Automatically-Designed Fault-Tolerant Systems: Failed Partitions Recovery

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

***Abstract*—This paper presents and describes our design automation toolkit for automatic synthesis of fault tolerant systems from unhardened systems. The toolkit is composed of various parts and tools and its aim is to design its internal algorithms in such way to be reusable among different HW description languages. In this paper, VHDL description is used to present the possibilities of the toolkit. The experimental part of the paper presents automatic synthesis of a benchmark system into a limited chip area. The optimization goal was to maximize the median time to failure (a.k.a. t50) parameter. The main part of the experimental activities comprises incorporation of a partial dynamic reconfiguration controller into the system design to recover the selected component of the system. Two systems utilizing recovery with the usage of the FPGA dynamic reconfiguration technique show promising results in terms of reliability. The recovered system, in which the controller is apart of the FPGA (e.g. in a different radiation-hardened chip), achieves by 70% better t50 parameter, compared to the system without recovery.**

***Keywords*—Fault-Tolerant System Design, Electronic Design Automation, Redundancy Insertion, Redundancy Allocation, Multiple-choice Knapsack Problem, FPGA, VHDL, t50.**

I. Introduction

Special types of electronic systems are required to withstand certain harsh environments, such as environments with increased radiation. Such systems must perform their function without interruption of the data processing and without altering their behavior. One possibility to treat this problematic is to incorporate the so-called *Fault Avoidance* (FA) [1]. This treatment lies in the usage of reliable components to produce reliable systems that comply with the specifications. The selection of components for the designer is limited by the specifications to such components that are produced for the given environment – e.g. increased radiation. Such components are thoroughly tested and usually do not incorporate the newest manufacturing process nor a newest architecture. The other treatment includes the so-called methods of *Fault Tolerance* (FT) [2]. FT accepts the fundamental fact that any component may fail and aims to solve high reliability on the architectural level. By incorporating the so-called fault-masking techniques, the system can appear to be fully functional, while one or more of its components are in a failed state.

Highly-reliable systems include those, that control potentially dangerous processes or can cause a loss of tangible or intangible assets. Other systems requiring to maintain high level of reliability include space probes and other space equipment. Such devices are nearly impossible to repair. Failure of the electronic control system of such device poses an incredible-high and very unnecessary risk of failure of the complete mission. Performance parameters of commercially-available components are, however, usually significantly better, as their design does not involve time-consuming testing. For example, the *National Aeronautics and Space Administration* (NASA) Perseverance rover [3], which landed on February, 18th 2021 on Mars, carries on board the Ingenuity helicopter [4], the control systems of which are designed from common commercially-produced components. One of the purposes is to test the service life of such components in a such harsh environment, which the Mars atmosphere surely is.

Certain reliable applications also utilize *Field Programmable Gate Arrays* (FPGAs), for their performance and ability to reconfigure, i.e. reprogram their functionality. A common FPGA holds the configuration bitstream in its SRAM memory. The SRAM is subject to the so-called *Single-Event Upsets* (SEUs), which have the potential to flip a configuration bit, thus effectively changing the implementation in the FPGA. One of the uses of FPGAs for the class of space probe applications is to perform scientific data processing on board of remotely-controlled rover. For example, the already-mentioned Perseverance rover uses Xilinx FPGAs for image processing and machine-learning algorithms for searching for signs of life on Mars [5]. Nearly 18-times faster data processing is achieved with the usage of FPGAs, compared to the previous approach.

The usual approach to fault masking is the so-called *Triple Modular Redundancy* (TMR), which triplicates the design and adds a voter to select the representative result. The TMR can be applied in two general ways: 1) Triplication of the whole design, which is called the *Coarse-Grained TMR* (CGTMR). With regard to maximal efficiency of chip area usage and maximal level of FT, the CGTMR is not ideal, as it results in equivalent amount of redundancy throughout the whole design. In the 2) approach, the design is partitioned to smaller units, and thereafter, such smaller units are triplicated. Such approach is called the *Fine-Grained TMR* (FGTMR). This allows to target the redundancy towards certain partitions of the system, based on their criticality.

The fault masking approach itself is not sufficient. The *N-Modular Redundancy* (N-MR) systems tend to decrease the *Mean Time To Failure* (MTTF) for longer mission times [1], as the failed components accumulate. For this reason, the fault-masking approaches are usually combined with reparation mechanisms. Specifically for FPGAs, such mechanisms include the so-called *Partial Dynamic Reconfiguration* (PDR). Although originally meant to change the FPGA configuration

978-1-6654-4503-0/21/$31.00 ©2021 IEEE

at a run time, the PDR can also be used to restore the configuration of the FPGA, which could have been altered as a result of the SEU. The PDR can be initiated from inside of the FPGA itself, making a possibility to create a reparation unit directly on the same FPGA as the electronic system. Of course, the bitstream restore can be initiated from the outside of the FPGA as well, for example by a radiation-hardened microcontroller.

With the increasing number of partitions, and with numerous FT architectures, the number of possible combinations rises drastically, thus, making the so-called redundancy allocation problem a great challenge. This creates a pressure to automate the complete process of FT system design. The objective of our research is to design such design automation methods. Further, our research focuses on the FGTMR and N-MR techniques for FPGA, partially oriented at the data processing systems. In this paper, the complete overview of our existing FT design automation toolkit, which was extended to provide possibility to incorporate mechanisms of reparation using PDR, is presented alongside with the case study on automatically-hardened data-processing oriented benchmark circuit on a real HW FPGA. The research that was previously presented in [6], is extended in this paper by the identification of weakest components of the system and the incorporation of recovery for such components.

