Microprocessors and Microsystems 37 (2013) 155–173

Contents lists available at SciVerse ScienceDirect

Microprocessors and Microsystems

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

# Fault tolerant system design and SEU injection based testing

Martin Straka *, Jan Kastil, Zdenek Kotasek, Lukas Miculka

*Brno University of Technology, Faculty of Information Technology, Bozetechova 2, 612 66 Brno, Czech Republic*

ARTICLE INFO

*Article history:*
Available online 8 September 2012

*Keywords:*
Fault tolerant system
FPGA
Partial reconfiguration
Reconfiguration controller
On-line checker
Duplex architecture
TMR architecture
SEU simulation framework
Fault injection

ABSTRACT

The methodology for the design and testing of fault tolerant systems implemented into an FPGA platform with different types of diagnostic techniques is presented in this paper. Basic principles of partial dynamic reconfiguration are described together with their impact on the fault tolerance features of the digital design implemented into the SRAM-based FPGA. The methodology includes detection and localization of a faulty module in the system and its repair and bringing the system back to the state in which it operates correctly. The automatic repair process of a faulty module is implemented by a partial dynamic reconfiguration driven by a generic controller inside the FPGA. The presented methodology was verified on the ML506 development board with Virtex5 FPGA for different types of RTL components. Fault tolerant systems developed by the presented methodology were tested by means of the newly developed SEU simulation framework. The framework is based on the SEU simulation through the JTAG interface and allows us to select the region of the FPGA where the SEU is placed. The simulator does not require any changes in the tested design and is fully independent of the functions in the FPGA. The external SEU generator into FPGA is implemented and its function is verified on an evaluation board ML506 for several types of fault tolerant architectures. The experimental results show the fault coverage and SEU occurrence causing faulty behavior of verified architectures.



## 1. Introduction

As digital systems become increasingly large and complex, their reliability and availability qualities play a critical role in supporting next-generation science, engineering and commercial applications. Reliability is defined as the ability of a system or component to perform its required functions under stated conditions for a specified period of time. In digital system design, different approaches such as fault avoidance, fault masking and fault tolerance can be used to increase system reliability. Real-time systems are often used in hazardous or remote applications, such as aircraft and spacecraft, where the systems are highly susceptible to errors due to radiation.

### 1.1. On-line testing and fault tolerance of systems

Fault tolerance (FT) is an important feature for many operating environments, from earth applications to space exploration. FT is the ability of a system to operate normally in the presence of faults. This type of reliability is usually attained through replication of hardware such as architectures based on a duplex system, *n*-modular redundancy or the application of self-correcting codes [1]. Triple Modular Redundancy (TMR) uses hardware redundancy to mask any single design failure by voting on the result of three identical copies of the Function Unit (FU). TMR is a popular technique used in many FT schemes. The architecture of a TMR system can be seen in Fig. 1A.

Duplication of a system ensures online fault security and is used in many FT techniques. It requires duplication of FUs, a comparator and a multiplexor of output data. The basic architecture of the duplex system is shown in Fig. 1B. Unfortunately, TMR and duplication of FUs are generally not a very cost effective solution.

The main problems combined with the modern FT systems include error detection during system operation, fast fault localization, quick recovery or repair and bringing the system back to the state in which it operates correctly. One approach on how to construct FT systems with fault detection and fault localization is through the use of on-line checkers [2]. Other approaches combine the principles of 2-rail logic, parity checkers, time redundancy and self-checking (see Fig. 2). These approaches are denoted as Concurrent Error Detection (CED) techniques [3,4].

### 1.2. Field Programmable Gate Arrays

A digital system can be implemented on various platforms. From among those which are widely used in many applications, the reconfigurable hardware should be mentioned. Nowadays, for digital system design, Field Programmable Gate Arrays (FPGA)

* Corresponding author. Tel.: +420 541141362.
*E-mail addresses:* strakam@fit.vutbr.cz (M. Straka), ikastil@fit.vutbr.cz (J. Kastil), kotasek@fit.vutbr.cz (Z. Kotasek), imiculka@fit.vutbr.cz (L. Miculka).

0141-9331/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.micpro.2012.09.006

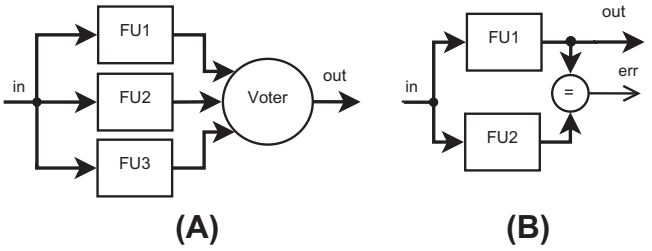


**Fig. 1.** FT architectures: (A) TMR system and (B) duplex system.

can be used [5]. In particular, FPGA-based systems are very valuable for remote and long-time missions because of the possibility of being reprogrammed by the user as many times as necessary in a very short period. These properties of FPGA circuits and concurrent online testing become a strong feature in the design of FT systems [6].

In an SRAM-based FPGA, the combinational and sequential logic are implemented in programmable Complex Logic Blocks (CLBs) which are customized by loading configuration data (bitstream) in the SRAM cells of the configuration memory. However, when a charged particle strikes a memory cell in the configuration memory, the effect can produce an inversion in the stored value and this can modify the function of the design. This event is denoted as the Single Event Upset (SEU) [7]. An efficient set of SRAM-based FPGA mitigation techniques should cope with the Single Event Transient (SET) occurring in combinational logic and SEUs in storage cells. In this way, transient faults in the combinational logic will never be stored in the storage cells, and bit flips in the storage cells will never occur or be immediately corrected. Each technique has some advantages and drawbacks, usually it must be an accepted compromise between area overhead, performance and fault tolerance efficiency.

In order to recover or repair a system quickly when SEU or SET is detected in the FPGA by a CED or bitstream scrubbing techniques, the reconfiguration or partial dynamic reconfiguration (PDR) of an FPGA circuit can be used [8]. The main reason why a PDR has become an available feature in FPGA based implementations is seen in the possibility of modifying or reloading configuration memory while the application is correctly working. This situation is shown in Fig. 3. On the Figure, the TMR architecture can be seen. If one of three function units of TMR is faulty, TMR still provides correct values at the output and the faulty module (FU3) can be repaired by PDR without stopping FPGA. Otherwise, if another module (FU1) is faulty then TMR produces incorrect results at the output and both modules (FU1, FU3) must be repaired by PDR as well.

This property can be used for the design of a highly dependable system based on FPGA. The next problem with an FT system design into FPGA is the occurrence of a permanent fault in the device. For this type of fault, the process of design rerouting and reconfiguration into other parts of the FPGA can be used. The FPGA-based FT methodologies which aim at increasing reliability parameters are based on an FU replication and the use of CED techniques. The FT system implemented into FPGA can be realized at various design levels.

### 1.3. Granularity of system replication in FPGA

The design of FT systems into FPGA with high reliability and availability parameters can be implemented with different levels of replication (see Fig. 4):

- The replication on the level of separate FPGA units, the component is tripled as a TMR of three separate FPGAs, their outputs are compared by one majority element outside the FPGA. This technique is often implemented in despite of higher prices, power consumption and the size of the implementation (Fig. 4A).
- The replication on the level of one FPGA unit where the system is replicated in the same FPGA (Fig. 4B).
- The replication on the level of functional units implemented into one FPGA (Fig. 4C).
- The replication on the level of basic elements in one FPGA (the components like counters, decoders, multiplexers, and adders) (Fig. 4D).

In the last three configurations, PDR of the FPGA can be used. This option is supported by some vendors, for example, Xilinx [9]. In recent years, several methodologies of an FT system design in an SRAM-based FPGA were widely discussed in numerous papers [10–14].

### 1.4. FT methodologies for FPGA-based designs

A methodology to design FT devices implemented to SRAM-based FPGAs that are able to recover from SEU faults based on the use of a duplication with a comparison technique and a PDR is presented in [8,15]. The first approach applies CED techniques to monitor the health of the system and to trigger the reconfiguration of the portion of the device that has been hit by the SEU, while the rest of the system need not be stopped or entirely reconfigured. The second approach uses PDR combined with TMR in SRAM-based FPGAs FT designs. The approach uses a large grain TMR with special voters capable of signalizing the faulty module and check point states that allow for the sequential synchronization of the recovered module. The synchronization of the recovered module is performed while the others are kept running. The architecture of the approach is shown in Fig. 5.

A hardware scheme allowing for the diagnosis of transient and permanent faults affecting a TMR system implemented by means of FPGA is proposed in [16]. The presented FT scheme allows one to easily identify whether a fault affects one of the replicated modules, the voter or the scheme itself and whether such a fault is permanent or transient. The global structure of the proposed scheme is shown in Fig. 6. The scheme can be used to drive the selection of

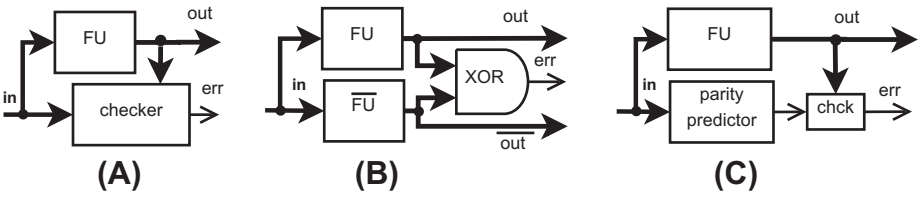


**Fig. 2.** CED techniques: (A) on-line checker, (B) 2-rail logic, and (C) self-checking.

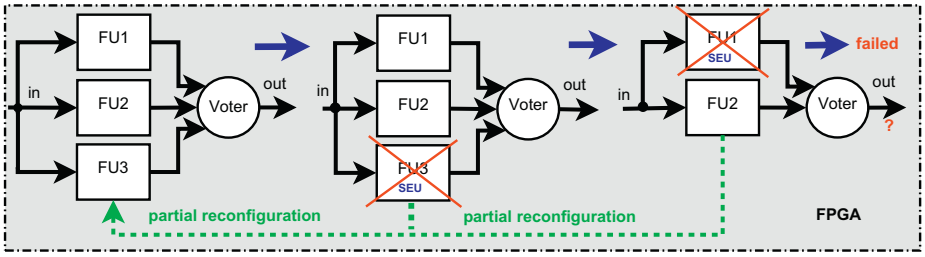


**Fig. 3.** TMR vs. reconfiguration of FPGA.

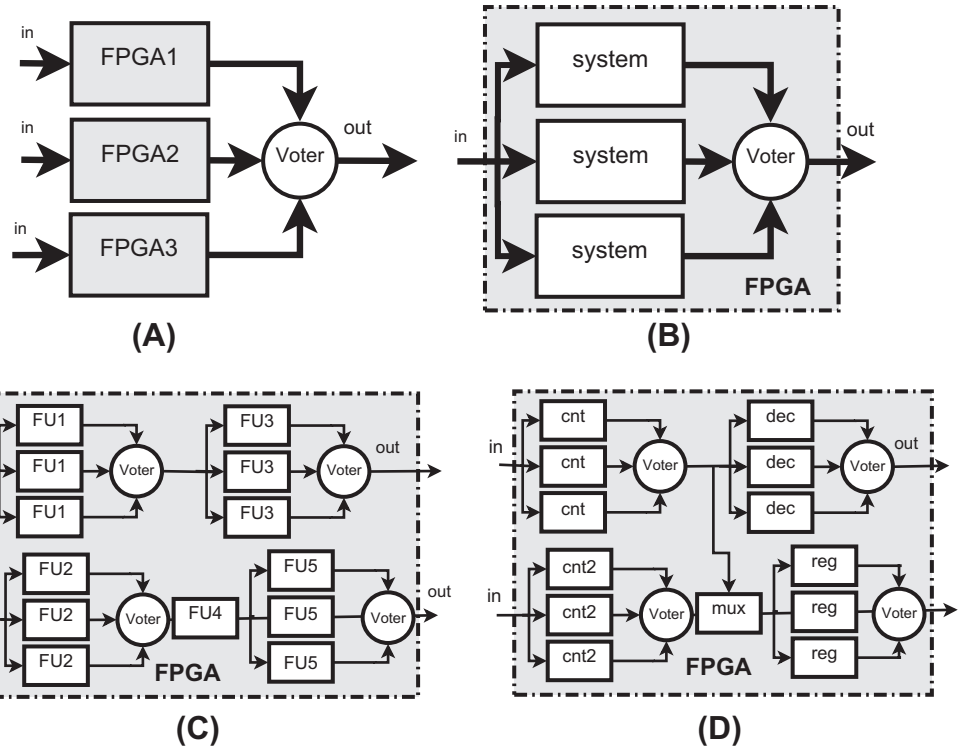


**Fig. 4.** Replication granularity of FT system based on FPGA.

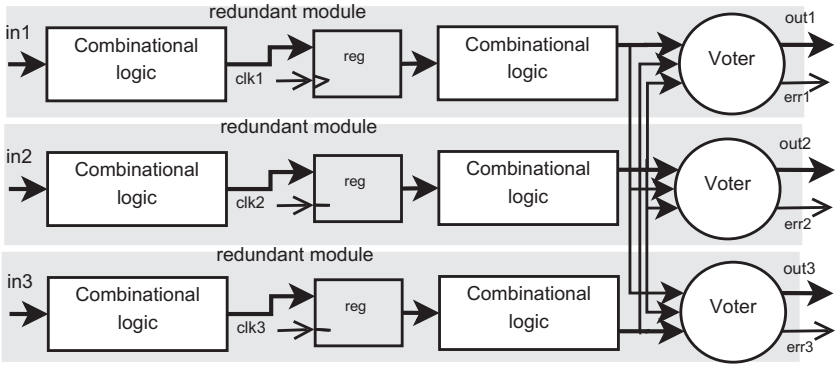


**Fig. 5.** The large grain TMR proposed scheme.

the best proper recovery technique for each kind of diagnosed fault (e.g. by the partial reconfiguration of FPGA).

