a failure occurred in the hardened version, it would not lead to a collision with the wall.

Multiple faults were also injected, again into the hardened and unhardened versions of the robot controller. During each run, faults were injected with a period of 5 s until an error at the output of the electronic controller was detected. The Table 3.1 shows how many injections caused the failure and what was the impact on the mechanical robot. In contrast to the injection of single disorders, it is evident that failures have occurred in a significantly larger number of runs, and the reduction in susceptibility to faults is not as

high as in the case of single faults. This is because no repair mechanism was used and failure could occur in multiple parallel modules at the same time.

Table 3.1: The impact of faults injected into the unhardened and hardened versions of robot controller both on the electronic controller and mechanical part.

Monitored impact	*single faults injection*						*multiple faults injection*					
	PEU		*BDU*		*ECU*		*PEU*		*BDU*		*ECU*	
	noft	*tmr*	*noft*	*tmr*	*noft*	*tmr*	*noft*	*tmr*	*noft*	*tmr*	*noft*	*tmr*
Electronic OK [–]	971	1000	813	996	952	990	656	977	361	917	226	622
Electronic failed [–]	29	0	187	4	48	10	344	23	639	83	774	378
Goal not reached [–]	29	0	187	4	48	10	344	23	639	83	774	378
Collision with wall [–]	0	0	5	0	1	0	0	0	0	0	15	3
Robot stop on place [–]	29	0	182	4	47	10	344	23	639	83	759	375
Reliability improvement [%]	100.0%		97.9%		79.2%		93.3%		87.0%		51.2%	

3.3 Scalability - Change of Electronic Controller

This chapter deals with scalability from the point of view of changing the experimental electronic controller. The whole platform was designed to allow the application and testing of various fault tolerance techniques on both a proposed experimental system and a specific user system. For this reason, it is necessary to demonstrate the possibility of changing the electronic controller and comparing the experimental results.

Key Publictions

(D) Podivinsky, J.; Zachariasova, M.; Cekan, O.; Kotasek, Z.: FPGA Prototyping and Accelerated Verification of ASIPs. *In: IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits and Systems.* Belgrade: IEEE Computer Society, 2015, pp. 145-148. ISBN 978-1-4799-6779-7.

The author participation: 46%, conference rank: B3 (Qualis), number of citations: 5

(E) Podivinsky, J.; Lojda, J.; Cekan, O.; Kotasek, Z.: Evaluation Platform for Testing Fault Tolerance Properties: Soft-core Processor-based Experimental Robot Controller. *In: Proceedings of the 2018 21st Euromicro Conference on Digital System Design. Prague: IEEE Computer Society,* 2018, pp. 229-236. ISBN 978-1-5386-7376-8.

The author participation: 38%, conference rank: B1 (Qualis), number of citations: 1

The paper published at the DDECS conference deals with accelerated functional verification of the processor, it is mainly the transfer of the verified processor to the FPGA, which is the basis for the use of the processor on the FPGA to verify the impact of faults. This is followed by the second article published at the DSD conference, where the processor is used as an experimental electronic controller of an electro-mechanical system and related experiments are presented.

Verification Environment for Processor Running on FPGA

The decision was made to use a Codix RISC processor as an experimental processor, which could be generated using Codasip Studio [16] (FIT owns an academic license). It is a 32-bit processor with the RISC (Reduced Instruction Set Computer) architecture with 7 stages pipelined processing, 32 registers, 512kB of memory and the instruction set contains 59 instructions. For the first phase of the proposed evaluation process, it is necessary to create a verification environment. The advantage is the possibility of automatic generation using Codasip Studio tools.

The second phase in the fault tolerance evaluation process is the functional verification of the system implemented in the FPGA. For this purpose, it is necessary to modify the original verification environment and move the processor into the FPGA. The verification environment modified in this way is shown in Figure 3.5. It should be noted that almost all UVM components have been moved to the FPGA, except of the Reference Model and the Scoreboard. Communication between the software and hardware part of the verification environment is ensured through a proprietary interface. The outputs of the reference model are compared with the outputs of the Device Under Verification (DUV), which are obtained from the FPGA. The contents of the memory and register field are compared after each program execution, as well as the data on the processor output port.

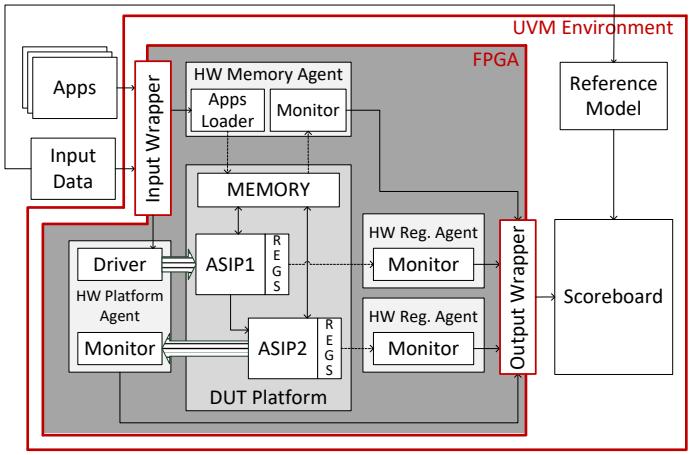


Figure 3.5: The architecture of accelerated verification environment with processor running on FPGA.

During experiments with such a modified verification environment, it was found that the FPGA-accelerated verification environment consumes less time to complete the same task. The acceleration ratio is, on average, 1,7x and slowly grows up with the number of the evaluated test programs. We expected better results. This is caused due to the complex reference model, which runs slower than the DUV on the FPGA. However, for the purpose of testing fault tolerance methodologies, the acceleration ratio is not important. What is more important is that the processor runs on the FPGA and communication with the SW side is ensured, which allows the use with the evaluation platform.

Processor Implemented in FPGA as Experimental Electronic Controller

Another processor was chosen for the implementation of the controller of the electromechanical system, the freely available NEO430 [54] was used. On the one hand, the reason

was free availability and independence from a commercial company, but mainly the request of a colleague who planned to perform experiments with fault tolerance with the NEO430 processor.

The NEO430 processor is a customizable processor for FPGA designs. This processor is based on Texas Instruments MSP430 [70] instruction set architecture. It is a processor with Hardward architecture that has separate data and program memory. The processor already implements standard features like a timer, a watchdog, UART and SPI serial interfaces, general purpose IO ports, an internal bootloader, and internal memory for program code and data. All of the peripheral modules are optional; it is possible to exclude them from implementation to reduce the size of the system. Any additional modules can be connected via a Wishbone bus. As Figure 3.6 shows, the NEO430 processor is the main element of the robot controller for searching the path in the maze. The interface of the robot controller remained the same, the difference is in internal implementation. The input data are processed by a CFU (*Custom Functional Unit*), which allows to implement any function and can also be used to process inputs. On the output side, the processor is complemented with a MOVE unit controlled via input/output ports (GPIO) and ensures the transformation of commands into signals for actuators. For the purposes of experiments, fault tolerance was also applied. The whole robot controller realized by the processor was implemented according to TMR architecture and supplemented by a voting circuit (*voter*).

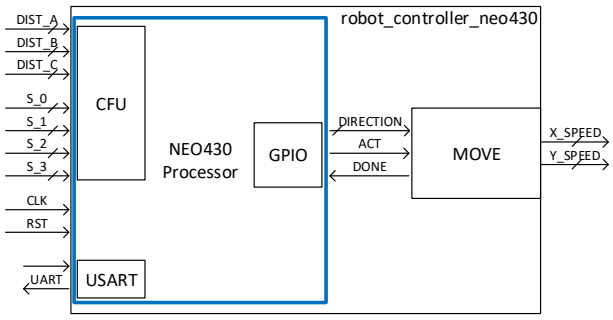


Figure 3.6: The architecture of the robot controller composed of the NEO430 Processor.

Experimental Results - the Comparison of the Original and New Robot Controller

Experiments corresponding with all phases of the verification process were performed. From the fault injection point of view, the second and the third phases are interesting, which monitors the impact of faults both on the output of the electronic controller and the mechanical part. Faults were injected according to two strategies: multiple and single faults. The experiments were performed both with the unhardened version and the triplicated version of the processor-based robot controller. The obtained results were compared with the original version of the robot controller implemented in VHDL.

During multiple fault injection, faults were injected with a period of 15s until the robot reached the goal or until a failure was detected. The results for all controller versions (processor, triplicated processor, original and triplicated original) are summarized in the Table 3.2. The table shows that the processor-based controller is more sensitive to faults. The processor is a more complex system, which leads to a greater susceptibility to faults. It is also obvious that triplication leads to a reduction of fault susceptibility. The last row of the table shows a percentage expression, calculated according to the Equation 3.1. The

table also summarizes the evaluation of the impact of faults on the mechanical part. It is clear that the most common consequence is freezing on the place, which is less critical than the crashing in the wall.

Table 3.2: A comparison of the impact of faults injected into the unhardened and hardened version of the processor-based and the original hard coded robot controller.

Monitored impact	*Multiple fault injection*				*Single fault injection*			
	Processor		*Original*		*Processor*		*Original*	
	noft	*tmr*	*noft*	*tmr*	*noft*	*tmr*	*noft*	*tmr*
Electronic OK [–]	2751	4593	3544	4839	4729	4997	4802	4998
Electronic failed [–]	2201	407	1456	161	271	3	198	2
Electronic failed [%]	44.02%	8.14%	29.12%	3.22%	5.42%	0.06%	3.96%	0.04%
Finish not reached [–]	2179	403	1429	161	271	3	195	2
Collision with wall [–]	55	7	11	0	16	0	1	0
Robot stop on place [–]	2124	396	1418	161	255	3	194	2
Reliability improvement [%]	81.5%		88.9%		98.8%		98.9%	

$$reliab\_improv = \frac{failures_{noft} - failures_{tmr}}{failures_{noft}} * 100 \tag{3.1}$$

The figure shows a statistical box plot where a number of faults which led to the electronic failure is shown. It is again visible that in case of a processor-based controller, a smaller number of injected faults is needed to fail. At the same time, it is obvious that in case of TMR, a higher number of failures is needed to controller failure. Thus, we can state that TMR leads not only to a smaller number of failures but when the failure occurs, it was necessary to inject a larger number of faults.

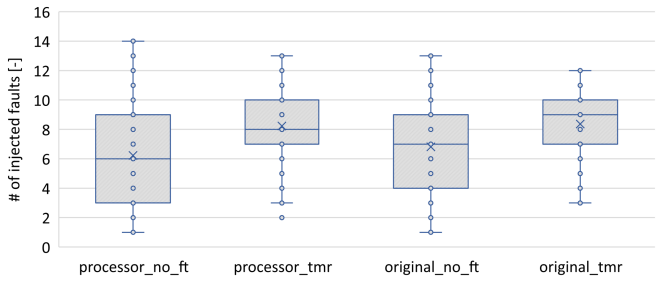


Figure 3.7: The box plot shows a statistical comparison of the number of injected faults which led to electronic failure for both the processor-based and the original robot controllers.

In case of single fault injection, one fault was injected at the beginning of the experiment and then the behavior of both the electronic and mechanical parts was monitored. The results are shown in the Table 3.2, which shows that the number of failures is significantly lower than in the case of multiple fault injection. It can also be seen that almost all injected faults into the TMR version were successfully masked. The results further confirm that the processor is more prone to single faults.

3.4 Comparison of Experimental Reliability Evaluation with Theoretical Calculation

One important question appeared during the experiments with monitoring the impact of artificially injected faults into the FPGA on the mechanical part. Is it possible to find a technique for increasing accuracy and accelerating the evaluation of experimental system susceptibility to faults? This chapter presents a combination of experimental evaluation together with theoretical reliability analysis.

Key Publications

(F) Podivinsky, J.; Lojda, J.; Cekan, O.; Panek, R.; Kotasek, Z.: Reliability Analysis and Improvement of FPGA-based Robot Controller. *In: Proceedings of the 2017 20th Euromicro Conference on Digital System Design.* Vienna: IEEE Computer Society, 2017, pp. 337-344. ISBN 978-1-5386-2146-2.

The author participation: 36%, conference rank: B1 (Qualis), number of citations: 2

(G) Podivinsky, J.; Lojda, J.; Kotasek, Z.: Extended Reliability Analysis of Fault-Tolerant FPGA-based Robot Controller. *In: Proceedings of the 2019 20th Latin American Test Symposium.* Santiago: IEEE Computer Society, 2019, pp. 97-100. ISBN 978-1-72811-755-3.

The author participation: 50%.

The paper published at the DSD conference presents a proposal for a combination of experimental evaluation and theoretical calculation. Experiments extension is the scope of paper published at the LATS conference, where another approach to reliability evaluation was presented and both the original robot control unit and a robot control unit based on a processor implemented in FPGA were used for experimental evaluation. The theoretical calculation of reliability was performed in collaboration with authors colleague J. Lojda.

Theoretical Reliability Analysis

This chapter is based on the theory of reliability and reliability indicators, which is not detailed here, the basics are presented in the attached papers (DSD 2017, LATS 2019), or in the literature [33, 64, 71, 21, 37, 7]. The reliability itself can be quantified with the support of the *theory of probability* as most of the reliability indicators are of a random nature. The length of a time period t of the system operation until the failure occurs is an important starting point in the reliability indicators computation. This variable can be considered the so-called random variable τ [71]. *Cumulative Distribution Function* (CDF) of random variable τ expresses a probability of the system being in a failure state at the time t, it is denoted as $Q(t)$ and is called the *failure function*. Another reliability indicator is the so-called *reliability function* which is denoted as $R(t)$. The reliability function expresses a probability of the system being in a fault-less state at the time t and it is a supplement of the $Q(t)$ according to the Equation 3.2.

$$R(t) = 1 - Q(t) \tag{3.2}$$

As a technique to improve reliability, we used triplication (TMR). It can be declared that this implementation allows us to mask the failure of one module. The structural schematic can be seen in 3.8. The two additional copies of the original functional unit are incorporated into the system. The resulting units are named F_1, F_2 a F_3. The vectors of the input signals x are connected in such a way that each of the functional units F_i works with the same input values. The output signals $f_i(x)$ are connected to the inputs of the so-called *voter* which implements the so-called *majority function*. If we suppose that each of the F_i modules has an equivalent reliability function $R(t)$, then Equation 3.3 that can be used to evaluate the resulting reliability function of the whole TMR module exists [21].

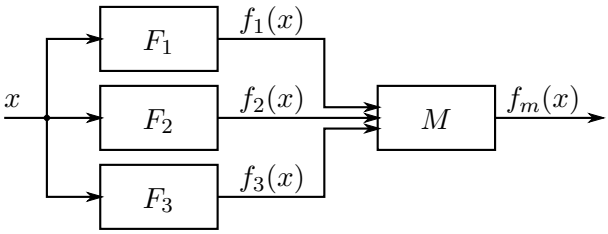


Figure 3.8: A system module whose reliability was improved according to the TMR method.

$$R_{TMR}(t) = 3[R(t)]^2 - 2[R(t)]^3 \tag{3.3}$$

Experimental Evaluation

In this work, the probability of a fault-free state $R_{noft}(t)$ of an unhardened controller of the robot for searching a path through a maze was experimentally determined. In the next step, the probability of a fault-free state $R_{est(t)}$ was calculated based on the equation 3.3 for the triplicated controller. The triplicated robot controller was also implemented in FPGA, which allowed the probability of a fault-free state $R_{tmr}(t)$ to be practically evaluated.

The fault injection was used for practical evaluation of the reliability function. If we suppose that component c is subject to fault injection, then the important parameter to unambiguously describe this type of fault injection is a time delay d_c between two consecutive SEUs injected to c. The d_c actually does not necessarily have to be constant, it can be represented by a random variable with a particular probability distribution. In our experiments, we have experimentally chosen the d_c to be described by the uniform distribution with a mean value of $12s$ and a variance of $2s$.

Faults were injected according to three strategies. The first strategy (*noft*) was fault injection into the unhardened robot controller, component c was defined as the whole robot controller (*nof*). The second strategy (*tmr*) was fault injection into the triplicated version of the robot controller, which respects increased area. Faults were injected into three components c_1, c_2, c_3 (instances of robot controller) concurrently. This led to three times higher fault intensity for the whole robot controller. The third strategy (*tmr1*) was also fault injection into the triplicated robot controller, but this time faults were injected into one component c, which represents the whole hardened robot controller (fault intensity is the same as in *noft* case). The experimental run was repeated 3500 times for all versions of the robot controller units.

The data obtained from the previously described experiments were then processed. The multi-set of all the times measured from the start of the operation of the system

to the first detection of an error on the system outputs was transformed to a discrete *failure function* $Q(t)$ which was then converted to the *reliability function* $R(t)$. The $R(t)$ functions for hard-coded robot controller are shown in Figure 3.9. Red lines show, that *tmr* version (injection into 3 components concurrently, increased area respected) is better than *noft* version, but significantly worse than estimation. Green line shows, that *tmr1* version (injection into whole controller, increased area is not taken into account) is almost the same as estimation. These experiments show, that Equation 3.3 does not consider increased area of TMR system.

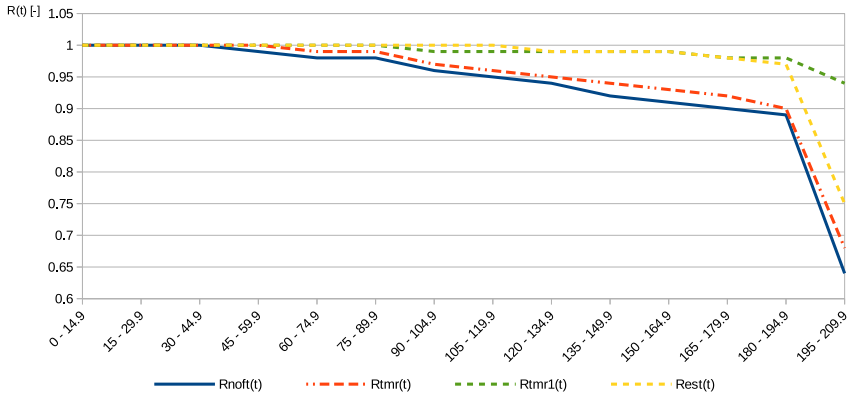


Figure 3.9: An experimental evaluation of the measured results of the reliability function for the *noft* and the *tmr* versions of the *original hard-coded* robot controller.

Additional experiments with processor-based robot controller were performed and presented in LASCAS2019 paper to confirm previous results. Figure 3.10 shows the same chart for processor-based robot controller. The biggest difference is that *tmr* version (injection into 3 components, increased area respected) is worse than *noft* version. The processor is a complex system and a fault injection with higher intensity led to its worse reliability. On the other hand, *tmr1* version (injection into whole controller, increased area is not taken into account) represented by green line is almost the same as estimated reliability. These experiments confirm, that Equation 3.3 does not work with increased area of triplication and does not take into account the possibility of increased fault intensity.

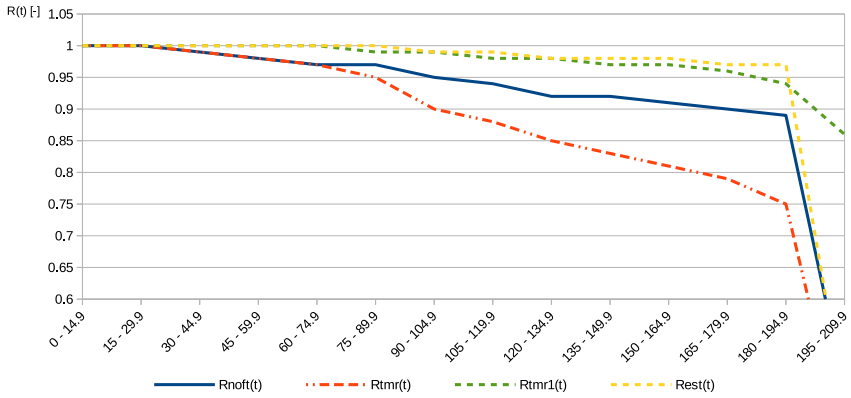


Figure 3.10: The measured results of the reliability function for the *noft* and the *tmr* versions of the *processor-based* robot controller.

3.5 Scalability - Change of the Evaluated System

The presented platform was designed to enable the application and testing of various fault tolerance techniques not only on a prepared experimental system but also on a specific system developed by the user. This chapter summarizes the steps necessary to change the experimental system, but also a demonstration of one particular electro-mechanical system from practice.

Key publication

(H) Podivinsky, J.; Lojda, J.; Panek, R,; Cekan, O.; Krcma, M.; Kotasek, Z.: Evaluation Platform For Testing Fault Tolerance: Testing Reliability of Smart Electronic Locks. *In: Proceedings of the 2020 11th IEEE Latin American Symposium on Circuits and Systems.* San José: IEEE Computer Society, 2020, pp. 1-4. ISBN 978-1-7281-3427-7.

The author participation: 35%, conference rank: B5 (Qualis)

The paper published at the LASCAS conference presents the use of the proposed platform for monitoring the effect of faults on the electronic lock, which represents a real system. In this paper, the evaluation platform is used to solve a real problem.

Scalability and Identification of Application Specific Components of the Evaluation Platform

As the main part of this thesis, a platform for evaluation impact of faults on the electromechanical system was developed and a verification process divided into three phases was provided. Although this platform is designed to the evaluation of any system, it has been demonstrated on only one experimental system which is a robot for searching path through maze and its robot controller implemented in two various ways. Before integration and evaluation of the impact of faults on another system, it is necessary to perform a careful analysis of the experimental system and identify individual application-dependent components.

Modified functional verification is used as a tool for monitoring the impact of faults both on the electronic controller and the mechanical part of an electro-mechanical system. The main difference is that the electronic controller is moved to the FPGA, which allows us to inject artificial faults directly into FPGA and monitor their impact. The verification environment is usually specific for each verified system, however, the versatility of the proposed platform is based on the fact that functional verification is usually used during electronic systems development. Therefore, the verification environment and the reference model (the most important elements dependent on the evaluated system) are available from the previous stage of system development and can be used for a fault tolerance evaluation. To change the experimental system, it is important to have a verification environment and modify it in such a way that the experimental controller works on an FPGA. Thus, an important condition for using the platform is that an electronic controller can be implemented in an FPGA. The basic components of the verification environment and their application dependence are summarized in Figure 3.11.

Each component requires different changes dependent on changed experimental applications. As already mentioned above, a verification environment with a reference model can be created for any system. In our case, there is a condition that it can be implemented in

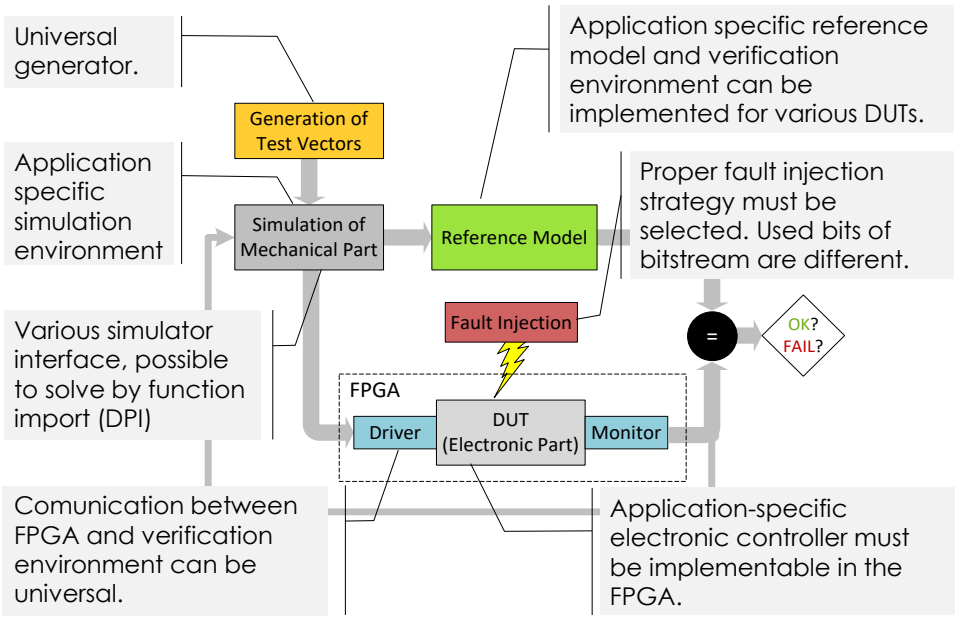


Figure 3.11: The use of functional verification to monitor the impact of faults on the FPGA-controlled electromechanical system and identification of application-specific components.

the FPGA. The mechanical part is also an important element. It is not important whether it is a real mechanical part or its simulation. The availability of sensors that provide feedback on the mechanical part behavior is important. The values provided by these sensors are monitored by the verification environment, which checks whether the system behaves according to its specification. Usually, the use of simulation leads to faster testing and is usually cheaper. The simulation environment of the mechanical part differs from system to system, as well as its interface, which, however, is not a big problem due to the possibility of functions import into the verification environment. Communication between the verification environment and the FPGA is mostly universal. An important component is also the fault injector and the associated selection of fault injection strategy suitable for the experimental system.

Electronic Lock as an Experimental System

The aim of research activities in the field of electronic locks [57] is to evaluate whether faults in the electronic controller can affect the mechanical part, i.e. whether it is possible to cause an unintentional unlocking caused by a fault or to prevent proper locking. This situation can be seen both from the point of view of the random occurrence of faults and from the point of view of artificial induction of faults in the electronic part [58]. The proposed platform for evaluation impact of faults on the electro-mechanical system is suitable for these purposes. The main limitation is that the control electronics must be implemented in the FPGA. Electronic locks usually use various types of motors as a mechanical element, in many cases, it is a stepper motor [72], which was used in the proposed paper.

Most smart embedded systems are based on a processor [32], so the mechanical part is controlled by the processor. Many processor implementations can be configured in the FPGA to meet the condition for using the created platform. The NEO430 [54] processor was chosen for experimental purposes, which is based on the often-used MSP430 [70] processor from Texas Instruments.

The block diagram in Figure 3.12 shows the use of functional verification for monitoring the impact of faults on the stepper motor. It is possible to see the stepper motor itself, more precisely its simulation; the FPGA, to which a processor-based controller is implemented, into which faults are injected; and a reference model for output values comparison. MATLAB and Simulink [47] are used for simulation, stepper motor simulation is a part of Simscape library [48]. The available simulation model is generic and it has been configured according to the stepper motor 28BYJ11-48 [36]. It is a 4-phase stepper motor with a gearbox and a step of 5.625°/64, 4096 steps are needed to turn 360°, without the gearbox 64 steps lead to a 360° rotation.

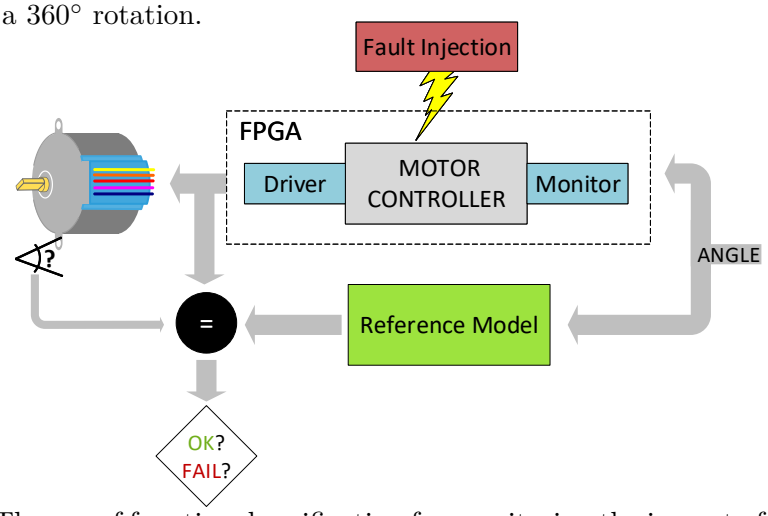


Figure 3.12: The use of functional verification for monitoring the impact of faults on stepper motor controller.

Experimental Results

The attached paper presents experiments corresponding to the second and third phases of the proposed process of monitoring the impact of faults on the electro-mechanical system, which are important in terms of detecting the possibility of unauthorized unlocking or prevent the locking. Both single and multiple faults were injected at regular intervals, and their impact on the output of the electronic controller and the behavior of the stepper motor was monitored. The minimum, maximum, and final rotation angles were monitored and stored. The faults were injected only into the used bits of the bitstream, which corresponds to the LUT tables, in total it was 58496 bits.

Single faults were injected into 6000 randomly selected bits of the bitstream, which is 10% of the total number. Five iterations of experiments with fault injection into the same bits were performed, which made it possible to verify whether the fault impact on electronics and mechanics is deterministic. The table 3.3 illustrates the failure rate of the electronic controller in all three experiment iterations (rows 1, 2, 3, 4 and 5). The total number of electronics failures for each iteration can be seen here. Obviously, the differences are already at the level of the number of faults that lead to failure. We divided the types of electronics failures into three classes: 1) the premature stopping — *Stuck*, 2) unending controller operation — *Time-out*, and 3) the correct termination, although with mismatching values — *Mismatch*. At the same time, it was found out that not all electronics failures led to mechanics failures, as it can be seen in the table in the *Mechanic OK - Total* column. The next columns show the number of these cases divided into individual classes of electronics failure. As can be seen, the highest number appears in the last column, that is when the controller terminated its activity correctly with mismatch values detected. The

reason is that the electronics failure was evaluated strictly, single mismatching output was considered as a failure. The table 3.3 also contains the row *Multi* with the evaluation of multiple fault injection, i.e. a scenario where one fault was injected after another with a period of 5s. It is obvious that the overall behavior is very similar to a single injection, but the numbers are significantly higher. If an attacker injects a set of failures, there is a bigger probability to fail. At the same time, it is more difficult to predict the behavior of the entire system.

Table 3.3: The results and failures classification of single and multiple fault injection.

Iter.	Electronic failure				Mechanic OK			
	Total	Stuck	Time-out	Mis-match	Total	Stuck	Time-out	Mis-match
1	633	159	260	214	210	5	2	203
2	633	174	270	189	186	5	3	178
3	572	93	145	334	331	6	3	322
4	624	172	269	183	183	5	3	175
5	574	100	147	327	327	6	3	318
Multi	5772	1248	1901	2605	592	35	26	531

The set of bits that have proven to cause failure was always a bit different between the iterations. We compared all combinations of two (e.g. $1 \cap 2$, $1 \cap 3$ ...), three (e.g. $1 \cap 2 \cap 3$, $1 \cap 3 \cap 4$ ...) and four (e.g. $1 \cap 2 \cap 3 \cap 4$, $1 \cap 3 \cap 4 \cap 5$ ...) sets of results. We observed that there were cases where the electronics did not fail in all iterations, but also cases where the electronics failed in all iterations, but the classification of failures was not always the same. The sets of failures that led to failure in individual iterations are not the same, but a relatively significant intersection can be found between them. In case of deliberate insertion of faults with the aim of unauthorized control of the lock, deterministic behavior cannot be expected, which complicates the potential attack.

Moreover, we examined the motor rotation angle. Figure 3.13 contains a boxplot chart which displays the maximal angle for all the iterations. As the chart illustrates, the required rotation angle was 4500. The majority of the electronics failures led to a smaller final rotation angle. Only a small number of faults caused a bigger rotation angle. If the goal of the fault injection is to unlock the lock without authorization, it will not be probably reached. On the other hand, if the goal is to prevent the door from locking it correctly, the chances are significantly higher.

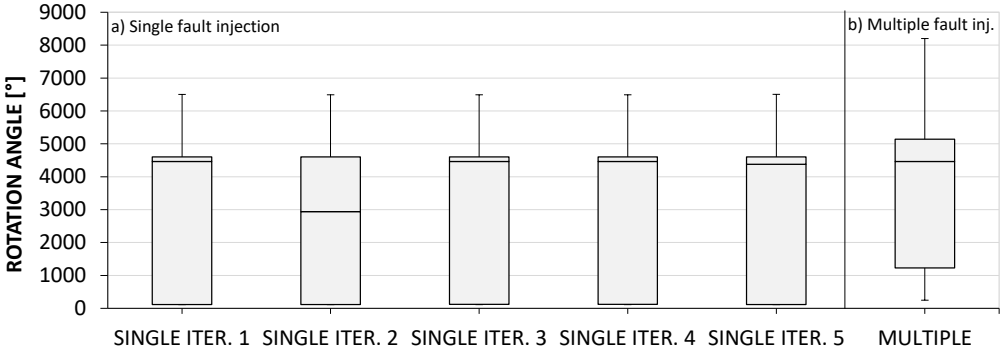


Figure 3.13: Boxplot graph with rotation angle for (a) three experiment iterations with single injection and (b) multiple injection.

3.6 Author's contributions to selected papers

The papers presented in this PhD thesis were prepared in collaboration with the Dependable Systems Research Group at the Faculty of Information Technology, Brno University of Technology led by authors supervisor doc. Kotasek. Although all co-authors contributed to the selected papers, author of this thesis has the main role in all of them, especially in terms of the presented results. This section explicitly summarizes author's contribution to selected papers.

- **Paper A** — the specification and the implementation of the experimental system, the definition of research objectives in this area, the definition of the basic components of the evaluation platform and the implementation of experiments with fault injection.
- **Paper B** — the development of platform for testing the impact of faults on the electro-mechanical system controlled by FPGA, definition of testing process divided into several phases, the demonstration on a specific example of a robot in a maze together with experiments with fault injection.
- **Paper C** — the application of fault tolerance technique to some components of experimental system, experimental evaluation with fault injection.
- **Paper D** — the implementation of the soft-core processor transfer to the FPGA and the subsequent administration and evaluation of experiments.
- **Paper E** — the implementation of the robot controller using a soft-core processor, its integration into the platform and the evaluation of experiments with fault injection.
- **Paper F** — the definition of the basic ideas of the theoretical reliability analysis concept, the implementation of experimental system and the evaluation with fault injection.
- **Paper G** — the implementation of the second experimental system and the evaluation of experiments with fault injection.
- **Paper H** — the implementation of the stepper motor controller, the integration of the verified system and the proposed platform, experimental evaluation.

3.7 List of Other Publications

In addition to proposed core publications, the author participates on a number of other publications, most of which were created within the team of doc. Kotásek and the topics are directly related to the thesis topic. The author also participated on several publications based on projects with different topics.

3.7.1 Publications Related to the Thesis Topic

- Podivinsky, J.; Simkova, M.; Kotasek, Z.: Complex Control System for Testing Fault-Tolerance Methodologies. *In: Proceedings of The Third Workshop on Manufacturable and Dependable Multicore Architectures at Nanoscale.* Dresden: COST, European Cooperation in Science and Technology, 2014, pp. 24-27. ISBN 978-2-11-129175-1.

The author participation: 70%

- Podivinsky, J.: Testing Fault-Tolerance Properties in FPGA based Electro-mechanical Applications. *In: Počítačové architektury a diagnostika 2014.* Liberec: Technical University of Liberec, 2014, pp. 13-18. ISBN 978-80-7494-027-9.

  The author participation: 100%

- Podivinsky, J.; Cekan, O.; Simkova, M.; Kotasek, Z.: The Evaluation Platform for Testing Fault-Tolerance Methodologies in Electro-mechanical Applications. *In: Microprocessors and Microsystems.* Amsterdam: Elsevier Science, 2015, vol. 39, no. 8, pp. 1215-1230. ISSN 0141-9331.

  The author participation: 32%, impact factor: 1,045 (Q3), number of citations: 2

- Cekan, O.; Podivinsky, J.; Kotasek, Z.: Software Fault Tolerance: the Evaluation by Functional Verification. *In: Proceedings of the 18th Euromicro Conference on Digital Systems Design.* Funchal: IEEE Computer Society, 2015, pp. 284-287. ISBN 978-1-4673-8035-5.

  The author participation: 45%, conference rank: B1 (Qualis), number of citations: 4

- Podivinsky, J.; Simkova, M.; Kotasek, Z.: Radiation Impact on Mechanical Application Driven by FPGA-based Controller. *In: Proceedings of The Fourth Workshop on Manufacturable and Dependable Multicore Architectures at Nanoscale.* Grenoble: COST, European Cooperation in Science and Technology, 2015, pp. 13-16.

  The author participation: 65%

- Podivinsky, J.: Využití funkční verifikace pro ověřování metodik pro zajištění odolnosti proti poruchám. *In: Počítačové architektury a diagnostika PAD 2015.* Zlín: Faculty of Applied Informatics, Tomas Bata University in Zlín, 2015,pp. 7-12. ISBN 978-80-7454-522-1.

  The author participation: 100%

- Podivinsky, J.; Cekan, O.; Lojda, J.; Kotasek, Z.: Verification of Robot Controller for Evaluating Impacts of Faults in Electro-mechanical Systems. *In: Proceedings of the 19th Euromicro Conference on Digital Systems Design.* Limassol: IEEE Computer Society, 2016, pp. 487-494. ISBN 978-1-5090-2816-0.

  The author participation: 45%, conference rank: B1 (Qualis)

- Podivinsky, J.; Cekan, O.; Lojda, J.; Kotasek, Z.: Functional Verification as a Tool for Monitoring Impact of Faults in SRAM-based FPGAs. *In: Proceedings of the 2016 International Conference on Field Programmable Technology.* Xi'an: IEEE Computer Society, 2016,pp. 293-294. ISBN 978-1-5090-5602-6.

  The author participation: 51%, number of citations: 1

- Podivinsky, J.: Funkční verifikace jako nástroj pro sledování vlivu poruch na elektromechanický systém. *In: Počítačové architektury a diagnostika PAD 2016.* Bořetice - Kraví Hora: Faculty of Information Technology BUT, 2016, pp. 101-104. ISBN 978-80-214-5376-0.

  The author participation: 100%

- Lojda, J.; Podivinsky, J.; Kotasek, Z.: Redundant Data Types and Operations in HLS and their Use for a Robot Controller Unit Fault Tolerance Evaluation. *In: Proceedings of IEEE East-West Design Test Symposium.* Novi Sad: IEEE Computer Society, 2017, pp. 359-364. ISBN 978-1-5386-3299-4.

  The author participation: 33%

- Lojda, J.; Podivinsky, J.; Kotasek, Z.; Krcma, M.: Data Types and Operations Modifications: a Practical Approach to Fault Tolerance in HLS. *In: Proceedings of IEEE East-West Design Test Symposium.* Novi Sad: IEEE Computer Society, 2017, pp. 273-278. ISBN 978-1-5386-3299-4.

  The author participation: 38%, number of citations: 1

- Cekan, O.; Podivinsky, J.; Kotasek, Z.: Program Generation Through a Probabilistic Constrained Grammar. *In: Proceedings of 21st Euromicro Conference on Digital System Design, DSD 2018.* Praha: IEEE Computer Society, 2018, pp. 214-220. ISBN 978-1-5386-7376-8.