II. Related Work and the Context

This work deals with various research themes, nonetheless, the main themes include 1) Redundancy Insertion, 2) Reliability Allocation and 3) Fault Tolerance Testing.

One commercially available redundancy insertion tool, the so-called Xilinx TMRTool [7], modifies the synthesized design during the design process. Another possibility to include redundancy, this time at the source-code level, is the TMRG [8]. The TMRG works with systems described in the Verilog language. It is focused towards creation of the TMR structure exclusively, as well as the TMRTool. Different option is to modify the synthesis tool itself, to produce reliable designs. For example, the TLegUp [9] is based on the modified version of the *High-Level Synthesis* (HLS) tool LegUP [10]. It generates TMR designs directly from the description in the C language.

Reliability Allocation methods exist throughout the literature as well. For example, in paper [11], the *Improved Surrogate Constraint* (ISC) method was examined, targeting the computational speed of the design method. In [12], the penalty guided artificial bee colony algorithm was presented. The use of particle swarm optimization method is proposed in [13], while the variable neighborhood search meta-heuristic method was presented in the paper [14].

Waiting for faults to appear naturally is not feasible during the testing procedure. Therefore, special techniques are used to increase the fault occurrence in order to examine the design during the presence of faults. One approach, utilizing the RapidSmith library [15], is presented in [16] and later demonstrated in [17]. The paper [18] shows a method of observing and modifying signals in the design through the *Joint Test Action Group* (JTAG) interface. The approach in [19] supports various fault models. Some extra gates must be added to the design before testing. Simulation-based evaluation is also present in literature [20], [21]. In [22], fault modeling in combination with design simulation is used. In paper [23] an approach is presented, in which the fault injection is fully controlled by a component on the FPGA itself, significantly improving the testing speed. In paper [24], evaluation platform designed in our research group is presented. The platform is executed on a PC, which also captures and evaluates data obtained from an FPGA. Nonetheless, the complete platform is more suitable for the final verification of reliability parameters. In this research, however, massively accelerated evaluations are needed to complete the design task in a reasonable time.

In our research, we target a comprehensive approach to automate every part of the FT design process. The main goal is to design most of the components to be reusable. For example, if the description code manipulation is isolated from the rest of the system, it must be possible to replace the code manipulation to instantly add support for a new description language. Nonetheless, it is beneficial if the language supports direct synthesis to an implementation, for example for an FPGA, such as VHDL or Verilog. Our goal is to research new methods of FT design automation, implement them and examine their aspects in practice. So far, we have implemented code manipulation to include FT on the behavioral level for the C++ language (in combination with HLS tools) and on the structural-level for the VHDL (in combination with traditional VHDL synthesis tools). The selection of FT methods based on combinatorial optimization problems and massively-accelerated evaluation of FT properties were also developed. In this paper, we extend our method with ability to incorporate system recovery with the usage of PDR and practically evaluate this approach on two versions of the system: 1) PDR controller on the FPGA itself, and 2) PDR controller outside the FPGA (e.g. in a radiation-hardened external chip).

III. Fault-Tolerant System Design Automation

Today's chip integration allows to implement large systems. These become increasingly complex. The difficulty to incorporate FT into such complex systems also grows, as it shows to be beneficial to split the system into smaller partitions and select the proper FT method for each partition exclusively. Such selection is complex and very time-consuming process, increasing the interest in automated design of FT systems.

Our design flow is based on the traditional process (i.e. the originally manual design). The input of both these approaches is a system description, which we call *unhardened* or also *original*. The other input of the development process is the specification of desired reliability parameters and the method of their measurement. The traditional process involved iterative development of the system by addressing its weak points and modifying them to remove their impact in the case of a failure. A designer does these modifications based on previous experiences and judgment. On the other side, the automated flow performs the so-called state space exploration of all the possible configurations. This might involve heuristic approaches, which reduce the number of states needed to explore. The output of the development is a system, which incorporates the needed FT techniques and complies with the reliability specifications. Also, the output system must be functionally equivalent. The original and automated flows are displayed side-by-side in Figure 1.

A. Description-code Modifications

In our design flow, the source code modifications are contained inside the so-called *Helpers*. These allow to incor-

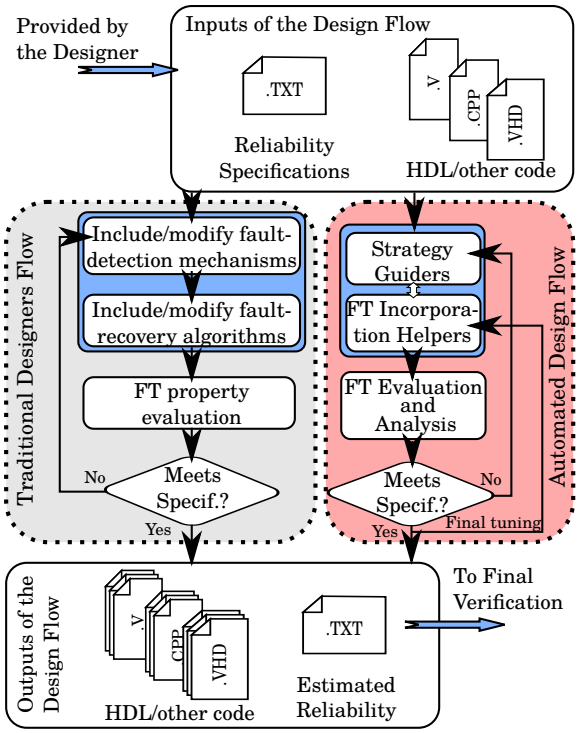


Figure 1: FT System Design with the Original and Automated Design Flows.

porate various FT methods into a specific description code language. The FT is incorporated throughout the modification of the system description, thereby isolating the specifics of the language in helpers. In our current research, we partition the system based on every entity instantiation in the top-level design. The project source code must be prepared with this fact in mind. The top-level VHDL source should contain number of components, that will be considered partitions in the design flow. Each helper incorporates one or many FT techniques into a specified partition of the system. Obviously, the helpers must be re-designed for a different language. However, the reliability allocation algorithms can be fully re-used. For example, in our previous research, we utilized our so-called *Redundancy Data Type* (RDT) helpers [25] to incorporate time and spatial redundancies into C++ algorithms and synthesize them using HLS. In our more recent work, we designed VHDL helpers and utilized them in a case study [6].