A methodology for SEU detection and voting that combines a duplex system with a CED based on full time redundancy for the users combinational logic in SRAM-based FPGAs is described in [17]. This methodology reduces the number of input and output pins of the user combinational logic. In addition, it can also reduce the area when large combinational blocks are used (see Fig. 7).

The [18] deals with the possibility of permanent fault in CLB. The paper describes a new type of CLB called CLBm. This type of CLB consists of several classical CLB. CLBm is surrounded by classical CLBs. If the fault occurs, the CLBm is reconfigured to perform

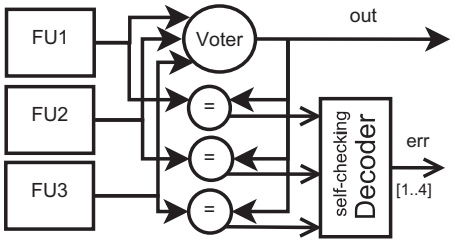


**Fig. 6.** Global structure of the proposed scheme for TMR.

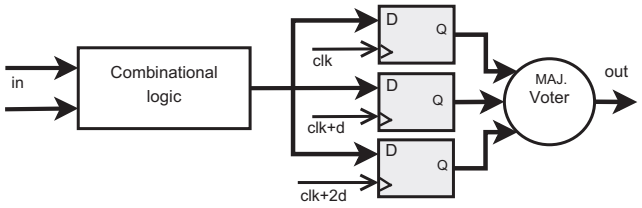


**Fig. 7.** Full time redundancy scheme to correct SET in combinational logic.

the function of the faulty CLB. Since there is several CLB inside one CLBm, multiple faults can be corrected. The disadvantage of this approach is that it works only for CLB and therefore it is not able to repair faults in routing resources or in hardblocks in FPGA (such as blockRAM).

Other FT techniques for SRAM-based FPGAs are discussed in [19,20]. These techniques are based on circuit level modifications with obvious modifications in the programmable architecture or techniques that can be implemented at a high-level description without modification in the FPGA architecture. The high-level method based on a TMR and a combination of a duplex with CED techniques which are able to cope with upsets in combinational and sequential logic is presented in [21].

In [22], the adoption of the TMR coupled with the PDR of SRAM-based FPGAs to mitigate the effects of SEU and SET in such classes of device platforms is shown. The authors propose an exploration of the design space with respect to several parameters (e.g. area and recovery time) in order to select the most convenient way to apply this technique to the device under consideration.

The dynamic scheme for Xilinx FPGA FT systems design is presented in [23]. The scheme consists of two parts: the Partially Reconfigurable Functional Region (PRFR) with several Partial Reconfigurable Modules (PRM) and a reconfiguration controller that is based on a built-in Xilinx PowerPC-405 processor (see Fig. 8).

Attempts to categorize and compare the various FT techniques and discuss how they can work together to provide a synergetic approach for a fault tolerant FPGA design are presented in [24], but unfortunately no experimental comparisons were presented.

### 1.5. Principles of partial dynamic reconfiguration in FPGA

PDR is a technique that allows us to change only a part of the design in FPGA without disturbing the operation of other parts of the design. It can be used to increase the security [25] of the design, to reduce resource consumptions through time multiplexing or to increase reliability of the system. But there still exists a disadvantage in complex methodology needed to prepare the bitstreams for PDR.

In Xilinx FPGA, the bitstream contains commands for the configuration engine followed by the actual configuration information. The main difference between dynamic and static reconfiguration is in the commands in the bitstream. During partial static reconfiguration, a command to stop all computation in FPGA is sent at the beginning of the reconfiguration process. In PDR bitstream, the stop command is missing and, therefore, all parts of the FPGA are active during the reconfiguration. This may lead to unexpected changes of output signals of the unit under reconfiguration.

### 1.6. Partially Reconfigurable Modules

The Partial Reconfigurable Region (PRR) is the part of the FPGA that can be modified during PDR. The size and the position of the region are determined by area constrains. Partially Reconfigurable Module (PRM) is the part of the design that can be implemented into PRR. The reconfigurable region can contain only the logic that is associated with some reconfigurable module. This limitation does not apply to the routing resources and it is up to the routing program to determine which route should be used for a particular signal. Therefore, routing may pass directly through the reconfigurable region. The connections between PRM and the rest of the design have to be realized through a special interface. In older versions of ISE this interface was built from BusMacros which were not supported by tools for timing analysis and automatic placement. Therefore, a user was forced to place every BusMacro in the design manually which was error prone work. However, since version 11, ISE uses ProxyLogic which can be placed automatically. ProxyLogic is supported by the timing analysis and it is possible to have a time critical path between PRM and the static part of the design.

The frame is the smallest part of the FPGA that can be reconfigured and it has a size of 1312 bits in the Virtex5 [9]. The minimal size of the reconfigurable module is theoretically one CLB, but due to the structure of the configuration memory, CLB configuration is contained in several frames and one frame contains a configuration of twenty CLBs (see Fig. 9). Since the configuration engine has to reconfigure the whole frame, every reconfiguration changes at least twenty CLBs.

### 1.7. Reconfigurable interfaces

Performing PDR is a simple process of downloading a partial bitstream into the device. The reconfiguration of the FPGA can be performed through various interfaces, but only some of them remain accessible after the first configuration is done.

- *JTAG* interface has the highest priority from among other configuration interfaces, it can even interrupt a running reconfiguration process. A relatively slow configuration can be seen as the disadvantage of JTAG because data are transferred in the serial mode at a maximum speed of 24 Mbps.
- *SelectMap* is a faster configuration interface than JTAG because it supports a parallel reconfiguration and higher clock frequencies. The parallel nature of the interface can be a disadvantage in applications which require high pinout utilization.
- *ICAP* – Internal Configuration Access Port is an internal interface for FPGA reconfiguration. ICAP can operate on up to 100 MHz and accepts 32 bits of the configuration bitstream per clock cycle. ICAP is often used together with a high speed serial connection to the bitstream storage unit. Reconfiguration by the ICAP interface can load the bitstream into the device with a speed of 3.2 Gbps, but it is often a problem to construct a bitstream storage capable of delivering a bitstream at such a speed. Since the smallest PRM block contains about 20 CLBs and its size is about 6 kB, the time required for the partial reconfiguration is about 28 μs.

### 1.8. FPGA bitstream readback vs. bitstream scrubbing

For some types of FPGAs (Xilinx), techniques exist which allow us to detect soft faults in bitstream or by means of scrubbing in designs implemented into the FPGA. The main idea is to read the

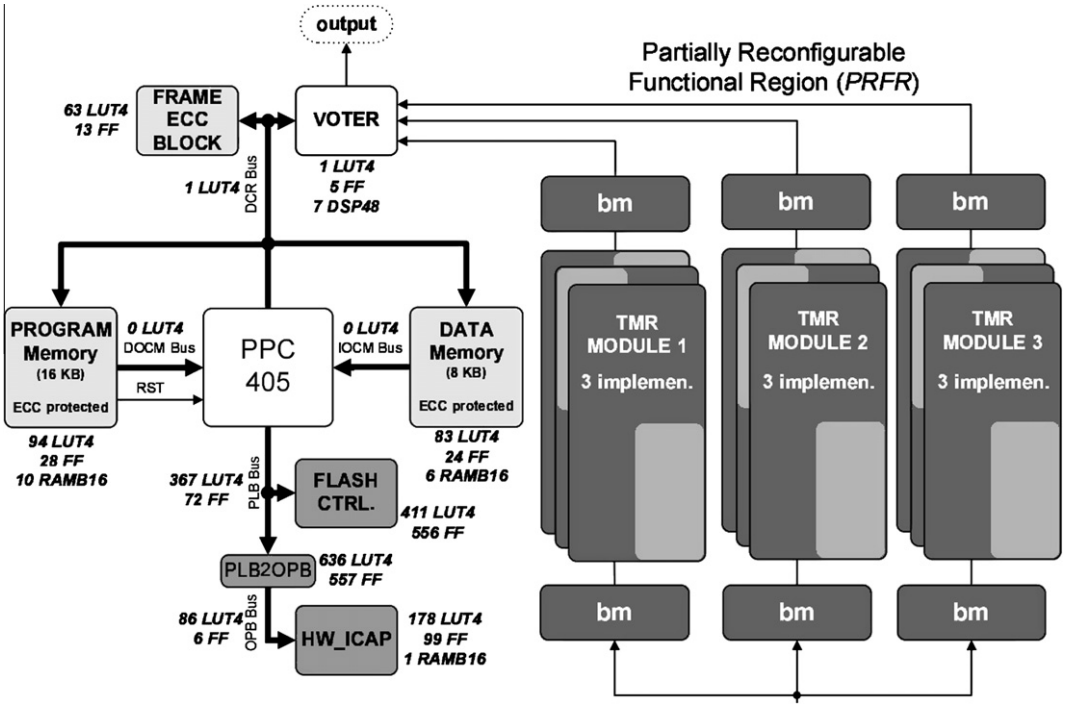


**Fig. 8.** FPGA V4 FT systems design scheme with PowerPC [23].

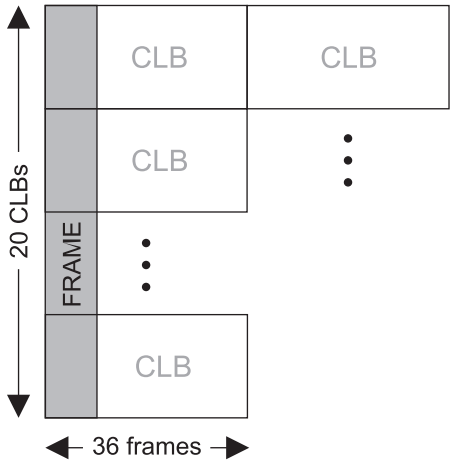


**Fig. 9.** Frame vs. CLB.

configuration of the FPGA and compare it to the image stored in the nonvolatile memory. It is possible to compute the hash value of different parts of the bitstream and then compare them. This will result in a smaller memory requirement for bitstream scrubbing. Bitstream scrubbing was integrated into modern Xilinx FPGA at the bitstream level. Every configuration frame contains several bits that serve as an Error Checking Code (ECC) for the frame value. Therefore, it is possible to read back only one frame and detect an SEU in it.

Bitstream scrubbing is not able to guarantee the correct operation of the design when an SEU is detected. Therefore, bitstream scrubbing is only an auxiliary method used in the design of FT systems. Moreover, it is not possible to detect SEUs in the memory and the registers since their values change in time and cannot be tested by the ECC. It is the responsibility of user designs to detect errors in these parts of the FPGA. Also, any error that is not caused by a memory failure, such as a faulty unit in the FPGA, cannot be detected by scrubbing. Therefore, for FT design in the FPGA, the bitstream scrubbing has to be combined with an error mitigation technique such as TMR [26].

## 2. Motivation for the research and problem definition

With the progress of modern technologies, reconfigurable architectures have penetrated into new fields of industry. FPGAs are used in real world products due to the combination of their computational power and the ability to reconfigure. Reconfiguration can be used to allow for fast and simple updates or upgrades after the deployment of the product; or to implement a new advanced feature such as a software defined radio in the hardware. With the increasing use of FPGAs there is also the need to increase the reliability of the solutions based on these platforms.

For the correct function, the FPGA has to be configured by loading configuration data into its configuration memory. Usually, FPGAs are equipped with an SRAM configuration memory which allows for fast and frequent reconfiguration, but which also makes them very susceptible to SEU errors. The high energy particle can change the function of the application by changing the internal configuration memory. Since FPGAs are starting to be used in devices where high reliability is required, such as car control systems or avionic systems, such an error can have serious consequences and has to be mitigated before it causes an incorrect function in the system. The probability of an SEU attack is relatively small due to a low radiation background on the Earth. However, if the FPGA is used outside the Earth's atmosphere or the FPGA based designs are produced in massive numbers, the probability of an SEU attack will rapidly increase. The FPGA vendors work on developing methods for preventing SEUs, such as using radiation hardened FPGAs, but these methods increase the cost of resulting products and introduce additional overhead. Another approach is to mitigate an SEU after it appears. The issue of SEU mitigation is an actively researched problem.