  The author participation: 5%, conference rank: B1 (Qualis), number of citations: 1

- Lojda, J.; Podivinsky, J.; Cekan, O.; Panek, R.; Kotasek, Z.: FT-EST Framework: Reliability Estimation for the Purposes of Fault-Tolerant System Design Automation. *In: Proceedings of the 2018 21st Euromicro Conference on Digital System Design.* Praha: IEEE Computer Society, 2018, pp. 244-251. ISBN 978-1-5386-7376-8.

  The author participation: 25%, conference rank: B1 (Qualis)

- Lojda, J.; Podivinsky, J.; Kotasek, Z.; Krcma, M.: Majority Type and Redundancy Level Influences on Redundant Data Types Approach for HLS. *In: 2018 16th Biennial Baltic Electronics Conference (BEC).* Tallinn: IEEE Computer Society, 2018, pp. 1-4. ISBN 978-1-5386-7312-6.

  The author participation: 35%

- Panek, R.; Lojda, J.; Podivinsky, J.; Kotasek, Z.: Partial Dynamic Reconfiguration in an FPGA-based Fault-Tolerant System: Simulation-based Evaluation. *In: Proceedings of IEEE East-West Design Test Symposium.* Kazaň: IEEE Computer Society, 2018, pp. 129-134. ISBN 978-1-5386-5710-2.

  The author participation: 31%

- Lojda, J.; Podivinsky, J.; Kotasek, Z.: Fault Tolerance Properties of Systems Generated with the Use of High-Level Synthesis. *In: Proceedings of IEEE East-West Design Test Symposium.* Kazan: IEEE Computer Society, 2018, pp. 80-86. ISBN 978-1-5386-5710-2.

  The author participation: 35%

- Cekan, O.; Podivinsky, J.; Lojda, J.; Panek, R.; Krcma, M.; Kotasek, Z.: Testing Reliability of Smart Electronic Locks: Analysis and the First Steps Towards. *In: Proceedings of the 2019 22nd Euromicro Conference on Digital System Design.* Kalithea: Institute of Electrical and Electronics Engineers, 2019, pp. 506-513. ISBN 978-1-7281-2861-0.

  The author participation: 19%, conference rank: B1 (Qualis)

- Lojda, J.; Podivinsky, J.; Kotasek, Z.: Reliability Indicators for Automatic Design and Analysis of Fault-Tolerant FPGA Systems. *In: 20th IEEE Latin American Test Symposium (LATS 2019).* Santiago: IEEE Computer Society, 2019, pp. 93-96. ISBN 978-1-7281-1756-0.

  The author participation: 30%

- Lojda, J.; Podivinsky, J.; Cekan, O.; Panek, R.; Krcma, M.; Kotasek, Z.: Automatic Design of Reliable Systems Based on the Multiple-choice Knapsack Problem. *In: Proceedings of the 23rd International Symposium on Design and Diagnostics of Electronic Circuits and Systems.* Novi Sad: Institute of Electrical and Electronics Engineers, 2020, pp. 1-4. ISBN 978-1-7281-9938-2.

  The author participation: 20%, conference rank: B3 (Qualis)

- Panek, R.; Lojda, J.; Podivinsky, J.; Kotasek, Z.: 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): proceedings of technical papers.* Hsinchu: IEEE Computer Society, 2020, pp. 121-124. ISBN 978-1-7281-6083-2.

  The author participation:10%, conference rank: B4 (Qualis)

- Lojda, J.; Panek, R.; Podivinsky, J.; Cekan, O.; Krcma, M.; Kotasek, Z.: Hardening of Smart Electronic Lock Software against Random and Deliberate Faults. *In: Proceedings of the 2020 23nd Euromicro Conference on Digital System Design.* Portorož: Institute of Electrical and Electronics Engineers, 2020, pp. 680-683. ISBN 978-1-7281-9535-3.

  The author participation: 18%, conference rank: B1 (Qualis)

- Lojda, J.; Panek, R.; Podivinsky, J.; Cekan, O.; Krcma, M.; Kotasek, Z.: Analysis of Software-Implemented Fault Tolerance: Case Study on Smart Lock. *In: 2020 IEEE East-West Design Test Symposium (EWDTS) Proceedings.* Varna: Institute of Electrical and Electronics Engineers, 2020, s. 24-28. ISBN 978-1-7281-9899-6.

  The author participation: 15%

3.7.2 Other Publications

These are the papers which were published but they describe problems which do not belong to the thesis.

- Podivinsky, J.; Cekan, O.; Krcma, M.; Burget, R.; Hruska, T.; Kotasek, Z.: A Processor Optimization Framework for a Selected Application. *In: Proceedings of IEEE East-West Design Test Symposium.* Kazan: IEEE Computer Society, 2018, pp. 564-574. ISBN 978-1-5386-5710-2.

  The author participation: 20%, number of citations: 1

- Podivinsky, J.; Cekan, O.; Krcma, M.; Burget, R.; Hruska, T.; Kotasek, Z.: Multidimensional Pareto Frontiers Intersection Determination and Processor Optimization Case Study. In: Proceedings of the 2019 22nd Euromicro Conference on Digital System Design. Kalithea: Institute of Electrical and Electronics Engineers, 2019, pp. 597-600. ISBN 978-1-7281-2861-0.

  The author participation: 20%, conference rank: B1 (Qualis)

- Podivinsky, J.; Cekan, O.; Krcma, M.; Burget, R.; Hruska, T.; Kotasek, Z.: Iterative Algorithm for Multidimensional Pareto Frontiers Intersection Determination. In: 2020 IEEE 11th Latin American Symposium on Circuits Systems (LASCAS). San José: IEEE Circuits and Systems Society, 2020, pp. 1-4. ISBN 978-1-7281-3427-7.

  The author participation: 20%, conference rank: B5 (Qualis)

3.8 Research Projects and Grants

- FIT-S-20-6309 — *Design, Optimization and Evaluation of Application Specific Computer Systems*, Brno University of Technology. Team member.
- 8A18014, Proposal ID 783119-2 — *SECREDAS - Product Security for Cross Domain Reliable Dependable Automated Systems*, ECSEL Joint Undertaking. Team member.
- FIT-S-17-3994 — *Advanced Parallel and Embedded Computer Systems*, Brno University of Technology. Team member.
- LQ1602 — *IT4Innovations Excellence in Science*, Ministry of Education, Youth and Sports of Czech Republic. Team member.
- 7H14002, 621439 — *Algorithms, Design Methods, and Many-Core Execution Platform for Low-Power Massive Data-Rate Video and Image Processing*, Artemis Joint Undertaking. Team member.
- FIT-S-14-2297 — *Architecture of Parallel and Embedded Computer Systems*, Brno University of Technology. Team member.
- LD12036 — *Methodologies for Fault Tolerant Systems Design Development, Implementation and Verification*, Ministry of Education, Youth and Sports of Czech Republic. Team member.
- ED1.1.00/02.0070 — *Centre of Excellence IT4Innovations*, Ministry of Education, Youth and Sports of Czech Republic. Team member.

Chapter 4

Conclusions

The research presented in this thesis is focused on monitoring the impact of faults on the FPGA-based system during fault tolerance properties evaluation. Both existing approaches and a newly proposed approach based on functional verification divided into three phases are presented. The use of functional verification proves to be very advantageous because the verification environment is usually implemented during the development of the system and it is possible to modify further and use it.

In the Chapter 1.1 questions that defined the basic motivation of this work were asked:

- *What will be the results of fault tolerance techniques on real systems?*
- *Is it possible to rely on the fact that not all faults in the electronic part of the system affect the behavior of the controlled mechanical application?*
- *Can functional verification be used to evaluate the effect of faults on fault-tolerant systems?*

I believe that during my research, a tool which helps to answer these questions was developed. The use of functional verification has proven to be a very good step, the verification environment after certain modifications can be used to advantage. Experiments have shown that not all faults with impact on the output of the electronic part of the system necessarily affect the behavior of the mechanical part.

The goals of the thesis are defined in the Chapter 1.1, they are listed again together with a short comment:

Goal 1. Design and create an evaluation platform targeted to FPGA technology which allows to test fault tolerant techniques and allows to monitor the impact of faults not only on the output of the electronic part, but also on controlled mechanical application.

1.1. Functional verification will serve as the basic technique that will be used to verify the correctness of the outputs of the tested system affected by fault injection.

1.2. An integral part of the whole platform will be a fault injection tool, which was previously developed by the team of doc. Kotásek. Thus, this thesis builds on the earlier work of this team.

1.3. The core of this platform forms an experimental electromechanical application, which means a system consisting of an electronic controller implemented on an FPGA and mechanical applications controlled by this controller. Such a system makes it possible to monitor the effect of faults not only on the output of the electronic controller, but also on the controlled mechanical application.

The platform for evaluation fault tolerance properties is the main topic of the paper [B] presented in the Chapter 3.2. Functional verification really serves as a basic technique for monitoring the correctness of electro-mechanical system behavior and a fault injector developed by the team of doc. Kotásek is used. The robot for searching a path through maze and its electronic controller represents the experimental electromechanical application presented in paper [A] and Chapter 3.1.

Goal 2. The developed test platform is followed by the design of a process for monitoring the impact of faults on the electro-mechanical application using the proposed evaluation platform.

2.1. The proposed process for monitoring the impact of faults will reflect the experiences obtained during experimental work with the designed platform and the created experimental system.

2.2. The proposed process also includes a description of the necessary activities related to scalability, it means the use of another experimental system.

2.3. The proposed process allows scalability, it is generalized in such way that it can be used for evaluation with the use of another experimental electromechanical system. It is also demonstrated on a second experimental electromechanical application.

The process of fault tolerance evaluation is defined and divided into three steps in paper [B] and Chapter 3.2. Lots of experiments were performed with the proposed platform, which were published in journals and on conferences. Experiments with the use of fault tolerance to hardening of selected components of the robot controller are the content of the paper [C]. Scalability is referred from the point of view of the change of the electronic controller, which is presented in the papers [D] and [E] mentioned in the Chapter 3.3, experiments are also included. In the Chapter 3.4 the publications [F] and [G] are presented. These publications deal with the comparison of theoretical analysis and experimental evaluation of reliability using implemented robot controllers (both hard coded and processor-based). Scalability is also referred from the perspective of changing the complete experimental system, which is the content of the Chapter 3.5 and paper [H].

4.1 Summary of Main Contributions

This chapter summarizes the most important contributions of this thesis.

The use of functional verification

- The main contribution of this work is a demonstration of the use of functional verification for monitoring the impact of faults on electromechanical systems controlled by FPGA. In the contrast with the literature [6], functional verification is used in combination with fault injection directly into the FPGA. Such an approach required to design a way to move the circuit under test to an FPGA, which allowed the use of a fault injector to inject faults directly into the FPGA. The verification environment is usually implemented during the implementation of the electronic system, so its use is also offered for verifying the impact of faults. Various tools and ready-made components are available for functional verification, which is a great advantage of this approach.

Evaluation platform and process for monitoring impact of faults on electro-mechanical system

- Within this work, a comprehensive platform for monitoring the impact of faults on the electro-mechanical system was designed and functional verification is only one important part. The platform also includes a proposed process of fault tolerance evaluation divided into three phases. In the first phase, the classic functional verification based on the simulation takes place and the performed verification scenarios are repeated in the next phases. The second and the third phases work with a verified circuit implemented in the FPGA and monitor the effect of faults on the electronic controller and the controlled mechanical part.

Demonstration of the platform using a series of experiments

- The proposed platform has been demonstrated in a number of experiments. The main experimental system was a robot for searching a path through a maze with two types of control units. Experiments which compare experimental reliability evaluations with theoretical reliability analyzes were also performed. The result is a useful tool that can be used to evaluate various fault tolerance techniques.

Scalability - the change of experimental system

- An important point is the identification of application-specific parts of the evaluation platform and the design of a method of scalability, i.e. changes of the experimental electro-mechanical system. As a second experimental system, a smart lock consisting of a stepper motor and its control processor implemented in an FPGA was introduced. The output is experiments with interesting results from this area. It turned out that the proposed platform can also be used to solve practical tasks, in this case monitoring the impact of faults on the electronic lock.

4.2 Possibilities of Future Research

It is possible to continue working on this topic and do further research. Several directions that seem promising for future research have been defined.

1. The first direction is definitely the use of the created platform and experimental system for the evaluation of fault tolerance techniques. The platform can thus become a tool for comparing the properties of individual techniques, both new and also existing techniques. As an example, the work of colleague whose research topic is the use of reconfiguration for faulty module recovery should be quoted. He investigates the properties of a reconfiguration controller in terms of its fault tolerance properties. He uses the proposed platform for experimental evaluation.

2. The second direction is the use of the platform, or the proposed process and principles, to evaluate the impact of faults on real systems. As an experimental system, the developer can use the real system which is developed and evaluate its fault tolerance properties by faults injection. Related to this point is the improvement of the platform in terms of user-friendliness, the creation of a more user-friendly graphical interface, and the like.

3. As a continuation of the research, it may also be improvements of the platform, both in terms of the above mentioned user interface, but also in terms of the used components. In this work, a fault injector previously developed by the team of doc. Kotásek is used. Nevertheless, it could be interesting to connect the platform with another fault injector and compare the results of the experimental evaluation.

4. Another possible extension is to simplify the way to change the experimental system. Currently, manual implementation of some components for monitoring the behavior both of the electronic and mechanical parts of the verified system after moving the electronic controller to the FPGA is required. An interesting direction of research could be the automatic generation of these components.

Bibliography

[1] *IEEE Std. 1800-2017, IEEE Standard for SystemVerilog — Unified Hardware Design, Specification, and Verification Language*. 2017.

[2] Alderighi, M.; Casini, F.; d'Angelo, S.; et al.: Evaluation of single event upset mitigation schemes for SRAM based FPGAs using the FLIPPER fault injection platform. In *Defect and Fault-Tolerance in VLSI Systems, 2007. DFT'07. 22nd IEEE International Symposium on*. IEEE. 2007. pp. 105–113.

[3] Alderighi, M.; D'Angelo, S.; Mancini, M.; et al.: A fault injection tool for SRAM-based FPGAs. In *On-Line Testing Symposium, 2003. IOLTS 2003. 9th IEEE*. IEEE. 2003. pp. 129–133.

[4] Araiza-Illan, D.; Western, D.; Pipe, A.; et al.: Coverage-Driven Verification. In *Haifa Verification Conference*. Springer. 2015. pp. 69–84.

[5] Bennett, J.; Jack, A.; Mecrow, B.; et al.: Fault-tolerant control architecture for an electrical actuator. In *Power Electronics Specialists Conference, 2004. PESC 04. 2004 IEEE 35th Annual*, vol. 6. June 2004. ISSN 0275-9306. pp. 4371–4377.

[6] Benso, A.; Bosio, A.; Di Carlo, S.; et al.: A Functional Verification based Fault Injection Environment. In *Defect and Fault-Tolerance in VLSI Systems, 2007. DFT'07. 22nd IEEE International Symposium on*. IEEE. 2007. pp. 114–122.

[7] Benso, A.; Prinetto, P.: *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*. Springer Publishing Company. first edition. 2010. ISBN 1-4020-7589-8.

[8] Bérard, B.; Bidoit, M.; Finkel, A.; et al.: *Systems and software verification: model-checking techniques and tools*. Springer Science & Business Media. 2013.

[9] Bergeron, J.: *Writing testbenches using SystemVerilog*. Springer Science & Business Media. 2007.

[10] Bernardeschi, C.; Cassano, L.; Domenici, A.; et al.: Accurate simulation of SEUs in the configuration memory of SRAM-based FPGAs. In *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2012 IEEE International Symposium on*. IEEE. 2012. pp. 115–120.

[11] Bleeker, H.; van Den Eijnden, P.; De Jong, F.: *Boundary-scan test: a practical approach*. Springer Science & Business Media. 2011.

[12] Bolchini, C.; Miele, A.; Santambrogio, M. D.: 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. ISBN 0-7695-2885-6. pp. 87–95.

[13] Bridgford, B.; Carmichael, C.; Tseng, C. W.: Single-event Upset Mitigation Selection Guide. *Xilinx Application Note, XAPP987 (v1. 0)*. 2008.

[14] Ceschia, M.; Violante, M.; Reorda, M.; et al.: Identification and classification of single-event upsets in the configuration memory of SRAM-based FPGAs. *IEEE Transactions on Nuclear Science*. vol. 50, no. 6. 2003: pp. 2088–2094. ISSN 0018-9499.

[15] Cheatham, J. A.; Emmert, J. M.; Baumgart, S.: A Survey of Fault Tolerant Methodologies for FPGAs. New York, NY, USA: ACM. 2006. ISSN 1084-4309. pp. 501–533.

[16] Codasip: Codasip Studio. 2020.
Retrieved from: https://codasip.com/codasip-studio/

[17] ČSN, I.: 50 (191). *Mezinárodnì elektrotechnickỳ slovník*. vol. 191. 1993.

[18] Cutts, S.: A collaborative approach to the More Electric Aircraft. In *Power Electronics, Machines and Drives, 2002. International Conference on (Conf. Publ. No. 487)*. June 2002. ISSN 0537-9989. pp. 223–228.

[19] De Sio, C.; Azimi, S.; Bozzoli, L.; et al.: Radiation-induced Single Event Transient effects during the reconfiguration process of SRAM-based FPGAs. *Microelectronics Reliability*. vol. 100. 2019: page 113342.

[20] De Sio, C.; Azimi, S.; Sterpone, L.: On the Evaluation of SEU Effects on AXI Interconnect Within AP-SoCs. In *International Conference on Architecture of Computing Systems*. Springer. 2020. pp. 215–227.

[21] Dhillon, B.: *Applied Reliability and Quality: Fundamentals, Methods and Procedures*. Springer Series in Reliability Engineering. Springer London. 2007. ISBN 9781846284984.

[22] Dorairaj, N.; Shiflet, E.; Goosman, M.: PlanAhead Software as a Platform for Partial Reconfiguration. *Xcell Journal*. vol. 55, no. 68-71. 2005: page 84.

[23] Einspruch, N.: *Application specific integrated circuit (ASIC) technology*. vol. 23. Academic Press. 2012.

[24] Feret, J.: Static analysis of digital filters. In *European Symposium on Programming*. Springer. 2004. pp. 33–48.

[25] Freitas, A.: *Hardware-Software Co-verification Using the SystemVerilog DPI*. Techn. Univ. Chemnitz, Fakultät für Informatik. 2007.

[26] Frigerio, L.; Salice, F.: RAM-based fault tolerant state machines for FPGAs. In *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems, DFT '07*. 2007. ISSN 1550-5774. pp. 312–320.

[27] Geffroy, J.-C.; Motet, G.: *Design of dependable computing systems*. Kluwer Academic Publishers. 2002. ISBN 1-4020-0437-0.

[28] Gerkey, B.; Vaughan, R. T.; Howard, A.: The Player/Stage Project: Tools for Multi-robot and Distributed Sensor Systems. In *Proceedings of the 11th International Conference on Advanced Robotics*, vol. 11. 2003. pp. 317–323.

[29] Goloubeva, O.; Rebaudengo, M.; Reorda, M. S.; et al.: *Software-implemented hardware fault tolerance*. Springer. 2006.

[30] Gong, L.; Wu, T.; Nguyen, N. T.; et al.: A Programmable Configuration Controller for fault-tolerant applications. In *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE. 2016. pp. 117–124.

[31] Guo, X.; Dutta, R. G.; Mishra, P.; et al.: Scalable SoC trust verification using integrated theorem proving and model checking. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE. 2016. pp. 124–129.

[32] Han, D.; Kim, H.; Jang, J.: Blockchain based smart door lock system. In *2017 International Conference on Information and Communication Technology Convergence (ICTC)*. 2017. pp. 1165–1167.

[33] Hlavička, J.: *Číslicové systémy odolné proti poruchám*. ČVUT. 1992. ISBN 80-01-00852-5. 330 pp.

[34] Kastensmidt, F. L.; Carro, L.; Reis, R.: *Fault-tolerance techniques for SRAM-based FPGAs*. Springer. 2006. ISBN 0-387-31068-1.

[35] Kastensmidt, F. L.; Neuberger, G.; Carro, L.; et al.: Designing and testing fault-tolerant techniques for SRAM-based FPGAs. In *Proceedings of the 1st conference on Computing frontiers*. ACM. 2004. pp. 419–432.

[36] Kiatronics®: 28BYJ-48 - 5V Stepper Motor. http://robocraft.ru/files/datasheet/28BYJ-48.pdf. 2015. accessed: 2019-03-26.

[37] Koren, I.; Krishna, C. M.: *Fault-Tolerant Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.. 2007. ISBN 978-0-12-088525-1.

[38] Krasich, M.: How to estimate and use MTTF/MTBF would the real MTBF please stand up? In *2009 Annual Reliability and Maintainability Symposium*. IEEE. 2009. pp. 353–359.

[39] Kropf, T.: *Introduction to Formal Hardware Verification*. Springer. 1999. ISBN 3-540-65445-3.

[40] Kuuhn, J. M.; Schweizer, T.; Peterson, D.; et al.: Testing Reliability Techniques for SoCs with Fault Tolerant CGRA by Using Live FPGA Fault Injection. In *2013 International Conference on Field-Programmable Technology (FPT)*. IEEE. 2013. pp. 462–465.

[41] Lavin, C.; Padilla, M.; Lamprecht, J.; et al.: Rapidsmith: Do-it-yourself cad tools for xilinx fpgas. In *2011 21st International Conference on Field Programmable Logic and Applications*. IEEE. 2011. pp. 349–355.

[42] Lavin, C.; Padilla, M.; Lundrigan, P.; et al.: Rapid Prototyping Tools for FPGA Designs: RapidSmith. In *The 2010 International Conference on Field-Programmable Technology (FPT)*. Dec 2010. pp. 353–356.

[43] Leen, G.; Heffernan, D.: Expanding automotive electronic systems. *Computer*. vol. 35, no. 1. Jan 2002: pp. 88–93. ISSN 0018-9162.

[44] Li, Y.; Nelson, B.; Wirthlin, M.: Reliability models for SEC/DED memory with scrubbing in FPGA-based designs. *IEEE Transactions on Nuclear Science*. vol. 60, no. 4. 2013: pp. 2720–2727.

[45] Liu, M.; Zeng, Z.; Su, F.; et al.: Research on Fault Injection Technology for Embedded Software based on JTAG Interface. In *2016 11th International Conference on Reliability, Maintainability and Safety (ICRMS)*. IEEE. 2016. pp. 1–6.

[46] López-Ongil, C.; Garcia-Valderas, M.; Portela-García, M.; et al.: Autonomous fault emulation: a new FPGA-based acceleration system for hardness evaluation. *IEEE Transactions on Nuclear Science*. vol. 54, no. 1. 2007: pp. 252–261.

[47] MathWork®: MATLAB and Simulink. https://www.mathworks.com/. 2018. accessed: 2019-03-20.

[48] MathWork®: Stepper motor. https://www.mathworks.com/help/phys-mod/sps\/powersys/ref/steppermotor.html. 2019. accessed: 2019-03-20.

[49] MentorGraphics: ModelSim User's Manual.
Retrieved from:
https://www.microsemi.com/document-portal/doc_view/131619-modelsim-user

[50] MentorGraphics: The Questa Verification Solution.
Retrieved from: https://www.mentor.com/products/fv/upload/questa-verification-solution-249ff119-dc3f-4341-8bab-a89af45d926d

[51] Meyer, A.: *Principles of Functional Verification*. Elsevier Science. 2003. ISBN 0-7506-7617-5.

[52] Naseer, M.; Sharma, P.; Kshirsagar, R.: Fault Tolerance in FPGA Architecture Using Hardware Controller - A Design Approach. In *International Conference on Advances in Recent Technologies in Communication and Computing, ARTCom '09*. 2009. pp. 906–908.

[53] Nidhin, T.; Bhattacharyya, A.; Behera, R.; et al.: Verification of Fault Tolerant Techniques in Finite State Machines Using Simulation based Fault Injection Targeted at FPGAs for SEU Mitigation. In *2017 4th International Conference on Electronics and Communication Systems (ICECS)*. IEEE. 2017. pp. 153–157.

[54] Nolting, S.: NEO430 Processor. https://github.com/stnolting/neo430. 2020.

[55] Oliveira, R.; Jagirdar, A.; Chakraborty, T. J.: 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. ISBN 0-7695-2795-7. pp. 905–910.

[56] OPENCORES.: Wishbone B4: WISHBONE System-on-Chip (SoC) Interconnection Architecture Portable IP Cores.

[57] Park, Y. T.; Sthapit, P.; Pyun, J.-Y.: Smart digital door lock for the home automation. In *TENCON 2009-2009 IEEE Region 10 Conference*. IEEE. 2009. pp. 1–6.

[58] Pavelić, M.; Lončarić, Z.; Vuković, M.; et al.: Internet of Things Cyber Security: Smart Door Lock System. In *2018 International Conference on Smart Systems and Technologies (SST)*. 2018. pp. 227–232.

[59] Piziali, A.: *Functional verification coverage measurement and analysis*. Springer Science & Business Media. 2007.

[60] Reis, G. A.; Chang, J.; Vachharajani, N.; et al.: SWIFT: Software implemented fault tolerance. In *Proceedings of the international symposium on Code generation and optimization*. IEEE Computer Society. 2005. pp. 243–254.

[61] Rollins, N.; Fuller, M.; Wirthlin, M. J.: A comparison of fault-tolerant memories in SRAM-based FPGAs. In *Aerospace Conference, 2010 IEEE*. IEEE. 2010. pp. 1–12.

[62] Rosenberg, S.; Meade, K.: *A practical guide to adopting the universal verification methodology (UVM)*. Cadence Design Systems. 2013.

[63] Rudrakshi, S.; Midasala, V.; Bhavanam, S.: Implementation of FPGA based fault injection Tool (FITO) for testing fault tolerant designs. *IACSIT International Journal of Engineering and Technology*. vol. 4, no. 5. 2012: pp. 522–526.

[64] Sangwine, S.: *Electronic Components and Technology, Third Edition*. Tutorial guides in electronic engineering. CRC Press. 2007. ISBN 9781420007688.

[65] Schweizer, T.; Peterson, D.; Kühn, J. M.; et al.: A Fast and Accurate FPGA-based Fault Injection System. In *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE. 2013. pp. 236–236.

[66] Spear, C.: *SystemVerilog for verification: a guide to learning the testbench language features*. Springer Science & Business Media. 2008.

[67] Sterpone, L.; Aguirre, M.; Tombs, J.; et al.: 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. ISBN 978-3-9810801-3-1. pp. 336–341.

[68] Straka, M.; Kastil, J.; Kotasek, Z.: 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. ISBN 978-0-7695-4494-6. pp. 223–230.

[69] Sutherland, S.; Mills, D.: HDVL+=(HDL & HVL) SystemVerilog 3.1 The Hardware Description AND Verification Language. 2003.

[70] Texas Instruments: MSP430 ultra-low-power sensing & measurement MCUs. February 2020.
Retrieved from: http://www.ti.com/microcontrollers/msp430-ultra-low-power-mcus/overview.html

[71] Trivedi, K.: *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. Wiley. 2016. ISBN 9781119314202.

[72] Verma, G. K.; Tripathi, P.: A digital security system with door lock system using RFID technology. *International Journal of Computer Applications*. vol. 5, no. 11. 2010: pp. 6–8.

[73] Wile, B.; Goss, J.; Roesner, W.: *Comprehensive functional verification: The complete industry cycle*. Morgan Kaufmann. 2005.

[74] Xilinx Inc.: Partial Reconfiguration User Guide.

[75] Xilinx Inc.: ML506 Evaluation Platform User Guide. *UG347 (v3. 1.2)*. 2011.

[76] Xilinx Inc.: FPGA. November 2020.
Retrieved from: http://www.xilinx.com/fpga/index.htm

Published Papers

Paper A

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

Jakub Podivinsky, Ondrej Cekan, Marcela Simkova and Zdenek Kotasek

In: 17th Euromicro Conference on Digital Systems Design. Verona: IEEE Computer Society, 2014, pp. 312-319. ISBN 978-1-4799-5793-4.

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 Arrays* (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).

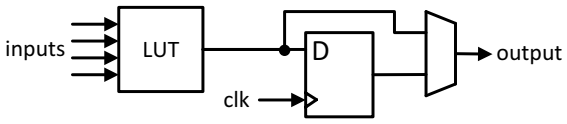


Fig. 1. Structure of *Configurable Logic Blocks*.

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].

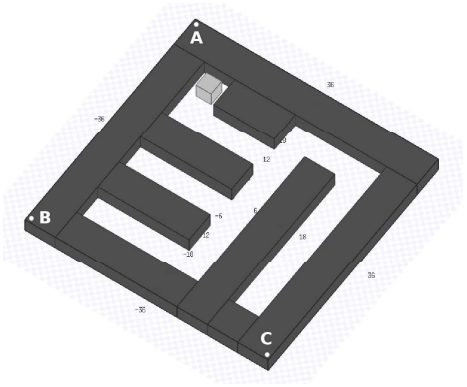


Fig. 2. The robot in a maze in Player/Stage simulation environment.

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.

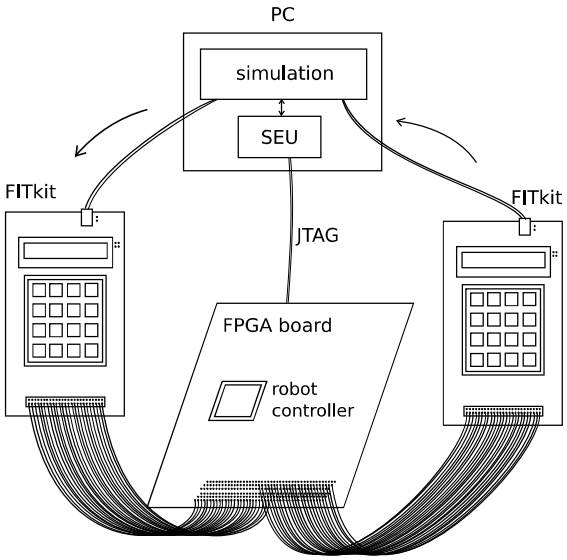


Fig. 3. The platform for testing fault-tolerance methodologies.

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.

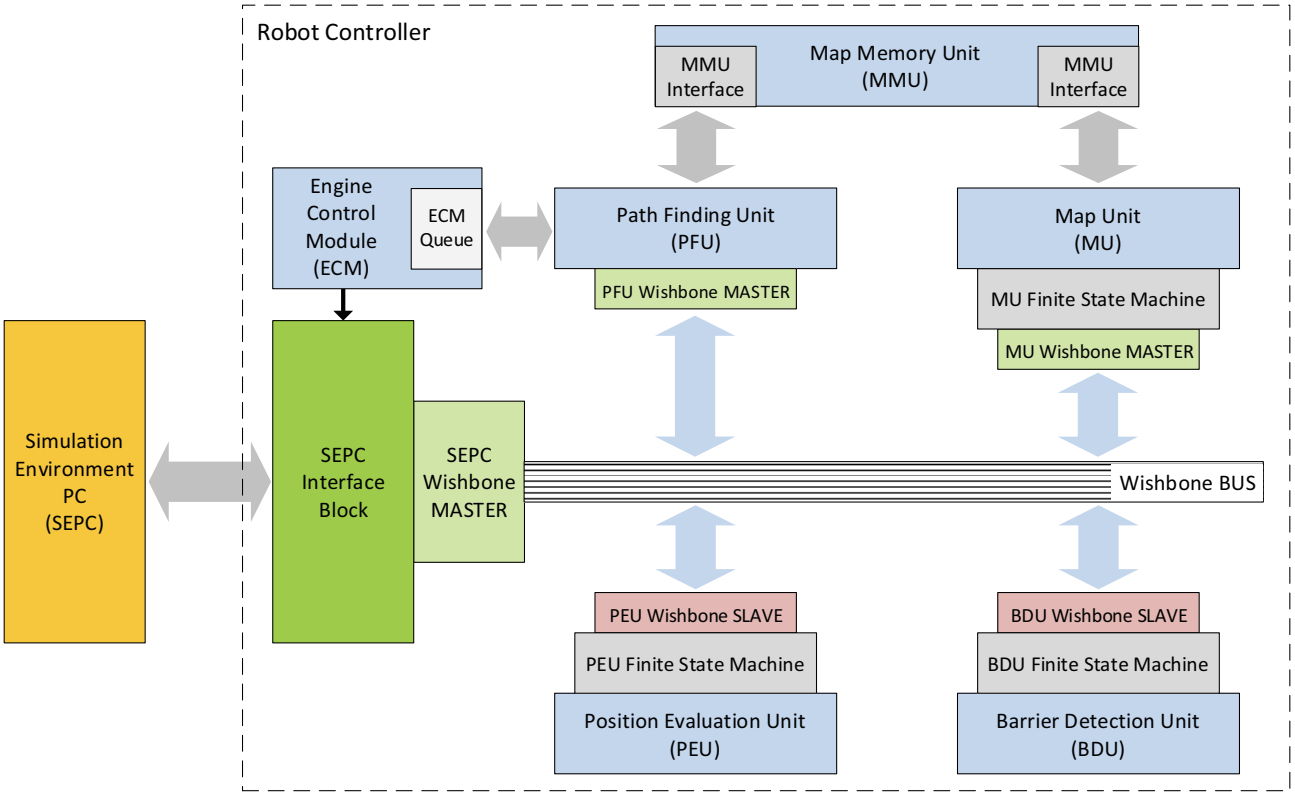


Fig. 4. The block diagram of the robot controller.

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

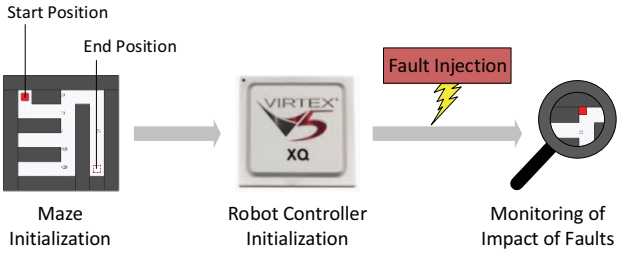


Fig. 5. The flow of one experiment.

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.

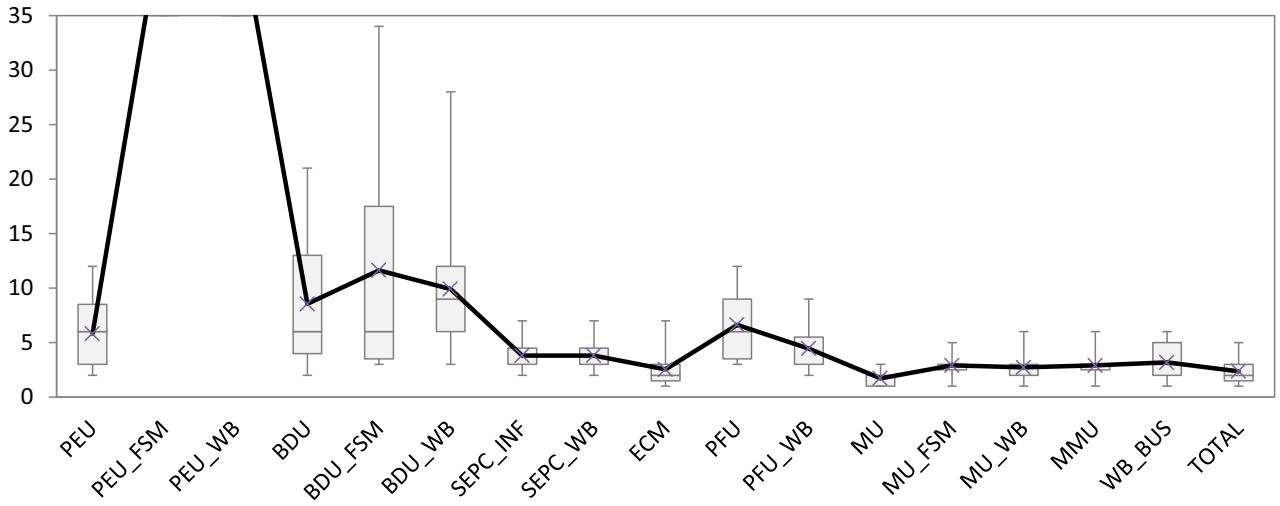


Fig. 6. The quartil graf of the results of experiments.

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.

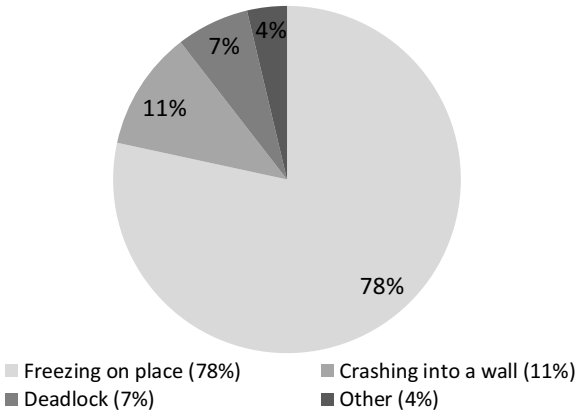


Fig. 7. The chart of typical consequences of injected faults on the mission of the robot.

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

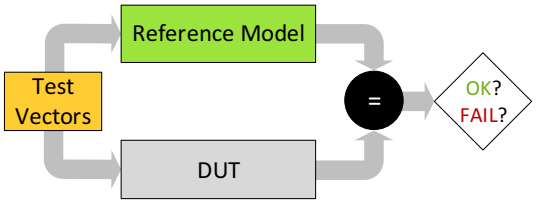


Fig. 8. The main principle of functional verification.

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.

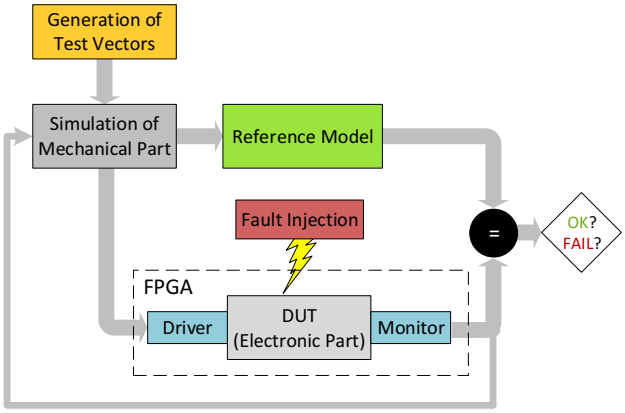


Fig. 9. Functional verification involvement in our platform with the fault injection.

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.