In the following case study, we use the VHDL helpers as well. The source description code must be prepared to instruct proper modifications of the code. This includes a special prefix and postfix source code comments before and after an entity instantiation. These store the *guiding* information for the helpers. These include the indication of the beginning (or ending) of the source code, type of FT method (e.g. TMR, 5-MR, duplex, etc.). Currently, we focus on entity instantiations exclusively. The proper setup of such *guiding* comments is the responsibility of the so-called *Guider* (which will be described further in the text). The products of helpers are generally compatible with existing synthesis tools, as soon as the helpers modification procedures comply with the synthesizable language. In certain scenarios, the inserted logic must also comply with the target technology (e.g. a PDR controller must keep communication compatibility with the PDR interface).

The VHDL helpers modify the original source code files during two basic steps: 1) In-place modifications, and 2) Out-of-place modifications. At first, for the in-place modification, an original source file is loaded and the so-called code-block token list is extracted. Blocks of code in the token list are classified (e.g. *out-of-interest* block, *apply modification* block, *already modified* block, etc.). These are later merged into lists of self-contained objects. One object thus includes its guider comments (i.e. the prefix and postfix *guiding* comments) as well as the body of its instantiation. These objects are already prepared to apply modifications to, which includes mainly changing of the VHDL entity name to instantiate a newly created FT entity. This new entity is prepared during the out-of-place modification. An architectural template is copied onto its place and all the important data is filled in the template. Such data include the name of the original component, the signal names, their bit widths, etc. These are collected from the project source files. The VHDL helper execution then stops by re-assembling the final code of the original file. The flow is represented graphically in Figure 2. A simplified example of the VHDL code is also shown in the figure.

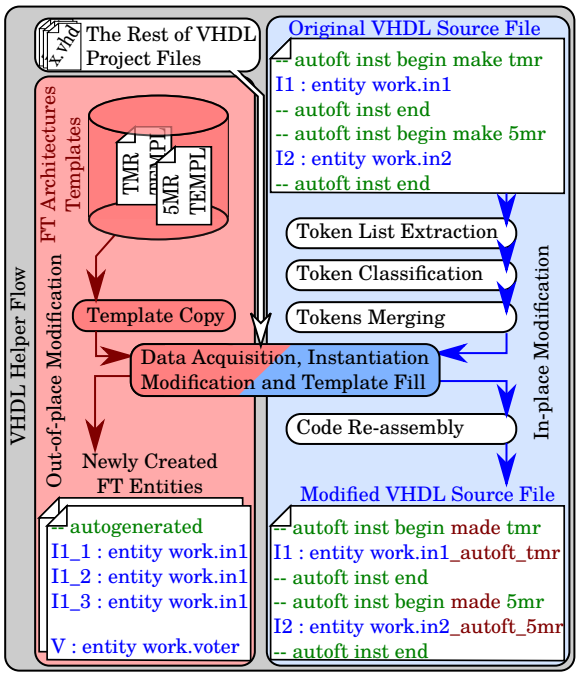


Figure 2: VHDL Helpers Code Modification Flow, Including a Simplified Code Example of Usage.

B. Guidance for the Modifications

In our research the helpers must be properly instructed using the *guiding* comments. Their content is provided by the so-called *Guider*. The Guider is an essential component of the FT design automation toolkit. It selects the proper FT methods based on their effect. Such selection must be performed for each component (i.e. partition of the system).

In our research we convert the problem of FT method selection to the so-called *Knapsack Problem* (KP). The KP is a well-known combinatorial problem, which aims to select the items from a given set of items. These are put into a hypothetical knapsack. Such knapsack has a limited capacity. The selected items must represent the most valuable selection.

A specific modification of this KP is the so-called *Multiple-Choice Knapsack Problem* (MCKP). In the MCKP, the set of all items is classified and exactly one item must be selected from each class. This is very similar to our FT selection problem and it is straightforwardly convertible to our problem: the items in sets are available variants for each partition (generated with the usage of helpers) and the knapsack capacity represents the given budget (e.g. FPGA area, power consumption, etc.). The item value is represented by the benefit of a certain component (e.g. lowering the number of sensitive bits of the bitstream, or increasing the *Time To Failure* (TTF) parameter). With this conversion, the FT methods can be selected automatically using the MCKP solver (i.e. a SW that solves an instance of the MCKP problem).