Fortunately, FPGA reconfiguration principles allow for simple SEU mitigation by reconfiguring the faulty part of the system. Modern FPGA supports PDR, which permits repair of the faulty unit

without disturbing the operation of other parts of the chip. The system has to satisfy two main conditions in order to be able to use PDR for SEU mitigation. Firstly, it has to be able to detect an occurrence of the SEUs and hard errors; and secondly, it has to be able to produce correct results before the SEUs or hard errors are detected and repaired. The methodology for construction designs which satisfy both of these criteria is the main contribution of this work together with the system for driving the PDR process.

At the moment, many techniques on how to develop SRAM-based FPGA systems exist. Many of them are based on replication of functional units (e.g. TMR and duplex architectures) combined with CED techniques. They are not usually tested on how vulnerable they are against SEU effects. It is also not described which technique was used to verify FT features of the architectures developed. It is important to say that some tools based on the use of ICAP to insert SEUs into the FPGA bitstream exist [27,28]. These techniques can achieve a high speed by their injection, but their main disadvantage is that the injector is implemented into the same FPGA as the application and thus can damage itself by injecting SEU. Moreover, the presence of testing circuitry affects place and route phases of the design and, therefore, produces different designs with different fault tolerant properties. Our approach is based on the external SEU generator which allows us to test the behavior of our FT architectures and their reaction to SEU inserted into particular positions of the bitstream.

The problem we are solving can be summarized in the following way:

- How to implement an FT system into an SRAM-based FPGA when the principles of PDR as correction and recovery mechanisms are used (reconfiguration of faulty modules)?
- How to detect multiple soft errors and hard errors in FPGA-based systems?
- How to mitigate soft errors and hard errors in FPGA-based systems?
- How to drive a partial reconfiguration process inside the FPGA as a recovery mechanism after errors are detected in the system?
- How to inject SEUs into the FPGA and how to test the behavior of FT architectures after SEUs are injected?

To summarize, in this paper we present a methodology of FT systems design based on the use of PDR (supported by the Virtex FPGA family) as an alternative to SEU mitigation techniques based on bitstream scrubbing with TMR. As described in the following sections, our approach allows to detect and recover from multiple transient faults in multiple modules of FT system, as e.g. SEU occurrence in FPGA configuration memory, registers or functional blocks of the design, permanent fault detection and localization in FPGA interconnections (the recovery from these faults is not solved), suppression of SET type faults by implementing multiple clock signals in FT architectures. The methodology satisfies basic conditions required for fault tolerant systems under the assumption that voters and multiplexors in FT architectures and GPDRC are not affected by faults. Two basic conditions for FT systems are:

- The operation of the system was not interrupted or stopped after a fault occurred.
- The system can produce correct outputs even when a fault exists in the system.

We also present an SEU simulation framework for the testing of FPGA-based fault tolerant systems. The framework is based on SEU generation in a Personal Computer (PC) and the transport of the bitstream through the JTAG interface and dynamic reconfiguration into the FPGA which allows us to select region of the FPGA for SEU placing.

The paper is organized as follows. In Section 3, detection techniques based on the use of on-line checkers and functional module duplication with a comparison are described. These techniques are used in the proposed FT methodology. The basic principles of the proposed methodology and FT architectures based on checkers are presented in Section 4 together with the description on how these FT architectures are implemented when reconfiguration modules are used. The correction scheme and recovery principles of the FT structure after detection of an error in the system together with architecture of partial reconfiguration controller are described in Section 5. An SEU simulation framework for testing fault tolerant system design in the FPGA and requirements on the external SEU generator together with its basic features and implementation are demonstrated in Section 6. Then, the results of our experiments (Section 7) together with the comparison of the proposed methodology with other FT techniques (Section 8) are described.

## 3. Detection of errors in FPGA-based designs

Our previous research was oriented towards creating a methodology which allows us to construct on-line checkers for components at different levels, for example, module or Register-Transfer Level (RTL) components. An on-line checker can be used for error detection in the component or for the identification of faulty units in FT architecture. Checkers can be used in on-line testing methodologies or in FTS design.

From among languages which can be used to describe functions checked by checkers, Property Specification Language (PSL) can be used [29]. Unfortunately, the tools which use PSL generate a checker VHDL code which is primarily supposed to be used for hardware simulation of conditions during design verification, not for the synthesis into a target platform and the use as a checker. Therefore, we do not see PSL and its software support as a proper alternative to satisfy our objectives. Other tools exist for the description of conditions required to be satisfied by the design (e.g. System Verilog Assertions (SVA) languages). The software packages which exist to support them are also intended to be used primarily for design verification purposes [30].

Therefore, we need another methodology which allows us to construct on-line checkers for components at different levels of RTL components and their combinations (see Fig. 10). For this purpose, the specialized formal model based on Finite State Machines (FSM) was developed which allows us to describe the properties to be checked. Different levels of properties can be described and transformed into the conditions for correct behavior of the circuit. The conditions are then compiled into a checker VHDL code which can be then integrated into the resulting design together with the FU which will be checked by the checker. The principles of methodology for generating an on-line checker for simple circuits can be seen in Fig. 11. The properties and description of the methodology were presented in [2] together with the definition of the formal model. The basic principles of checkers application into FT architectures were described in [6]. This methodology allows us to develop on-line checkers for a communication protocol as well.

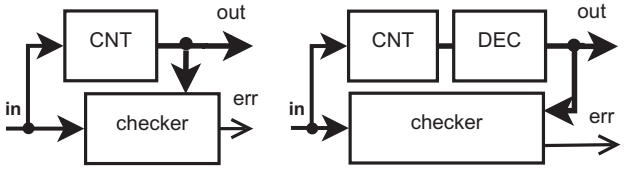


**Fig. 10.** Example of RTL component with a checker and their combinations.

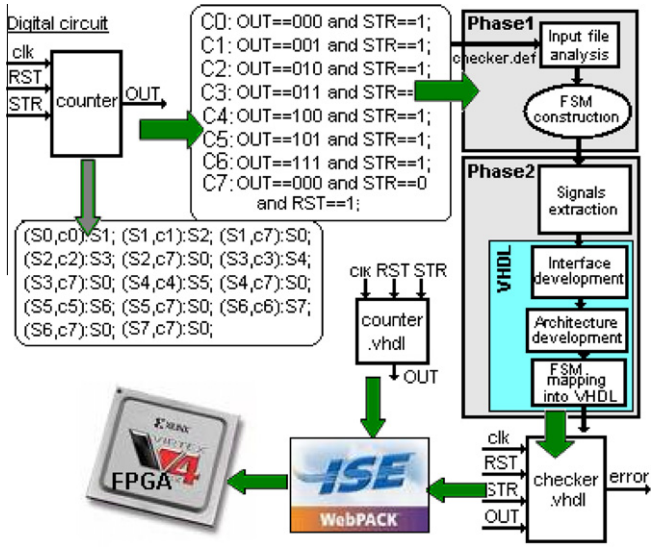


**Fig. 11.** Demonstration of methodology principles for a counter. For the counter and its checker synthesis, the XILINX ISE 11.3 tool was used.

Finally, we use on-line checkers and the Duplication with Comparator (DwC) technique for error detection in the TMR and duplex systems. These concepts are then used in FT architectures design based on a PDR.

## 4. Highly dependable systems design into FPGA based on partial dynamic reconfiguration

FPGA based designs offer new possibilities for the activities which aim at designing an FT system with high reliability and availability of the system. In FPGA, a faulty module can be repaired by reconfiguring the chip. In this situation, the function mode of the system is interrupted and the reconfiguration process of an FPGA chip is initiated. For this purpose, the principles of the PDR can be used where the reconfiguration process is applied on the faulty module without interrupting the system. This type of fault repair during the system runtime is supported by hardware redundancy architectures, such as the TMR or the duplex system. The proposed methodology implements these principles:

- Detection of multiple soft and hard errors in the FPGA-based system (on-line checkers, DwC).
- Mitigation of all errors for FT architectures based on replication of FUs.
- Localization of soft errors in FT architectures and recovery from these errors (reconfiguration controller).
- Localization of hard errors in FT architectures (the recovery from these errors is not solved).
- Synchronization of the design after the reconfiguration process.
- Testing of correct behavior of FT architectures on SEUs (SEU simulation framework).

The FT design based on PDR for the recovery of faulty modules must be divided into its dynamic and static part. In the dynamic part, the components which will be designed as FT architectures are included, where they will be reconfigured by means of PDR in the case of an incorrect operation. These FT architectures must be equipped with the necessary diagnostic background (i.e. fault detection techniques). In the static part such components will be included which are not designed as FT and are not supposed to be reconfigured (as they are not critical for overall functionality). The methodology supports the detection and correction of all faults caused by SEUs and the detection of hard errors. The methodology satisfies basic conditions required for fault tolerant systems as the operation of the system was not interrupted or stopped after a fault occurred and the system can produce correct outputs even when a fault exists in the system. The requirements on the robustness of the system and its power consumption must also be taken into account. The main structure of the proposed methodology for the FPGA-based FT system design with a Partial Reconfiguration Controller (PRC) inside the FPGA can be seen in Fig. 12.

FT architectures in the dynamic part of FPGA can be mutually interconnected sequentially, through a bus, they can form any type of graph structure, the number of interconnected architectures is not limited, the only limitation factor being the size of FPGA. In our methodology, PRC was developed to be used for PDR recovery and the synchronization of PRM in case a fault was identified in it. To store partial bitstreams reflecting PRC and FT architectures, external flash memory was used.

As most of components of PRC are implemented as fault tolerant units (through TMR and duplex architectures) and they can be reconfigured by PDR, they are included into the dynamic part of FPGA. The PRC contains also other components as FIFO and ICAP wrapper which must be included into the static part of the design.

### 4.1. Detection of faulty modules in FT architectures

In this section we present FT architectures designed in a way which allows us to detect their internal failures. We use on-line checkers and the DwC technique for error detection in FT architectures. The basic idea is based on the assumption that each FU and its checker are configured into one PRM. Therefore, it is possible to recover its function (through reconfiguration of the PRM) while the rest of the implementation is still working. An on-line checker can be used for error detection and fault localization of PRM in the FT architecture. The methodology for generating an on-line checker for the digital system and simple circuit based on RTL was presented in Section 3. The FT architectures based on PRMs can be seen in Fig. 13.

The first FT architecture (TMRcmp) uses a classical TMR scheme with comparators in which the outputs of all units are checked by the voter. Other FT architectures use duplex architecture with checkers (DUPLchck) or duplex architecture with a checker and comparators (DUPLchckcmp).

### 4.2. Main structure of FT design based on PRMs

As a result of our research, we can make suggestions on how a PDR can be used to increase fault tolerance parameters (e.g. availability and dependability). In this section we describe the structure of FT architecture which is designed in a way that allows us to detect internal failures in PRMs. The concept of checkers and comparators is used in our strategy. Each FU and its checker are configured into one PRM. Therefore, it is possible to adjust its correct function (through reconfiguration of the PRM) while the rest of the implementation is still working.