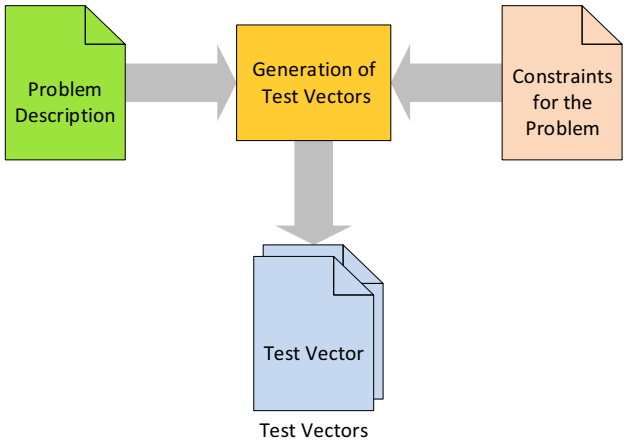


Fig. 10. The principle of the constraint generator.

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.

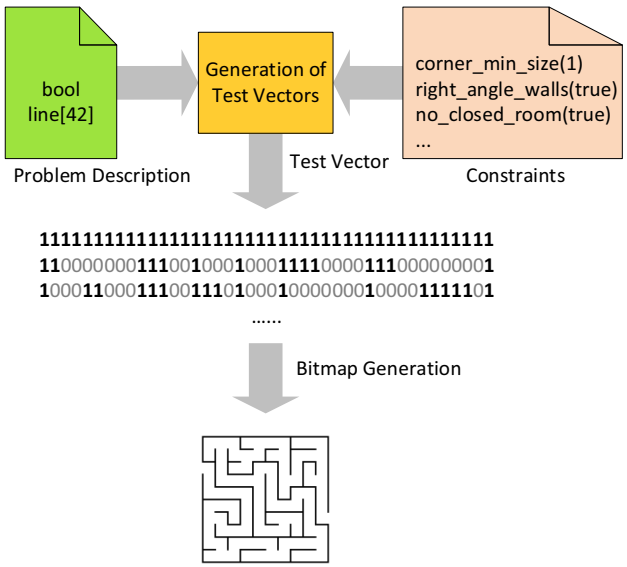


Fig. 11. An example of generating a maze for the robot controller.

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.

Paper B

Functional Verification Based Platform for Evaluating Fault Tolerance Properties

Jakub Podivinsky, Ondrej Cekan, Jakub Lojda, Marcela Zachariasova, Martin Krcma and Zdenek Kotasek

In: Microprocessors and Microsystems. Amsterdam: Elsevier Science, 2017, vol. 52, no. 5, pp. 145-159. ISSN 0141-9331.

Microprocessors and Microsystems 52 (2017) 145–159

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Functional verification based platform for evaluating fault tolerance properties

Jakub Podivinsky*, Ondrej Cekan, Jakub Lojda, Marcela Zachariasova, Martin Krcma, Zdenek Kotasek

Brno University of Technology, Faculty of Information Technology, Centre of Excellence IT4Innovations, Bozetechova 2, 612 66 Brno, Czech Republic

ARTICLE INFO

Article history:
Received 9 January 2017
Revised 15 March 2017
Accepted 4 June 2017
Available online 12 June 2017

Keywords:
Functional verification
Robot controller
Electro-mechanical systems
Fault tolerance
Maze generation

ABSTRACT

The fundamental topic of this article is the interconnection of simulation-based functional verification, which is standardly used for removing design errors from simulated hardware systems, with fault-tolerant mechanisms that serve for hardening electro-mechanical FPGA SRAM-based systems against faults. For this purpose, an evaluation platform that connects these two approaches was designed and tested for one particular casestudy: a robot that moves through a maze (its electronic part is the robot controller and the mechanical part is the robot itself). However, in order to make the evaluation platform generally applicable for various electro-mechanical systems, several subtopics and sub-problems need to solved. For example, the electronic controller can have several representations (hard-coded, processor based, neural-network based) and for each option, extendability of verification environment must be possible. Furthermore, in order to check complex behavior of verified systems, different verification scenarios must be prepared and this is the role of random generators or effective regression tests scenarios. Also, despite the transfer of the controller to the SRAM-based FPGA which was solved together with an injection of artificial faults, many more experiments must be done in order to create a sufficient fault-tolerant methodology that indicates how a general electronic controller can be hardened against faults by different fault-tolerant mechanisms in order to make it reliable enough in the real environment. All these additional topics are presented in this article together with some side experiments that led to their integration into the evaluation platform.

1. Introduction

Digital systems play an important role in our everyday lives. They are widely used in industry, medicine and other safety critical sectors. Not only the loss of a huge amount of money, but also the loss of human lives may occur in case of their failure. The current trend is that the complexity of digital systems is rising, which leads to an increased susceptibility to faults. It is possible to specify two main sources of faults [1]: 1) *Design faults* (bugs) are always the consequence of an incorrect design, an ambiguous specification or misinterpretation of the specification and 2) *Hardware/physical faults* (defects) which arise during manufacturing or system operation.

The approach dealing with *design faults* is called *functional verification* [2] which currently has an irreplaceable position in the development cycle of digital systems. It runs in a simulation (RTL - *Register-Transfer Level* simulators are typically used, like QuestaSim from Mentor Graphics or VCS from Synopsys) and uses sophisticated testbenches which are prepared according to UVM (Universal Verification Methodology) [3,4] which ensures scalability and re-usability. Functional verification checks whether a hardware system satisfies a given specification. The main purpose is to find as many design faults as possible before the system is deployed. The main principle of functional verification is to apply a huge number of input stimuli to the input ports of the verified circuit (DUT - *Device Under Test*) and on the input ports of the reference model. Afterwards, the behavior of DUT and the reference model is compared for these stimuli. The reference model is prepared by a verification engineer in SystemVerilog, C/C++ or other supported language and implements the reference behavior.

Coverage is an important metric in verification. It measures how well input stimuli cover the behavior of DUT and provides feedback that determines when the verification process can be ended. Depending on the coverage criterion considered, the following *coverage* metrics can serve as an example:

* Corresponding author.
E-mail addresses: ipodivinsky@fit.vutbr.cz (J. Podivinsky), icekan@fit.vutbr.cz (O. Cekan), ilojda@fit.vutbr.cz (J. Lojda), zachariasova@fit.vutbr.cz (M. Zachariasova), ikrcma@fit.vutbr.cz (M. Krcma), kotasek@fit.vutbr.cz (Z. Kotasek).

http://dx.doi.org/10.1016/j.micpro.2017.06.004
0141-9331/

- *Code coverage* measures how well input stimuli cover the source code of DUT. Typical code coverage metrics are toggle, statement, branch, condition, expression or FSM coverage.
- *Functional coverage* measures how well input stimuli cover the functional specification of DUT. The user defines the coverage points for the functions to be covered in a verified circuit, e.g.: Did the verification test cover all possible commands or did the simulation trigger a buffer overflow?

Moreover, standard languages, methodologies and libraries were defined for functional verification. The most commonly known ones are the SystemVerilog IEEE language standard [5], Universal Verification Methodology and the open-source UVM library (with all the basic components of verification environments).

Of course, UVM-based functional verification does not guarantee 100% correctness of the system as formal verification does. The reason is that formal verification is based on an exhaustive exploration of the state space of DUT, hence it is potentially able to formally prove its correctness. However, the main disadvantage of this method is a state space explosion for real-world systems and the need to provide formal specifications of the behavior of the system which makes this method often hard to use. On the other hand, UVM-based functional verification is much easier to use and aims at covering properties determined by the specification, not the whole state space. Nevertheless, if these properties are selected accurately, all key aspects of the system are properly verified.

The approaches which deal with *hardware/physical* faults are techniques called *Fault avoidance* or *Fault tolerance* [6]. *Fault avoidance* is mainly obtained by the use of radiation hardened technologies, improved design of storage elements or asynchronous circuits. *Fault tolerance* is the ability of a system to continue performing its correct function even in the presence of unexpected faults. Many fault-tolerant methodologies have been developed inclined, among others, to *Field Programmable Gate Arrays* (FPGAs) and new ones are under investigation [7], because FPGAs are becoming more popular due to their flexibility and reconfigurability. The second reason why so many techniques are inclined to FPGAs is their sensitivity to faults and ability to be reconfigured in the case of fault occurrence. FPGAs are composed of configurable logic blocks [8] which are connected by programmable interconnections. The configuration is stored as a *bitstream* in SRAM memory. The problem is that FPGAs are quite sensitive to faults caused by charged particles [9]. This particle can induce an inversion of a bit in the bitstream and this may lead to a change in its behaviour. This event is called *Single Event Upset* (SEU).

It is important to test and evaluate these techniques. Various approaches to the evaluation of fault tolerance exist and some of them are performed on a theoretical level, for example, a simulation method for SEU emulation is presented in [10]. Another approach is in the use of fault injection directly into the design implemented in FPGA. Special evaluation boards are developed for these purposes, one of them is presented in [11] or [12].

The systems implemented as fault-tolerant very often consist of two parts - an electronic one and a mechanical one. The mechanical part is controlled by its electronic controller. It can be stated that such areas exist in which electro-mechanical applications are implemented as fault-tolerant - aerospace and space applications can serve as an example. Until now, our work was dedicated to verification of fault-tolerant qualities that allow us to check just the resilience of electronic components. However, for electro-mechanical systems, the approach must be different. It must be possible to check what are the reactions of the mechanical component if the functionality of its electronic controller is corrupted by external attacks.

This paper is organized as follows. The goals of our research are described in Section 2. Section 3 introduces three phases of the evaluation process based on our platform. We focus on introducing every phase theoretically and at the same time, we elaborate on making the platform general for various electro-mechanical systems. The first phase is mentioned in Section 4 together with verification environment architecture. Different possibilities for implementing the electronic controller (DUT) are mentioned in Section 5. This can be considered as the first step to generalization. The second step is preparing various verification scenarios for different DUTs and this process is summarized in Section 6. FPGA-based verification environment which is needed for the second phase is presented in Section 7. Principles which are used for checking reactions of the mechanical part in the third phase are introduced in Section 8. For the demonstration of our evaluation platform we created a case-study presented in Section 9 which is supplemented by experiments and their results in Section 10. Section 11 summarizes the results and proposes our plans for future research.

2. Goals of the research

Based on our previous analysis of actual research in the area of fault tolerance methodologies and their evaluation, we have identified the main goals that we would like to focus on in our research of fault-tolerant FPGA-based systems.

- The first point is to develop an *evaluation platform based on FPGA technology for testing fault tolerance techniques*. The basic concepts and the first version of the evaluation platform were presented in our previous work [13]. Based on experiments with our platform we realized the necessity to automate the process of a fault impact evaluation. We found functional verification as an appropriate technique for this purpose.
- The important task is to propose the *process describing the use of the developed evaluation platform for fault tolerance properties improvement* in general electro-mechanical systems. It means that our evaluation platform will be supplemented with a description on how to configure the environment for the selected experimental system, especially how to evaluate fault tolerance properties and search for the possibilities of its improvement.
- As was mentioned above, we need to *take into account the mechanical part* which is usually driven by an electronic controller in real systems. Therefore, the verification environment should take into account also the operation of the mechanical part when evaluating the correctness of operations.

The following sections describe our progress in achieving these goals. Firstly, the basic concept of the evaluation process is described and is divided into three phases. Each of these phases needs a specific verification environment with a specific configuration, so the evaluation platform is described on a theoretical level for every phase separately. The evaluation platform is demonstrated on one case-study: a robot searching a path through a maze and its electronic controller.

3. Basic concept of the evaluation process

The proposed process of the fault impact evaluation, which is shown in Fig. 1, is divided into three phases. In the first phase, we use the simulation-based functional verification where the VHDL description of the electronic controller is used as the DUT. In this phase, the correctness of the electronic controller design is evaluated. The main output of the first phase is a test on whether the electronic controller works correctly according to the specification. It is important because we have to ensure that the electronic controller does not contain any functional errors in the implementation. It is also important to point out that in this phase we acquire

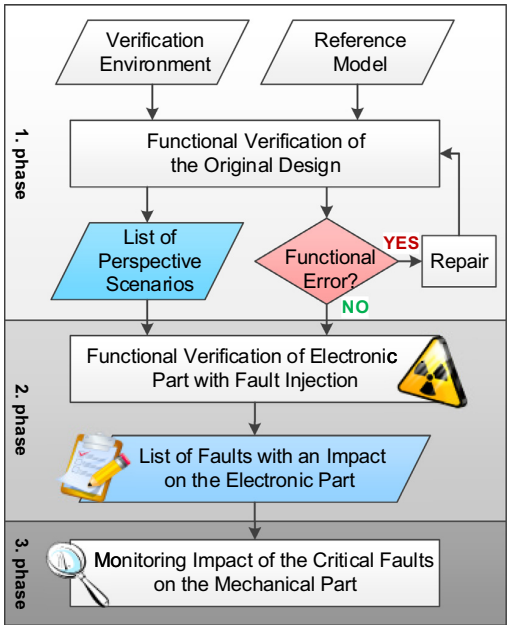


Fig. 1. The flow of phases in the FT evaluation systems verification.

a set of verification scenarios that will also be used in the subsequent phase.

The second phase consists of the verification of the electronic controller implemented into FPGA with the scenarios obtained during the previous phase, but in addition, artificial faults are injected into FPGAs using implemented fault injector.

The analysis of the faults which corrupted the mechanical part is the goal of the third phase. The outputs of the second phase are previously verified verification scenarios supplemented by information about injected faults and its impact on the electronic part. The injected faults are divided into two categories, faults with no impact on electronic part and faults which cause mismatches on the output of the electronic part. Various strategies of fault injection may be used in this phase (e.g. one fault for one verification run, multiple faults in the same functional unit or multiple faults in different functional units).

The development of the verification environment and a reference model for the electronic control unit (the electronic controller) are the first steps towards this whole three-phase process. Both of these activities are described in detail in this paper. The second step is to implement the DUT into the FPGA and achieve interconnection with the simulation environment of the mechanical part. The architecture of the verification environment with the electronic controller implemented in the FPGA is also presented in this paper.

4. The first phase - verification environment architecture

The verification environment architecture, its basic components and used techniques are described in this section. First, the UVM based verification environment for one verification scenario (one model of real environment, one task for electro-mechanical system) is presented, which forms the core of an extended verification environment for multiple verification scenarios evaluation.

The verification environment for the electronic controller is designed according to UVM, so that it corresponds with current trends and requirements. The basic architecture of the verification environment with main components is shown in Fig. 2. It should be noted that the verification environment is connected with the

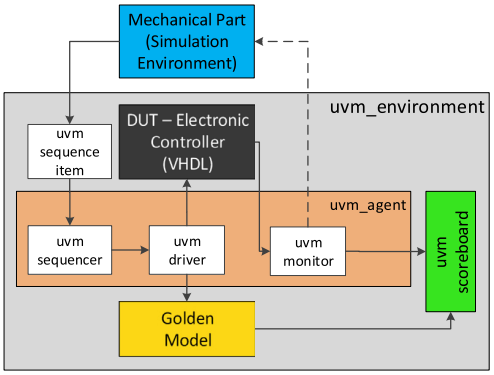


Fig. 2. General verification environment for a single verification scenario.

mechanical part, especially the simulation of the mechanical part. The mechanical part in real environment is controlled by the outputs of the electronic controller (DUT) while the outputs of the mechanical part (information from sensors) are inputs for the electronic controller. The information whether the DUT satisfies (or does not satisfy) specification and coverage report for the verified scenario are the outputs of the verification environment. These are the components of the system together with their description:

- *The electronic controller* under a verification which can be implemented into FPGA in the next phase. Several approaches how to implement DUT exist, they will be described in Section 5.
- *The golden (reference) model* implemented in C/C++ according to the same specification as the electronic controller performs the same operations as DUT.
- *The sequence* is the component which receives data from sensors placed in the mechanical part. Received data are transformed to the inputs of the verification environment.
- *The driver* sends input values (data from sensors) to reference model and the DUT (electronic controller).
- *The monitor* reads the outputs from the DUT (instructions for mechanical part) and forwards them to the scoreboard and to the mechanical part which operates according to these values.
- *The scoreboard* compares the outputs of the monitor and reference model for equality and checks mismatches on the outputs. The detected mismatch shows that there are differences between the DUT and the reference model outputs.

The presented verification environment is not able to evaluate multiple verification scenarios automatically so it must have been extended by components such as verification scenarios random generator or a simulator of the mechanical part. All the final components, their inputs, outputs and connections are shown in Fig. 3 and their description is as follows:

- *The verification scenarios generator* allows us to generate a sufficient number of verification scenarios with respect to specified parameters in order to achieve the required coverage. In our work, we use a verification scenario generator based on our universal generating principle described in Section 6.
- *The mechanical part simulation* replaces the real mechanical part in case we do not have a real one.
- *The UVM verification environment* is the core of the extended evaluation.
- *Store the verification scenario* allows us to use it in the second phase which utilizes a fault injector (Fig. 1). A certain part of the stored verification scenario is also a report about the coverage which was obtained by this scenario.

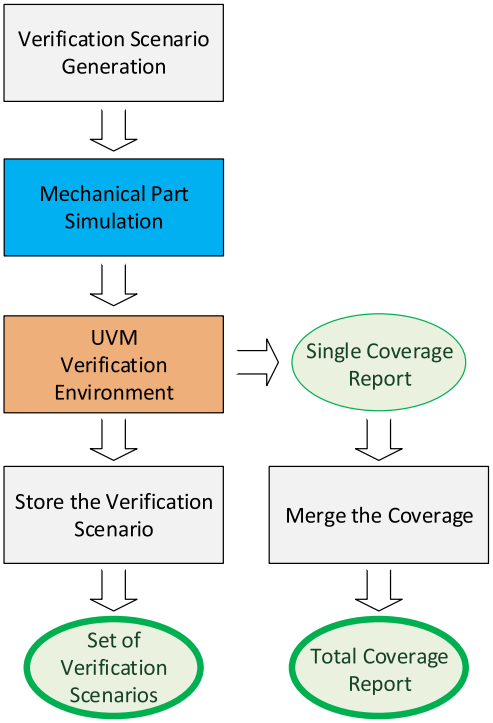


Fig. 3. Extension of the verification environment for multiple scenarios evaluation.

- *Merge the coverage* achieved by the single verification scenario is important in order to obtain a final coverage report gained by stored sets of verification scenarios.

Fig. 3 also shows the outputs of the first phase of the fault impact evaluation process presented in Section 3 which are *Set of Verification Scenarios* and obtained *Total Coverage Report*.

5. Electronic controller - implementation alternatives

As mentioned above, an electronic controller can be implemented in several ways, each having different advantages and disadvantages. Several key features exist which must be taken into account. From one point of view it can be flexibility, scalability and extensibility. Another point of view may be fault tolerance, durability and maintainability. From the economic point of view it is the cost, power consumption and time to market (the difficulty of development).

In this paper we mention three possibilities of controller implementation - a processor, a hard-coded controller and a controller based on neural networks.

5.1. The use of processor as electronic controller of mechanical part

The usage of a processor as the controller is the most universal and flexible way of controlling the system. A lot of different classes and types of processors are in the market and it is possible to find a suitable device for the application. The main advantages are the flexibility of the solution due to the possibility of changing the software, short time to market and the low cost (if the processor is selected properly).

The usage of a processor also offers different ways to build a fault-tolerant controller. The processor can be secured using hardware redundancy - more processors can be used to build a robust system, or the processor can be secured on the level of its inner components. Another way is to secure the application at the software level using time and space redundancies [14].

5.2. Hard coded electronic controller

In this case, the controller is composed of components described using a hardware description language (HDL). This solution is less flexible than the previous one, but it can be more efficient. The hard coded controller can be designed directly to the application in order to utilize resources effectively which can lead to high performance and/or low power consumption. High performance can be reached due to the possibility of utilizing high parallelism and application-designed specific computing units. The effective computing algorithms can then be implemented. The low power consumption is related to the effective usage of resources. Unneeded components can also be omitted (unlike in case of processor).

Several ways on how to ensure fault tolerance exist in this case. Replicating the whole system is the well known way, however in the case of hardcoding, it is easier to construct the redundancy on the level of inner components - the computing units, registers, multiplexers and other components the system is composed of. This makes it possible to secure only selected components, therefore, making the fault-tolerant design more effective by avoiding the redundancy where it is not needed. The component specific techniques can be deployed as well. It is also possible to use more specific techniques like securing using partial TMR [15]. It is also possible to use different coding schemes for securing the data - for example, parity codes, BCH codes and others error detection/correction codes. If the FPGA is used, the reconfiguration can offer suitable tools for fault-tolerant techniques implementation [16].

This solution can generally be very effective in relation to performance, power consumption and fault tolerance. However, the time to market is longer and costs can be higher than in case of using the processors. Nevertheless, this solution can be suitable for tasks with specific recommendations.

5.3. The use of neural networks as fault-tolerant electronic controller

Artificial neural networks are one of the traditional disciplines in the field of artificial intelligence and softcomputing. Even though the neural networks were almost forgotten for some period of time, at present their popularity is continually growing. The main advantage of neural networks is their capability of learning and memorizing the data which make them suitable for different tasks such as classification, function approximation, timeseries prediction, etc. They are widely used, especially in the deep form which disposes of very interesting properties for the tasks such as image recognition. The neural networks are also used in control tasks [17], the control of mobile robots included [18,19]. Therefore, we are going to experiment with a neural-network-based controller as well.

The neural networks dispose of interesting fault-tolerant properties which are used with different techniques. It is possible to modify a learning algorithm to train the network, not only to perform the task, but to be fault-tolerant as well [20]. It is also possible to retrain the network after a fault occurs [21]. Other approaches use adding an extra redundancy into the network [22].

In our research we are dealing with the specific FPGA resource saving implementation of neural networks called FPNN [23]. We especially deal with its neural network approximation capabilities [24] and also with designing fault-tolerant techniques. In this manner, we designed the fault-tolerant type of FPNNs [25].

6. Verification scenarios generation

A very important part in the verification process is preparing and applying input stimuli. Just by using a significant number of diverse inputs, it is possible to cover most of the behavior of DUT and thus to be sure that DUT behaves as specified. When we consider the standard approach, stimuli are represented by transactions in the UVM-based verification environments. Transaction stands for a setting of input ports of DUT in one clock cycle. So, for example, when DUT has three input ports A (8 bits), B (16 bits), C (1 bit), the transaction can be represented by a triplet of values {8'h87,16'h11FF,1'b0} which is applied on these input ports on the rising edge of synchronization clock signal (or other specified synchronization event).

Two approaches for preparing input stimuli exist:

- Directed stimuli - a verification engineer prepares transactions manually. This approach is recommended at the beginning of the verification process for checking basic scenarios.
- Pseudo-randomly generated stimuli - transactions are generated by a random generator according to the given constraints (restrictions for values of inputs). That is the reason why this approach is often called constraint-based generation. An example of constraining an input is the following. Let us have an input port A (8 bits), then a constraint can restrict the generated values for this port in the range of 0–100 (from the possible range 0–255).

In the following text, we will introduce the pseudo-random generation of stimuli using our proprietary generator. This generator is unique in the way that it can be used for different scenarios and works with two formal models that significantly improves its performance. Afterwards, we will show how an evolutionary algorithm can be used for preparing optimal regression suites for different systems. The motivation for using regression suites is simple. If there is a need for running verification of DUT repeatedly - just to check that everything still works or after minor changes to DUT (like small optimizations), it is worth using an optimized small suite of tests (input stimuli) with a high level of DUT coverage rather than start verification from scratch every time (it is very time consuming).

The same applies to the evaluation platform. Depending on the electronic controller that is verified, we can either utilize direct stimuli, pseudo-randomly generated stimuli or an effective regression test suite (especially when we want to evaluate fault injection in the second and third phases of our evaluation process introduced in Section 3).

6.1. Pseudo-random generation of verification scenarios

Pseudo-random generation is also an integral part of our research. We are creating the solution for the stimuli generation which has to be versatile for our evaluation platform. We need a different stimuli for a different system that we verify. The original way of the generation that we presented in our previous paper [26] is generalized by using probabilistic context-free grammars [27] that we found as a suitable means for the stimuli generation. We gained universal description of verification scenarios (stimuli) while maintaining our originally designed architecture. We benefit from grammar systems which allow us to generate a defined language. This language will form stimuli for a given system. We are also able to control the generation process through the defined probabilities in the grammar. In our architecture, we use constraints which allow us to modify stimuli during their generation and verification.

In our previous research [26], we proposed our solution of the generator which was based on our own proprietary description of the stimuli. Although the designed architecture was general, the description of the stimuli was not versatile for any system. A specific dependencies in stimuli creation had to be implemented when a new system should be supported. About 13 types of the constraints had to be implemented for valid generating of assembly programs for a RISC processor. The set of the constraints also increased with the inclusion of support for other systems. For these reasons, we were looking for another suitable solution that will be built on any mathematical apparatus. As the best solution, we have found the use of the grammatical system [28] from the theoretical computer science.

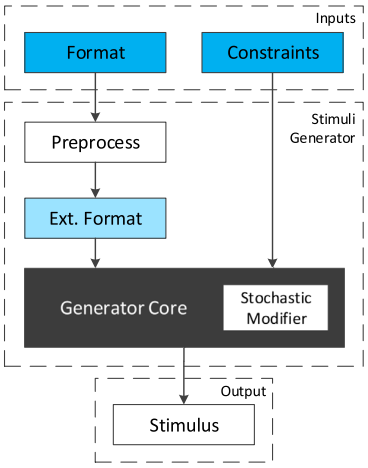


Fig. 4. The detailed architecture for a probabilistic context-free grammar based stimuli generation.

In our actual research, we use the probabilistic context-free grammars. The probabilistic context-free grammar is the quintuplet:

$G = (N,T,R,S,P)$; where:

N is a finite set of non-terminal symbols.
T is a finite set of terminal symbols, applies $N \cap T = 0$.
R is a finite set of rewrite rules with form $A \rightarrow \alpha$, where $A \in N$ a $\alpha \in (N \cup T)^*$.
S is starting non-terminal.
P is a finite set of probabilities for rewrite rules.

The probabilistic context-free grammar looks like a common context-free grammar, but it has the special set of probabilities which represent how likely a rewrite rule of the grammar is applied. It allow us to define the format of the stimuli through the formal description provided by grammars. We benefit from the probability definition for the rules, because it allows us to control the application of the rules in the grammar and gives us the possibility to influence the stimuli creation. The probabilities are defined statically for the grammar definition which is not optimal for stimuli creation. Therefore, we extended this description with a special logic that is described through the constraints which allow us to dynamically change defined probabilities.

The constraints represent restrictions and limitations for the application of the rewrite rules of the grammar and their use will change defined probabilities for specific rules during generation process such that the result is a valid stimulus. After application of any rule, the eligible constraints are performed and the new probabilities are set anywhere in the grammar.

The architecture of the generation is shown in Fig. 4. The probabilistic context-free grammar is defined in the input structure called *Format*, while special rules for restricting the grammar are defined in the *Constraints* structure. For an easier description of

the input structure, we use certain elements of the library Jinja2 [29] which is a templating system for the Python programming language [30]. The templating system allows us to define cycles, branches and other special macros in the structure description. The preprocessing (*Preprocess*) expands these special macros and a complete description of stimuli (*Ext. Format*) is obtained. The extended format suffices to be generated only when the original format is changed, otherwise, the generator works directly with this extended format and is not generated any more.

The Ext. Format and the Constraints are processed by the core of the generator. It performs the application of the rules from the starting non-terminal with leftmost derivations. After the derivation of any rule is performed, the constraints for the relevant rule are triggered and thus the new probabilities are set for the given rules. Probabilities are adjusted using a special block *Stochastic Modifier*. Through this, the next derivation valid for the given stimulus is directed and prescribed constraints will be respected for generating a valid one.

6.2. Regression test suites optimization using evolutionary algorithms

Our optimization technique was firstly introduced in paper [31] and later on extended in paper [32]. It works off-line and takes a suite of input stimuli that were evaluated in the process of UVM-based functional verification and optimizes them automatically using the genetic algorithm (one of the evolutionary algorithms). The aim of optimization and the main contributions of this technique are:

1. *Eliminating the redundancy* in the original suite of stimuli so the optimized suite is smaller and therefore, it will be running faster in simulation.
2. *Preserving the same level of coverage* (the term coverage was explained in Section 1) as was achieved by the original (unoptimized) suite of stimuli. It guarantees that the behavior of DUT will be checked properly.
3. *Reusing already created verification environment for running regressions* after minor changes in DUT are made so it is not necessary to utilize a separate approach for regression testing.

Redundancy in the original suite of stimuli is caused by their randomness. In the first phase of verification when DUT is firstly created, redundancy in stimuli is often a beneficial factor [33], because key properties of the system have a chance to be checked more times (for example, we want to check multiplication, but it is good to check it repeatedly with different data) and it is almost always wanted. But after this phase, for example during regression testing, the redundancy is not needed anymore (it is enough to check every key property of the system just once), so it is good to have regression stimuli that are effectively reduced from the original suite and thus are running faster (in order to spend less time by running regressions). That is the reason why we decided to apply our optimization after the first phase of verification with the aim to reduce redundancy.

The survey of the proposed optimization technique follows; the process of optimization is divided into several steps:

1. Run the UVM-based functional verification for a selected DUV and collect stimuli until the threshold in coverage is reached.
2. Optimize stimuli by the proposed technique.
3. Use the optimized suite everytime regression testing is needed. It is even possible to use existing verification environment for running regressions.

The optimization technique incorporates the genetic algorithm as the main optimization tool. As described in [34], the genetic algorithm employs a population of candidate solutions that are evolved through several generations. The quality of candidate solutions is determined by the *fitness function*. According to the fitness, the best solutions are selected and serve as parents for the next generation. Offsprings are created by mutation and crossover genetic operators. If the algorithm works well, the average fitness of the population is rising because profitable parts of the search space are explored. At the same time, genetic operators ensure diversity, so the algorithm is resilient to the problem of local optimum.

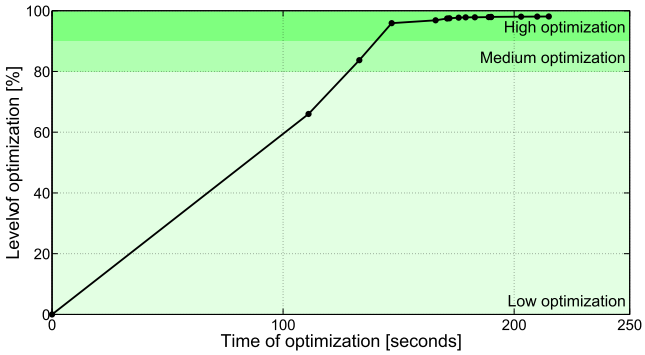


Fig. 5. The dependency between the optimization runtime and the level of optimization.

The main result is that the presented optimization technique was able to reduce the number of stimuli to the *0.522% of the original size* and the resulting coverage statistics remained at the same level as was achieved by the original suite. What is more important, the simulation runtime of the optimized regression suite was much shorter and was reduced by *98.1%*. See more details in paper [32].

Fig. 5 demonstrates the dependency between the achieved level of optimization of the regression suite (the y axis in the graph) and the time of optimization (the x axis in the graph).

It can be seen that the longer the optimization runs, the shorter is the simulation runtime for regression testing.

7. The second phase - evaluation platform architecture

The second phase of the evaluation process is the functional verification of the design implemented into the FPGA. Moreover, the fault injection into the FPGA is performed in this phase. The experimental platform which is composed of a few components running on a computer or on an FPGA evaluation board was designed for these purposes:

1. software part of verification environment for the electronic controller running on a computer,
2. software simulation environment for mechanical part simulation running on a computer,
3. electronic controller implemented into FPGA, and
4. external fault injector [35] running on a computer which allows us to simulate real faults in the FPGA.

The overall experimental platform interconnection is shown in Fig. 6. The connection between a computer and an FPGA is realized by JTAG and Ethernet. JTAG interface is used for FPGA programming and the software and hardware part of verification environment are connected through Ethernet. The fault injector also uses JTAG for placing faults into the FPGA configuration memory. The description of the architecture of the verification environment and of the fault injection process follows below.

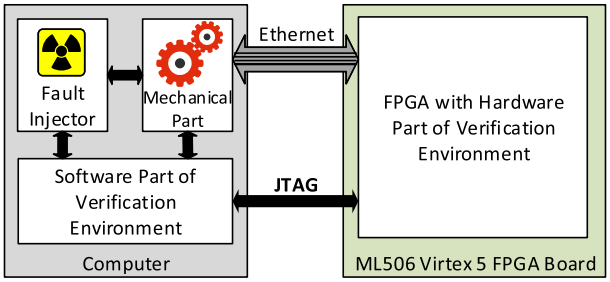


Fig. 6. The structure of the experimental platform.

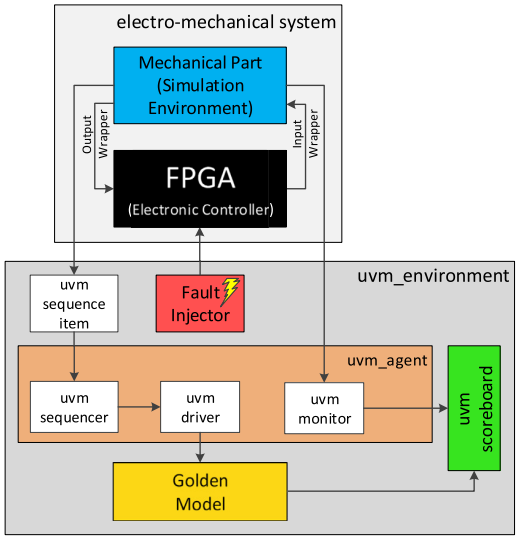


Fig. 7. The general architecture of the FPGA-based verification environment.

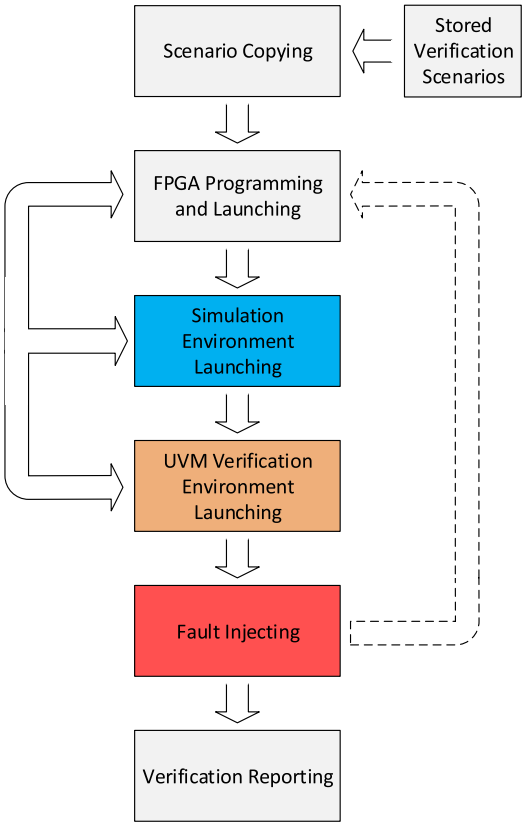


Fig. 8. FPGA-based verification environment for multiple evaluation with fault injection.

7.1. Architecture of FPGA-based verification environment

For these purposes, the FPGA-based verification environment, which is displayed in Fig. 7, is derived from the version created in the first phase. The architecture of the verification environment is divided into two parts. The first part is the simulation environment of a mechanical part which is controlled by the electronic controller implemented into the FPGA. The communication between the software and the hardware parts is accomplished using a proprietary interface (more details about the communication are provided in the subsequent subsections). This part operates autonomously, and the electronic controller receives information from the sensors which is produced by the simulation environment and sends them to the FPGA through Output Wrapper. On the other hand, speed and direction of movement are sent through Input Wrapper from the electronic controller implemented in the FPGA to the mechanical part in a simulation.

The second part is the UVM-based verification environment which operates as an observer without direct intervention to data transfers between the electronic controller and a mechanical part in a simulation environment. The verification environment just checks the correctness of transferred data which are resent to the verification environment as can be seen in Fig. 7. Information from the sensors is received in the Sequence component where they are transformed to transactions and transferred to the Golden Model which produces reference output data. Instructions for mechanical part are received in the Monitor component and sent to the Scoreboard component. The Scoreboard compares received data with reference data obtained from the Golden Model.

Both parts are synchronized by signals sent from the Sequence and Monitor components to the mechanical part simulation environment. These signals indicate that the verification environment is ready to observe operations of mechanical part.

The presented FPGA-based verification environment evaluates only one verification scenario, but automated evaluation of multiple verification scenarios with a fault injection is needed which is shown in Fig. 8. The second phase eliminates the need for verification scenarios generation because scenarios pregenerated and verified in the first phase are used. Conversely, there are new steps as a consequence of implementing electronic controller into the FPGA and the creation of an autonomous connection between the FPGA and the mechanical part. The first necessary step is programming the FPGA through the JTAG interface which must be done before each verification run. This step ensures that the correct functionality of the electronic controller is verified and is without faults.

The next step is launching the mechanical part into a simulation and verification environment which enables signals to the simulation environment when it is ready to start monitoring. Then, the mechanical part starts its operation which is the proper time for fault injection. It should be noted that fault injection proceeds according to the selected strategy. Our fault injector allows us to inject faults into specified functional units which can be advantageously used. For example, we can inject single faults during one verification run into the specified functional unit, multiple faults into the specified functional unit or inject multiple faults into multiple functional units. After fault injection, the verification run is finished or timeout has expired and then results of the verification are recorded into the verification report.

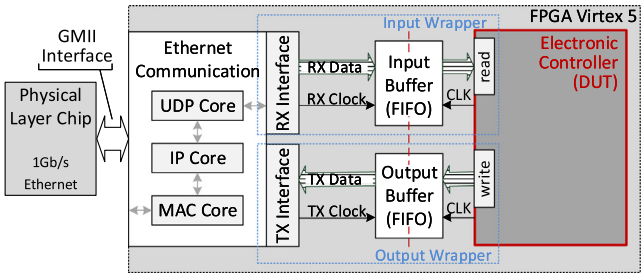


Fig. 9. The architecture of communication between SW and HW part.

During fault injection, it is worth utilizing effective regression test suites for experiments. There are two reasons for this.

The first reason is that regressions contain stimuli that achieve a high level of total coverage (the coverage of DUT behavior is very high) and, therefore, after fault injection, we can be sure that such stimuli/tests discover all potential problems combined with injected faults regarding functionality. To be precise, it is guaranteed for DUT that if no artificial faults are injected, it always behaves correctly for regression tests. After a fault is injected and an error or errors occur, it only means that the fault caused a critical problem inside the system. The result of this phase is a list of faults and their locations, which caused discrepancy on the outputs of DUT for a specific regression stimulus/test. Furthermore, it is possible to run an advanced analysis and harden some critical parts of DUT by fault-tolerant techniques, and to check the results for the selected regression tests again, until we are satisfied with the result.