There are two possibilities how to use the MCKP solver as a guider: 1) the MCKP solver can call the helpers, design synthesis and demand the test and evaluation of the synthesized design. This is useful in cases, when we do not want to evaluate each combination of FT technique and partition in advance, as the method can start to compose systems and evaluate them instantly. The main disadvantage of such approach is the number of unsatisfactory-reliable systems that are unnecessarily evaluated and the quantity of which is relatively large, significantly prolonging the time to obtain the finished design. 2) The other possibility is to evaluate each partition separately and in advance and then use such data to operate the MCKP solver. The disadvantage lies in the necessity of such preliminary evaluation of each partition. Also, the resulting parameters of composed systems are slightly inaccurate, as these are computed inside the MCKP solver. One advantage is, that the time-consuming evaluation of unsatisfactory compositions is eliminated, as their parameters are estimated in SW during the MCKP solver execution. For example, if the median time to failure, usually called the t50, is monitored in the constraint specifications, the resulting t50 of systems with serially-dependent partitions can be approximated with equations for the MTTF. The approximation we use in the following case study is thus based on Equation 1 to approximate the λ (i.e. the failure rate). After that, Equation 2 can be used to compute the overall failure rate of the system. Subsequently, the t50 is approximated using Equation 1 again.

$$MTTF = \frac{1}{\lambda} \tag{1}$$

$$\lambda_{sys} = \sum_{\forall c \in C} \lambda_c \tag{2}$$

We use this second approach, as it eliminates the unsatisfactory evaluations. Nonetheless, in other cases, the method 1) is still usable for smaller system, despite the automated design then takes longer.

C. System Recovery Mechanisms

The solver is also modified to provide the guidance on which component (i.e. partition) is a candidate to be hardened with recovery mechanisms. This guidance is based on the significance of deviation from the average value. With the usage of these parameters, we obtain a set of partitions that should be recovered after their failure occurs.

The function *avg_param(S)* calculates the arithmetic average for the complete system set (i.e. the already selected one version of component per each partition of composed reliable system; these versions are denoted as *S*), as shown in Equation 3. The function *card(S)* denotes the cardinality (i.e. the number of elements) of the set *S*. Equations 4 and 5 describe the calculation of significance (i.e. *sig*) parameter from the deviation (i.e. *dev*) and the selection of the proper comparison operator. This is because in certain cases, the target parameter is maximized (e.g. in the case of the *Time To Failure* (TTF) parameter). In other cases, the parameter is minimized (e.g. in the case of the percentage of critical bits of bitstream on an FPGA). The final set of partitions that should be recovered after their failure is denoted as *R*. Equation 6 contains such calculation of the set.

$$\mathbf{avg\_param}(S) = \frac{1}{\mathbf{card}(S)} \times \sum_{\forall c \in S} \mathbf{param}(c) \tag{3}$$

$$sig = \begin{cases} 1 - dev, & \textit{if } \mathbf{param} \textit{ is maximized}, \\ 1 + dev, & \textit{if } \mathbf{param} \textit{ is minimized}. \end{cases} \tag{4}$$

$$op = \begin{cases} <, & \textit{if } \mathbf{param} \textit{ is maximized}, \\ >, & \textit{if } \mathbf{param} \textit{ is minimized}. \end{cases} \tag{5}$$

$$R = \{\, c \mid c \in S,\ \mathbf{param}(c)\ op\ sig \times \mathbf{avg\_param}(S) \,\} \tag{6}$$

We empirically selected the deviation to be 0.4 of the average. This means that the significance parameter will be 0.6 or 1.4 for maximized or minimized parameters respectively. Thus, every partition with its parameter being worse than the average (i.e. for more than the deviation of 0.4) will be considered suitable for recovery.

The component recovery is provided by adding a *Reconfiguration Controller* (RC) component that repairs a faulty module. The RC can be either internal, on the same FPGA with the system, or external, on a different FPGA chip. This principle has been described by the authors of paper [26]. For a system described in VHDL language, the *Generic Partial Dynamic Reconfiguration Controller* (GPDRC) [27] implemented directly into the FPGA logic can be used, which is able to reconfigure any predefined module. The ability to detect a fault is important. The hardened component must be able to identify its faulty modules with an erroneous outputs. The information on faulty modules is provided to the GPDRC, which then reads the relevant data needed to reconfigure, the so-called golden bitstream. These are stored in flash memory, which is more resilient to SEUs. Subsequently, the reconfiguration itself is performed by sequentially uploading the golden bitstream via the *Internal Configuration Access Port* (ICAP) interface.

D. Testing of Fault Tolerance Methods

For the automation of FT system design, it is also important to test the resulting degree of FT. The testing is, however, held in huge quantities, depending on the size of the system and the number of its partitions. The duration of test basically denotes the time needed to find the solution. As the previous two stages

incorporate mainly text manipulations, their time complexity is nearly negligible.

In our research, we achieved automated generation of greatly accelerated testbeds by implementing the *Fault-Tolerance Estimation* (FT-EST) framework [28] in VHDL using the so-called VHDL generics. Using this approach, it is possible to create a fully functional testbed generator, which is configurable and easily adapts to demands of the tested design. The FT-EST is configurable in one configuration file, which holds settings. These include, for example, the bit width of input and output pins of the tested system. The method of the test or the test termination conditions are also configurable in this file using the prepared enumerated-type constants. In one VHDL file, the tested unit (i.e. a component or a system) is instantiated and the needed port connections are routed to it. After the FT-EST is configured, it is possible to synthesize it with any common VHDL synthesis SW. A part of the solution is also a script, that prepares the placement of tested units, as these must be placed properly to avoid their overlap between themselves or even with the FT-EST controller components.

With FT-EST testbed, it is possible (among others) to measure the TTF of each run in milliseconds, excluding the periods in which the clock signal of tested units was paused. The pausing from the SW during the fault injection is necessary to simulate higher fault injection intensities, for which the speed of fault injection during the circuit run time would not be sufficient. It also eliminates possible collisions between the bitstream recovery (e.g. the GPDRC) and the fault injection mechanism. The re-configurable devices are ideal for testing, as a system failure can be easily triggered through the bitstream manipulation and the design can be easily repaired by reverting the bitstream. Our solution performs tests at-speed and directly on the real FPGA, utilizing the target technology of the tested system.