The methodology supports detection and correction of all kinds of faults caused by SEUs. If the fault still exists in the module after reconfiguration, it will be considered as a hard error. In this case the hard error can be removed by loading the latest bitstream after the new mapping of the system into FPGA. Therefore, the static part reconfigures the module to perform tests of its FPGA fabric and the results of these tests can be used to resynthesize FU of the module without the use of the faulty fabric (this circuitry must be excluded from the new implementation). Unfortunately, the structure of the FPGA fabric is confidential and, therefore, cooperation with the FPGA vendor company would be

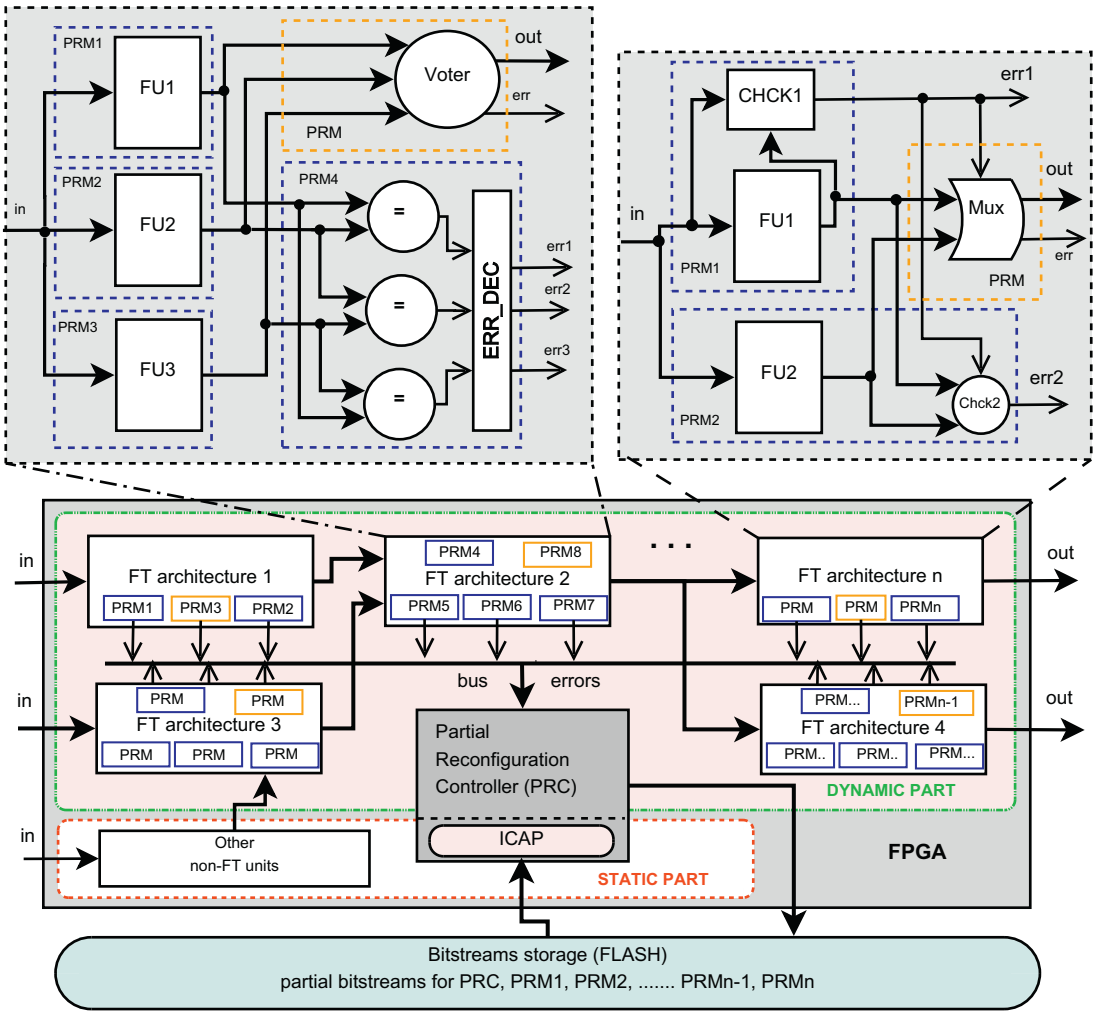


**Fig. 12.** The structure of FT designs in FPGA based on PDR.

required in order to implement the extension. Hard error simulation is not seen as a simple problem and, therefore, in our research we focused on soft errors only. The function of the FT structure and control flow of the repair process is shown in Fig. 14.

If more than one SEU occurs, a round robin algorithm will select one of the PRMs for reconfiguration. After successful reconfiguration, the system continues with the next fault until all of them are removed. As long as every module operation is attacked by one SEU, it is guaranteed that the whole system operates correctly. If some of the errors cannot be repaired by PDR, a round robin guarantees that such errors will not terminate the recovery process of remaining PRMs.

PRMs modules containing voters or multiplexors are critical parts of the design and they are responsible for evaluating the outputs of FUs. These PRM modules are designed as self checking units (for example, by using 2-wire logic). Therefore, it is possible to detect error in these module, but it is not possible to correct it without destroying the function of the device. The presented structure solves this problem by suspending the design for the time needed to perform voter or multiplexor PRM reconfiguration. PRMs are stopped by disabling the clock signal which can be implemented in the Virtex5 FPGA by a clock buffer (BUFGCE). A static part of the FPGA can also contain non-critical logic, its failure will not cause any problems with covering the function, such as identification registers.

## 5. Error correction and recovery of faulty modules

The proposed methodology relies on the possibility of a module implemented into the FPGA to be repaired. The repair process has to be performed without disturbing any other part of the design, except for the faulty unit. Partial dynamic reconfiguration is one of the options that can be used for the implementation of the repair process.

### 5.1. Partial reconfiguration controller

Partial reconfiguration is a process during which the bitstream is loaded into the configuration memory. The complexity of this process may vary from a simple memory copy into an extremely complex problem with bitstream modifications. However, even the simplest partial reconfiguration has to be initiated. The initiation of the reconfiguration process and its driving is the responsibility of the Partial Reconfiguration Controller (PRC). In order to limit the number of FPGA chips in the system, we consider a partial reconfiguration controller to be implemented into FPGA. With PRC in the FPGA, a higher reconfiguration speed is gained.

There are several possibilities on how to implement PRC. One way is to use a micro processor to drive partial reconfiguration. Micro processors are easy to use and offer many operations on the bitstream such as decompression or even modifications of the configuration bits [31]. It is important to note that a malfunction

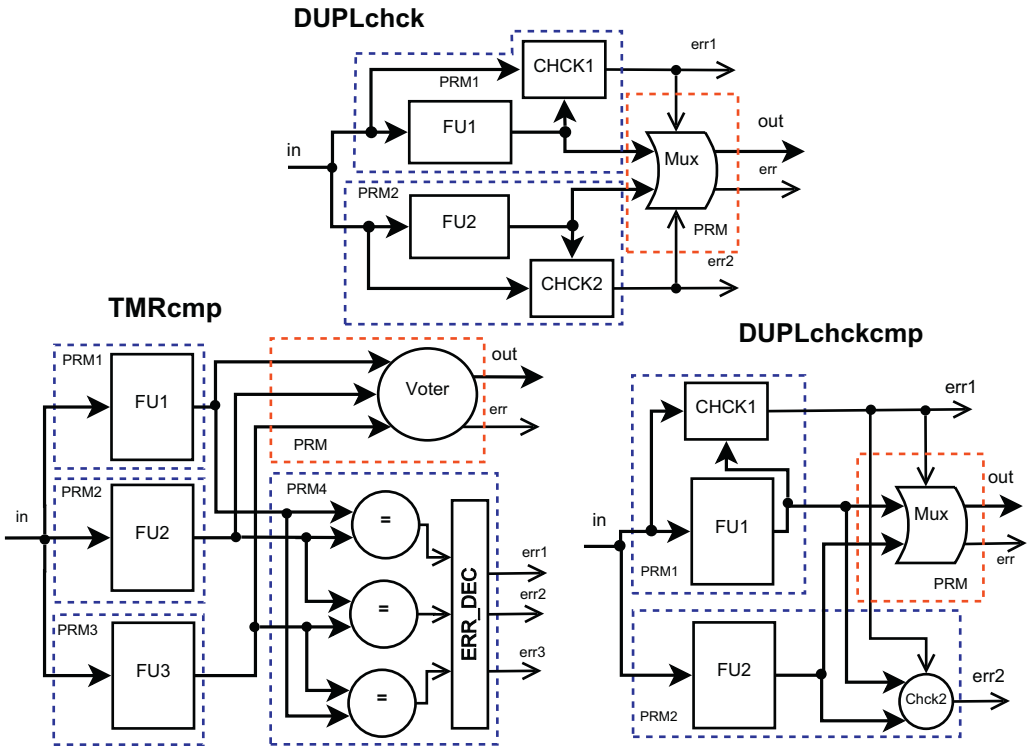


**Fig. 13.** Fault tolerant architectures based on PRMs for a design implemented in the FPGA.

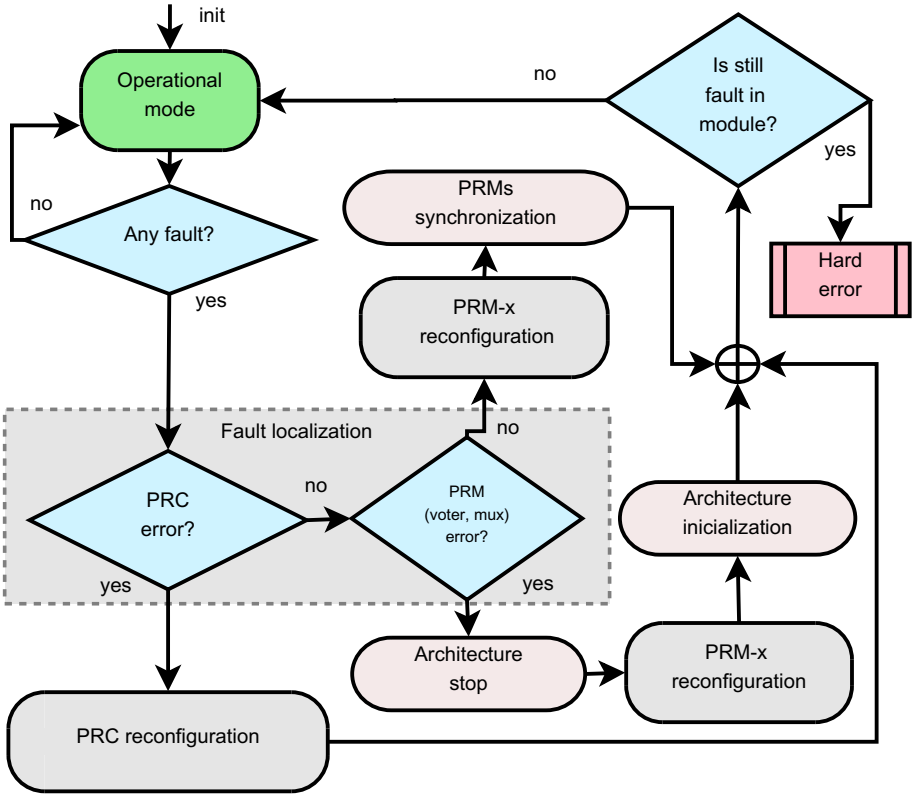


**Fig. 14.** Control flow of repair process.

in the reconfiguration process can break the whole system. Proper behavior of a processor depends on the software running on it, therefore, software of the PRC should be formally verified. Moreover, it is possible that an SEU will affect the values in the processor registers. It is hard to design software which is FT to such types of errors.

Some FPGAs contain hard blocks implementing processors. The Xilinx Virtex family contained the PowerPC cores. However, newer versions of Virtex FPGA are not equipped with PowerPC. If the FPGA does not contain a processor hard block, it has to be implemented in the FPGA fabric as a soft core. MicroBlaze is an example of a soft core processor. Soft core processors are susceptible to SEU errors.

It is important to keep in mind that processors have often more computational power than is necessary to perform partial reconfiguration. If they are used only for the PRC, their performance is wasted. The wasted performance causes higher power consumption and higher complexity of the solution which increases the probability of failure. The micro processor can perform another computation, however, an error in any software module may delay or even stop the reconfiguration process.

We decided to implement PRC as a hardware unit instead of using a processor to reduce resource utilization and thus reduce the failure probability in the controller. The developed PRC is called a Generic Partial Dynamic Reconfiguration Controller (GPDRC).

### 5.2. Architecture and features of GPDRC

The interface of a GPDRC is shown in Fig. 15. The GPDRC unit has three logical interfaces, one for reading the error status from the architecture (Err-PRMs, rst, clk), the second one is used to communicate with external memory (Bitstream,valid,addr-bits) and the last one reports hard errors (hard, PRM-index).

Error signals from all PRMs in an FT structure are brought to a GPDRC and connected to bus **Err-PRMs**. If the GPDRC decides to start reconfiguration it uses **addr_bitstr** bus to write a bitstream address into the external memory (Flash or ROM). The memory returns the configuration data through a **bitstream bus** and uses a **valid** signal to confirm data correctness. In the case of hard error detection, signal **Hard** is set to high and the **PRM_index** bus contains the index of the PRM containing hard error.

In Fig. 16, the detailed architecture of a GPDRC developed by our team is shown. The architecture of a GPDRC consists of five units, one FIFO memory and one FSM which drives each unit.

If more than one SEU occurs (more error signals are active), a round robin algorithm chooses one of the PRMs which will be reconfigured. The Generic Error Encoder (GEE) decodes the binary index of this PRM and forwards its identification number together with an error identification signal to the Look Up Unit (LUU) and Hard Error Unit (HEU). The HEU unit checks if the fault still exists after the reconfiguration and synchronization of the faulty unit. If the fault exists after the repair process, it is considered to be a hard error.