The second reason is that regressions are usually running much faster in RTL simulation as they represent significantly filtered pseudo-randomly generated stimuli. And when we consider running verification after every single injected fault, the time-saving is significant.

7.2. Communication between software and hardware parts

Communication between the software and hardware parts of verification environment can be accomplished in various ways. One way is the use of some proprietary interface, or another way is to use one of the standardized interfaces which are used in verification based on emulation in FPGAs [36]. One of them is Standard Co-Emulation Modelling Interface (SCE-MI) [37] proposed by the Accellera organization. Thanks to the standard SCE-MI interface, users are able to reuse the existing hardware cores in FPGA in order to develop their system prototype.

In our case we chose to use Ethernet interface supplemented with our proprietary protocol based on UDP. The communication between the electronic controller implemented in the FPGA (hardware part) and the mechanical part in a simulation environment (software part) is accomplished through Input and Output Wrapper. We chose the ML506 development board [38] equipped with Xillinx Virtex 5 FPGA as the hardware platform. This board offers various peripherals and some of them can provide communication with a PC (e.g. PCIe, UART, USB or Ethernet). The chip implementing the Ethernet physical layer is connected to the FPGA and to the user design which implements higher layers of the Ethernet protocol stack that can communicate with this chip. However, we do not implement a full Ethernet protocol stack, instead we use an existing implementation presented in [39].

Fig. 9 shows the architecture of the communication layer. Although we use an existing implementation of Ethernet communication, we must solve a problem with different clock signals on receive (RX) and transmit (TX) interfaces. These clock signals are generated by a physical layer chip and the designer is not able to modify the frequency and the phase offset. We use a FIFO memory as an input and output buffer with different writing and reading clock signals. This not only solves the problem with clock domain crossing, but also the problem with data storing before their processing. Data received from the Ethernet are buffered in the input buffer and data ready to be sent are buffered in the output buffer. We use FIFO as the interface of the DUT which allow us to exchange a communication layer with another one which uses FIFO buffers.

7.3. Evaluation of reliability by fault injection

The simulation of the effects of faults in the FPGA can be done by a direct change of the configuration bitstream which is loaded into the configuration memory. For this purpose, we developed a fault injector [35] which allows us to prepare the bitstream for our FPGA and also 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 similarly influenced by a real fault which strikes the hardware architecture of the FPGA in a real environment.

The injector is based on the SEU generation outside of the FPGA (in PC), so it is not targeted to a specific FPGA board (testing was performed on the ML506 card with the Virtex 5 FPGA technology). The original and the modified bitstreams are transported through the JTAG interface. The process of the SEU generation is divided into four steps: 1) specifying the location of the fault injection, 2) reading the related part of the configuration bitstream, 3) the SEU generation (i.e. the inversion of the specified bit of the bitstream), and 4) applying the bitstream using *Partial Dynamic Reconfiguration* (PDR) without stopping the FPGA.

The implemented fault injector is able to inject a fault into a specified bit of bitstream. If we are able to find a relation between bits of bitstream and functional units, we can inject faults into the specified functional unit. For this purpose, the analysis of FPGA can be done by the RapidSmith [40] tool. This tool identifies the bits of bitstream which are related to a specified area in the FPGA. Functional units placement in the FPGA is done by the PlanAhead [41] tool, so we know where each of the functional units is placed. This process allows us to inject faults into specified functional units during our experiments. Unfortunately, the process actually finds only the bits of the bitstream corresponding with Look-up tables (LUTs).

8. The third phase - mechanical part reactions

The task for the third phase is to check reactions of the mechanical part and there are several methods on how to do it. We can look either at the mechanical part or on its simulation and check whether it works. These methods require an observer, which can be a person or a digital camera supplemented with some kind of an algorithmic image processing.

Another way is to use a type of information which represents the state of mechanical part. In modern electro-mechanical systems, there are lots of sensors placed on a mechanical part which informs us about its state. These sensors are usually used as an input for the electronic controller, and we can use the information from these sensors for monitoring the behaviour of the mechanical part. For example, if we are monitoring the movement of an autonomously driven car, we can observe its behaviour by monitoring the GPS location and a speed sensor.

Our evaluation platform is based on functional verification which only observes the electronic outputs of verified circuits implemented into FPGA. These electronic outputs are compared with the outputs of the reference model which operates as a part of the

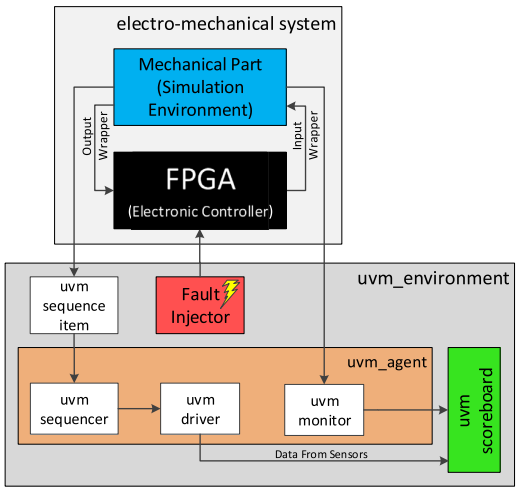


Fig. 10. Checking of the mechanical part behaviour by functional verification.

verification environment according to the same specification as the verified circuit. Input for the reference model is the information from sensors produced by the mechanical part which is the same as for the verified circuit. Values from sensors received by the verification environment can be used not only as inputs for the reference model, but also as inputs for monitoring behaviour of the mechanical part. The modified verification environment is shown in Fig. 10 where the reference model is missing, information from sensors is routed directly to the scoreboard which can monitor the operation of the mechanical part. It means that scoreboard implements functions for checking reactions of mechanical part based on information from sensors.

9. Casestudy: robot searching a path through a maze and its electronic controller

General principles used in our platform were presented in previous sections and our experimental electro-mechanical system will serve as a demonstration example in the following text. As an experimental system we chose a robot which searches for a path through the maze. The mechanical part is a robot in the maze and the electronic part is its electronic controller. Unfortunately, we do not have a real robot device, so we used simulation environment for a robot and its environment. We use Player/Stage [42] simulation environment which is freely available and offers lots of possibilities for robot configuration.

Our robot is a simple cubical robot which goes through the maze. The robot is equipped with a few sensors, three sensors which inform us about distances from three control points placed at fixed positions in the robot environment. They are used for determining its location (inspired by Global Positioning System). Four sensors are located on the sides of the cubical robot and inform us about the distances from barriers in the robot surrounding. The operation of the robot is driven by two inputs - speed of robot in x-axis and y-axis directions (*x_speed, y_speed*).

The robot controller, whose structure is shown in Fig. 11 consists of various blocks, their function is described in [43]. The controller is connected to the PC on which the robot simulation environment (SEPC) runs via the Interface Block. Through this block, data from the simulation are received, and in the opposite direction, instructions defining the required movement of the robot are sent back. The central block of the robot controller is a bus through which communication between blocks is accomplished. The Position Evaluation Unit (PEU) calculates the positions 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 the information about the distance to the surrounding barriers. The map updating provided by the Map Unit (MU) is based on the information about the positions of the robot and the barriers vector. The Map Memory Unit (MMU) stores the information about an up-to-date map. The Path Finding Unit (PFU) implements a simple iteration algorithm for finding a path through the maze. The mechanical parts of the robot are driven by setting the speed in the required direction of the movement by the Engine Control Module (ECM). The communication of functional units with a bus is accomplished through the bus wrapper (FU_WB) and controlled by the finite state machine (FU_FSM).

Our electro mechanical system was introduced, but verification environments for three phases of verification process must also be proposed. These verification environments are implemented according to principles presented above.

9.1. Simulation based verification environment (the first phase)

The general verification environment for the first phase shown in Fig. 2 is a standard UVM-based functional verification environment which is usually created during electronic systems development. In our example, the electronic controller is a robot controller and the mechanical part is a robot going through a maze. The reference model is implemented with respect to the same specification as the robot controller, the inputs are the information from sensors and outputs are speed values in x-axis and y-axis directions. These speeds are compared with the speed values received from the robot controller. Naturally, the compared speed values must be the same. The verification environment and reference model are presented in more details in [44].

The verification environment is able to process multiple verification scenarios (see 3), while one verification scenario is, in the case of robot, represented by a maze and start and goal positions of the mission. Therefore, stimuli generation in this casestudy means mazes generation.

9.2. Maze generation

Maze generation is a well known and explored area for which a considerable number of algorithms generate simple or sophisticated mazes [45]. The vast majority of algorithms operate in a two-dimensional space, keeping their current state and can constantly change cell values of a maze in time. These algorithms are highly unsuitable for our proposed architecture of the universal generation, because the output of the generator (a line of the maze) cannot be determined in one step, therefore, it is determined gradually by many factors and dependencies between different cells of the maze. However, an algorithm exists such that is based on a binary tree and a particular line of the maze can be determined only from the previous one. This principle is completely satisfactory for our generator and the output maze is fully sufficient for our needs.

The basic principle of the binary tree algorithm is shown in Fig. 12. It starts from the basic matrix of the maze (a) in which some cells are tightly specified - either the corner or the wall. We represent the corridors by zeros and the walls by ones. Cells marked with a question mark represent areas that can take the value of 0 or 1, thus the corridor or the wall. In order to maintain the continuity from any corner of the maze to another, it is necessary to perform a modification of the basic matrix of the maze so that each two adjacent sides of the maze must contain the corridor over its entire dimension (b). In our case, we chose this corridor to the northern and the western side of the maze. The final and most critical task is to determine cells A, B, C, D which allows us to have the maximal continuous maze (c).

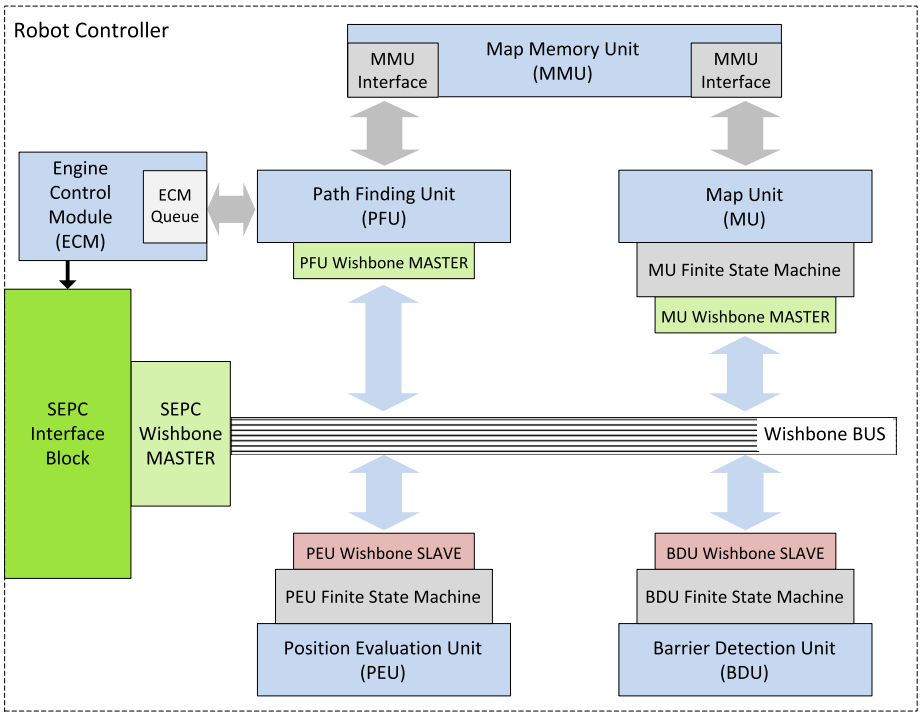


Fig. 11. The block diagram of the robot controller.

Fig. 12. The demonstration of a conversion of the basic matrix of the maze for needs of the generator.

The original description of the algorithm [45] divides cells of the maze in a line into groups of corridors bordered by walls. For each group, an algorithm determines one entrance, either in the northern or western part of the border. This ensures the passage from the northern part of the maze to the south and the same applies for the passage from the west to the east. We transferred this principle into one line dependency in the maze and the result is the following dependence. If cell A, respectively C, was randomly selected for the corner in Fig. 12.b, then cell B, respectively D, will be a wall and vice versa.

9.3. FPGA-based verification environment (the second phase)

The second phase of our verification process is based on the FPGA-based verification environment. The environment is shown in Fig. 13. It can be seen that the electronic controller is represented by the robot controller implemented into FPGA and the mechanical part is represented by a robot in a maze simulated in the Player/Stage environment for robot simulation. The verification environment just observes the communication between the robot in the maze and its robot controller.

Multiple verification scenarios evaluation is done with respect to the process shown in Fig. 8. Stored verification scenarios are mazes with start and goal positions saved during the previous

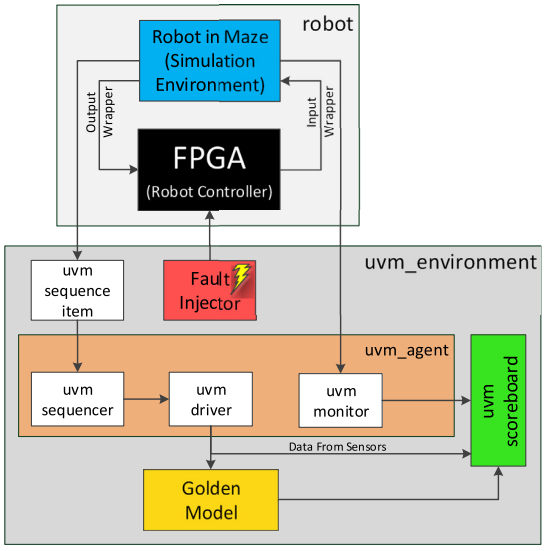


Fig. 13. The architecture of the FPGA-based verification environment for the robot controller.

phase. The important step in this process is fault injection which allow us to inject faults according to various strategies.

9.4. Mechanical part reactions (the third phase)

Checking behaviour of the mechanical part is done by monitoring the information provided by sensors on a robot. The distances from three control points are used for monitoring robot trajectory through the maze, especially checking if the robot finds a goal position. The information about barriers are used for the detection of collisions with a wall. The information about barri-

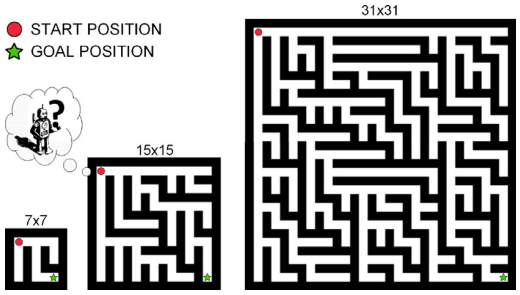


Fig. 14. Three types of mazes.

Table 1
Average number of robot steps.

Maze size	7 × 7	15 × 15	31 x 31
Average number of steps	16	93	433

ers are represented by four values with distances from the wall in four-neighbourhoods of the robot in the maze. These values can be compared with predefined minimal values and verification can detect if the robot is closer to the wall or if the robot crashes into the wall.

Fig. 10 shows general functional verification environment for the third phase. The values from sensors are routed directly to the scoreboard and this verification environment is dedicated just to checking behavior of the mechanical part. In our example, we created one combined verification environment which serves both for the second and the third phase. This verification environment is shown in Fig. 13 where values from sensors and values from a reference model are inputs to the scoreboard which checks electronical and mechanical parts concurrently. It means that scoreboard implements functions both for checking outputs of robot controller and also for checking reactions of mechanical robot. Behavior of mechanical robot is checked by monitoring distances of robot to the wall.

10. Casestudy: experiments and results

Performed experiments correspond to the activities of all phases of the fault tolerance evaluation process.

10.1. The first phase - simulation based verification

The outputs of the first phase are: 1) the electronic part without bugs (robot controller), 2) the list of the used verification scenarios, and 3) achieved coverage. Fig. 14 shows three types of mazes which were used in our experiments. The presented mazes differ in their dimensions and we chose 7x7, 15x15 and 31x31 cells. Examples of start and goal positions are also shown in Fig. 14. With the growing size of the maze the number of steps that the robot must go through increases. The average number of the robot steps in various types of mazes is shown in Table 1. The main goal of the experiments, including debugging the robot controller, was to determine the optimal size of the maze and the number of generated mazes (verification scenarios) which will lead to the best code coverage.

For the experiment, we chose the number of performed verification scenarios equal to 10, 100, 200 and 500, for which we monitored an achieved code coverage. The numbers of performed verification scenarios were the same for all types of mazes and in total 1500 verification scenarios were performed with a variety of mazes. Various bugs were identified and debugged during the verification process. It can be stated that the robot controller operates according to its specification for the performed 1500 verification scenarios.

Experimental results are presented in Table 2. It can be recognized that the maximal achieved total code coverage is 91.85%. The inability of achieving an ideal 100% is caused by the default branches in the source code which are never executed (which is correct), and also by some of the control expressions that are used only when an abnormal situation occurs (e.g. a fault). The table also shows that a rising number of verification scenarios does not increase the achieved code coverage. It is probably because in one scenario multiple input transactions are packed.

On the other hand, resizing the maze from 7 x 7 to 15 x 15 cells led to a slight increase of code coverage, which is possibly due to the effect of the maze. When increasing the size of maze to 31x31 cells, the coverage was not changed. Such studies show that the 7 x 7 cells maze is too small for the next phase of fault impact evaluation process. This trend is shown in the bar chart in Fig. 15 which shows the code coverage for different sizes of mazes for 100 verification scenarios (the part of Table 2).

The results needed to perform the next phase of the fault impact evaluation were obtained in the experiment. Faults will be injected into the electronic controller during each verification scenario in the second phase of evaluation. Each verification scenario will be repeated several times and during each run, various faults or various sequences of faults will be injected.

10.2. The second phase - controller reactions

The second phase in the proposed evaluation process is targeted towards evaluating the correct function of a robot controller implemented into the FPGA. For this purpose fault injection is used. No fault tolerance methodology implemented in the robot controller for these experiments was used and the goals of the experiment were: 1) detailed reliability analysis of the robot controller and its functional units, and 2) a demonstration that the evaluation platform can be used for a fault tolerance evaluation.

As was mentioned above, faults can be injected in a way which reflects various strategies. Similar experiments were done in our previous work [13], but significant differences in evaluation strategies are presented in this paper. We have decided to perform 50 verification runs and inject one fault into one functional unit (single fault) during one verification run and to use mazes of larger dimensions, the mazes of 15 x 15 for this phase. The robot controller consists of 15 functional units which leads to 750 verification runs

Table 2
The experimental results.

# of verification scenarios	10			100			200			500		
Size of mazes	7 x 7	15 x 15	31 x 31	7 x 7	15 x 15	31 x 31	7 x 7	15 x 15	31 x 31	7 x 7	15 x 15	31 x 31
Statement coverage	93,54%	93,70%	93,70%	93,54%	93,70%	93,70%	93,54%	93,70%	93,70%	93,54%	93,70%	93,70%
Branch coverage	94,91%	95,07%	95,07%	94,91%	95,07%	95,07%	94,91%	95,07%	95,07%	94,91%	95,07%	95,07%
Expression coverage	81,33%	81,33%	81,33%	81,33%	81,33%	81,33%	81,33%	81,33%	81,33%	81,33%	81,33%	81,33%
Condition coverage	88,28%	89,18%	89,18%	88,28%	89,18%	89,18%	88,28%	89,18%	89,18%	88,28%	89,18%	89,18%
Total coverage	91,61%	91,85%	91,85%	91,61%	91,85%	91,85%	91,61%	91,85%	91,85%	91,61%	91,85%	91,85%

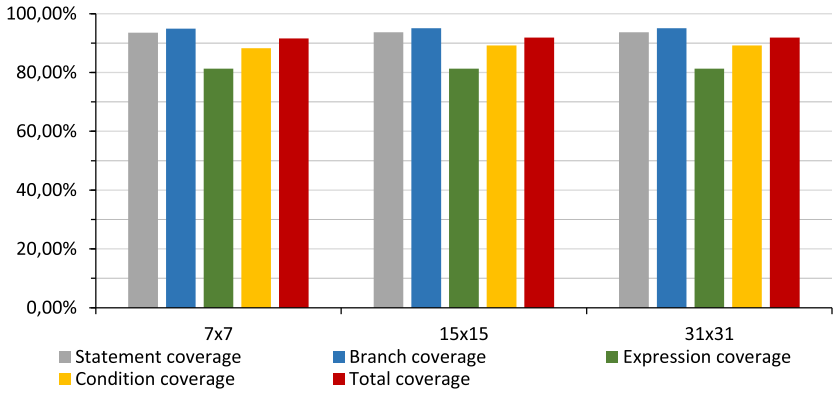


Fig. 15. Code coverage for each type of mazes for 100 verification scenarios.

Table 3
Experimental results in functional verification.

Unit	Number of fails	Fails in %	Unit	Number of fails	Fails in %
bdu	40	80.00	mu_wb	32	64.00
bdu_fsm	20	40.00	peu	39	78.00
bdu_wb	35	70.00	peu_fsm	40	80.00
ecu	38	76.00	pfu	34	68.00
intercon	30	60.00	pfu_wb	28	56.00
mmu	31	62.00	sif_fsm	50	100.00
mu	25	50.00	sif_wb	34	68.00
mu_fsm	1	2.00			

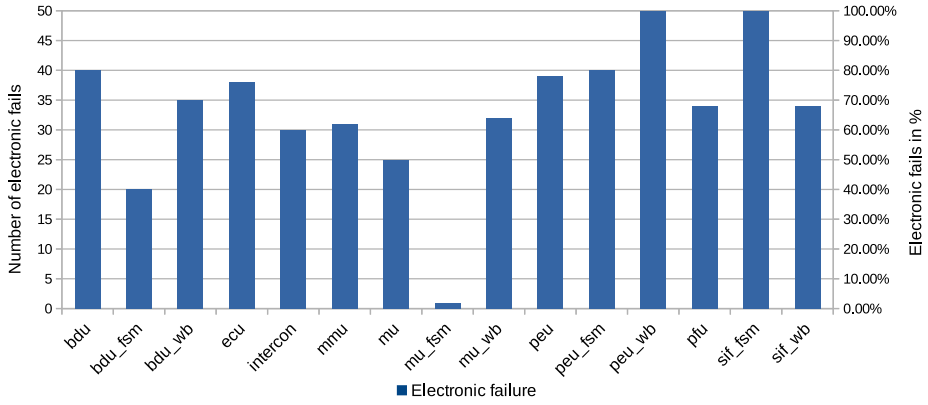


Fig. 16. Experimental results in functional verification.

and injected faults in total. The task of the verification environment was to compare the outputs of the robot controller and check the impact of injected fault. Table 3 shows the number of verification runs where the incorrect outputs of the robot controller were caused by faults (percentage values are shown as well). The total number of verification runs for each functional unit is 50 and the main reason for this is the time complexity of the verification runs, because the robot has to go through the whole maze.

The results of our experiments are shown in Fig. 16 as well. The bar chart expresses a percentage number of faults with their impact on the robot controller. As can be seen, some anomalies in the results of the experiments exist. These include results combined with three functional units *mu_fsm, peu_fsm* and *sif_fsm*. In the case of *mu_fsm*, it is apparently a low number of faults with an impact on the correct function of the robot controller. The *peu_fsm* and *sif_fsm* functional units represent a completely different situation, the number of faults with an impact that is significantly higher than for other units. That is why we repeated the experiments on a higher number of verification runs (225 in this case) with these functional units. Table 4 shows additional verification runs that were performed in order to analyse these anomalies in detail. As can be seen, the additional results are closer to the overall average.

Table 4
Extended experimental results.

Unit	Number of fails	Fails in %
mu_fsm	18	8
peu_fsm	181	80.4
sif_fsm	219	97.3

10.3. The third phase - mechanical part reactions

The evaluation of mechanical robot behaviour was the main task for the third phase. In this phase fault injection is also used. Faults are injected according to the same strategy as in the second phase because the second and the third phases share the

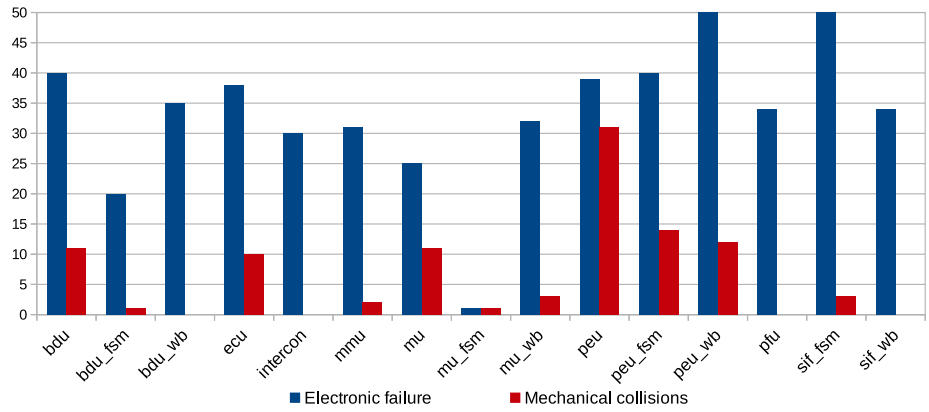


Fig. 17. Experimental results in functional verification.

Table 5
Number of robot collisions.

Unit	Number of el. fails	Number of collisions	Collisions in %
bdu	40	11	27.50
bdu_fsm	20	1	5.00
bdu_wb	35	0	0.00
ecu	38	10	26.32
intercon	30	0	0.00
mmu	31	2	20.00
mu	25	11	4.00
mu_fsm	1	1	22.00
mu_wb	32	3	2.00
peu	39	31	6.00
peu_fsm	40	14	62.00
pfu	34	12	28.00
pfu_wb	28	0	0.00
sif_fsm	50	3	6.00
sif_wb	34	0	0.00

same verification environment. The robot controller was also used without fault tolerance methodologies application and the goal of the experiments corresponding to the third phase was 1) to show the most frequent incorrect behavior of mechanical part; and 2) a demonstration that the evaluation platform based on functional verification is able to monitor the behaviour of the mechanical part.

In these experiments, we found that the most common consequences of injected faults are robot stopping at one place and robot collision with a wall. We can say that all verification runs with electronic failure leads to one of these consequences. If the electronic failure leads to the robot stopping at one place, it usually does not cause any damage. But on the other hand, the collision of the robot with the wall can lead to economical losses. Table 5 summarizes the number of electronic failures for each component and this information is supplemented by the number of collisions with the wall. Also, the number of percentage of electronic failure which lead to a collision is shown.

These results are also presented in the graphical version in the bar chart shown in Fig. 17. It is evident that the percentage number of collisions is different for each functional unit. It shows that some of functional units are more important to robot navigation than others. For example, the percentage for *peu_fsm* functional unit is the highest and the main task of this unit is routing information about its position to the bus. The path through the maze is searched by *pfu* and the movement of the robot is controlled by *ecu*, so the percentage number of collisions corresponding to these two functional units is also quite high.

We have made a fault injection analysis of the robot controller. We found out that some blocks are more prone to faults than others. As can be recognized in the chart showing the results, the functional unit *mu_fsm* is less prone to faults than other units. On the other hand, the units *peu_fsm* and *sif_fsm* are the most prone units for faults. A failure of electronic part usually leads to the robot stopping at a place and to its collision with a wall. As was mentioned above, some of the damaged functional units lead to collisions in more cases than others. So, these functional units are more problematic from a safety point of view. This analysis is especially important for future applications of fault-tolerant methodologies. A system designer obtains the information on which blocks need more attention from a reliability point of view.

The second finding is that we are able to use functional verification in conjunction with the fault injector to determine the impact of faults on the electro-mechanical system. If fault tolerance methodologies will be applied to the electro-mechanical system (in our case, the robot controller) our platform would be used to monitor impact of faults on system hardened against faults and therefore to automate the evaluation of these fault tolerance methodologies.

11. Conclusions and future research

In this work, we presented our evaluation platform for testing fault-tolerant methodologies and evaluation impact of faults on correct operation of electro-mechanical systems. Our evaluation platform is based on functional verification where the verified circuit is running on FPGA which allows us to inject faults directly to the FPGA. Our evaluation process is divided into three phases and each of these phases needs a specific verification environment. Firstly, a basic verification environment was introduced for the first phase of the evaluation process which is able to evaluate a single verification scenario and the creation of an extension that allows automated evaluation of multiple verification scenarios which were presented as well. This automated evaluation uses the verification scenarios produced by our generator or those which are part of the optimized regression test suite. The verification environment for the second phase where DUT is implemented to the FPGA was also mentioned. In the proposed methodology, the verification environment acts as an observer that checks data transferred between the electronic and mechanical parts. During the last phase, the impact of faults to the mechanical part is monitored by checking values received from sensors placed on a robot in a maze. The verification environment for the third phase was also introduced.

For a demonstration of our evaluation platform we proposed one demonstration example which uses a robot and its electronic

controller as an experimental electro-mechanical application. Performed experiments correspond to all phases of a fault impact evaluation process. The output of the first phase was the debugged electronic controller and the list of verification scenarios for the next phase. During the concurrent second and third phase, the reliability analysis was done by means of fault injection into the FPGA. The result was the ratio of faults that caused an incorrect output of the electronic controller and the number of faults that caused the robot collision.

The goal of our future work is to apply various fault tolerance methodologies on the robot controller and evaluate them with our evaluation platform. For example, we plan to construct our robot controller as a fault-tolerant neural network mentioned in this paper. We can also use more conventional fault-tolerant methodologies, such as TMR, on-line checkers or error correction codes. We will focus on testing fault tolerance methodologies targeted to FPGAs in the context of electro-mechanical systems which is often the way of using fault-tolerant electronic controllers. On the basis of these results, we are going to develop generally usable principles of developing systems for evaluating fault-tolerant qualities of electro-mechanical systems.

Acknowledgment

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II); project IT4Innovations excellence in science - LQ1602, ARTEMIS JU under grant agreement no 621439 (ALMARVI) and BUT project FIT-S-17-3994.

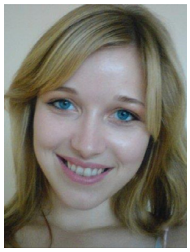
References

[1] A. Benso, P. Prinetto, Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation, Frontiers in Electronic Testing, vol. 23, Springer Science & Business Media, 2003.
[2] A. Meyer, Principles of Functional Verification, Elsevier Science, 2003 http://books.google.cz/books?id=qaIiX3hYWL4C.
[3] A. standard, Universal Verification Methodology, 2016 http://www.accellera.org/downloads/standards/uvm.
[4] V.R. Cooper, Getting Started with UVM: A Beginner's Guide, Austin, TX:Verilab, 2013.
[5] IEEE Standard for Systemverilog - Unified Hardware Design, Specification, and Verification Language, 2005, doi:10.1109/IEEESTD.2005.97972.
[6] I. Koren, C.M. Krishna, Fault-Tolerant Systems, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.
[7] F. Siegle, T. Vladimirova, J. Ilstad, O. Emam, Mitigation of radiation effects in SRAM-based FPGAs for space applications, ACM Comput. Surv. 47 (2) (2015) 37:1–37:34, doi:10.1145/2671181.
[8] XILINX, FPGA, 2014 http://www.xilinx.com/fpga/index.htm.
[9] M. Ceschia, M. Violante, M. Reorda, A. Paccagnella, P. Bernardi, M. Rebaudengo, D. Bortolato, M. Bellato, P. Zambolin, A. Candelori, Identification and Classification of Single-event Upsets in the Configuration Memory of SRAM-based FPGAs, vol. 50, 2003, pp. 2088–2094.
[10] C. Bernardeschi, L. Cassano, A. Domenici, L. Sterpone, Accurate simulation of SEUs in the configuration memory of SRAM-based FPGAs, in: Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2012 IEEE International Symposium on, IEEE, 2012, pp. 115–120.
[11] M. Alderighi, S. D'Angelo, M. Mancini, G.R. Sechi, A fault injection tool for SRAM-based FPGAs, in: On-Line Testing Symposium, 2003. IOLTS 2003. 9th IEEE, IEEE, 2003, pp. 129–133.
[12] M. Alderighi, F. Casini, S. d'Angelo, M. Mancini, S. Pastore, G.R. Sechi, Evaluation of single event upset mitigation schemes for SRAM-based FPGAs using the FLIPPER fault injection platform, in: Defect and Fault-Tolerance in VLSI Systems, 2007. DFT'07. 22nd IEEE International Symposium on, IEEE, 2007, pp. 105–113.
[13] J. Podivinsky, O. Cekan, M. Simkova, 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.
[14] O. Goloubeva, M. Rebaudengo, M.S. Reorda, M. Violante, Software-Implemented Hardware Fault Tolerance, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
[15] H.R. Mahdiani, S.M. Fakhraie, C. Lucas, Relaxed fault-tolerant hardware implementation of neural networks in the presence of multiple transient errors, IEEE Trans. Neural Netw. Learn. Syst. 23 (8) (2012) 1215–1228, doi:10.1109/TNNLS.2012.2199517.
[16] L. Miculka, Z. Kotasek, Generic partial dynamic reconfiguration controller for transient and permanent fault mitigation in fault tolerant systems implemented into FPGA, in: Design and Diagnostics of Electronic Circuits Systems, 17th International Symposium on, 2014, pp. 171–174, doi:10.1109/DDECS.2014.6868784.
[17] M. Chen, R. Mei, Actuator fault tolerant control for a class of nonlinear systems using neural networks, in: Control Automation (ICCA), 11th IEEE International Conference on, 2014, pp. 101–106, doi:10.1109/ICCA.2014.6870903.
[18] Z. Li, Fault diagnosis and fault tolerant control of mobile robot based on neural networks, in: Machine Learning and Cybernetics, 2009 International Conference on, vol. 2, 2009a, pp. 1077–1081, doi:10.1109/ICMLC.2009.5212376.
[19] Z. Li, Application of fault tolerant controller based on RBF neural networks for mobile robot, in: Intelligent Ubiquitous Computing and Education, 2009 International Symposium on, 2009b, pp. 531–534, doi:10.1109/IUCE.2009.140.
[20] B.S. Arad, A. El-Amawy, On fault tolerant training of feedforward neural networks, Neural Netw. 10 (3) (1997) 539–553, doi:10.1016/S0893-6080(96)00089-5.
[21] C. Sequin, R. Clay, Fault tolerance in artificial neural networks, in: Neural Networks, 1990., 1990 IJCNN International Joint Conference on, 1990, pp. 703–708 vol.1, doi:10.1109/IJCNN.1990.137651.
[22] Z.-H. Zhou, S.-F. Chen, Z.-Q. Chen, Improving tolerance of neural networks against multi-node open fault, in: Neural Networks, 2001. Proceedings. IJCNN '01. International Joint Conference on, . 3, 2001, pp. 1687–1692 vol.3, doi:10.1109/IJCNN.2001.938415.
[23] B. Girau, FPNA: Concepts and Properties, in: A.R. Omondi, J.C. Rajapakse (Eds.), FPGA Implementations of Neural Networks, Springer US, 2006, pp. 63–101, doi:10.1007/0-387-28487-7-3.
[24] M. Krcma, J. Kastil, Z. Kotasek, Mapping trained neural networks to FPNNs, in: Design and Diagnostics of Electronic Circuits Systems (DDECS), 2015 IEEE 18th International Symposium on, 2015a, pp. 157–160, doi:10.1109/DDECS.2015.50.
[25] M. Krcma, Z. Kotasek, J. Kastil, Fault tolerant field programmable neural networks, in: Nordic Circuits and Systems Conference (NORCAS): NORCHIP International Symposium on System-on-Chip (SoC), 2015, 2015b, pp. 1–4, doi:10.1109/NORCHIP.2015.7364381.
[26] J. Podivinsky, O. Cekan, M. Simková, Z. Kotásek, The evaluation platform for testing fault-tolerance methodologies in electro-mechanical applications, Microprocess. Microsyst. 39 (8) (2015) 1215–1230, doi:10.1016/j.micpro.2015.05.011.
[27] R. Giegerich, Introduction to Stochastic Context Free Grammars, Humana Press, Totowa, NJ, 2014, doi:10.1007/978-1-62703-709-9_5.
[28] A. Meduna, Formal Languages and Computation: Models and Their Applications, first, Auerbach Publications, Boston, MA, USA, 2014.
[29] A. Ronacher, Jinja2 (the python template engine), 2014 http://jinja.pocoo.org/.
[30] M. Lutz, Learning Python, second, O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2003.
[31] M. Belesova, M.Simkova, Z.Kotasek, T.Hruska, Application of evolutionary algorithms for regression suites optimization., in: IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits and Systems, IEEE Computer Society, 2015, pp. 91–949.
[32] M. Belesova, M.Zachariasova, Z.Kotasek, Regression test suites optimization for application-specific instruction-set processors and their use for dependability analysis., in: Proceedings of the 19th Euromicro Conference on Digital Systems Design, IEEE Computer Society, 2016, pp. 380–387.
[33] B. Wille, J. Goss, W. Roesner, Comprehensive Functional Verification, Elsevier, 2005.
[34] T. Bäck, J. Kok, Handbook of Natural Computing, Springer-Verlag, Berlin Heidelberg, 2012.
[35] M. Straka, J. Kastil, 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.
[36] C.-Y. Huang, Y.-F. Yin, C.-J. Hsu, T.B. Huang, T.-M. Chang, SoC hw/sw verification and validation, in: 16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011), IEEE, 2011, pp. 297–300.
[37] Accellera, Standard Co-Emulation Modeling Interface (SCE-MI) Reference Manual, 2011 http://accellera.org/images/downloads/standards/sce-mi/SCEMIv21110112final.pdf.
[38] Xilinx Inc., Ml506 Evaluation Platform User Guide, UG347 (v3. 1.2), 2011.
[39] P. Skibik, Implementation of Ethernet Communication Interface into FPGA Chip, Technical Report, 2011 https://www.vutbr.cz/wwwbase/zavpracesouborverejne.php?fileid=40494.
[40] C. Lavin, M. Padilla, P. Lundrigan, B. Nelson, B. Hutchings, Rapid prototyping tools for FPGA Designs: RapidSmith, in: Field-Programmable Technology (FPT), 2010 International Conference on, 2010, pp. 353–356, doi:10.1109/FPT.2010.5681429.
[41] N. Dorairaj, E. Shiflet, M. Goosman, PlanAhead software as a platform for partial reconfiguration, vol. 55, 2005, pp. 68–71.
[42] B. Gerkey, R.T. Vaughan, 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.
[43] J. Podivinsky, M. Simkova, 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.
[44] S. Krajcir, Functional Verification of Robotic System Using UVM, Technical Report, 2015 http://www/study/DP/DP.php?id=15154.
[45] J. Buck, Maze generation: algorithm recap, 2011 http://weblog.jamisbuck.org/2011/2/7/maze-generation-algorithm-recap.