The tests are executed autonomously on the HW. The PC downloads the bitstream to the FPGA and instructs the testbed to start the test. It is also possible to pause the clock signal of the tested units, which is useful to modify the design for the simulation of failure. If the clock signal is paused, such failure then appears instantly for the tested units. SEUs are injected using our *Fault Injector* (FI) [29]. In order to inject faults only into utilized parts of tested unit, specifically utilized *Look-up Tables* (LUTs). Special SW for detection of utilized bits of a specified block is used. The SW was developed previously in our research group and is based on the RapidSmith SW [15]. It is also possible to instantiate multiple equivalent tested units at the same time, significantly accelerating the whole approach.

The whole test is controlled by the *Experiment Control Unit* (XCU), which is formed by a *Finite State Machine* (FSM). Tested units alongside with one golden (i.e. reference) unit, are instantiated in the *Unit Instantiation Area* (UIA). The test data are generated by the *Input Generation Unit* (IGU). Primarily, the FT-EST does not tolerate timing deviations caused by a failure, however, this strict behavior can be adjusted by modifying the *Output Compare Unit* (OCU), which compares outputs to the reference ones. The *Failure Capture Unit* (FCU) stores the parameters of the failure, such as the number of deviations or the real time of the first observation of an output data deviation. All the stored data, state registers and configuration registers are accessible through the universal *Communication Interface* (CI). This is connected to the *Communication Module* (CM), holding the actual vendor-specific communication implementations. In our project, we use the Xilinx ChipScope Pro *Integrated Controller* (ICON) core [30] and the *Virtual Input/Output* (VIO) core [31] for communication through the USB JTAG interface. The only technology-dependent specifics are isolated in the communication and fault injection components. The rest of the FT-EST testbed compatibility is not bound to a specific VHDL synthesis tool nor an FPGA technology, as the testbed is designed in plain VHDL. The structure of generated testbed is displayed in Figure 3.

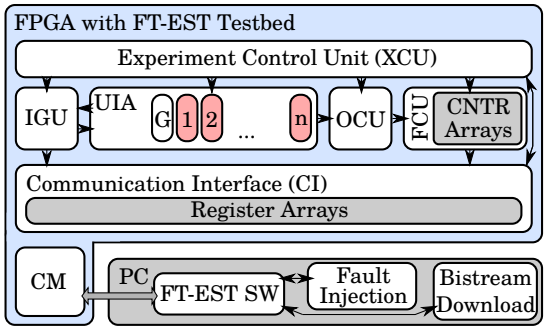


Figure 3: Structure of Automatically Generated Testbed Architecture, Red Parts are Subject to Fault Injection.

IV. Experiments and Results

As part of our experimental activities, we created an artificial benchmark system and used our FT system design automation toolkit to prepare its FT enhanced version to a limited FPGA area space.

A. Benchmark System

Our benchmark system is composed of four components, which will also be considered partitions of the system. These include: 1) addition of two 16-bit unsigned numbers; 2) 16-bit constant addition to a 16-bit unsigned number; 3) numones, which calculates the number of high bits in a 16-bit input data; and 4) *Cyclic Redundancy Check* (CRC) calculated on 8 bits (i.e. CRC-8) based on 32 bit wide input. The schematic diagram of the benchmark system can be seen in Figure 4.

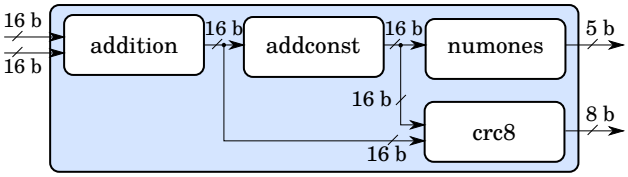


Figure 4: Diagram of the Benchmark System.

B. On-HW Testing Setup

In the following text, various partitions and complete systems are tested. For this purpose, we utilized our FT-EST framework and generated a completely autonomous testbed for each of the tests. The FT-EST was utilizing *Linear Feedback Shift Register* (LFSR) as a generator of stimuli during the test. Different polynomials were used in the implementation, to always suit the bit width of the tested unit. The FT-EST was configured to hold the test with a constant fault

injection intensity parameter, which we defined in [32]. The test continues until the tested unit delivers incorrect results on its output ports. The fault injection intensity determines the number of randomly-placed fault injections per second, related to the size of the design. This implies that for a larger design, faults are injected more frequently, although the injection intensity parameter remains constant. This is because the larger component occupies a larger chip area, thus the area exposed to radiation is also higher. And this must be reflected to obtain fair results. In our experiments, we empirically chose the injection intensity of $2\mathrm{e}{-5}\,\mathrm{inj/s/bit}$.

In a real scenario, multiple tested units fit into one FPGA. As parallel unit execution is supported by the FT-EST, the test controller actually waits until each of the tested units delivers incorrect results, while keeping the real time of the first failure observation (in milliseconds) for each of the tested units. These TTFs are then downloaded to the control PC. On the PC, the results are stored in files on a hard drive and further analyzed to obtain statistical data, mainly the t50 parameter.

C. Partitions Variants

At the beginning of our design flow, various versions of partitions were created. Variants for a partition are made from the original version of the partition with the usage of helpers. As part of our previous research, we created two templates for our VHDL helpers: the TMR and *5-Modular Redundancy* (5-MR). The number of generated variants for each partition, is dependent on the helpers that are used. For example, in our experiments, one TMR and one 5-MR variant is generated to the original implementation of a partition. Each partition was then tested on the FT-EST testbed and its t50 parameter was measured. Resulting parameters of the variants are shown in Table I. These results were already obtained in our previous research [6].