The LUU returns the address to the bitstream storage where the PRM is located. The index of the PRM which should be reconfigured

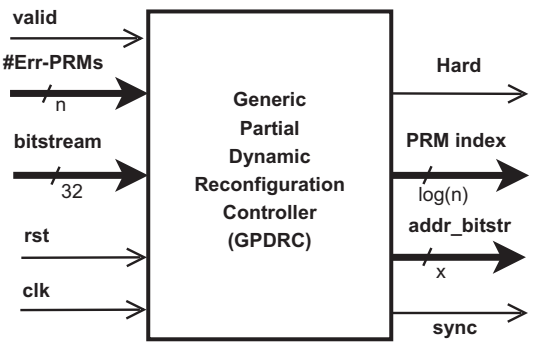


**Fig. 15.** Interface of the GPDRC for FT system implemented in SRAM-based FPGA as PRMs.

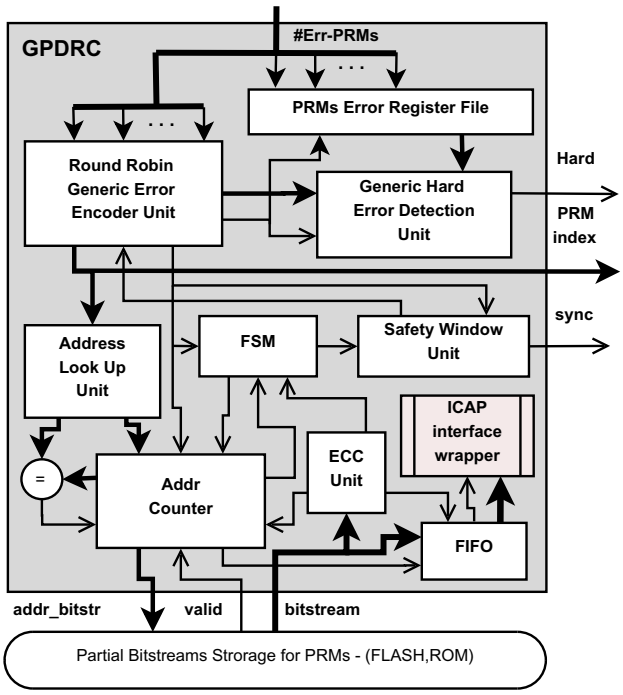


**Fig. 16.** Architecture of the GPDRC for an FT system implemented in SRAM-based FPGA as PRMs.

is the input to LUU. After LUU translates the index of the PRM into the bitstream storage address, the FSM unit starts the reconfiguration process. The LUU also returns the last address of the partial bitstream. The Address Counter is used to read the complete partial bitstream. When the value in the address counter is equal to the end address returned from the LUU, the FSM finishes the reconfiguration process.

Before entering the ICAP, data are stored in the FIFO. The size of FIFO is exactly one frame. When the whole frame is in the FIFO, the ECC unit is able to verify that the actual frame is without errors. The FIFO also performs the transformation from the memory bus to the ICAP interface by widening bus width.

The Safety Window Unit (SWU) checks that every unit has enough time for synchronization. Since the GPDRC contains only one SWU, all functional units must be able to synchronize in the same time window.

After successful reconfiguration, the system will continue with processing the next fault until all of them are repaired. As long as every module operation is attacked by at most one SEU, it is guaranteed that the whole system will operate correctly.

### 5.3. GPDRC implemented as TMR architecture with PDR

As GPDRC is a very important component in our methodology (it controls the process of reconfiguration), it must be implemented as FT. Most of GPDRC units are configured into the dynamic part of FPGA, they can be implemented as independent PRMs or FT architectures, their function can be recovered by means of PDR. Therefore, GPDRC was designed as an FT component, all GPDRC were implemented as TMR architectures with a voter. These TMR architectures form an independent PRM which can be repaired for example by bitstream scrubbing.

### 5.4. Synchronization problem and safety window

The PDR is able to repair a fault that caused an error in a PRM, but the state of the module after the reconfiguration process is

undefined. Two approaches exist to set the internal state of the unit to a correct value. The first method copies the current state from the other implementation of the unit in the system. This method is relatively complex, but it can be used in any type of system.

The second method can be used in systems which process packets. Before a packet is received, the units are synchronized by a local reset signal at the end of preceding packet. This packet based method should be preferred whenever possible because its implementation is simple and cheap. Therefore, we focused on the second synchronization method (see Fig. 17).

In packet based processing, the synchronization of the unit is done after the current packet is processed. If an error occurs at the beginning of the packet, the recovery process can be finished a long time before the end of the current packet. Without synchronization, the unit may still report an error and initiate a new recovery process. In order to solve this problem, we introduced the concept of the safety window.

The safety window is the minimal time interval between reconfigurations of the same unit which guarantees that there will be at least one synchronization pulse. The length of the safety window depends on the implemented system and the synchronization method.

Every unit has its own safety window. However, every safety window has to be implemented by its own SWU. It is possible to group several units so as to have one common SWU. In this case, the Safety Window length has to reflect the longest safety window. The length of the safety window determines the maximum frequency of the repair process. The trade off between implementation complexity and the frequency of the recovery process should be considered with design specifics in mind.

The implementation of a GPDRC contains only one SWU to limit resource utilization and thus increases system reliability. The SWU counter is activated when the shift register implementing the round robin algorithm shifts "1" to its last position. The SWU does not allow the next shift until the time limit is reached. This implementation allows it to make several reconfigurations of different units with a very simple implementation of SWU. It is important to note that every complexity in the GPDRC requires additional logic which may be affected by the SEU. Therefore, a simple algorithm is always preferred.

## 6. SEU simulation framework for testing fault tolerant systems based on FPGA

In this section, the principles of an SEU simulation scenario and experiments with an FT structure are described. The SEU simulation framework is the last step of the proposed methodology and can be used for FT architectures testing.

### 6.1. SEU simulation techniques

SEU simulation is the process of changing one bit of information in the configuration memory or in the memory of the FPGA design such as registers or BlockRAM. SEUs do not necessarily cause any errors since all parts of the configuration memory are not used in the design. As described in [32], only 10% of the configuration memory is used to define the design function on average. Unfortunately, it is not possible to make a prediction if a given bit is required for the design function or not, because the structure of the configuration memory is not usually documented.

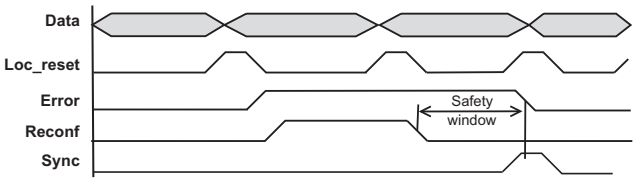


**Fig. 17.** Synchronization of PRMs after reconfiguration.

### 6.2. Requirements on testing platform and SEU generator

Every SEU simulator should meet a few criteria in order to be useful for the testing of the FTS. The proposed criteria was selected according to the authors experience with developing a fault tolerant methodology for FPGAs. The main requirements for SEU generator are:

- *Universality* – the SEU generator should be able to place an SEU at any place on the FPGA, not only to the configuration memory, but also to the circuity in which the function is implemented. The universality property is required for testing design level mitigation techniques, such as TMR architecture or a duplex system with checkers and a multiplexer.
- *Locality* – the SEU generator should be able to place the SEU into a pre-determined area of the FPGA and guarantee that other areas will remain unmodified. This property allows for a different level of testing to be used in different parts of FT architectures. Every reliable architecture has its weak points. For example, NMR and TMR architectures have the voter unit as a very weak point of the architecture. If this unit fails, then the entire architecture fails as well. However, if implemented correctly, TMR will mask any error in the function unit. The property of locality ensures that the SEU generator is able to do exhaustive testing of the function unit without attacking the voter.
- *Separateness* – the SEU generator should be separated and independent on the function implemented into the FPGA. The separateness property also means that the SEU generator should be able to operate on any FPGA design without the need to rebuild the design. There are several reasons for this property to be satisfied. First, if the design is rebuilt for testing purposes, then the unit under testing is different from the unit used in production and, therefore, the units can have different reactions to SEU injections. The second reason for the separateness is to guarantee that the generator will not damage itself. This reason is valid only for some parts of the designs. The PDR is the legitimate function of the FPGA and it can be also used in an FT system. The SEU generator has to be separated from the design in order to ensure that it will not interfere with the PDR done by the design itself.
- *Atomicity* – the SEU injection should be seen as an atomic process from the design point of view so as to ensure, for example, that the SEU in the register will not be replaced by a new value during its injection. The atomic property means that the SEU injection has to be performed faster than the period between two pulses of maximal clock frequency in the design or that the FPGA logic has to be shut down during the SEU injection. On the other hand, shutting down the logic can cause problems if the design is interacting with its environment, such as DRAM, Ethernet or other external function units. Since at least one frame has to be written into the configuration memory, it is virtually impossible to place the SEU in a one clock cycle at any reasonable clock speed. For these reasons, the implementation of the atomicity in the generator will present problems and some users might wish to disable it for specific situations.

### 6.3. Xilinx's solution

Xilinx offers its own solution called Soft Error Mitigation (SEM) [27]. However, this solution is designed primarily for the SEU

mitigation and the SEU generation is only an additional function. This presents a problem from a theoretical point of view since it can be argued that a unit that is tested should not be used for test generation. SEM is an IPcore generated only for Virtex6.

SEM has to be connected to the ICAP inside the FPGA in order to be able to mitigate the SEU and, therefore, the SEM does not meet the separateness property. Due to this fact, it is not possible to use ICAP without the functional change in the design itself.

The previous version of the SEM was called SEU Controller Macro and had only a limited support for locality. The user was able to address every bit in the bitstream by linear addressing, but it was not clear on how the linear address was transferred to the frame address to test a specified part of the FPGA only. The SEM supports a physical frame addressing which allows easier support for locality.

The atomicity issue is not solved by SEM. SEM operates inside the FPGA and, therefore, it is not possible to shut down the FPGA for the injection of an SEU. According to [33] the switching characteristics of Select MAP and, therefore, also ICAP for Virtex6 is 100 MHz. The injection of an SEU at this speed may take up to hundreds of clock cycles in the design.

The SEU generation may be used for a better understanding of the design behavior in the presence of an SEU and even for simple tests. However, SEM is not sufficient for the full evaluation of the fault tolerant designs. Another problem may arise when the design itself uses an ICAP interface.

Other SEU injection techniques for SRAM-based FPGAs exist, e.g. [34].

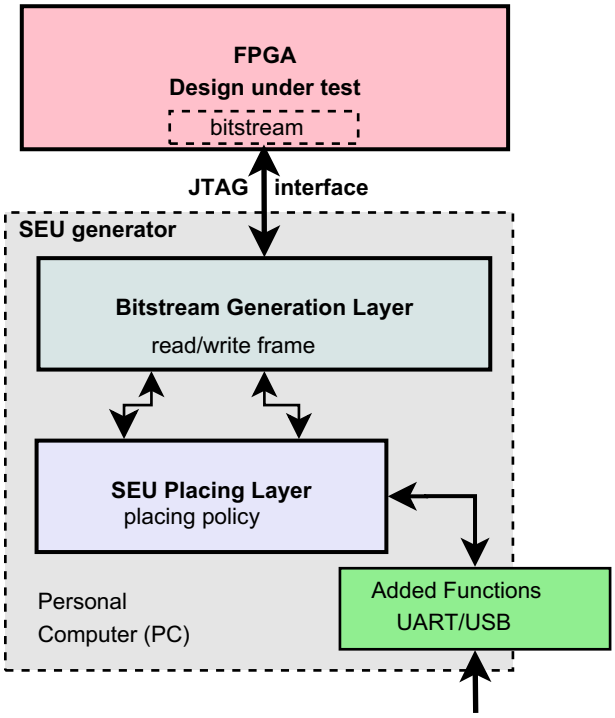


**Fig. 18.** External PC SEU generator structure.

### 6.4. Proposed solution: external SEU generator

This research proposes to implement SEU generation as a distinct tool working outside FPGA. The SEU generator should use a JTAG interface since this interface has the highest priority and, therefore, it can interrupt any other configuration interface.

The basic principle of an SEU generator is to combine readback and dynamic reconfiguration. The process of the SEU generation can be described in four steps.

1. *Frame selecting* – This step selects a frame where SEU will be generated. The selected frame has to be described by four variables (row index, column index and minor address together with top or bottom bit). The generator then constructs the frame address.
2. *Readback of the frame* – This step readbacks the whole frame without interrupting the computation in the FPGA.
3. *SEU generation* – This step converts one bit of the read data. The position of the changed data can be generated by a random function or by the given SEU generation policy.
4. *Write frame* – This step writes the changed frame back into the configuration memory of the FPGA. The write frame is currently implemented without interrupting FPGA.

### 6.5. SEU generator implementation

The proposed solution was implemented in the TCL language with the use of the ChipScope libraries. The correct function of the implementation was experimentally verified on the number of designs. No changes in the design were needed. The external PC SEU generator structure can be seen in Fig. 18.