Jakub Podivinsky was born in 1989. In 2013 he graduated (MSc) at the Department of Computers Systems of the Faculty of Information Technology, Brno University of Technology. In 2013 he started his PhD studies at the Department of Computers Systems. His scientific research is focused on evaluation quality of fault tolerant systems and FPGA-based functional verification of digital systems.

Ondrej Cekan was born in 1989. In 2013 he graduated (MSc) at the Department of Computers Systems of the Faculty of Information Technology, Brno University of Technology. In 2013 he started his PhD studies at the Department of Computers Systems. His scientific research is focused on functional verification and stimuli generation.

Jakub Lojda was born in 1991. In 2015 he graduated (MSc) at the Department of Computers Systems of the Faculty of Information Technology, Brno University of Technology. In 2015 he started his PhD studies at the Department of Computers Systems. His scientific research is focused on fault-tolerant systems design automation.



Marcela Zachariasova was born in 1987. In 2011 she graduated (MSc) at the Department of Computers Systems of the Faculty of Information Technology, Brno University of Technology. In 2011 she started her PhD studies at the same university and successfully finished in 2015. The topics of her PhD thesis and the main areas of interest are optimization of UVM-based functional verification, automated verification of processors and fault-tolerant system design.

Martin Krcma was born in 1988. In 2014 he graduated (MSc) at the Department of Computers Systems of the Faculty of Information Technology, Brno University of Technology. In 2014 he started his PhD studies at the Department of Computers Systems. His scientific research is focused on fault tolerant implementations of artificial neural networks in FPGAs.

Zdenek Kotasek was born in 1947. He received his MSc. and PhD. degrees (in 1969 and 1991) from Brno University of Technology (BUT), both in computer science. Between 1969 and 2001, he worked at the Department of Computer Science of the Faculty of Electrical Engineering and Computer Science, since 2002 at the Department of Computer Systems (DCSY) of the Faculty of Information Technology, both at BUT. He is an Associate Professor at BUT since 2000, he was in the position of the DCSY head since 2005 till 2015. His research interests include digital circuit diagnostics and testing, testability analysis and design and synthesis for testability and reliability, fault tolerant system design. He is an IEEE senior member (since 2015).

Paper C

An Experimental Evaluation of Fault-Tolerant FPGA-based Robot Controller

Jakub Podivinsky, Jakub Lojda and Zdenek Kotasek

In: Proceedings of IEEE East-West Design & Test Symposium. Kazan: IEEE Computer Society, 2018, pp. 63-69. ISBN 978-1-5386-5709-6.

An Experimental Evaluation of Fault-Tolerant FPGA-based Robot Controller

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

Abstract

Field Programmable Gate Arrays (FPGAs) are becoming more popular in various areas. Single Event Upsets (SEUs) are faults caused by a charged particle in the configuration memory of SRAM-based FPGAs. Such a charged particle can cause incorrect behavior in the whole system. This problem becomes greater if such a system operates in an environment with increased radiation (e.g. space applications). Lots of techniques to harden FPGAs against faults exist and new techniques targeted to FPGA are in scope of many researchers. One such technique is called Triple Modular Redundancy (TMR). It is important to evaluate these techniques on a real system with a real FPGA. An evaluation platform based on an artificial fault injection and a functional verification for testing fault tolerance methodologies is introduced in this paper. Parts of our experimental system are hardened by using TMR and its experimental evaluation is one of the main parts of this paper. In this paper, we focus on the TMR fault tolerance method and change the target functional unit, on which the method is applied. This allows us to determine the reliability gain obtained through the hardening of a particular functional unit and allows us to compare the results. We propose experiments with various fault injection strategies (multiple and single faults) and monitor impact of faults on both the electronic and mechanical parts of the experimental system.

1. Introduction

Field Programmable Gate Arrays (FPGAs) are becoming more popular for a number of reasons. SRAM-based FPGAs can provide hardware implementation of applications that are often faster than processor-based implementations and require lower costs than Application-Specific Integrated Circuits (ASICs) [4]. Moreover, their reconfigurability makes them flexible almost like processors do and offers us to change implemented application during the life cycle of such a system. FPGAs can be used in different areas, e.g. automotive, aerospace or space. SRAM-based FPGAs consist of programmable components (configurable logic blocks, look-up tables, flip-flops, etc.) and their interconnection. Configuration of FPGA is stored as a bitstream in the configuration memory. Many FPGAs use an SRAM memory as their configuration memory. Sensitivity to faults caused by charged particles is the problem of FPGAs from the reliability point of view. A hit of charged particles can lead to the inversion of a bit in configuration memory which can change implemented behavior of the whole system. This problem is called Single Event Upset (SEU). This may be a problem, especially when FPGAs are used in areas with increased radiation, e.g. space applications [12].

Although 80-99% of SEUs hit unused bits of the configuration bitstream [4], it is important to harden FPGA-based systems against faults. Lots of fault tolerance techniques targeted to FPGAs exist and new ones are subject of investigation. The use of spatial redundancy for hardening user logic against SEUs is presented in [12]. One of the main approaches of spatial redundancy is Triple Modular Redundancy (TMR), which many researchers are trying to improve. For example, paper [14] proposes a new fault tolerance method targeted to SRAM-based FPGAs. This technique is based on a heuristic and uses inaccurate modules in combination with TMR. This approach reduces the power and area overhead of the hardened design.

Paper [15] presents improved TMR which interest *don't care bits* of LUT configuration bits. These LUTs are classified to the set of SEU-sensitive and SEU-insensitive. This improved approach just triplicates

978-1-5386-5710-2/18/$31.00

sensitive LUTs and reduces the area overhead unlike the classical TMR approach.

It is important to evaluate fault tolerance techniques targeted to FPGAs. Three main approaches are presented in [4]: 1) modeling tools, 2) fault emulation testing and 3) accelerated radiation testing. Authors of [2] present emulation method which emulates effect of SEUs in the configuration of FPGAs. Presented method is based on combination of simulation and topological analysis of the circuit configured in FPGA. Methotds based on synthesizable model of faults are presented in [11]. Authors propose fault injection tool which allows to inject faults to FPGA, but various synthesizable fault models must be added to the original design. This method requires modifications of the original circuit in the VHDL, additional gates and wires must be inserted. FLIPPER, the evaluation platform based on the fault injection directly into an physical FPGA without modification of the original circuit description, is presented in [1]. Thank to the dynamic reconfiguration, it is possible to read, modify and write back the configuration memory. FLIPPER uses two boards with FPGAs, one is the main control board and second is board where tested circuit is implemented. The main board control fault injection which can be driven by the application operating on a computer. Fault injection can be controller by a additional functional unit implemented the same FPGA. This approach is presented in [6]. The main reason is the acceleration of the fault impact evaluation. Fault injection process can be initialized from computer to which experimental FPGA is connected.

In our previous work, we focused on several architectures with various fault tolerance methods. Such as in paper [9], we target a processor based system and its hardening. In paper [5], we focused on systems generated with the use of High-Level Synthesis (HLS) and proposed a mechanism to incorporate fault tolerance into storage elements and operations through redefinition of data types behavior.

In our last research, we developed the evaluation platform which allow us to test fault tolerance properties [8]. The developed platform uses functional verification and previously developed fault injector [13]. The main advantage of our platform is its ability to test the impacts of injected faults not only on electronic controller implemented in FPGA, but also on the mechanical part controlled by the tested electronic controller, because lots of electronic systems control the mechanical part which is an important component of the whole system. In this paper, triplication (TMR) is applied on our experimental system which is composed of robot in a maze and its electronic controller and our platform is used to analyze the reliability gained. It should be noted that a great number of fault tolerant systems are electromechanical applications. As an example, the FT systems in planes can be used. That is why we concentrated on a robot with its controller as the experimental system.

In our research group, we also investigate new methods in the area of fault-tolerant systems design automation. Our aim is to create a fully automated environment to fault-tolerant systems design and its evaluation. The experiments presented in this paper are a step towards this new methodology, as it is important to understand the behavior of various components of the system utilizing different proportions of FPGA primitive types during the presence of faults in these components.

The paper has the following structure. Section 2 introduces our functional verification-based evaluation platform and experimental electro-mechanical system. Experiments and results with multiple and single fault injection are presented in Section 4. Section 5 contains the conclusion of this work and presents plans for our future research.

2. Evaluation Platform and Experimental System

This section briefly describes our evaluation platform for evaluating fault tolerance methodologies which was presented in journal publication [8]. Our platform is based on functional verification [7] and standardized Universal Verification Methodology (UVM) [3]. The task of functional verification is to check if a hardware system matches a given specification. In our case, functional verification is used as a tool for checking if injected faults caused some discrepancy on the output of the tested system. The platform is shown in Figure 1. It is composed of several components running on a computer and on an FPGA development board. The important part of the platform is the software part of the verification environment which is running on a computer. The verification environment observes communication between both parts of the experimental electro-mechanical system (electronic controller and controlled mechanical part). The electronic controller is run on an FPGA which is connected to the simulation of the mechanical part (running on a computer) through the Ethernet interface. Artificial faults are injected through JTAG interface which is used by the fault injector [13]. The fault injector uses partial reconfiguration, it reads specified part of the bitstream stored in configuration memory, inverts the specified bit and writes it back to the configuration memory.

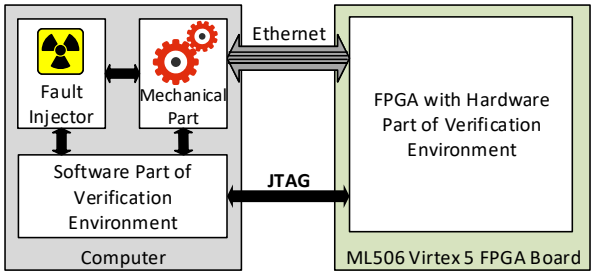


Figure 1. The evaluation platform architecture.

Together with the platform, we need to introduce a process for verifying fault tolerance properties which is composed of three phases. The *first phase* of evaluation process is simulation based verification. Verification is completely done in an RTL simulator (ModelSim) in this phase. The task of this phase is to check if the electronic controller operates correctly according to the given specification. The task of the *second phase* is verification of the electronic controller which is implemented into an FPGA. Scenarios which was obtained during first phase are repeated together artificial faults injection into FPGA. Previously developed fault injector is used in this phase. The impact of injected faults on the behavior of electronic part is monitored in this phase. The goal of the *third phase* is the analysisof the faults which lead to corruption of the mechanical part. The second and third phases use the FPGA-based verification environment, where a device under test is run on the FPGA. The second phase monitors the effect of faults on communication between the electronic controller an the mechanical part. The third phase checks the values of sensors on the mechanical part and monitors its behavior.

3. Case Study: Robot in A Maze

Our goal is to demonstrate the proposed platform and evaluation process on a real electro-mechanical system. Our experimental electro-mechanical system was developed for these purposes. It is composed of a robot for seeking a path in a maze and its electronic controller implemented in FPGA. The robot controller is not a very complex system, but it is split into various components (bus, finite state machines, memories, etc.) which allow us to evaluate a wide scale of fault tolerance methodologies. We do not have a real robot, and, thus, we simulate its behavior. We use the Player/Stage simulation tool which is able to simulate robot and its environment. In our case we simulate the robot in a maze. The simulation tool is running on a computer from which data must be transfered to FPGA board. We use Ethernet interface which allows us to transfer data between the robot in simulation (computer) and its electronic controller (FPGA).

The main component of the Robot controller [10] is central bus through which communication between various functional units is accomplished. Controller consist of 16 functional units, the most important are Position Evaluation Unit (PEU) together with the Barrier Detection Unit (BDU). The main task of BDU is to calculate actual position of robot and also to detect obstacles in the neighborhood. The obtained informations are stored in the Map Memory Unit (MMU) through the Map Unit (MU). Path Finding Unit (PFU) implements the algorithm for seeking path in maze which is based on informations stored in memory (MMU). The Engine Control Unit (ECU) controls mechanical parts of the robot in maze. A control finite state machine (FSM) and bus wrapper are important accessories of almost all functional units.

Figure 2 shows a combined FPGA-based verification environment for the second and the third phases of the proposed evaluation process. The verification environment is composed of two parts: 1) the UVM-based verification environment and 2) the experimental electromechanical system (robot in a maze). The verification environment operates just as an observer which checks communication of the robot in a maze with the FPGA without direct intervention. The golden model is used for the comparison of expected data and really transfered data. Some discrepancy is indicated as an error on the output of the electronic controller. On the other hand, as the third phase, data from sensors and the correct behavior of the mechanical robot are monitored.

4. Experiments and Results

In our experiments, we decided to examine the impact of faults on particular components of the robot controller unit. The experiments comprise the comparison of results obtained from selected unhardened components of the controller unit with their hardened versions. As a method of redundancy insertion, the TMR was selected. Three components of the robot controller unit, the *PEU*, the *BDU* and the *ECU*, were selected for comparison. The reason for this choice was that the *PEU* and the *BDU* components compute the input values for the path-searching algorithm. From this point of view, these components are very important and the complete controller unit is function-dependent on them. The *ECU* component directly affects the

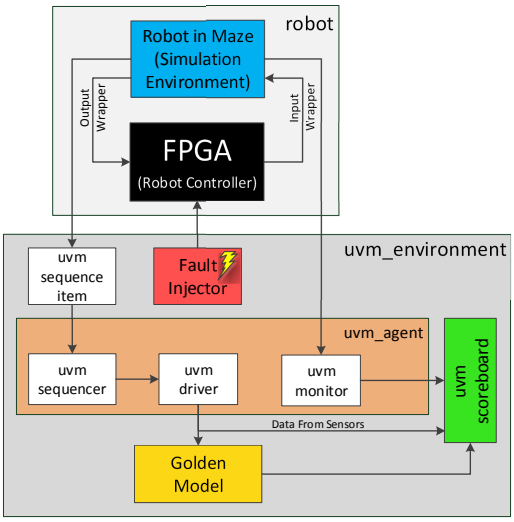


Figure 2. The FPGA-based verification environment for the second and third phases of the robot controller evaluation.

movement of the robot, thus failure of this component would cause the complete controller unit to fail.

In this experimentation, we use our evaluation platform, which is based on a permanent configuration bitstream bit-flip, and, thus, we use this type of error in our evaluation. The experiments were performed with the usage of two fault injection strategies. The first strategy was to inject a *single fault* into an individual component per verification run before the robot was started and observe its ability to reach the target position. The second strategy comprised *multiple faults* injections per verification run. In this case, a number of faults were injected until the first failure propagated to the controller outputs was observed.

4.1. Multiple Faults Injection

In the *multiple faults* injection experiment, permanent bit-flips were injected into utilized *Look-up Tables* (LUTs) contents with a constant period of $5s$. This period was experimentally chosen based on the system failure manifestation time. This means that each $5s$ only one SEU was injected into the particular component of the robot controller unit LUT contents (only utilized LUT bits are considered) until the robot failed or reached the target position.

At first, the *multiple faults* were injected into the *unhardened* version of the robot controller. The reason for this was to find out the behavior of the whole system without any fault tolerance method involved. In this stage, one set of 1000 verification runs was done for each of the selected components in which faults were only injected into the particular controller unit LUTs contents. The statistical results are shown by the *PEU_noft*, the *BDU_noft* and the *ECU_noft* bars of the box plot in Figure 3. Box plot shows for each component the minimum, the first quartile (25%), median, the second quartile (75%) and maximum of the numbers of faults injected into FPGA that for each particular run were enough to manifest a failure on the controller outputs. As can be seen, each component has its own level of susceptibility to SEUs.

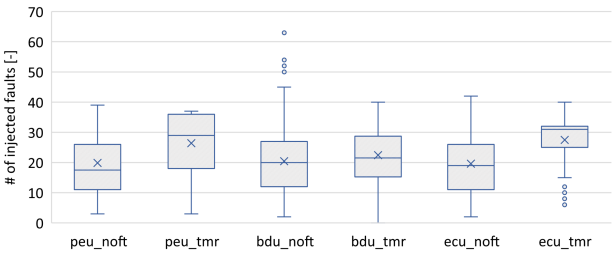


Figure 3. Box plot shows statistical evaluation of number of faults injected into FPGA which led to the electronic failure.

Then, three other robot controller unit designs with the TMR applied selectively to the *PEU*, the *BDU* and the *ECU* components were created. Equivalent experiments were repeated with the three new designs in which only the faults were injected into the particular *hardened* component. The bars *PEU_tmr*, *BDU_tmr* and *ECU_tmr* in the box plot in Figure 3 show the susceptibility to faults with the TMR applied. As can be seen, each of the bars representing the *hardened* version is above the *unhardened* one, therefore, more fault injections were required to cause a malfunction of the complete controller unit.

We must note that when multiple faults were injected, the *hardened* version failed in a smaller number of cases than the *unhardened* version. The numbers of cases in which the complete controller unit failed while faults were injected into the selected components are shown in Table 1. As can be seen, the application of the TMR led up to 93.3% decrease of failure manifestations. We can conclude that the TMR led to a lower number of electronic failures and also led to the increased number of faults injected into FPGA which caused a failure.

Besides the influence of faults on the electronic part of the system, we also observed its influence on the mechanical part. The electronic failure usually stopped

Table 1. The impact of multiple faults injected into the unhardened and hardened versions of robot controller both on the electronic controller and mechanical part.

Monitored impact	*PEU*		*BDU*		*ECU*	
	noft	*tmr*	*noft*	*tmr*	*noft*	*tmr*
Electronic OK [–]	656	977	361	917	226	622
Electronic failed [–]	344	23	639	83	774	378
Goal not reached [–]	344	23	639	83	774	378
Collision with wall [–]	0	0	0	0	15	3
Robot stop on place [–]	344	23	639	83	759	375
Reliability improvement [%]	93.3%		87.0%		51.2%	

the robot on its position and in some cases the failure led to a collision with a wall. It can be noted that the stopping of the robot on its position is a less serious failure consequence than the collision. Table 1 also shows the numbers of cases in which the robot crashed into the wall and the numbers of cases in which the robot stopped at a place.

4.2. Single Faults Injection

In the case of the *single faults* injection experiment, exactly one bit-flip of the utilized LUT contents of a particular component was injected per verification run and its impact on the behavior of the whole controller unit was observed. At first, 1000 verification runs with the *unhardened* design of our robot controller unit were performed. Table 2 shows the numbers of runs that led to an electronic failure. As can be seen, if the *single faults* are injected, the number of failures is significantly lower than in the case of *multiple faults* injection.

Then, the verification runs with *single fault* injections were repeated on the *hardened* design. Table 2 also shows the numbers of runs in which an injection into the selected components with the TMR applied led to a failure (the columns *PEU_tmr*, *BDU_tmr* and *ECU_tmr*). We believe that the fact the *hardened* unit occasionally fails after the *single fault* injection is caused by hitting the *voter* which is needed to interface the particular component with the rest of the system.

The Figure 4 shows the number of faults which lead to the failure of the electronic controller. The figure shows the graphical comparison of the *hardened* and *unhardened* versions. One can see that the *BDU* component is the most vulnerable to faults and this shows that the *BDU* is a really important component of the whole robot controller. One can see, for the *hardened* version of the design, the number of failures is lower for each of the selected components.

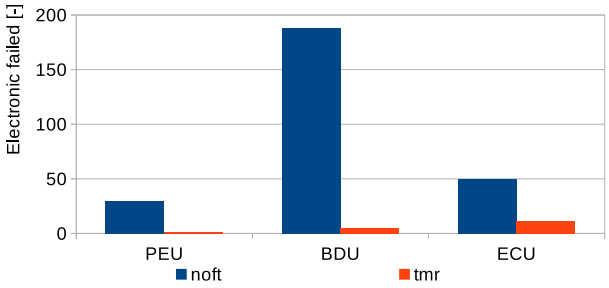


Figure 4. Number of faults injected into FPGA which cause the electronic failure.

As in the previous case, we observed the impact of faults both on the electronic and the mechanical parts of the experimental system as well. Not all of the faults injected into FPGA that caused the electronic controller failure caused the robot to collide with the wall. Table 2 also shows the numbers of wall collisions and cases where the robot stopped during its journey.

5. Conclusions and Future Research

The use of functional verification as a tool for evaluation of impact of artificial faults injected into the configuration of SRAM-based FPGAs was presented in this paper. Our platform was demonstrated on the evaluation of the impact of injected faults on the robot controller which navigates the robot in a maze. The paper shows experimental results with the *hardened* (triplication) along with the *unhardened* robot controller version. Both *single* and *multiple faults* injection strategies were used. Our experiments show the benefit of triplication which in the case of the *single fault* injections led to a lower number of electronic failures. We believe the susceptibility of the *hardened* unit to the *single fault* injection is caused by hitting the

Table 2. The impact of single faults injected on the unhardened and hardened versions of robot controller both on the electronic controller and mechanical part.

Monitored impact	*PEU*		*BDU*		*ECU*	
	noft	*tmr*	*noft*	*tmr*	*noft*	*tmr*
Electronic OK [−]	971	1000	813	996	952	990
Electronic failed [−]	29	0	187	4	48	10
Goal not reached [−]	29	0	187	4	48	10
Collision with wall [−]	0	0	5	0	1	0
Robot stop on place [−]	29	0	182	4	47	10
Reliability improvement [%]	100.0%		97.9%		79.2%	

voter, which is needed to interface the particular component with the rest of the system. In the case of the *multiple faults* injection, it is clearly visible that the triplication led to a lower number of electronic failures, but experiments have also shown that the number of injected faults which cause a failure is higher than in the case of the *unhardened* robot controller. The number of failures is significantly higher than in the case of the *single fault* injections, as in this experiment, *multiple faults* were injected during one verification run.

The results presented in this paper are integral part of our future research in which we will integrate faulty module recovery into our robot controller, which significantly increases the resource utilization, and, thus, we aim to harden only the most sensitive functional units of the robot controller. The results obtained in this research will help us to increase the efficiency of the reliability method.

As for future research, our goal is to use the reconfiguration as a tool for faulty module recovery. We expect that the benefit of recovery will be most obvious in the case of *multiple faults* injection. This expectation will be confirmed or refuted by repeating similar experiments as shown in this paper.

Acknowledgements

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II), the project IT4Innovations excellence in science – LQ1602, the BUT project FIT-S-17-3994 and the JU ECSEL Project SECREDAS (Product Security for Cross Domain Reliable Dependable Automated Systems), Grant agreement No. 783119.

References

[1] M. Alderighi, F. Casini, S. d'Angelo, M. Mancini, S. Pastore, and G. R. Sechi. Evaluation of Single Event Upset Mitigation Schemes for SRAM-based FPGAs Using the FLIPPER Fault Injection Platform. In *Defect and Fault-Tolerance in VLSI Systems, 2007. DFT'07. 22nd IEEE International Symposium on*, pages 105–113. IEEE, 2007.

[2] C. Bernardeschi, L. Cassano, A. Domenici, and L. Sterpone. Accurate Simulation of SEUs in the Configuration Memory of SRAM-based FPGAs. In *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2012 IEEE International Symposium on*, pages 115–120. IEEE, 2012.

[3] V. R. Cooper. *Getting Started with UVM: A Beginner's Guide*. Austin, TX : Verilab, 2013.

[4] P. Gaillardon. *Reconfigurable Logic: Architecture, Tools, and Applications*. Devices, Circuits, and Systems. CRC Press, 2015.

[5] J. Lojda, J. Podivinsky, and Z. Kotasek. Redundant Data Types and Operations in HLS and Their use for a Robot Controller Unit Fault Tolerance Evaluation. In *East-West Design & Test Symposium (EWDTS), 2017 IEEE*, pages 273–278. IEEE, 2017.

[6] C. López-Ongil, M. Garcia-Valderas, M. Portela-García, and L. Entrena. Autonomous fault emulation: a new fpga-based acceleration system for hardness evaluation. *Nuclear Science, IEEE Transactions on*, 54(1):252–261, 2007.

[7] A. Meyer. *Principles of Functional Verification*. Elsevier Science, 2003.

[8] 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*, 52:145–159, 2017.

[9] J. Podivinsky, J. Lojda, O. Cekan, R. Panek, and Z. Kotasek. Evaluation Platform for Testing Fault Tolerance Properties: Soft-core Processor-based Experimental Robot Controller. In *Digital System Design (DSD), 2018 Euromicro Conference on*. IEEE, 2018.

[10] J. Podivinsky, M. Simkova, and Z. Kotasek. Complex Control System for Testing Fault-Tolerance Methodologies. In *Proceedings of The Third Workshop MEDIAN 2014*, pages 24–27. COST, 2014.

[11] S. Rudrakshi, V. Midasala, and S. Bhavanam. Implementation of fpga based fault injection tool (fito) for testing fault tolerant designs. *IACSIT International Journal of Engineering and Technology*, 4(5):522–526, 2012.

[12] F. Siegle, T. Vladimirova, J. Ilstad, and O. Emam. Mitigation of Radiation Effects in SRAM-Based FPGAs for Space Applications. *ACM Comput. Surv.*, 47(2):37:1–37:34, Jan. 2015.

[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*, pages 223–230. IEEE Computer Society, 2011.

[14] S. Venkataraman, R. Santos, and A. Kumar. A Flexible Inexact tmr Technique for SRAM-based FPGAs. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, pages 810–813. EDA Consortium, 2016.

[15] M. S. Zheng, Z. L. Wang, J. Tu, J. Y. Wang, and L. J. Li. Reliability Oriented Selective Triple Modular Redundancy for SRAM-Based FPGAs. In *Applied Mechanics and Materials*, volume 713, pages 1127–1131. Trans Tech Publ, 2015.

Paper D

FPGA Prototyping and Accelerated Verification of ASIPs

Jakub Podivinsky, Marcela Zachariasova, Ondrej Cekan and Zdenek Kotasek

In: IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits and Systems. Belgrade: IEEE Computer Society, 2015, pp. 145-148. ISBN 978-1-4799-6779-7.

FPGA Prototyping and Accelerated Verification of ASIPs

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

***Abstract*—In current SoC verification, the trend is to create verification solutions that are tailored to specific issues in SoC or to specific architectures. The reason is that the complexity of these systems makes it difficult to use general verification approaches such as formal or simulation-based verification. This paper presents a solution that is targeted to one particular area - Application-Specific Instruction-Set Processors (ASIP) and multi-processor systems containing several ASIPs. We propose automated FPGA prototyping and accelerated verification of these systems while the accelerated verification environment corresponds to the principles of UVM (Universal Verification Methodology) therefore can easily be integrated. Automated generation of verification environments and acceleration of verification runnning on a real hardware platform makes this solution very unique and beneficial, not only in speed, but also in debugging specific hardware issues.**

Keywords*—*UVM, Acceleration, FPGA Prototyping, ASIP.

I. Introduction

The current embedded systems, such as *Systems on Chip* (SoC), *multi-processor systems* (MPSoC) or equipment for *Internet of Things* (IoT), are more and more complex. They usually consist of one or more processors (either *General Purpose Processors* (GPPs) or *Application Specific Instruction-set Processors* (ASIPs)) and various types of peripherals. An important phase in the development cycle of these systems is the verification of their functionality. Various approaches for verification currently exist, such as formal verification, assertion-based verification or simulation-based verification (also called functional verification). But in general, functional verification is easier to apply for hardware engineers as they are familiar with simulation tools and this approach does not require a deep knowledge of formal specifications. Moreover, standard languages, methodologies and libraries were defined for functional verification. The most commonly known are the SystemVerilog IEEE language standard [1], Universal Verification Methodology (UVM) [2] and the open-source UVM library (with all the basic components of verification environments). They work well for unit level verification, but for processors, SoC or MPSoC, they do not scale well. The reason is not only in the complexity of these systems, but also the fact that software embedded into processors must be taken into account as well [3], [4]. Moreover, their verification is time consuming and this can lead to undesirable prolongation of the time to market.

Therefore, because of its complexity, it seems to be the current state-of-the-art in SoC verification to come with a verification solution that is adjusted to SoC (digital vs. analogy, verification IPs, graph-based IP connections, etc.); or their application domain (e.g. multimedia, DSP applications, smart devices, etc.) and is often connected to the development tool of these systems. For example, Breker [5] introduces a graph-based approach to functional verification. Users capture with graphs the IP level scenarios as nodes and connections make the SoC level scenario. Cadence [6], Synopsys [7] or Mentor Graphics [8] provides verification IPs for more than 40 communication protocols and 60 memory interfaces in order to facilitate SoC verification. Duolog [9] focuses on IP integration problems and generates UVM verification environments from interface-based executable specification. Codasip company [10] provides Codasip Framework that is targeted to ASIP and MPSoC development and offers automated generation of UVM verification environments for these systems that are customized for a class of applications that run on their embedded processor(s).

In our previous work we focused on automated generation of UVM verification environments for ASIPs and MPSoC from their high-level description in *architecture description language* (ADL) [11]. This feature is now integrated into the Codasip Framework. Moreover, we designed an open-source framework HAVEN for FPGA acceleration of simulation-based verification of various systems [12].

Our current research is a continuation of our previous work. We designed and implemented a new feature for automated FPGA prototyping and accelerated verification of ASIPs and MPSoC. We realised that simulation-based verification of ASIPs and MPSoC is valuable, but it runs slowly when we need to evaluate thousands of embedded software applications. Therefore, in the accelerated version of verification we replicate the main principle of HAVEN and move the *Device Under Test* (DUT), which is ASIP or MPSoC, from the software simulation into an FPGA. All other parts such as loading package applications, a running reference model and scoreboarding, remain in the software. If a bug is detected in the accelerated version, we can use the pure software version of the verification environment running in the simulator (the non-accelerated version) for easier debugging of the problem. Another important benefit of using FPGA is that the ASIP prototype will run on real hardware. This helps us to uncover bugs, which are related to the placement of the design into real hardware and which are not detectable in simulation.

Regarding acceleration of verification in general (not necessarily for processors), there already exist some commercial

978-1-4799-6780-3/15 $31.00 © 2015 IEEE
DOI 10.1109/DDECS.2015.33

solution which are quite similar to our work. Mentor Graphics Veloce technology [13] accelerates simulation by synthesising the DUT and placing it into a proprietary emulator. Emulation and acceleration of verification also offers Cadence company in their Cadence Palladium Series [14]. However, our solution is different as it aims exclusively at verification and FPGA-prototyping of ASIPs and MPSoC. But as mentioned above, targeting verification to a specific domain can be much more precise. At the same time, we support system-level verification as we are able to verify not only the hardware architecture of ASIPs, but also various software applications that are executed on them.

This paper is organized as follows. The architecture of the accelerated verification environment is described in Section II. The case study in Section III shows the process of generating the verification environment for a selected ASIP (the accelerated and non-accelerated version) as well as experimental results for different verification runs. Section IV summarizes the results and proposes our plans for future research.

II. Architecture of the Accelerated Environment

In order to get an idea on how the accelerated verification environment may look like, we prepared a simple demonstration model. The verified system (DUT) is a simple MPSoC consisting of two ASIPs: ASIP1 and ASIP2. ASIP1 receives input data and functions as a pre-processor for ASIP2, ASIP2 sends results to its output ports. But of course, DUT can be represented by any other ASIP or MPSoC.

The non-accelerated UVM verification environment in Figure 1 (in SystemVerilog) for the demonstrated MPSoC is generated automatically in Codasip Framework. The accelerated verification environment in Figure 2 is derived from the non-accelerated version. It should be noted that almost all UVM components are moved into the FPGA, except for the Reference Model and Scoreboard. Nevertheless, we aim at designing consistent verification architecture in FPGA too. Therefore, UVM Agents and their inbuilt components are just replaced by HW Agents. We believe that consistent FPGA verification architecture is beneficial not only for the automated generation of the accelerated version, but it also remains understandable for verification engineers.

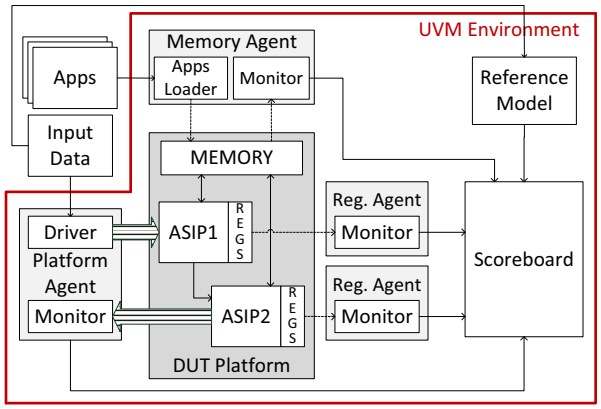


Fig. 1. The architecture of the non-accelerated verification environment.

In the accelerated version, UVM testbench, Reference Model and Scoreboard are running in software simulation and the remaining parts are running in FPGA. Communication between the software and hardware parts of the verification environment is accomplished using the framwork HAVEN (for more details please see [12]). More details about the components of both parts are provided below in the following subsections.

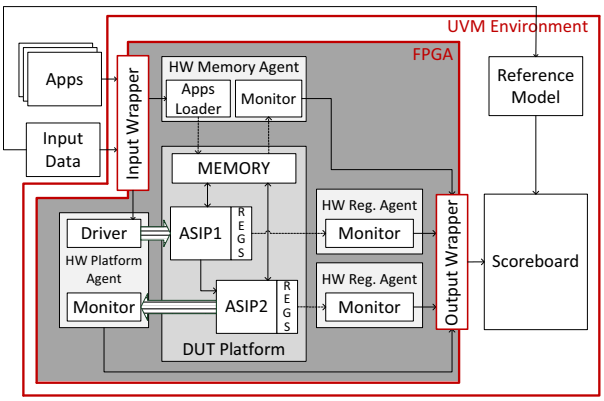


Fig. 2. The architecture of accelerated verification environment.

A. Software Part of the Verification Environment

The main components of the software part are Reference Model and Scoreboard. Reference Model is generated automatically from the high-level specification. Scoreboard compares results of the Reference Model to the results of DUT (received from the hardware part through the Output Wrapper component). In particular, we compare the content of memories and register fields when the specific data set is processed, and we continuously check data from the output ports. The role of Input Wrapper is to send applications (they are loaded to ASIPs and define their functionality) and input data.

The applications are obtained from our designed and implemented stimuli generator which is also at the forefront of our overall research plan. The stimuli generator is especially expected to be used in functional verification. The conception of the stimuli generator is designed for versatile purposes. The aim of the generator is changing its input parameters and achieving its different behavior and thus its different outputs. The generator is based on constraint solving [15] and it takes problem specification as an input. For our purposes of the ASIPs verification, the generator takes the assembly instruction set specification and constraints for this instruction set as the input. Instruction set defines what is to be generated and constraints defines how it has to be generated. When generator is operating, it must deal with numerous conditions and restrictions (constraints). At a processor, it is needed to deal mainly with jump instructions, memory access instructions and latencies for each instruction. Thanks to the constraints, there are reduced possible invalid outputs. The output from the generator is an assembly program (stimulus) which is transformed into machine code and passed into the Input Wrapper. The basic principle of the presented generator are show in Figure 3.

The key part of the generator is the definition of the constraints and their fast interpretation. We defined 20 constraints for valid generating an assembly code. For example absolute jump instructions need 6 types of constraints for ensuring valid and unique label generation in whole program. The generator does not work with semantics of instructions. This allows to focus on more application domains with different stimuli. Therefore the generator is not limited only for processors and can be used in many areas. We are able to generate valid assembly programs for RISC (Reduced Instruction Set Computing) and VLIW (Very Long Instruction Word) processors so far. Some more information about the presented generator is in [16].

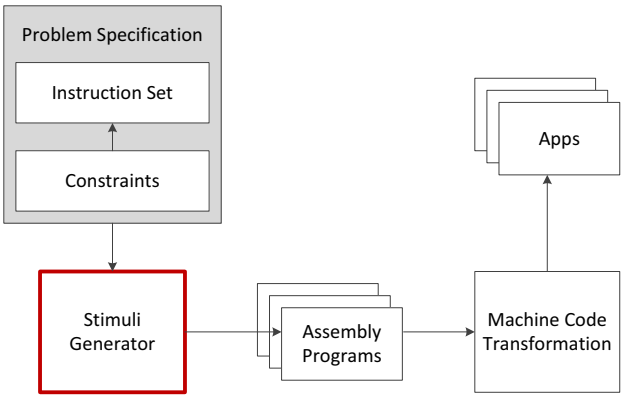


Fig. 3. The basic principle of the generator.

B. Hardware Part of the Verification Environment

Hardware components are currently implemented manually in VHDL. During the following months we plan to generate them automatically in VHDL/Verilog from the high-level specification (similarly as we generate UVM verification environments in the non-accelerated version). A short description of the main hardware components follows.

Hardware Agents are similar to UVM Agents and their main components are Drivers and Monitors. Drivers drive input ports of DUT and Monitors, which on the other hand, collect data from output ports. In Figure 2 you can see the Hardware Memory Agent, two Hardware Register Agents and the Hardware Platform Agent. The Hardware Memory Agent is connected to the main memory. It contains the Driver called the Application Loader that drives the loading of applications into the program part of the memory at the beginning of computation. The second component is the Monitor that takes an image of the memory at the end of computation and sends it to the software Scoreboard for comparison to reference results. The Hardware Register Agent contains only the Monitor that takes an image of register fields at the end of computation and sends it to the software Scoreboard. The Hardware Platform Agent is active during the whole computation; it contains the Driver that during the computation stimulates input ports of ASIP1 with data and Monitor that sends the valid output data of ASIP2 to the software Scoreboard.

III. The Case Study

We performed our experiments and measures with the DUT consisting of one ASIP called Codix RISC [17]. The aim of these experiments was to evaluate and compare the performance of the non-accelerated version of verification to the FPGA-accelerated version.

The non-accelerated verification environment is generated automatically in Codasip Framework. All parts are in SystemVerilog (except of DUT in VHDL or Verilog) and are simulated in Mentor Graphics ModelSim SE-64 10.0c simulator on the server with two quad-core Intel Xeon E5620@2.40 GHz processors and 24 GiB of RAM. The accelerated verification environment contains the DUT, Hardware Platform Agent, Hardware Monitor Agent and Hardware Register Agent on the FPGA site (Xilinx Virtex-5 FPGA) and Scoreboard and Reference Model on the software site (simulated again in ModelSim on the server). The amount of consumed FPGA resources (slices) is the following: 1,428 (5.8%) for the Codix RISC processor and 1,669 (6.9%) for the hardware verification environment.

Results of these experiments are depicted in Table I. We have the measured verification time (the accelerated and the non-accelerated version) for a different number of application programs. Moreover, the acceleration ratio was computed.

TABLE I. The acceleration ratio and the run time of verification.

Number of programs [-]	Run time		Acceleration ratio [-]
	Non-accelerated [s]	FPGA-accelerated [s]	
500	3458	2010	1.719
1,000	6841	3974	1.721
2,000	13634	7917	1.722
4,000	27208	15784	1.723
6,000	40845	23682	1.724
8,000	54384	31451	1.729
10,000	67965	39372	1.726

The measured values are also presented in the graphs of Figure 4 and 5. The acceleration ratio is on average 1,7x and slowly grows up with the number of the evaluated test programs. Because we expected better results, we have performed some specific additional measurements and have identified candidates for further improvements.