TABLE I: Partitions Implementation Versions Including Their Size and t50 Parameter under Fault Injection Intensity of $2\mathrm{e}{-5}\,\mathrm{inj/s/bit}$ [6]

Partition Name	FT Technique	Bitstream Area [b]	t50 [ms]	t50 Compared to Simplex [ms]	[%]
addition	simplex	4 288	197 635	+ 0	+ 0.00
	TMR	7 552	208 793	+ 11 158	+ 5.64
	5-MR	9 856	225 042	+ 27 407	+ 13.87
addconst	simplex	3 264	337 843	+ 0	+ 0.00
	TMR	6 656	271 246	- 66 597	- 19.71
	5-MR	9 088	345 745	+ 7 902	+ 2.34
crc8	simplex	4 800	39 484	+ 0	+ 0.00
	TMR	9 792	47 222	+ 7 738	+ 19.6
	5-MR	14 272	60 227	+ 20 743	+ 52.54
numones	simplex	3 072	94 549	+ 0	+ 0.00
	TMR	6 848	102 603	+ 8 054	+ 8.52
	5-MR	10 304	119 195	+ 24 646	+ 26.07

The results for these variants can be also seen in a box plot chart in Figure 5. As can be observed, each FT technique has various impacts, depending on the type and structure of the partition. Increased redundancy leads to better TTF results, except of the addconst partition. The TMR version of this partition is less tolerant to faults, compared to its simplex (i.e. original) version. We believe that this was caused by the different structure of this partition, as the addconst is very dependent on the internally stored constant value, which is not the case for the otherwise very similar addition partition.

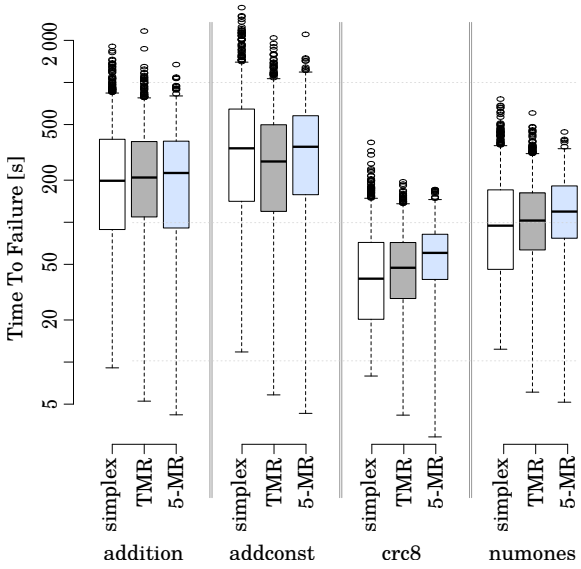
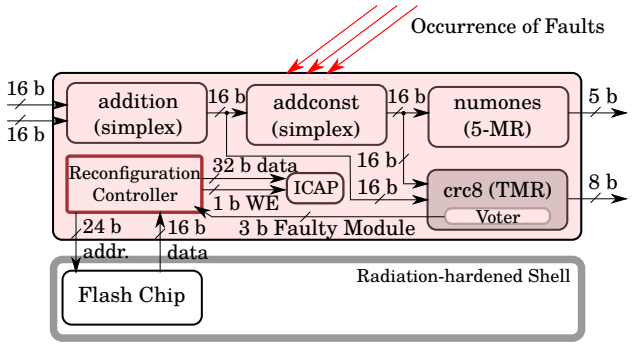


Figure 5: Time to Failure for Each Partition Version on a Box Plot Chart [6].

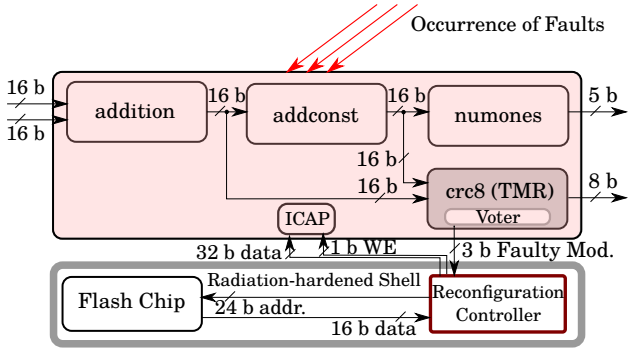
D. Generated Systems and Their Parameters

At first, we evaluated the original system, i.e. a system that was composed of the original (unhardened) partition versions. Furthermore, we prepared two additional (i.e. reference) systems, where each of them is composed of TMR and 5-MR partition versions exclusively. These are measured in the first part of Table II. Then, with the usage of previously described guidance method and the modified MCKP solver, we created another system of components of mixed type. The guider was set to maximize the TTF parameter, while limiting the FPGA chip area to $30\,000\,\mathrm{bits}$. The reference and automatically generated systems were presented as a part of our previous paper [6]. In this follow-up research, however, the PDR technique is further incorporated, which improves fault resiliency. With the previously described deviation parameter being set to 0.4, the guider algorithm also provides the recommendation on which partition recovery would increase reliability the most. This is the case for the partition marked by a dot in the second part of Table II. We, thus, degraded the FT method to TMR and used the PDR to further harden the system. Two systems are created: 1) the reconfiguration controller is on the chip (i.e. it is subject to the fault injection), and 2) the reconfiguration controller is outside of the injection area (i.e. it is outside the FPGA, for example on a standalone radiation-hardened chip). The structure of both the recovered systems are shown in Figure 6. Results for these systems can be seen in the third part of Table II.