The TCL implementation is divided into two basic layers. The first layer is responsible for communication with the FPGA. It is called a Bitstream Generation Layer. This layer uses the ChipScope libraries to send and read data through the JTAG interface. However, it is possible to change this layer to use any other JTAG drivers. Our decision to use ChipScope was based on the fact that ChipScope offers TCL functions for manipulation with JTAG. The Bitstream generation layer accepts the frame address and frame data from the SEU placing layer and generates bitstreams that will readback or write data for the given frame. The frame address is specified by four values corresponding to the parts of the Frame Address Register (FAR) in FPGA. The first variable is `TOP`. This variable can be 1 or 0 and specifies which half of the FPGA contains the frame. The second variable specifies the `ROW` of the FPGA in the given half (top or bottom). The last two variables describe the column. The first is called `COLUMN` and specifies which column of the device is used for counting by the columns visible in the PlanAhead software. Therefore, by setting these three values a user is able to address any given column of the CLB. The configuration of the CLB is contained in 36 frames, but this number is different for other types of columns, such as BlockRAMs. The fourth variable (`MINOR`) is used to specify which frame of the given CLBs should be addressed. It is important to keep in mind that the frame contains a configuration of 20 CLBs.

The SEU placing layer is responsible for the generation of the read and write frame requests according to the given SEU placing policy. The user will probably make its own modifications in this layer by adding a new functions for SEU placing. The typical functionality of the function in this layer is to compute the position of the SEU according to the implemented policy; readback the frame containing the affected part of the memory; change the bit at the computed position and write the frame back into the FPGA. Since the frame is the smallest addressable part of the configuration memory, the SEU has to be placed into the given position by the function of this layer. Currently, several placing policies are implemented:

- change any bit in one frame of bitstream or change several bits in the frame (multiple SEU),
- change the random bit in one frame of bitstream or change randomly several bits in the frame,
- fill one frame with a zero value or set several frames to zero values,

- fill one CLB with zero values.

The implemented solution was evaluated with four basic requirements presented in this paper. The solution is universal, since no requirements on the design in the FPGA exist. The locality is guaranteed by the addressing scheme which is the same as the addressing scheme used by the FPGA itself. Therefore, the SEU generator achieves the same locality as the FPGA reconfiguration process. Separateness property is saitisfied by the implementation in the TCL on the PC. Atomicity is another feature available in the implementation. The extension of atomicity is fairly simple by interrupting the FPGA. However, interrupting the whole FPGA for SEU generation can present synchronization problems between the FPGA and the other components on the board. Therefore, we decided to have a shutdown sequence as an additional feature that is not part of the SEU generator.

The last block in the diagram of the SEU generator is called Added Functions. This block contains functions for interfacing the SEU generator with the environment. These functions make it possible to drive SEU generation by external sources, such as UART or external program. Currently, only serial communication is implemented.

### 6.6. SEU simulation framework

The SEU simulation framework allows us to insert multiple SEUs in one run and simulate the occurrence of a higher number of SEUs. The architecture of the framework is shown in Fig. 19. The framework consists of two components.

The first component is the Unit Under Test (UUT) developed as a PRM or an architecture consisting of more PRMs. The other component is used for the evaluation – it contains another copy of UTT and a timing unit, a control unit, a comparator and an UART controller. The timing unit allows us to set the operation speed of the application related to the speed of SEU generation, the evaluation of the SEU effect and the transport of the result to a PC through an UART interface. The following information can be transported to the PC through UART:

- SEU position and the number of SEUs which changed the behavior of the unit under testing;

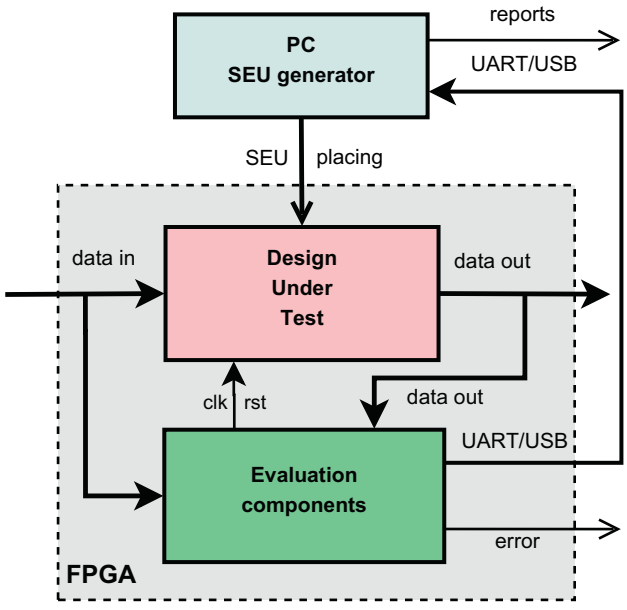


**Fig. 19.** Proposed SEU simulation framework for the testing of FT architectures.

- the number of incorrect values on outputs of sequential logic caused by SEUs together with the reaction of checkers to SEUs;
- the number of SEUs generated in the same time slice.

We intend to do extensive experiments with various FT architectures and use the implemented SEU simulation framework for these experiments. It will allow us to compare these approaches.

## 7. Experimental results

The above described FT scheme for an SRAM-based FPGA design and GPDRC (see Fig. 12) were implemented in VHDL language, and for the synthesis XILINX ISE 11.3 was used. ISE 11.3 supports the implementation of the PDR into FPGA Virtex5-XC5VSX50T on an ML506 development board. An ML506 board and SEU simulation framework were used to verify the proper function of different types of FT architectures and GPDRC. The following experiments were considered: (1) the comparison of FT architectures sizes (in slices) and the size of a GPDRC and a MicroBlaze solution in Virtex5; (2) the size of PRMs covering various functions (i.e. the function and its checker); (3) the size of a GPDRC which depends on the number of PRMs in Virtex5; (4) the experiments with SEU framework and FT architectures testing; and (5) the probability that the FT system will be in an operable state and will produce correct results for several SEU occurrences in the FPGA.

### 7.1. Digital circuit for testing purposes

To be able to verify the methodology, a digital circuit for testing purposes was developed. For the first experiments, a 7 segment display controller was implemented which contains several 3-bit and 8-bit counters, 8-bit decoders, multiplexors, shift-registers and other additional logic. The component counts repeatedly from 0 to 9999. It has an output to 4 LED displays on which the states of four counters are displayed, the correct operation of the component can be checked also visually. A fault can cause an incorrect counting or incorrect displaying of counter states, problems with clock signal, incorrect values in registers, etc. The components of test circuit (couters and decoders) were used in all experiments mentioned in all FT architectures.

### 7.2. Properties and experiments with FT architectures

In this part of experiments we compared properties of selected FT architectures for the SRAM-based FPGA design. The tested designs and FT architectures can be seen in Fig. 20.

The sizes of the FT architectures in FPGA are seen in Table 1. The meaning of the columns is as follows: column 1 – the type of FT architecture; column 2 – FT size in Virtex5 and the utilization of FPGA resources; column 3 – the number of PRMs and (PRMs with voter or multiplexor) in FT architecture; column 4 – the size of the static part and the utilization of FPGA resources; column 5 – the same for the dynamic part.

The size of PRMs for counters and decoders can be seen in Table 2. From the last two columns, it is clear that PRMs were not completely utilized. Based on these experiments it is reasonable to develop a methodology which will organize PRMs in such a way that will result in a more effective utilization of PRMs. Then, the number of PRMs is expected to be reduced together with the number of error signals entering the GPDRC.

In Table 3 the information about the area overhead of each FT architecture for 8-bit counter implementation can be seen. The size of FU (contain the 8-bit counter) is 5 slices. The smallest PRM block with one FU contains 20 CLBs and its size is 40 slices in Virtex5. The utilization of one PRMs with counter is about 16.5%.

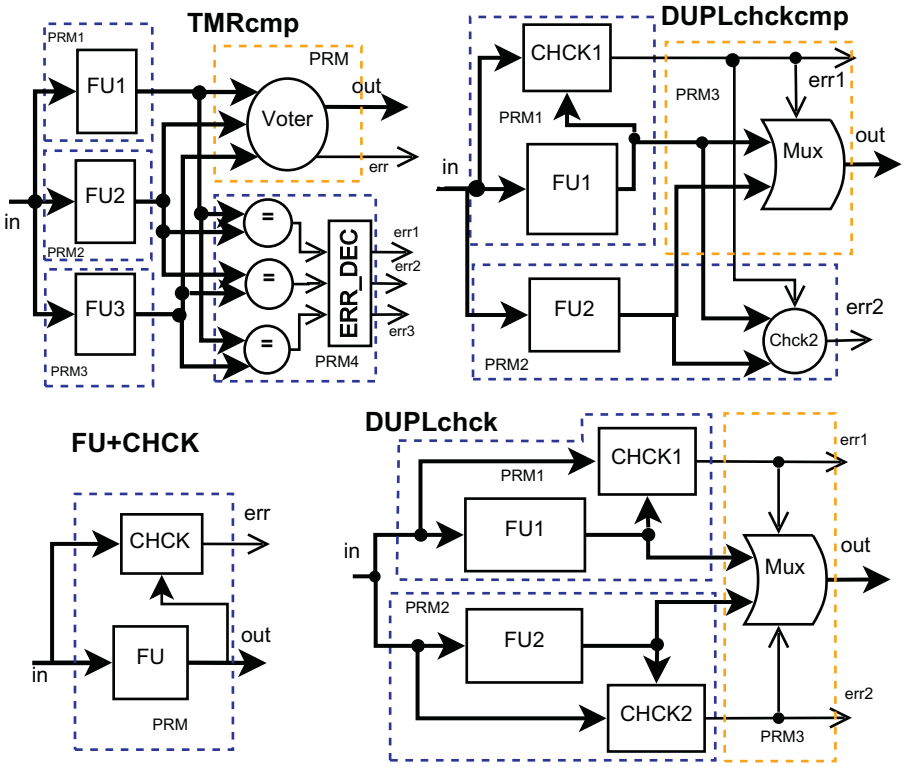


**Fig. 20.** Tested designs and FT architectures.

**Table 1**
Number of slices for FT architectures.

| ML506-Virtex5 XC5VSX50T counters + decoders | Size of FT.arch (slices) | # PRMs (–) | Size of static part (slices) | Size of dynamic part (slices) |
|---|---|---|---|---|
| TMRcmp | 145 (2%) | 4 (1) | 26 (18%) | 119 (82%) |
| DUPLchck | 86 (1%) | 2 (1) | 25 (29%) | 61 (71%) |
| DUPLchckcmp | 91 (1%) | 2 (1) | 28 (31%) | 63 (69%) |
| FU + CHCK | 44 (1%) | 1 (0) | 22 (42%) | 35 (78%) |

**Table 2**
Number of PRMs slices for FT architectures.

| ML506-Virtex5 XC5VSX50T counters + decoders | Size of cnt PRMs (slices) | Size of dec PRMs (slices) | Size of PRMs area (slices) | Usage of PRMs (%) |
|---|---|---|---|---|
| TMRcmp | 42 | 77 | 400 | 36.2 |
| DUPLchck | 23 | 38 | 240 | 35.8 |
| DUPLchckcmp | 22 | 41 | 240 | 37.9 |
| FU + CHCK | 11 | 20 | 80 | 55.0 |

**Table 3**
Overheads of FT architectures and PRMs.

| ML506-Virtex5 XC5VSX50T counter as FU | Size of FT arch (slices) | Non-PRM FT arch overhead | # PRMs (–) | Size of FT PRMs (slices) | PRM FT arch overhead |
|---|---|---|---|---|---|
| FU + CHCK | 12 | 2.2× | 1 | 40 | 0× |
| DUPLchckcmp | 18 | 3.3× | 3 | 120 | 3× |
| DUPLchck | 22 | 4.2× | 3 | 120 | 3× |
| TMRcmp | 28 | 5.5× | 5 | 200 | 5× |

The meaning of the columns in Table 3 is as follows: column 1 – the type of FT architecture; column 2 – size of FT architecture in Virtex5 for 8-bits counter; column 3 – the overhead of FT architecture without PRM implementation; column 4 – the number of PRMs in FT architecture; column 5 – the size of FT architecture implemented as PRMs; and column 6 – the overhead between FT architecture implemented as PRMs and one FU as PRM.

### 7.3. Experiments with GPDRC and MicroBlaze

During these experiments we were evaluating GPDRC basic parameters and features and tried to compare them with the solution based on MicroBlaze. All GPDRC components were described on VHDL, then they were simulated and synthesized into FPGA. The same was done for GPDR implemented as TMR architecture.

MicroBlaze was developed in ISE EDK environment. We gained the information about the size of GPDRC and MicroBlaze implementations and their comparison including the probability of faulty operation of both implementations.

The results of the GPDRC and MicroBlaze synthesis into a Virtex5 XC5VSX50T and the number of resources can be seen in Table 4. The meaning of the columns is as follows: column 1 – the name of the component in a GPDRC architecture; column 2 – the size of the component and the utilization of FPGA resources; column 3 (4) – the numbers of LUTs (FlipFlops); column 5 – the size of the FT GPDRC and overhead.