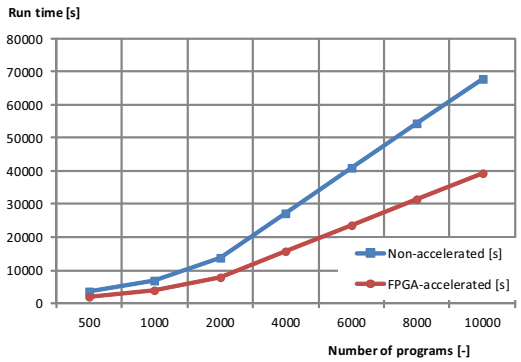


Fig. 4. The relation between the runtime of the non-accelerated and the FPGA-accelerated verification and the number of processor programs.

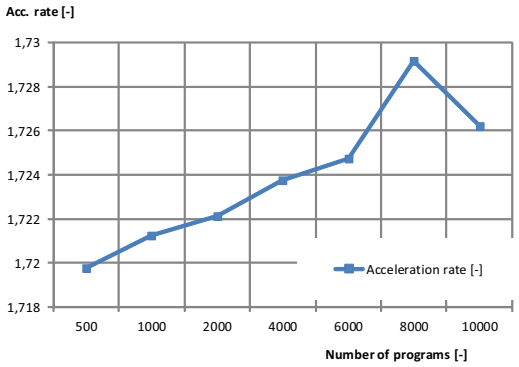


Fig. 5. The relation between the acceleration ratio and the number of processor programs.

The software site of the verification environment is still too complex and the runtime of the accelerated verification depends on the speed of the SW verification environment. The graphical reprezentation of this analysis is shown in Figure 6. The DUT consumes longest time in the non-accelerated verification. But on the other hand, the time consumption of DUT in the FPGA-accelerated verification is shortest. In this case, the runtime of the whole verification is based on the time consumption of the SW verification environment which is the same for both the FPGA-accelerated and the non-accelerated version. There we see a space for possible improvments.

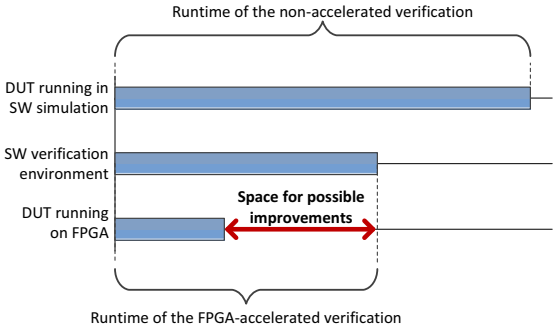


Fig. 6. The graphical representation of the time consuption of the SW verification environment, the simuletd DUT and FPGA-accelerated DUT.

The second problem is that we have executed a huge number of small applications, but only with the basic data sets. For precise verification, every application should be evaluated with more transaction data. The computation burden will be higher, so the acceleration will be more beneficial.

IV. Conclusions and Future Research

In this paper, the environment for FPGA-prototyping and accelerated verification of ASIPs and MPSoC were presented. The case study shows that by means of acceleration on FPGA we are able to detect errors faster (1,7x) and debug not only the software model but directly the hardware prototype on FPGA. As for the great advantage of the accelerated verification environment we see its correspondence to UVM, so it is easily understandable for verification engineers. However, the acceleration ratio was not so good as we expected so we should find a way how to optimize the SW verification environment even more so it will not be a bottlenck in the whole system.

In our future research we intend to interconnect the accelerated verification environment with our fault injector that also operates on FPGA. In this way we will connect our research in the verification area to our research in fault-tolerant systems design [16]. The aim is to create a robust platform for validation of FT methodologies in which the introduced stimuli generator will be also applied.

Acknowledgment

This work was supported by the following projects: National COST LD12036 - ”Methodologies for Fault Tolerant Systems Design Development, Implementation and Verification”, project Centre of Excellence 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] *IEEE Std. 1800-2005, IEEE Standard for SystemVerilog— Unified Hardware Design, Specification, and Verification Language*, 2005. [Online]. Available: http://ieeexplore.ieee.org/xpl/standardstoc.jsp?isnumber=33132&isYear=2005

[2] S. Rosenberg and K. Meade, *A practical guide to adopting the universal verification methodology (UVM)*. Cadence Design Systems, 2013.

[3] R. Backasch, C. Hochberger, A. Weiss, M. Leucker, and R. Lasslop, “Runtime verification for multicore soc with high-quality trace data,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 2, p. 18, 2013.

[4] R. Drechsler, C. Chevallaz, F. Fummi, A. J. Hu, R. Morad, F. Schirrmeister, and A. Goryachev, “Future soc verification methodology: Uvm evolution or revolution?” in *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2014, p. 372.

[5] Breker. (2014, Dec.) Treksoc. [Online]. Available: http://www.brekersystems.com/products/treksoc/

[6] Cadence. (2014, Dec.) Verification IP. [Online]. Available: http://ip.cadence.com/ipportfolio/verification-ip

[7] Synopsys. (2014, Dec.) Verification IP. [Online]. Available: http://www.synopsys.com/Tools/Verification/FunctionalVerification /VerificationIP/Pages/default.aspx

[8] Mentor Graphics. (2014, Dec.) Mentor verification IP. [Online]. Available: http://www.mentor.com/products/fv/verification-ip

[9] Duolog. (2014, Dec.). [Online]. Available: http://www.duolog.com/products/

[10] Codasip. (2014, Dec.) Codasip framework. [Online]. Available: http://www.codasip.com

[11] M. Šimková, Z. Přikryl, Z. Kotásek, and T. Hruška, “Automated functional verification of application specific instruction-set processors,” in *Embedded Systems: Design, Analysis and Verification*. Springer, 2013, pp. 128–138.

[12] M. Šimková and O. Lengál, “Towards beneficial hardware acceleration in haven: evaluation of testbed architectures,” in *Hardware and Software: Verification and Testing*. Springer, 2013, pp. 266–273.

[13] Mentor Graphics. (2014, Dec.) Veloce2. [Online]. Available: http://www.mentor.com/products/fv/emulationsystems/veloce

[14] Cadence. (2014, Dec.) Palladium xp series. [Online]. Available: http://www.cadence.com/products/sd/palladium_xp_series

[15] L. Kotthoff, “Constraint solvers: An empirical evaluation of design decisions,” *CoRR*, vol. abs/1002.0134, 2010. [Online]. Available: http://arxiv.org/abs/1002.0134

[16] 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.

[17] Codasip. (2014, Dec.) Codix RISC. [Online]. Available: https://www.codasip.com/products/codixrisc/

Paper E

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

Jakub Podivinsky, Jakub Lojda, Ondrej Cekan and Zdenek Kotasek

In: Proceedings of the 2018 21st Euromicro Conference on Digital System Design. Praha: IEEE Computer Society, 2018, pp. 229-236. ISBN 978-1-5386-7376-8.

2018 21st Euromicro Conference on Digital System Design

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

Jakub Podivinsky, Jakub Lojda, Ondrej Cekan, Zdenek Kotasek
Faculty of Information Technology, Brno University of Technology, Centre of Excellence IT4Innovations
Bozetechova 2, 612 66 Brno, Czech Republic
Tel.: +420 54114-{1361, 1360, 1361, 1223}
Email: {ipodivinsky, ilojda, icekan, kotasek}@fit.vutbr.cz

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

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

I. Introduction

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

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

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

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

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

It is important to test and evaluate fault-tolerance techniques. Various approaches to the evaluation of fault tolerance exist. Some of them are performed on a theoretical level; for example, a simulation method for SEU emulation is presented in [11]. Another approach is the use of artificial fault injection directly into the design implemented in the FPGA. Special evaluation boards are developed for these purposes; one of them is proposed in [12] and [13]. The combination of simulation method and hardware evaluation is discussed in [14].

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

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

II. The Evaluation Platform and The Evaluation Process

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

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

Our evaluation platform is based on *Functional Verification* [16]. The main task of functional verification is to check whether a verified circuit meets its specifications. It compares the outputs of a verified circuit running in an RTL simulator with those of a reference model. In the case of the fault injection, the verified circuit must be implemented into the FPGA, so we do not use classical simulation-based functional verification, but modified FPGA-based functional verification. Our platform uses functional verification as a tool for monitoring the impacts of faults injected into an electronic controller implemented into the FPGA.

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

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

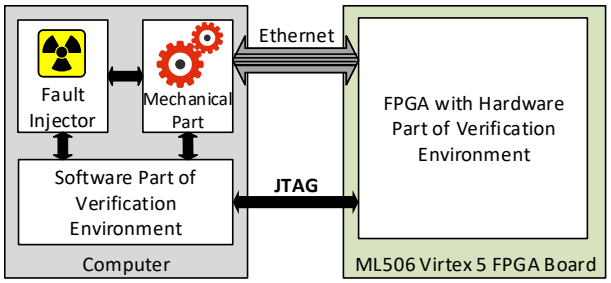


Fig. 1. The architecture of the proposed evaluation platform.

An important metric in functional verification is *coverage*. It measures how well the verification scenarios cover the behavior of the DUT and provide feedback that determines when the verification process can be ended. Depending on the required coverage criteria, the *Code coverage* metrics can serve as an example. *Code coverage* measures how well the verification scenarios cover the source code of the DUT. Typical code coverage metrics are toggle, statement, branch, condition, expression, and FSM coverage.

B. The Three Phases of the Evaluation Process

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

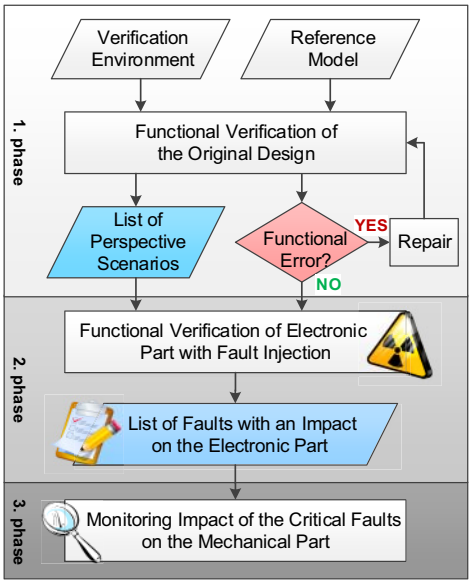


Fig. 2. The flow of phases in the FT evaluation system verification.

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

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

C. Verification Scenario Generation

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

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

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

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

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

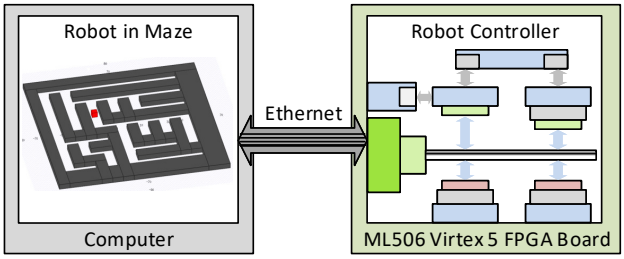


Fig. 3. The robot in the maze and its electronic controller.

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

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

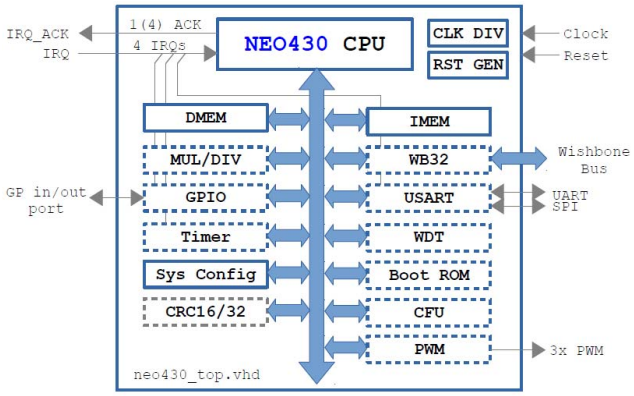


Fig. 4. The architecture of the NEO430 Processor [20].

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

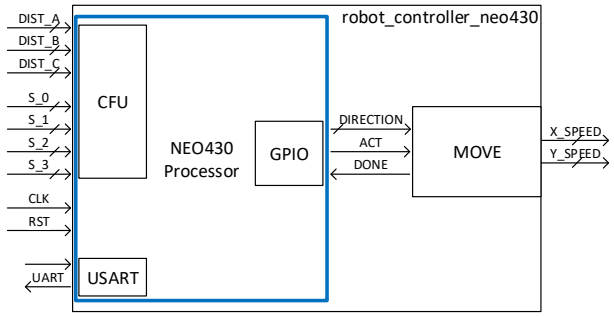


Fig. 5. The architecture of the robot controller composed of the NEO430 Processor.

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

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

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

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

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

IV. Case Study: Experimental Results

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

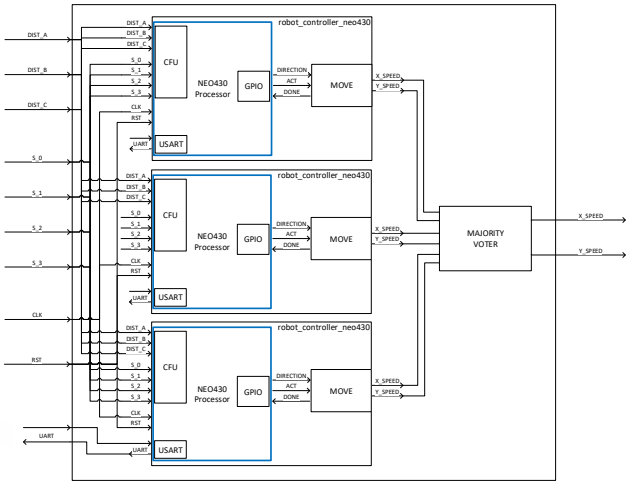


Fig. 6. The architecture of the TMR version of the robot controller composed of NEO430 Processor.

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

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

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

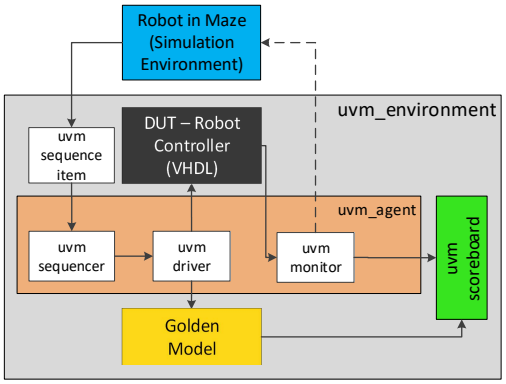


Fig. 7. The architecture of the verification environment for the robot controller.

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

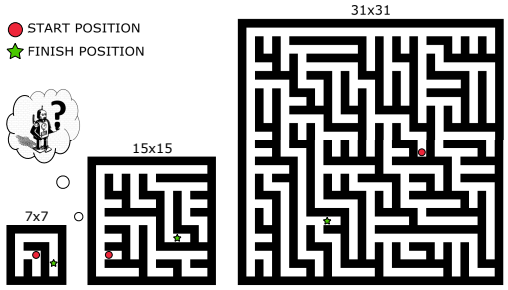


Fig. 8. Three types of mazes - the example of one scenario for each type.

TABLE I. AVERAGE NUMBER OF ROBOT STEPS

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

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

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

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

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

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

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

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

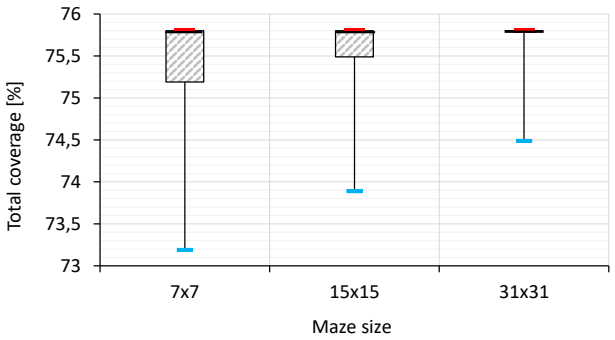


Fig. 9. The box and whisker chart for the selection of the right maze for the robot controller.

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

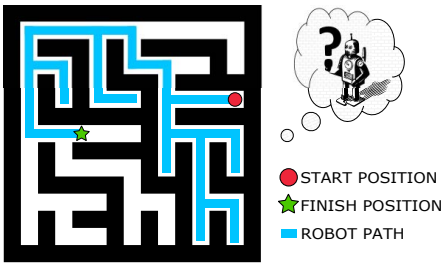


Fig. 10. The selected maze for the robot controller and its path between the start and finish position, which the robot found.

B. The Second and the Third Phases

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

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

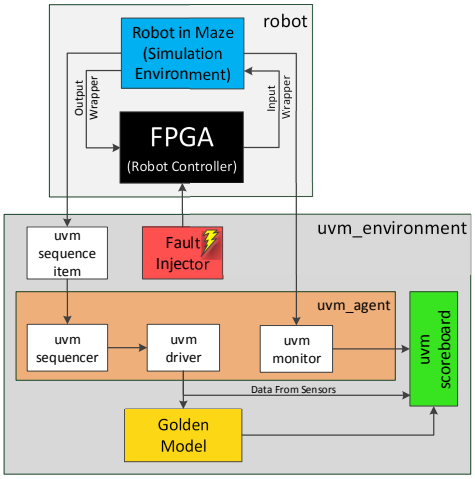


Fig. 11. The architecture of the FPGA-based verification environment for the robot controller.

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

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

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

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

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

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

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

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

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

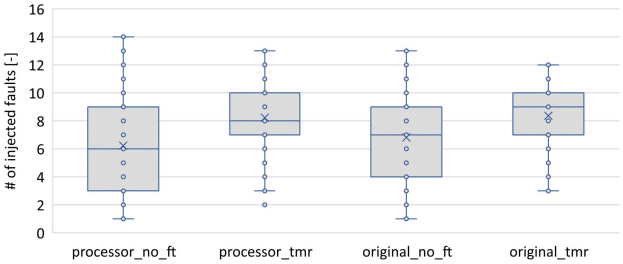


Fig. 12. The box plot shows a statistical comparison of the number of injected faults which led to electronic failure for both the processor-based and the original robot controllers.

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

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

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

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

V. Conclusions and Future Research

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

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

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

Acknowledgements

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II); project IT4Innovations excellence in science - LQ1602 and the BUT project FIT-S-17-3994.

References

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

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

[3] F. L. Kastensmidt, G. Neuberger, L. Carro, and R. Reis, "Designing and Testing Fault-tolerant Techniques for SRAM-based FPGAs," in *Proceedings of the 1st Conference on Computing Frontiers*. ACM, 2004, pp. 419–432.

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

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

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

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

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

[9] D. White, "Considerations Surrounding Single Event Effects in FPGAs, ASICs, and Processors," http://www.xilinx.com/support/documentation/white_papers/wp402_SEE_Considerations.pdf, Mar. 2012, accessed: 2016-09-15.

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

[11] C. Bernardeschi, L. Cassano, A. Domenici, and L. Sterpone, "Accurate Simulation of SEUs in the Configuration Memory of SRAM-based FPGAs," in *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 115–120.

[12] M. Alderighi, S. D'Angelo, M. Mancini, and G. R. Sechi, "A Fault Injection Tool for SRAM-based FPGAs," in *On-Line Testing Symposium, 2003. IOLTS 2003. 9th IEEE*. IEEE, 2003, pp. 129–133.

[13] M. Alderighi, F. Casini, S. d'Angelo, M. Mancini, S. Pastore, and G. R. Sechi, "Evaluation of Single Event Upset Mitigation Schemes for SRAM-based FPGAs Using the FLIPPER Fault Injection Platform," in *Defect and Fault-Tolerance in VLSI Systems, 2007. DFT'07. 22nd IEEE International Symposium on*. IEEE, 2007, pp. 105–113.

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

[15] J. Podivinsky, O. Cekan, J. Lojda, M. Zachariasova, M. Krcma, and Z. Kotasek, "Functional Verification based Platform for Evaluating Fault Tolerance Properties," *Microprocessors and Microsystems*, vol. 52, pp. 145 – 159, 2017.

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

[17] M. Straka, J. Kastil, and Z. Kotasek, "SEU Simulation Framework for Xilinx FPGA: First Step Towards Testing Fault Tolerant Systems," in *14th EUROMICRO Conference on Digital System Design*. IEEE Computer Society, 2011, pp. 223–230.

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

[19] B. Gerkey, R. T. Vaughan, and A. Howard, "The Player/Stage Project: Tools for Multi-robot and Distributed Sensor Systems," in *Proceedings of the 11th international conference on advanced robotics*, vol. 1, 2003, pp. 317–323.

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

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

Paper F

Reliability Analysis and Improvement of FPGA-based Robot Controller

Jakub Podivinsky, Jakub Lojda, Ondrej Cekan, Richard Panek and Zdenek Kotasek

In: Proceedings of the 2017 20th Euromicro Conference on Digital System Design. Vienna: IEEE Computer Society, 2017, pp. 337-344. ISBN 978-1-5386-2146-2.

2017 Euromicro Conference on Digital System Design

Reliability Analysis and Improvement of FPGA-based Robot Controller

Jakub Podivinsky, Jakub Lojda, Ondrej Cekan, Richard Panek, Zdenek Kotasek
Faculty of Information Technology, Brno University of Technology, Centre of Excellence IT4Innovations
Bozetechova 2, 612 66 Brno, Czech Republic
Tel.: +420 54114-{1361, 1360, 1361, 1362, 1223}
Email: {ipodivinsky, ilojda, icekan, ipanek, kotasek}@fit.vutbr.cz

***Abstract*—Faults occurring in the safety-critical systems can lead to the failure of the whole system and cause high economical losses or endanger human health. As an example, space, aerospace or medical systems which are working in the environment with increased occurrence of faults can serve. Fault avoidance and fault tolerance are the main techniques, the goal of which is to avoid such situations. This paper is the continuation of the previously published work and presents an approach to evaluate fault tolerance techniques by monitoring the impact of faults in the experimental electro-mechanical system which consists of the robot in a maze and its robot controller. The experiments with the robot controller hardened against faults are combined with the reliability analysis on a theoretical level in this paper. The impact of faults artificially injected into the robot controller, in which Triple Modular Redundancy is applied, is monitored and used for statistic reliability analysis.**

***Keywords*—Reliability Analysis, TMR, FPGA, Fault Tolerance, Robot Controller, Reconfiguration.**

I. Introduction

Various electronic systems play an important role in our everyday lives. We can meet them in various types of commonly used devices such as cars, intelligent buildings, or some entertainment systems. For example, electronic systems make our lives easier, supervise our health or provide new opportunities. The reliability of these systems is a problem, especially in the case of systems in which failure can result in injury or heavy financial losses or can endanger human health. One of the reasons which leads to a higher susceptibility to faults is an increase of chip-level integration. The current trend is to make electronic systems smaller and integrate more functionality to smaller area on the chip which leads to greater sensitivity to faults. The number of digital systems with high demand on reliability, such as medicine, space, industry, is growing as well.

Two main approaches to increase reliability are currently used. The first one is called *fault avoidance* [1]. As the name indicates, the main goal is to completely avoid failures in the system using the means of more reliable parts, manufacturing processes, etc., which is very challenging and expensive.

The second approach is a technique called *fault tolerance* [2]. Fault tolerance accepts the fact a fault can appear, but the goal of this approach is to keep the system functional, even in the presence of faults. Techniques based on the various types of redundancy are used for this purpose. The most common ones are hardware and time redundancy. Hardware redundancy usually uses n-copies of the same functional unit and comparator to guarantee the proper function. On the other hand, time redundancy is based on computation repeating and the results from the independent runs are then compared. Many fault tolerance methodologies exist, which combine and improve these basic methods, e.g. hardware and time redundancy is combined in the approach presented in [3].

There have been many fault-tolerant methodologies inclined, among others, to *Field Programmable Gate Arrays* (FPGAs) developed and new ones are under investigation [4], because FPGAs are becoming more popular due to their flexibility and re-configurability. The second reason why so many techniques are inclined to FPGAs is their sensitivity to faults and ability to be reconfigured in the case of fault occurrence. FPGAs are composed of configurable logic blocks [5] which are connected by programmable interconnection. The configuration is stored as a *bitstream* in SRAM memory. The problem from the reliability point of view is that FPGAs are quite sensitive to faults caused by charged particles [6]. This particle can induce inversion of a bit in bitstream and this may lead to a change in its behaviour. This event is called *Single Event Upset* (SEU) [7]. The advantage is that faults which occurred in configuration memory can be repaired by *Partial Dynamic Reconfiguration* (PDR) [8].

It is important to test and evaluate these techniques. Various approaches to the evaluation of fault tolerance exist, some of them are performed on a theoretical level, for example, a simulation method for SEU emulation is presented in [9]. Another approach is in the use of fault injection directly into the design implemented in FPGA. Special evaluation boards are developed for these purposes, one of them is presented in [10] or [11]. The evaluation of fault tolerance techniques is one of the goals of our research, in our previously published work where we develop the platform for experimental evaluation of the impact of faults that occurred in an experimental system [12]. Our evaluation platform was tested and demonstrated on the experimental electro-mechanical system (a robot in a maze and its controller) without any hardening against faults. *The next step in our experiments is to apply a fault tolerance technique on our experimental system and evaluate it experimentally which is the main topic of this paper. We feel that only a experimental evaluation is not good enough so the theoretical reliability analysis based on experimental results is also mentioned in this paper.*

This paper is organized as follows. Section II introduces reliability analysis and improvement. An evaluation platform and experimental system on which reliability is experimentally evaluated is presented in Section III. Verification scenarios generation presented in Section IV is the important part of evaluation platform. Section V is dedicated to reliability

978-1-5386-2146-2/17 $31.00 © 2017 IEEE
DOI 10.1109/DSD.2017.15

analysis based on experimental evaluation. The possibility of using a dynamic reconfiguration for the faulty module recovery is shown in Section VI. Section VII concludes the paper and presents future plans of our research.

II. Reliability Analysis and its Improvement

A fault-tolerant system development usually starts with a *nondurable* system that does not tolerate faults [13]. This *nondurable* system is usually designed with minimum redundancy and serves as a starting point for the process of development. An experienced fault-tolerant system designer then suggests the modifications that are to be made to the *nondurable* system in order to to achieve a higher level of fault tolerance. After these changes are incorporated into the design, the result must be evaluated to be sure the applied redundancy has the desired improvement on the reliability of the system. The usual approach is to iterate between the phases of development and the reliability analysis. Multiple designs with various combinations of fault tolerance methods assigned to the partitions of the design are created. The system development ends either with the system complying with the specification or the findings of the specifications not being achievable. In this research we try to accelerate this procedure of the development with an ability to estimate the reliability of the resulting system even *before* the application of the method itself. This allows for a designer to exclude such combinations of reliability methods that do not look perspective from the final specification needs point of view. The final specification usually contains a list of so-called *reliability indicators* and the corresponding ranges of values that must be achieved in order to accept the resulting solution.

A. Reliability Analysis

The reliability itself can be quantified with the support of the *theory of probability* as most of the *reliability indicators* are of a random nature. The length of a time period of the system operation until the failure occurs is an important starting point in the *reliability indicators* computation. This variable can be considered the so-called *random variable*. The simplified definition of *random variable* according to [14] is shown in Definition 1.

Definition 1: Random variable X on a sample space S is a function $X : S \to \mathbb{R}$ that assigns a real number $X(s)$ to each sample point $s \in S$.

The *Cumulative Distribution Function* (CDF) is an important concept of studying *random variables*. A CDF expresses a probability the *random variable* takes a value lower than a given non-negative real number t which is defined in Equation 1. The CDF is a *nondecreasing function*.

$$F(t) = \mathcal{P}(\tau < t) \tag{1}$$

1) Failure Function: If a *random variable* τ expresses a length of a time interval from the systems start of the operation to the point a fault occurs, then the CDF $F(t)$ of *random variable* τ expresses a probability of the system being in a failure state at the time t. In this case, the CDF $F(t)$ is denoted as $Q(t)$ and is called the *failure function*.

2) Reliability Function: Another *reliability indicator* is the so-called *reliability function* which is denoted as $R(t)$. The *reliability function* expresses a probability of the system being in an fault-less state at the time t and it is a supplement of the $Q(t)$ as expressed in Equation 2.

$$R(t) = 1 - Q(t) \tag{2}$$

3) Failure Density: The *failure density* $f(t)$ is defined by the time derivative of a CDF $Q(t)$ if the *random variable* is continuous and the derivative exists, as shown in Equation 3.

$$f(t) = \frac{\mathrm{d}Q(t)}{\mathrm{d}t} \tag{3}$$

The product of $f(t)\mathrm{d}t$ then expresses the probability of a fault occurrence for a short period of time $\mathrm{d}t$ that is immediately following after the time t. Although, the case in which the fault occurred earlier before the time t is not taken into account.

4) Failure Rate: The next *reliability indicator* is *failure rate* which is denoted by $\lambda(t)$. The *failure rate* expresses a conditional *failure density* at the time t assuming the failure has not occurred yet. Equation 4 gives a relationship between the $\lambda(t)$ and $Q(t)$.

$$\lambda(t) = \frac{f(t)}{R(t)} = \frac{f(t)}{1 - Q(t)} \tag{4}$$

Again, the product $\lambda(t)\mathrm{d}t$ gives a probability of a fault occurrence for a short period of time $\mathrm{d}t$ immediately following after the time t given the examined system was in a *fault-less* state at the time t.

Throughout the whole lifetime of the system, the $\lambda(t)$ usually forms the so-called *bathtub curve*. The *bathtub curve* can be divided into three main time periods with the first one related to early failures, the middle one related to a constant failure period and the last one corresponding with the wear-out phase [15]. The time interval $\langle t_1, t_2 \rangle$ in which the *failure rate* keeps an approximate constant value is the most important part for us, as this is the part where the random failures *rate* is revealed.

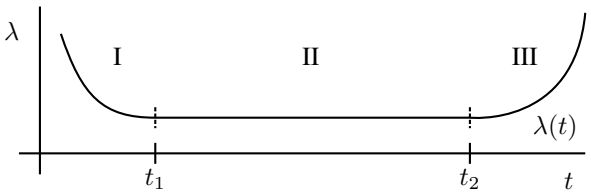


Fig. 1. The usual *failure rate* displayed as a function of a time *t*, the so-called *bathtub curve*.

5) Mean Time To Failure: The *Mean Time To Failure* (MTTF) which is in the following text denoted as T_s represents a mean value of the random variable τ observed. The mean value can be seen as a mean time of all the time period lengths since the system started its operation to the first failure occurrence. If the mentioned system is *non-recoverable*, the value can be considered a *mean time to the first failure* as well. To calculate the T_s, the Equation 5 can be used.

$$T_s = \int_0^{\infty} R(t)\mathrm{d}t \tag{5}$$

B. Reliability Improvement

The reliability improvement can be achieved through several means, all of which are based on the concept of *redundancy*. The concept of *time redundancy* is based on increasing the time spent by a particular computation. Another concept of reliability improvement is to utilize redundant information. The *information redundancy* is based on *Error Detection And Correction* (EDAC) codes. Although all of the means of redundancy are closely related, the most fundamental principle is to utilize the *hardware redundancy*.

Probably the most known principle of the *hardware redundancy* is the *Triple Modular Redundancy* (TMR, 3MR) which is based on a *triplication* of the component we intend to improve reliability of [16]. The TMR is based on a *static backup* using three *functionally* and *structurally* equivalent elements. The structural schematic can be seen in Figure 2. The two additional copies of the original functional unit are incorporated into the system. The resulting units are named F_1, F_2 a F_3. The vectors of the input signals x are connected in such a way that each of the functional units F_i works with the same input values. The output signals $f_i(x)$ are connected to the inputs of the so-called *voter* which implements the so-called *majority function*. The *majority function* can be implemented in different ways, the selection of the majority can be performed on the level of bits, whole vector, etc. It is important to note that in the following text, the TMR version using a *voter* that works on the per-bit basis is consulted. The *voter* is built with the use of n instances of the so-called *majority gate* with the n equal to the number of output bits of F_1 (which is the same for all the F_i units).

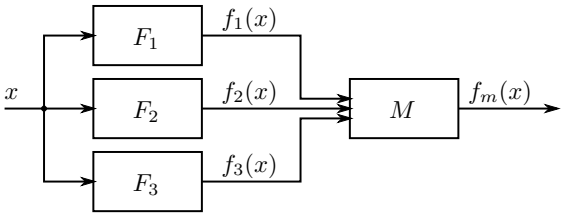


Fig. 2. A system module whose reliability was improved according to the TMR method.

If we omit some edge cases, such as the case when the faults compensate each other, it can be declared that this implementation allows us to mask the failure of one module. If we suppose that each of the F_i modules has an equivalent reliability function $R(t)$, then Equation 6 that can be used to evaluate the resulting reliability function of the whole TMR module exists.

$$R_{TMR}(t) = 3[R(t)]^2 - 2[R(t)]^3 \tag{6}$$

The TMR method is useful for tasks that last for a short period of time where an improvement of the reliability function is desired. For longer lasting tasks an option for faulty modules *recovery* can be added. An overview of the reliability indicators of the TMR systems is shown in Table I [13], [16].

TABLE I. AN OVERVIEW OF THE TWO MAIN RELIABILITY INDICATORS OF THE TMR SYSTEMS.

Reliability indicator	Input variables		Value
Reliability Function $R_{TMR}(t)$	R(t)	R(t) of the original functional unit	$R_{TMR}(t) = 3[R(t)]^2 - 2[R(t)]^3$
Mean Time To Failure $T_{s(TMR)}$	λ	λ of the original functional unit	$T_{s(TMR)} = \frac{5}{6\lambda}$

III. Evaluation Platform and Robot Controller Case Study

The development of the experimental system and the evaluation platform for monitoring faults injected into FPGA-based system was the scope, among other, of our previous work [12]. A created experimental system can serve as a case study for a discussed techniques demonstration which is also in this paper. Because the digital controlling systems very often control some mechanical part, we decided to create an evaluation platform which can use an electro-mechanical application as an experimental system. It can be stated that such areas exist in which electro-mechanical applications are implemented as fault-tolerant - aerospace and space applications can serve as an example.

A. The Robot Controller - Experimental Electro-mechanical System

Our experimental electro-mechanical system consists of a robot for searching a path through a maze and its electronic controller implemented in FPGA. Unfortunately, we do not have a real robot device, so we use the simulation tool Player/Stage [17] which allows us to simulate the robot and its environment (in our case the robot in a maze). The robot simulation is executed on a computer which is connected with the FPGA board by the Ethernet (see Figure 3) interface through which data between the robot and its controller are transmitted. The robot controller is composed of various functional units which are interconnected through the central bus. There is in total 16 functional units, the main ones are the Position Evaluation Unit (PEU) and the Barrier Detection Unit (BDU) which calculate actual position of the robot in a maze and detect barriers in robot 4-neighborhoods. The Map Unit (MU) stores calculated informations into the Map Memory Unit (MMU) on which path searching realized by Path Finding Unit (PFU) is based. Mechanical parts of the robot are controlled by Engine Control Unit (ECU). Almost all of these functional units are equipped with a control finite stat machine (FSM) and a bus wrapper.

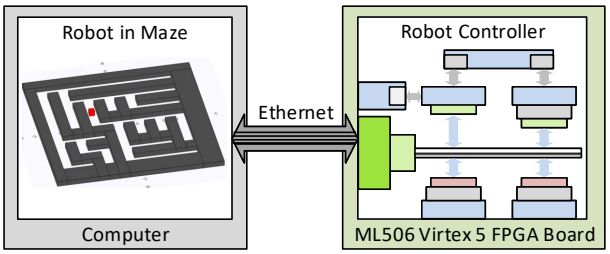


Fig. 3. The robot in maze and its electronic controller.

B. The Evaluation Platform for Monitoring Impact of Faults on Electro-mechanical System

The evaluation platform is based on *Functional Verification* [18]. The main task of the functional verification is to check if

a verified circuit satisfies its specification. It compares outputs of a verified circuit running in a RTL simulator with a reference model implemented in another programming language (e.g. C/C++). In the case of the fault injection, the verified circuit must operate in FPGA, so we do not use classical simulation-based functional verification, but modified FPGA-based functional verification. Our platform uses functional verification as a tool for monitoring impacts of faults injected into electronic controller implemented into the FPGA.

The two main parts of the implemented evaluation platform (see Figure 4) are a computer and an FPGA development board. It allows us to implement a verified electronic controller in FPGA and inject faults directly into FPGA. The fault injector is a component which runs on the computer. Our fault injector [19] is based on the partial reconfiguration. It reads part of the configuration bitstream from the configuration memory, then the required number of specified bits of the bitstream are inverted and a modified bitstream is configured back to the configuration memory through the JTAG interface. The platform is designed to evaluate the impact of faults on the electro-mechanical application, so the simulation of the mechanical part is important and is also runned on the computer. The simulation of the mechanical part is connected with FPGA through the Ethernet interface. The software part of verification environment is also runned on the computer and performs the evaluation of impacts of injected faults on both the electronic and mechanical parts.

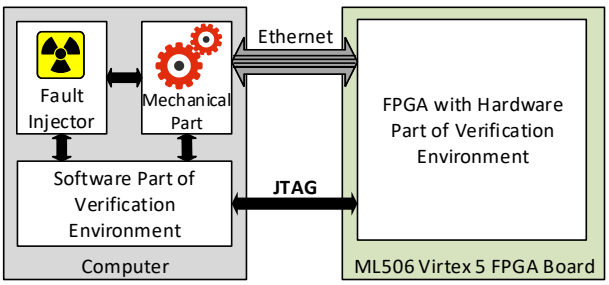


Fig. 4. The architecture of our evaluation platform.

An important metric in functional verification is the *coverage*. It measures how well input stimuli cover the behavior of DUT and provide the feedback that determines when the verification process can be ended. Depending on the required coverage criteria, the *Code coverage* metrics can serve as an example. *Code coverage* measures how well input stimuli cover the source code of DUT. Typical code coverage metrics are toggle, statement, branch, condition, expression or FSM coverage.

IV. Verification Scenarios Generation

Input stimuli are values on the input of the electronic controller on which output values are based on. In the case of the robot controller, input values are changed during the evaluation of one verification scenario (maze, start and goal position). High code coverage was achieved by the set of verification scenarios in our previous work. Actual experiments are simplified because we are using only one verification scenario (one maze) which proposes sufficient code coverage and a suitable number of steps from start to goal position. The principles of finding such a scenario are described in the following text.