The system with reconfiguration controller on the FPGA is significantly larger, because of the controller. The PDR controller increases the size of the system approximately by 85%. It, however, even if the large controller is under equivalent fault injection intensity as the rest of the system, still achieved nearly equivalent results compared to the variant without reconfiguration. In this case, the reconfiguration controller on the same chip does not improve the parameter. However, this is still a considerable result, as the reconfiguration controller in our tests is not hardened in any manner. We expect that for larger systems, the efficiency of the on-chip PDR controller will be significantly better. The system with the



(a)



(b)

Figure 6: Recovered Systems with PDR controller (a) Inside the System on the FPGA; and (b) Outside the FPGA.

TABLE II: Automatically Generated and Manually Created System Compositions (as a Reference) [6] and the Recovered System Including the t50 Parameter under Fault Injection Intensity of $2e{-}5\,\mathrm{inj/s/bit}$, *(partition marked with the dot is recommended for recovery by the guider algorithm)*

	System Name	FT Techniques addition	addconst	crc8	numones	Bitstream Area [b]	t50 [ms]
Reference	simplex	simplex	simplex	simplex	simplex	9 152	23 559
	TMR	TMR	TMR	TMR	TMR	24 704	42 173
	5-MR	5-MR	5-MR	5-MR	5-MR	37 376	55 900
auto_30000		simplex	simplex	5-MR •	5-MR	25 856	49 675
auto_30000	+Rec. On Chip	simplex	simplex	TMR+Rec. On Chip	5-MR	48 000	48 584
	+Rec. Outside	simplex	simplex	TMR+Rec. Outside	5-MR	29 376	84 631

external reconfiguration controller is, of course, significantly more reliable. Its t50 parameter is by 70% better, compared to the automatically generated system without the PDR. In comparison to the 5-MR reference system, the t50 parameter is still more than 50% better. This, however, assumes that the external reconfiguration controller is resilient against failure. If such external component is available, the guider algorithm suggests to utilize it. As can be observed, the size of this system increased slightly, which might look non-intuitive, considering the replacement of the 5-MR crc8 with the TMR recovered version. This is because in the case of PDR, the redundant modules must be strictly separated (e.g. foreign module signals must be routed outside of the module), further complicating the synthesis optimization processes and thus the logic. Also, the voting component inside the crc8 must be slightly more complicated, as it signalizes the failing crc8 module (i.e. compared to the previous 5-MR without PDR, which had to select the representative result only).

Figure 7 displays TTF for each of the systems. As can be observed, the helpers generally work, as the reference systems increase their t50 with growing redundancy. The auto-generated system (i.e. the one automatically composed using the guider) reaches better t50 parameter than the TMR. From the box plot it is obvious, that the dissipation of TTF is nearly equivalent to the reference TMR system [6]. Although the auto-generated variant without PDR and with PDR on chip achieves nearly equivalent t50 times, the box plot clearly illustrates, that the recovered variant has much higher dissipation towards (only) higher values of TTF, which might be a positive feature. The second recovered variant with PDR controller outside the FPGA achieves the best results. It is important to note, that only one component, the crc8, was recovered with the PDR, yet the result is significantly better. This definitely confirms the benefit of PDR for recovery of FPGA systems.

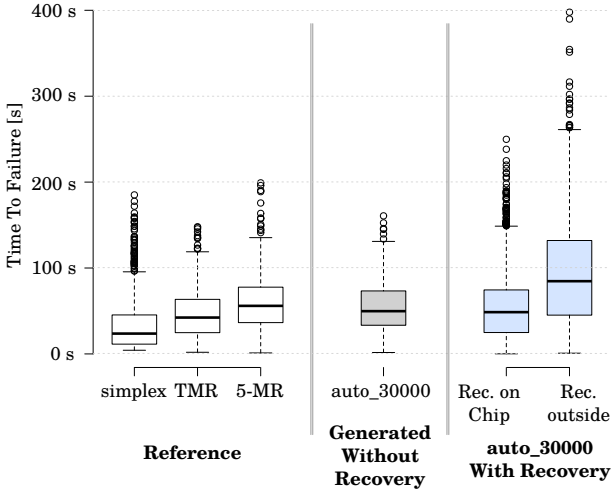


Figure 7: Time to Failure for Each of the Systems (i.e. Reference, Auto-generated [6] and Auto-generated with PDR Controller) on a Box Plot Chart.

V. Conclusions and Future Research

In this paper, a description of our FT system design automation toolkit was presented. The toolkit is composed of various parts and components and each of them was, at least briefly, described in this paper. The experimental part of the paper presents incorporation of PDR controller into the system design. Two systems utilizing recovery with the usage of the FPGA PDR technique show promising results in terms of reliability. The recovered system with the on-chip PDR controller shows similar results, compared to the system without recovery. This is because the PDR controller is relatively large, compared to the system and is also subject to fault injection with equivalent intensity based on the chip area. The second recovered system, in which the PDR controller is apart the FPGA (e.g. in a radiation-hardened chip) shows significantly better results. Its t50 parameter is by 70% better, compared to the system without recovery.

The future ideas to further extend our FT design automation toolkit can be directed towards *Software-Implemented Fault Tolerance* (SIFT) methods. It would be interesting to use such approach to harden SW programs and test them on a real HW during the presence of faults. Such approach, combined with HW resiliency against faults, could significantly improve the overall system reliability.

ACKNOWLEDGEMENTS

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

REFERENCES

[1] J.-C. Geffroy and G. Motet, *Design of dependable computing systems*. Springer Science & Business Media, 2013.