The probability that the GPDRC fails if the SEU occurs in the design is 5.33%, and for the MicroBlaze IP core it is 7.52%. In Virtex5-XC5VSX50T FPGA the total number of 204 PRMs can be created. In the biggest type of Virtex5 up to 1460 PRMs can be developed. But in practical applications, the number of PRMs appears to be an unreal number because certain space is needed for the implementation of GPDRC and interconnections needed for the implementation. Thus, the number of available PRMs is significantly reduced. Besides, in many applications the required size of PRMs is bigger than 36 frames which again causes a reduced number of PRMs to be available.

Fig. 21 shows how the number of slices increases with the number of PRMs up to 250. The size of the GPDRC increases almost linearly with the number of PRMs and the frequency of the GPDRC is around 171 MHz for 100 PRMs. The frequency of the GPDRC is sufficient because the frequency of ICAP interface is 100 MHz.

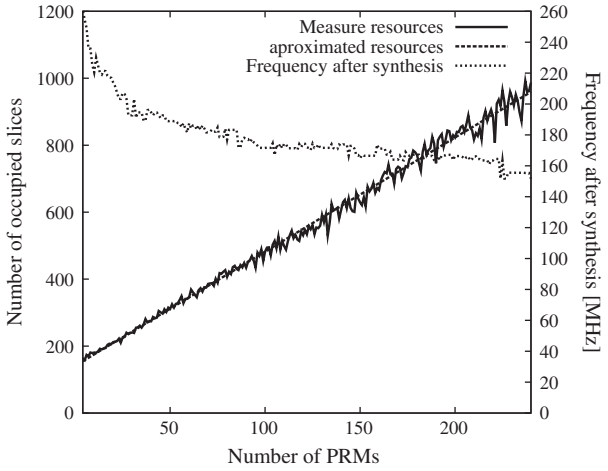


**Fig. 21.** Size and frequency of the GPDRC based on #PRMs.

### 7.4. Experiments with SEU simulation framework and FT architectures

For the following sets of experiments, the SEU simulation framework was used which was proposed above.

#### 7.4.1. Parameters of tested FT architectures

The basic parameters of tested designs and FT architectures which contain 8-bit counters and 8-bit decoders are available in Table 5. The meaning of the columns is as follows: column 1 – the type of architecture; column 2 – the design size in Virtex5; column 3 – the number of LUTs; 4 – the number of FlipFlop registers; column 4 – the number of frames; and the last column 5 – the size of the bitstream in bits for the placing of the SEU.

#### 7.4.2. SEU simulation framework settings

The parameters of the SEU framework were set in the following way: the generation frequency of one SEU into a tested design, including evaluation time (time measured from SEU placing to receiving a response of the system), was set to 100 μs. The communication between the FPGA UART controller and a PC was set to 115,200 Baud, 8-bit data, even parity and 2 stop bits.

#### 7.4.3. The first experiment – how many SEUs will affect the correct function of the architecture

The goal of the first experiment was to verify on how many SEUs will affect the correct function of the architecture under testing and to discover what will be the consequences for the architecture if SEUs are injected into functional units only. One-bit errors were injected into all positions of the bitstream. It means that faults were injected bit by bit into the particular part of bitstream related to PRM under test or architecture in FPGA. In one simulation step, one bit of the bitstream is modified and the reaction of PRM is checked. It was also tested on how many errors will be detected by checkers. The results are provided in Table 6.

The meaning of the columns is as follows: column 1 – the type of architecture; column 2 – the size of a bitstream in bits; column 3 – the number of incorrect data on the outputs of architecture; column 4 – the number of detected SEUs in FUs of the architecture; and finally column 5 – the number of detected SEUs in FUs by checkers.

#### 7.4.4. The second experiment – how many SEUs destroy the correct function of the architecture

During the second experiment it was verified on how many SEUs destroy the correct function of the architecture under testing including checkers and output logic. It was done in the same way as the above mentioned experiments (i.e. one bit errors were injected into all bitstream positions). It was evaluated to find out how many SEUs will have an impact on the correct function and how many errors cause complete non-operation of the system (see Table 7).

The meaning of the columns is as follows: column 1 – the type of the architecture; column 2 – the size of the bitstream in bits for

**Table 4**
Numbers of FPGA resources for the GPDRC.

| ML506-Virtex5 100 PRMs | Size (slices) | # LUTs (–) | # F/Fs (–) | TMR (slices) |
|---|---|---|---|---|
| Round robin unit | 71 (1.0%) | 101 | 202 | 288 (4×) |
| Error Encoder | 42 (0.7%) | 107 | 0 | 160 (3.8×) |
| Hard Error Unit | 74 (1.1%) | 152 | 202 | 237 (3.2×) |
| Safety Window Unit | 11 (0.2%) | 30 | 25 | 32 (2.9×) |
| ECC unit | 18 (0.3%) | 19 | 37 | 44 (2.4×) |
| Address unit | 21 (0.3%) | 51 | 21 | 51 (2.4×) |
| FSM | 26 (0.4%) | 45 | 60 | 60 (2.4×) |
| FIFO | 52 (0.8%) | 52 | 124 | 52 (0.0×) |
| FLASH control | 3 (0.1%) | 2 | 4 | 29 (9.6×) |
| GPDRC error input | 140 (2.0%) | 63 | 102 | 262 (2.0×) |
| **GPDRC total** | **458 (5.3%)** | **626** | **777** | **1215** (2.6×) |
| **MicroBlaze**IP core | **613 (7.5%)** | **1333** | **1328** | **1531** (2.5×) |

**Table 5**
Size of tested designs and FT architectures.

| Virtex5 XC5VSX50T counter8 + decoder8 | Size (slices) | # LUT (–) | # FF (–) | # Frame (–) | # Bits (–) |
|---|---|---|---|---|---|
| FU + CHCK | 12 | 41 | 12 | 36 | 47,232 |
| DUPLchckcmp | 18 | 64 | 16 | 72 | 94,464 |
| DUPLchck | 22 | 88 | 24 | 72 | 94,464 |
| TMRcmp | 28 | 102 | 32 | 144 | 188,928 |

**Table 6**
Number of detected SEUs in FUs of the architectures.

| XC5VSX50T CNT8 + DEC8 | Bitsream size (bits) | # Output data errs | # Detected SEUs in FUs | SEU detected by checker (%) |
|---|---|---|---|---|
| FU + CHCK | 47,232 | 1996 | 1996 | 100 |
| DUPLchckcmp | 94,464 | 2988 | 4040 | 99 |
| DUPLchck | 94,464 | 3064 | 4423 | 100 |
| TMRcmp | 188,928 | 1035 | 8633 | 100 |

**Table 7**
Number of detected SEUs in the FT architectures.

| Virtex5 XC5VSX50T counter8 + decoder8 | Bitsream size (bits) | # Detected SEU | # Total errors |
|---|---|---|---|
| FU + CHCK | 47,232 | 1996 | 1348 |
| DUPLchckcmp | 94,464 | 5857 | 1262 |
| DUPLchck | 94,464 | 6850 | 1606 |
| TMRcmp | 188,928 | 10,456 | 1698 |

the placing of SEU; column 3 – the number of detected SEUs in architecture; 4 – the number of errors, where all states of sequential logic were incorrect.

Finally, it was verified that all checkers and other checking units were able to detect the SEUs injected into the architecture. The digital component continued to cover its function during the existence of the SEU and also during the reconfiguration process where it provided correct outputs. It was proven that a unit equipped with checkers is fault tolerant if correctly designed. However, routing tools route signals through the reconfiguration region which allows for the static part of the design to be damaged by an SEU occurence in the dynamic part.

*7.4.5. The third experiment – experiments with sequential component*

In these experiments we tried to check how often the state of an 8-bit counter is modified during SEUs injection. After SEU is placed into the bitstream (one simulation step during which the counter counts between 0 and 255), we measured how many faulty values were identified on the counter output (when compared with a reference counter). During the experiments all the bits in the bitstream were modified.

The results for one FU in DUPLchckcmp architecture are seen in the histogram in Fig. 22. The histogram demonstrates how many different values appeared on FU output (caused by an SEU injection) in an 8-bit counter and on how often they occurred.

In the histogram, *X* axis represents the values on counter outputs which were different from values on the output of reference counter while SEUs were injected to all bitstream positions. *Y* axis reflects the frequency of fault occurrences, i. e. the number of the fault occurrences during the test.

In Fig. 23, the same situation is demonstrated for the DUPLchck FT architecture when SEUs were injected to all positions of bitstream of the whole FT architecture. *X* and *Y* axis have the same meaning as they have in histogram 22.

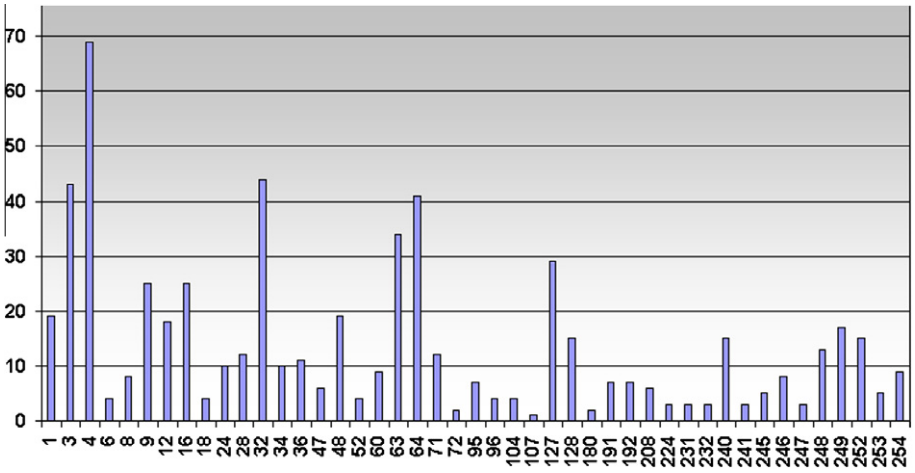


**Fig. 22.** Histogram of error states of an FU.

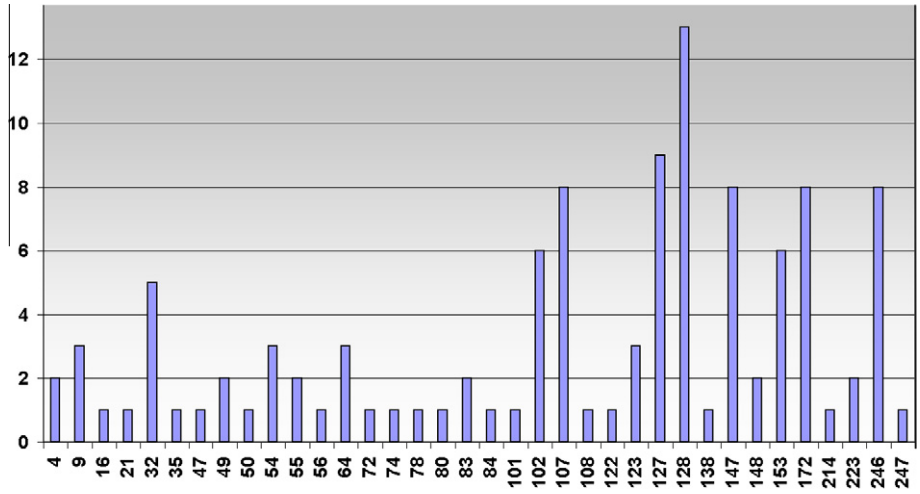


**Fig. 23.** Histogram of error states of the FT architecture.

### 7.5. Evaluation of correct working of FT structure

Finally, during our experiments we demonstrated how the availability of the system implemented with the use of the methodology presented in this paper, has increased.

Graphs in Figs. 24 and 25 reflect how to increase probability that the FT system will work correctly with the different number of PRMs for different number of SEUs occurrences at the same time in FPGA. The first graph shows the results for FT system based on TMR architecture (TMRcmp) and the second graph shows the results for duplex architectures (DUPLchck and DUPLchckcmp).

## 8. The comparison of presented methodology with other techniques

In comparison with other methodologies [19,16] which try to implement fault tolerant systems into the FPGA, our methodology allows us to detect and locate all transient faults caused by SEUs in the application, its registers and in the bitstream. Besides, it allows one to localize permanent faults in the interconnection net which is not possible in the approaches based on bitstream scrubbing with TMR [26].

Our methodology allows to develop fault tolerant system because it is able to mask errors caused by multiple faults and the system is able to produce correct results. This is not guaranteed in bitstream scrubbing methodologies where an incorrect result can exist on the output until the bitstream is overwritten by a correct sequence. Bitstream based methodologies in combination with TMR suppresses the incorrect results on the FPGA output although the fault cannot be localized and a permanent fault in the FPGA cannot be identified. As far as we are informed, no such methodology exists which has all of the features mentioned above.