In our previous research [12], we used a set of different verification scenarios (mazes) for monitoring the impact of faults on the robot controller. These mazes achieved sufficient code coverage for the verification of the proper function of the robot controller, but the evaluation was very time consuming. The reason was the execution of a large number of experiments which monitors the influence monitoring on the mechanical part of the robot that simulates path finding in a maze. For these experiments, a set of mazes with size 15x15 cells was used. The set reached the maximal code coverage 91.85%. To accelerate the experiments presented in this paper and also future experiments, we build on the premise of finding the one ideal maze (including the start and goal position of the robot) to ensure the correct behavior of the robot controller and to achieve the previously reached maximal coverage. The found mazes should contain a reasonable number of the robot steps from the start to goal position in order to avoid insignificant prolongation of the experiments. Finding the ideal maze and its positions can be divided into three steps - maze generation, maze selection based on the coverage, and maze selection with the optimal number of steps.

A. Maze Generation

We constantly improve and generalize our test stimuli generator which is based on our designed universal architecture. This architecture allows us to describe the desired test stimuli through two specific input structures. The first structure is used to describe the format of the stimuli, while the second structure defines restrictive conditions on how the format has to be constructed into a valid stimulus. In our previous research, we used to define these structures by using our description (language). The language was not general, therefore, a special functionality to provide a valid stimulus had to be implemented with every new type of supported system. We have managed to generalize the test stimuli generator while maintaining the defined universal architecture. Instead of our own language, probabilistic context-free grammar (PCFG) was used. PCFG was extended by restrictive conditions which dynamically adjust the probabilities for rewriting the rules of grammar during the test stimuli generation. In this way, we define a new grammar the expressive power of which is much higher. More information about PCFG extended with constraints may be found in [20].

Using this new grammar, we are also able to describe and generate a maze, where a way between any two points exists. To generate the maze, we use the principles of the binary tree algorithm which can be encoded into our grammar. The basic principle of the binary tree algorithm is shown in Figure 5. It starts from the basic matrix of the maze (a) in which some cells are tightly specified - either a corridor or a wall. The corridors are represented by a white color and walls by a black color. Cells marked with a question mark represent areas that can take the white or black color. In order to maintain the continuity from any corner of the maze to another, it is necessary to perform a modification of the basic matrix of the maze so that each two adjacent sides of the maze must contain the corridor over its entire dimension (b). In our case, we chose this corridor to the northern and the western side of the maze.

The final and most critical task is to determine cells A, B, C, D which allows us to have the maximal continuous maze (c). If cell A, respectively C, was randomly selected for the corridor in Figure 5.b, then cell B, respectively D, will be a wall and vice versa. Such mazes may be generated directly into the binary file (picture) in Bitmap format (BMP), including start and goal positions which can be differentiated by a color.

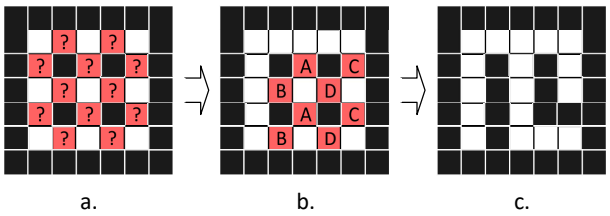


Fig. 5. The demonstration of a conversion of the basic matrix of the maze for needs of the generator.

For the selection of the one maze with maximal coverage, we have generated 300 different mazes with dimensions of 7x7, 15x15, and 31x31 cells by using the described approach. The mazes also have different start and goal positions. The coverage of individual mazes and the differences between different dimensions are shown in Figure 6 with a box plot graph. The upper dash shows the maximal achieved coverage and the lower dash shows the minimal achieved coverage for the set of mazes. The inner line in the chart represents the median value of the coverage. The last shown range defines the first (25% of all values) and third (75% of all values) quartiles. In the figure, it can be seen that with increasing the size of the maze, accomplishment of the maximal coverage occurs more frequently due to the execution of more steps of the robot during the path finding in the maze. It is also evident that some mazes with dimensions 15x15 and 31x31 cells are able to achieve maximal coverage. Therefore, we calculated the average number of steps of the robot for each size of the maze - 13 steps for dimensions of 7x7, 125 steps for 15x15, and 539 steps for 31x31 cells. Based on this information, we have chosen the maze with dimensions of 15x15 cells with maximal coverage 91.85% for further experiments. The inability of achieving an ideal of 100% is caused by the default branches in the source code which are never executed (which is correct), and also by some of the control expressions that are used only when an abnormal situation occurs (e.g. a fault).

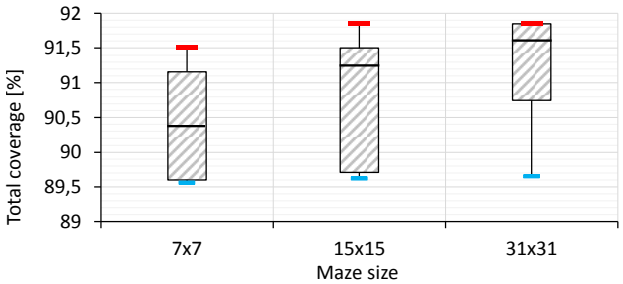


Fig. 6. The Box and whisker chart helps select the right one maze for the robot controller with maximal total coverage in functional verification.

B. Maze Selection With the Optimal Number of Steps

The final step is to select the one maze with dimensions of 15x15 cells which has the optimal number of steps of the robot from the start to the goal position. This condition is important because a maze with the long way has the same coverage as the maze with the short way, but multiple steps do not bring any profit and just prolong the time to perform the experiments. On the other hand, the short mazes can cause a problem with the detection of a fault which may not occur in a short time. Among our test set of generated mazes with dimensions of 15x15 cells, we chose the maze with the maximal coverage of 91.85% and with the number of steps equal to 51 which is an optimal number from our point of view. The selected maze, including start and goal positions, and the way that the robot must follow, is shown in Figure 7.

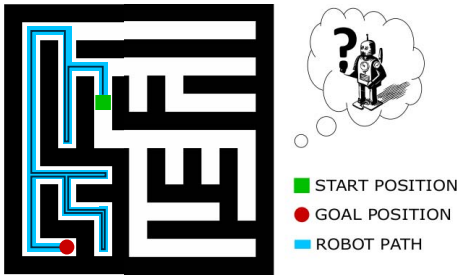


Fig. 7. The selected maze for the robot controller and its path between the start and goal position which the robot found.

V. Reliability Analysis with Experimental Evaluation

The subject of this research is to evaluate some of the basic *reliability indicators* of the robot controller unit which had already been used in our previous research presented in Section III and its fault-tolerant version that is implemented the TMR method. The other subject of this research is to find out if the process of the evaluation technique of the fault-tolerant robot controller could be estimated by an analysis of the *nondurable* robot controller system followed by an application of equations to convert the *reliability indicators* to estimate the indicators of the fault-tolerant controller.

A. Data Acquisition

At first, to be able to practically evaluate the parameters of the robot controller and to verify that the equation mentioned is applicable in our concept, two robot controller versions were created. The first version that is referenced to as to the *noft* version is actually the initial *nondurable* system we started with. The *noft* version includes one component only – the instance of robot controller alone named the *rc*. The second version that is labeled the *tmr* was implemented using three instances of the *rc* robot controller component from the previously mentioned *noft* version and a voter. The components are named *rc1*, *rc2* and *rc3*. It is important to note the voter itself was not subject to evaluation. The resources consumed for both of the versions are shown in Table II.

TABLE II. Resources consumed for both versions of the robot controller units.

Version	Occupied slices [–]	Slice reg. [–]	Slice LUTs [–]	Max freq. [MHz]	LUT bits used [–]
noft	1080	1617	1708	93.76	108480
tmr	2991	4755	5165	93.76	329824

For the practical evaluation of these two robot controller versions, *verification environment* presented in Section III was

used. The fault injection our *verification environment* utilizes was set up with a constant SEU injection rate. The SEUs were injected to the bits of the bitstream that are utilized in the design and represent the content of *Look-Up Tables* (LUTs) at the same time. If we suppose that component c is subject to SEU fault injection, then the important parameter to unambiguously describe this type of fault injection is a time delay d_c between two consecutive SEUs injected to c. The d_c actually does not necessarily have to be constant for the whole time of the SEU simulation, it can be represented by a *random variable* with a particular *probability distribution*. In our experiments, we have experimentally chosen the d_c to be described by the *uniform distribution* with a *mean* value of 12 s and a *variance* of 2 s.

The scenario of one verification run was as follows:

1) the robot controller unit was configured into its initial state, the maze map as well as its starting and target positions were the same for all the verification runs,
2) the *Player/Stage* simulation environment was started, the robot was placed on the starting position,
3) after 15 s, for each component c, the SEU injection started with the d_c time period between SEUs based on the *uniform distribution* with the *mean* value of 12 s and *variance* of 2 s, the bits to inject SEU to were selected *uniformly at random*,
4) mainly, the time from the robot start to the first failure observed was monitored, moreover, the ability of the robot to reach the target position was observed as well.

This verification scenario was repeated 3500 times for both of the two versions of the robot controller units. The data acquired included the time of the first failure occurrence and information on whether the robot successfully reached the target position.

B. Method of Reliability Indicators Calculation

The data obtained from the previously described experiments were then processed. The *multi-set* of all the times measured from the start of the operation of the system to the first detection of an error on the system outputs was transformed to a discrete *failure function* $Q(t)$ which was then converted to the *reliability function* $R(t)$. The other *reliability indicators* included are the *failure density* $f(t)$ and *failure rate* $\lambda(t)$.

All the data for the *noft*, *tmr* robot controller unit versions and the estimation of parameters for the *tmr* version are discretized with the time-step of 15 s in Table III. The rows that are marked with *est.* contain the estimations of the *reliability function* for the given time-steps calculated using Equation 6. From there, the other reliability indicators (*failure function*, *failure density* and *failure rate*) were calculated using Equations 2, 3 and 4 from Section II respectively. The final values of the *reliability functions* on the bottom of Table III are not close to the limit value of zero, as in most cases the faults injected did not appear in the form of an error on the outputs of the robot controller unit. The threshold time length the robot had to find its path within was evaluated to 204 s, that is also the maximum time for which the system has been verified.

TABLE III. A DISCRETIZATION OF THE MEASURED RELIABILITY PARAMETERS OF THE *noft* AND *tmr* ROBOT CONTROLLER UNITS WITH THE ESTIMATION OF THE PARAMETERS FOR THE *tmr* ROBOT CONTROLLER.

Time t [s]		First err. detect. [−]	First err. detect. [%]	$Q(t)$ [−]	$R(t)$ [−]	$f(t)$ [−]	$\lambda(t)$ [−]
$0-14.\bar{9}$	*noft*	0	0.0%	0.00	1.00	0.0000	0.0000
	tmr	0	0.0%	0.00	1.00	0.0000	0.0000
	est.	–	0.0%	0.00	1.00	0.0000	0.0000
$15-29.\bar{9}$	*noft*	6	0.2%	0.00	1.00	0.0001	0.0001
	tmr	1	0.0%	0.00	1.00	0.0000	0.0000
	est.	–	0.0%	0.00	1.00	0.0000	0.0000
$30-44.\bar{9}$	*noft*	9	0.3%	0.00	1.00	0.0002	0.0002
	tmr	0	0.0%	0.00	1.00	0.0000	0.0000
	est.	–	0.0%	0.00	1.00	0.0000	0.0000
$45-59.\bar{9}$	*noft*	35	1.0%	0.01	0.99	0.0007	0.0007
	tmr	8	0.2%	0.00	1.00	0.0002	0.0002
	est.	–	0.0%	0.00	1.00	0.0000	0.0000
$60-74.\bar{9}$	*noft*	28	0.8%	0.02	0.98	0.0005	0.0005
	tmr	25	0.7%	0.01	0.99	0.0005	0.0005
	est.	–	0.0%	0.00	1.00	0.0000	0.0000
$75-89.\bar{9}$	*noft*	8	0.2%	0.02	0.98	0.0002	0.0002
	tmr	11	0.3%	0.01	0.99	0.0002	0.0002
	est.	–	0.0%	0.00	1.00	0.0000	0.0000
$90-104.\bar{9}$	*noft*	47	1.3%	0.04	0.96	0.0009	0.0009
	tmr	53	1.5%	0.03	0.97	0.0010	0.0010
	est.	–	0.2%	0.00	1.00	0.0001	0.0001
$105-119.\bar{9}$	*noft*	42	1.2%	0.05	0.95	0.0008	0.0008
	tmr	36	1.0%	0.04	0.96	0.0007	0.0007
	est.	–	0.2%	0.00	1.00	0.0001	0.0001
$120-134.\bar{9}$	*noft*	52	1.5%	0.06	0.94	0.0010	0.0011
	tmr	34	1.0%	0.05	0.95	0.0006	0.0007
	est.	–	0.2%	0.01	0.99	0.0002	0.0002
$135-149.\bar{9}$	*noft*	36	1.0%	0.08	0.92	0.0007	0.0007
	tmr	47	1.3%	0.06	0.94	0.0009	0.0010
	est.	–	0.4%	0.01	0.99	0.0003	0.0003
$150-164.\bar{9}$	*noft*	42	1.2%	0.09	0.91	0.0008	0.0009
	tmr	33	0.9%	0.07	0.93	0.0006	0.0007
	est.	–	0.3%	0.01	0.99	0.0002	0.0002
$165-179.\bar{9}$	*noft*	50	1.4%	0.10	0.90	0.0010	0.0011
	tmr	48	1.4%	0.08	0.92	0.0009	0.0010
	est.	–	0.6%	0.02	0.98	0.0004	0.0004
$180-194.\bar{9}$	*noft*	45	1.3%	0.11	0.89	0.0009	0.0010
	tmr	46	1.3%	0.10	0.90	0.0009	0.0010
	est.	–	0.7%	0.03	0.97	0.0004	0.0004
$195-209.\bar{9}$	*noft*	856	24.5%	0.36	0.64	0.0163	0.0254
	tmr	787	22.5%	0.32	0.68	0.0150	0.0221
	est.	–	21.8%	0.25	0.75	0.0146	0.0193

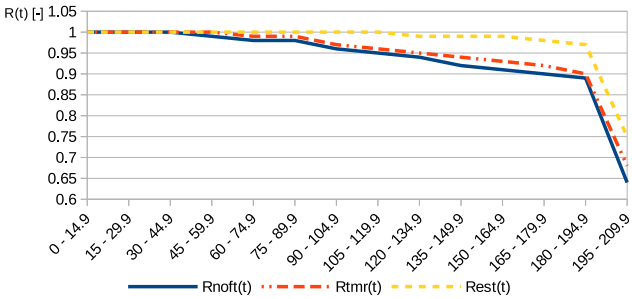


Fig. 8. An experimental evaluation of the measured results of the *reliability function* for the *noft* and the *tmr* versions.

The results of our research can be seen from two different points of view. The first point of view is from the improvement of the reliability by the TMR application. As can be seen in the chart on Figure 8, it is evident that the implementation of the TMR method improved the reliability of the robot controller unit in each of the time-steps we used to discretize the calculation. As a known fact, the chart shows the TMR without faulty modules *recovery* is useful mainly for shorter mission times. From the second point of view, there is a considerable difference between the estimated and measured parameters for the *tmr* version of the robot controller unit. One reason we believe could possibly cause this phenomenon

is the fact the evaluation was done with the fault injections to the LUTs only. Another consideration includes a possible problem within the method of the calculation. In order to find out the true reason for this, disproportion is seen by us as a good direction for future research.

VI. Reliability Improvement by Partial Dynamic Reconfiguration

The results of our research demonstrated in the previous parts of this paper is the increase of system reliability by using the TMR approach. TMR is naturally only a passive approach [21], which is capable of delivering correct output when a fault occurs. However, just fault masking means that FPGA is not fully utilized – namely the ability of changing its configuration. Therefore, the extension of the passive by active approach is very effective. The active approach of avoiding the impact of fault occurrence is based on a reconfiguration. The combination of passive and active approaches is described in [22]. The utilization of a reconfiguration controller to eliminate faults in FPGA based applications is a key technique for such approaches. This controller is responsible for scrubbing and other activities combined with bitstream relocation. A Partial Dynamic Reconfiguration (PDR) is very advantageous when used because it does not interrupt other functions implemented in FPGA. Therefore, PDR is a very important approach for critical applications and Generic Partial Dynamic Reconfiguration Controller (GPDRC) [23] is an example of such a controller used for this purpose. An extension of the TMR majority voter function is demanded to acquire information on which module a fault has occurred. This extension resides in adding an output from the voter, which identifies the malfunction module whose output value is different from the other two. Mitigation of faults that occur in individual TMR modules is possible due to the GPDRC existence in the design. Such an approach increases the reliability of a particular system.

A. PDR Controller Requirement

The requirement of including more components to FPGA necessarily represents sufficient FPGA size. However, larger and more expensive FPGA will be a requisite for achieving fault tolerance improvement. Possibly, a reconfiguration controller can be configured in FPGA with the application, or as another component which has to be included into the system (i.e. an extra FPGA). However, the increase in occupied FPGA area is manifested in both cases. The probability of hitting the utilized part of FPGA by some fault, which can cause a circuit malfunction, increases concurrently with its reliability. Faults, which hit the TMR module, are not important because the reconfiguration controller will fix them. Nevertheless, some faults can hit the reconfiguration controller that will behave in an incorrect way. It can cause ultimate system breakdown even if this system behavior is presumed. The malfunctioning controller might occasionally destroy a correct circuit configuration due to a reconfiguration which is performed unexpectedly. If the controller area in FPGA is considerably smaller than the application in TMR, the controller should provide longer failure-free time compared with triplication.

The reconfiguration controller is utilized to deliver bitstreams into the FPGA configuration memory during the PDR procedure. These bitstreams are called *golden bitstreams*. Special memory can be used to save them. When a fault is identified, the controller then loads and applies corresponding bitstreams from this memory which should be protected against faults as well. The records can be equipped with a correction code. Another component which can possibly be included in the FPGA produces higher FPGA area requirements. Another possibility is the utilization of module triplication (TMR). When there is a fault in one module, the remaining two modules are still faultless. Therefore, three bitstreams of three modules are downloaded from the FPGA configuration memory. Then, a new bitstream is prepared as a majority value of each bit of these bistreams. This new bitstream is used for fixing faults instead of using golden bitstream. This approach is called *Lazy Scrubbing* [24] by M. Garvie.

B. Fault Tolerant PDR Controller

Fault tolerance of a reconfiguration controller might be also required in some applications. The same technique as for protected circuit can possibly be used (i.e. TMR). The reconfiguration controller will then be implemented three times in FPGA and each of its instances will operate in parallel with the remaining two. The outputs from each instance will be compared and when a controller malfunction is detected, the other two correctly functioning controllers will be capable of reconfiguring the incorrectly working one. Even in this case, when two controllers reconfigure the third one simultaneously, it is possible to check fault occurrence. Because outputs of the two correctly working controllers are compared and if they are equal, no fault exists there. This approach corresponds to the Duplication with Comparison (DwC) method. Nevertheless, in the case of fault occurrence in DwC architecture it is impossible to determine which controller works incorrectly. It is important to note that the fault tolerance for reconfiguration controller has a negative impact on FPGA area and circuit delay – because of a majority voter addition to the system.

C. Research in the Area of GPDRC

Two versions of the PDR controller for FPGA, which we designate as GPDRC, were designed and implemented within our research group. The first version of the controller [23] is capable to mitigate transient faults i.e. SEUs. The second expanded version of the controller [25] is focused on the mitigation of permanent faults too. The permanent fault is a fault when some bits of FPGA configuration memory are in an incorrect state without a possibility to change it. Some spare modules for TMR are available in the FPGA to be used in these situations. These modules are able to be used instead of another TMR module with a permanent fault. When all spare modules are exhausted and one more permanent fault occurs, fault protection degrades to only DwC. The next permanent fault causes circuit function termination.

Previous research aims at ensuring fault tolerance for the circuit function itself. However, fault tolerance for the reconfiguration controller itself is not provided. The controller has to be in a radiation protection component outside FPGA, which performs its function. It is certainly not the only alternative of protecting the reconfiguration controller against faults.

Future research in the area of GPDRC will be oriented to bringing fault tolerance into the GPDRC design. The utilization of the TMR approach for the design of a fault tolerant GPDRC is an idea for future research. This approach is based on placing three equivalent controllers into the FPGA and connecting their outputs to a special majority voter. The experiments

with this approach will be executed and compared with the version which contains just one reconfiguration controller. In both cases, the reconfiguration controller will guarantee the proper function of a robot controller, which is described herein. Certainly, this is only a passive approach for fault tolerance, which is not sufficient for proper attenuation of the SEU impact on the FPGA. Therefore, we will solve how to perform reconfiguration of the malfunction controller by another two controllers which operate correctly. Obviously, the malfunction reconfiguration controller must not interfere in its reconfiguration. However, that problem will be solved by a majority voter, which masks one incorrect output due to another two correct outputs. We will ensure fault tolerance for majority voters after thorough testing and comparison with previous versions. The majority voters remain as the last unprotected component of the system.

VII. Conclusions and Future Research

In this paper we introduced the combination of fault injection-based experimental evaluation and theoretical reliability analysis of our previously developed robot controller. Our previous work was targeted mainly towards the experimental evaluation, but we feel that the theoretical reliability analysis has also important place in our evaluation process. We applied a commonly used TMR on the top level of the robot controller (there were three instances of the robot controller complemented with the majority voter). The first step was the fault injection-based experimental evaluation of the robot controller without TMR applied and its reliability indicators calculation. The calculation of reliability indicators of the TMR version was done in the second step, then we gained estimated reliability indicators. Reliability indicators of the TMR version were also evaluated by fault injection. These experiments show us that TMR has a significant impact on the reliability indicator and improves reliability of the hardened robot controller. When we compare a measured and estimated reliability indicator we found that the measured values are not so good as the estimated ones. We outlined possible causes which are in the scope of our future research.

Acknowledgements

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II); project IT4Innovations excellence in science - LQ1602, ARTEMIS JU under grant agreement no 621439 (ALMARVI) and BUT project FIT-S-14-2297.

References

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

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

[3] F. L. Kastensmidt, G. Neuberger, L. Carro, and R. Reis, "Designing and testing fault-tolerant techniques for sram-based fpgas," in *Proceedings of the 1st conference on Computing frontiers*. ACM, 2004, pp. 419–432.

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

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

[6] D. White, "Considerations surrounding single event effects in fpgas, asics, and processors," http://www.xilinx.com/support/documentation/white_papers/wp402_SEE_Considerations.pdf, Mar. 2012, accessed: 2016-09-15.

[7] 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," vol. 50, no. 6, 2003, pp. 2088–2094.

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

[9] C. Bernardeschi, L. Cassano, A. Domenici, and L. Sterpone, "Accurate Simulation of SEUs in the Configuration Memory of SRAM-based FPGAs," in *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 115–120.

[10] M. Alderighi, S. D'Angelo, M. Mancini, and G. R. Sechi, "A Fault Injection Tool for SRAM-based FPGAs," in *On-Line Testing Symposium, 2003. IOLTS 2003. 9th IEEE*. IEEE, 2003, pp. 129–133.

[11] M. Alderighi, F. Casini, S. d'Angelo, M. Mancini, S. Pastore, and G. R. Sechi, "Evaluation of Single Event Upset Mitigation Schemes for SRAM-based FPGAs Using the FLIPPER Fault Injection Platform," in *Defect and Fault-Tolerance in VLSI Systems, 2007. DFT'07. 22nd IEEE International Symposium on*. IEEE, 2007, pp. 105–113.

[12] J. Podivinsky, O. Cekan, J. Lojda, and Z. Kotasek, "Verification of Robot Controller for Evaluating Impacts of Faults in Electro-mechanical Systems," in *Digital System Design (DSD), 2016 19th Euromicro Conference on*. IEEE, 2016, pp. 487–494.

[13] J. Hlavička, *Číslicové systémy odolné proti poruchám*. ČVUT, 1992.

[14] K. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. Wiley, 2016. [Online]. Available: https://books.google.cz/books?id=_yOXDAAAQBAJ

[15] S. Sangwine, *Electronic Components and Technology, Third Edition*, ser. Tutorial guides in electronic engineering. CRC Press, 2007. [Online]. Available: https://books.google.cz/books?id=vVHvBQAAQBAJ

[16] B. Dhillon, *Applied Reliability and Quality: Fundamentals, Methods and Procedures*, ser. Springer Series in Reliability Engineering. Springer London, 2007. [Online]. Available: https://books.google.cz/books?id=rRp7DkTegMEC

[17] 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.

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

[19] 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.

[20] O. Cekan, J. Podivinsky, and Z. Kotasek, "Random stimuli generation based on a stochastic context-free grammar," in *Proceedings of the 2016 International Conference on Field Programmable Technology*. IEEE Computer Society, 2016, pp. 291–292.

[21] J. Jiang and X. Yu, "Fault-tolerant control systems: A comparative study between active and passive approaches," *Annual Reviews in Control*, vol. 36, no. 1, pp. 60–72, 2012.

[22] X. Yu and J. Jiang, "Hybrid fault-tolerant flight control system design against partial actuator failures," *IEEE Transactions on Control Systems Technology*, vol. 20, no. 4, pp. 871–886, July 2012.

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

[24] M. Garvie, "Reliable electronics through artificial evolution," Ph.D. dissertation, University of Sussex, January 2005.

[25] L. Miculka and Z. Kotasek, "Generic partial dynamic reconfiguration controller for transient and permanent fault mitigation in fault tolerant systems implemented into fpga," in *17th International Symposium on Design and Diagnostics of Electronic Circuits Systems*, April 2014, pp. 171–174.

Paper G

Extended Reliability Analysis of Fault-Tolerant FPGA-based Robot Controller

Jakub Podivinsky, Jakub Lojda and Zdenek Kotasek

In: Proceedings of the 2019 20th Latin American Test Symposium. Santiago: IEEE Computer Society, 2019, pp. 1-4. ISBN 978-1-72811-755-3.

Extended Reliability Analysis of Fault-Tolerant FPGA-based Robot Controller

Jakub Podivinsky, Jakub Lojda, Zdenek Kotasek
Faculty of Information Technology, Brno University of Technology, Centre of Excellence IT4Innovations
Bozetechova 2, 612 66 Brno, Czech Republic
Tel.: +420 54114-{1361, 1360, 1223}
Email: {ipodivinsky, ilojda, kotasek}@fit.vutbr.cz

***Abstract*—The reliability of safety-critical systems is very important especially in case of electronic systems which are working in environment with increased occurrence of faults. As an example, space, aerospace or medical systems can serve. Fault tolerance is one of the techniques the goal of which is to avoid the impact of faults on such systems. Lots of fault tolerance techniques exists and new ones are under investigation. This paper is targeted mainly to *Field Programmable Gate Arrays* (FPGAs) which are also the target technology of many fault tolerant techniques. It is important to evaluate and test these techniques. This paper is the continuation of our previously published research results which presents experimental approach to evaluate such fault tolerance techniques by monitoring the impact of faults in the experimental electro-mechanical system utilizing robot navigation in a maze. However, in this paper, we research and compare similarities of the theoretical estimation to various methods of the SEU injection approaches. The theoretical estimation is calculated using known equations. The impact of artificially faults injected into the electronic controller, in which Triple Modular Redundancy is applied, is monitored and used for statistic reliability analysis. This approach serves as a tool for the fast reliability evaluation during the development process of fault tolerance systems.**

***Keywords*—Reliability Analysis, TMR, FPGA, Fault Tolerance, Robot Controller.**

I. Introduction

The reliability of safety-critical electronic systems which are working in environment with increased occurrence of faults is a very challenging topic. A technique called *fault tolerance* [1] is commonly used technique which makes electronic systems more reliable. The goal of this approach is to keep the system functional, even in the presence of faults. It means that fault tolerance accepts the fact a fault can appear in electronic system. Various types of redundancy are the core of such techniques. Hardware and time redundancy are the most common ones. Combination and improvements of these basic methods are still under investigation, e.g. authors of [2] present approach which is based on the combination of hardware and time redundancies.

Many fault-tolerant methodologies targeted to *Field Programmable Gate Arrays* (FPGAs) have been developed and new ones are under investigation [3]. The main reason is that FPGAs are more popular thanks to their flexibility and ability to be reconfigured in case of fault occurrence. Sensitivity of FPGAs to faults caused by charged particles [4] is the problem from the reliability point of view. The configuration of FPGA is stored as a *bitstream* in SRAM memory and charged particle can cause inversion of bit in the *bitstream*. This event is called *Single Event Upset* (SEU) [5].

978-1-7281-1756-0/19/$31.00

A fault-tolerant system development usually starts with a *nondurable* system that does not tolerate faults [6]. This *nondurable* system is usually designed with minimum redundancy and serves as a starting point for the process of hardening against faults. Then the modifications that should be made to the *nondurable* system in order to achieve a higher level of fault tolerance are proposed by an experienced fault-tolerant system designer. After integration of proposed changes into the design, the system must be evaluated to ensure that the applied changes have the expected impact on the reliability of the system. The iteration between the phase of development and the reliability evaluation is the usual approach. Multiple designs with various combinations of fault tolerance methods assigned to the partitions of the design are created. Development ends if two conditions are met: 1) the system complying with the specification or 2) the findings of the specifications not being achievable. We to accelerate this procedure of the development with the capability to evaluate the estimation of reliability of the resulting system even *before* the integration of the method itself. This allows a designer to exclude such combinations of reliability methods that do not look perspective.

This work is the continuation of our previously published paper. Additional experiments were done and experimental results were compared with results obtained in our previous publication [7]. This paper is organized as follows. Section II described reliability analysis and reliability improvement. The experimental platform which allow us to done experimental evaluation on real FPGA is introduced in Section III. Section IV presents reliability analysis and its experimental evaluation. Section V concludes the paper and mentions plans for our future research.

II. Reliability Analysis and its Improvement

The reliability itself can be quantified with the support of the *theory of probability* as most of the *reliability indicators* are of a random nature. The length of a time period of the system operation until the failure occurs is an important starting point in the *reliability indicators* computation.

1) Failure Function: If a *random variable* τ expresses a length of a time interval from the systems start of the operation to the point a fault occurs, then the *Cumulative Distribution Function* (CDF) [8] $F(t)$ of *random variable* τ expresses a probability of the system being in a failure state at the time t. In this case, the CDF $F(t)$ is denoted as $Q(t)$ and is called the *failure function*.

2) Reliability Function: Another *reliability indicator* is the so-called *reliability function* which is denoted as $R(t)$. The

reliability function expresses a probability of the system being in an fault-less state at the time t and it is a supplement of the $Q(t)$ as expressed in Equation 1.

$$R(t) = 1 - Q(t) \tag{1}$$

3) Failure Density: The *failure density* $f(t)$ is defined by the time derivative of a CDF $Q(t)$ if the *random variable* [8] is continuous and the derivative exists, as shown in Equation 2.

$$f(t) = \frac{\mathrm{d}Q(t)}{\mathrm{d}t} \tag{2}$$

The product of $f(t)\mathrm{d}t$ then expresses the probability of a fault occurrence for a short period of time $\mathrm{d}t$ that is immediately following after the time t. Although, the case in which the fault occurred earlier before the time t is not taken into account.

4) Failure Rate: The next *reliability indicator* is *failure rate* which is denoted by $\lambda(t)$. The *failure rate* expresses a conditional *failure density* at the time t assuming the failure has not occurred yet. Equation 3 gives a relationship between the $\lambda(t)$ and $Q(t)$.

$$\lambda(t) = \frac{f(t)}{R(t)} = \frac{f(t)}{1 - Q(t)} \tag{3}$$

5) Mean Time To Failure: The *Mean Time To Failure* (MTTF) which is in the following text denoted as T_s represents a mean value of the random variable τ observed. The mean value can be seen as a mean time of all the time period lengths since the system started its operation to the first failure occurrence. If the mentioned system is *non-recoverable*, the value can be considered a *mean time to the first failure* as well. To calculate the T_s, the Equation 4 can be used.

$$T_s = \int_0^\infty R(t)\mathrm{d}t \tag{4}$$

The reliability improvement [9] can be done by several techniques, lots of them are based on *redundancy*. *Triple Modular Redundancy* (TMR, 3MR) which is based on a *triplication* of the component is the most known application of the *hardware redundancy*. In this paper, we plan to analyze the reliability improvement just for TMR. The TMR is based on using of three equivalent functional units, Figure 1 shows the structural schematic. The TMR system is composed of original functional unit and two additional copies of the same functional unit, which are labeled F_1, F_2 and F_3. The input signals x are connected as an input for each of the functional units F_i. The output signals $f_i(x)$ are connected to the *voter* unit which implements *majority function*. The *majority function* can be performed on the level of bits, whole vector, etc. It should be noted, that *voter* used in this paper operates on the per-bit basis.

If we are not taking into account the corner cases, TMR by its nature allow us to mask the failure of one module. Assuming that all F_i units have the same reliability function $R(t)$, then Equation 5 can be used to calculate the reliability function of the whole TMR module. An overview of the reliability indicators of the system with TMR implemented is given in Table I [6], [9].

$$R_{TMR}(t) = 3[R(t)]^2 - 2[R(t)]^3 \tag{5}$$

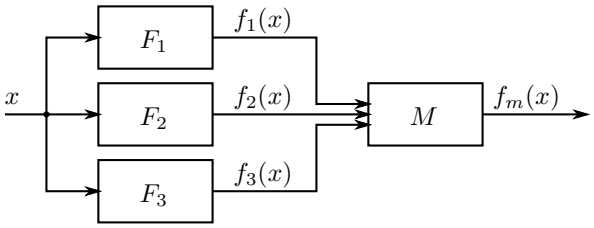


Fig. 1. The schematic representation of the TMR method.

TABLE I. TWO MAIN RELIABILITY INDICATORS OF THE TMR SYSTEMS.

Reliability indicator	Input variables		Value
Reliability Function $R_{TMR}(t)$	R(t)	R(t) of the original functional unit	$R_{TMR}(t) = 3[R(t)]^2 - 2[R(t)]^3$
Mean Time To Failure $T_{s(TMR)}$	λ	λ of the original functional unit	$T_{s(TMR)} = \frac{5}{6\lambda}$

III. EVALUATION PLATFORM AND EXPERIMENTAL SYSTEM

The development of the evaluation platform for monitoring impact of faults injected into FPGA-based system was the scope, among other, of our previous work [10]. Developed evaluation platform is designed for monitoring impact of faults on electro-mechanical system. The main reason is that lots of digital systems very often control some mechanical part. The use of the electro-mechanical system allows us to monitor not only the impact of faults on the electronic controller but also on the mechanical part. Our evaluation platform uses *Functional Verification* [11] as a tool for checking reactions of experimental system on injected faults. Functional verification is usually used for checking if electronic system corresponds with its specification by monitoring inputs and outputs in design simulation. We propose extended version of the functional verification where verified circuit is running on FPGA as appropriate tool for our purposes. The evaluation platform which is described in our previous work (e.g. [10]) is composed of:

1) software part of verification environment for the electronic controller which checks the reaction of electronic controller and mechanical part on injected faults running on computer,
2) software simulation environment for mechanical part simulation running on computer,
3) electronic controller implemented into FPGA, and
4) external fault injector [12] running on a computer which allows us to simulate real faults in FPGA.

Our experimental electro-mechanical system consists of a robot for searching a path through a maze and its electronic controller implemented in FPGA. Unfortunately, we do not have a real robot device, so we use the simulation tool Player/Stage [13] which allows us to simulate the robot and its environment (in our case the robot in a maze). The robot simulation is executed on a computer which is connected with the FPGA board by the Ethernet interface through which data between the robot and its controller are transmitted. Two versions of the robot controller are used for our experiments and results comparison. The first version is hard-coded robot controller which is composed of various functional units interconnected through the central bus. The second version is processor-based robot controller which consist of soft-core

processor NEO430 [14] and some external components implemented in FPGA. The searching algorithm is implemented in C/C++ and performed on the processor.

IV. Reliability Analysis and Experimental Evaluation

The goal of this work is to evaluate some of the basic *reliability indicators* of the two versions of robot controller which was mention in Section III and their fault-tolerant versions with TMR applied (noted as *noft* and *tmr*).

The fault injection was set up with a constant SEU injection rate. The SEUs were injected to the utilized bits of the bitstream that represent the content of *Look-Up Tables* (LUTs). The important parameter of fault injection is a time delay d_c between two injected faults. The d_c actually does not necessarily have to be constant. We have experimentally chosen the d_c to be described by the *uniform distribution* with a *mean* value of 12 s and a *variance* of 2 s.

The scenario of one verification run was as follows:

1) the robot controller unit was initialized, the maze and starting and target positions were the same during all the verification runs,
2) the *Player/Stage* simulation environment was started with the robot placed on the starting position,
3) after 15 s, for each component c, the SEU injection started with the d_c time period, the bits into which faults were injected were selected *uniformly at random*,
4) the time from the robot start to the first failure was monitored, the ability of the robot to reach the target position was observed as well.

This verification scenario was repeated 3500 times for all versions of the robot controller units. In detail, experimental strategies follow:

- fault injection into unhardened robot controller, component c is whole robot controller (*noft*),
- fault injection into TMR version of robot controller which respect increased area, faults were injected into three component c_1, c_2, c_3 (instances of robot controller) concurrently which led to three times fault intensity for whole robot controller (*tmr*),
- fault injection into TMR version of robot controller, faults were injected into one component c, which represents whole hardened robot controller (fault intensity is the same as in *noft* case) (*tmr1*).

The data acquired included the time of the first failure occurrence and information on whether the robot successfully reached the target position.

The data obtained from the previously described experiments were then processed. The *multi-set* of all the times measured from the start of the operation of the system to the first detection of an error on the system outputs was transformed to a discrete *failure function* $Q(t)$ which was then converted to the *reliability function* $R(t)$. The other *reliability indicators failure density* $f(t)$ and *failure rate* $\lambda(t)$ was computed according to proposed equations. All the data are discretized with the time-step of 15 s in Table II. The final values of the *reliability functions* on the bottom of Table II are notclose to the limit value of zero, as in most cases the faults injected did not appear in the form of an error on the outputs of the robot controller unit. The threshold time length the robot had to find its path within was evaluated to 204 s, that is also the maximum time for which the system has been verified.

TABLE II. A discretization of the measured *failure function* Q(t) of the *noft* and *tmr* version of the *hard-coded* and *processor-based* robot controller with the estimation for the *tmr* robot controller.