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

[3] NASA, "Mars 2020 Perseverance Rover," 2020, accessed: 2021-04-11. [Online]. Available: https://mars.nasa.gov/mars2020/

[4] NASA, "Mars Helicopter," 2020, accessed: 2021-04-11. [Online]. Available: https://mars.nasa.gov/technology/helicopter/

[5] Farhad Fallahlalehzari, "How does the Mars Perseverance rover benefit from FPGAs as the main processing units?" accessed: 2021-04-11. [Online]. Available: https://www.aldec.com/en/company/blog/188–how-does-the-mars-perseverance-rover-benefit-from-fpgas-as-the-main-processing-units

[6] J. Lojda, R. Panek, and Z. Kotasek, "Automatic Design of Fault-Tolerant Systems for VHDL and SRAM-based FPGAs," in *Accepted for Presentation on: 2021 24th Euromicro Conference on Digital System Design (DSD), Palermo, Sicily*, Sep 2021.

[7] Xilinx Inc., "TMRTool: The Industry's First Development Tool to Automatically Generate Triple Module Redundancy (TMR) for Space-grade Re-programmable FPGAs," https://www.xilinx.com/products/design-tools/tmrtool.html, accessed: 2021-04-13.

[8] S. Kulis, "Single Event Effects Mitigation with TMRG Tool," *Journal of Instrumentation*, vol. 12, no. 01, p. C01082, 2017. [Online]. Available: http://stacks.iop.org/1748-0221/12/i=01/a=C01082

[9] G. Lee, D. Agiakatsikas, T. Wu, E. Cetin, and O. Diessel, "TLegUp: A TMR Code Generation Tool for SRAM-Based FPGA Applications Using HLS," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 129–132.

[10] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. Brown, and J. Anderson, "LegUp: An Open-Source High-Level Synthesis Tool for FPGA-Based Processor/Accelerator Systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, 09 2013.

[11] J. Onishi, S. Kimura, R. J. James, and Y. Nakagawa, "Solving the Redundancy Allocation Problem with a Mix of Components Using the Improved Surrogate Constraint Method," *IEEE Transactions on Reliability*, vol. 56, no. 1, pp. 94–101, 2007.

[12] W.-C. Yeh and T.-J. Hsieh, "Solving Reliability Redundancy Allocation Problems Using an Artificial Bee Colony Algorithm," *Computers & Operations Research*, vol. 38, no. 11, pp. 1465–1473, 2011.

[13] K. Khalili-Damghani, A.-R. Abtahi, and M. Tavana, "A New Multi-objective Particle Swarm Optimization Method for Solving Reliability Redundancy Allocation Problems," *Reliability Engineering & System Safety*, vol. 111, pp. 58–75, 2013.

[14] Y.-C. Liang and Y.-C. Chen, "Redundancy Allocation of Series-parallel Systems Using a Variable Neighborhood Search Algorithm," *Reliability Engineering & System Safety*, vol. 92, no. 3, pp. 323–331, 2007.

[15] C. Lavin, M. Padilla, P. Lundrigan, B. Nelson, and B. Hutchings, "Rapid Prototyping Tools for FPGA Designs: RapidSmith," in *Field-Programmable Technology (FPT), 2010 International Conference on*, Dec 2010, pp. 353–356.

[16] T. Schweizer, D. Peterson, J. M. Kühn, T. Kuhn, and W. Rosenstiel, "A Fast and Accurate FPGA-based Fault Injection System," in *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*. IEEE, 2013, pp. 236–236.

[17] J. M. Kuuhn, T. Schweizer, D. Peterson, T. Kuhn, and W. Rosenstiel, "Testing Reliability Techniques for SoCs with Fault Tolerant CGRA by Using Live FPGA Fault Injection," in *Field-Programmable Technology (FPT), 2013 International Conference on*. IEEE, 2013, pp. 462–465.

[18] M. Liu, Z. Zeng, F. Su, and J. Cai, "Research on Fault Injection Technology for Embedded Software based on JTAG Interface," in *Reliability, Maintainability and Safety (ICRMS), 2016 11th International Conference on*. IEEE, 2016, pp. 1–6.

[19] S. Rudrakshi, V. Midasala, and S. Bhavanam, "Implementation of FPGA based Fault Injection Tool (FITO) for Testing Fault Tolerant Designs," *IACSIT International Journal of Engineering and Technology*, vol. 4, no. 5, pp. 522–526, 2012.

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

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

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

[23] C. López-Ongil, M. Garcia-Valderas, M. Portela-García, and L. Entrena, "Autonomous Fault Emulation: A New FPGA-based Acceleration System for Hardness Evaluation," *Nuclear Science, IEEE Transactions on*, vol. 54, no. 1, pp. 252–261, 2007.

[24] J. Podivinsky, J. Lojda, O. Cekan, and Z. Kotasek, "Evaluation Platform for Testing Fault Tolerance Properties: Soft-core Processor-based Experimental Robot Controller," in *2018 21st Euromicro Conference on Digital System Design (DSD)*, 2018, pp. 229–236.

[25] J. Lojda, J. Podivinsky, Z. Kotasek, and M. Krcma, "Data Types and Operations Modifications: A Practical Approach to Fault Tolerance in HLS," in *2017 IEEE East-West Design Test Symposium (EWDTS)*, Sept 2017, pp. 1–6.

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

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

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

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

[30] Xilinx Inc., "LogiCORE IP ChipScope Pro Integrated Controller (ICON) Documentation," https://www.xilinx.com/support/documentation/ip_documentation/chipscope_icon/v1_05_a/chipscope_icon.pdf, Jun. 2011, accessed: 2018-02-15.

[31] Xilinx Inc., "ChipScope Pro VIO Documentation," https://www.xilinx.com/support/documentation/ip_documentation/chipscope_vio.pdf, Sep. 2009, accessed: 2018-02-15.

[32] J. Lojda, J. Podivinsky, Z. Kotasek, and M. Krcma, "Majority Type and Redundancy Level Influences on Redundant Data Types Approach for HLS," in *2018 16th Biennial Baltic Electronics Conference (BEC)*, Oct 2018, pp. 1–4.