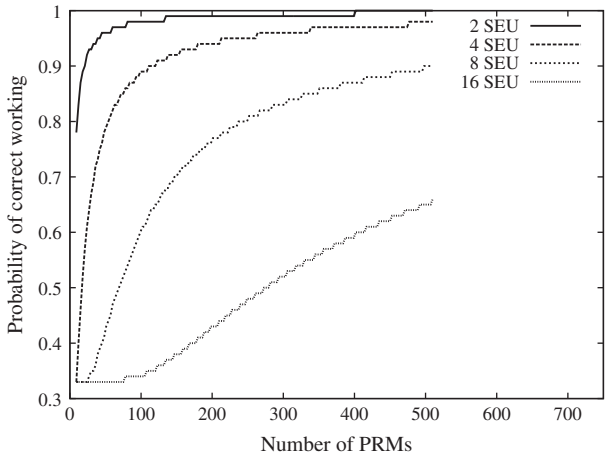


**Fig. 24.** Probability of a correct working of design on a number of PRMs based on TMR architecture.

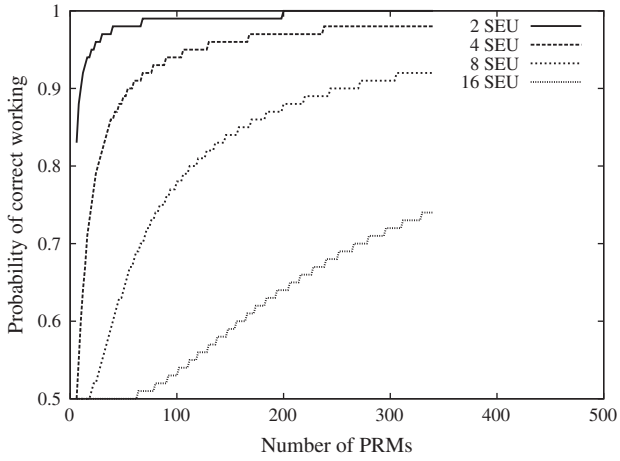


**Fig. 25.** Probability of a correct working of design on a number of PRMs based on duplex architectures.

Many techniques use PowerPC or MicroBlaze processors inside FPGA to control the reconfiguration and represent a crucial part of the system and they are unrecoverable after they fail [15,23]. Besides, their implementation requires more space on the chip than our simple reconfiguration controller and they must be programmed.

## 9. Conclusions and suggestions for future research

In this paper, the fault tolerant methodology for the SRAM-based FPGA via the PDR with the GPDRC inside the FPGA was presented. The main role of the GPDRC in an FT system is seen in the identification of faulty PRM and the initiation of the reconfiguration process of the faulty module in FT architectures. The main structure and basic parameters of the GPDRC were described

together with the problems of PRMs synchronization after one of them is reconfigured.

The results of our research can be summarized as follows: (1) three different FT architectures were developed and implemented into Virtex 5; (2) internal GPDRC was developed to avoid the necessity of controlling the reconfiguration process by ICAP and achieve higher speeds of reconfiguration. The results of synthesis demonstrate that the GPDRC has lower overhead than the controller implemented as MicroBlaze; (3) the architecture of the GPDRC was described; (4) the basic parameters of the FT structure for the SRAM-based FPGA applying PDR were evaluated; (5) the SEU simulation framework for testing fault tolerant system designs implemented into FPGA was used. The experimental results demonstrate that all checkers of FT architectures and other checking units are able to detect SEUs injected into a design.

### 9.1. Future research

In the future, the methodology will be verified on the latest Virtex 6 installed to an ML605 development board. We shall concentrate on a more effective implementation of PRMs synchronization in FT architectures and the relationship between GPDRC and its dependability parameters. We shall experiment with larger designs where the following aspects will be considered: time needed for reconfiguration, effective communication between the GPDRC and ICAP interface (speed optimization) and the relationship between the FT structure and its dependability parameters (availability, maintainability).

We intend to experiment with MicroBlaze as another way how to control reconfiguration in our FT architecture as well. Then, we shall be able to compare the features and behavior of three reconfiguration tools: GPDRC, GPDRC implemented as TMR and MicroBlaze.

## Acknowledgements

This work was supported by the following Projects: National COST LD12036-"Methodologies for Fault Tolerant Systems Design Development, Implementation and Verification"; MSMT RECOMP-"National Support for Project Reduced Certification Costs Using Trusted Multi-core Platforms"; research Project No. MSM 0021630528-"Security-Oriented Research in Information Technology"; GACR No.102/09/H042-"Mathematical and Engineering Approaches to Developing Reliable and Secure Concurrent and Distributed Computer Systems", by the IT4Innovations Centre of Excellence project (ED1.1.00/02.0070) and Grant "FIT-S-11-1".

## References

[1] J.A. Cheatham, J.M. Emmert, S. Baumgart, A survey of fault tolerant methodologies for fpgas, ACM Trans. Des. Autom. Electron. Syst. 11 (2) (2006) 501–533.
[2] M. Straka, Z. Kotasek, J. Winter, Digital systems architectures based on on-line checkers, in: 11th EUROMICRO Conference on Digital System Design DSD 2008, IEEE Computer Society, 2008, pp. 81–87.
[3] S.-Y. Yu, E.J. McCluskey, On-line testing and recovery in tmr systems for real-time applications, in: ITC '01: Proceedings of the 2001 IEEE International Test Conference, IEEE Computer Society, Washington, DC, USA, 2001, pp. 240–250.
[4] C. Galke, M. Grabow, H.T. Vierhaus, Perspectives of combining on-line and off-line test technology for dependable systems on a chip, in: IOLTS '03: Proceedings of the 13th IEEE International On-Line Testing Symposium, Los Alamitos, CA, USA, 2003, pp. 183–189.
[5] B. Osterloh, H. Michalik, S.A. Habinc, B. Fiethe, Dynamic partial reconfiguration in space applications, NASA/ESA Conf. Adapt. Hardw. Syst. 0 (2009) 336–343.
[6] M. Straka, Z. Kotasek, High availability fault tolerant architectures implemented into fpgas, in: 12th EUROMICRO Conference on Digital System Design DSD 2009, IEEE Computer Society, 2009, pp. 108–116.
[7] R. Oliveira, A. Jagirdar, 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, IEEE Computer Society, Washington, DC, USA, 2007, pp. 905–910.
[8] C. Bolchini, D. Quarta, M.D. Santambrogio, Seu mitigation for sram-based fpgas through dynamic partial reconfiguration, in: GLSVLSI '07: Proceedings of the 17th ACM Great Lakes symposium on VLSI, ACM, New York, NY, USA, 2007, pp. 55–60.
[9] XILINX, Ug208: Early Access Partial Reconfiguration User Guide. <http://www.xilinx.com>.
[10] P. Kubalik, R. Dobias, H. Kubatova, Dependable design for FBGA based on duplex system and reconfiguration, in: DSD '06: Proceedings of the 9th EUROMICRO Conference on Digital System Design, Dubrovnik, Croatia, 2006, pp. 139–145.
[11] O. Heron, T. Arnaout, H.-J. Wunderlich, On the reliability evaluation of sram-based fpga designs, in: FPL '05: International Conference on Field Programmable Logic and Applications, ACM, Tampere, Finland, 2005, pp. 403–408.
[12] M. Straka, J. Kastil, Z. Kotasek, Modern fault tolerant architectures based on partial dynamic reconfiguration in fpgas, in: 13th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems, IEEE Computer Society, New York, NY, USA, 2010, pp. 336–341.
[13] M. Niknahad, O. Sander, J. Becker, A study on fine granular fault tolerance methodologies for fpgas, in: Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on, 2011, pp. 1–5.
[14] M. Pereira, L. Braun, M. Hubner, J. Becker, L. Carro, Run-time resource instantiation for fault tolerance in fpgas, in: Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on, 2011, pp. 88–95.
[15] C. Pilotto, J.R. Azambuja, F.L. Kastensmidt, Synchronizing triple modular redundant designs in dynamic partial reconfiguration applications, in: SBCCI '08: Proceedings of the 21st Annual Symposium on Integrated Circuits and System Design, ACM, New York, NY, USA, 2008, pp. 199–204.
[16] S. D'Angelo, G.R. Sechi, C. Metra, Transient and permanent fault diagnosis for fpga-based tmr systems, in: Proceedings of the 14th International Symposium on Defect and Fault-Tolerance in VLSI Systems, IEEE Computer Society, Washington, DC, USA, 1999, pp. 330–338.
[17] F. Lima, L. Carro, R. Reis, Designing fault tolerant systems into sram-based fpgas, in: DAC '03: Proceedings of the 40th Annual Design Automation Conference, ACM, New York, NY, USA, 2003, pp. 650–655.
[18] F. Lahrach, A. Doumar, E. Chatelet, Fault tolerance of multiple logic faults in sram-based fpga systems, in: Digital System Design (DSD), 2011 14th Euromicro Conference on, 2011, pp. 231–238.
[19] F.L. Kastensmidt, G. Neuberger, L. Carro, R. Reis, Designing and testing fault-tolerant techniques for sram-based fpgas, in: CF '04: Proceedings of the 1st Conference on Computing frontiers, ACM, New York, NY, USA, 2004, pp. 419–432.
[20] L. Sterpone, M. Aguirre, J. Tombs, 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, ACM, New York, NY, USA, 2008, pp. 336–341.
[21] M.G. Gericota, L.F. Lemos, G.R. Alves, J.M. Ferreira, On-line self-healing of circuits implemented on reconfigurable fpgas, in: IOLTS '07: Proceedings of the 13th IEEE International On-Line Testing Symposium, IEEE Computer Society, Washington, DC, USA, 2007, pp. 217–222.
[22] C. Bolchini, A. Miele, 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, IEEE Computer Society, Washington, DC, USA, 2007, pp. 87–95.
[23] X. Iturbe, M. Azkarate, I. Martinez, J. Perez, A. Astarloa, A novel seu, MBU and SHE handling strategy for Xilinx Virtex-4 FPGAs, in: International Conference on Field Programmable Logic and Applications, 2009. FPL 2009, IEEE Computer Society, Washington, DC, USA, 2009, pp. 569–573.
[24] U. Sharma, Fault tolerant techniques for reconfigurable platforms, in: A2CWiC '10: Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India, ACM, New York, NY, USA, 2010, pp. 1–4.
[25] C.T. Rathgeb, G.D. Peterson, Secure processing using dynamic partial reconfiguration, in: CSIIRW '09: Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research, ACM, New York, NY, USA, 2009, pp. 1–4.
[26] J. Heiner, B. Sellers, M. Wirthlin, J. Kalb, Fpga partial reconfiguration via configuration scrubbing, in: FPL '09: International Conference on Field Programmable Logic and Applications, ACM, New York, NY, USA, 2009, pp. 99–104.
[27] XILINX, User Guide: Logicoretm IP Soft Error Mitigation Controller v1.1.
[28] [] L. Sterpone, M. Violante, A new partial reconfiguration-based fault-injection system to evaluate SEU effects in SRAM-based FBGAs, IEEE Trans. Nucl. Sci. 54 (2007) 965–970.
[29] K. Morin-Allory, D. Borrione, Proven correct monitors from psl specifications, in: DATE '06: Proceedings of the Conference on Design, Automation and Test in Europe, European Design and Automation Association, 3001 Leuven, Belgium, Belgium, 2006, pp. 1246–1251.
[30] M. Boule, Z. Zilic, Automata-based assertion-checker synthesis of psl properties, ACM Trans. Des. Autom. Electron. Syst. 13 (1) (2008) 1–21.
[31] J.-B. Note, E. Rannaud, From the bitstream to the netlist, in: FPGA '08: Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays, ACM, New York, NY, USA, 2008, pp. 264–268.
[32] XILINX, Xapp864: Seu Strategies for Virtex-5 Devices. <http://www.xilinx.com>.
[33] XILINX, Virtex-6 FPGA Data Sheet: DC and Switching Characteristics.
[34] G. Foucard, P. Peronnard, R. Velazco, Reliability limits of tmr implemented in a sram-based fpga: heavy ion measures vs. fault injection predictions, in: Test Workshop (LATW), 2010 11th Latin American, 2010, pp. 1–5.

**Martin Straka** was born in 1981. In 2006 he graduated (MSc) at the Department of Computers Systems of the Faculty of Information Technology, Brno University of Technology. In 2006 he started his PhD studies at the Department of Computers Systems. His scientific research is focused on fault tolerant systems design and on-line testing of FPGA based systems. He is author or co-author of more than 25 scientific papers.

**Jan Kastil** was born in 1983. He received his Bc and MSc degrees in 2006 respectively 2008 from Brno University of Technology. Currently he is working towards his PhD at the same university. His research interests include FPGAs, the application of the partial dynamic reconfiguration and the acceleration of the high speed networks. He is author or co-author of more than 10 scientific papers.

**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 Department of Computer Science of the 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 and the head of the DCSY (since 2005). 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 member (since 2003).

**Lukas Miculka** was born in 1985. In 2010 he graduated (MSc) at the Department of Computers Systems of the Faculty of Information Technology, Brno University of Technology. In 2010 he started his PhD studies at the Department of Computers Systems. His scientific research is focused on fault tolerant systems design and partial dynamic reconfiguration based on FPGA systems.