		Hard-coded robot cont.			Processor-based robot cont.		
Time t $[s]$		First err. detect. $[-]$	$[\%]$	$Q(t)$ $[-]$	First err. detect. $[-]$	$[\%]$	$Q(t)$ $[-]$
$0-14.\bar{9}$	*noft*	0	0.0%	0.00	0	0.0%	0.00
	tmr	0	0.0%	0.00	0	0.0%	0.00
	tmr1	0	0.0%	0.00	0	0.0%	0.00
	est.	–	0.0%	0.00	–	0.0%	0.00
$15-29.\bar{9}$	*noft*	6	0.2%	0.00	16	0.5%	0.00
	tmr	1	0.0%	0.00	8	0.2%	0.00
	tmr1	0	0.0%	0.00	0	0.0%	0.00
	est.	–	0.0%	0.00	–	0.0%	0.00
$30-44.\bar{9}$	*noft*	9	0.3%	0.00	12	0.3%	0.01
	tmr	0	0.0%	0.00	10	0.3%	0.01
	tmr1	0	0.0%	0.00	2	0.1%	0.00
	est.	–	0.0%	0.00	–	0.0%	0.00
$45-59.\bar{9}$	*noft*	35	1.0%	0.01	45	1.3%	0.02
	tmr	8	0.2%	0.00	35	1.0%	0.02
	tmr1	3	0.1%	0.00	5	0.1%	0.00
	est.	–	0.0%	0.00	–	0.1%	0.00
$60-74.\bar{9}$	*noft*	28	0.8%	0.02	35	1.0%	0.03
	tmr	25	0.7%	0.01	61	1.7%	0.03
	tmr1	4	0.1%	0.00	3	0.1%	0.00
	est.	–	0.0%	0.00	–	0.2%	0.00
$75-89.\bar{9}$	*noft*	8	0.2%	0.02	11	0.3%	0.03
	tmr	11	0.3%	0.01	69	2.0%	0.05
	tmr1	1	0.0%	0.00	9	0.3%	0.01
	est.	–	0.0%	0.00	–	0.1%	0.00
$90-104.\bar{9}$	*noft*	47	1.3%	0.04	64	1.8%	0.05
	tmr	53	1.5%	0.03	169	4.8%	0.10
	tmr1	18	0.5%	0.01	26	0.7%	0.01
	est.	–	0.2%	0.00	–	0.5%	0.01
$105-119.\bar{9}$	*noft*	42	1.2%	0.05	30	0.9%	0.06
	tmr	36	1.0%	0.04	76	2.2%	0.12
	tmr1	8	0.2%	0.01	18	0.5%	0.02
	est.	–	0.2%	0.00	–	0.3%	0.01
$120-134.\bar{9}$	*noft*	52	1.5%	0.06	51	1.5%	0.08
	tmr	34	1.0%	0.05	92	2.6%	0.15
	tmr1	4	0.1%	0.01	21	0.6%	0.02
	est.	–	0.2%	0.01	–	0.6%	0.02
$135-149.\bar{9}$	*noft*	36	1.0%	0.08	30	0.9%	0.08
	tmr	47	1.3%	0.06	66	1.9%	0.17
	tmr1	4	0.1%	0.01	19	0.5%	0.03
	est.	–	0.4%	0.01	–	0.4%	0.02
$150-164.\bar{9}$	*noft*	42	1.2%	0.09	24	0.7%	0.09
	tmr	33	0.9%	0.07	85	2.4%	0.19
	tmr1	6	0.2%	0.01	16	0.5%	0.03
	est.	–	0.3%	0.01	–	0.3%	0.02
$165-179.\bar{9}$	*noft*	50	1.4%	0.10	17	0.5%	0.10
	tmr	48	1.4%	0.08	64	1.8%	0.21
	tmr1	8	0.2%	0.02	26	0.7%	0.04
	est.	–	0.6%	0.02	–	0.2%	0.03
$180-194.\bar{9}$	*noft*	45	1.3%	0.11	50	1.4%	0.11
	tmr	46	1.3%	0.10	129	3.7%	0.25
	tmr1	15	0.4%	0.02	58	1.7%	0.06
	est.	–	0.7%	0.03	–	0.8%	0.03
$195-209.\bar{9}$	*noft*	856	24.5%	0.36	1520	43.4%	0.54
	tmr	787	22.5%	0.32	1373	39.2%	0.64
	tmr1	129	3.7%	0.06	296	8.5%	0.14
	est.	–	21.8%	0.25	–	53.3%	0.57

Measured data were also transformed to *reliability functions* $R(t)$ for both versions of robot controller. The $R(t)$ functions for hard-coded robot controller are shown in Figure 2. Red lines show, that *tmr* version (injection into 3 component, respect increased area) is better than *noft* version, but significantly worse than estimation. These results were also presented in [7] and additional experiments with version *tmr1* (injection into whole controller, increased area is not taken into

account) were performed. Green line shows, that *tmr1* version is almost the same as estimation.

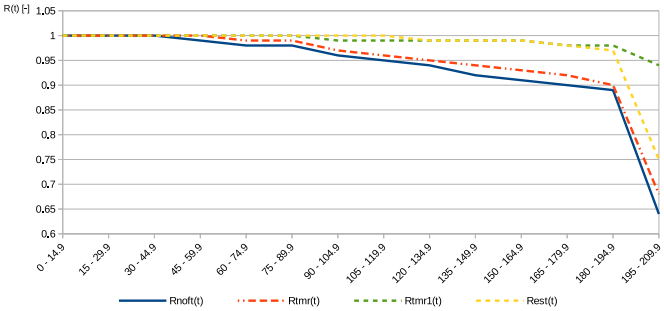


Fig. 2. An experimental evaluation of the measured results of the *reliability function* for the *noft* and the *tmr* versions of the *hard-coded* robot controller.

Additional experiments with processor-based robot controller were performed to confirmation of previous results. Figure 3 show the same chart for processor-based robot controller. The big difference is that *tmr* version (injection into 3 component, respect increased area) is worse than *noft* version. The processor is a complex system and a fault injection with higher intensity led to its worse reliability. On the other hand, *tmr1* version (injection into whole controller, increased area is not taken into account) represented by green line is almost the same as estimated reliability. These experiments confirm, that equation 5 does not take into account increased area of TMR system.

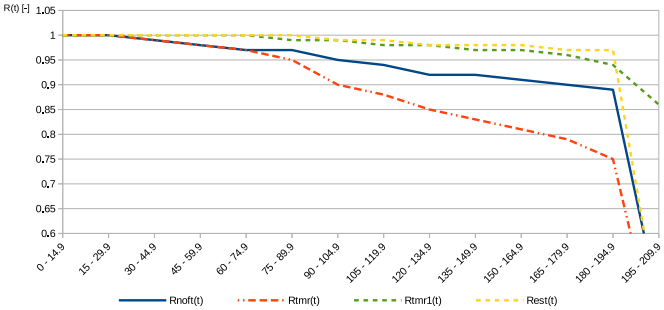


Fig. 3. The measured results of the *reliability function* for the *noft* and the *tmr* versions of the *processor-based* robot controller.

V. Conclusions and Future Research

In this paper we present the combination of experimental and theoretical evaluation of robot controller reliability. We applied a commonly used TMR on the robot controller (there were three instances of the robot controller complemented with the majority voter). The first step was fault injection into unhardened version of robot controller, reliability indicators calculation and then estimated reliability of TMR version were calculated according to commonly used equation 5. Next step was experimental evaluation of estimated reliability indicators. The first experiments were done with fault injection into all robot controller instances concurrently with respect to increased area. The experimentally measured results indicated significantly worse reliability than the estimation predicted. The second experiment was done with fault injection just into the whole robot controller and the obtained results correspond with the estimated reliability. These experiments confirm, that equation 5 does not take into account increased area of TMR system.

Presented results were obtained using TMR without faulty module recovery. The faulty module recovery significantly increases the operation time without failure. The scope of our future research is to apply reconfiguration as a tool for faulty module recovery and perform similar experiments and examine benefits and negatives.

Acknowledgements

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science – LQ1602, the BUT project FIT-S-17-3994 and the JU ECSEL Project SECREDAS (Product Security for Cross Domain Reliable Dependable Automated Systems), Grant agreement No. 783119.

References

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

[2] F. L. Kastensmidt, G. Neuberger, L. Carro, and R. Reis, "Designing and testing fault-tolerant techniques for sram-based fpgas," in *Proceedings of the 1st conference on Computing frontiers*. ACM, 2004, pp. 419–432.

[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] D. White, "Considerations surrounding single event effects in fpgas, asics, and processors," http://www.xilinx.com/support/documentation/white_papers/wp402_SEE_Considerations.pdf, Mar. 2012, accessed: 2016-09-15.

[5] 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," vol. 50, no. 6, 2003, pp. 2088–2094.

[6] J. Hlavička, *Číslicové systémy odolné proti poruchám*. ČVUT, 1992.

[7] J. Podivinsky, J. Lojda, O. Cekan, R. Panek, and Z. Kotasek, "Reliability Analysis and Improvement of FPGA-Based Robot Controller," in *Digital System Design (DSD), 2017 Euromicro Conference on*. IEEE, 2017, pp. 337–344.

[8] K. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. Wiley, 2016. [Online]. Available: https://books.google.cz/books?id=_yOXDAAAQBAJ

[9] B. Dhillon, *Applied Reliability and Quality: Fundamentals, Methods and Procedures*, ser. Springer Series in Reliability Engineering. Springer London, 2007. [Online]. Available: https://books.google.cz/books?id=rRp7DkTegMEC

[10] 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.

[11] A. Meyer, *Principles of Functional Verification*. Elsevier Science, 2003. [Online]. Available: http://books.google.cz/books?id=qaliX3hYWL4C

[12] 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.

[13] 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.

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

Paper H

Evaluation Platform For Testing Fault Tolerance: Testing Reliability of Smart Electronic Locks

Jakub Podivinsky, Jakub Lojda, Richard Panek, Ondrej Cekan, Martin Krcma and Zdenek Kotasek

In: Proceedings of the 2020 11th IEEE Latin American Symposium on Circuits and Systems. San José: IEEE Computer Society, 2020, pp. 97-100. ISBN 978-1-72811-755-3.

Evaluation Platform For Testing Fault Tolerance: Testing Reliability of Smart Electronic Locks

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

***Abstract*—This research paper presents the examination of the influences of faults on a control unit of smart electronic locks. A stepper motor is often used as an actuator of such smart locks and its motor controller is usually implemented in a processor. The aim of this paper is to examine the impact of faults occurring in the control processor. It should be noted that faults in such electronic systems can also be induced artificially, usually with ulterior motives. The processor can be implemented in an FPGA (Field Programmable Gate Array) in order to be able to emulate HW faults inside the processor. This allows us to use previously developed evaluation platform for fault tolerance testing. This platform allows us to monitor the impact of faults both on electronic and mechanical parts of electro-mechanical system. In this paper, the evaluation of faults artificially injected in FPGA-based processor is proposed. Experiments with both single and multiple fault injections were performed. In our research, we found out that a fault in the same position of the design does not always affect the electronics in the same way. Also, the mechanics may still operate correctly despite the electronics failure.**

Keywords*—*Electronic Lock, Stepper Motor, FPGA, Fault Tolerance, Fault Injection.

I. Introduction

Nowadays, *smart devices* [1] are on the rise, their goal is to make our lives more efficient, simpler and more pleasant. The smart electronic lock [2] is an example of smart device that can be met in our everyday life. Connecting electronics with mechanical elements and a remote server brings new possibilities for users to control these locks. The smart electronic locks have many advantages, since they can be controlled remotely.

Various types of faults can arise in electronic systems, especially if such systems are operated in environments with increased level of electrostatic electricity, increased occurrence of charged particles, etc. It should be noted that faults can also be injected artificially, usually with malicious intentions. For example, sensitive data from an embedded memory (parts of an algorithm, encryption keys, etc.) can be extracted by bumping attacks [3]. Attacks on smart cards based on fault injection which modifies the behavior [4] can serve as another example. Unauthorized or accidental unlocking of electronic locks may also be caused by fault injection. Opening a lock or preventing its closing can cause financial losses or endanger human health or life. Unauthorized unlocking can be caused by various types of attacks, either to the server part of the whole system, or directly to the lock in the door [5]. Intently inducing faults is one of the possible attacks which is yet an unexplored topic that we will deal with in our research.

The evaluation of the effects of faults on an electro-mechanical system is in focus of our research. Especially, we focus on systems composed of SRAM-based FPGAs. Another example of electro-mechanical system where faults can cause undesirable consequences, is an electronic lock. In this work, we are going to use previously developed tools to analyze the impact of faults on electronic locks. Electronic locks are usually controlled by an embedded processor that can be implemented in an FPGA which gives us the possibility to simulate faults. We have identified three main goals that we are going to achieve during our research targeted on electronic locks which are in detail presented in paper [6]:

1) *The evalution of faults injected directly into stepper motor control signals and the estimation of the risk of an unauthorized unlocking* (solved in [6]).
2) *To implement the stepper motor controller with a processor configured into an FPGA and evaluate: a) the impact of faults injected into processor and b) the possibility of unauthorized unlocking (topic of this paper).*
3) *To verify the possibility of using standard fault tolerance techniques for eliminating unauthorized lock unlocking.*

During solving of the goals we will use three levels of evaluation which were presented in [6] with various architectures of component realization and interconnection. This paper is based on interconnections of a PC for simulation running and an FPGA where electronic controller is implemented.

This paper is organized as follows. Electronic lock components are presented in Section II. Section III introduces an evaluation platform for monitoring the impact of faults on electro-mechanical applications. Section IV describes experimental environment architecture. Experiments with fault injection are presented in Section V. Section VI concludes the paper and mentions plans for our future research.

II. Electronic lock

Smart electronic locks are quite complex electronic devices. The basis of the lock can be divided into three blocks – control module, I/O module and motor module [7]. The important block is the motor module which performs the operation with the mechanical part of the lock. Stepper motor is very often found in electronic locks [8] [9], that is why we focus on experiments with stepper motor in our research.

The stepper motor is a DC electric motor whose full rotation can be divided into several equal steps. The stepper motor is controlled by input pulses (typically square pulses) that precisely rotate the shaft position based on an angle which is given by the number of motor steps. It consists of a cylindrical rotor, a number of stators, a number of yokes, and a set of coils [10]. We have chosen a conventional bipolar stepper motor with a permanent magnet in its rotor which operates on the attraction or repulsion between the rotor and the stator electromagnets. The particular model of this type of stepper

978-1-7281-3427-7/20/31.00/31.00

motor that we have chosen is 28BYJ-48 [11]. It is a small stepper motor operating at 5V which is equipped with a 1/64 transmission gearbox. It has 4 phases with a single step angle of 5.625°/64 and 4,096 steps are needed to perform the full rotation (64 steps without the gearbox).

III. The Verification-based Evaluation Platform

The main evaluation tool used in this paper is our previously developed platform [12] for the evaluation of the impact of faults injected in electronic part of electro-mechanical system. The proposed evaluation platform is based on well known functional verification technique [12]. The core of the platform is an ML506 evaluation board with Virtex 5 FPGA which allows us to inject faults directly to the FPGA in which the electronic control unit is implemented. The fault injector based on Partial Dynamic Reconfiguration (PDR) is implemented in a computer and faults are injected through a JTAG interface. The communication between the simulation of mechanical part running on the computer and the electronic controller on the FPGA is accomplished through the Ethernet.

Functional verification is used as a tool for monitoring the impact of faults both on the electronic controller and the mechanical part of an electro-mechanical system. The modification of basic functional verification is shown in Figure 1. The main difference is that Device Under Test (DUT) is moved to the FPGA, which allows to inject artificial faults directly into FPGA and monitor their impact. The versatility of the proposed platform is based on the fact that functional verification is usually used during electronic systems development. Therefore, the verification environment and the reference model (the most important elements dependent on the evaluated system) are available from the previous stage of system development and can be used for a fault tolerance evaluation. The verification scenario generation is usually a part of the verification environment or we can use our previously developed universal generator [13]. An important condition for using the platform is that an electronic controller can be implemented in an FPGA. The DUT implementation in the FPGA and proper communication with the software part of verification environment are realized by the *driver* and the *monitor* components. These components are partly universal, but they need to be customized for a particular DUT.

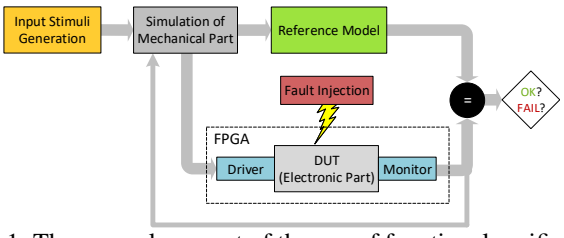


Fig. 1: The general concept of the use of functional verification for monitoring impact of faults.

The mechanical part is also an important element which allows to monitor the impact of faults not only on electronics, but also on mechanics. It is not important whether it is a real mechanical part or its simulation. The availability of sensors that provide feedback of the mechanical part behavior is important. The values provided by these sensors are monitored by the verification environment which checks if the system behaves according to its specification. Usually, the use of simulation leads to a faster testing and is usually cheaper.

Together with the evaluation platform we proposed the process of evaluation which is divided into three phases: 1) Classical simulation-based functional verification is done. After this phase we can be sure that the detected failures in following phases are caused by the injected faults. 2) During the second phase, modified verification environment (DUT implemented on FPGA) is used and artificial faults are injected. The output of this phase is a list of faults with the impact on electronic controller. 3) The third phase is focused on the evaluation of the behaviour of the mechanical part if the electronic part is corrupted by a fault.

IV. Experimental Setup

In this work, we deal with monitoring the impact of faults on the electronic lock, more precisely on the main mechanical element which is the stepper motor and its control processor. In this case, the verification environment is modified since no feedback is used when the stepper motor is controlled only by control signals. The measurement of the angle of rotation (both continuous and resulting angle) is needed for monitoring the behaviour of the mechanical part. The angle measurement can be easily realized both in the case of experiments with a real stepper motor (e.g. a stepper motor can rotate a potentiometer), and in the case of simulation. Figure 2 shows the use of functional verification-based platform for checking the impact of faults injected into the control processor implemented in FPGA.

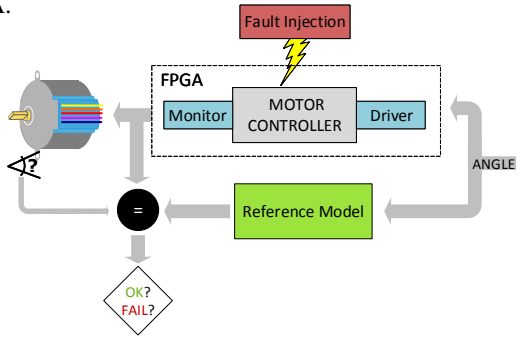


Fig. 2: The use of functional verification for monitoring impact of faults on stepper motor controller.

For our experimental purposes, we chose the NEO430 soft-core processor [14] which is based on the Texas Instruments MSP430 [15] instruction set architecture. The NEO430 is based on the Harvard architecture and uses program (IMEM) and data (DMEM) memory with configurable sizes. The processor implements configurable peripherals like a timer, a watchdog, UART and SPI serial interfaces (implemented together as a USART unit), general purpose IO ports and an internal bootloader. The peripheral modules are optional due to the possibility to reduce the size of the implemented system. In our experimental implementation (shown in Figure 3), we use UART interface for debugging purposes, Custom Functional Unit (CFU) is used as an input interface for the required number of steps and output signals for stepper motor are provided through the General-purpose inputs and outputs (GPIOs). The program is stored in ROM memory, it means that the program starts working after the processor is activated without any need to load the program from an external memory. The program itself repeatedly sets the stepper motor signals according to its specification. It is possible to parameterize the required number of steps through a signed number on the input ”STEPS”. The negative number causes

opposite direction of rotation. The outputs include four signals for the stepper motor control.

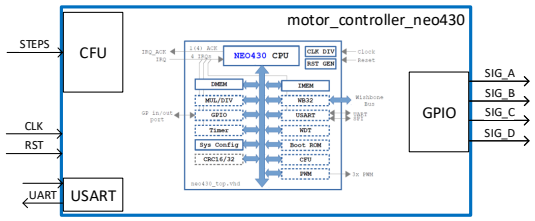


Fig. 3: The use of functional verification for monitoring impact of faults on the stepper motor controller.

We decided to use MATLAB and Simulink [16] for the simulation of the stepper motor, especially Simscape library [17] which proposes the stepper motor simulation. This model is generic, however, proper parameters from the stepper motor datasheet [11] must be used. More accurately, it is the 4-phase stepper motor with a permanent-magnet rotor. The output from the complete model is a current angle of the stepper motor. It is also necessary to model a voltage controller which converts logic inputs to the electric impulses which excite the motor coils.

V. Experiments and Experimental Results

The following subsections describe experiments corresponding to the second and the third phase of the evaluation process. There is no space for the first phase which is not so important from the reliability point of view. Faults were injected during the second and the third phase into the experimental processor according to two strategies – single bit-flip faults injected at the start of experiment and multiple bit-flip faults injected at a regular interval. We injected the faults into the bits of the bitstream that are utilized by LUTs of the controller implemented in the FPGA. The total number of the utilized bits was 58496. The impact of faults was monitored on the output of the electronic controller and then on the behaviour of the mechanical part with corrupted electronic controller.

During the mechanics simulation we recorded the rotation angle. The collected information covers the minimal, maximal and the final rotation angle of the stepper motor. The minimal angle was identical during all iterations as the motor always started at the same position and it never started to rotate in the opposite direction. The maximal and the final angle depended on an injected fault effect. Almost always the angles were equal. The maximal rotation angle is the most important variable in the context of a possible unwanted locking or unlocking the lock.

A. Single fault injection

Since an exhaustive evaluation would be demanding, we uniformly selected at random 6000 random bit faults that were injected into the FPGA. We performed five iterations of experiments with the selected bits in order to evaluate whether the controller behaviour is affected randomly by the particular injected fault or it remains the same. We compared the iterations in order to extract common features of the fault affected controller behaviour. The experiments were repeated five times using the same set of faults. Table I illustrates the failure rate of the electronic controller in all three experiment iterations (rows 1, 2, 3, 4 and 5). The *Electronic failure—Total* column covers the number of faults that caused the electronics failure. We divided the types of the electronics failures into three classes: 1) the premature stopping—*Stuck*, 2) unending controller operation—*Time-out* and 3) the correct termination, although with mismatching values—*Mismatch*. The numbers of failures of the particular types are listed in Table I. In the third phase we evaluated the mechanics behaviour. The *Mechanic OK—Total* column shows the number of cases when the electronics has failed but the mechanics functioned correctly and reached the proper position as was evaluated in the mechanics simulation. This number of faults therefore affected the electronic controller without affecting the overall lock behaviour at the same time. As can be seen, the highest number appears in the last column, that is when the controller terminated its activity correctly with mismatching values. The reason is we evaluated the electronics failure strictly when we considered even a single mismatching output as a failure.

TABLE I: The results of single and multiple injection experiments with the failures classification.

Iter.	Electronic failure				Mechanic OK			
	Total	Stuck	Time-out	Mis-match	Total	Stuck	Time-out	Mis-match
1	633	159	260	214	210	5	2	203
2	633	174	270	189	186	5	3	178
3	572	93	145	334	331	6	3	322
4	624	172	269	183	183	5	3	175
5	574	100	147	327	327	6	3	318
Multi	5772	1248	1901	2605	592	35	26	531

The set of bits that have proven to cause a failure was always a bit different between the iterations. We compared all combinations of two (e.g. $1 \cap 2$, $1 \cap 3$...), three (e.g. $1 \cap 2 \cap 3$, $1 \cap 3 \cap 4$...) and four (e.g. $1 \cap 2 \cap 3 \cap 4$, $1 \cap 3 \cap 4 \cap 5$...) sets of results. The result statistics are listed in Table II. The table does not cover all the combinations but describes only the minimum, average and the maximum of all of the combinations. The last row contains a combination of all iterations, so it presents a precise value. The *Electronic Failure* column covers the number of faults that caused an electronic failure in all the compared iterations. The *Mechanic OK* column covers the number of faults which have not affected the mechanics during any iteration. Even though, the same fault caused the electronics failure, the electronic did not fail always in the same way (stuck, time-out and mismatch). The column *Different El. Failure* covers the number of such faults. As can be seen, the growing size of compared sets leads to a smaller number of faults that cause electronic failure for each iteration. At the same time, the number of faults that did not cause any mechanic failure is lowering while on the other hand the number of faults with different electronic failure grows. We also detected unique bits, out of the total 6000, that caused a failure of the controller when fault was injected during almost one iteration which are shown in the last column. The number of these unique bits is rising with the growing size of the compared sets.

TABLE II: The iterations results comparison (single faults).

Num. of Comp. Runs	Electronic Failure			Mechanic OK			Different El. Failure			El. Failure Unique Bits		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
2	509	527	560	46	109	260	36	146	213	633	687	715
3	430	481	511	13	55	86	112	207	292	710	728	742
4	460	467	475	30	37	44	208	216	224	740	750	755
all	395			32			312			764		

Moreover, we examined the motor rotation angle. Figure 4a

contains a boxplot chart which displays the maximal angle for all the iterations. As the chart illustrates, the required rotation angle was 4500°. The majority of the electronics failures led to a smaller final rotation angle. Only a small number of faults caused a bigger rotation angle. If the goal of the fault injection is to unlock the lock unauthorized, most likely it will not be reached. On the contrary, if the goal is to prevent the door to lock properly, the chances are significantly higher.

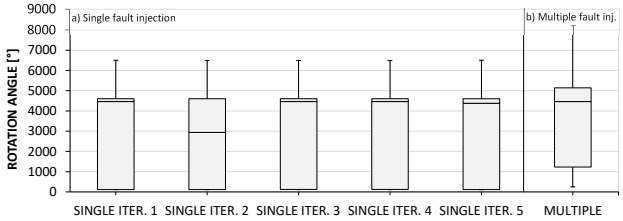


Fig. 4: Boxplot graph with rotation angle for (a) three experiment iterations with single injection and (b) multiple injection.

B. Multiple fault injection

Furthermore, we inspected the impact of a multiple fault injection on both the electronics and the mechanics. We performed 6000 experiments in this scenario as well. We uniformly-at-random injected a single fault into utilized bits of the motor controller design each five seconds. The particular experiment was terminated when a fault impact was detected or after 280 seconds if the faults proved to have no effect. The time limit is important as some faults do not cause a significant number of incorrect transactions on the controller output as other faults, but they cause that the controller would not stop the motor rotation when expected. The multiple injection proved to cause a significantly higher rate of the controller failures. Out of the 6000 total, 5772 experiments resulted in incorrect controller output transactions when multiple faults were injected.

The last row of Table I shows the number of faults that led to the electronics failure and their classification. It also contains the number that did not lead to the mechanics failure as well as the relations with the electronics failures types. As can be seen, the overall behaviour is similar to single fault cases, therefore the mechanics did not fail despite the inputs mismatch. The mechanics reactions on the incorrect controller outputs are illustrated in Figure 4b. As can be seen, unlike in the case of single faults, the smallest rotation angles did not occur and the span between the minimum and maximum grew.

VI. Conclusions and Future Research

In this paper we realized the second phase of our evaluation of fault injections effects on the electronic controller of an electronic lock implemented in HW. The stepper motor controlled by the controller implemented in a processor was used as our experimental platform. The processor was implemented in FPGA which allowed us a repeated and nondestructive testing of the faults effects. We examined the faults effects on electronics and mechanics of the lock, when we inspected how the faults affected the rotation of the motor. For these purposes, we further developed our fault tolerant system testing platform that we presented in our previous papers. The results indicate that random single faults cause a failure in about 10 % cases which is not suitable for an exploitation. The same faults proved not to always cause the same failures during the multiple iterations, moreover they did not lead to the mechanics failure every time. The mechanics failures most often led to a smaller angle rotation than desired, therefore these failures may prevent the door to lock. The failures proved to be more probable when injecting multiple faults.

In our future research, we will focus on the third phase which will utilize simulation accelerated on an FPGA in order to speed up the evaluation.

Acknowledgements

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II), the project IT4Innovations excellence in science – LQ1602, the BUT project FIT-S-17-3994 and the JU ECSEL Project SECREDAS (Product Security for Cross Domain Reliable Dependable Automated Systems), Grant agreement No. 783119.

References

[1] C. Salzmann, S. Govaerts, W. Halimi, and D. Gillet, "The smart device specification for remote labs," in *Proceedings of 2015 12th International Conference on Remote Engineering and Virtual Instrumentation (REV)*. IEEE, 2015, pp. 199–208.

[2] Y. T. Park, P. Sthapit, and J.-Y. Pyun, "Smart digital door lock for the home automation," in *TENCON 2009-2009 IEEE Region 10 Conference*. IEEE, 2009, pp. 1–6.

[3] S. Skorobogatov, "Flash memory bumpingattacks," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2010, pp. 158–172.

[4] J.-B. Machemie, C. Mazin, J.-L. Lanet, and J. Cartigny, "Smartcm a smart card fault injection simulator," in *2011 IEEE International Workshop on Information Forensics and Security*. IEEE, 2011, pp. 1–6.

[5] M. Pavelić, Z. Lončarić, M. Vuković, and M. Kušek, "Internet of Things Cyber Security: Smart Door Lock System," in *2018 International Conference on Smart Systems and Technologies (SST)*. IEEE, 2018, pp. 227–232.

[6] O. Cekan, J. Podivinsky, J. Lojda, R. Panek, M. Krcma, and Z. Kotasek, "Testing Reliability of Smart Electronic Locks: Analysis and the First Steps Towards," in *Proceedings of the 2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2019, pp. 506–513.

[7] Y. T. Park, P. Sthapit, and J. Pyun, "Smart digital door lock for the home automation," in *TENCON 2009 - 2009 IEEE Region 10 Conference*, Jan 2009, pp. 1–6.

[8] Z. Fonea, "Electronic lock system," Nov. 14 2000, uS Patent 6,147,622.

[9] G. K. Verma and P. Tripathi, "A digital security system with door lock system using RFID technology," *International Journal of Computer Applications*, vol. 5, no. 11, pp. 6–8, 2010.

[10] G. Saether, "Electric stepper motor," Nov. 29 1994, uS Patent 5,369,324.

[11] Kiatronics®, "28BYJ-48 - 5V Stepper Motor," http://robocraft.ru/files/datasheet/28BYJ-48.pdf, 2015, accessed: 2019-03-26.

[12] 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.

[13] O. Cekan, J. Podivinsky, and Z. Kotasek, "Random Stimuli Generation Based on a Stochastic Context-Free grammar," in *Proceedings of the 2016 International Conference on Field Programmable Technology*. IEEE Computer Society, 2016, pp. 291–292.

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

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

[16] MathWork®, "MATLAB and Simulink," https://www.mathworks.com/, 2018, accessed: 2019-03-20.

[17] MathWork®, "Stepper motor," https://www.mathworks.com/help/physmod/sps /powersys/ref/steppermotor.html, 2019, accessed: 2019-03-20.

Appendices

Appendix I

Publications cited by other authors

- Podivinsky, J.; Cekan, O.; Simkova, M.; Kotasek, Z.: The Evaluation Platform for Testing Fault-Tolerance Methodologies in Electro-mechanical Applications. *In: Microprocessors and Microsystems. Amsterdam: Elsevier Science*, 2015, vol. 39, no. 8, pp. 1215-1230. ISSN 0141-9331.

  - McWilliam, R.; Khan, S.; Farnsworth, M.; et al.: Zero-maintenance of Electronic Systems: Perspectives, Challenges, and Opportunities. *Microelectronics Reliability*, volume 85, 2018: pp. 122–139
  - Hao, Z.; Zhang, M.: Cultivation of Mechanical Application Talents Based on FPGA and Machine Learning. *Microprocessors and Microsystems*, 2020, 103499

- Podivinsky, J.; Zachariasova, M.; Cekan, O.; Kotasek, Z.: FPGA Prototyping and Accelerated Verification of ASIPs. *In: IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits and Systems.* Belgrade: IEEE Computer Society, 2015, pp. 145-148. ISBN 978-1-4799-6779-7.

  - Jordans, R.; Jóźwiak, L.; Corporaal, H.; et al.: Automatic Instruction-set Architecture Synthesis for VLIW Processor Cores in the ASAM Project. *Microprocessors and Microsystems*, volume 51, 2017: pp. 114–133.
  - Caba, J.; Cardoso, J. M.; Rincón, F.; et al.: Rapid Prototyping and Verification of Hardware Modules Generated Using HLS. *In: International Symposium on Applied Reconfigurable Computing*, Springer, 2018, pp. 446–458.
  - Caba, J.; Rincón, F.; Dondo, J.; et al.: Testing Framework For in-hardware Verification of the Hardware Modules Generated Using HLS. *In: 2018 28th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, IEEE, 2018, pp. 103–110.
  - Caba, J.; Rincon, F.; Dondo, J.; et al.:FPGA-Based Solution for On-Board Verification of Hardware Modules Using HLS. *Electronics p*, no 12, 2020.
  - Caba, J.; Rincon, F.;Barba, J.; et al.: Testing Framework for on-board Verification of HLS Modules Using Grey-box Technique and FPGA Overlays. *Integration*, volume 68, 2019: pp. 129–138.

- Cekan, O.; Podivinsky, J.; Kotasek, Z.: Software Fault Tolerance: the Evaluation by Functional Verification. *In: Proceedings of the 18th Euromicro Conference on Digital Systems Design.* Funchal: IEEE Computer Society, 2015, s. 284-287. ISBN 978-1-4673-8035-5.

  - Syed Riffat, A.: Next Generation and Advanced Network Reliability Analysis Using Markov Models and Software Reliability Engineering. 2019.
  - Zhang, T.; Wang, J.: A Spatial-Temporal Model for Software Fault Tolerance in Safety-Critical Applications. *In: 2017 IEEE International Conference on Software-Quality, Reliability and Security Companion (QRS-C)*, IEEE, 2017, pp. 575–576.
  - Zhang, T.; Wang, X.: High-Reliable Testing for FPGA Software in Space Utilization Engineering. *In: 2017 International Conference on Dependable Systems and Their Applications (DSA)*, IEEE, pp. 86–91.
  - Cai, B.; Liu, Y.; Liu, Z.; et al.: Bayesian Networks for Reliability Engineering. Springer, 2020.

- Podivinsky J.; Čekan O.; Lojda J.; Kotasek Z.: Functional Verification as a Tool for Monitoring Impact of Faults in SRAM-based FPGAs. In: *Proceedings of the 2016 International Conference on Field Programmable Technology.* Xi'an: IEEE Computer Society, 2016, pp. 293-294. ISBN 978-1-5090-5602-6.

  – Peng, X.; et al.: Function Verification of SRAM Controller Based on UVM. In: *2019 IEEE 13th International Conference on Anti-counterfeiting, Security, and Identification (ASID).* Xiamen, China, 2019, pp. 1-5.

- Podivinsky, J.; Cekan, O.; Lojda, J.; Zachariasova, M.; Krcma M.; Kotasek, Z.: Functional Verification Based Platform for Evaluating Fault Tolerance Properties. *In: Microprocessors and Microsystems. Amsterdam: Elsevier Science*, 2017, vol. 52, no. 5, pp. 145-159. ISSN 0141-9331.

  - Liu, X.; Youan, G.; Qiao, S.: Accelerating Functional Verification for Digital Circuit with FPGA Hard Processor System. *Journal of Electronics Information Technology*, volume 41, 2019: p. 5.
  - Qamar, S.; Butt, W. H.; Anwar, M. W.; et al.: A Comprehensive Investigation of Universal Verification Methodology (UVM) Standard for Design Verification. *In: Proceedings of the 2020 9th International Conference on Software and Computer Applications*, 2020, pp. 339–343.
  - Sabamoniri, S.; Souri, A.: A Weighted Resource Discovery Approach in Grid Computing. *International Journal of Pervasive Computing and Communications*, 2019.
  - Souri, A.; Rahmani, A. M.; Navimipour, N. J.; et al.: A Symbolic Model Checking Approach in Formal Verification of Distributed Systems. *Human-centric Computing and Information Sciences*, volume 9, no. 1, 2019: pp. 4.

- Podivinsky, J.; Lojda, J.; Cekan, O.; Panek, R.; Kotasek, Z.: Reliability Analysis and Improvement of FPGA-based Robot Controller. *In: Proceedings of the 2017 20th Euromicro Conference on Digital System Design.* Vienna: IEEE Computer Society, 2017, pp. 337-344. ISBN 978-1-5386-2146-2.

- Ruiz-Rosero, J.; Ramirez-Gonzalez, G.; Khanna, R.: Field Programmable Gate Array Applications — A Scientometric Review. *Computation*, volume 7, no. 4, 2019: p. 63.
- Wang, L.; Wang, X.; Jia, P.; et al.: Reliability, Safety and Time - Domain Sensitivity Analysis of Double 2-out-of-2 Redundancy System Based on Markov Process and Multiple Beta Factor Model. *In: 2018 3rd International Conference on System Reliability and Safety (ICSRS)*, IEEE, 2018, pp. 153–161.

- Lojda, J.; Podivinsky, J.; Kotasek, Z.; Krcma, M.: Data Types and Operations Modifications: a Practical Approach to Fault Tolerance in HLS. *In: Proceedings of IEEE East-West Design Test Symposium.* Novi Sad: IEEE Computer Society, 2017, pp. 273-278. ISBN 978-1-5386-3299-4.

  - Cao, J.; Zhang, W.; Xiao, Z.; et al.: Reactive Power Optimization for Transient Voltage Stability in Energy Internet via Deep Reinforcement Learning Approach. *Energies*, volume 12, no. 8, 2019: p. 1556

- Podivinsky, J.; Lojda, J.; Cekan, O.; Kotasek, Z.: Evaluation Platform for Testing Fault Tolerance Properties: Soft-core Processor-based Experimental Robot Controller. *In: Proceedings of the 2018 21st Euromicro Conference on Digital System Design. Praha: IEEE Computer Society,* 2018, pp. 229-236. ISBN 978-1-5386-7376-8.

  - Ruiz-Rosero, J.; Ramirez-Gonzalez, G.; Khanna, R.: Field Programmable Gate Array Applications — A Scientometric Review. *Computation*, volume 7, no. 4, 2019: p. 63.

- Čekan, O.; Podivinsky J.; Kotasek Z.: Program Generation Through a Probabilistic Constrained Grammar. In: *Proceedings - 21st Euromicro Conference on Digital System Design, DSD 2018.* Praha: IEEE Computer Society, 2018, pp. 214-220. ISBN 978-1-5386-7376-8.

  – Fitzpatrick, G.: Mind the gap: Modelling the human in human-centric computing. *In: 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC).* IEEE Computer Society, 2018. p. 3-3.}

- Podivinsky, J.; Čekan, O.; Krčma, M.; Burget, R.; Hruška, T.; Kotasek, Z.;. A Framework for Optimizing a Processor to Selected Application. In: *Proceedings of IEEE East-West Design Test Symposium.* Kazan: IEEE Computer Society, 2018, pp. 564-574. ISBN 978-1-5386-5710-2.

  – Mazurek P.: BOSON - Application-Specific Instruction Set Processor (ASIP) for Educational Purposes. In: *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV).* Shenzhen, China, 2020, pp. 1323-1328.